

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

Prior Art Reference: U.S. Patent No. 7,861,060 (“Nickolls”)

Nickolls was filed on December 15, 2005, and therefore qualifies as prior art to U.S. Patent No. RE48,438 (“the ’438 patent”) under at least pre-AIA 35 U.S.C. § 102(e). Nickolls anticipates and/or renders obvious claims 1–10, 12–14, 16–18, 20–32, and 40–54 of the ’438 patent (the “Asserted Claims”).

The chart below provides representative examples of where each element of each claim is found within Nickolls. Citations are meant to be exemplary, not exhaustive, and NVIDIA reserves the right to identify and discuss additional portions of the reference in support of its contentions and/or to rebut arguments made by Plaintiff. Where NVIDIA states that Nickolls “discloses” a limitation, that disclosure may be express, implicit, and/or inherent. Citations to figures, drawings, tables, and the like include reference to any accompanying or related text. All internal cross-references are meant to incorporate the cross-referenced material as if fully set forth therein.

It is NVIDIA’s position that Plaintiff’s Infringement Contentions have not established that any accused product or service infringes any valid claim. Thus, NVIDIA’s statements below should not be treated as an admission, implication, or suggestion that NVIDIA agrees with Plaintiff regarding either the scope, construction, or interpretation of any of the Asserted Claims or the infringement theories advanced by Plaintiff in its Infringement Contentions, including whether any Asserted Claim satisfies 35 U.S.C. §§ 101 or 112. The exemplary disclosures in these invalidity contentions are informed by the apparent interpretation of the asserted claims by Plaintiff. These statements are not intended to suggest that NVIDIA agrees with Plaintiff’s application of any claim term, suggest a proposed construction at this stage of the case, or suggest that construction is needed.

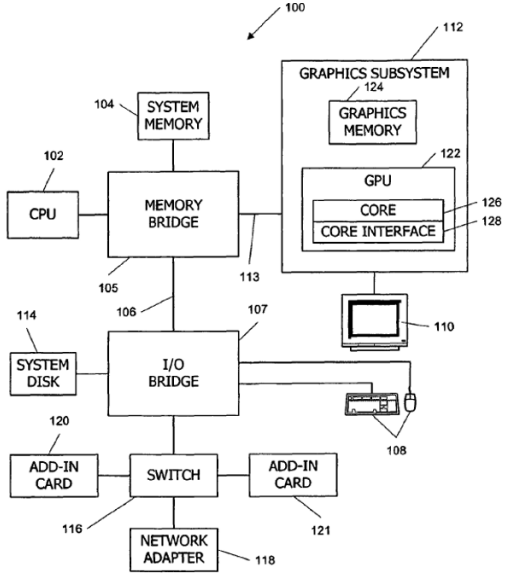
To the extent Plaintiff argues that any element below is not disclosed by Nickolls, a person of ordinary skill in the art would have found it obvious to combine the teachings of Nickolls with the background knowledge of a person of ordinary skill in the art and/or the additional references, and exemplary teachings, set forth in NVIDIA’s Invalidity Contentions, including all exhibits and appendices thereto. Additional exemplary combinations and reasons for obviousness are described in the cover pleading, and are incorporated by reference herein.

Plaintiff has yet to identify any limitation of the Asserted Claims that it contends is not anticipated and/or rendered obvious by Nickolls. NVIDIA therefore expressly reserves the right to respond to any such contention, including by identifying additional obviousness combinations, if Plaintiff makes any such contention. NVIDIA reserves the right to rely on additional citations or sources of evidence that also may be applicable, or that may become applicable in light of claim construction, changes in Plaintiff’s infringement contentions, and/or information obtained during discovery as the case progresses.

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
<p>1[pre] “A computer system, comprising:”</p>	<p>To the extent the preamble is limiting, Nickolls discloses “a computer system.” <i>See e.g.:</i></p> <p><i>As a non-limiting example, Nickolls discloses a computer system (e.g., computer system 100) and a computer implemented method for processing data:</i></p> <p>The present invention relates in general to parallel data processing, and in particular to parallel data processing methods using arrays of threads that are capable of sharing data, including intermediate results, with other threads in a thread-specific manner. 1:21–26.</p> <p>According to one aspect of the present invention, a computer implemented method for processing data includes defining, in a processor, a thread array having multiple threads, each thread configured to execute a same program on an input data set. In some embodiments, the program represents a unit of a data parallel decomposition of a data processing algorithm. The processor assigns each of the threads in the thread array a unique thread identifier (which can be, e.g., a one-dimensional or multidimensional identifier). The processor executes the threads in the thread array concurrently with each other; during execution, each thread uses the unique thread identifier to determine at least one processing behavior. Any processor, including but not limited to a graphics processor, may be used to implement the method. Parallel and multithreaded processors can execute multiple threads of a CTA in parallel for enhanced performance. 2:65–3:13.</p> <p>1. A computer-implemented method for processing data, the method comprising:</p> <ul style="list-style-type: none"> defining, in a processor, a thread array having a plurality of threads, each thread configured to execute a same program on a subset of an input data set; assigning, by the processor, each of the threads in the thread array a unique thread identifier value to be used to determine the subset of the input data set to be processed by each respective thread, wherein each unique thread identifier value is unique within the thread array; and executing, by the processor, the plurality of threads in the thread array concurrently with each other, wherein during execution, each thread uses the unique thread identifier value

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>assigned thereto as an input to compute at least one program-specified function of the unique thread identifier value, wherein the plurality of threads includes a first thread configured to execute the same program on a first subset of the input data set and a second thread configured to execute the same program on a second subset of the input data set, and wherein data in the first subset overlaps with the second subset.</p> <p>Claim 1.</p> <p>FIG. 1 is a block diagram of a computer system according to an embodiment of the present invention; 4:49–50.</p>  <p style="text-align: center;"><i>FIG. 1</i></p> <p>FIG. 1. Computer System Overview</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>FIG. 1 is a block diagram of a computer system 100 according to an embodiment of the present invention. Computer system 100 includes a central processing unit (CPU) 102 and a system memory 104 communicating via a bus path that includes a memory bridge 105. Memory bridge 105 is connected via a bus path 106 to an I/O (input/output) bridge 107. I/O bridge 107 receives user input from one or more user input devices 108 (e.g., keyboard, mouse) and forwards the input to CPU 102 via bus 106 and memory bridge 105. Visual output is provided on a pixel based display device 110 (e.g., a conventional CRT or LCD based monitor) operating under control of a graphics subsystem 112 coupled to memory bridge 105 via a bus 113. A system disk 114 is also connected to I/O bridge 107. A switch 116 provides connections between I/O bridge 107 and other components such as a network adapter 118 and various add-in cards 120, 121. Other components (not explicitly shown), including USB or other port connections, CD drives, DVD drives, and the like, may also be connected to I/O bridge 107. Bus connections among the various components may be implemented using bus protocols such as PCI (Peripheral Component Interconnect), PCI Express (PCI-E), AGP (Accelerated Graphics Port), HyperTransport, or any other bus protocol(s), and connections between different devices may use different protocols as is known in the art.</p> <p>6:8–34.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>1[a] “a central processing unit to receive input data;”</p>	<p>Nickolls discloses “a central processing unit to receive input data.” <i>See e.g.:</i></p> <p><i>As a non-limiting example, Nickolls discloses a central processing unit (e.g., CPU 102) that receives input data:</i></p> <p>FIG. 1 is a block diagram of a computer system according to an embodiment of the present invention; 4:49–50.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438

Nickolls

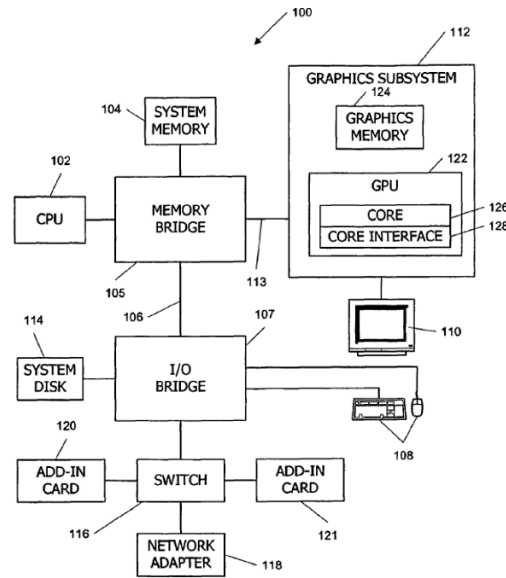


FIG. 1

FIG. 1.

Computer System Overview

FIG. 1 is a block diagram of a computer system 100 according to an embodiment of the present invention. Computer system 100 includes a central processing unit (CPU) 102 and a system memory 104 communicating via a bus path that includes a memory bridge 105. Memory bridge 105 is connected via a bus path 106 to an I/O (input/output) bridge 107. I/O bridge 107 receives user input from one or more user input devices 108 (e.g., keyboard, mouse) and forwards the input to CPU 102 via bus 106 and memory bridge 105. Visual output is provided on a pixel based display device 110 (e.g., a conventional CRT or LCD based monitor) operating under control of a graphics subsystem 112 coupled to memory bridge 105 via a bus 113. A system disk 114 is also connected to I/O bridge 107. A switch 116 provides connections between I/O bridge 107 and other components such as a network adapter 118 and various add-in cards 120, 121. Other components (not explicitly shown), including USB or other

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>port connections, CD drives, DVD drives, and the like, may also be connected to I/O bridge 107. Bus connections among the various components may be implemented using bus protocols such as PCI (Peripheral Component Interconnect), PCI Express (PCI-E), AGP (Accelerated Graphics Port), HyperTransport, or any other bus protocol(s), and connections between different devices may use different protocols as is known in the art. 6:8–34.</p> <p>Graphics processing subsystem 112 includes a graphics processing unit (GPU) 122 and a graphics memory 124, which may be implemented, e.g., using one or more integrated circuit devices such as programmable processors, application specific integrated circuits (ASICs), and memory devices. GPU 122 may be configured to perform various tasks related to generating pixel data from graphics data supplied by CPU 102 and/or system memory 104 via memory bridge 105 and bus 113, interacting with graphics memory 124 to store and update pixel data, and the like. For example, GPU 122 may generate pixel data from 2-D or 3-D scene data provided by various programs executing on CPU 102. 6:35–46.</p> <p>CPU 102 operates as the master processor of system 100, controlling and coordinating operations of other system components. In particular, CPU 102 issues commands that control the operation of GPU 122. In some embodiments, CPU 102 writes a stream of commands for GPU 122 to a command buffer, which may be in system memory 104, graphics memory 124, or another storage location accessible to both CPU 102 and GPU 122. GPU 122 reads the command stream from the command buffer and executes commands asynchronously with operation of CPU 102. 6:62–7:4.</p> <p>It will be appreciated that the system shown herein is illustrative and that variations and modifications are possible. The bus topology, including the number and arrangement of bridges, may be modified as desired. For instance, in some embodiments, system memory 104 is connected to CPU 102 directly rather than through a bridge, and other devices communicate with system memory 104 via memory bridge 105 and CPU 102. In other alternative topologies, graphics subsystem 112 is connected to I/O bridge 107 rather than to memory bridge 105. In still other embodiments, I/O bridge 107 and memory bridge 105 might be integrated into a single chip. The particular components shown herein are optional; for instance, any number of add-in cards or peripheral devices might be supported. In some</p>

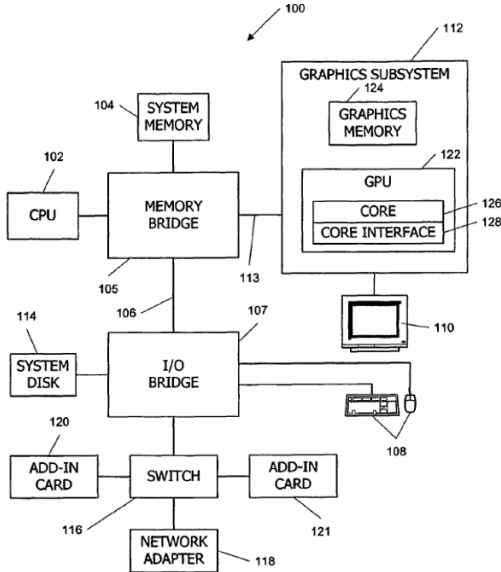
Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>embodiments, switch 116 is eliminated, and network adapter 118 and add-in cards 120, 121 connect directly to I/O bridge 107. 7:5–20.</p> <p>It will be appreciated that the core interface described herein is illustrative and that variations and modifications are possible. Components such as incrementers, step size calculators, and thread ID generators shown herein may be modified as desired. In some embodiments, computations described as being performed by the core interface can be performed elsewhere. For instance, a component of GPU 122 (FIG. 1) that supplies signals to the core interface might compute the initial thread IDs and/or the step size parameters and supply these values to the core interface as additional state parameters. In another alternative embodiment, a driver program executing on CPU 102 receives a CTA size parameter (e.g., dimensions D2, D1, and D0) from an application program and uses CPU resources to compute the initial P thread IDs and/or the step size, then supplies these parameters to core interface 128 of GPU 122 as state information, and incrementer 506 and step calculator 508 may both be omitted. 18:55–19:4.</p> <p>On completion of the CTA program, the final results (output data) produced by the threads are advantageously placed in memory for use by a subsequent CTA program or made accessible to CPU 102 (FIG. 1). For example, the final instructions in a CTA program might include an instruction to write some or all of the data generated by the thread to graphics memory 124. After execution of the CTA is finished, GPU 122 may transfer the data (e.g., using a conventional DMA operation) to system memory 104, making it available to application programs executing on CPU 102. Other data transfer mechanisms may also be used. Core 126 advantageously signals core interface 128 upon completion of a CTA, so that core interface 128 can initiate execution of a next CTA, reusing the resources that became free when the first CTA was completed. 20:61–21:8.</p> <p>In some embodiments, CTAs executed by GPU 122 (FIG. 1) are used to perform computations under the direction of an application program executing on CPU 102. An application program interface (API) for defining and executing CTAs is advantageously provided to allow application programmers to access CTA functionality as desired. 25:65–26:4.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>As is known in the art, communication between CPU 102 and GPU 122 can be managed by a driver program that executes on CPU 102. The driver program supports an application program interface (API) that defines function calls supported by GPU 122, and an application programmer can invoke the GPU functions by including suitable function calls from the API at appropriate places in the program code. 26:5–11.</p> <p>For instance, an image processing algorithm executing on CPU 102 might require application of a convolution filter as one step in a larger process. Via API function calls, the application programmer can define a CTA that applies the convolution filter (e.g., as described below) and call a function that invokes CTA processing at appropriate points in the program. It should be noted that the application programmer does not need to know details of where or how the CTA will be executed, only that the CTA will be executed and that resulting data will be written to a well-defined and accessible storage location, such as an area in system memory 104 specified by the application program. 26:36–47.</p> <p>In some instances, the application program executing on CPU 102 might need to wait for GPU 122 to finish processing one or more CTAs, e.g., if the data generated by the CTA(s) is needed for a subsequent processing step, and this can introduce some latency. (Such latency will generally be less than the latency associated with sequential processing techniques.) In some instances it may be possible to hide some or all of this latency through suitable sequencing or scheduling of program instructions, e.g., by arranging the program sequence so that the CTA is processed by GPU 122 while CPU 102 performs other functions that do not rely on the CTA data. It will be recognized that the extent to which latency can be hidden through software techniques will depend on the particular application. 26:48–62.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>1[b] “main memory, operably coupled to the central processing unit via a bus, to store the input data received by the central processing unit;”</p>	<p>Nickolls discloses “main memory, operably coupled to the central processing unit via a bus, to store the input data received by the central processing unit.” <i>See e.g.:</i></p> <p><i>As a non-limiting example, Nickolls discloses main memory (e.g., system memory 104), which communicates with a central processing unit (e.g., CPU 102) directly and/or via a bus, and stores input data received by the central processing unit:</i></p> <p>FIG. 1 is a block diagram of a computer system according to an embodiment of the present invention; 4:49–50.</p>  <p style="text-align: center;">FIG. 1</p> <p>FIG. 1.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>Computer System Overview</p> <p>FIG. 1 is a block diagram of a computer system 100 according to an embodiment of the present invention. Computer system 100 includes a central processing unit (CPU) 102 and a system memory 104 communicating via a bus path that includes a memory bridge 105. Memory bridge 105 is connected via a bus path 106 to an I/O (input/output) bridge 107. I/O bridge 107 receives user input from one or more user input devices 108 (e.g., keyboard, mouse) and forwards the input to CPU 102 via bus 106 and memory bridge 105. Visual output is provided on a pixel based display device 110 (e.g., a conventional CRT or LCD based monitor) operating under control of a graphics subsystem 112 coupled to memory bridge 105 via a bus 113. A system disk 114 is also connected to I/O bridge 107. A switch 116 provides connections between I/O bridge 107 and other components such as a network adapter 118 and various add-in cards 120, 121. Other components (not explicitly shown), including USB or other port connections, CD drives, DVD drives, and the like, may also be connected to I/O bridge 107. Bus connections among the various components may be implemented using bus protocols such as PCI (Peripheral Component Interconnect), PCI Express (PCI-E), AGP (Accelerated Graphics Port), HyperTransport, or any other bus protocol(s), and connections between different devices may use different protocols as is known in the art.</p> <p>6:8–34.</p> <p>Graphics processing subsystem 112 includes a graphics processing unit (GPU) 122 and a graphics memory 124, which may be implemented, e.g., using one or more integrated circuit devices such as programmable processors, application specific integrated circuits (ASICs), and memory devices. GPU 122 may be configured to perform various tasks related to generating pixel data from graphics data supplied by CPU 102 and/or system memory 104 via memory bridge 105 and bus 113, interacting with graphics memory 124 to store and update pixel data, and the like. For example, GPU 122 may generate pixel data from 2-D or 3-D scene data provided by various programs executing on CPU 102.</p> <p>6:35–46.</p> <p>CPU 102 operates as the master processor of system 100, controlling and coordinating operations of other system components. In particular, CPU 102 issues commands that control the operation of GPU 122. In some embodiments, CPU 102 writes a stream of commands for GPU 122 to a command buffer, which may be in system memory 104, graphics memory 124, or another storage location accessible to</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>both CPU 102 and GPU 122. GPU 122 reads the command stream from the command buffer and executes commands asynchronously with operation of CPU 102. 6:62–7:4.</p> <p>It will be appreciated that the system shown herein is illustrative and that variations and modifications are possible. The bus topology, including the number and arrangement of bridges, may be modified as desired. For instance, in some embodiments, system memory 104 is connected to CPU 102 directly rather than through a bridge, and other devices communicate with system memory 104 via memory bridge 105 and CPU 102. In other alternative topologies, graphics subsystem 112 is connected to I/O bridge 107 rather than to memory bridge 105. In still other embodiments, I/O bridge 107 and memory bridge 105 might be integrated into a single chip. The particular components shown herein are optional; for instance, any number of add-in cards or peripheral devices might be supported. In some embodiments, switch 116 is eliminated, and network adapter 118 and add-in cards 120, 121 connect directly to I/O bridge 107. 7:5–20.</p> <p>Each processing engine 302 also has access, via a crossbar switch 305, to a (shared) global register file 306 that is shared among all of the processing engines 302 in core 126. Global register file 306 may be as large as desired, and in some embodiments, any processing engine 302 can read to or write from any location in global register file 306. In addition to global register file 306, some embodiments also provide an on-chip shared memory 308, which may be implemented, e.g., as a conventional RAM. On-chip memory 308 is advantageously used to store data that is expected to be used in multiple threads, such as coefficients of attribute equations, which are usable in pixel shader programs. In some embodiments, processing engines 302 may also have access to additional off-chip shared memory (not shown), which might be located, e.g., within graphics memory 124 and/or system memory 104 of FIG. 1. 11:3–18.</p> <p>For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group. 13:12–45.</p> <p>The CTA program may also include instructions to read from and/or write to the shared global register file 306, on-chip shared memory 308, and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1. For instance, the input data set to be processed by the CTA may be stored in graphics memory 124 or system memory 104. Intermediate results may be written to global register file 306 and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1, where they can be shared with other threads. Final results (output data) may be written to graphics memory 124 or system memory 104. 20:35–45.</p> <p>On completion of the CTA program, the final results (output data) produced by the threads are advantageously placed in memory for use by a subsequent CTA program or made accessible to CPU 102 (FIG. 1). For example, the final instructions in a CTA program might include an instruction to write some or all of the data generated by the thread to graphics memory 124. After execution of the CTA is finished, GPU 122 may transfer the data (e.g., using a conventional DMA operation) to system</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>memory 104, making it available to application programs executing on CPU 102. Other data transfer mechanisms may also be used. Core 126 advantageously signals core interface 128 upon completion of a CTA, so that core interface 128 can initiate execution of a next CTA, reusing the resources that became free when the first CTA was completed. 20:61–21:8.</p> <p>For instance, an image processing algorithm executing on CPU 102 might require application of a convolution filter as one step in a larger process. Via API function calls, the application programmer can define a CTA that applies the convolution filter (e.g., as described below) and call a function that invokes CTA processing at appropriate points in the program. It should be noted that the application programmer does not need to know details of where or how the CTA will be executed, only that the CTA will be executed and that resulting data will be written to a well-defined and accessible storage location, such as an area in system memory 104 specified by the application program. 26:36–47.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>1[c] “an accelerator, operably coupled to the central processing unit and the main memory via the bus, to receive at least a portion of the input data from the main memory, the accelerator comprising:”</p>	<p>Nickolls discloses “an accelerator, operably coupled to the central processing unit and the main memory via the bus, to receive at least a portion of the input data from the main memory.” <i>See e.g.:</i></p> <p><i>As a non-limiting example, Nickolls discloses an accelerator (e.g., GPU 122 and/or processing engines), coupled to a central processing unit (e.g., CPU 102) and main memory (e.g., system memory 104) via a bus (e.g., bus 113), that receives input data (e.g., at input 402 of core interface 128) from the main memory:</i></p> <p>FIG. 3 is a block diagram of a processing core according to an embodiment of the present invention;</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438

Nickolls

4:64–65.

