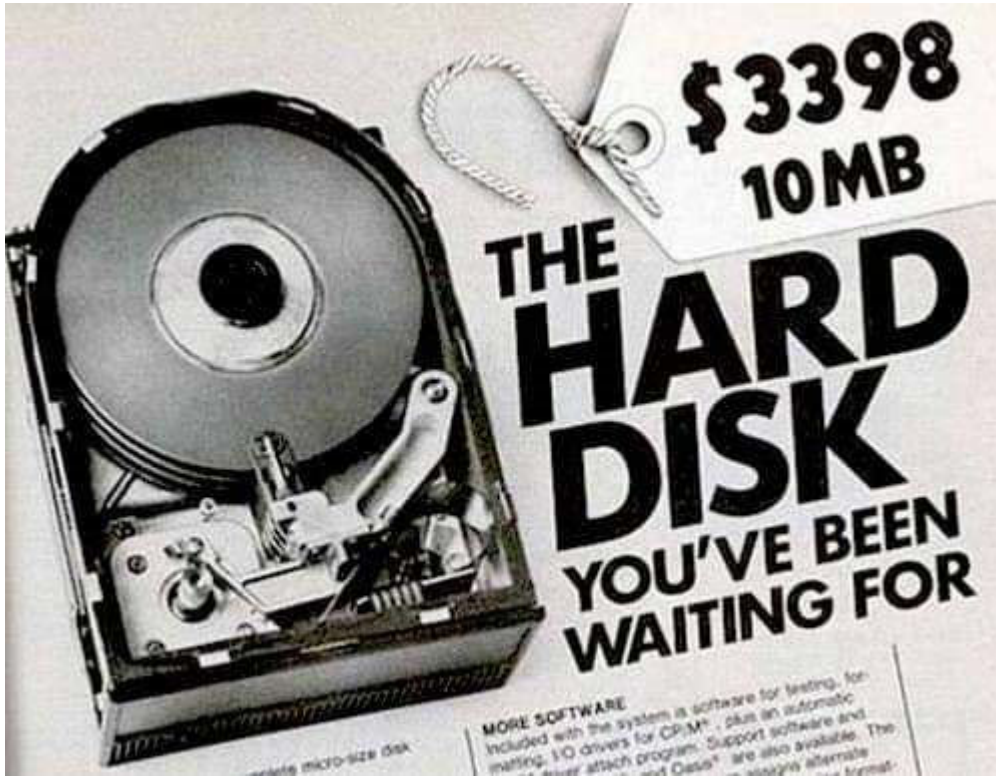
A look at the past…



A look at today…

WD - easystore® 8TB External USB 3.0 Hard Drive - Black

Model: WDBCKA0080HBK-NESN **SKU:** 5792401

4.7 (3,225 Customers) | 88 Answered Questions

Only @ Best Buy

$199.99 Or **$33.34/mo.*** suggested payments with 6-Month Financing Show me how >

Save $80

Was $279.99

Free item with purchase

Protect your product

Learn about Replacement Plans

2 Years $19.99 | No plan selected

Add to Cart

Save for Later

Compare

Hard Drive Capacity:

4000GB | 8000GB | 10000GB

FREE Shipping: Get it by Mon, Dec 3

Want it faster? Pick it up at Salem or see more shipping options in checkout to 97330.

Cardmember Offers

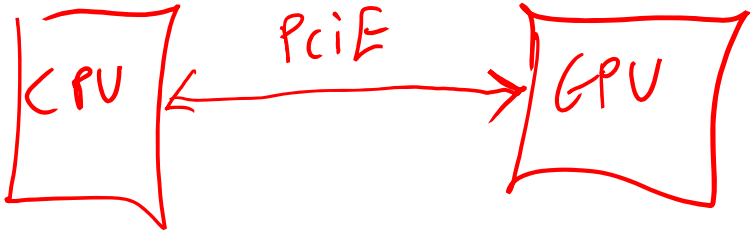
6 Month Financing

Get 5% Back in Rewards

Moving into GPUs…

Graphical Processing Units

- Given the hardware invested to do graphics well, how can be supplement it to improve performance of a wider range of applications?
- Basic idea:



  - Heterogeneous execution model
    - CPU is the *host*, GPU is the *device*
  - Develop a C-like programming language for GPU
  - Unify all forms of GPU parallelism as *CUDA thread*
  - Programming model is “Single Instruction Multiple Thread”

