ELE 655
Microprocessor System Design

Section 3 – Data Level Parallelism

Class 3 – GPU

DLP
Data Level Parallelism

- GPU overview
  - Massively parallel HW solution to massively parallel Data problems
  - Originated as co-processors for Graphics applications
  - Converged with other SIMD platforms as the instructions set and hardware became more capable of control operations
  - Cheapest path to huge numbers of parallel processors

DLP

Data Level Parallelism

- GPU Basics
  - Thread
    - In normal computing terms this is a series of instructions that can be run independently
    - In GPU world – sequence of instructions that can be independently executed in a single SIMD lane
    - Each with its own PC and registers
    - E.g. – one iteration of the body of a parallelized loop

DLP

Data Level Parallelism

- GPU Basics
  - SIMD Thread (CUDA Thread)
    - A group of 32 threads
    - Often just referred to as a thread
    - Eg. Thread 14, instruction 12

DLP

Data Level Parallelism

- GPU Basics
  - Thread Block
    - A group of 16 SIMD Threads
    - Can communicate between threads via local memory

DLP

Data Level Parallelism

- GPU Basics
  - Grid
    - A group of 16 Thread Blocks

DLP
Data Level Parallelism

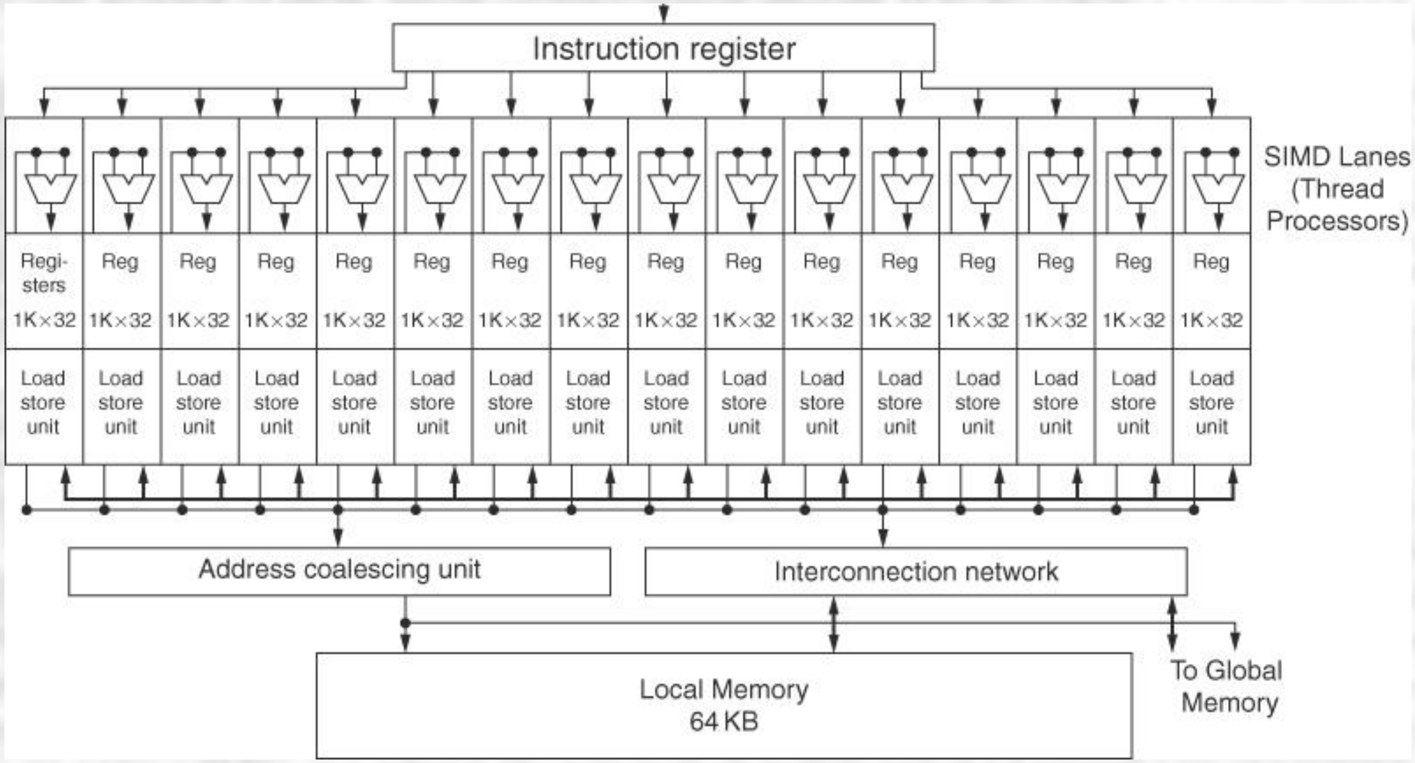
- GPU Basics
  - Example
    - Vector-vector multiply with 8192 elements / vector
    - Fully independent
    - SIMD thread – 32 independent elements
    - Thread block – 16 SIMD threads
      → 512 elements / thread block
    - 8192/512 = 16 Thread blocks = 1 Grid

DLP
Data Level Parallelism

- GPU Basics
  - Multi-threaded SIMD processor
    - Multi-lane processor (16, 32 lanes)
    - Load/Store, FPU
    - Operates on 1 Thread Block
    - Large register file
      - 16 SIMD Threads/Block x 32 threads/SIMD Thread x 64 registers thread = 32,768 32-bit registers
    - Local memory

DLP
Data Level Parallelism

- GPU Basics
  - Multi-threaded SIMD processor – 16 lanes



DLP
Data Level Parallelism

- GPU Basics
  - Operation
    - Grid is created by the compiler
    - Thread Block Scheduler
      - Determines how many thread blocks are needed
      - Assigns thread blocks from the grid to multithreaded SIMD processors
      - Continues until all blocks are assigned
    - Thread Scheduler (in each multithreaded SIMD processor)
      - Warp Scheduler in Nvidia
      - Selects the next thread (instruction) to execute
      - Based on data and resource availability
      - Dynamic Scheduling