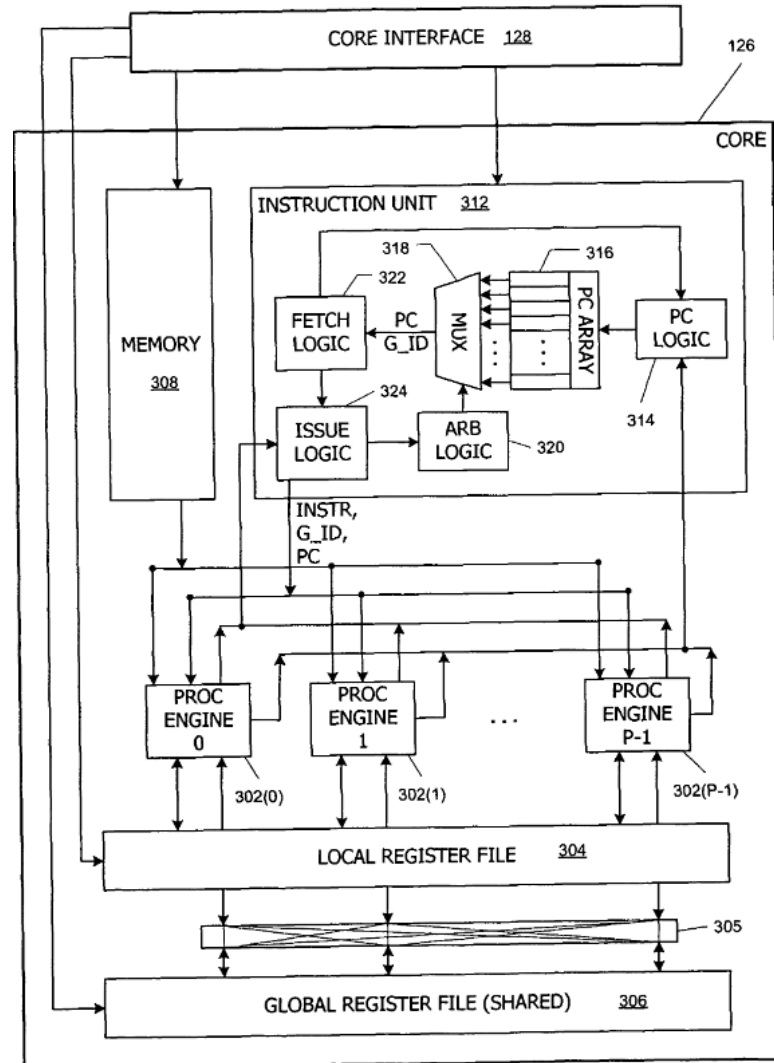
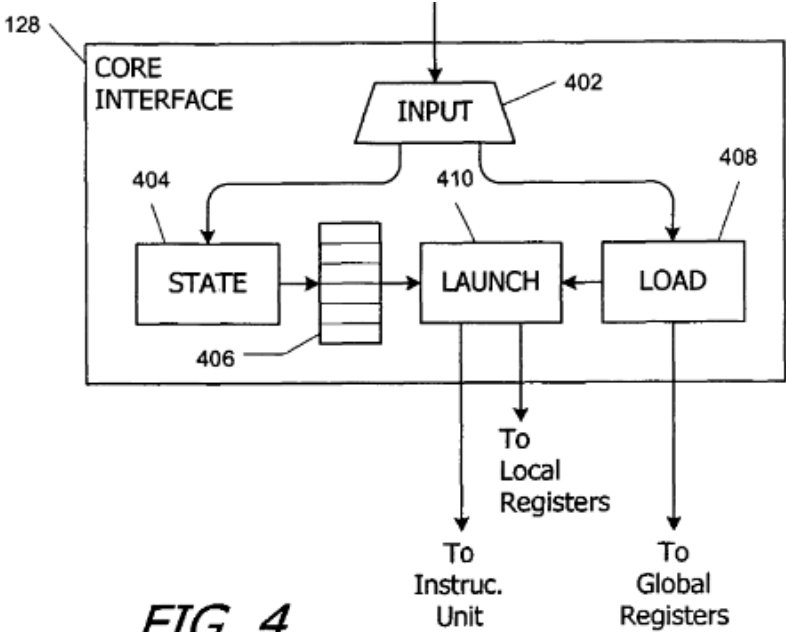
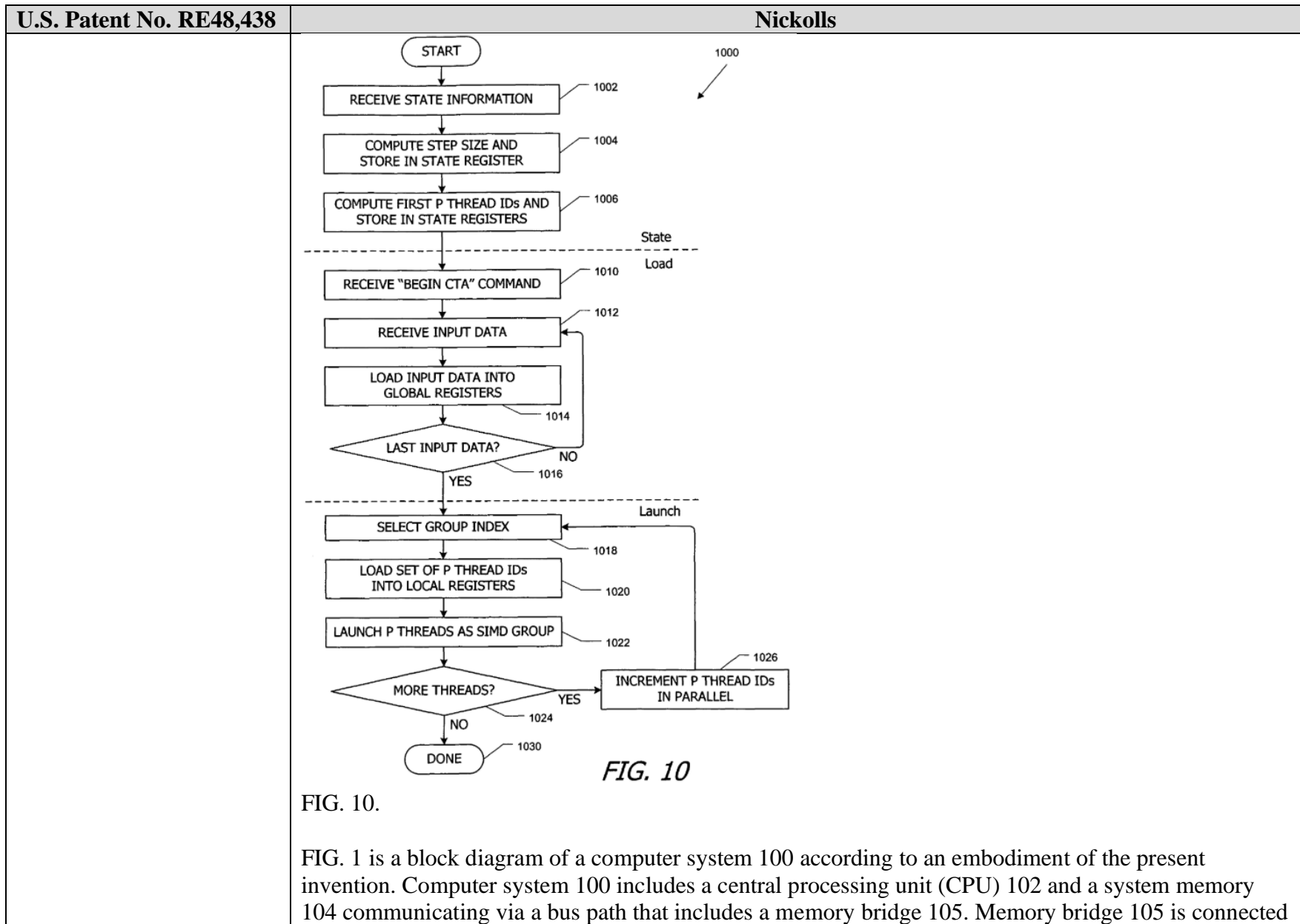


FIG. 3

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p data-bbox="583 235 676 264">FIG. 3.</p> <p data-bbox="583 310 1877 410">FIG. 4 is a block diagram of a core interface for a processing core according to an embodiment of the present invention; 4:66–67.</p>  <p data-bbox="722 1044 877 1089">FIG. 4</p> <p data-bbox="583 1118 676 1148">FIG. 4.</p> <p data-bbox="583 1193 1759 1294">FIG. 10 is a flow diagram of a control process performed by a core interface according to an embodiment of the present invention; 5:16–17.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls



Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>via a bus path 106 to an I/O (input/output) bridge 107. I/O bridge 107 receives user input from one or more user input devices 108 (e.g., keyboard, mouse) and forwards the input to CPU 102 via bus 106 and memory bridge 105. Visual output is provided on a pixel based display device 110 (e.g., a conventional CRT or LCD based monitor) operating under control of a graphics subsystem 112 coupled to memory bridge 105 via a bus 113. A system disk 114 is also connected to I/O bridge 107. A switch 116 provides connections between I/O bridge 107 and other components such as a network adapter 118 and various add-in cards 120, 121. Other components (not explicitly shown), including USB or other port connections, CD drives, DVD drives, and the like, may also be connected to I/O bridge 107. Bus connections among the various components may be implemented using bus protocols such as PCI (Peripheral Component Interconnect), PCI Express (PCI-E), AGP (Accelerated Graphics Port), HyperTransport, or any other bus protocol(s), and connections between different devices may use different protocols as is known in the art.</p> <p>6:9–34</p> <p>Graphics processing subsystem 112 includes a graphics processing unit (GPU) 122 and a graphics memory 124, which may be implemented, e.g., using one or more integrated circuit devices such as programmable processors, application specific integrated circuits (ASICs), and memory devices. GPU 122 may be configured to perform various tasks related to generating pixel data from graphics data supplied by CPU 102 and/or system memory 104 via memory bridge 105 and bus 113, interacting with graphics memory 124 to store and update pixel data, and the like. For example, GPU 122 may generate pixel data from 2-D or 3-D scene data provided by various programs executing on CPU 102.</p> <p>6:35–46.</p> <p>GPU 122 has at least one processing core 126 for generating pixel data and a core interface 128 that controls operation of core 126. Core 126 advantageously includes multiple parallel processing engines that can be used to execute various shader programs, including vertex shader programs, geometry shader programs, and/or pixel shader programs, in the course of generating images from scene data. Core 126 can also be leveraged to perform general-purpose computations as described below.</p> <p>6:47–55.</p> <p>GPU 122 may also include other components, not explicitly shown, such as a memory interface that can store pixel data received via memory bridge 105 to graphics memory 124 with or without further</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>processing, a scan out module configured to deliver pixel data from graphics memory 124 to display device 110, and so on. 6:56–61.</p> <p>CPU 102 operates as the master processor of system 100, controlling and coordinating operations of other system components. In particular, CPU 102 issues commands that control the operation of GPU 122. In some embodiments, CPU 102 writes a stream of commands for GPU 122 to a command buffer, which may be in system memory 104, graphics memory 124, or another storage location accessible to both CPU 102 and GPU 122. GPU 122 reads the command stream from the command buffer and executes commands asynchronously with operation of CPU 102. 6:62–7:4.</p> <p>The connection of GPU 122 to the rest of system 100 may also be varied. In some embodiments, graphics system 112 is implemented as an add-in card that can be inserted into an expansion slot of system 100. In other embodiments, a GPU is integrated on a single chip with a bus bridge, such as memory bridge 105 or I/O bridge 107. 7:21–26.</p> <p>CTAs can be executed using various hardware architectures, provided that the architecture can support concurrent execution of all threads of the CTA, the ability to share data among concurrent threads, and the ability to assign a unique thread ID to each thread of the CTA. In some embodiments, a suitable architecture is provided within a graphics processor such as GPU 122 of FIG. 1. 10:2–8.</p> <p>In one embodiment, GPU 122 implements a rendering pipeline that includes vertex and geometry processing, primitive setup and rasterization, attribute assembly, and pixel processing. The rendering pipeline supports various shader programs including vertex shaders, geometry shaders, and pixel shaders, examples of which are known in the art. Shader programs of arbitrary complexity are advantageously supported using a “unified shader” architecture in which one or more processing cores 126 support concurrent execution of a (preferably large) number of instances of vertex, geometry, and/or pixel shader programs in various combinations. In accordance with an embodiment of the present invention, processing core 126 is leveraged to execute CTAs for general-purpose computations.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>10:9–23.</p> <p>Input unit 402 receives all incoming signals, including state information, data and commands, and directs the signals to state module 404 or load module 406. In one embodiment, input unit 402 receives a data word (e.g., 32 bits) along with control signals indicating whether the data word corresponds to state information or data to be processed. Based on the control signals, input unit 402 determines where to direct the data word, and all or part of the control signal may be forwarded together with the data word. For instance, if the data word corresponds to state information, the control signals may be used by state module 404 to determine where to store the data word, as described below.</p> <p>14:29–40.</p> <p>It will be appreciated that the core interface described herein is illustrative and that variations and modifications are possible. Components such as incrementers, step size calculators, and thread ID generators shown herein may be modified as desired. In some embodiments, computations described as being performed by the core interface can be performed elsewhere. For instance, a component of GPU 122 (FIG. 1) that supplies signals to the core interface might compute the initial thread IDs and/or the step size parameters and supply these values to the core interface as additional state parameters. In another alternative embodiment, a driver program executing on CPU 102 receives a CTA size parameter (e.g., dimensions D2, D1, and D0) from an application program and uses CPU resources to compute the initial P thread IDs and/or the step size, then supplies these parameters to core interface 128 of GPU 122 as state information, and incrementer 506 and step calculator 508 may both be omitted.</p> <p>18:55–19:4.</p> <p>Core interface 128 remains in the state phase and continues to update the state registers as new state information is received, until such time as core interface 128 receives a “begin CTA” command (step 1010) indicating that input data is to follow. At that point, core interface 128 enters the load phase. During the load phase, core interface 128 receives input data (step 1012) and loads the input data into (shared) global register file 306 of core 126 (step 1014). At step 1016, core interface 128 determines whether the last input data has been received. If not, core interface 128 returns to step 1012 to receive more input data.</p> <p>19:40–50.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>On completion of the CTA program, the final results (output data) produced by the threads are advantageously placed in memory for use by a subsequent CTA program or made accessible to CPU 102 (FIG. 1). For example, the final instructions in a CTA program might include an instruction to write some or all of the data generated by the thread to graphics memory 124. After execution of the CTA is finished, GPU 122 may transfer the data (e.g., using a conventional DMA operation) to system memory 104, making it available to application programs executing on CPU 102. Other data transfer mechanisms may also be used. Core 126 advantageously signals core interface 128 upon completion of a CTA, so that core interface 128 can initiate execution of a next CTA, reusing the resources that became free when the first CTA was completed. 20:61–21:8.</p> <p>In some embodiments, CTAs executed by GPU 122 (FIG. 1) are used to perform computations under the direction of an application program executing on CPU 102. An application program interface (API) for defining and executing CTAs is advantageously provided to allow application programmers to access CTA functionality as desired. 25:65–26:4.</p> <p>As is known in the art, communication between CPU 102 and GPU 122 can be managed by a driver program that executes on CPU 102. The driver program supports an application program interface (API) that defines function calls supported by GPU 122, and an application programmer can invoke the GPU functions by including suitable function calls from the API at appropriate places in the program code. 26:5–11.</p> <p>In accordance with an embodiment of the present invention, the API for a driver program for GPU 122 enables an application program to invoke the CTA-processing functions of GPU 122. The API advantageously allows the application program to define a CTA program, e.g., by reference to a location in memory where the first instruction of the CTA program is stored; this aspect of the interface can be implemented analogously to existing APIs that allow application programs to define shader programs to be executed by a GPU. Thus, an application developer may write an arbitrary CTA program to be executed as part of an application. In other embodiments, a maker of GPU 122 or a third party may provide a library of CTA programs from which application developers can select programs,</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>and the developer may have the option of selecting a CTA program from the library or creating a custom CTA program. 26:12–27.</p> <p>In some instances, the application program executing on CPU 102 might need to wait for GPU 122 to finish processing one or more CTAs, e.g., if the data generated by the CTA(s) is needed for a subsequent processing step, and this can introduce some latency. (Such latency will generally be less than the latency associated with sequential processing techniques.) In some instances it may be possible to hide some or all of this latency through suitable sequencing or scheduling of program instructions, e.g., by arranging the program sequence so that the CTA is processed by GPU 122 while CPU 102 performs other functions that do not rely on the CTA data. It will be recognized that the extent to which latency can be hidden through software techniques will depend on the particular application. 26:48–62.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>1[c][i] “at least one graphics processing unit to perform a sequence of computations on the at least a portion of the input data so as to generate output data, the sequence of computations representing an artificial neural network,</p>	<p>Nickolls discloses “at least one graphics processing unit to perform a sequence of computations on the at least a portion of the input data so as to generate output data, the sequence of computations representing an artificial neural network, intermediate computations in the sequence of computations representing respective layers of the artificial neural network and yielding intermediate results.” <i>See e.g.:</i></p> <p><i>As a non-limiting example, Nickolls discloses at least one graphics processing unit (e.g., GPU 122) with cores and/or processing engines that can perform general-purpose computations on input data to generate output data (e.g., intermediate results and/or final output data):</i></p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438

intermediate computations in the sequence of computations representing respective layers of the artificial neural network and yielding intermediate results; and”

Nickolls

FIG. 1 is a block diagram of a computer system according to an embodiment of the present invention; 4:49–50.

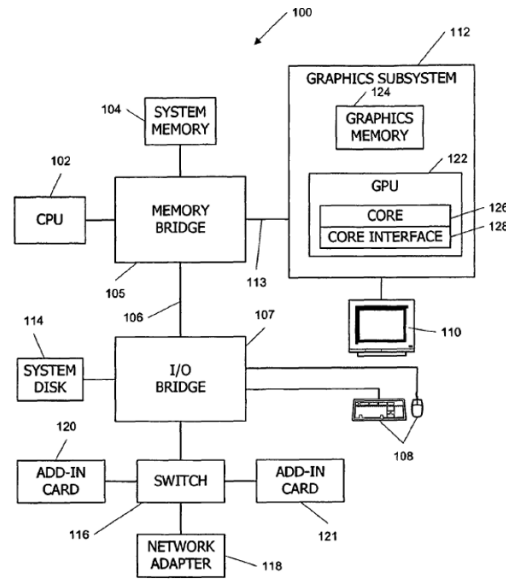


FIG. 1

FIG. 1.

FIG. 3 is a block diagram of a processing core according to an embodiment of the present invention; 4:64–65.

U.S. Patent No. RE48,438

Nickolls

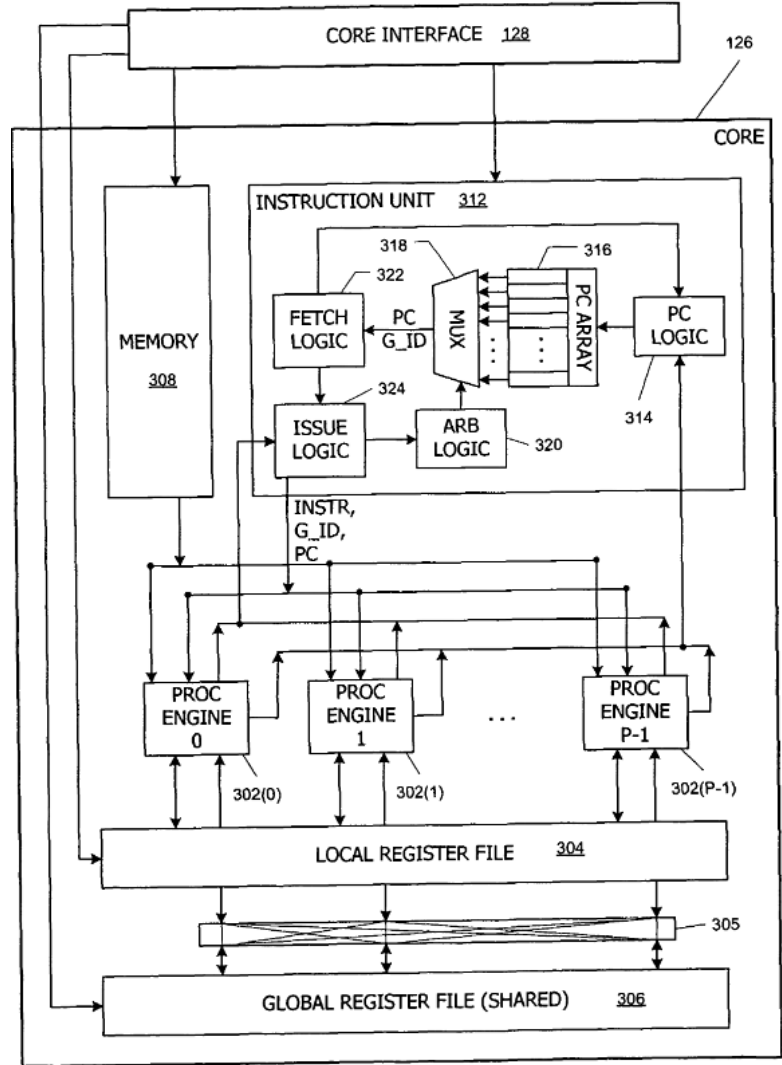


FIG. 3

FIG. 3.

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>Parallel data processing systems and methods use cooperative thread arrays (CTAs), i.e., groups of multiple threads that concurrently execute the same program on an input data set to produce an output data set. Each thread in a CTA has a unique identifier (thread ID) that can be assigned at thread launch time. The thread ID controls various aspects of the thread's processing behavior such as the portion of the input data set to be processed by each thread, the portion of an output data set to be produced by each thread, and/or sharing of intermediate results among threads. Mechanisms for loading and launching CTAs in a representative processing core and for synchronizing threads within a CTA are also described.</p> <p>Abstract.</p> <p>Graphics processing subsystem 112 includes a graphics processing unit (GPU) 122 and a graphics memory 124, which may be implemented, e.g., using one or more integrated circuit devices such as programmable processors, application specific integrated circuits (ASICs), and memory devices. GPU 122 may be configured to perform various tasks related to generating pixel data from graphics data supplied by CPU 102 and/or system memory 104 via memory bridge 105 and bus 113, interacting with graphics memory 124 to store and update pixel data, and the like. For example, GPU 122 may generate pixel data from 2-D or 3-D scene data provided by various programs executing on CPU 102.</p> <p>6:35–55.</p> <p>In accordance with an embodiment of the present invention, core 126 of GPU 122 is a multithreaded or parallel processing core that can be leveraged for general-purpose computations by executing cooperative thread arrays (CTAs). As used herein, a “CTA” is a group of multiple threads that concurrently execute the same program on an input data set to produce an output data set. Each thread in the CTA is assigned a unique thread identifier (“thread ID”) that is accessible to the thread during its execution. The thread ID controls various aspects of the thread's processing behavior. For instance, a thread ID may be used to determine which portion of the input data set a thread is to process, to identify one or more other threads with which a given thread is to share an intermediate result, and/or to determine which portion of an output data set a thread is to produce or write.</p> <p>7:51–65.</p> <p>CTAs are advantageously employed to perform computations that lend themselves to a data parallel decomposition, i.e., application of the same processing algorithm to different portions of an input data</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>set in order to effect a transformation of the input data set to an output data set. The processing algorithm is specified in a “CTA program,” and each thread in a CTA executes the same CTA program on a different subset of an input data set. A CTA program can implement algorithms using a wide range of mathematical and logical operations, and the program can include conditional or branching execution paths and direct and/or indirect memory access. 7:66–8:9.</p> <p>FIG. 3 is a block diagram of a processing core 126 according to an embodiment of the present invention. Core 126 is advantageously configured to execute a large number of threads in parallel. During a rendering operation, a thread might be an instance of a vertex shader program executing on the attributes of a single vertex or an instance of a pixel shader program executing on a given primitive and pixel. During general-purpose computing, a thread can be an instance of a CTA program executing on a portion of an input data set. In core 126, single-instruction, multiple-data (SIMD) instruction issue techniques are used to support parallel execution of a large number of threads by multiple processing engines without requiring multiple instruction units. 10:24–36.</p> <p>In one embodiment, core 126 includes an array of P (e.g., 8, 16, or any other number) parallel processing engines 302 configured to receive and execute SIMD instructions from a single instruction unit 312. Each parallel processing engine 302 advantageously includes an identical set of functional units such as arithmetic logic units, load/store units, and the like (not explicitly shown). The functional units may be pipelined, allowing a new instruction to be issued before a previous instruction has finished, as is known in the art. Any combination of functional units may be provided. In one embodiment, the functional units support a variety of operations including integer and floating point arithmetic (e.g., addition and multiplication), comparison operations, Boolean operations (AND, OR, XOR), bit-shifting, and computation of various algebraic functions (e.g., planar interpolation, trigonometric, exponential, and logarithmic functions, etc.); and the same functional-unit hardware can be leveraged to perform different operations. A particular implementation of processing engines 302 is not critical to the present invention, and a detailed description has been omitted. 10:37–56.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>Each processing engine 302 is allocated space in a local register file 304 for storing its local input data, intermediate results, and the like. In one embodiment, local register file 304 is divided into P lanes, each having some number of entries (where each entry might be, e.g., a 32-bit word). One lane is allocated to each processing unit, and corresponding entries in different lanes can be populated with data for corresponding thread types to facilitate SIMD execution of multiple threads in parallel as described below. The number of entries in local register file 304 is advantageously large enough to support multiple concurrent threads per processing engine 302; and in some embodiments, the number of entries allocated to a thread is dynamically configurable. 10:57–11:2.</p> <p>Once all of the input data has been received, core interface 128 enters the launch phase. In the launch phase, core interface 128 selects a group index GID for a first SIMD group to be launched (step 1018); any group index GID that is not already in use in core 126 may be selected. At step 1020, core interface 128 loads the initial set of P thread IDs into the local register file 304 of core 126 in locations corresponding to the selected group index GID or into per-thread registers in core 126 dedicated to storing thread IDs. Core interface 126 then instructs core 126 to launch the P threads as a SIMD group (step 1022). 19:51–61.</p> <p>As described above, core 126 advantageously implements a multithreaded architecture, with SIMD groups of threads being executed in parallel and multiple SIMD groups executing concurrently. Maximum efficiency is obtained when the threads in a SIMD group do not diverge; however, in practice, threads are allowed to diverge without restriction. For instance, a program being executed by all threads in a SIMD group may include a conditional branch instruction, and the branch may be taken by some threads and not taken by others; as described above, core 126 can be configured to handle such instances. Threads may also diverge in other ways, e.g., due to conditional execution of various instructions that might be executed by some threads but not others within a SIMD group. Thus, CTA programs of arbitrary complexity can be implemented. 20:20–45.</p> <p>On completion of the CTA program, the final results (output data) produced by the threads are advantageously placed in memory for use by a subsequent CTA program or made accessible to CPU</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>102 (FIG. 1). For example, the final instructions in a CTA program might include an instruction to write some or all of the data generated by the thread to graphics memory 124. After execution of the CTA is finished, GPU 122 may transfer the data (e.g., using a conventional DMA operation) to system memory 104, making it available to application programs executing on CPU 102. Other data transfer mechanisms may also be used. Core 126 advantageously signals core interface 128 upon completion of a CTA, so that core interface 128 can initiate execution of a next CTA, reusing the resources that became free when the first CTA was completed. 20:61–21:8.</p> <p>In some embodiments using a GPU, CTA threads and rendering threads can coexist in the same processor, e.g., in different cores or in the same core. Further, in systems with multiple GPUs, one GPU may be used for rendering images while another is used for general-purpose computations including CTAs. Alternatively, each GPU may be assigned a different portion of a general-purpose computation; allowing multiple GPUs to execute CTAs in parallel further increases the number of CTAs that can be executed in parallel. 30:52–60.</p> <p><i>See also</i> 1[c].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>1[c][ii] “accelerator memory, operably coupled to the at least one graphics processing unit, to store the</p>	<p>Nickolls discloses “accelerator memory, operably coupled to the at least one graphics processing unit, to store the results of the sequence of computations.” <i>See e.g.:</i></p> <p><i>As a nonlimiting example, Nickolls discloses accelerator memory (e.g., local register file 304, shared global register file 306, and/or memory 308) operably coupled to a graphics processing unit (e.g., GPU</i></p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
<p>results of the sequence of computations; and”</p>	<p><i>122), which can store input and output data, such as the results of general-purpose computations performed by the graphics process unit:</i></p> <p>Threads in a CTA can share intermediate results with other threads in the same CTA using a shared memory that is accessible to all of the threads, an interconnection network, or other technologies for inter-thread communication, including other technologies known in the art. In some embodiments, the CTA program includes an instruction to compute an address in shared memory to which particular data is to be written, with the address being a function of thread ID. Each thread computes the function using its own thread ID and writes to the corresponding location. The function is advantageously defined such that different threads write to different locations; as long as the function is deterministic, the location written to by any thread is well-defined. The CTA program can also include an instruction to compute an address in shared memory from which data is to be read, with the address being a function of thread ID. By defining suitable functions, data can be written to a given location by one thread and read from that location by a different thread in a predictable manner. Consequently, any desired pattern of data sharing among threads can be supported, and any thread in a CTA can share data with any other thread in the same CTA.</p> <p>8:10–30.</p> <p>Each processing engine 302 is allocated space in a local register file 304 for storing its local input data, intermediate results, and the like. In one embodiment, local register file 304 is divided into P lanes, each having some number of entries (where each entry might be, e.g., a 32-bit word). One lane is allocated to each processing unit, and corresponding entries in different lanes can be populated with data for corresponding thread types to facilitate SIMD execution of multiple threads in parallel as described below. The number of entries in local register file 304 is advantageously large enough to support multiple concurrent threads per processing engine 302; and in some embodiments, the number of entries allocated to a thread is dynamically configurable.</p> <p>10:57–11:2.</p> <p>Each processing engine 302 also has access, via a crossbar switch 305, to a (shared) global register file 306 that is shared among all of the processing engines 302 in core 126. Global register file 306 may be as large as desired, and in some embodiments, any processing engine 302 can read to or write from any location in global register file 306. In addition to global register file 306, some embodiments also</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>provide an on-chip shared memory 308, which may be implemented, e.g., as a conventional RAM. On-chip memory 308 is advantageously used to store data that is expected to be used in multiple threads, such as coefficients of attribute equations, which are usable in pixel shader programs. In some embodiments, processing engines 302 may also have access to additional off-chip shared memory (not shown), which might be located, e.g., within graphics memory 124 and/or system memory 104 of FIG. 1. 11:3–18.</p> <p>For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302 with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group. 13:12–45.</p> <p>The CTA program may also include instructions to read from and/or write to the shared global register file 306, on-chip shared memory 308, and/or other memory such as graphics memory 124 or system</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>memory 104 of FIG. 1. For instance, the input data set to be processed by the CTA may be stored in graphics memory 124 or system memory 104. Intermediate results may be written to global register file 306 and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1, where they can be shared with other threads. Final results (output data) may be written to graphics memory 124 or system memory 104. 20:35–45.</p> <p>FIG. 3 is a block diagram of a processing core according to an embodiment of the present invention; 4:64–65.</p>

