Graphics Processing Units:

- For a few hundred dollars, anyone can buy a GPU with hundreds of parallel floating-point units, which makes high-performance computing more accessible. The interest in GPU computing blossomed when this potential was combined with a programming language that made GPUs easier to program. Hence, many programmers of scientific and multimedia applications today are pondering whether to use GPUs or CPUs.
- The primary ancestors of GPUs are graphics accelerators, as doing graphics well is the reason why GPUs exist. While GPUs are moving toward mainstream computing, they can't abandon their responsibility to continue to excel at graphics.
- **A graphics processing unit** (GPU), is similar CPU Designed specifically for performing the complex mathematical and geometric calculations that are necessary for graphics rendering.
- A graphics processing unit (GPU) is a computer chip that performs rapid mathematical calculations, primarily for the purpose of rendering images.
- They can also occasionally be called **visual processing unit** (**VPU**)
- GPU is able to render images more quickly than a CPU because of its parallel processing architecture
- Nvidia introduced the first GPU, the GeForce 256, in 1999. In 2012, Nvidia released a virtualized GPU, which offloads graphics processing from the server CPU in a virtual desktop infrastructure. Others include AMD, Intel and ARM.
- GPUs are used in :
    - Embedded Systems
    - Mobile phones
    - Personal computers
    - Workstations
    - Game consoles
- A GPU is tailored for highly parallel operation while a CPU executes programs serially. For this reason, GPUs have many parallel execution units and higher transistor counts, while CPUs have few execution units

and higher clock speeds.A GPU is for the most part deterministic in its operation.

- GPUs have much deeper pipelines (several thousand stages vs 10-20 for CPUs).GPUs have significantly faster and more advanced memory interfaces as they need to shift around a lot more data than CPUs
- Very Efficient For
    - Fast Parallel Floating Point Processing
    - Single Instruction Multiple Data Operations
    - High Computation per Memory Access
- Not Efficient For
    - Double Precision
    - Logical Operations on Integer Data
    - Branching-Intensive Operations
    - Random Access, Memory-Intensive Operations

- The mapping of a Grid (vectorizable loop), Thread Blocks (SIMD basic blocks), and threads of SIMD instructions to a vector-vector multiply, with each vector being 8192 elements long. Each thread of SIMD instructions calculates 32 elements per instruction, and in this example each Thread Block contains 16 threads of SIMD instructions and the Grid contains 16 Thread Blocks. The hardware Thread Block Scheduler assigns Thread Blocks to multithreaded SIMD Processors and the hardware Thread Scheduler picks which thread of SIMD instructions to run each clock cycle within a SIMD Processor. Only SIMD Threads in the same Thread Block can communicate via Local Memory. (The maximum number of SIMD Threads that can execute simultaneously per Thread Block is 16 for Teslageneration GPUs and 32 for the later Fermi-generation GPUs.)

- A Thread Block is assigned to a processor that executes that code, which we call a multithreaded SIMD Processor, by the Thread Block Scheduler. The Thread Block Scheduler has some similarities to a control processor in a vector architecture. It determines the number of thread blocks needed

for the loop and keeps allocating them to different multithreaded SIMD Processors until the loop is completed. In this example, it would send 16 Thread Blocks to multithreaded SIMD Processors to compute all 8192 elements of this loop.