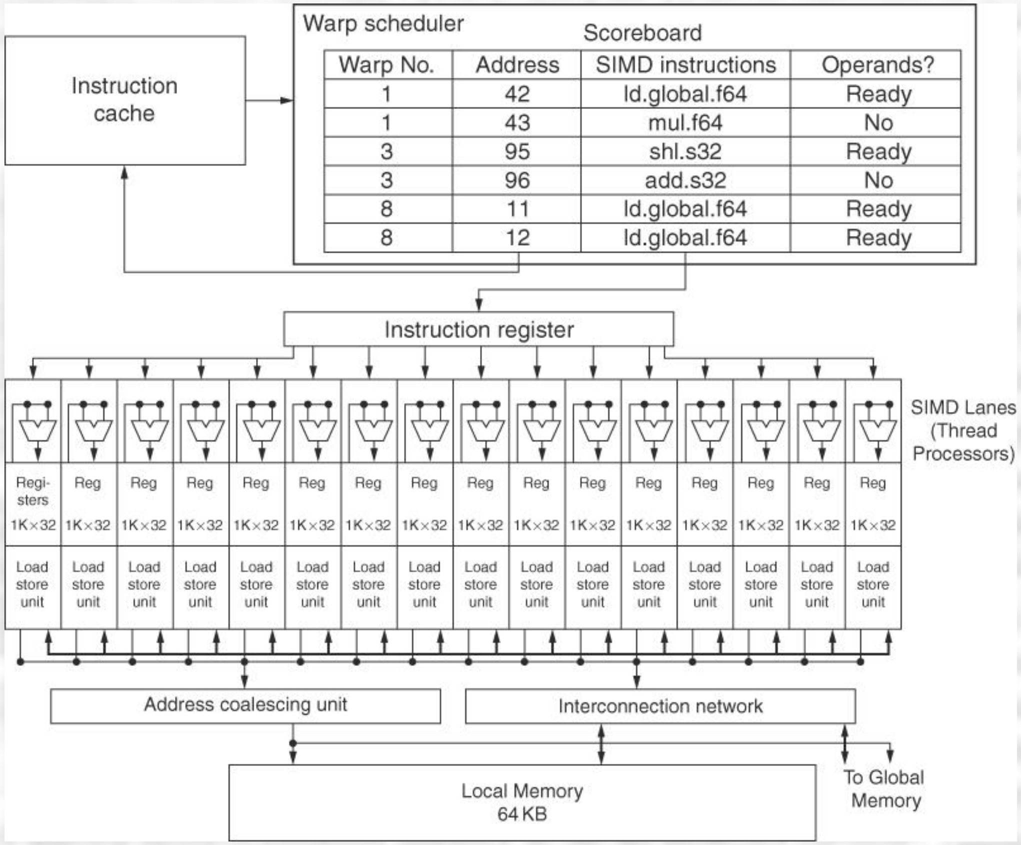
DLP

Data Level Parallelism

- GPU Basics
  - Operation
    - Execution
      - Each multithreaded SIMD processor must
        - Load 32 elements of each of 2 vectors
        - Perform the required operation
        - Store 32 elements of the result

DLP
Data Level Parallelism

- GPU Basics
  - Operation

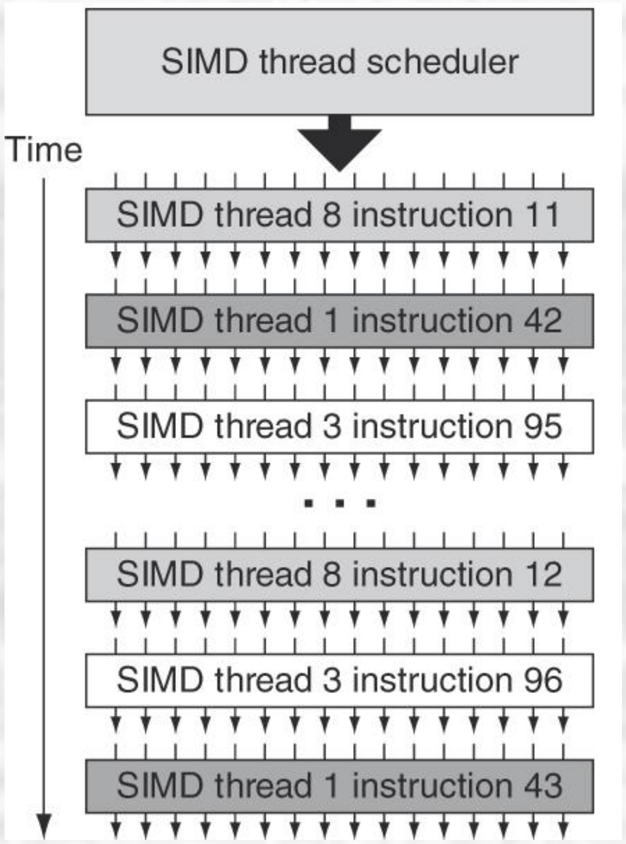


Warp No.	Address	SIMD instructions	Operands?
1	42	ld.global.f64	Ready
1	43	mul.f64	No
3	95	shl.s32	Ready
3	96	add.s32	No
8	11	ld.global.f64	Ready
8	12	ld.global.f64	Ready