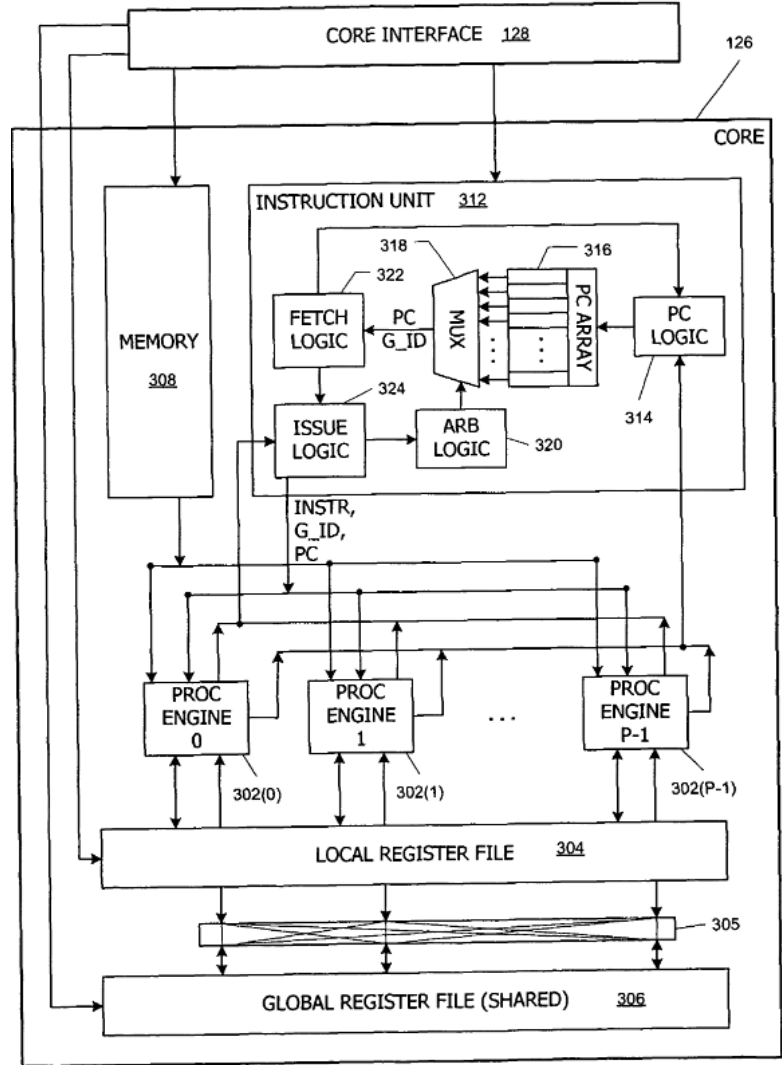


FIG. 3

FIG. 3.

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p><i>See also</i> 1[c].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>1[d] “a controller, operably coupled to the at least one graphics processing unit and the accelerator memory,”</p>	<p>Nickolls discloses “a controller, operably coupled to the at least one graphics processing unit and the accelerator memory.” <i>See e.g.:</i></p> <p><i>As a non-limiting example, Nickolls discloses a controller (e.g., core interface 128 and/or instruction unit 312), which is coupled to at least one graphics processing unit (e.g., core 126 and/or processing engines) and accelerator memory (e.g., local register file 304, shared global register file 306, and/or memory 308):</i></p> <p>GPU 122 has at least one processing core 126 for generating pixel data and a core interface 128 that controls operation of core 126. Core 126 advantageously includes multiple parallel processing engines that can be used to execute various shader programs, including vertex shader programs, geometry shader programs, and/or pixel shader programs, in the course of generating images from scene data. Core 126 can also be leveraged to perform general-purpose computations as described below. 6:47–55.</p> <p>Each processing engine 302 is allocated space in a local register file 304 for storing its local input data, intermediate results, and the like. In one embodiment, local register file 304 is divided into P lanes, each having some number of entries (where each entry might be, e.g., a 32-bit word). One lane is allocated to each processing unit, and corresponding entries in different lanes can be populated with data for corresponding thread types to facilitate SIMD execution of multiple threads in parallel as described below. The number of entries in local register file 304 is advantageously large enough to</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>support multiple concurrent threads per processing engine 302; and in some embodiments, the number of entries allocated to a thread is dynamically configurable. 10:57–11:2.</p> <p>Each processing engine 302 also has access, via a crossbar switch 305, to a (shared) global register file 306 that is shared among all of the processing engines 302 in core 126. Global register file 306 may be as large as desired, and in some embodiments, any processing engine 302 can read to or write from any location in global register file 306. In addition to global register file 306, some embodiments also provide an on-chip shared memory 308, which may be implemented, e.g., as a conventional RAM. On-chip memory 308 is advantageously used to store data that is expected to be used in multiple threads, such as coefficients of attribute equations, which are usable in pixel shader programs. In some embodiments, processing engines 302 may also have access to additional off-chip shared memory (not shown), which might be located, e.g., within graphics memory 124 and/or system memory 104 of FIG. 1. 11:3–18.</p> <p>For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302 with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group. 13:12–45.</p> <p>It will be appreciated that the processing core described herein is illustrative and that variations and modifications are possible. Any number of processing units may be included. In some embodiments, each processing unit has its own local register file, and the allocation of local register file entries per thread can be fixed or configurable as desired. 13:46–51.</p> <p>In some embodiments, core 126 is operated at a higher clock rate than core interface 128, allowing the core to process more data in a given amount of time. For instance, core 126 can be operated at a clock rate that is twice the clock rate of core interface 128. If core 126 includes P processing engines 302 producing data at twice the core interface clock rate, then core 126 can produce 2*P data values per core interface clock cycle. Provided there is sufficient space in local register file 304, from the perspective of core interface 128, the situation is effectively identical to a core with 2*P processing units. Thus, P-way SIMD parallelism could be produced either by including P processing units in core 126 and operating core 126 at the same clock rate as core interface 128 or by including P/2 processing units in core 126 and operating core 126 at twice the clock rate of core interface 128. Other timing variations are also possible. 13:52–67.</p> <p>FIG. 4 is a block diagram of core interface 128 according to an embodiment of the present invention. Core interface 128 includes an input unit 402, a state module 404, state registers 406, a load module 408, and a launch module 410. As described above, core interface 128 (FIG. 1) controls operation of core 126. In particular, core interface 128 loads and launches threads of a CTA in SIMD groups until all threads have been launched. In one embodiment, core interface 128 generates thread IDs for each thread in a SIMD group and writes the thread ID into a suitable location in local register file 304, then launches the SIMD group. These aspects of core interface 128 will now be described. 14:17–28.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	FIG. 3 is a block diagram of a processing core according to an embodiment of the present invention; 4:64–65.

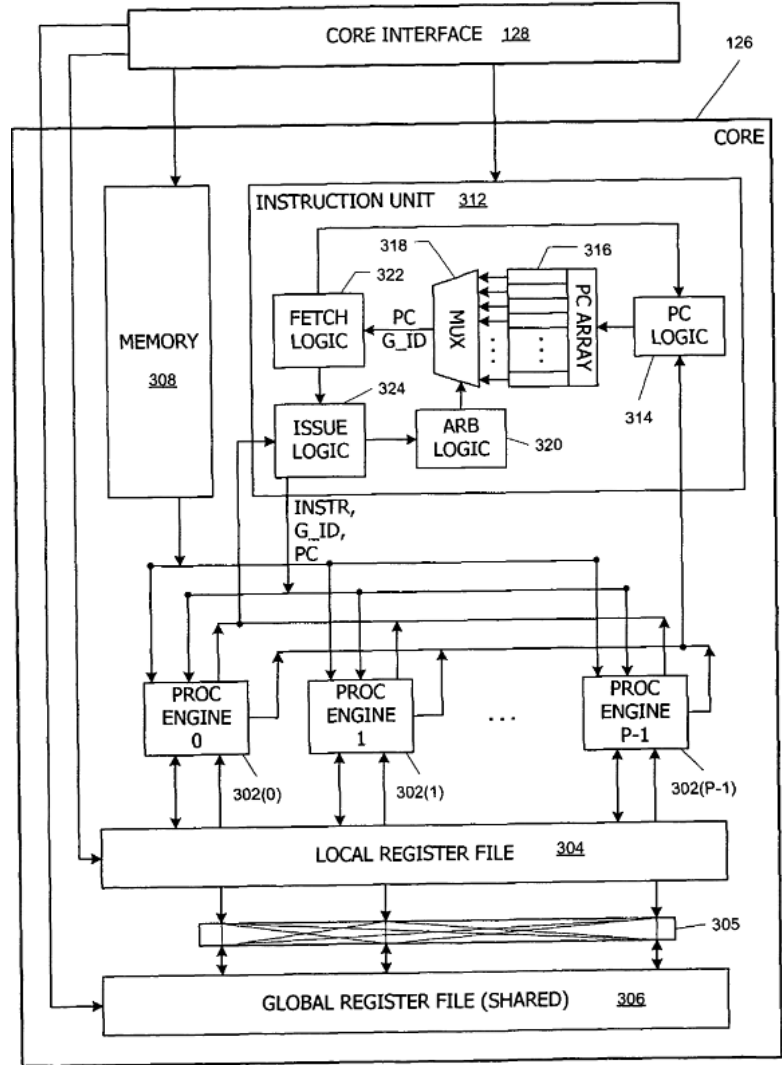


FIG. 3

FIG. 3.

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
---------------------------------	-----------------

FIG. 4 is a block diagram of a core interface for a processing core according to an embodiment of the present invention;
4:66–67.

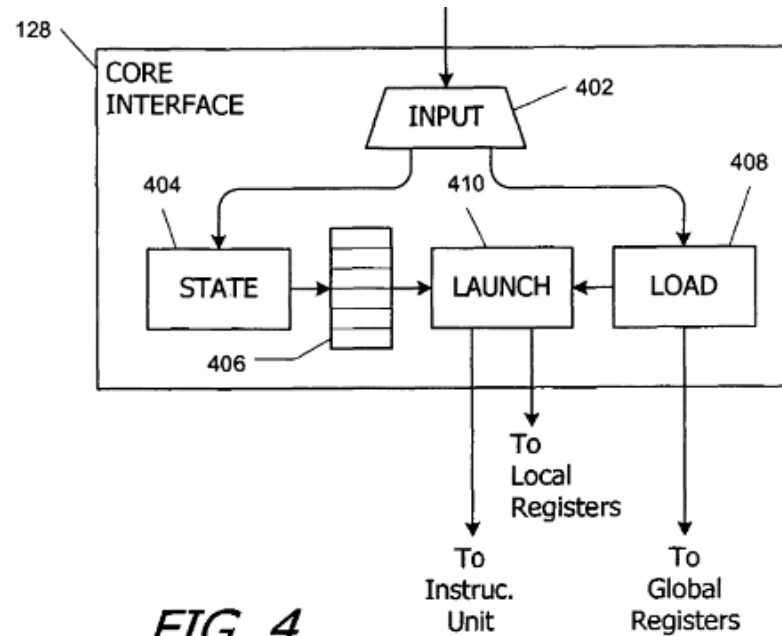


FIG. 4.

Instruction unit 312 advantageously manages instruction fetch and issue for each SIMD group so as to ensure that threads in a group that have diverged eventually resynchronize. In one embodiment, instruction unit 312 includes program counter (PC) logic 314, a program counter register array 316, a multiplexer 318, arbitration logic 320, fetch logic 322, and issue logic 324. Program counter register array 316 stores G program counter values (one per SIMD group), which are updated independently of each other by PC logic 314. PC logic 314 updates the PC values based on information received from processing engines 302 and/or fetch logic 322. PC logic 314 is advantageously configured to track

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>divergence among threads in a SIMD group and to select instructions in a way that ultimately results in the threads re-synchronizing. 12:19–33.</p> <p>In some embodiments of the present invention, a barrier synchronization technique is advantageously used to support fast synchronization of any number of CTA threads. More specifically, barrier instructions are inserted into the CTA program at points (referred to herein as “barrier points”) where synchronization is desired. A thread executes a barrier instruction to indicate that it has arrived at a barrier point and waits at that point until all other participating threads have also arrived at that point, thus synchronizing the participating threads before resuming execution. Arrival of each thread (or group of threads) at a barrier point is detected, and this information is used to synchronize two or more threads (or groups of threads). In one embodiment, execution of barrier instructions, i.e., arrival of threads (or SIMD groups) at barrier points, is detected by issue logic 324 of instruction unit 312 of FIG. 3, which can suspend the issue of instructions to any threads that are waiting at a barrier point while continuing to issue instructions to threads that are not at a barrier point. Eventually, all relevant threads reach the barrier point, and execution of the waiting threads resumes. 21:25–44.</p> <p>FIG. 11A is a block diagram of barrier synchronization logic 1100 according to an embodiment of the present invention. In some embodiments, barrier synchronization logic 1100 is implemented in issue logic 324 of instruction unit 312 and synchronizes SIMD groups rather than individual threads. 21:45–50.</p> <p>As shown in FIG. 11A, instruction unit 312 also includes selection logic 1110 that selects a next instruction to issue. Selection logic 1110 may be of generally conventional design, and a detailed description is omitted as not being critical to understanding the present invention. Barrier detection circuit 1112 receives each selected instruction (INST), along with the group identifier (GID) of the SIMD group for which the instruction is being issued. If the selected instruction is a barrier instruction, barrier detection circuit 1112 directs the instruction to barrier synchronization logic 1100; otherwise, barrier detection circuit 1112 forwards the instruction to the next issue stage for eventual delivery to processing engines 302 of FIG. 3. 21:51–63.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>1[d][i] “to initialize textures and shaders in the accelerator memory for performing the sequence of computations,”</p>	<p>Nickolls discloses a controller “to initialize textures and shaders in the accelerator memory for performing the sequence of computations.” <i>See e.g.:</i></p> <p><i>As a non-limiting example, Nickolls discloses initializing textures (e.g., input parameters and/or input data) and shaders (e.g., programs and/or instructions) in accelerator memory (e.g., local register file 304, shared global register file 306, and/or memory 308) for performing computations:</i></p> <p>Embodiments of the present invention provide data processing systems and methods that use cooperative thread arrays (CTAs) to perform computations. As used herein, a “cooperative thread array,” or “CTA,” is a group of multiple threads that concurrently execute the same program on an input data set to produce an output data set. 2:22–27.</p> <p>Graphics processing subsystem 112 includes a graphics processing unit (GPU) 122 and a graphics memory 124, which may be implemented, e.g., using one or more integrated circuit devices such as programmable processors, application specific integrated circuits (ASICs), and memory devices. GPU 122 may be configured to perform various tasks related to generating pixel data from graphics data supplied by CPU 102 and/or system memory 104 via memory bridge 105 and bus 113, interacting with graphics memory 124 to store and update pixel data, and the like. For example, GPU 122 may generate pixel data from 2-D or 3-D scene data provided by various programs executing on CPU 102. 6:35–46.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>GPU 122 has at least one processing core 126 for generating pixel data and a core interface 128 that controls operation of core 126. Core 126 advantageously includes multiple parallel processing engines that can be used to execute various shader programs, including vertex shader programs, geometry shader programs, and/or pixel shader programs, in the course of generating images from scene data. Core 126 can also be leveraged to perform general-purpose computations as described below. 6:47–55.</p> <p>In accordance with an embodiment of the present invention, core 126 of GPU 122 is a multithreaded or parallel processing core that can be leveraged for general-purpose computations by executing cooperative thread arrays (CTAs). As used herein, a “CTA” is a group of multiple threads that concurrently execute the same program on an input data set to produce an output data set. Each thread in the CTA is assigned a unique thread identifier (“thread ID”) that is accessible to the thread during its execution. The thread ID controls various aspects of the thread's processing behavior. For instance, a thread ID may be used to determine which portion of the input data set a thread is to process, to identify one or more other threads with which a given thread is to share an intermediate result, and/or to determine which portion of an output data set a thread is to produce or write. 7:51–65.</p> <p>CTAs are advantageously employed to perform computations that lend themselves to a data parallel decomposition, i.e., application of the same processing algorithm to different portions of an input data set in order to effect a transformation of the input data set to an output data set. The processing algorithm is specified in a “CTA program,” and each thread in a CTA executes the same CTA program on a different subset of an input data set. A CTA program can implement algorithms using a wide range of mathematical and logical operations, and the program can include conditional or branching execution paths and direct and/or indirect memory access. 7:66–8:9.</p> <p>Threads in a CTA can share intermediate results with other threads in the same CTA using a shared memory that is accessible to all of the threads, an interconnection network, or other technologies for inter-thread communication, including other technologies known in the art. In some embodiments, the CTA program includes an instruction to compute an address in shared memory to which particular data is to be written, with the address being a function of thread ID. Each thread computes the function using</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>its own thread ID and writes to the corresponding location. The function is advantageously defined such that different threads write to different locations; as long as the function is deterministic, the location written to by any thread is well-defined. The CTA program can also include an instruction to compute an address in shared memory from which data is to be read, with the address being a function of thread ID. By defining suitable functions, data can be written to a given location by one thread and read from that location by a different thread in a predictable manner. Consequently, any desired pattern of data sharing among threads can be supported, and any thread in a CTA can share data with any other thread in the same CTA. 8:10–30.</p> <p>Each processing engine 302 is allocated space in a local register file 304 for storing its local input data, intermediate results, and the like. In one embodiment, local register file 304 is divided into P lanes, each having some number of entries (where each entry might be, e.g., a 32-bit word). One lane is allocated to each processing unit, and corresponding entries in different lanes can be populated with data for corresponding thread types to facilitate SIMD execution of multiple threads in parallel as described below. The number of entries in local register file 304 is advantageously large enough to support multiple concurrent threads per processing engine 302; and in some embodiments, the number of entries allocated to a thread is dynamically configurable. 10:57–11:2.</p> <p>Each processing engine 302 also has access, via a crossbar switch 305, to a (shared) global register file 306 that is shared among all of the processing engines 302 in core 126. Global register file 306 may be as large as desired, and in some embodiments, any processing engine 302 can read to or write from any location in global register file 306. In addition to global register file 306, some embodiments also provide an on-chip shared memory 308, which may be implemented, e.g., as a conventional RAM. On-chip memory 308 is advantageously used to store data that is expected to be used in multiple threads, such as coefficients of attribute equations, which are usable in pixel shader programs. In some embodiments, processing engines 302 may also have access to additional off-chip shared memory (not shown), which might be located, e.g., within graphics memory 124 and/or system memory 104 of FIG. 1. 11:3–18.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302 with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group. 13:12–45.</p> <p>State module 404 receives state information from input unit 402 and loads the state information into state registers 406. “State information,” as used herein, includes any information (other than input data) relevant to defining a CTA. For example, in one embodiment, state information includes the size of the input data set, the amount of local register file space required for each thread, and a starting program counter (e.g., memory address) for a program to be executed by each thread. State information advantageously also includes size information for the CTA; for example, referring to FIG. 2A, array dimensions D0, D1 and D2 may be provided. In some embodiments, the total number (7) of threads is also provided; in other embodiments, T can be computed from the array dimensions (e.g., $T=D0*D1*D2$ for the embodiment of FIG. 2A). When T is provided, T is advantageously less than or equal to $D0*D1*D2$, in order to ensure that each thread is assigned a unique thread ID.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>14:41–57.</p> <p>Referring again to FIG. 4, load module 408 receives input parameters for a CTA and loads the input parameters into (shared) global register file 306 (FIG. 3) of core 126. These parameters may include, e.g., an (x,y) position of the CTA within a grid of CTAs or other CTA identifier and/or other information specific to the CTA. In some embodiments, the input parameters may also include some or all of an input data set to be processed by the CTA.</p> <p>16:63–17:3.</p> <p>Core interface 128 remains in the state phase and continues to update the state registers as new state information is received, until such time as core interface 128 receives a “begin CTA” command (step 1010) indicating that input data is to follow. At that point, core interface 128 enters the load phase. During the load phase, core interface 128 receives input data (step 1012) and loads the input data into (shared) global register file 306 of core 126 (step 1014). At step 1016, core interface 128 determines whether the last input data has been received. If not, core interface 128 returns to step 1012 to receive more input data.</p> <p>19:40–50.</p> <p>The CTA program may also include instructions to read from and/or write to the shared global register file 306, on-chip shared memory 308, and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1. For instance, the input data set to be processed by the CTA may be stored in graphics memory 124 or system memory 104. Intermediate results may be written to global register file 306 and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1, where they can be shared with other threads. Final results (output data) may be written to graphics memory 124 or system memory 104.</p> <p>20:35–45.</p> <p>In the embodiment shown in FIG. 12, processing clusters 1202 are designed to process vertex and/or pixel data during rendering operations of GPU 122, and execution of CTAs for general-purpose computation is supported by leveraging the existing rendering hardware to the extent possible. Accordingly, in this embodiment, core interface 128 can receive control signals and data from a geometry controller 1204 or a pixel/CTA controller 1206. During rendering operations, geometry</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>controller 1204 receives geometry data (GDATA) from the rendering pipeline (not explicitly shown) of GPU 122 and forwards the data to core interface 128, which controls execution of vertex and/or geometry shader programs on the geometry data. The processed geometry data (GDATA') is returned to the rendering pipeline via geometry controller 1204. 28:9–23.</p> <p>FIG. 3 is a block diagram of a processing core according to an embodiment of the present invention; 4:64–65.</p>