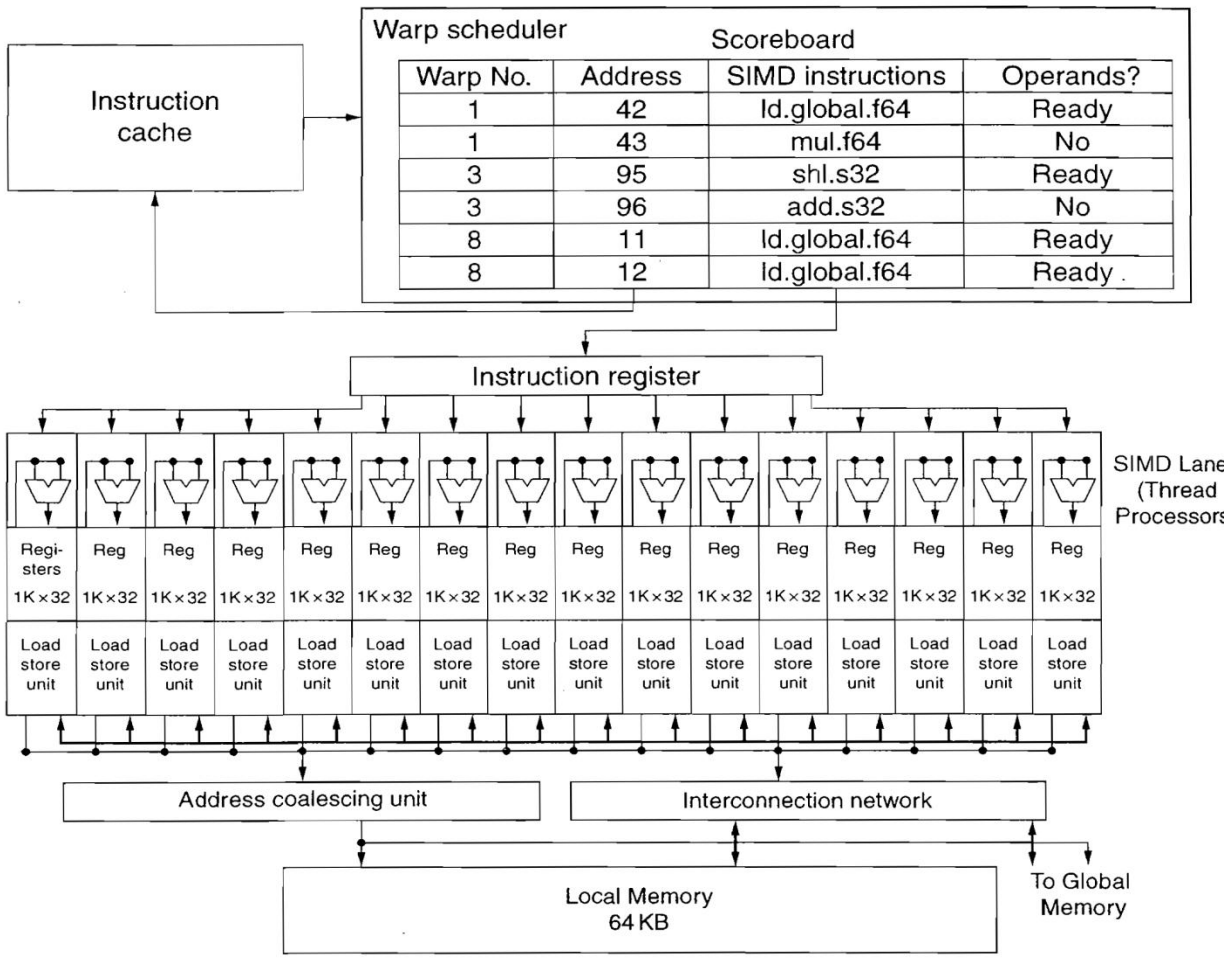

Fig: Block Diagram of Multithreaded SIMD Processor.

- Fig above shows a simplified block diagram of a multithreaded S1MD Processor. It is similar to a Vector Processor, but it has many parallel functional units instead of a few that are deeply pipelined, as does a

Vector Processor. each multithreaded SIMD Processor is assigned 512 elements of the vectors to work on. SIMD Processors are full processors with separate PCs and are programmed using threads.