DLP

Data Level Parallelism

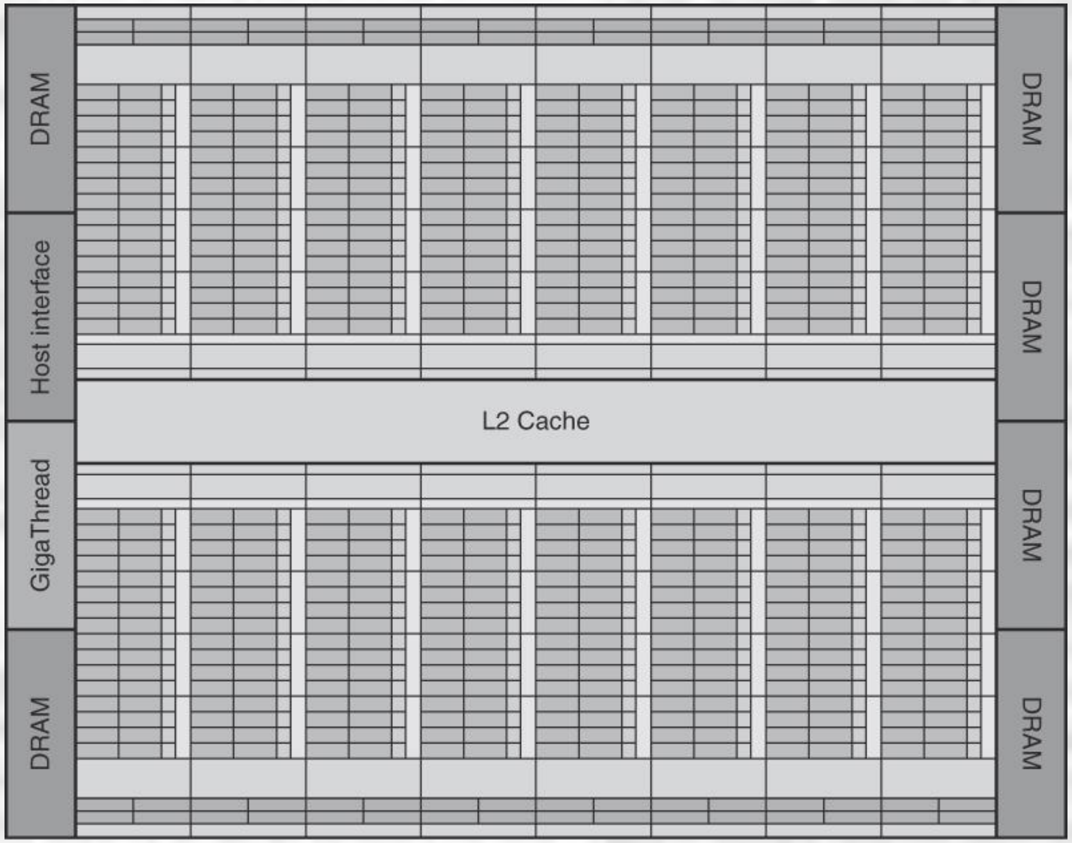
- GPU Basics
  - Operation


SIMD thread scheduler
Time
SIMD thread 8 instruction 11
SIMD thread 1 instruction 42
SIMD thread 3 instruction 95
SIMD thread 8 instruction 12
SIMD thread 3 instruction 96
SIMD thread 1 instruction 43

DLP

Data Level Parallelism

- GPU Basics
  - Implementation
  - Nvidia – Fermi GTX 480
    - 16 SIMD processors
    - 16 SIMD Lanes
    - Local and Global Memory
    - Thread Block Scheduler
      - GigaThread



DLP

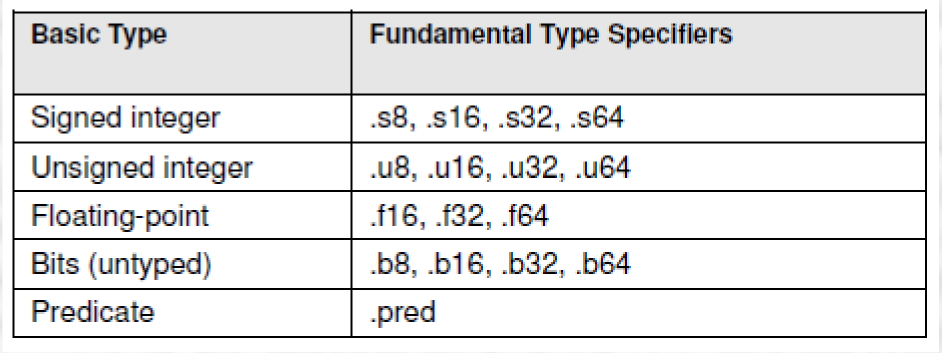
Data Level Parallelism

- GPU ISA

  - Instruction set is an abstraction instead of direct implementation
    - PTX – Parallel Thread Execution
    - Simplifies compiler compatibility across HW variations
    - Usually map 1-1 with the HW instructions
    - Can represent multiple HW instructions in 1 PTX instruction
    - Uses virtual registers
      - Compiler assigns resources