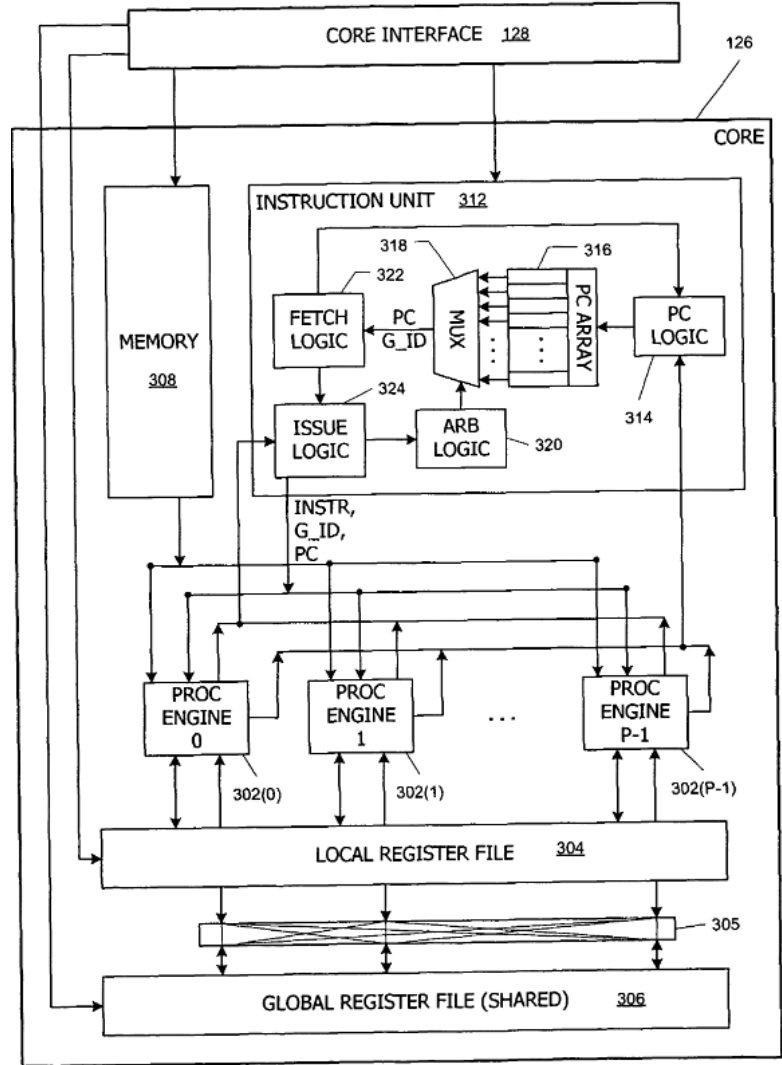
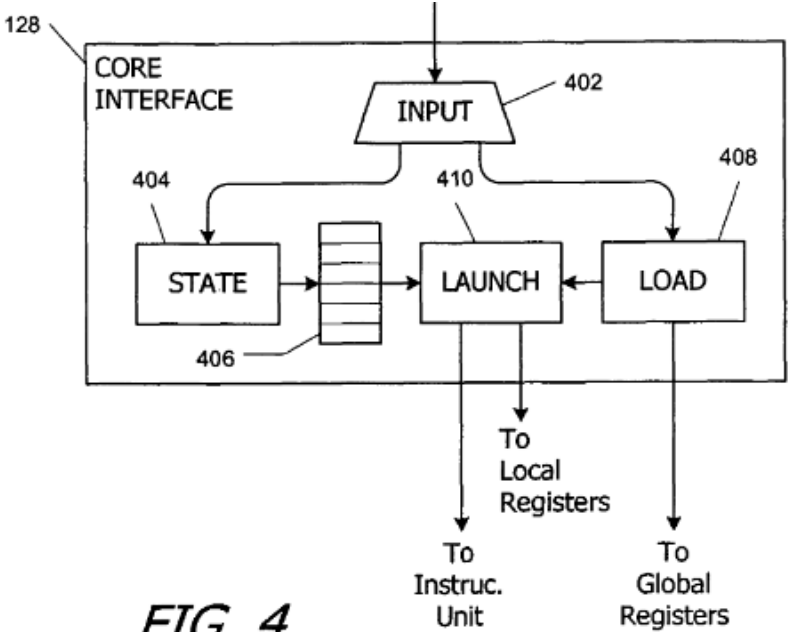


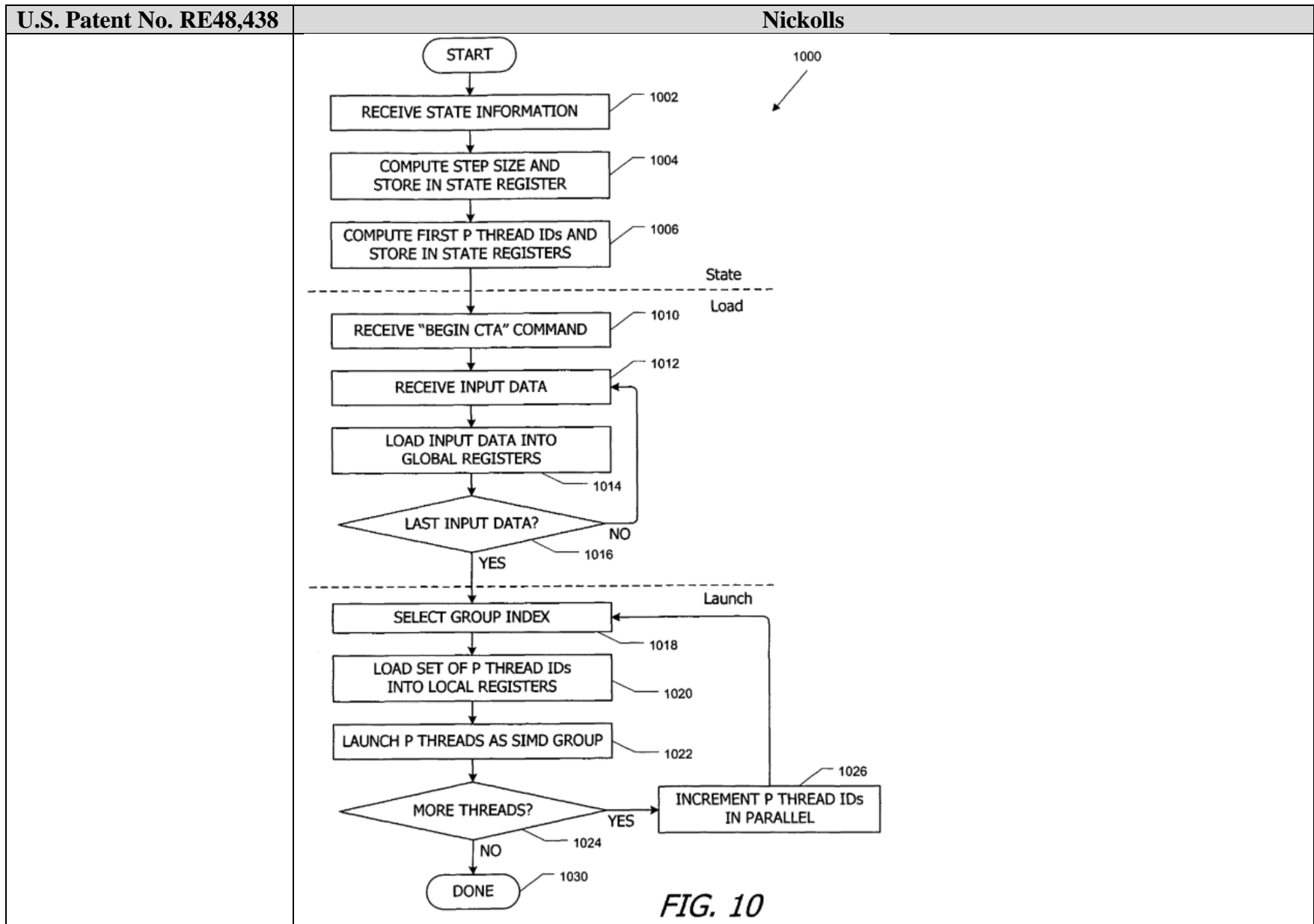
FIG. 3

FIG. 3.

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p data-bbox="583 235 1879 337">FIG. 4 is a block diagram of a core interface for a processing core according to an embodiment of the present invention; 4:66–67.</p>  <p data-bbox="722 971 877 1019">FIG. 4</p> <p data-bbox="583 1047 676 1079">FIG. 4.</p> <p data-bbox="583 1122 1759 1224">FIG. 10 is a flow diagram of a control process performed by a core interface according to an embodiment of the present invention; 5:16–17.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls



Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>FIG. 10.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>1[d][ii] “to control performance of the sequence of computations by the at least one graphics processing unit,”</p>	<p>Nickolls discloses a controller “to control performance of the sequence of computations by the at least one graphics processing unit.” <i>See e.g.:</i></p> <p><i>As a non-limiting example, Nickolls discloses that the controller (e.g., core interface 128 and/or instruction unit 312 control operation of core 126) controls performance of the computations by the at least one graphics processing unit (e.g., GPU 122 and/or processing engines):</i></p> <p>GPU 122 has at least one processing core 126 for generating pixel data and a core interface 128 that controls operation of core 126. Core 126 advantageously includes multiple parallel processing engines that can be used to execute various shader programs, including vertex shader programs, geometry shader programs, and/or pixel shader programs, in the course of generating images from scene data. Core 126 can also be leveraged to perform general-purpose computations as described below. 6:47–55.</p> <p>Instruction unit 312 is configured such that, for any given processing cycle, the same instruction (INSTR) is issued to all P processing engines 302. Thus, at the level of a single clock cycle, core 126 implements P-way SIMD execution. Each processing engine 302 is also multithreaded, supporting up to G concurrent threads, and instructions for different ones of the G threads can be issued in any order relative to each other. Accordingly, core 126 in this embodiment can have up to P*G threads in flight concurrently. For instance, if P=16 and G=24, then core 126 can support up to 384 concurrent threads. In some embodiments, P*G determines an upper limit on the number of threads that can be included in a CTA; it is to be understood that some CTAs may include fewer than this number of threads.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>11:27–40.</p> <p>Instruction unit 312 advantageously manages instruction fetch and issue for each SIMD group so as to ensure that threads in a group that have diverged eventually resynchronize. In one embodiment, instruction unit 312 includes program counter (PC) logic 314, a program counter register array 316, a multiplexer 318, arbitration logic 320, fetch logic 322, and issue logic 324. Program counter register array 316 stores G program counter values (one per SIMD group), which are updated independently of each other by PC logic 314. PC logic 314 updates the PC values based on information received from processing engines 302 and/or fetch logic 322. PC logic 314 is advantageously configured to track divergence among threads in a SIMD group and to select instructions in a way that ultimately results in the threads re-synchronizing.</p> <p>12:19–33.</p> <p>Arbitration logic 320 and multiplexer 318 determine the order in which instructions are fetched. More specifically, on each clock cycle, arbitration logic 320 selects one of the G possible group indices GID as the SIMD group for which a next instruction should be fetched and supplies a corresponding control signal to multiplexer 318, which selects the corresponding PC. Arbitration logic 320 may include conventional logic for prioritizing and selecting among concurrent threads (e.g., using round-robin, least-recently serviced, or the like), and selection may be based in part on feedback information from fetch logic 322 or issue logic 324 as to how many instructions have been fetched but not yet issued for each SIMD group.</p> <p>12:43–55.</p> <p>For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302 with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group. 13:12–45.</p> <p>It will be appreciated that the processing core described herein is illustrative and that variations and modifications are possible. Any number of processing units may be included. In some embodiments, each processing unit has its own local register file, and the allocation of local register file entries per thread can be fixed or configurable as desired. 13:46–51.</p> <p>In some embodiments, core 126 is operated at a higher clock rate than core interface 128, allowing the core to process more data in a given amount of time. For instance, core 126 can be operated at a clock rate that is twice the clock rate of core interface 128. If core 126 includes P processing engines 302 producing data at twice the core interface clock rate, then core 126 can produce 2*P data values per core interface clock cycle. Provided there is sufficient space in local register file 304, from the perspective of core interface 128, the situation is effectively identical to a core with 2*P processing units. Thus, P-way SIMD parallelism could be produced either by including P processing units in core 126 and operating core 126 at the same clock rate as core interface 128 or by including P/2 processing units in core 126 and operating core 126 at twice the clock rate of core interface 128. Other timing variations are also possible. 13:52–67.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>FIG. 4 is a block diagram of core interface 128 according to an embodiment of the present invention. Core interface 128 includes an input unit 402, a state module 404, state registers 406, a load module 408, and a launch module 410. As described above, core interface 128 (FIG. 1) controls operation of core 126. In particular, core interface 128 loads and launches threads of a CTA in SIMD groups until all threads have been launched. In one embodiment, core interface 128 generates thread IDs for each thread in a SIMD group and writes the thread ID into a suitable location in local register file 304, then launches the SIMD group. These aspects of core interface 128 will now be described. 14:17–28.</p> <p>Referring again to FIG. 4, load module 408 receives input parameters for a CTA and loads the input parameters into (shared) global register file 306 (FIG. 3) of core 126. These parameters may include, e.g., an (x,y) position of the CTA within a grid of CTAs or other CTA identifier and/or other information specific to the CTA. In some embodiments, the input parameters may also include some or all of an input data set to be processed by the CTA. 16:63–17:3.</p> <p>In one embodiment, the first received input parameter is accompanied by a “begin CTA” control signal indicating that what follows are input parameters for a CTA that is to be executed using the currently defined state. The control signal may also indicate the size of the input parameter set, where in global register file 306 the parameters are to be stored, and other information, e.g., a control signal identifying the last of the input parameters so that core interface 128 immediately recognizes when the CTA is ready to be launched. Alternatively, state information defining the size of the input parameter set may be stored in state registers 406, and load module 408 may use this information to determine when loading of input parameters is complete. For instance, if state registers 406 store information about the number of input parameters, load module 408 might count received input parameters until the expected number had been received. 17:4–19.</p> <p>Once all the input parameters have been loaded into global register file 306, load module 408 issues a “GO” control signal to launch module 410. Launch module 410 is configured to respond to the GO control signal by launching all of the threads for one CTA based on the current definition of CTA size stored in state registers 406. Launch module 410 supplies each thread with a unique thread ID, which is written to one of the local registers 304 (FIG. 3) allocated to that thread or to a per-thread register</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>dedicated to storing a thread ID. Once all the threads in a SIMD group have been supplied with thread IDs, launch module 410 advantageously launches that SIMD group before beginning to supply thread IDs for the next SIMD group. Launch module 410 is advantageously designed to generate and assign thread IDs rapidly so as to minimize the delay between launching successive SIMD groups. In one embodiment, the delay can be as little as one clock cycle between successive launches. 17:20–36.</p> <p>Core interface 128 remains in the state phase and continues to update the state registers as new state information is received, until such time as core interface 128 receives a “begin CTA” command (step 1010) indicating that input data is to follow. At that point, core interface 128 enters the load phase. During the load phase, core interface 128 receives input data (step 1012) and loads the input data into (shared) global register file 306 of core 126 (step 1014). At step 1016, core interface 128 determines whether the last input data has been received. If not, core interface 128 returns to step 1012 to receive more input data. 19:40–50.</p> <p>Once all of the input data has been received, core interface 128 enters the launch phase. In the launch phase, core interface 128 selects a group index GID for a first SIMD group to be launched (step 1018); any group index GID that is not already in use in core 126 may be selected. At step 1020, core interface 128 loads the initial set of P thread IDs into the local register file 304 of core 126 in locations corresponding to the selected group index GID or into per-thread registers in core 126 dedicated to storing thread IDs. Core interface 126 then instructs core 126 to launch the P threads as a SIMD group (step 1022). 19:51–61.</p> <p>In some embodiments of the present invention, a barrier synchronization technique is advantageously used to support fast synchronization of any number of CTA threads. More specifically, barrier instructions are inserted into the CTA program at points (referred to herein as “barrier points”) where synchronization is desired. A thread executes a barrier instruction to indicate that it has arrived at a barrier point and waits at that point until all other participating threads have also arrived at that point, thus synchronizing the participating threads before resuming execution. Arrival of each thread (or group of threads) at a barrier point is detected, and this information is used to synchronize two or more</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>threads (or groups of threads). In one embodiment, execution of barrier instructions, i.e., arrival of threads (or SIMD groups) at barrier points, is detected by issue logic 324 of instruction unit 312 of FIG. 3, which can suspend the issue of instructions to any threads that are waiting at a barrier point while continuing to issue instructions to threads that are not at a barrier point. Eventually, all relevant threads reach the barrier point, and execution of the waiting threads resumes. 21:25–44.</p> <p>FIG. 11A is a block diagram of barrier synchronization logic 1100 according to an embodiment of the present invention. In some embodiments, barrier synchronization logic 1100 is implemented in issue logic 324 of instruction unit 312 and synchronizes SIMD groups rather than individual threads. 21:45–50.</p> <p>As shown in FIG. 11A, instruction unit 312 also includes selection logic 1110 that selects a next instruction to issue. Selection logic 1110 may be of generally conventional design, and a detailed description is omitted as not being critical to understanding the present invention. Barrier detection circuit 1112 receives each selected instruction (INST), along with the group identifier (GID) of the SIMD group for which the instruction is being issued. If the selected instruction is a barrier instruction, barrier detection circuit 1112 directs the instruction to barrier synchronization logic 1100; otherwise, barrier detection circuit 1112 forwards the instruction to the next issue stage for eventual delivery to processing engines 302 of FIG. 3. 21:51–63.</p> <p>In operation, when barrier synchronization logic 1130 receives a first barrier instruction pertaining to a barrier point BarID, the target value is loaded into the corresponding target register 1135. For every barrier instruction pertaining to barrier point BarID (including the first), the corresponding counter 1134 is incremented. In addition, if the barrier instruction indicates that the SIMD group is to wait for synchronization, the wait/go bit corresponding to group GID is set to the wait state in wait/go registers 1138, and the barrier identifier BarID is stored in the BarID field for that group. As described above, wait/go registers 1138 are advantageously read by selection logic 1110, and selection logic 1110 does not select instructions for SIMD groups that are in the wait state, thereby suspending execution of such groups. Selection logic 1110 may continue to select instructions for other SIMD groups. 23:52–67.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438 Nickolls

FIG. 4 is a block diagram of a core interface for a processing core according to an embodiment of the present invention;
4:66–67.

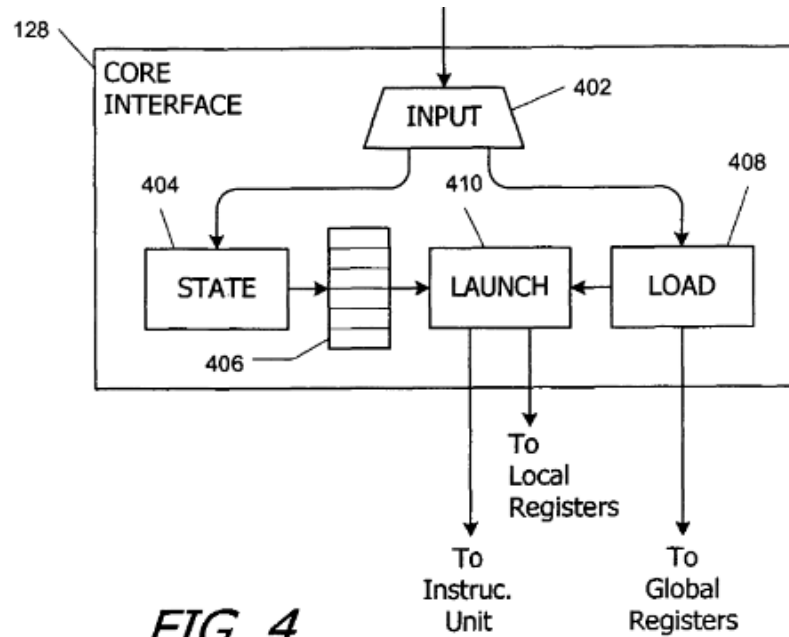


FIG. 4

FIG. 4.