NVIDIA GPU Architecture

- Similarities to vector machines:
  - Works well with data-level parallel problems
  - Scatter-gather transfers
  - Mask registers
  - Large register files
- Differences:
  - No scalar processor
  - Uses multithreading to hide memory latency
  - Has many functional units, as opposed to a few deeply pipelined units like a vector processor

Terminology

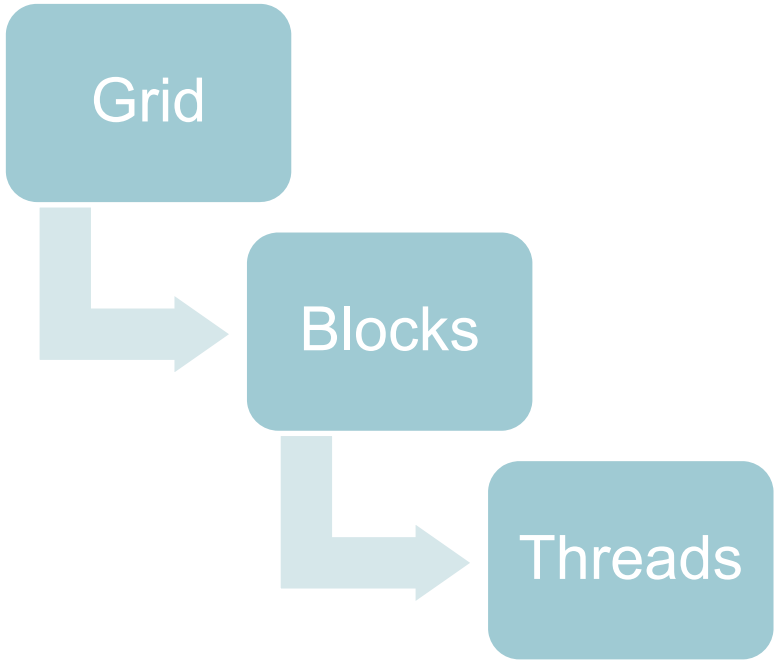
- *Threads of SIMD instructions*
  - Each has its own PC
  - Thread scheduler uses scoreboard to dispatch
  - No data dependencies between threads!
  - Keeps track of up to 48 threads of SIMD instructions
    - Hides memory latency
- Thread block scheduler schedules blocks to SIMD processors
- Within each SIMD processor:
  - 32 SIMD lanes
  - Wide and shallow compared to vector processors