    - Format
      - Opcode.type d,a,b,d
      - 8/16/32/64 bit operands

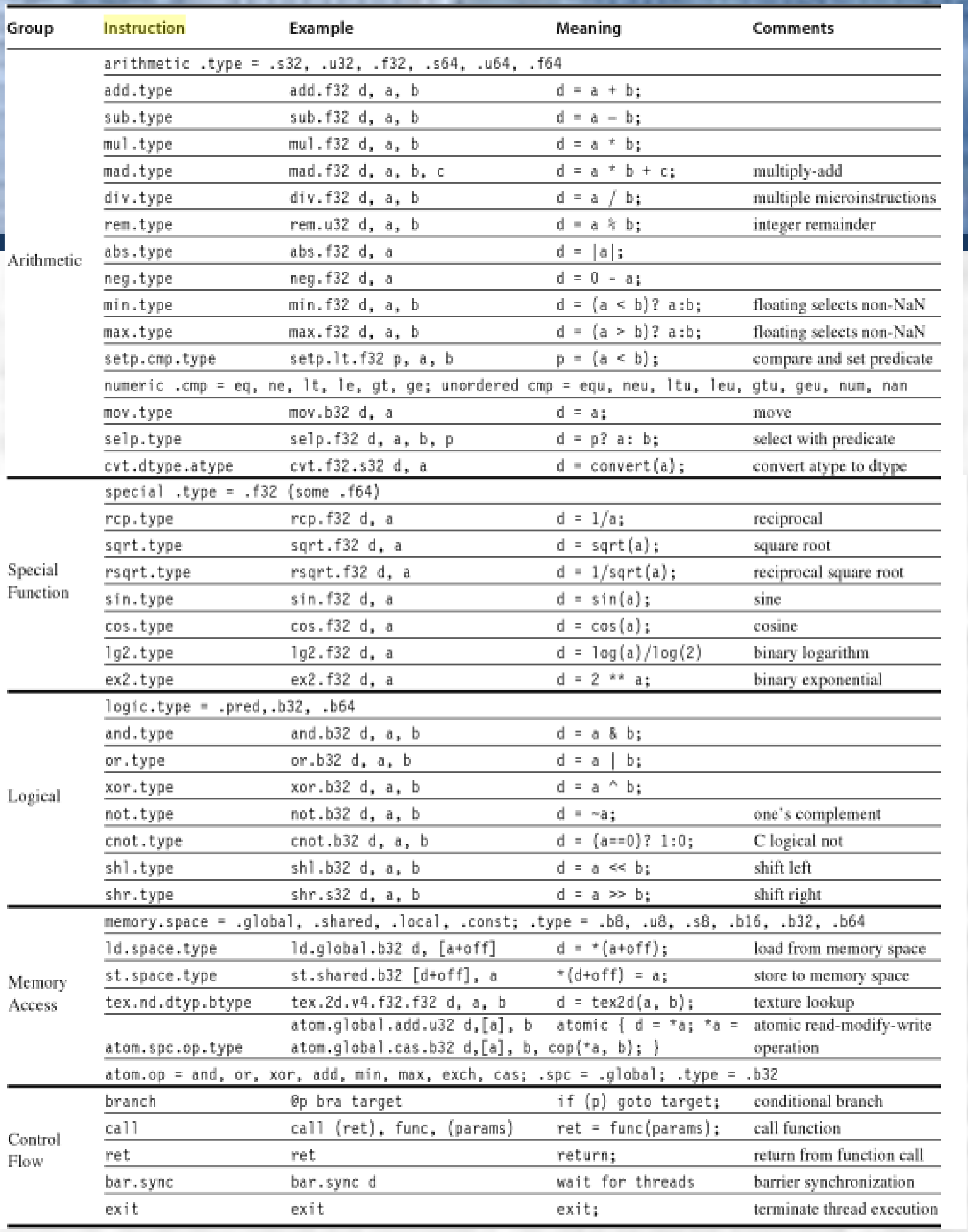
      - 1 bit predicate registers



Basic Type	Fundamental Type Specifiers
Signed integer	.s8, .s16, .s32, .s64
Unsigned integer	.u8, .u16, .u32, .u64
Floating-point	.f16, .f32, .f64
Bits (untyped)	.b8, .b16, .b32, .b64
Predicate	.pred