To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>1[d][iii] “to transfer the at least a portion of the input data into the accelerator memory during performance of the intermediate computations in the sequence of computations by the at least one graphics processing unit, and”</p>	<p>Nickolls discloses a controller “to transfer the at least a portion of the input data into the accelerator memory during performance of the intermediate computations in the sequence of computations by the at least one graphics processing unit.” <i>See e.g.:</i></p> <p><i>See</i> 1[c], 1[c][ii].</p> <p><i>As a non-limiting example, Nickolls discloses the controller (e.g., core interface 128 and/or instruction unit 312) transferring at least a portion of the input data into the accelerator memory (e.g., local register file 304, shared global register file 306, and/or memory 308) during performance of the intermediate computations in the sequence of computations by at least one graphics processing unit (e.g., GPU 122 and/or processing engines):</i></p> <p>Each processing engine 302 is allocated space in a local register file 304 for storing its local input data, intermediate results, and the like. In one embodiment, local register file 304 is divided into P lanes, each having some number of entries (where each entry might be, e.g., a 32-bit word). One lane is allocated to each processing unit, and corresponding entries in different lanes can be populated with data for corresponding thread types to facilitate SIMD execution of multiple threads in parallel as described below. The number of entries in local register file 304 is advantageously large enough to support multiple concurrent threads per processing engine 302; and in some embodiments, the number of entries allocated to a thread is dynamically configurable.</p> <p>10:57–11:2.</p> <p>Each processing engine 302 also has access, via a crossbar switch 305, to a (shared) global register file 306 that is shared among all of the processing engines 302 in core 126. Global register file 306 may be as large as desired, and in some embodiments, any processing engine 302 can read to or write from any location in global register file 306. In addition to global register file 306, some embodiments also provide an on-chip shared memory 308, which may be implemented, e.g., as a conventional RAM. On-chip memory 308 is advantageously used to store data that is expected to be used in multiple threads, such as coefficients of attribute equations, which are usable in pixel shader programs. In some embodiments, processing engines 302 may also have access to additional off-chip shared memory (not</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>shown), which might be located, e.g., within graphics memory 124 and/or system memory 104 of FIG. 1. 11:3–18.</p> <p>For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302 with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group. 13:12–45.</p> <p>Referring again to FIG. 4, load module 408 receives input parameters for a CTA and loads the input parameters into (shared) global register file 306 (FIG. 3) of core 126. These parameters may include, e.g., an (x,y) position of the CTA within a grid of CTAs or other CTA identifier and/or other information specific to the CTA. In some embodiments, the input parameters may also include some or all of an input data set to be processed by the CTA. 16:63–17:3.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>Core interface 128 remains in the state phase and continues to update the state registers as new state information is received, until such time as core interface 128 receives a “begin CTA” command (step 1010) indicating that input data is to follow. At that point, core interface 128 enters the load phase. During the load phase, core interface 128 receives input data (step 1012) and loads the input data into (shared) global register file 306 of core 126 (step 1014). At step 1016, core interface 128 determines whether the last input data has been received. If not, core interface 128 returns to step 1012 to receive more input data. 19:40–50.</p> <p>The CTA program may also include instructions to read from and/or write to the shared global register file 306, on-chip shared memory 308, and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1. For instance, the input data set to be processed by the CTA may be stored in graphics memory 124 or system memory 104. Intermediate results may be written to global register file 306 and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1, where they can be shared with other threads. Final results (output data) may be written to graphics memory 124 or system memory 104. 20:35–45.</p> <p>In some instances, the application program executing on CPU 102 might need to wait for GPU 122 to finish processing one or more CTAs, e.g., if the data generated by the CTA(s) is needed for a subsequent processing step, and this can introduce some latency. (Such latency will generally be less than the latency associated with sequential processing techniques.) In some instances it may be possible to hide some or all of this latency through suitable sequencing or scheduling of program instructions, e.g., by arranging the program sequence so that the CTA is processed by GPU 122 while CPU 102 performs other functions that do not rely on the CTA data. It will be recognized that the extent to which latency can be hidden through software techniques will depend on the particular application. 26:48–62.</p> <p>CPU 102 operates as the master processor of system 100, controlling and coordinating operations of other system components. In particular, CPU 102 issues commands that control the operation of GPU 122. In some embodiments, CPU 102 writes a stream of commands for GPU 122 to a command buffer,</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>which may be in system memory 104, graphics memory 124, or another storage location accessible to both CPU 102 and GPU 122. GPU 122 reads the command stream from the command buffer and executes commands asynchronously with operation of CPU 102. 6:62–7:4.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>1[d][iv] “to transfer at least a portion of the output data from the accelerator memory to the main memory during performance of the intermediate computations in the sequence of computations by the at least one graphics processing unit.”</p>	<p>Nickolls discloses a controller “to transfer at least a portion of the output data from the accelerator memory to the main memory during performance of the intermediate computations in the sequence of computations by the at least one graphics processing unit.” <i>See e.g.:</i></p> <p><i>As a non-limiting example, Nickolls discloses transferring at least a portion of output data from accelerator memory (e.g., local register file 304, shared global register file 306, and/or memory 308) to main memory (e.g., system memory 104) during performance of intermediate computations in a sequence of computations by at least one graphics processing unit (e.g., GPU 122 and/or processing engines):</i></p> <p>Computer System Overview FIG. 1 is a block diagram of a computer system 100 according to an embodiment of the present invention. Computer system 100 includes a central processing unit (CPU) 102 and a system memory 104 communicating via a bus path that includes a memory bridge 105. Memory bridge 105 is connected via a bus path 106 to an I/O (input/output) bridge 107. I/O bridge 107 receives user input from one or more user input devices 108 (e.g., keyboard, mouse) and forwards the input to CPU 102 via bus 106 and memory bridge 105. Visual output is provided on a pixel based display device 110 (e.g., a conventional CRT or LCD based monitor) operating under control of a graphics subsystem 112 coupled to memory bridge 105 via a bus 113. A system disk 114 is also connected to I/O bridge 107. A switch</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>116 provides connections between I/O bridge 107 and other components such as a network adapter 118 and various add-in cards 120, 121. Other components (not explicitly shown), including USB or other port connections, CD drives, DVD drives, and the like, may also be connected to I/O bridge 107. Bus connections among the various components may be implemented using bus protocols such as PCI (Peripheral Component Interconnect), PCI Express (PCI-E), AGP (Accelerated Graphics Port), HyperTransport, or any other bus protocol(s), and connections between different devices may use different protocols as is known in the art. 6:8–34.</p> <p>The CTA program may also include instructions to read from and/or write to the shared global register file 306, on-chip shared memory 308, and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1. For instance, the input data set to be processed by the CTA may be stored in graphics memory 124 or system memory 104. Intermediate results may be written to global register file 306 and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1, where they can be shared with other threads. Final results (output data) may be written to graphics memory 124 or system memory 104. 20:35–45.</p> <p>In one embodiment, core 126 includes an array of P (e.g., 8, 16, or any other number) parallel processing engines 302 configured to receive and execute SIMD instructions from a single instruction unit 312. Each parallel processing engine 302 advantageously includes an identical set of functional units such as arithmetic logic units, load/store units, and the like (not explicitly shown). The functional units may be pipelined, allowing a new instruction to be issued before a previous instruction has finished, as is known in the art. Any combination of functional units may be provided. In one embodiment, the functional units support a variety of operations including integer and floating point arithmetic (e.g., addition and multiplication), comparison operations, Boolean operations (AND, OR, XOR), bit-shifting, and computation of various algebraic functions (e.g., planar interpolation, trigonometric, exponential, and logarithmic functions, etc.); and the same functional-unit hardware can be leveraged to perform different operations. A particular implementation of processing engines 302 is not critical to the present invention, and a detailed description has been omitted. 10:37–56.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302 with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group. 13:12–45.</p> <p>On completion of the CTA program, the final results (output data) produced by the threads are advantageously placed in memory for use by a subsequent CTA program or made accessible to CPU 102 (FIG. 1). For example, the final instructions in a CTA program might include an instruction to write some or all of the data generated by the thread to graphics memory 124. After execution of the CTA is finished, GPU 122 may transfer the data (e.g., using a conventional DMA operation) to system memory 104, making it available to application programs executing on CPU 102. Other data transfer mechanisms may also be used. Core 126 advantageously signals core interface 128 upon completion of a CTA, so that core interface 128 can initiate execution of a next CTA, reusing the resources that became free when the first CTA was completed. 20:61–21:8.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>2 “The computer system of claim 1, wherein the central processing unit is configured to receive the input data in response to a user interaction.”</p>	<p>Nickolls discloses “[t]he computer system of claim 1, wherein the central processing unit is configured to receive the input data in response to a user interaction.” <i>See e.g.</i>:</p> <p><i>See claim 1.</i></p> <p><i>As a non-limiting example, Nickolls discloses the central processing unit (e.g., CPU 102) is configured to receive input data in response to a user interaction (e.g., the application program interface (API)):</i></p> <p>I/O bridge 107 receives user input from one or more user input devices 108 (e.g., keyboard, mouse) and forwards the input to CPU 102 via bus 106 and memory bridge 105. 6:15–17.</p> <p>In some embodiments, CTAs executed by GPU 122 (FIG. 1) are used to perform computations under the direction of an application program executing on CPU 102. An application program interface (API) for defining and executing CTAs is advantageously provided to allow application programmers to access CTA functionality as desired. 25:65–26:4.</p> <p>As is known in the art, communication between CPU 102 and GPU 122 can be managed by a driver program that executes on CPU 102. The driver program supports an application program interface (API) that defines function calls supported by GPU 122, and an application programmer can invoke the GPU functions by including suitable function calls from the API at appropriate places in the program code.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>26:5–11.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>3[a] “The computer system of claim 1, wherein the central processing unit is configured to receive the input data at a first rate; and”</p>	<p>Nickolls discloses “[t]he computer system of claim 1, wherein the central processing unit is configured to receive the input data at a first rate.” <i>See e.g.</i>:</p> <p>Computer System Overview</p> <p>FIG. 1 is a block diagram of a computer system 100 according to an embodiment of the present invention. Computer system 100 includes a central processing unit (CPU) 102 and a system memory 104 communicating via a bus path that includes a memory bridge 105. Memory bridge 105 is connected via a bus path 106 to an I/O (input/output) bridge 107. I/O bridge 107 receives user input from one or more user input devices 108 (e.g., keyboard, mouse) and forwards the input to CPU 102 via bus 106 and memory bridge 105. Visual output is provided on a pixel based display device 110 (e.g., a conventional CRT or LCD based monitor) operating under control of a graphics subsystem 112 coupled to memory bridge 105 via a bus 113. A system disk 114 is also connected to I/O bridge 107. A switch 116 provides connections between I/O bridge 107 and other components such as a network adapter 118 and various add-in cards 120, 121. Other components (not explicitly shown), including USB or other port connections, CD drives, DVD drives, and the like, may also be connected to I/O bridge 107. Bus connections among the various components may be implemented using bus protocols such as PCI (Peripheral Component Interconnect), PCI Express (PCI-E), AGP (Accelerated Graphics Port), HyperTransport, or any other bus protocol(s), and connections between different devices may use different protocols as is known in the art.</p> <p>6:8–34.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>Graphics processing subsystem 112 includes a graphics processing unit (GPU) 122 and a graphics memory 124, which may be implemented, e.g., using one or more integrated circuit devices such as programmable processors, application specific integrated circuits (ASICs), and memory devices. GPU 122 may be configured to perform various tasks related to generating pixel data from graphics data supplied by CPU 102 and/or system memory 104 via memory bridge 105 and bus 113, interacting with graphics memory 124 to store and update pixel data, and the like. For example, GPU 122 may generate pixel data from 2-D or 3-D scene data provided by various programs executing on CPU 102. 6:35–46.</p> <p>CPU 102 operates as the master processor of system 100, controlling and coordinating operations of other system components. In particular, CPU 102 issues commands that control the operation of GPU 122. In some embodiments, CPU 102 writes a stream of commands for GPU 122 to a command buffer, which may be in system memory 104, graphics memory 124, or another storage location accessible to both CPU 102 and GPU 122. GPU 122 reads the command stream from the command buffer and executes commands asynchronously with operation of CPU 102. 6:62–7:4.</p> <p>It will be appreciated that the system shown herein is illustrative and that variations and modifications are possible. The bus topology, including the number and arrangement of bridges, may be modified as desired. For instance, in some embodiments, system memory 104 is connected to CPU 102 directly rather than through a bridge, and other devices communicate with system memory 104 via memory bridge 105 and CPU 102. In other alternative topologies, graphics subsystem 112 is connected to I/O bridge 107 rather than to memory bridge 105. In still other embodiments, I/O bridge 107 and memory bridge 105 might be integrated into a single chip. The particular components shown herein are optional; for instance, any number of add-in cards or peripheral devices might be supported. In some embodiments, switch 116 is eliminated, and network adapter 118 and add-in cards 120, 121 connect directly to I/O bridge 107. 7:5–20.</p> <p>It will be appreciated that the core interface described herein is illustrative and that variations and modifications are possible. Components such as incrementers, step size calculators, and thread ID generators shown herein may be modified as desired. In some embodiments, computations described as</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>being performed by the core interface can be performed elsewhere. For instance, a component of GPU 122 (FIG. 1) that supplies signals to the core interface might compute the initial thread IDs and/or the step size parameters and supply these values to the core interface as additional state parameters. In another alternative embodiment, a driver program executing on CPU 102 receives a CTA size parameter (e.g., dimensions D2, D1, and D0) from an application program and uses CPU resources to compute the initial P thread IDs and/or the step size, then supplies these parameters to core interface 128 of GPU 122 as state information, and incrementer 506 and step calculator 508 may both be omitted. 18:55–19:4.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>3[b] “the at least one graphics processing unit is configured to perform the sequence of computations at a second rate different than the first rate.”</p>	<p>Nickolls discloses that “the at least one graphics processing unit is configured to perform the sequence of computations at a second rate different than the first rate.” <i>See e.g.:</i></p> <p>Instruction unit 312 is configured such that, for any given processing cycle, the same instruction (INSTR) is issued to all P processing engines 302. Thus, at the level of a single clock cycle, core 126 implements P-way SIMD execution. Each processing engine 302 is also multithreaded, supporting up to G concurrent threads, and instructions for different ones of the G threads can be issued in any order relative to each other. Accordingly, core 126 in this embodiment can have up to P*G threads in flight concurrently. For instance, if P=16 and G=24, then core 126 can support up to 384 concurrent threads. In some embodiments, P*G determines an upper limit on the number of threads that can be included in a CTA; it is to be understood that some CTAs may include fewer than this number of threads. 11:27–40.</p> <p>Instruction unit 312 advantageously manages instruction fetch and issue for each SIMD group so as to ensure that threads in a group that have diverged eventually resynchronize. In one embodiment,</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>instruction unit 312 includes program counter (PC) logic 314, a program counter register array 316, a multiplexer 318, arbitration logic 320, fetch logic 322, and issue logic 324. Program counter register array 316 stores G program counter values (one per SIMD group), which are updated independently of each other by PC logic 314. PC logic 314 updates the PC values based on information received from processing engines 302 and/or fetch logic 322. PC logic 314 is advantageously configured to track divergence among threads in a SIMD group and to select instructions in a way that ultimately results in the threads re-synchronizing. 12:19–33.</p> <p>Arbitration logic 320 and multiplexer 318 determine the order in which instructions are fetched. More specifically, on each clock cycle, arbitration logic 320 selects one of the G possible group indices GID as the SIMD group for which a next instruction should be fetched and supplies a corresponding control signal to multiplexer 318, which selects the corresponding PC. Arbitration logic 320 may include conventional logic for prioritizing and selecting among concurrent threads (e.g., using round-robin, least-recently serviced, or the like), and selection may be based in part on feedback information from fetch logic 322 or issue logic 324 as to how many instructions have been fetched but not yet issued for each SIMD group. 12:43–55.</p> <p>For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302 with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group. 13:12–45.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>4 “The computer system of claim 1, wherein the main memory is configured to store a copy of the output data stored in the accelerator memory.”</p>	<p>Nickolls discloses “[t]he computer system of claim 1, wherein the main memory is configured to store a copy of the output data stored in the accelerator memory.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][iv].</p> <p><i>As a non-limiting example, Nickolls discloses main memory (e.g., system memory 104) configured to store a copy of output data stored in accelerator memory (e.g., local register file 304, shared global register file 306, and/or memory 308):</i></p> <p>The CTA program may also include instructions to read from and/or write to the shared global register file 306, on-chip shared memory 308, and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1. For instance, the input data set to be processed by the CTA may be stored in graphics memory 124 or system memory 104. Intermediate results may be written to global register file 306 and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1, where they</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>can be shared with other threads. Final results (output data) may be written to graphics memory 124 or system memory 104. 20:35–45.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>5 “The computer system of claim 1, wherein an output of at least one computation in the sequence of computations represents an output of at least one neuron in an artificial neural network.”</p>	<p>Nickolls discloses “[t]he computer system of claim 1, wherein an output of at least one computation in the sequence of computations represents an output of at least one neuron in an artificial neural network.”</p> <p><i>See</i> 1[c][i].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>6[a] “The computer system of claim 1, wherein accelerator memory comprises: a first memory bank to store parameters common to all of the</p>	<p>Nickolls discloses “[t]he computer system of claim 1, wherein accelerator memory comprises: a first memory bank to store parameters common to all of the computations in the sequence of computations.”</p> <p><i>See e.g.:</i></p> <p><i>See</i> claim 1.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
<p>computations in the sequence of computations; and”</p>	<p><i>As a non-limiting example, Nickolls discloses a first “memory bank” (e.g., a logical portion of global register file 306 or local register file 304) that stores parameters (e.g., input parameters) common to all computations in a sequence of computations:</i></p> <p>Numerous existing processor architectures support parallel processing. The earliest such architectures used multiple discrete processors networked together. More recently, multiple processing cores have been fabricated on a single chip. These cores are controlled in various ways. In some instances, known as multiple-instruction, multiple data (MIMD) machines, each core independently fetches and issues its own instructions to its own processing engine (or engines). In other instances, known as single-instruction, multiple-data (SIMD) machines, a core has a single instruction unit that issues the same instruction in parallel to multiple processing engines, which execute the instruction on different input operands. SIMD machines generally have advantages in chip area (since only one instruction unit is needed) and therefore cost; the downside is that parallelism is only available to the extent that multiple instances of the same instruction can be executed concurrently.</p> <p>1:36–52.</p> <p>Each processing engine 302 also has access, via a crossbar switch 305, to a (shared) global register file 306 that is shared among all of the processing engines 302 in core 126. Global register file 306 may be as large as desired, and in some embodiments, any processing engine 302 can read to or write from any location in global register file 306. In addition to global register file 306, some embodiments also provide an on-chip shared memory 308, which may be implemented, e.g., as a conventional RAM. On-chip memory 308 is advantageously used to store data that is expected to be used in multiple threads, such as coefficients of attribute equations, which are usable in pixel shader programs. In some embodiments, processing engines 302 may also have access to additional off-chip shared memory (not shown), which might be located, e.g., within graphics memory 124 and/or system memory 104 of FIG. 1.</p> <p>11:3–18.</p> <p>For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group. 13:12–45.</p> <p>Referring again to FIG. 4, load module 408 receives input parameters for a CTA and loads the input parameters into (shared) global register file 306 (FIG. 3) of core 126. These parameters may include, e.g., an (x,y) position of the CTA within a grid of CTAs or other CTA identifier and/or other information specific to the CTA. In some embodiments, the input parameters may also include some or all of an input data set to be processed by the CTA. 16:63–17:3.</p> <p>In some embodiments, the target value used to determine when synchronization is achieved may be specified as being equal to the total number of executing threads of the CTA, which can be dynamically determined by barrier synchronization logic 1100 or 1130. Although the total number of threads in a CTA can be an input parameter, as described above, in some instances, not all threads are necessarily executing at a given time; accordingly, a dynamic determination of the total is advantageous. Specifying “all executing threads” as the target value can be done, e.g., by using a predefined special value (e.g., zero) for the argument that specifies the target value or by providing a separate barrier</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>instruction that signifies that the target value is “all executing threads.” (Such an instruction would not include a target value as an argument.) Where dynamic determination of the target number is used, barrier synchronization logic 1100 or 1130 advantageously recomputes the target number from time to time so that the target remains current. 24:39–57.</p> <p>CTAs can be executed in a variety of processing architectures, including sequential single processors, parallel processors, multithreaded processors, and any architecture that can support multiple concurrent threads and that provides at least some shared memory, interconnection network, or other technology that allows threads to communicate with each other. In some embodiments, CTAs are executed using a processing core of a graphics processor that has multiple parallel processing engines, each capable of supporting multiple concurrent threads. The threads are advantageously executed in SIMD (single instruction, multiple data) groups, with one thread of the group being associated with each processing engine. A single instruction unit issues an instruction to an entire SIMD group in parallel, and each processing engine executes the instruction in the context of its thread of the current SIMD group; instructions for different SIMD groups can be issued in any order. By executing each instruction in the appropriate context, each processing engine executes one thread in each of multiple concurrent SIMD groups. 2:36–54.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
6[b] “a second memory bank to store data specific to at least one computation	Nickolls discloses “a second memory bank to store data specific to at least one computation in the sequence of computations.” <i>See e.g.:</i>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
<p>in the sequence of computations.”</p>	<p><i>As a non-limiting example, Nickolls discloses a second memory bank (e.g., a logical portion of global register file 306 or local register file 304) that stores data (e.g., input data to be processed) specific to at least one computation in a sequence of computations:</i></p> <p>Each processing engine 302 also has access, via a crossbar switch 305, to a (shared) global register file 306 that is shared among all of the processing engines 302 in core 126. Global register file 306 may be as large as desired, and in some embodiments, any processing engine 302 can read to or write from any location in global register file 306. In addition to global register file 306, some embodiments also provide an on-chip shared memory 308, which may be implemented, e.g., as a conventional RAM. On-chip memory 308 is advantageously used to store data that is expected to be used in multiple threads, such as coefficients of attribute equations, which are usable in pixel shader programs. In some embodiments, processing engines 302 may also have access to additional off-chip shared memory (not shown), which might be located, e.g., within graphics memory 124 and/or system memory 104 of FIG. 1.</p> <p>11:3–18.</p> <p>For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302 with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group. 13:12–45.</p> <p>Each processing engine 302 is allocated space in a local register file 304 for storing its local input data, intermediate results, and the like. In one embodiment, local register file 304 is divided into P lanes, each having some number of entries (where each entry might be, e.g., a 32-bit word). One lane is allocated to each processing unit, and corresponding entries in different lanes can be populated with data for corresponding thread types to facilitate SIMD execution of multiple threads in parallel as described below. The number of entries in local register file 304 is advantageously large enough to support multiple concurrent threads per processing engine 302; and in some embodiments, the number of entries allocated to a thread is dynamically configurable. 10:57–11:2.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>7 “The computer system of claim 1, wherein the controller is configured to transfer the output data from the accelerator memory to the main memory without transferring any of the</p>	<p>Nickolls discloses “[t]he computer system of claim 1, wherein the controller is configured to transfer the output data from the accelerator memory to the main memory without transferring any of the intermediate results from the accelerator memory to the main memory so as to reduce data transfer via the bus.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][iv].</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
<p>intermediate results from the accelerator memory to the main memory so as to reduce data transfer via the bus.”</p>	<p><i>As a non-limiting example, Nickolls discloses the controller (e.g., core interface 128 and/or instruction unit 312) is configured to transfer the output data from accelerator memory (e.g., memory 308, local register file 304, and/or global register file 306) to main memory (e.g., system memory 104) without transferring intermediate results:</i></p> <p>In one embodiment, core 126 includes an array of P (e.g., 8, 16, or any other number) parallel processing engines 302 configured to receive and execute SIMD instructions from a single instruction unit 312. Each parallel processing engine 302 advantageously includes an identical set of functional units such as arithmetic logic units, load/store units, and the like (not explicitly shown). The functional units may be pipelined, allowing a new instruction to be issued before a previous instruction has finished, as is known in the art. Any combination of functional units may be provided. In one embodiment, the functional units support a variety of operations including integer and floating point arithmetic (e.g., addition and multiplication), comparison operations, Boolean operations (AND, OR, XOR), bit-shifting, and computation of various algebraic functions (e.g., planar interpolation, trigonometric, exponential, and logarithmic functions, etc.); and the same functional-unit hardware can be leveraged to perform different operations. A particular implementation of processing engines 302 is not critical to the present invention, and a detailed description has been omitted. 10:37–56.</p> <p>Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group. 13:41–45.</p> <p>The CTA program may also include instructions to read from and/or write to the shared global register file 306, on-chip shared memory 308, and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1. For instance, the input data set to be processed by the CTA may be stored in graphics memory 124 or system memory 104. Intermediate results may be written to global register file 306 and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1, where they can be shared with other threads. Final results (output data) may be written to graphics memory 124 or system memory 104. 20:35–45.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>8 “The computer system of claim 1, wherein the controller is configured to transfer at least a portion of the output data from the accelerator memory to the main memory after the at least one graphics processing unit has begun to perform another sequence of computations.”</p>	<p>Nickolls discloses “[t]he computer system of claim 1, wherein the controller is configured to transfer at least a portion of the output data from the accelerator memory to the main memory after the at least one graphics processing unit has begun to perform another sequence of computations.” <i>See e.g.:</i></p> <p><i>See 1[d][iv].</i></p> <p><i>As a non-limiting example, Nickolls discloses the controller (e.g., core interface 128 and/or instruction unit 312) is configured to transfer output data from accelerator memory (e.g., memory 308, local register file 304, and/or global register file 306) to main memory (e.g., system memory 104) after the graphics processing unit (e.g., GPU 122) begins another sequence of computations:</i></p> <p>In one embodiment, core 126 includes an array of P (e.g., 8, 16, or any other number) parallel processing engines 302 configured to receive and execute SIMD instructions from a single instruction unit 312. Each parallel processing engine 302 advantageously includes an identical set of functional units such as arithmetic logic units, load/store units, and the like (not explicitly shown). The functional units may be pipelined, allowing a new instruction to be issued before a previous instruction has finished, as is known in the art. Any combination of functional units may be provided. In one embodiment, the functional units support a variety of operations including integer and floating point arithmetic (e.g., addition and multiplication), comparison operations, Boolean operations (AND, OR, XOR), bit-shifting, and computation of various algebraic functions (e.g., planar interpolation, trigonometric, exponential, and logarithmic functions, etc.); and the same functional-unit hardware can be leveraged to perform different operations. A particular implementation of processing engines 302 is not critical to the present invention, and a detailed description has been omitted.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>10:37–56.</p> <p>Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group.</p> <p>13:41–45.</p> <p>The CTA program may also include instructions to read from and/or write to the shared global register file 306, on-chip shared memory 308, and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1. For instance, the input data set to be processed by the CTA may be stored in graphics memory 124 or system memory 104. Intermediate results may be written to global register file 306 and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1, where they can be shared with other threads. Final results (output data) may be written to graphics memory 124 or system memory 104.</p> <p>20:35–45.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>9 “The computer system of claim 8, wherein the controller is configured to initiate transfer of the at least a portion of the input data and to transfer the at least a portion of the output data in parallel with</p>	<p>Nickolls discloses “[t]he computer system of claim 8, wherein the controller is configured to initiate transfer of the at least a portion of the input data and to transfer the at least a portion of the output data in parallel with performance of at least one computation in the other sequence of computations by the at least one graphics processing unit.” <i>See e.g.</i>:</p> <p><i>See</i> 1[d][iii], 1[d][iv]; claim 8.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
<p>performance of at least one computation in the other sequence of computations by the at least one graphics processing unit.”</p>	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>10 “The computer system of claim 1, wherein the controller is configured to control execution of the sequence of computations by the at least one graphics processing unit.”</p>	<p>Nickolls discloses “[t]he computer system of claim 1, wherein the controller is configured to control execution of the sequence of computations by the at least one graphics processing unit.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][ii].</p> <p><i>As a non-limiting example, Nickolls discloses the controller (e.g., core interface 128 and/or instruction unit 312) controls execution of computations by the graphics processing unit (e.g., GPU 122):</i></p> <p>GPU 122 has at least one processing core 126 for generating pixel data and a core interface 128 that controls operation of core 126. Core 126 advantageously includes multiple parallel processing engines that can be used to execute various shader programs, including vertex shader programs, geometry shader programs, and/or pixel shader programs, in the course of generating images from scene data. Core 126 can also be leveraged to perform general-purpose computations as described below. 6:47–55.</p> <p>Instruction unit 312 is configured such that, for any given processing cycle, the same instruction (INSTR) is issued to all P processing engines 302. Thus, at the level of a single clock cycle, core 126 implements P-way SIMD execution. Each processing engine 302 is also multithreaded, supporting up to G concurrent threads, and instructions for different ones of the G threads can be issued in any order relative to each other. Accordingly, core 126 in this embodiment can have up to P*G threads in flight concurrently. For instance, if P=16 and G=24, then core 126 can support up to 384 concurrent threads. In some embodiments, P*G determines an upper limit on the number of threads that can be included in a CTA; it is to be understood that some CTAs may include fewer than this number of threads. 11:27–40.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>Instruction unit 312 advantageously manages instruction fetch and issue for each SIMD group so as to ensure that threads in a group that have diverged eventually resynchronize. In one embodiment, instruction unit 312 includes program counter (PC) logic 314, a program counter register array 316, a multiplexer 318, arbitration logic 320, fetch logic 322, and issue logic 324. Program counter register array 316 stores G program counter values (one per SIMD group), which are updated independently of each other by PC logic 314. PC logic 314 updates the PC values based on information received from processing engines 302 and/or fetch logic 322. PC logic 314 is advantageously configured to track divergence among threads in a SIMD group and to select instructions in a way that ultimately results in the threads re-synchronizing. 12:19–33.</p> <p>Arbitration logic 320 and multiplexer 318 determine the order in which instructions are fetched. More specifically, on each clock cycle, arbitration logic 320 selects one of the G possible group indices GID as the SIMD group for which a next instruction should be fetched and supplies a corresponding control signal to multiplexer 318, which selects the corresponding PC. Arbitration logic 320 may include conventional logic for prioritizing and selecting among concurrent threads (e.g., using round-robin, least-recently serviced, or the like), and selection may be based in part on feedback information from fetch logic 322 or issue logic 324 as to how many instructions have been fetched but not yet issued for each SIMD group. 12:43–57.</p> <p>For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302 with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group. It will be appreciated that the processing core described herein is illustrative and that variations and modifications are possible. Any number of processing units may be included. In some embodiments, each processing unit has its own local register file, and the allocation of local register file entries per thread can be fixed or configurable as desired.</p> <p>In some embodiments, core 126 is operated at a higher clock rate than core interface 128, allowing the core to process more data in a given amount of time. For instance, core 126 can be operated at a clock rate that is twice the clock rate of core interface 128. If core 126 includes P processing engines 302 producing data at twice the core interface clock rate, then core 126 can produce 2*P data values per core interface clock cycle. Provided there is sufficient space in local register file 304, from the perspective of core interface 128, the situation is effectively identical to a core with 2*P processing units. Thus, P-way SIMD parallelism could be produced either by including P processing units in core 126 and operating core 126 at the same clock rate as core interface 128 or by including P/2 processing units in core 126 and operating core 126 at twice the clock rate of core interface 128. Other timing variations are also possible.</p> <p>13:12–67.</p> <p>FIG. 4 is a block diagram of core interface 128 according to an embodiment of the present invention. Core interface 128 includes an input unit 402, a state module 404, state registers 406, a load module 408, and a launch module 410. As described above, core interface 128 (FIG. 1) controls operation of core 126. In particular, core interface 128 loads and launches threads of a CTA in SIMD groups until all threads have been launched. In one embodiment, core interface 128 generates thread IDs for each</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>thread in a SIMD group and writes the thread ID into a suitable location in local register file 304, then launches the SIMD group. These aspects of core interface 128 will now be described. 14:17–28.</p> <p>Once all of the input data has been received, core interface 128 enters the launch phase. In the launch phase, core interface 128 selects a group index GID for a first SIMD group to be launched (step 1018); any group index GID that is not already in use in core 126 may be selected. At step 1020, core interface 128 loads the initial set of P thread IDs into the local register file 304 of core 126 in locations corresponding to the selected group index GID or into per-thread registers in core 126 dedicated to storing thread IDs. Core interface 126 then instructs core 126 to launch the P threads as a SIMD group (step 1022). 19:51–61.</p> <p>In one embodiment, execution of barrier instructions, i.e., arrival of threads (or SIMD groups) at barrier points, is detected by issue logic 324 of instruction unit 312 of FIG. 3, which can suspend the issue of instructions to any threads that are waiting at a barrier point while continuing to issue instructions to threads that are not at a barrier point. Eventually, all relevant threads reach the barrier point, and execution of the waiting threads resumes. FIG. 11A is a block diagram of barrier synchronization logic 1100 according to an embodiment of the present invention. In some embodiments, barrier synchronization logic 1100 is implemented in issue logic 324 of instruction unit 312 and synchronizes SIMD groups rather than individual threads. As shown in FIG. 11A, instruction unit 312 also includes selection logic 1110 that selects a next instruction to issue. Selection logic 1110 may be of generally conventional design, and a detailed description is omitted as not being critical to understanding the present invention. 21:37–55.</p> <p>As described above, wait/go registers 1138 are advantageously read by selection logic 1110, and selection logic 1110 does not select instructions for SIMD groups that are in the wait state, thereby suspending execution of such groups. Selection logic 1110 may continue to select instructions for other SIMD groups. 23:61–67.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>12[pre] “A method of performing a sequence of computations representing an artificial neural network on a computer system comprising a central processing unit (CPU), a main memory operably coupled to the central processing unit via a bus, an accelerator operably coupled to the CPU and the main memory via the bus, the accelerator comprising a graphics processing unit (GPU) and an accelerator memory, the method comprising:”</p>	<p>To the extent the preamble is limiting, Nickolls discloses “[a] method of performing a sequence of computations representing an artificial neural network on a computer system comprising a central processing unit (CPU), a main memory operably coupled to the central processing unit via a bus, an accelerator operably coupled to the CPU and the main memory via the bus, the accelerator comprising a graphics processing unit (GPU) and an accelerator memory.” <i>See e.g.:</i></p> <p><i>See</i> 1[a]-1[c][ii].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>12[a] “(A) performing, by the GPU, the sequence of computations on a first portion of input data so as to generate a first portion of output data, the first</p>	<p>Nickolls discloses “performing, by the GPU, the sequence of computations on a first portion of input data so as to generate a first portion of output data, the first portion of the output data representing an output of a neuron in a first layer of the artificial neural network, intermediate computations in the sequence of computations yielding intermediate results.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][i].</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