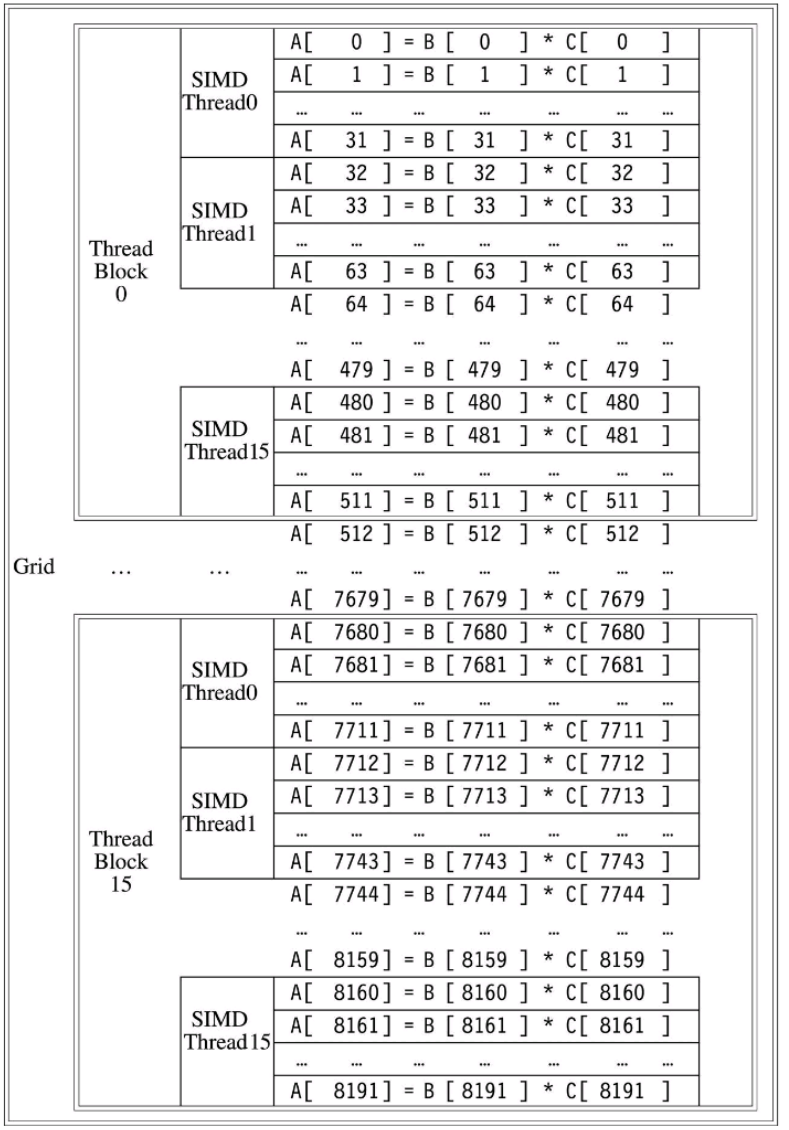
Threads and Blocks

- A thread is associated with each data element
- Threads are organized into blocks
- Blocks are organized into a grid