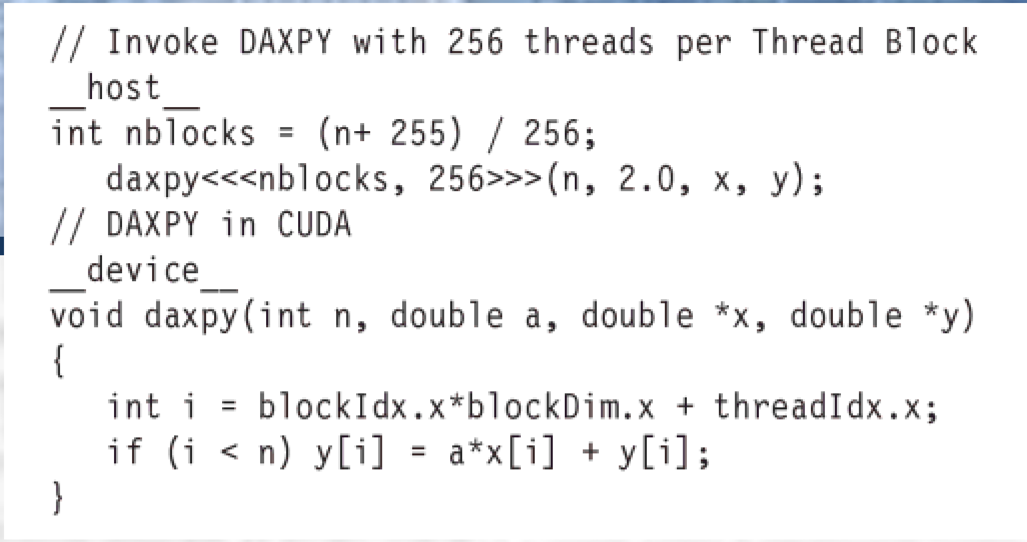
DLP
Data Level Parallelism

- GPU ISA



Group	Instruction	Example	Meaning	Comments
Arithmetic	arithmetic .type = .s32, .u32, .f32, .s64, .u64, .f64			
	add.type	add.f32 d, a, b	d = a + b;	
	sub.type	sub.f32 d, a, b	d = a – b;	
	mul.type	mul.f32 d, a, b	d = a * b;	
	mad.type	mad.f32 d, a, b, c	d = a * b + c;	multiply-add
	div.type	div.f32 d, a, b	d = a / b;	multiple microinstructions
	rem.type	rem.u32 d, a, b	d = a % b;	integer remainder
	abs.type	abs.f32 d, a	d = |a|;	
	neg.type	neg.f32 d, a	d = 0 - a;	
	min.type	min.f32 d, a, b	d = (a < b)? a:b;	floating selects non-NaN
	max.type	max.f32 d, a, b	d = (a > b)? a:b;	floating selects non-NaN
	setp.cmp.type	setp.lt.f32 p, a, b	p = (a < b);	compare and set predicate
	numeric .cmp = eq, ne, lt, le, gt, ge; unordered cmp = equ, neu, ltu, leu, gtu, geu, num, nan			
	mov.type	mov.b32 d, a	d = a;	move
	selp.type	selp.f32 d, a, b, p	d = p? a: b;	select with predicate
	cvt.dtype.atype	cvt.f32.s32 d, a	d = convert(a);	convert atype to dtype
Special Function	special .type = .f32 (some .f64)			
	rcp.type	rcp.f32 d, a	d = 1/a;	reciprocal
	sqrt.type	sqrt.f32 d, a	d = sqrt(a);	square root
	rsqrt.type	rsqrt.f32 d, a	d = 1/sqrt(a);	reciprocal square root
	sin.type	sin.f32 d, a	d = sin(a);	sine
	cos.type	cos.f32 d, a	d = cos(a);	cosine
	lg2.type	lg2.f32 d, a	d = log(a)/log(2)	binary logarithm
	ex2.type	ex2.f32 d, a	d = 2 ** a;	binary exponential
Logical	logic.type = .pred,.b32, .b64			
	and.type	and.b32 d, a, b	d = a & b;	
	or.type	or.b32 d, a, b	d = a | b;	
	xor.type	xor.b32 d, a, b	d = a ^ b;	
	not.type	not.b32 d, a, b	d = ~a;	one's complement
	cnot.type	cnot.b32 d, a, b	d = (a==0)? 1:0;	C logical not
	shl.type	shl.b32 d, a, b	d = a << b;	shift left
	shr.type	shr.s32 d, a, b	d = a >> b;	shift right
Memory Access	memory.space = .global, .shared, .local, .const; .type = .b8, .u8, .s8, .b16, .b32, .b64			
	ld.space.type	ld.global.b32 d, [a+off]	d = *(a+off);	load from memory space
	st.space.type	st.shared.b32 [d+off], a	*(d+off) = a;	store to memory space
	tex.nd.dtyp.btype	tex.2d.v4.f32.f32 d, a, b	d = tex2d(a, b);	texture lookup
	atom.spc.op.type	atom.global.add.u32 d,[a], b atom.global.cas.b32 d,[a], b, c	atomic { d = *a; *a = op(*a, b); }	atomic read-modify-write operation
	atom.op = and, or, xor, add, min, max, exch, cas; .spc = .global; .type = .b32			
Control Flow	branch	@p bra target	if (p) goto target;	conditional branch
	call	call (ret), func, (params)	ret = func(params);	call function
	ret	ret	return;	return from function call
	bar.sync	bar.sync d	wait for threads	barrier synchronization
	exit	exit	exit;	terminate thread execution

DLP
Data Level Parallelism



```
// Invoke DAXPY with 256 threads per Thread Block
__host__
int nblocks = (n+ 255) / 256;
   daxpy<<<nblocks, 256>>>(n, 2.0, x, y);
// DAXPY in CUDA
__device__
void daxpy(int n, double a, double *x, double *y)
{
   int i = blockIdx.x*blockDim.x + threadIdx.x;
   if (i < n) y[i] = a*x[i] + y[i];
}
```

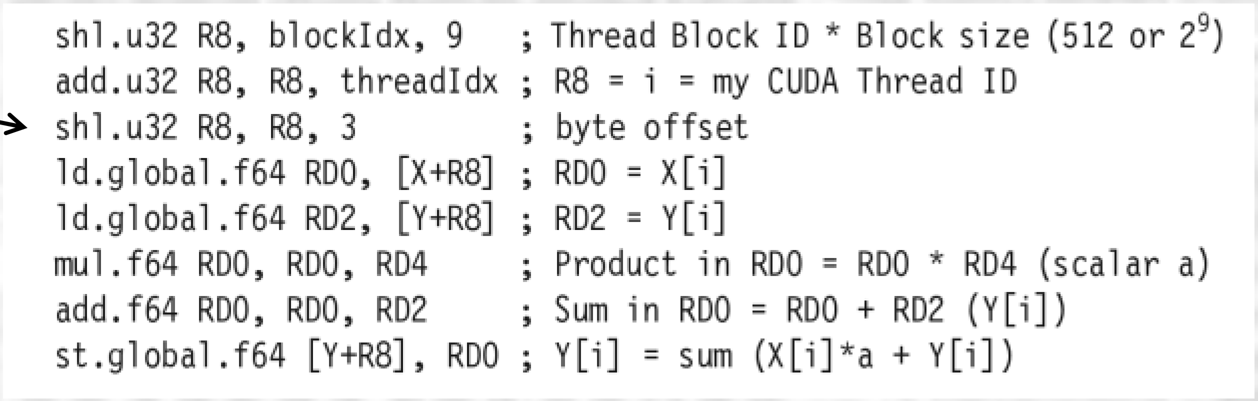
- GPU ISA
  - CUDA example DAXPY
    - Format for a function call
      - Name <<< dimGrid, dimBlock>>> (… param list…)
    - Identifiers
      - Each block has an identifier – blockIdx
      - Each thread has an identifier inside the block – threadIdx
      - BlockDim = dimBlock

DLP

Data Level Parallelism

- GPU ISA
  - PTX example DAXPY – 1 thread

R8 now points to my thread ID in memory relative to the base address



```
shl.u32 R8, blockIdx, 9      ; Thread Block ID * Block size (512 or 2^9)
add.u32 R8, R8, threadIdx ; R8 = i = my CUDA Thread ID
shl.u32 R8, R8, 3            ; byte offset
ld.global.f64 RD0, [X+R8] ; RD0 = X[i]
ld.global.f64 RD2, [Y+R8] ; RD2 = Y[i]
mul.f64 RD0, RD0, RD4        ; Product in RD0 = RD0 * RD4 (scalar a)
add.f64 RD0, RD0, RD2        ; Sum in RD0 = RD0 + RD2 (Y[i])
st.global.f64 [Y+R8], RD0 ; Y[i] = sum (X[i]*a + Y[i])
```