<p>portion of the output data representing an output of a neuron in a first layer of the artificial neural network, intermediate computations in the sequence of computations yielding intermediate results, wherein performing the sequence of computations on the first portion of the input data comprises “</p>	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>12[a][i] “(i) assigning an output variable to a first texture and a second texture, the output variable being included in a first computational element of a plurality of computational elements, the plurality of computational elements representing the sequence of computations and”</p>	<p>Nickolls discloses “assigning an output variable to a first texture and a second texture, the output variable being included in a first computational element of a plurality of computational elements, the plurality of computational elements representing the sequence of computations.” <i>See e.g.:</i></p> <p><i>See claim 1.</i></p> <p>Threads in a CTA can share intermediate results with other threads in the same CTA using a shared memory that is accessible to all of the threads, an interconnection network, or other technologies for inter-thread communication, including other technologies known in the art. In some embodiments, the CTA program includes an instruction to compute an address in shared memory to which particular data is to be written, with the address being a function of thread ID. Each thread computes the function using its own thread ID and writes to the corresponding location. The function is advantageously defined such that different threads write to different locations; as long as the function is deterministic, the location written to by any thread is well-defined. The CTA program can also include an instruction to compute an address in shared memory from which data is to be read, with the address being a function of thread ID. By defining suitable functions, data can be written to a given location by one thread and read from that location by a different thread in a predictable manner. Consequently, any desired pattern of data sharing among threads can be supported, and any thread in a CTA can share data with any other thread in the same CTA.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>8:10–30.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>12[a][ii] “(ii) accumulating a first value for the output variable in the first texture during a first time step;”</p>	<p>Nickolls discloses “accumulating a first value for the output variable in the first texture during a first time step.” <i>See e.g.:</i></p> <p><i>See</i> 12[a][i].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>12[b] “(B) in parallel with performing the sequence of computations by the GPU in (A), transferring a second portion of the input data from the main memory to the accelerator via the bus;”</p>	<p>Nickolls discloses “in parallel with performing the sequence of computations by the GPU in (A), transferring a second portion of the input data from the main memory to the accelerator via the bus.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][iii]; claim 9.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>any supplements thereto and the relevant section of charts for other prior art for the '438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>12[c] “(C) in parallel with performing the sequence of computations by the GPU in (A), transferring a second portion of the output data from the accelerator memory to the main memory via the bus, the second portion of the output data representing an output of a neuron in a second layer in the artificial neural network; and”</p>	<p>Nickolls discloses “in parallel with performing the sequence of computations by the GPU in (A), transferring a second portion of the output data from the accelerator memory to the main memory via the bus, the second portion of the output data representing an output of a neuron in a second layer in the artificial neural network.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][iv]; claim 8.</p> <p><i>As a non-limiting example, Nickolls discloses that the accelerator (e.g., at the direction and/or under the control of core interface 128 and/or instruction unit 312) transfers outputs of computations from accelerator memory (e.g., memory 308, local register file 304, and/or global register file 306) to main memory (e.g., system memory 104):</i></p> <p>The CTA program may also include instructions to read from and/or write to the shared global register file 306, on-chip shared memory 308, and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1. For instance, the input data set to be processed by the CTA may be stored in graphics memory 124 or system memory 104. Intermediate results may be written to global register file 306 and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1, where they can be shared with other threads. Final results (output data) may be written to graphics memory 124 or system memory 104. 20:35–45.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the '438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>12[d] “(D) performing, by the GPU, the sequence of computations on the second portion of the input data, wherein performing the sequence of computations on the second portion of the input data comprises “</p>	<p>Nickolls discloses “performing, by the GPU, the sequence of computations on the second portion of the input data.” <i>See e.g.:</i></p> <p><i>See</i> 12[a], 12[b].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>12[d][i] “(i) accumulating a second value for the output variable in the second texture during a second time step and “</p>	<p>Nickolls discloses “accumulating a second value for the output variable in the second texture during a second time step.” <i>See e.g.:</i></p> <p><i>See</i> 12[a][i], 12[a][ii].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>12[d][ii] “(ii) making the first value of the output variable in the first texture accessible to other</p>	<p>Nickolls discloses “making the first value of the output variable in the first texture accessible to other computational elements in the plurality of computational elements during the second time step.” <i>See e.g.:</i></p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
<p>computational elements in the plurality of computational elements during the second time step.”</p>	<p><i>See</i> 12[d][i], 1[c][ii].</p> <p><i>As a non-limiting example, Nickolls discloses making the first value of the output variable in the first texture accessible to other computational elements during the second time step (e.g., by using the output of one computational cycle, such as an intermediate result, as an input for another cycle of computation):</i></p> <p>Threads in a CTA can share intermediate results with other threads in the same CTA using a shared memory that is accessible to all of the threads, an interconnection network, or other technologies for inter-thread communication, including other technologies known in the art. In some embodiments, the CTA program includes an instruction to compute an address in shared memory to which particular data is to be written, with the address being a function of thread ID. Each thread computes the function using its own thread ID and writes to the corresponding location. The function is advantageously defined such that different threads write to different locations; as long as the function is deterministic, the location written to by any thread is well-defined. The CTA program can also include an instruction to compute an address in shared memory from which data is to be read, with the address being a function of thread ID. By defining suitable functions, data can be written to a given location by one thread and read from that location by a different thread in a predictable manner. Consequently, any desired pattern of data sharing among threads can be supported, and any thread in a CTA can share data with any other thread in the same CTA.</p> <p>8:10–30.</p> <p>Each processing engine 302 is allocated space in a local register file 304 for storing its local input data, intermediate results, and the like. In one embodiment, local register file 304 is divided into P lanes, each having some number of entries (where each entry might be, e.g., a 32-bit word). One lane is allocated to each processing unit, and corresponding entries in different lanes can be populated with data for corresponding thread types to facilitate SIMD execution of multiple threads in parallel as described below. The number of entries in local register file 304 is advantageously large enough to support multiple concurrent threads per processing engine 302; and in some embodiments, the number of entries allocated to a thread is dynamically configurable.</p> <p>10:57–11:2.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>Each processing engine 302 also has access, via a crossbar switch 305, to a (shared) global register file 306 that is shared among all of the processing engines 302 in core 126. Global register file 306 may be as large as desired, and in some embodiments, any processing engine 302 can read to or write from any location in global register file 306. In addition to global register file 306, some embodiments also provide an on-chip shared memory 308, which may be implemented, e.g., as a conventional RAM. On-chip memory 308 is advantageously used to store data that is expected to be used in multiple threads, such as coefficients of attribute equations, which are usable in pixel shader programs. In some embodiments, processing engines 302 may also have access to additional off-chip shared memory (not shown), which might be located, e.g., within graphics memory 124 and/or system memory 104 of FIG. 1.</p> <p>11:3–18.</p> <p>For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302 with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>13:12–45.</p> <p>The CTA program may also include instructions to read from and/or write to the shared global register file 306, on-chip shared memory 308, and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1. For instance, the input data set to be processed by the CTA may be stored in graphics memory 124 or system memory 104. Intermediate results may be written to global register file 306 and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1, where they can be shared with other threads. Final results (output data) may be written to graphics memory 124 or system memory 104.</p> <p>20:35–45.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>13 “The method of claim 12, further comprising: storing the input data in the main memory in response to a user interaction.”</p>	<p>Nickolls discloses “[t]he method of claim 12, further comprising: storing the input data in the main memory in response to a user interaction.” <i>See e.g.</i>:</p> <p><i>See</i> 1[b]; claim 2.</p> <p><i>As a non-limiting example, Nickolls discloses storing input data in main memory (e.g., system memory 104) in response to a user interaction received through an API:</i></p> <p>FIG. 1 is a block diagram of a computer system 100 according to an embodiment of the present invention. Computer system 100 includes a central processing unit (CPU) 102 and a system memory 104 communicating via a bus path that includes a memory bridge 105. Memory bridge 105 is connected via a bus path 106 to an I/O (input/output) bridge 107. I/O bridge 107 receives user input from one or more user input devices 108 (e.g., keyboard, mouse) and forwards the input to CPU 102 via bus 106</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>and memory bridge 105. Visual output is provided on a pixel based display device 110 (e.g., a conventional CRT or LCD based monitor) operating under control of a graphics subsystem 112 coupled to memory bridge 105 via a bus 113. A system disk 114 is also connected to I/O bridge 107. A switch 116 provides connections between I/O bridge 107 and other components such as a network adapter 118 and various add-in cards 120, 121. Other components (not explicitly shown), including USB or other port connections, CD drives, DVD drives, and the like, may also be connected to I/O bridge 107. Bus connections among the various components may be implemented using bus protocols such as PCI (Peripheral Component Interconnect), PCI Express (PCI-E), AGP (Accelerated Graphics Port), HyperTransport, or any other bus protocol(s), and connections between different devices may use different protocols as is known in the art. 6:8–34.</p> <p>In some embodiments, CTAs executed by GPU 122 (FIG. 1) are used to perform computations under the direction of an application program executing on CPU 102. An application program interface (API) for defining and executing CTAs is advantageously provided to allow application programmers to access CTA functionality as desired. As is known in the art, communication between CPU 102 and GPU 122 can be managed by a driver program that executes on CPU 102. The driver program supports an application program interface (API) that defines function calls supported by GPU 122, and an application programmer can invoke the GPU functions by including suitable function calls from the API at appropriate places in the program code. 25:66–26:11.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
14[a] “The method of claim 12, further comprising: receiving the input data at a first rate; and”	<i>See</i> 3[a]; claim 12.
14[b] “wherein (A) comprises performing the sequence of computations at a second rate different than the first rate.”	<i>See</i> 3[b].
16 “The method of claim 12, wherein (C) comprises: transferring the second portion of the output data from the accelerator memory to the main memory without transferring any of the intermediate results of the plurality of sequential computations from the accelerator memory to the main memory so as to reduce data transfer via the bus.”	<p>Nickolls discloses “[t]he method of claim 12, wherein (C) comprises: transferring the second portion of the output data from the accelerator memory to the main memory without transferring any of the intermediate results of the plurality of sequential computations from the accelerator memory to the main memory so as to reduce data transfer via the bus.” <i>See e.g.:</i></p> <p><i>See</i> claim 7.</p> <p><i>As a non-limiting example, Nickolls discloses transferring a “second portion” of output data from accelerator memory (e.g., memory 308, local register file 304, and/or global register file 306) to main memory (e.g., system memory 104) without transferring intermediate results:</i></p> <p>In one embodiment, core 126 includes an array of P (e.g., 8, 16, or any other number) parallel processing engines 302 configured to receive and execute SIMD instructions from a single instruction unit 312. Each parallel processing engine 302 advantageously includes an identical set of functional units such as arithmetic logic units, load/store units, and the like (not explicitly shown). The functional units may be pipelined, allowing a new instruction to be issued before a previous instruction has finished, as is known in the art. Any combination of functional units may be provided. In one embodiment, the functional units support a variety of operations including integer and floating point arithmetic (e.g., addition and multiplication), comparison operations, Boolean operations (AND, OR, XOR), bit-shifting, and computation of various algebraic functions (e.g., planar interpolation, trigonometric, exponential, and logarithmic functions, etc.); and the same functional-unit hardware can</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>be leveraged to perform different operations. A particular implementation of processing engines 302 is not critical to the present invention, and a detailed description has been omitted. 10:37–56.</p> <p>For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302 with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group. 13:12–45.</p> <p>The CTA program may also include instructions to read from and/or write to the shared global register file 306, on-chip shared memory 308, and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1. For instance, the input data set to be processed by the CTA may be stored in graphics memory 124 or system memory 104. Intermediate results may be written to global register file 306 and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1, where they</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>can be shared with other threads. Final results (output data) may be written to graphics memory 124 or system memory 104. 20:35–45.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>17 “The method of claim 12, wherein (C) comprises: transferring the second portion of the output data from the accelerator memory to the main memory after the GPU has begun to perform another sequence of computations.”</p>	<p>Nickolls discloses “[t]he method of claim 12, wherein (C) comprises: transferring the second portion of the output data from the accelerator memory to the main memory after the GPU has begun to perform another sequence of computations.” <i>See e.g.</i>:</p> <p><i>See claim 8.</i></p> <p><i>As a non-limiting example, Nickolls discloses transferring a “second portion” of output data from accelerator memory (e.g., memory 308, local register file 304, and/or global register file 306) to main memory (e.g., system memory 104) after the graphics processing unit (e.g., GPU 122) begins another sequence of computations:</i></p> <p>In one embodiment, core 126 includes an array of P (e.g., 8, 16, or any other number) parallel processing engines 302 configured to receive and execute SIMD instructions from a single instruction unit 312. Each parallel processing engine 302 advantageously includes an identical set of functional units such as arithmetic logic units, load/store units, and the like (not explicitly shown). The functional units may be pipelined, allowing a new instruction to be issued before a previous instruction has finished, as is known in the art. Any combination of functional units may be provided. In one embodiment, the functional units support a variety of operations including integer and floating point arithmetic (e.g., addition and multiplication), comparison operations, Boolean operations (AND, OR, XOR), bit-shifting, and computation of various algebraic functions (e.g., planar interpolation,</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>trigonometric, exponential, and logarithmic functions, etc.); and the same functional-unit hardware can be leveraged to perform different operations. A particular implementation of processing engines 302 is not critical to the present invention, and a detailed description has been omitted. 10:37–56.</p> <p>For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302 with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group. 13:12–45.</p> <p>The CTA program may also include instructions to read from and/or write to the shared global register file 306, on-chip shared memory 308, and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1. For instance, the input data set to be processed by the CTA may be stored in graphics memory 124 or system memory 104. Intermediate results may be written to global register file 306 and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1, where they</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>can be shared with other threads. Final results (output data) may be written to graphics memory 124 or system memory 104. 20:35–45.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>18 “The method of claim 17, wherein (C) further comprises: initiating transfer of the second portion of the output data in parallel with performance of at least one computation in the other sequence of computations.”</p>	<p>Nickolls discloses “[t]he method of claim 17, wherein (C) further comprises: initiating transfer of the second portion of the output data in parallel with performance of at least one computation in the other sequence of computations.” <i>See e.g.:</i></p> <p><i>See</i> claim 9.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>20[a] “The method of claim 12, further comprising: storing parameters common to all of the computations in the sequence of computations</p>	<p>Nickolls discloses “[t]he method of claim 12, further comprising: storing parameters common to all of the computations in the sequence of computations in a first memory bank in the accelerator memory.” <i>See e.g.:</i></p> <p><i>See</i> 6[a]; claim 12.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
<p>in a first memory bank in the accelerator memory; and”</p>	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>20[b] “storing data specific to at least one computation in the sequence of computations in a second memory bank in the accelerator memory.”</p>	<p>Nickolls discloses “storing data specific to at least one computation in the sequence of computations in a second memory bank in the accelerator memory.” <i>See e.g.:</i></p> <p><i>See</i> 6[b].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>21[pre] “A method of performing a sequence of computations representing an artificial neural network, the method comprising:”</p>	<p>Nickolls discloses “[a] method of performing a sequence of computations representing an artificial neural network.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][i], 12[pre].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>21[a] “receiving, at a central processing unit (CPU), first input data acquired from an external system in real time;”</p>	<p>Nickolls discloses “receiving, at a central processing unit (CPU), first input data acquired from an external system in real time.” <i>See e.g.:</i></p> <p><i>See</i> 1[a].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>21[b] “initializing, by a controller operably coupled to a graphics processing unit (GPU), textures and shaders in a memory operably coupled to the GPU;”</p>	<p>Nickolls discloses “initializing, by a controller operably coupled to a graphics processing unit (GPU), textures and shaders in a memory operably coupled to the GPU.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][ii], 1[d], 1[d][i].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>21[c] “transferring the first input data received by the CPU to the memory</p>	<p>Nickolls discloses “transferring the first input data received by the CPU to the memory operably coupled to the GPU.” <i>See e.g.:</i></p> <p><i>See</i> 1[b], 1[c], 1[d][iii].</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
operably coupled to the GPU;”	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
21[d] “performing, by the graphics processing unit (GPU), a first computation in the sequence of computations on the first input data based on the textures and shaders to generate first output data, computations in the sequence of computations representing respective layers of neurons in the artificial neural network, an output of the first computation in the sequence of computations representing an output of a first neuron in a first layer in the artificial neural network;”	<p>Nickolls discloses “performing, by the graphics processing unit (GPU), a first computation in the sequence of computations on the first input data based on the textures and shaders to generate first output data, computations in the sequence of computations representing respective layers of neurons in the artificial neural network, an output of the first computation in the sequence of computations representing an output of a first neuron in a first layer in the artificial neural network.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][i], 12[a].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
21[e] “storing, in the memory operably coupled to the GPU, the first input	<p>Nickolls discloses “storing, in the memory operably coupled to the GPU, the first input data and the first output data.” <i>See e.g.:</i></p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
<p>data and the first output data; and”</p>	<p><i>See</i> 1[c][iii], 1[d][iii], 12[a].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>21[f] “transferring second input data acquired from the external system in real time into the memory operably coupled to the GPU after the GPU starts the first computation and before the GPU starts a second computation of the sequence of computations, an output of the second computation in the sequence of computations representing an output of a second neuron in a second layer in the artificial neural network.”</p>	<p>Nickolls discloses “transferring second input data acquired from the external system in real time into the memory operably coupled to the GPU after the GPU starts the first computation and before the GPU starts a second computation of the sequence of computations, an output of the second computation in the sequence of computations representing an output of a second neuron in a second layer in the artificial neural network.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][iii], 1[d][iv], 12[b], 12[c].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>22 “The method of claim 21, wherein transferring the second input data comprises transferring the second input data via a bus</p>	<p>Nickolls discloses “[t]he method of claim 21, wherein transferring the second input data comprises transferring the second input data via a bus operably coupled to the CPU.” <i>See e.g.:</i></p> <p><i>See</i> 1[c], 12[b].</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
operably coupled to the CPU.”	To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.
23 “The method of claim 21, further comprising: transferring the first output data from the memory to another memory during the second computation in the sequence of computations.”	<p>Nickolls discloses “[t]he method of claim 21, further comprising: transferring the first output data from the memory to another memory during the second computation in the sequence of computations.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][iv].</p> <p><i>As a non-limiting example, Nickolls discloses transferring output data from one memory (e.g., local register file 304, global register file 306, or memory 308) to another memory (e.g., a different memory in the accelerator or system memory 104).</i></p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
24[a] “The method of claim 23, further comprising: storing intermediate results of the sequence of computations in the memory, and”	<p>Nickolls discloses “[t]he method of claim 23, further comprising: storing intermediate results of the sequence of computations in the memory.” <i>See e.g.:</i></p> <p><i>See</i> 12[a]; claim 16.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p data-bbox="583 235 1856 305"><i>As a non-limiting example, Nickolls discloses storing intermediate results in memory (e.g., memory 308, local register file 304, and/or global register file 306):</i></p> <p data-bbox="583 345 1913 813">Threads in a CTA can share intermediate results with other threads in the same CTA using a shared memory that is accessible to all of the threads, an interconnection network, or other technologies for inter-thread communication, including other technologies known in the art. In some embodiments, the CTA program includes an instruction to compute an address in shared memory to which particular data is to be written, with the address being a function of thread ID. Each thread computes the function using its own thread ID and writes to the corresponding location. The function is advantageously defined such that different threads write to different locations; as long as the function is deterministic, the location written to by any thread is well-defined. The CTA program can also include an instruction to compute an address in shared memory from which data is to be read, with the address being a function of thread ID. By defining suitable functions, data can be written to a given location by one thread and read from that location by a different thread in a predictable manner. Consequently, any desired pattern of data sharing among threads can be supported, and any thread in a CTA can share data with any other thread in the same CTA.</p> <p data-bbox="583 821 701 849">8:10–30.</p> <p data-bbox="583 894 1898 1182">Each processing engine 302 is allocated space in a local register file 304 for storing its local input data, intermediate results, and the like. In one embodiment, local register file 304 is divided into P lanes, each having some number of entries (where each entry might be, e.g., a 32-bit word). One lane is allocated to each processing unit, and corresponding entries in different lanes can be populated with data for corresponding thread types to facilitate SIMD execution of multiple threads in parallel as described below. The number of entries in local register file 304 is advantageously large enough to support multiple concurrent threads per processing engine 302; and in some embodiments, the number of entries allocated to a thread is dynamically configurable.</p> <p data-bbox="583 1187 743 1214">10:57–11:2.</p> <p data-bbox="583 1260 1902 1399">Each processing engine 302 also has access, via a crossbar switch 305, to a (shared) global register file 306 that is shared among all of the processing engines 302 in core 126. Global register file 306 may be as large as desired, and in some embodiments, any processing engine 302 can read to or write from any location in global register file 306. In addition to global register file 306, some embodiments also</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>provide an on-chip shared memory 308, which may be implemented, e.g., as a conventional RAM. On-chip memory 308 is advantageously used to store data that is expected to be used in multiple threads, such as coefficients of attribute equations, which are usable in pixel shader programs. In some embodiments, processing engines 302 may also have access to additional off-chip shared memory (not shown), which might be located, e.g., within graphics memory 124 and/or system memory 104 of FIG. 1. 11:3–18.</p> <p>The CTA program may also include instructions to read from and/or write to the shared global register file 306, on-chip shared memory 308, and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1. For instance, the input data set to be processed by the CTA may be stored in graphics memory 124 or system memory 104. Intermediate results may be written to global register file 306 and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1, where they can be shared with other threads. Final results (output data) may be written to graphics memory 124 or system memory 104. 20:35–45.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>24[b] “wherein transferring the first output data from the memory to the other memory occurs without transferring the intermediate results of the sequence of computations.”</p>	<p>Nickolls discloses that “transferring the first output data from the memory to the other memory occurs without transferring the intermediate results of the sequence of computations.” <i>See e.g.:</i></p> <p><i>See</i> claims 7, 16.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>25 “The method of claim 23, wherein transferring the second input data and transferring the first output data occurs in parallel.”</p>	<p>Nickolls discloses “[t]he method of claim 23, wherein transferring the second input data and transferring the first output data occurs in parallel.” <i>See e.g.:</i></p> <p><i>See</i> 12[b], 12[c]; claim 9.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>26 “The method of claim 21, further comprising: storing, in a first memory partition of the memory, parameters common to all of the computations in the sequence of computations.”</p>	<p>Nickolls discloses “[t]he method of claim 21, further comprising: storing, in a first memory partition of the memory, parameters common to all of the computations in the sequence of computations.” <i>See e.g.:</i></p> <p><i>See</i> 6[a].</p> <p><i>As a non-limiting example, Nickolls discloses a first “memory partition” of the memory (e.g., a logical portion of global register file 306 or local register file 304) that stores parameters (e.g., input parameters) common to all computations in a sequence of computations:</i></p> <p>Threads in a CTA can share intermediate results with other threads in the same CTA using a shared memory that is accessible to all of the threads, an interconnection network, or other technologies for inter-thread communication, including other technologies known in the art. In some embodiments, the CTA program includes an instruction to compute an address in shared memory to which particular data</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>is to be written, with the address being a function of thread ID. Each thread computes the function using its own thread ID and writes to the corresponding location. The function is advantageously defined such that different threads write to different locations; as long as the function is deterministic, the location written to by any thread is well-defined. The CTA program can also include an instruction to compute an address in shared memory from which data is to be read, with the address being a function of thread ID. By defining suitable functions, data can be written to a given location by one thread and read from that location by a different thread in a predictable manner. Consequently, any desired pattern of data sharing among threads can be supported, and any thread in a CTA can share data with any other thread in the same CTA. 8:10–30.</p> <p>Each processing engine 302 is allocated space in a local register file 304 for storing its local input data, intermediate results, and the like. In one embodiment, local register file 304 is divided into P lanes, each having some number of entries (where each entry might be, e.g., a 32-bit word). One lane is allocated to each processing unit, and corresponding entries in different lanes can be populated with data for corresponding thread types to facilitate SIMD execution of multiple threads in parallel as described below. The number of entries in local register file 304 is advantageously large enough to support multiple concurrent threads per processing engine 302; and in some embodiments, the number of entries allocated to a thread is dynamically configurable. 10:57–11:2.</p> <p>State module 404 receives state information from input unit 402 and loads the state information into state registers 406. “State information,” as used herein, includes any information (other than input data) relevant to defining a CTA. For example, in one embodiment, state information includes the size of the input data set, the amount of local register file space required for each thread, and a starting program counter (e.g., memory address) for a program to be executed by each thread. State information advantageously also includes size information for the CTA; for example, referring to FIG. 2A, array dimensions D0, D1 and D2 may be provided. In some embodiments, the total number (7) of threads is also provided; in other embodiments, T can be computed from the array dimensions (e.g., $T=D0*D1*D2$ for the embodiment of FIG. 2A). When T is provided, T is advantageously less than or equal to $D0*D1*D2$, in order to ensure that each thread is assigned a unique thread ID. 14:41–57.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>27 “The method of claim 26, further comprising: storing, in a second memory partition of the memory, data specific to the first computation in the sequence of computations.”</p>	<p>Nickolls discloses “[t]he method of claim 26, further comprising: storing, in a second memory partition of the memory, data specific to the first computation in the sequence of computations.” <i>See e.g.:</i></p> <p><i>See claim 6.</i></p> <p><i>As a non-limiting example, Nickolls discloses a second “memory partition” of the memory (e.g., a logical portion of global register file 306 or local register file 304) that stores data (e.g., input data to be processed) specific to at least one computation in a sequence of computations.</i></p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>28 “The method of claim 27, further comprising: storing, in the second memory partition, external input data patterns, representations of internal</p>	<p>Nickolls discloses “[t]he method of claim 27, further comprising: storing, in the second memory partition, external input data patterns, representations of internal variables, an input of the computation in the sequence of computations, and the output of the computation in the sequence of computations.” <i>See e.g.:</i></p> <p><i>See claim 27.</i></p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
<p>variables, an input of the computation in the sequence of computations, and the output of the computation in the sequence of computations.”</p>	<p><i>As a non-limiting example, Nickolls discloses storing external input data patterns (e.g., input from an external input device) and representations of internal variables:</i></p> <p>Embodiments of the present invention provide data processing systems and methods that use cooperative thread arrays (CTAs) to perform computations. As used herein, a “cooperative thread array,” or “CTA,” is a group of multiple threads that concurrently execute the same program on an input data set to produce an output data set. Each thread in a CTA has a unique identifier (thread ID) assigned at thread launch time that controls various aspects of the thread's processing behavior. For instance, a thread ID may be used to determine which portion of the input data set a thread is to process, to identify one or more other threads with which a given thread is to share an intermediate result, and/or to determine which portion of the output data set the thread is to compute or write. 2:22–35.</p> <p>According to another aspect of the present invention, a method for processing an input data set using a processor (which might be, e.g., a graphics processor) configured to execute a plurality of threads concurrently includes communicating to the processor information defining a thread array having a plurality of threads, each thread being identified by a unique thread identifier (which can be, e.g., a one-dimensional or multidimensional identifier). Also communicated to the processor is a program to be executed by each thread of the thread array on at least a portion of an input data set stored in a shared memory to produce at least a portion of an output data set; during program execution, the unique thread identifier is used by each thread to determine at least one processing behavior. The input data set is communicated to the processor, and the processor stores the input data set in a shared memory communicably coupled to the processing engines. An instruction to begin concurrently executing all of the threads of the thread array to process the input data set in the shared memory to produce the output data set is communicated to the processor. In some embodiments, each thread processes a different portion of the input data set, and in some embodiments, each thread produces a different portion of the output data set. 4:16–38.</p> <p>Each processing engine 302 is allocated space in a local register file 304 for storing its local input data, intermediate results, and the like. In one embodiment, local register file 304 is divided into P lanes,</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>each having some number of entries (where each entry might be, e.g., a 32-bit word). One lane is allocated to each processing unit, and corresponding entries in different lanes can be populated with data for corresponding thread types to facilitate SIMD execution of multiple threads in parallel as described below. The number of entries in local register file 304 is advantageously large enough to support multiple concurrent threads per processing engine 302; and in some embodiments, the number of entries allocated to a thread is dynamically configurable.</p> <p>Each processing engine 302 also has access, via a crossbar switch 305, to a (shared) global register file 306 that is shared among all of the processing engines 302 in core 126. Global register file 306 may be as large as desired, and in some embodiments, any processing engine 302 can read to or write from any location in global register file 306. In addition to global register file 306, some embodiments also provide an on-chip shared memory 308, which may be implemented, e.g., as a conventional RAM. On-chip memory 308 is advantageously used to store data that is expected to be used in multiple threads, such as coefficients of attribute equations, which are usable in pixel shader programs. In some embodiments, processing engines 302 may also have access to additional off-chip shared memory (not shown), which might be located, e.g., within graphics memory 124 and/or system memory 104 of FIG. 1.</p> <p>10:57–11:18.</p> <p>For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302 with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group. 13:12–45.</p> <p>The CTA program may also include instructions to read from and/or write to the shared global register file 306, on-chip shared memory 308, and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1. For instance, the input data set to be processed by the CTA may be stored in graphics memory 124 or system memory 104. Intermediate results may be written to global register file 306 and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1, where they can be shared with other threads. Final results (output data) may be written to graphics memory 124 or system memory 104. 20:35–45.</p> <p>A barrier arrival instruction might be used in preference to a barrier arrive-and-wait instruction, e.g., where one thread produces an intermediate result to be consumed by one or more other threads but does not consume intermediate results from those threads. The producer thread would not need to wait at the barrier point for the consumer threads, but the consumer threads would all need to wait for the producer thread to arrive at the barrier point before reading the data. Thus, the program instructions for the producer thread might include a barrier arrival instruction subsequent to an instruction to write the intermediate result to a shared memory location while the program instructions for the consumer thread might include a barrier arrive-and-wait instruction prior to an instruction to read the intermediate result from the shared memory location. In one embodiment, the barrier arrival and barrier arrive-and-wait instructions can be conditional, with each thread using its thread ID to determine which (if either) to execute, depending on whether the thread ID indicates that the thread is a producer or consumer. 25:7–25.</p> <p><i>Nickolls further discloses storing inputs and outputs of computations:</i></p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p data-bbox="583 235 856 267"><i>See 1[c][iii], 1[d][iii].</i></p> <p data-bbox="583 310 1917 776">Threads in a CTA can share intermediate results with other threads in the same CTA using a shared memory that is accessible to all of the threads, an interconnection network, or other technologies for inter-thread communication, including other technologies known in the art. In some embodiments, the CTA program includes an instruction to compute an address in shared memory to which particular data is to be written, with the address being a function of thread ID. Each thread computes the function using its own thread ID and writes to the corresponding location. The function is advantageously defined such that different threads write to different locations; as long as the function is deterministic, the location written to by any thread is well-defined. The CTA program can also include an instruction to compute an address in shared memory from which data is to be read, with the address being a function of thread ID. By defining suitable functions, data can be written to a given location by one thread and read from that location by a different thread in a predictable manner. Consequently, any desired pattern of data sharing among threads can be supported, and any thread in a CTA can share data with any other thread in the same CTA.</p> <p data-bbox="583 786 701 818">8:10–30.</p> <p data-bbox="583 859 1917 1399">For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302 with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>ID into a thread ID register for each thread or into a predetermined register in the allocated portion of local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group. 13:12–45.</p> <p>Core interface 128 remains in the state phase and continues to update the state registers as new state information is received, until such time as core interface 128 receives a “begin CTA” command (step 1010) indicating that input data is to follow. At that point, core interface 128 enters the load phase. During the load phase, core interface 128 receives input data (step 1012) and loads the input data into (shared) global register file 306 of core 126 (step 1014). At step 1016, core interface 128 determines whether the last input data has been received. If not, core interface 128 returns to step 1012 to receive more input data. 19:40–50.</p> <p><i>Nickolls discloses storing each type of data in a second “memory partition”:</i></p> <p>According to another aspect of the present invention, a method for processing an input data set using a processor (which might be, e.g., a graphics processor) configured to execute a plurality of threads concurrently includes communicating to the processor information defining a thread array having a plurality of threads, each thread being identified by a unique thread identifier (which can be, e.g., a one-dimensional or multidimensional identifier). Also communicated to the processor is a program to be executed by each thread of the thread array on at least a portion of an input data set stored in a shared memory to produce at least a portion of an output data set; during program execution, the unique thread identifier is used by each thread to determine at least one processing behavior. The input data set is communicated to the processor, and the processor stores the input data set in a shared memory communicably coupled to the processing engines. An instruction to begin concurrently executing all of the threads of the thread array to process the input data set in the shared memory to produce the output data set is communicated to the processor. In some embodiments, each thread processes a different portion of the input data set, and in some embodiments, each thread produces a different portion of the output data set.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>4:16–38.</p> <p>Each processing engine 302 is allocated space in a local register file 304 for storing its local input data, intermediate results, and the like. In one embodiment, local register file 304 is divided into P lanes, each having some number of entries (where each entry might be, e.g., a 32-bit word). One lane is allocated to each processing unit, and corresponding entries in different lanes can be populated with data for corresponding thread types to facilitate SIMD execution of multiple threads in parallel as described below. The number of entries in local register file 304 is advantageously large enough to support multiple concurrent threads per processing engine 302; and in some embodiments, the number of entries allocated to a thread is dynamically configurable.</p> <p>10:57–11:2.</p> <p>Each processing engine 302 also has access, via a crossbar switch 305, to a (shared) global register file 306 that is shared among all of the processing engines 302 in core 126. Global register file 306 may be as large as desired, and in some embodiments, any processing engine 302 can read to or write from any location in global register file 306. In addition to global register file 306, some embodiments also provide an on-chip shared memory 308, which may be implemented, e.g., as a conventional RAM. On-chip memory 308 is advantageously used to store data that is expected to be used in multiple threads, such as coefficients of attribute equations, which are usable in pixel shader programs. In some embodiments, processing engines 302 may also have access to additional off-chip shared memory (not shown), which might be located, e.g., within graphics memory 124 and/or system memory 104 of FIG. 1.