- GPU hardware handles thread management, not applications or OS

Graphical Explanation





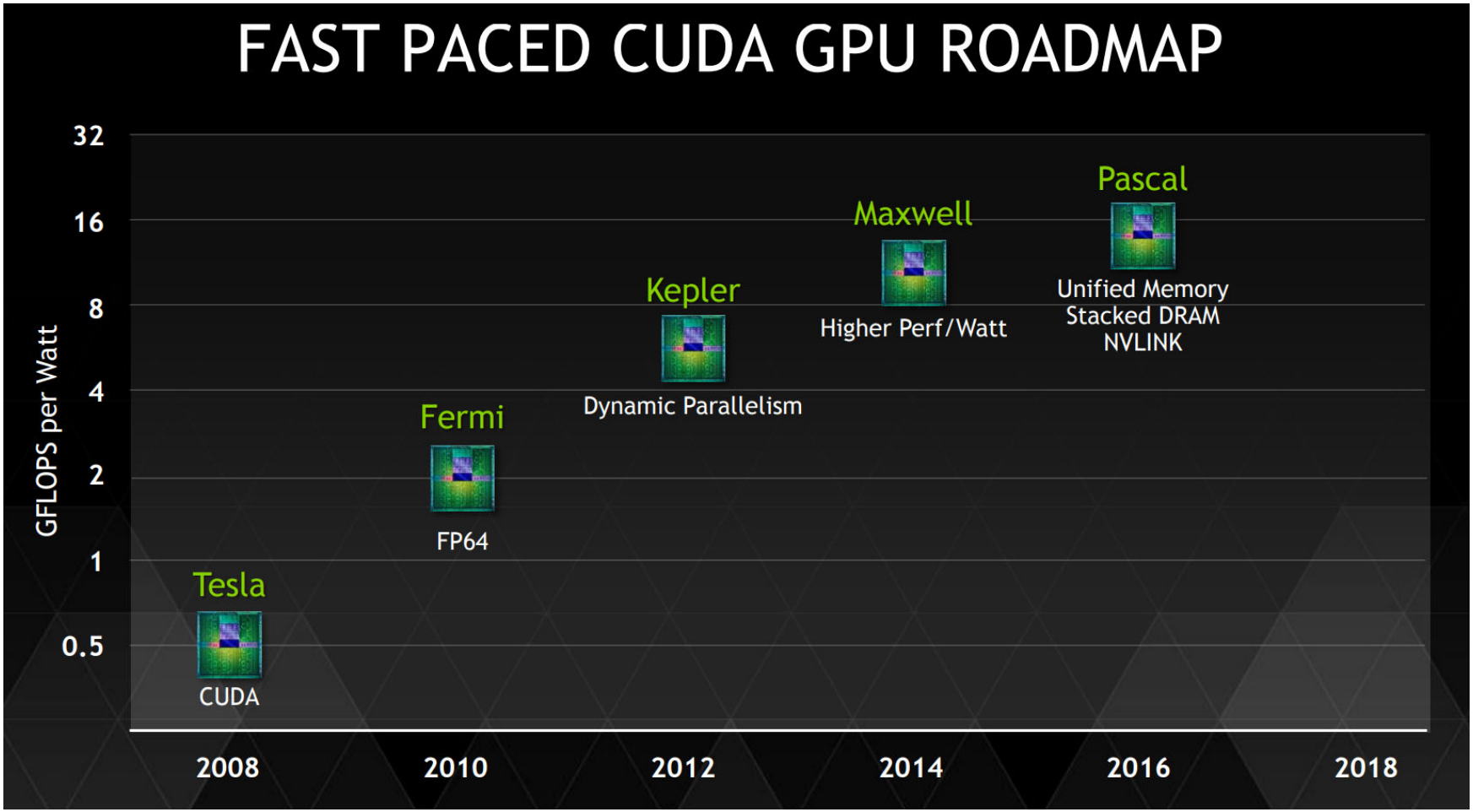
Example

- Multiply two vectors of length 8192
  - Code that works over all elements is the grid
  - Thread blocks break this down into manageable sizes
    - Up to 512 elements per block
  - SIMD instruction executes 32 elements at a time
  - Thus grid size = 16 blocks
  - Block is analogous to a strip-mined vector loop with vector length of 32
  - Block is assigned to a *multithreaded SIMD processor* by the *thread block scheduler*
  - Fermi GPUs have 7-15 multithreaded SIMD processors

Example

- NVIDIA GPU has 32,768 registers
  - Divided into lanes
  - Each SIMD thread is limited to 64 registers
  - SIMD thread has up to:
    - 64 vector registers of 32 32-bit elements
    - 32 vector registers of 32 64-bit elements
  - Fermi has 16 physical SIMD lanes, each containing 2048 registers

GPUs Continue to Advance



NVIDIA Instruction Set Arch.

- ISA is an abstraction of the hardware instruction set
  - "Parallel Thread Execution (PTX)"
  - Uses virtual registers
  - Translation to machine code is performed in software
  - Example:

```
shl.s32       R8, blockIdx, 9    ; Thread Block ID * Block size (512 or 2^9)
add.s32       R8, R8, threadIdx  ; R8 = i = my CUDA thread ID
ld.global.f64 RD0, [X+R8]        ; RD0 = X[i]
ld.global.f64 RD2, [Y+R8]        ; RD2 = Y[i]
mul.f64 R0D, RD0, RD4            ; Product in RD0 = RD0 * RD4 (scalar a)
add.f64 R0D, RD0, RD2            ; Sum in RD0 = RD0 + RD2 (Y[i])
st.global.f64 [Y+R8], RD0        ; Y[i] = sum (X[i]*a + Y[i])
```