DLP

Data Level Parallelism

- GPU ISA
  - Conditional Branching
    - Each Lane as its own mask bit to determine whether to execute the current instruction or not
    - Each Thread has its own stack to keep track of branch return addresses
    - Assembler optimizes
      - Use branch instructions for complex situations
        - Branch diverges when only some lanes branch
      - Use predicates for simple situations
        - Only lanes with predicate=1 execute

DLP

Data Level Parallelism

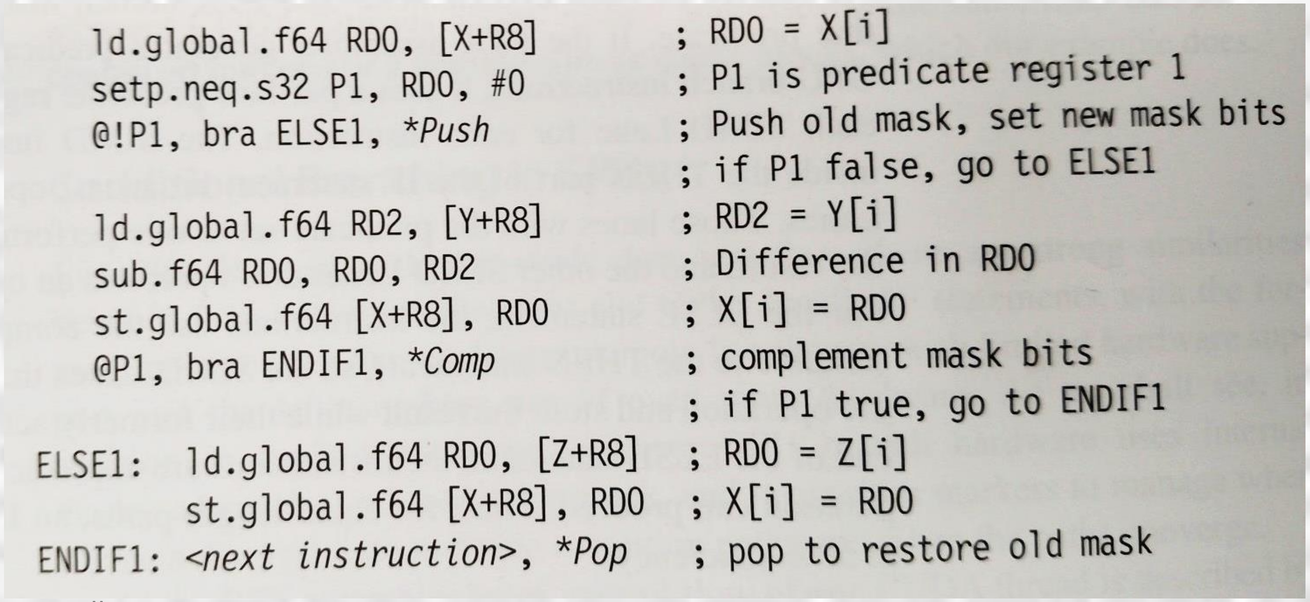
- GPU ISA
  - Conditional Branching
    - Regardless of how the branch is handled – all lanes stay synchronized to the same instructions
    - This leads to potential inefficiency when few lanes are actually executing

DLP

Data Level Parallelism

- GPU ISA
  - Conditional Branching

```
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
```

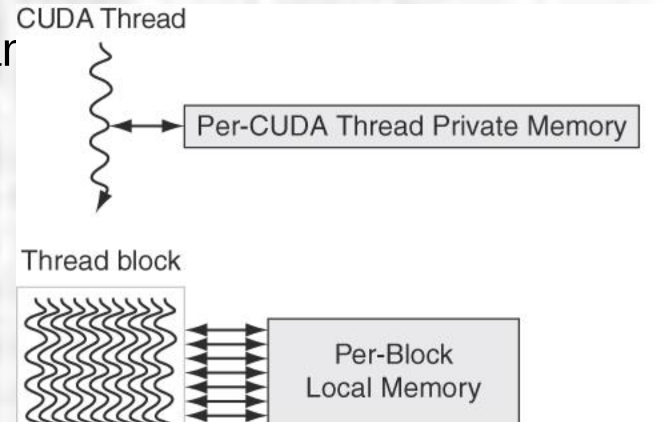
DLP
Data Level Parallelism

- GPU Memory
  - Each thread (lane) has a private memory block
  - Private Memory
    - Off chip
    - Not shared with anyone
    - Holds stack, private variables, …
    - Can be cached to speed up access
  - Each Multithreaded SIMD processor has local memory
  - Local Memory
    - On chip
    - Shared by SIMD lanes
    - Not shared across multithreaded SIMD processors
    - Allocates portions of Local Memory to each thread block
      - Private to a thread block