</p> <p>11:3–18.</p> <p>The CTA program may also include instructions to read from and/or write to the shared global register file 306, on-chip shared memory 308, and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1. For instance, the input data set to be processed by the CTA may be stored in graphics memory 124 or system memory 104. Intermediate results may be written to global register file 306 and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1, where they can be shared with other threads. Final results (output data) may be written to graphics memory 124 or system memory 104.</p> <p>20:35–45.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>29 “The method of claim 21, wherein storing the first output data comprises: accumulating, in the memory, outputs of computational elements executed by the GPU in performing the first computation in the sequence of computations.”</p>	<p>Nickolls discloses “[t]he method of claim 21, wherein storing the first output data comprises: accumulating, in the memory, outputs of computational elements executed by the GPU in performing the first computation in the sequence of computations.” <i>See e.g.:</i></p> <p><i>See</i> 12[a][i], 12[a][ii].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>30[a] “The method of claim 21, further comprising: storing, in the memory, an output of a previous computation in the sequence of computations; and”</p>	<p>Nickolls discloses “[t]he method of claim 21, further comprising: storing, in the memory, an output of a previous computation in the sequence of computations.” <i>See e.g.:</i></p> <p><i>See</i> 12[a][i], 12[a][ii], claim 21.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>any supplements thereto and the relevant section of charts for other prior art for the '438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>30[b] “accessing, by the GPU, the output of the previous computation during performance of the computation in the sequence of computations.”</p>	<p>Nickolls discloses “accessing, by the GPU, the output of the previous computation during performance of the computation in the sequence of computations.” <i>See e.g.:</i></p> <p><i>See</i> 12[d][i], 12[d][ii].</p> <p><i>Nickolls further discloses a GPU (e.g., GPU 122 and/or processing engines) accessing the output of the previous computation during performance of another computation (e.g., by using the output of one computational cycle as the input for the next computational cycle):</i></p> <p>Embodiments of the present invention provide data processing systems and methods that use cooperative thread arrays (CTAs) to perform computations. As used herein, a “cooperative thread array,” or “CTA,” is a group of multiple threads that concurrently execute the same program on an input data set to produce an output data set. Each thread in a CTA has a unique identifier (thread ID) assigned at thread launch time that controls various aspects of the thread's processing behavior. For instance, a thread ID may be used to determine which portion of the input data set a thread is to process, to identify one or more other threads with which a given thread is to share an intermediate result, and/or to determine which portion of the output data set the thread is to compute or write.</p> <p>2:22–35.</p> <p>The CTA program may also include instructions to read from and/or write to the shared global register file 306, on-chip shared memory 308, and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1. For instance, the input data set to be processed by the CTA may be stored in graphics memory 124 or system memory 104. Intermediate results may be written to global register file 306 and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1, where they can be shared with other threads. Final results (output data) may be written to graphics memory 124 or system memory 104.</p> <p>20:35–45.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>A barrier arrival instruction might be used in preference to a barrier arrive-and-wait instruction, e.g., where one thread produces an intermediate result to be consumed by one or more other threads but does not consume intermediate results from those threads. The producer thread would not need to wait at the barrier point for the consumer threads, but the consumer threads would all need to wait for the producer thread to arrive at the barrier point before reading the data. Thus, the program instructions for the producer thread might include a barrier arrival instruction subsequent to an instruction to write the intermediate result to a shared memory location while the program instructions for the consumer thread might include a barrier arrive-and-wait instruction prior to an instruction to read the intermediate result from the shared memory location. In one embodiment, the barrier arrival and barrier arrive-and-wait instructions can be conditional, with each thread using its thread ID to determine which (if either) to execute, depending on whether the thread ID indicates that the thread is a producer or consumer. 25:7–25.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>31 “The method of claim 21, wherein performing the first computation comprises executing a plurality of computational elements representing a layer of neurons in an artificial neural network.”</p>	<p>Nickolls discloses “The method of claim 21, wherein performing the first computation comprises executing a plurality of computational elements representing a layer of neurons in an artificial neural network.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][i], 12[a][i].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>32 “The method of claim 31, wherein all neurons in the layer of neurons are described by the same equation.”</p>	<p>Nickolls discloses “The method of claim 31, wherein all neurons in the layer of neurons are described by the same equation.” <i>See e.g.:</i></p> <p><i>As a non-limiting example, Nickolls discloses all threads in the GPU (e.g., processing engines) execute the same equations (e.g., the same shader):</i></p> <p>Parallel data processing systems and methods use cooperative thread arrays (CTAs), i.e., groups of multiple threads that concurrently execute the same program on an input data set to produce an output data set. Each thread in a CTA has a unique identifier (thread ID) that can be assigned at thread launch time. The thread ID controls various aspects of the thread's processing behavior such as the portion of the input data set to be processed by each thread, the portion of an output data set to be produced by each thread, and/or sharing of intermediate results among threads. Mechanisms for loading and launching CTAs in a representative processing core and for synchronizing threads within a CTA are also described.</p> <p>Abstract.</p> <p>Embodiments of the present invention provide data processing systems and methods that use cooperative thread arrays (CTAs) to perform computations. As used herein, a “cooperative thread array,” or “CTA,” is a group of multiple threads that concurrently execute the same program on an input data set to produce an output data set. Each thread in a CTA has a unique identifier (thread ID) assigned at thread launch time that controls various aspects of the thread's processing behavior. For instance, a thread ID may be used to determine which portion of the input data set a thread is to process, to identify one or more other threads with which a given thread is to share an intermediate result, and/or to determine which portion of the output data set the thread is to compute or write.</p> <p>2:22–35.</p> <p>Embodiments of the present invention provide data processing systems and methods that use cooperative thread arrays (CTAs) to perform computations. As used herein, a “cooperative thread</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>array,” or “CTA,” is a group of multiple threads that concurrently execute the same program on an input data set to produce an output data set. Each thread in a CTA has a unique identifier (thread ID) assigned at thread launch time that controls various aspects of the thread's processing behavior. For instance, a thread ID may be used to determine which portion of the input data set a thread is to process, to identify one or more other threads with which a given thread is to share an intermediate result, and/or to determine which portion of the output data set the thread is to compute or write. 5:31–44.</p> <p>Instruction unit 312 is configured such that, for any given processing cycle, the same instruction (INSTR) is issued to all P processing engines 302. Thus, at the level of a single clock cycle, core 126 implements P-way SIMD execution. Each processing engine 302 is also multithreaded, supporting up to G concurrent threads, and instructions for different ones of the G threads can be issued in any order relative to each other. Accordingly, core 126 in this embodiment can have up to P*G threads in flight concurrently. For instance, if P=16 and G=24, then core 126 can support up to 384 concurrent threads. In some embodiments, P*G determines an upper limit on the number of threads that can be included in a CTA; it is to be understood that some CTAs may include fewer than this number of threads. 11:27–40.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>40[pre] “A system for executing an artificial neural network, the system comprising:”</p>	<p>Nickolls discloses “[a] system for executing an artificial neural network.” <i>See e.g.:</i></p> <p><i>See</i> 21[pre].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>40[a] “a central processing unit (CPU) to provide first input data;”</p>	<p>Nickolls discloses “a central processing unit (CPU) to provide first input data.” <i>See e.g.:</i></p> <p><i>See</i> 1[a].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>40[b] “a memory, operably coupled to the CPU, to store the first input data in a first partition, referenced by a first pointer, before computing a first layer of neurons of the artificial neural network;”</p>	<p>Nickolls discloses “a memory, operably coupled to the CPU, to store the first input data in a first partition, referenced by a first pointer, before computing a first layer of neurons of the artificial neural network.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][ii].</p> <p><i>As a non-limiting example, Nickolls discloses a memory (e.g., memory 308, local register file 304, and/or global register file 306) coupled to the CPU (e.g., CPU 102) that stores input data in a first “partition.” Nickolls also discloses referencing partitions by pointers (e.g., using indirect memory access, including calculating an address—which can be stored in a pointer—at which data can be accessed):</i></p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>In operation, each thread of the thread array uses the unique thread identifier to determine at least one processing behavior, such as a subset of an input data set to process and/or a portion of an output data set to produce or write.</p> <p>In one embodiment, the processing core includes a local register file having a number of entries allocated to each of the plurality of threads, and the thread identifier for each thread is stored in one of the local register file entries allocated to that thread. 3:54–62.</p> <p>The CTA program can also include an instruction to compute an address in shared memory from which data is to be read, with the address being a function of thread ID. By defining suitable functions, data can be written to a given location by one thread and read from that location by a different thread in a predictable manner. Consequently, any desired pattern of data sharing among threads can be supported, and any thread in a CTA can share data with any other thread in the same CTA. 8:22–30.</p> <p>For example, in one embodiment, state information includes the size of the input data set, the amount of local register file space required for each thread, and a starting program counter (e.g., memory address) for a program to be executed by each thread. 14:44–48.</p> <p>Instructions to access shared memory or shared global register file space advantageously identify locations to be read and/or written as a function of the thread ID. Where such an instruction is encountered, the thread reads its thread ID and computes the target address from the thread ID using the function specified in the CTA program. For instance, referring to FIG. 2C, a thread with ID (i1, i0) might be instructed to read the input data corresponding to the pixel to its right, with the address being specified as InputData[i1, i0+1]. In some embodiments, a conditional branch or conditional instruction in the CTA program might be provided to handle cases where the computed address is not within the valid range (e.g., for a thread processing a pixel at the right edge of the screen, an address such as InputData [i1, i0+1] would not be a valid pixel address). 20:46–60.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>40[c] “a processing unit, operably coupled to the memory, to perform, during computation of the first layer of neurons, at least one calculation on the first input data so as to generate first output data, the first output data representing an output of at least one neuron in the first layer of neurons; and”</p>	<p>Nickolls discloses “a processing unit, operably coupled to the memory, to perform, during computation of the first layer of neurons, at least one calculation on the first input data so as to generate first output data, the first output data representing an output of at least one neuron in the first layer of neurons.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][i], 21[d].</p> <p><i>As a non-limiting example, Nickolls discloses a processing unit (e.g., GPU 122) coupled to the memory (e.g., memory 308, local register file 304, and/or global register file 306) that performs general-purpose calculations on input data.</i></p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>40[d] “a controller, operably coupled to the processing unit and the memory, to:”</p>	<p>Nickolls discloses “a controller, operably coupled to the processing unit and the memory.” <i>See e.g.:</i></p> <p><i>See</i> 1[d].</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>40[d][i] “store the first output data in a second partition of the memory, the second partition referenced by a second pointer, and to swap the first pointer with the second pointer at the end of the computation of the first layer of neurons, such that the first output data becomes an input for a second layer of neurons of the artificial neural network,”</p>	<p>Nickolls discloses a controller “store the first output data in a second partition of the memory, the second partition referenced by a second pointer, and to swap the first pointer with the second pointer at the end of the computation of the first layer of neurons, such that the first output data becomes an input for a second layer of neurons of the artificial neural network.” <i>See e.g.:</i></p> <p><i>See</i> 12[a][i], 21[d], 40[b].</p> <p><i>As a non-limiting example, Nickolls discloses a memory (e.g., memory 308, local register file 304, and/or global register file 306) that stores output data in a second “partition.” Nickolls also discloses referencing partitions by pointers (e.g., using indirect memory access, including calculating an address—which can be stored in a pointer—at which data can be accessed). When using pointers, as disclosed in Nickolls, this use of an output as a subsequent input is most efficiently performed by swapping the first pointer to output data with the second pointer to input data:</i></p> <p>Embodiments of the present invention provide data processing systems and methods that use cooperative thread arrays (CTAs) to perform computations. As used herein, a “cooperative thread array,” or “CTA,” is a group of multiple threads that concurrently execute the same program on an input data set to produce an output data set. Each thread in a CTA has a unique identifier (thread ID) assigned at thread launch time that controls various aspects of the thread's processing behavior. For instance, a thread ID may be used to determine which portion of the input data set a thread is to process, to identify one or more other threads with which a given thread is to share an intermediate result, and/or to determine which portion of the output data set the thread is to compute or write.</p> <p>2:22–35.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>A barrier arrival instruction might be used in preference to a barrier arrive-and-wait instruction, e.g., where one thread produces an intermediate result to be consumed by one or more other threads but does not consume intermediate results from those threads. The producer thread would not need to wait at the barrier point for the consumer threads, but the consumer threads would all need to wait for the producer thread to arrive at the barrier point before reading the data. Thus, the program instructions for the producer thread might include a barrier arrival instruction subsequent to an instruction to write the intermediate result to a shared memory location while the program instructions for the consumer thread might include a barrier arrive-and-wait instruction prior to an instruction to read the intermediate result from the shared memory location. In one embodiment, the barrier arrival and barrier arrive-and-wait instructions can be conditional, with each thread using its thread ID to determine which (if either) to execute, depending on whether the thread ID indicates that the thread is a producer or consumer. 25:7–25.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>40[d][ii] “transfer the first output data to another memory during computation of the second layer of neurons, and”</p>	<p>Nickolls discloses a controller to “transfer the first output data to another memory during computation of the second layer of neurons.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][iv], 12[c].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>40[d][iii] “dictate an order of execution of instructions to the processing unit to perform the computation of the first layer of neurons.”</p>	<p>Nickolls discloses a controller to “dictate an order of execution of instructions to the processing unit to perform the computation of the first layer of neurons.” <i>See e.g.:</i></p> <p><i>As a non-limiting example, Nickolls discloses a controller (e.g., core interface 128 and/or instruction unit 312) that dictates an order of instructions to a processing unit (e.g., processing engines):</i></p> <p>Arbitration logic 320 and multiplexer 318 determine the order in which instructions are fetched. More specifically, on each clock cycle, arbitration logic 320 selects one of the G possible group indices GID as the SIMD group for which a next instruction should be fetched and supplies a corresponding control signal to multiplexer 318, which selects the corresponding PC. Arbitration logic 320 may include conventional logic for prioritizing and selecting among concurrent threads (e.g., using round-robin, least-recently serviced, or the like), and selection may be based in part on feedback information from fetch logic 322 or issue logic 324 as to how many instructions have been fetched but not yet issued for each SIMD group. 12:43–55.</p> <p>FIG. 8 is a block diagram of launch module 410 according to an embodiment of the present invention. Launch module 410 includes counter logic 804, a set of P parallel adders 806, a P-fold selection unit 808, and a valid/PC signaling block 810. Counter logic 804 receives the GO control signal from load module 408 (FIG. 4) and the CTA size T from state registers 416. Counter logic 804 uses the CTA size T to determine how many SIMD groups the CTA requires (in some embodiments, the number of groups required is T/P, rounded up to the next integer) and maintains a count of how many of the required SIMD groups have been launched. For each SIMD group, launch module 410 generates a set of P thread IDs, loads the P thread IDs into local register file 304 (FIG. 3), then signals instruction unit 312 of core 126 to begin executing the group. 17:37–51.</p> <p>To launch the first group of threads of the CTA, the P initial thread IDs are supplied to selection unit 808. In response to a control signal from counter logic 804 indicating that the first thread group is being launched, selection unit 808 selects the initial thread IDs for writing to registers in local register file</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>304. For all subsequent thread groups in the CTA, selection unit 808 selects the P new thread IDs computed by adders 806. (It should be noted that the thread IDs for a SIMD group need not be correlated with the group index GID assigned to the group for purposes of controlling execution.) 17:63–18:5.</p> <p>Once all of the input data has been received, core interface 128 enters the launch phase. In the launch phase, core interface 128 selects a group index GID for a first SIMD group to be launched (step 1018); any group index GID that is not already in use in core 126 may be selected. At step 1020, core interface 128 loads the initial set of P thread IDs into the local register file 304 of core 126 in locations corresponding to the selected group index GID or into per-thread registers in core 126 dedicated to storing thread IDs. Core interface 126 then instructs core 126 to launch the P threads as a SIMD group (step 1022). 19:51–61.</p> <p>In some embodiments of the present invention, a barrier synchronization technique is advantageously used to support fast synchronization of any number of CTA threads. More specifically, barrier instructions are inserted into the CTA program at points (referred to herein as “barrier points”) where synchronization is desired. A thread executes a barrier instruction to indicate that it has arrived at a barrier point and waits at that point until all other participating threads have also arrived at that point, thus synchronizing the participating threads before resuming execution. Arrival of each thread (or group of threads) at a barrier point is detected, and this information is used to synchronize two or more threads (or groups of threads). In one embodiment, execution of barrier instructions, i.e., arrival of threads (or SIMD groups) at barrier points, is detected by issue logic 324 of instruction unit 312 of FIG. 3, which can suspend the issue of instructions to any threads that are waiting at a barrier point while continuing to issue instructions to threads that are not at a barrier point. Eventually, all relevant threads reach the barrier point, and execution of the waiting threads resumes. 21:25–44.</p> <p>FIG. 11A is a block diagram of barrier synchronization logic 1100 according to an embodiment of the present invention. In some embodiments, barrier synchronization logic 1100 is implemented in issue logic 324 of instruction unit 312 and synchronizes SIMD groups rather than individual threads. 21:45–50.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>As shown in FIG. 11A, instruction unit 312 also includes selection logic 1110 that selects a next instruction to issue. Selection logic 1110 may be of generally conventional design, and a detailed description is omitted as not being critical to understanding the present invention. Barrier detection circuit 1112 receives each selected instruction (INST), along with the group identifier (GID) of the SIMD group for which the instruction is being issued. If the selected instruction is a barrier instruction, barrier detection circuit 1112 directs the instruction to barrier synchronization logic 1100; otherwise, barrier detection circuit 1112 forwards the instruction to the next issue stage for eventual delivery to processing engines 302 of FIG. 3. 21:51–63.</p> <p>In operation, when barrier synchronization logic 1130 receives a first barrier instruction pertaining to a barrier point BarID, the target value is loaded into the corresponding target register 1135. For every barrier instruction pertaining to barrier point BarID (including the first), the corresponding counter 1134 is incremented. In addition, if the barrier instruction indicates that the SIMD group is to wait for synchronization, the wait/go bit corresponding to group GID is set to the wait state in wait/go registers 1138, and the barrier identifier BarID is stored in the BarID field for that group. As described above, wait/go registers 1138 are advantageously read by selection logic 1110, and selection logic 1110 does not select instructions for SIMD groups that are in the wait state, thereby suspending execution of such groups. Selection logic 1110 may continue to select instructions for other SIMD groups. 23:52–67.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
41 “The system of claim 40, wherein the processing	Nickolls discloses “[t]he system of claim 40, wherein the processing unit comprises a graphics processing unit.” <i>See e.g.:</i>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
<p>unit comprises a graphics processing unit.”</p>	<p><i>See</i> 1[c][i].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>42 “The system of claim 40, wherein the controller is configured to send instructions for performing the at least one calculation to the processing unit.”</p>	<p>Nickolls discloses “[t]he system of claim 40, wherein the controller is configured to send instructions for performing the at least one calculation to the processing unit.” <i>See e.g.</i>:</p> <p><i>See</i> 1[d][ii].</p> <p><i>As a non-limiting example, Nickolls discloses a controller (e.g., core interface 128 alone or with instruction unit 312) that “controls operation of core 126,” including sending instructions to the processing engines:</i></p> <p>GPU 122 has at least one processing core 126 for generating pixel data and a core interface 128 that controls operation of core 126. 6:47–49.</p> <p>Instruction unit 312 is configured such that, for any given processing cycle, the same instruction (INSTR) is issued to all P processing engines 302. Thus, at the level of a single clock cycle, core 126 implements P-way SIMD execution. Each processing engine 302 is also multithreaded, supporting up to G concurrent threads, and instructions for different ones of the G threads can be issued in any order relative to each other. Accordingly, core 126 in this embodiment can have up to P*G threads in flight concurrently. For instance, if P=16 and G=24, then core 126 can support up to 384 concurrent threads. In some embodiments, P*G determines an upper limit on the number of threads that can be included in a CTA; it is to be understood that some CTAs may include fewer than this number of threads.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>11:27–40.</p> <p>Instruction unit 312 advantageously manages instruction fetch and issue for each SIMD group so as to ensure that threads in a group that have diverged eventually resynchronize. In one embodiment, instruction unit 312 includes program counter (PC) logic 314, a program counter register array 316, a multiplexer 318, arbitration logic 320, fetch logic 322, and issue logic 324. Program counter register array 316 stores G program counter values (one per SIMD group), which are updated independently of each other by PC logic 314. PC logic 314 updates the PC values based on information received from processing engines 302 and/or fetch logic 322. PC logic 314 is advantageously configured to track divergence among threads in a SIMD group and to select instructions in a way that ultimately results in the threads re-synchronizing.</p> <p>12:19–33.</p> <p>Arbitration logic 320 and multiplexer 318 determine the order in which instructions are fetched. More specifically, on each clock cycle, arbitration logic 320 selects one of the G possible group indices GID as the SIMD group for which a next instruction should be fetched and supplies a corresponding control signal to multiplexer 318, which selects the corresponding PC. Arbitration logic 320 may include conventional logic for prioritizing and selecting among concurrent threads (e.g., using round-robin, least-recently serviced, or the like), and selection may be based in part on feedback information from fetch logic 322 or issue logic 324 as to how many instructions have been fetched but not yet issued for each SIMD group.</p> <p>12:43–55.</p> <p>For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302 with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group. 13:12–45.</p> <p>It will be appreciated that the processing core described herein is illustrative and that variations and modifications are possible. Any number of processing units may be included. In some embodiments, each processing unit has its own local register file, and the allocation of local register file entries per thread can be fixed or configurable as desired. 13:46–51.</p> <p>In some embodiments, core 126 is operated at a higher clock rate than core interface 128, allowing the core to process more data in a given amount of time. For instance, core 126 can be operated at a clock rate that is twice the clock rate of core interface 128. If core 126 includes P processing engines 302 producing data at twice the core interface clock rate, then core 126 can produce 2*P data values per core interface clock cycle. Provided there is sufficient space in local register file 304, from the perspective of core interface 128, the situation is effectively identical to a core with 2*P processing units. Thus, P-way SIMD parallelism could be produced either by including P processing units in core 126 and operating core 126 at the same clock rate as core interface 128 or by including P/2 processing units in core 126 and operating core 126 at twice the clock rate of core interface 128. Other timing variations are also possible. 13:52–67.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>FIG. 4 is a block diagram of core interface 128 according to an embodiment of the present invention. Core interface 128 includes an input unit 402, a state module 404, state registers 406, a load module 408, and a launch module 410. As described above, core interface 128 (FIG. 1) controls operation of core 126. In particular, core interface 128 loads and launches threads of a CTA in SIMD groups until all threads have been launched. In one embodiment, core interface 128 generates thread IDs for each thread in a SIMD group and writes the thread ID into a suitable location in local register file 304, then launches the SIMD group. These aspects of core interface 128 will now be described.</p> <p>14:17–28.</p> <p>Once all of the input data has been received, core interface 128 enters the launch phase. In the launch phase, core interface 128 selects a group index GID for a first SIMD group to be launched (step 1018); any group index GID that is not already in use in core 126 may be selected. At step 1020, core interface 128 loads the initial set of P thread IDs into the local register file 304 of core 126 in locations corresponding to the selected group index GID or into per-thread registers in core 126 dedicated to storing thread IDs. Core interface 126 then instructs core 126 to launch the P threads as a SIMD group (step 1022).</p> <p>19:51–61.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>43[a] “The system of claim 40, wherein the memory further comprises: a third partition to store internal variables; and”</p>	<p>Nickolls discloses “[t]he system of claim 40, wherein the memory further comprises: a third partition to store internal variables.” <i>See e.g.:</i></p> <p><i>See</i> 6[a]; claims 28, 40.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p data-bbox="583 235 1871 342"><i>As a non-limiting example, Nickolls discloses a memory (e.g., memory 308, local register file 304, and/or global register file 306) that stores “internal variables,” such as input parameters, in a third “partition” (e.g., a logical partition in the memory):</i></p> <p data-bbox="583 381 1822 488">Nickolls discloses a “third partition” (e.g., a logical partition in memory in GPU 122) for storing “internal variables.” For example, the data stored in shared memory 306 for the processing cores includes “input parameters.”</p> <p data-bbox="583 527 1913 852">Embodiments of the present invention provide data processing systems and methods that use cooperative thread arrays (CTAs) to perform computations. As used herein, a “cooperative thread array,” or “CTA,” is a group of multiple threads that concurrently execute the same program on an input data set to produce an output data set. Each thread in a CTA has a unique identifier (thread ID) assigned at thread launch time that controls various aspects of the thread's processing behavior. For instance, a thread ID may be used to determine which portion of the input data set a thread is to process, to identify one or more other threads with which a given thread is to share an intermediate result, and/or to determine which portion of the output data set the thread is to compute or write. 2:22–35.</p> <p data-bbox="583 893 1898 1218">Each processing engine 302 is allocated space in a local register file 304 for storing its local input data, intermediate results, and the like. In one embodiment, local register file 304 is divided into P lanes, each having some number of entries (where each entry might be, e.g., a 32-bit word). One lane is allocated to each processing unit, and corresponding entries in different lanes can be populated with data for corresponding thread types to facilitate SIMD execution of multiple threads in parallel as described below. The number of entries in local register file 304 is advantageously large enough to support multiple concurrent threads per processing engine 302; and in some embodiments, the number of entries allocated to a thread is dynamically configurable. 10:57–11:2.</p> <p data-bbox="583 1258 1898 1399">For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group. 13:12–45.</p> <p>Referring again to FIG. 4, load module 408 receives input parameters for a CTA and loads the input parameters into (shared) global register file 306 (FIG. 3) of core 126. These parameters may include, e.g., an (x,y) position of the CTA within a grid of CTAs or other CTA identifier and/or other information specific to the CTA. In some embodiments, the input parameters may also include some or all of an input data set to be processed by the CTA. 16:63–17:3.</p> <p>In some embodiments, the target value used to determine when synchronization is achieved may be specified as being equal to the total number of executing threads of the CTA, which can be dynamically determined by barrier synchronization logic 1100 or 1130. Although the total number of threads in a CTA can be an input parameter, as described above, in some instances, not all threads are necessarily executing at a given time; accordingly, a dynamic determination of the total is advantageous. Specifying “all executing threads” as the target value can be done, e.g., by using a predefined special value (e.g., zero) for the argument that specifies the target value or by providing a separate barrier</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>instruction that signifies that the target value is “all executing threads.” (Such an instruction would not include a target value as an argument.) Where dynamic determination of the target number is used, barrier synchronization logic 1100 or 1130 advantageously recomputes the target number from time to time so that the target remains current. 24:39–57.</p> <p>A barrier arrival instruction might be used in preference to a barrier arrive-and-wait instruction, e.g., where one thread produces an intermediate result to be consumed by one or more other threads but does not consume intermediate results from those threads. The producer thread would not need to wait at the barrier point for the consumer threads, but the consumer threads would all need to wait for the producer thread to arrive at the barrier point before reading the data. Thus, the program instructions for the producer thread might include a barrier arrival instruction subsequent to an instruction to write the intermediate result to a shared memory location while the program instructions for the consumer thread might include a barrier arrive-and-wait instruction prior to an instruction to read the intermediate result from the shared memory location. In one embodiment, the barrier arrival and barrier arrive-and-wait instructions can be conditional, with each thread using its thread ID to determine which (if either) to execute, depending on whether the thread ID indicates that the thread is a producer or consumer. 25:7–25.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>43[b] “a fourth partition to store data used as input at a particular layer of neurons of the artificial neural network.”</p>	<p>Nickolls discloses “a fourth partition to store data used as input at a particular layer of neurons of the artificial neural network.” <i>See e.g.:</i></p> <p><i>See</i> 6[b], 43[a], claim 28.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p data-bbox="583 235 1864 342"><i>As a non-limiting example, Nickolls discloses a memory (e.g., memory 308, local register file 304, and/or global register file 306) that stores input data to be processed in a fourth “partition” (e.g., a logical partition in the memory):</i></p> <p data-bbox="583 383 1902 776">Each processing engine 302 also has access, via a crossbar switch 305, to a (shared) global register file 306 that is shared among all of the processing engines 302 in core 126. Global register file 306 may be as large as desired, and in some embodiments, any processing engine 302 can read to or write from any location in global register file 306. In addition to global register file 306, some embodiments also provide an on-chip shared memory 308, which may be implemented, e.g., as a conventional RAM. On-chip memory 308 is advantageously used to store data that is expected to be used in multiple threads, such as coefficients of attribute equations, which are usable in pixel shader programs. In some embodiments, processing engines 302 may also have access to additional off-chip shared memory (not shown), which might be located, e.g., within graphics memory 124 and/or system memory 104 of FIG. 1. 11:3–18.</p> <p data-bbox="583 821 1902 1399">For optimal performance, all threads within a SIMD group are advantageously launched on the same clock cycle so that they begin in a synchronized state. In one embodiment, core interface 128 advantageously loads SIMD groups into core 126, then instructs core 126 to launch the group. “Loading” a thread, as used herein, includes supplying instruction unit 312 and processing engines 302 with various input parameters required to execute the program. In some instances, the input parameters may include the input data to be processed by the program; in other instances, the input data is stored in global register file 306 or other shared memory (e.g., graphics memory 124 or system memory 104 of FIG. 1) prior to loading of any threads, and the input parameters may include a reference to a location where the input data is stored. For example, in the case of CTA processing, the input data set may be loaded into graphics memory 124 or system memory 104 before core interface is instructed to begin CTA processing. Core interface 128 loads the starting PC value for the CTA program into a slot in PC array 316 that is not currently in use; this slot corresponds to the group index GID assigned to the new SIMD group that will process P of the CTA threads. Core interface 128 allocates sufficient space in the local register file for each processing engine 302 to execute one CTA thread, then loads input parameters into shared memory (e.g., global register file 306). Core interface 128 loads a unique thread ID into a thread ID register for each thread or into a predetermined register in the allocated portion of</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>local register file 304 for each processing engine 302. In one embodiment, thread IDs for P threads are loaded in parallel, as described below. Once the input parameters and thread IDs for all threads in the SIMD group have been loaded, core interface 128 launches the group by signaling instruction unit 312 to begin fetching and issuing instructions corresponding to the group index GID of the new group. 13:12–45.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>44[pre] “A computer system, comprising:”</p>	<p><i>See</i> 1[pre].</p>
<p>44[a] “a central processing unit to receive input data acquired from an external system;”</p>	<p>Nickolls discloses “a central processing unit to receive input data acquired from an external system.” <i>See e.g.</i>:</p> <p><i>See</i> 1[a], 21[a].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>44[b] “main memory, operably coupled to the central processing unit via</p>	<p>Nickolls discloses “main memory, operably coupled to the central processing unit via a bus, to store the input data received by the central processing unit.” <i>See e.g.</i>:</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
<p>a bus, to store the input data received by the central processing unit;”</p>	<p><i>See</i> 1[b].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>44[c] “an accelerator, operably coupled to the central processing unit and the main memory via the bus, to receive at least a portion of the input data from the main memory, the accelerator comprising:”</p>	<p>Nickolls discloses “an accelerator, operably coupled to the central processing unit and the main memory via the bus, to receive at least a portion of the input data from the main memory.” <i>See e.g.:</i></p> <p><i>See</i> 1[c].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>44[c][i] “at least one processing unit to perform a sequence of computations representing an artificial neural network on the at least a portion of the input data so as to generate output data, intermediate computations in the</p>	<p>Nickolls discloses “at least one processing unit to perform a sequence of computations representing an artificial neural network on the at least a portion of the input data so as to generate output data, intermediate computations in the sequence of computations representing layers of the neural network and yielding intermediate results.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][i].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
<p>sequence of computations representing layers of the neural network and yielding intermediate results; and”</p>	<p>the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>44[c][ii] “accelerator memory, operably coupled to the at least one processing unit, to store the results of the sequence of computations; and”</p>	<p>Nickolls discloses “accelerator memory, operably coupled to the at least one processing unit, to store the results of the sequence of computations.” <i>See e.g.:</i></p> <p><i>See</i> 1[c][ii].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>44[d] “a controller, operably coupled to the at least one processing unit and the accelerator memory,”</p>	<p>Nickolls discloses “a controller, operably coupled to the at least one processing unit and the accelerator memory.” <i>See e.g.:</i></p> <p><i>See</i> 1[d].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
<p>44[d][i] “to control transfer of the at least a portion of the input data into the accelerator memory during performance of the intermediate computations in the sequence of computations by the at least one processing unit,”</p>	<p>Nickolls discloses a controller “to control transfer of the at least a portion of the input data into the accelerator memory during performance of the intermediate computations in the sequence of computations by the at least one processing unit.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][iii].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>44[d][ii] “to control transfer at least a portion of the output data from the accelerator memory to the main memory during performance of the intermediate computations in the sequence of computations by the at least one processing unit, and”</p>	<p>Nickolls discloses a controller “to control transfer at least a portion of the output data from the accelerator memory to the main memory during performance of the intermediate computations in the sequence of computations by the at least one processing unit.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][iv].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>44[d][iii] “to control performance of the sequence of computations</p>	<p>Nickolls discloses a controller “to control performance of the sequence of computations by the at least one processing unit.” <i>See e.g.:</i></p> <p><i>See</i> 1[d][ii].</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
<p>by the at least one processing unit.”</p>	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>45 “The computer system of claim 44, wherein the central processing unit is configured to receive the input data in response to a user interaction.”</p>	<p>Nickolls discloses “[t]he computer system of claim 44, wherein the central processing unit is configured to receive the input data in response to a user interaction.” <i>See e.g.:</i></p> <p><i>See</i> claim 2.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>46[pre] “The computer system of claim 44, wherein:”</p>	<p>Nickolls discloses “[t]he computer system of claim 44.” <i>See e.g.:</i></p> <p><i>See</i> claim 44.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>46[a] “the central processing unit is configured to receive the input data at a first rate; and”</p>	<p>Nickolls discloses that “the central processing unit is configured to receive the input data at a first rate.” <i>See e.g.:</i></p> <p><i>See</i> 3[a].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>46[b] “the at least one processing unit is configured to perform the sequence of computations at a second rate different than the first rate.”</p>	<p>Nickolls discloses that “the at least one processing unit is configured to perform the sequence of computations at a second rate different than the first rate.” <i>See e.g.:</i></p> <p><i>See</i> 3[b].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>47 “The computer system of claim 44, wherein the main memory is configured to store a copy of the</p>	<p>Nickolls discloses “[t]he computer system of claim 44, wherein the main memory is configured to store a copy of the output data stored in the accelerator memory.” <i>See e.g.:</i></p> <p><i>See</i> claim 4.</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
output data stored in the accelerator memory.”	To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.
48 “The computer system of claim 44, wherein an output of at least one computation in the sequence of computations represents an output of at least one neuron in an artificial neural network.”	Nickolls discloses “[t]he computer system of claim 44, wherein an output of at least one computation in the sequence of computations represents an output of at least one neuron in an artificial neural network.” <i>See e.g.:</i> <i>See</i> claim 5. To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.
49[pre] “The computer system of claim 44, wherein accelerator memory comprises:”	<i>See</i> claim 44.
49[a] “a first memory partition to store parameters common to all of the computations in the	Nickolls discloses “a first memory partition to store parameters common to all of the computations in the sequence of computations.” <i>See e.g.:</i> <i>See</i> 6[a]; claim 26.