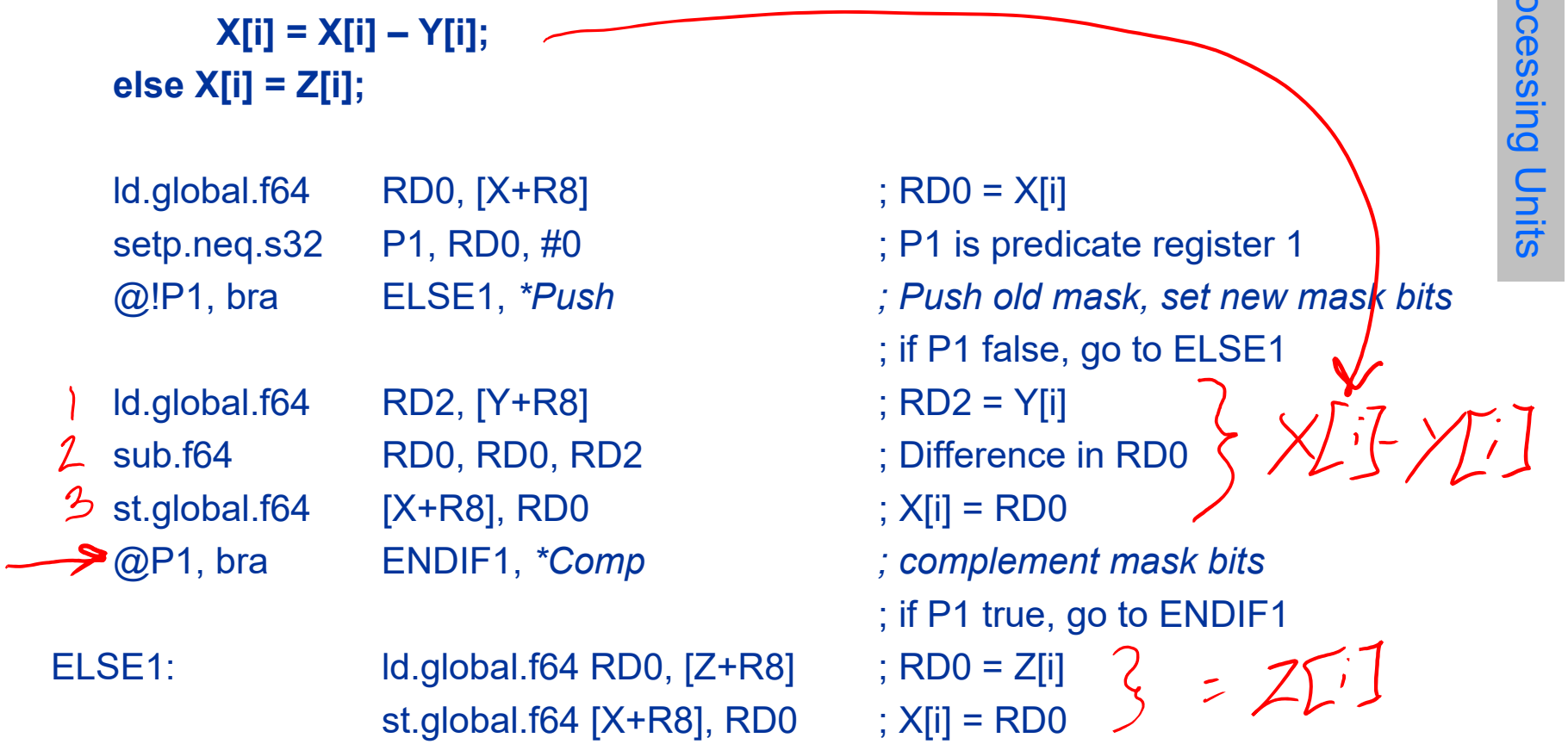
Conditional Branching

- Like vector architectures, GPU branch hardware uses internal masks
- Also uses
  - Branch synchronization stack
    - Entries consist of masks for each SIMD lane
    - I.e. which threads commit their results (all threads execute)
  - Instruction markers to manage when a branch diverges into multiple execution paths
    - Push on divergent branch
  - …and when paths converge
    - Act as barriers
    - Pops stack
- Per-thread-lane 1-bit predicate register, specified by programmer

Example


```
if (X[i] != 0)
        X[i] = X[i] – Y[i];
else X[i] = Z[i];

    ld.global.f64    RD0, [X+R8]          ; RD0 = X[i]
    setp.neq.s32     P1, RD0, #0          ; P1 is predicate register 1
    @!P1, bra        ELSE1, *Push         ; Push old mask, set new mask bits
                                          ; if P1 false, go to ELSE1
    ld.global.f64    RD2, [Y+R8]          ; RD2 = Y[i]
    sub.f64          RD0, RD0, RD2        ; Difference in RD0
    st.global.f64    [X+R8], RD0          ; X[i] = RD0
    @P1, bra         ENDIF1, *Comp        ; complement mask bits
                                          ; if P1 true, go to ENDIF1
ELSE1:              ld.global.f64 RD0, [Z+R8]   ; RD0 = Z[i]
                    st.global.f64 [X+R8], RD0   ; X[i] = RD0
ENDIF1:  <next instruction>, *Pop         ; pop to restore old mask
```