DLP

Data Level Parallelism

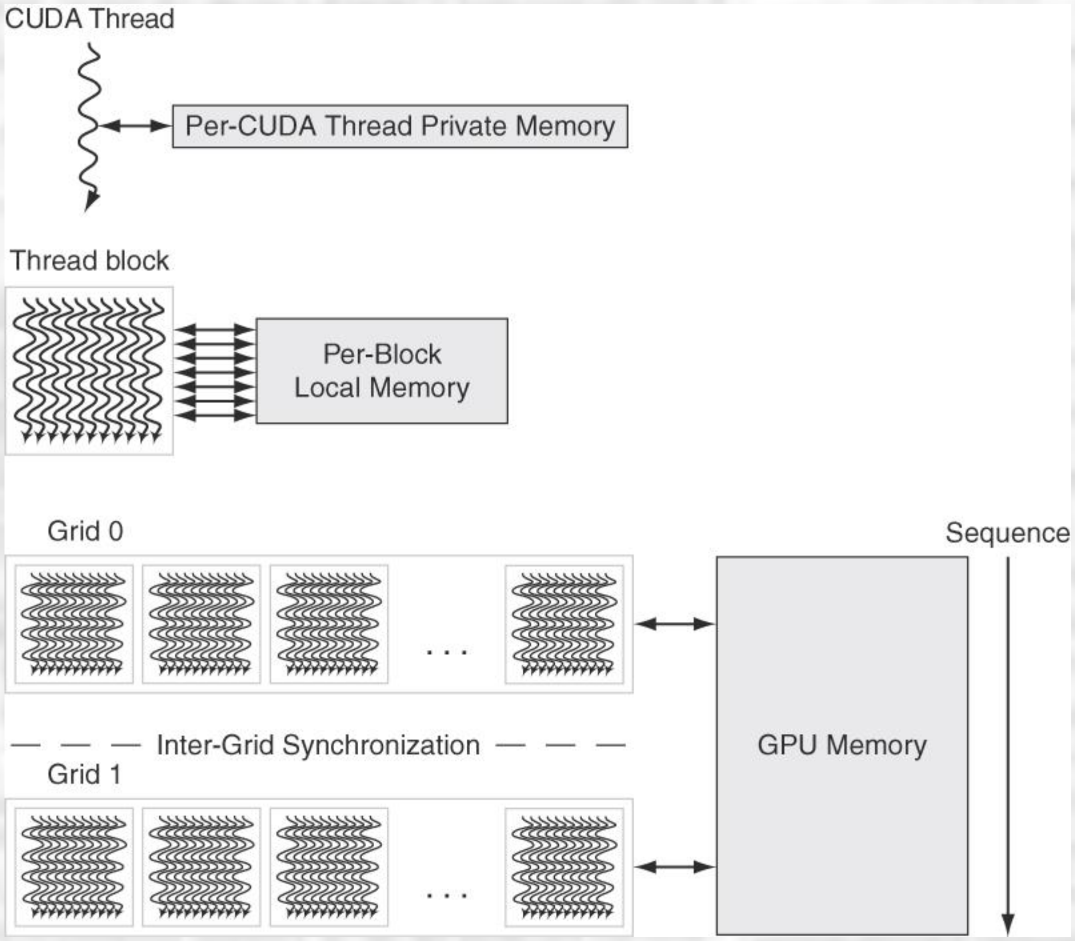
- GPU Memory
  - External memory available to the whole GPU
  - GPU memory
    - Available to the Host for R/W
    - Pipelined accesses by the GPU
    - Latency hidden by multithreading
  - Special memory hardware
    - Coalesce memory accesses from individual threads in a SIMD thread into a single pipelined access
    - Hold some requests to group requests to the sar



DLP

Data Level Parallelism

- GPU Memory



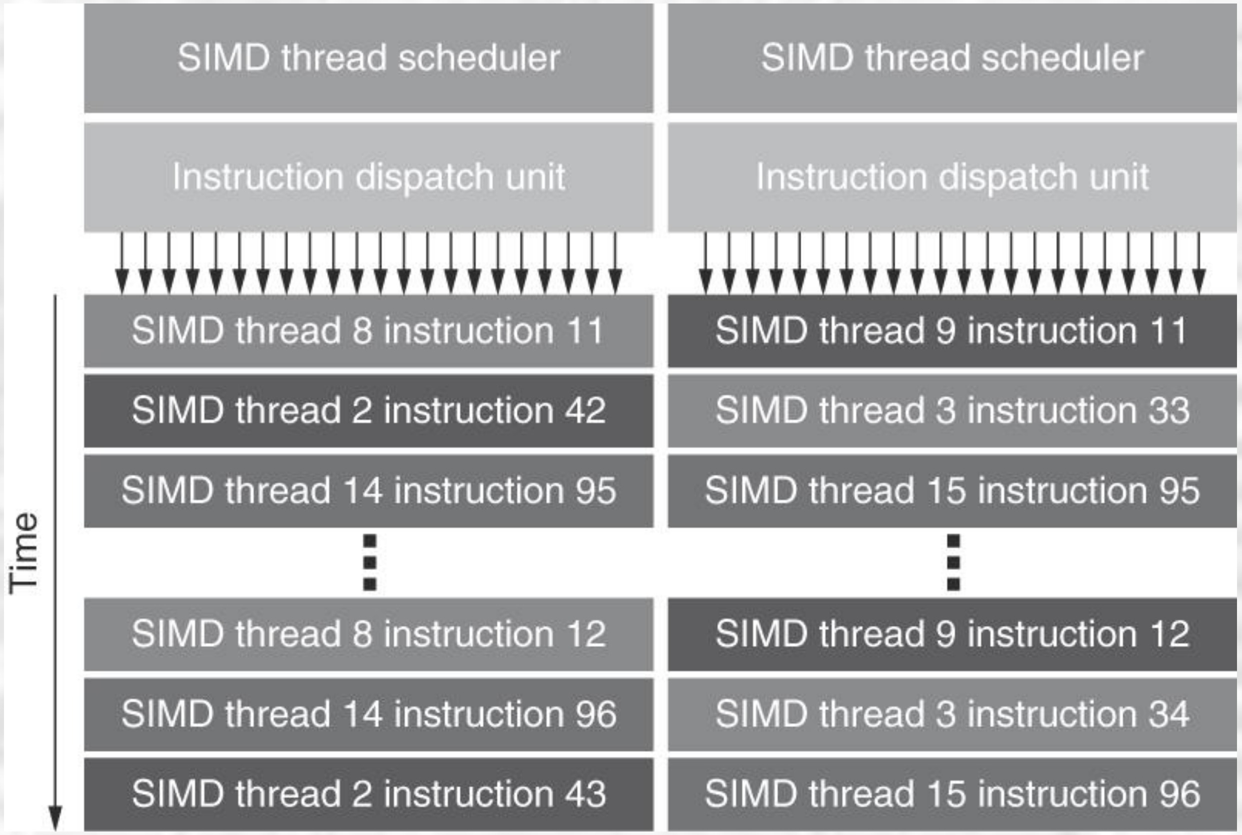
DLP
Data Level Parallelism

- FERMI – multithreaded SIMD processor core
  - Dual scheduler – dual dispatch
  - 2 sets of 16 Lanes
  - 16 L/S units
  - 4 Special Function Units (SFU)
  - Looks a little like a superscalar with 2 ALU, 1 L/S and 1 SFU
    - But here we are executing 32 threads in each of 2 of the execution units in 2 clock cycles **