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
sequence of computations; and”	<p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
49[b] “a second memory partition to store data specific to at least one computation in the sequence of computations.”	<p>Nickolls discloses “a second memory partition to store data specific to at least one computation in the sequence of computations.” <i>See e.g.:</i></p> <p><i>See</i> 6[b]; claim 27.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
50 “The computer system of claim 44, wherein the controller is configured to transfer the output data from the accelerator memory to the main memory without transferring any of the intermediate results from the accelerator memory to the main memory so as to	<p>Nickolls discloses “[t]he computer system of claim 44, wherein the controller is configured to transfer the output data from the accelerator memory to the main memory without transferring any of the intermediate results from the accelerator memory to the main memory so as to reduce data transfer via the bus.” <i>See e.g.:</i></p> <p><i>See</i> claim 7.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidation Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
<p>reduce data transfer via the bus.”</p>	<p>manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>51 “The computer system of claim 44, wherein the controller is configured to transfer at least a portion of the output data from the accelerator memory to the main memory after the at least one processing unit has begun to perform another sequence of computations.”</p>	<p>Nickolls discloses “[t]he computer system of claim 44, wherein the controller is configured to transfer at least a portion of the output data from the accelerator memory to the main memory after the at least one processing unit has begun to perform another sequence of computations.” <i>See e.g.:</i></p> <p><i>See claim 8.</i></p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>52 “The computer system of claim 51, wherein the controller is configured to initiate transfer of the at least a portion of the input data and to transfer the at least a portion of the output data in parallel with performance of at least one computation in the other sequence of computations by the at least one processing unit.”</p>	<p>Nickolls discloses “[t]he computer system of claim 51, wherein the controller is configured to initiate transfer of the at least a portion of the input data and to transfer the at least a portion of the output data in parallel with performance of at least one computation in the other sequence of computations by the at least one processing unit.” <i>See e.g.:</i></p> <p><i>See claim 9.</i></p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the</p>

Ex. B8 – Invalidity of U.S. Patent No. RE48,438 – Nickolls

U.S. Patent No. RE48,438	Nickolls
	<p>teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>53 “The computer system of claim 44, wherein the controller is configured to control execution of the sequence of computations by the at least one processing unit.”</p>	<p>Nickolls discloses “[t]he computer system of claim 44, wherein the controller is configured to control execution of the sequence of computations by the at least one processing unit.” <i>See e.g.:</i></p> <p><i>See</i> claim 10.</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>
<p>54 “The computer system of claim 44, further comprising: at least one of a video camera, a microphone, or a cell recording electrode, operably coupled to the central processing unit, to acquire the input data in real time.”</p>	<p>Nickolls discloses “[t]he computer system of claim 44, further comprising: at least one of a video camera, a microphone, or a cell recording electrode, operably coupled to the central processing unit, to acquire the input data in real time.” <i>See e.g.:</i></p> <p><i>See</i> 1[a].</p> <p>To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Nickolls and the knowledge of a POSITA, it would have been obvious to combine the teachings of Nickolls with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’438 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Nickolls. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts B1–B12, and Appendix B.</p>