- The GPU hardware then contains a collection of multithreaded SIMD Processors that execute a Grid of Thread Blocks (bodies of vectorized loop); that is, a GPU is a multiprocessor composed of multithreaded SIMD Processors. Thread Block Scheduler assigns Thread Blocks (bodies of a vectorized loop) to multithreaded SIMD Processors. . It is a traditional thread that contains exclusively SIMD instructions. These threads of SIMD instructions have their own PCs and they run on a multithreaded SIMD Processor.
- The SIMD Thread Scheduler includes a scoreboard that lets it know which threads of SIMD instructions are ready to run, and then it sends them off to a dispatch unit to be run on the multithreaded SIMD Processor. It is identical to a hardware thread scheduler in a traditional multithreaded processor just that it is scheduling threads of SIMD instructions.
- GPU hardware has two levels of hardware schedulers:
  1) the *Thread Block Scheduler* that assigns Thread Blocks (bodies of vectorized loops) to multithreaded SIMD Processors, which ensures that thread blocks are assigned to the processors whose local memories have the corresponding data.
  (2) the SIMD Thread Scheduler within a SIMD Processor, which schedules when threads of SIMD instructions should run.
- The SIMD instructions of these threads are 32 wide, so each thread of SIMD instructions in this example would compute 32 of the elements of the computation. In this example, Thread Blocks would contain 512/32 = 16 SIMD threads. Since the thread consists of SIMD instructions, the SIMD Processor must have parallel functional units to perform the operation. We call them SIMD Lanes.

- The number of lanes per SIMD processor varies across GPU generations. With Fermi, each 32-wide thread of SIMD instructions is mapped to 16 physical SIMD Lanes, so each SIMD instruction in a thread of SIMD instructions takes two clock cycles to complete. Each thread of SIMD instructions is executed in lock step and only scheduled at the beginning.
- The threads of SIMD instructions are independent, the SIMD Thread Scheduler can pick whatever thread of SIMD instructions is ready, and need not stick with the next SIMD instruction in the sequence within a thread.
- SIMD Thread Scheduler picking threads of SIMD instructions in a different order over time. The assumption of GPU architects is that GPU applications have so many threads of SIMD instructions that multithreading can both hide the latency to DRAM and increase utilization of multithreaded SIMD Processors.

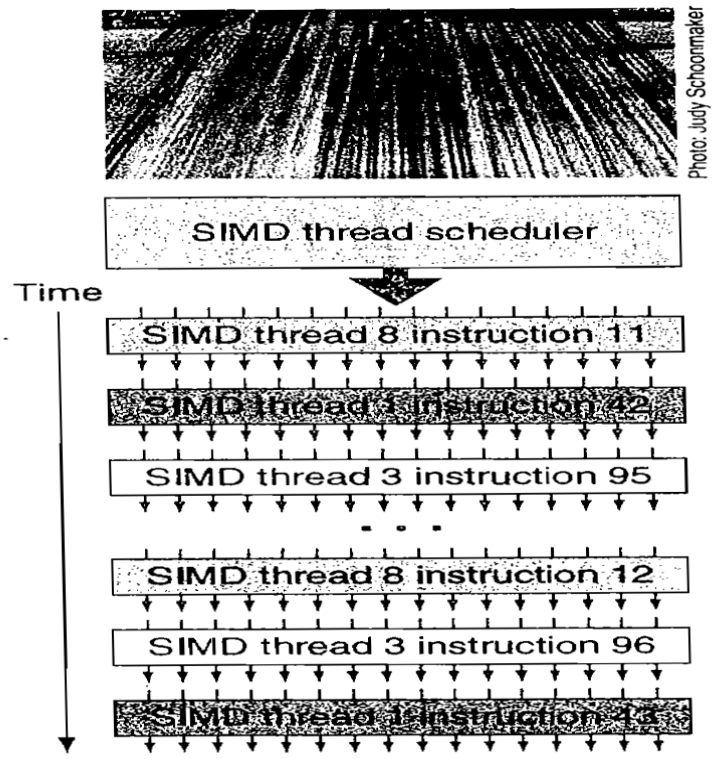

Fig: Schduling of Threads of SIMD Instructions

- The scheduler selects a ready thread of SIMD instructions and issues an instruction synchronously to all the SIMD Lanes executing the SIMD

thread. Because threads of SIMD instructions are independent, the scheduler may select a different SIMD thread each time.

- Thread as having up to 64 vector registers, with each vector register having 32elements and each element being 32 bits wide. Since Fermi has 16 physical SIMD Lanes, each contains 2048 registers) Each CUDA Thread gets one element of each of the vector registers. To handle the 32elements of each thread of SIMD instructions with 16 SIMD Lanes, the CUDA. Threads of a Thread block collectively can use up to half of the 2048 registers.