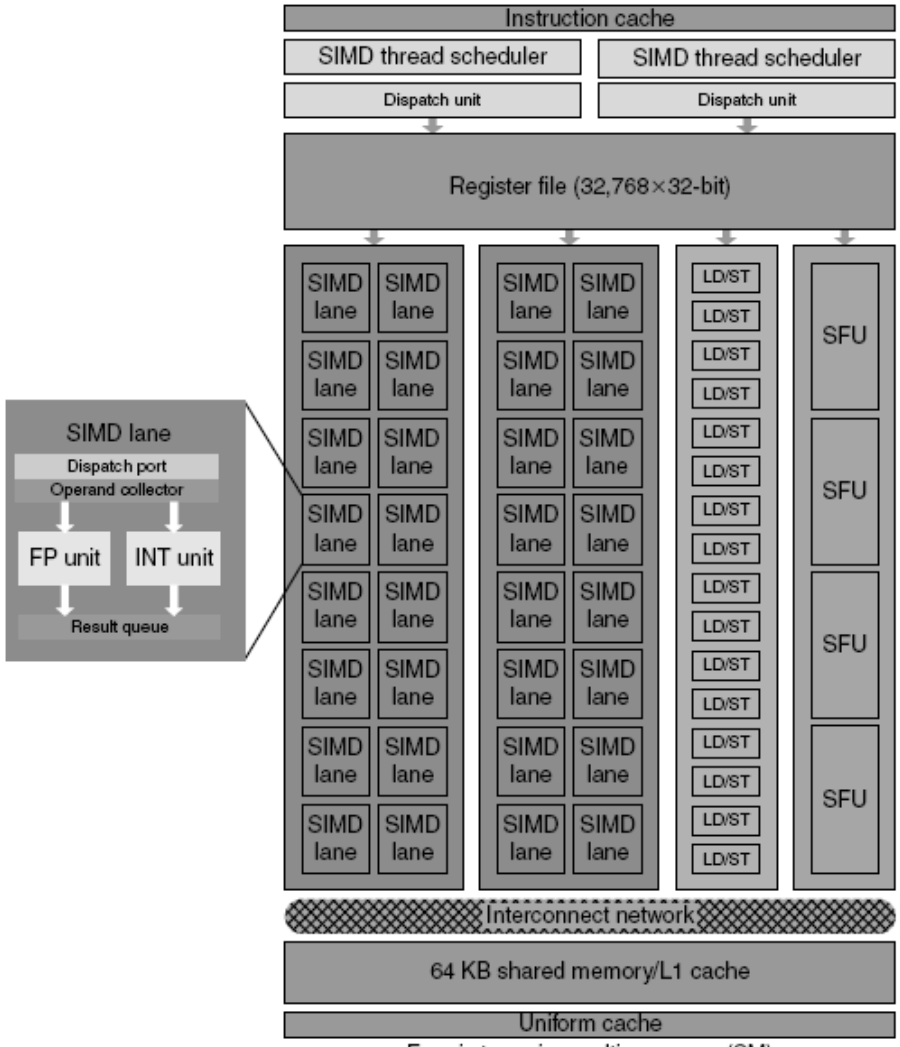
NVIDIA GPU Memory Structures

- Each SIMD Lane has private section of off-chip DRAM
  - "Private memory"
  - Contains stack frame, spilling registers, and private variables
- Each multithreaded SIMD processor also has local memory
  - Shared by SIMD lanes / threads within a block
- Memory shared by SIMD processors is GPU Memory
  - Host can read and write GPU memory

Fermi Architecture Innovations

- Each SIMD processor has
  - Two SIMD thread schedulers, two instruction dispatch units
  - 16 SIMD lanes (SIMD width=32, chime=2 cycles), 16 load-store units, 4 special function units
  - Thus, two threads of SIMD instructions are scheduled every two clock cycles
- Fast double precision
- Caches for GPU memory
- 64-bit addressing and unified address space
- Error correcting codes
- Faster context switching
- Faster atomic instructions

Fermi Multithreaded SIMD Proc.



Fermi streaming multiprocessor (SM)

Loop-Level Parallelism

- Focuses on determining whether data accesses in later iterations are dependent on data values produced in earlier iterations
  - Loop-carried dependence

- Example 1:

```
for (i=999; i>=0; i=i-1)
        x[i] = x[i] + s;
```

- No loop-carried dependence

Loop-Level Parallelism

- Example 2:

```
for (i=0; i<100; i=i+1) {
        A[i+1] = A[i] + C[i]; /* S1 */
        B[i+1] = B[i] + A[i+1]; /* S2 */
}
```

- S1 and S2 use values computed by S1 in previous iteration
- S2 uses value computed by S1 in same iteration

Loop-Level Parallelism

- Example 3:

```
for (i=0; i<100; i=i+1) {
        A[i] = A[i] + B[i]; /* S1 */
        B[i+1] = C[i] + D[i]; /* S2 */
}
```

- S1 uses value computed by S2 in previous iteration but dependence is not circular so loop is parallel
- Transform to:

```
A[0] = A[0] + B[0];
for (i=0; i<99; i=i+1) {
        B[i+1] = C[i] + D[i];
        A[i+1] = A[i+1] + B[i+1];
}
B[100] = C[99] + D[99];
```