DLP

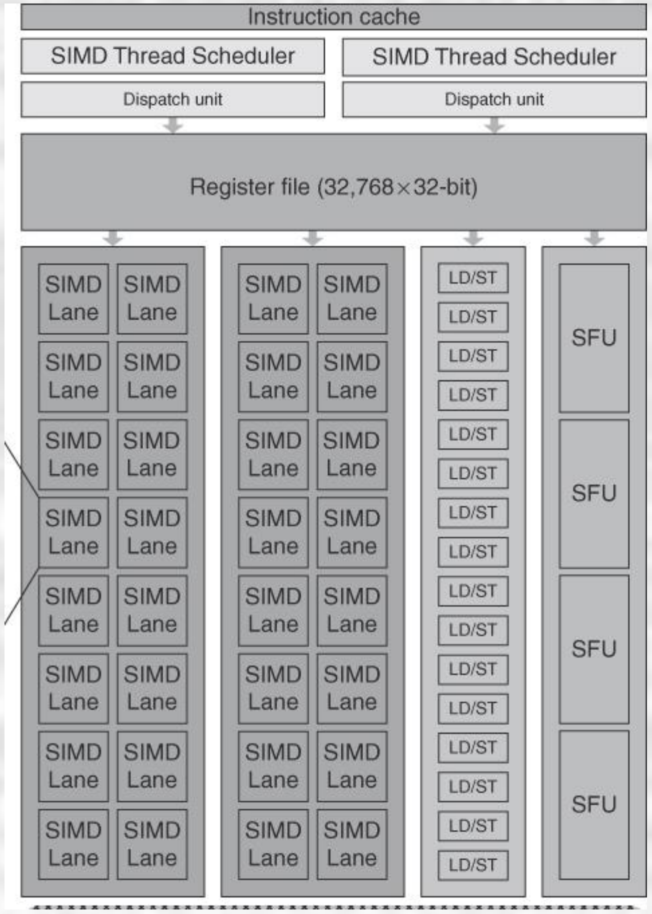
Data Level Parallelism

- FERMI – multithreaded SIMD processor core


SIMD thread scheduler
SIMD thread scheduler
Instruction dispatch unit
Instruction dispatch unit
SIMD thread 8 instruction 11
SIMD thread 9 instruction 11
SIMD thread 2 instruction 42
SIMD thread 3 instruction 33
SIMD thread 14 instruction 95
SIMD thread 15 instruction 95
SIMD thread 8 instruction 12
SIMD thread 9 instruction 12
SIMD thread 14 instruction 96
SIMD thread 3 instruction 34
SIMD thread 2 instruction 43
SIMD thread 15 instruction 96
Time

DLP
Data Level Parallelism

- FERMI – multithreaded SIMD processor core

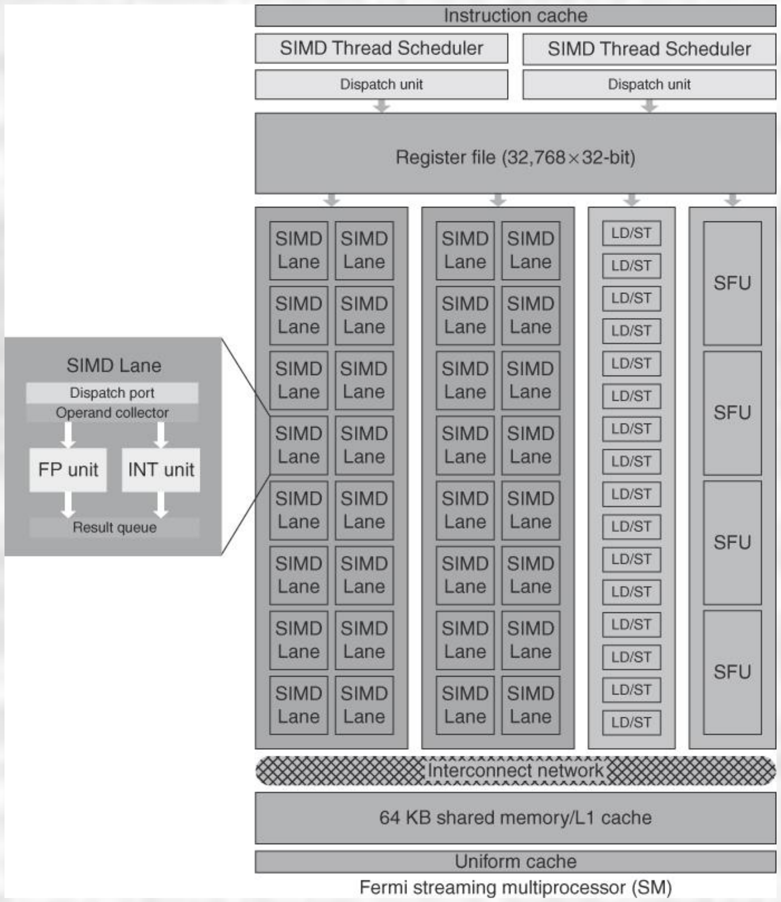


DLP
Data Level Parallelism

- FERMI – multithreaded SIMD processor core
  - Fast Double-Precision Floating Point
    - 2x single precision
  - Caches for GPU memory
    - L1 Data and Memory
      - Memory array is shared with Local Memory
      - Split is programmable – 16KB/48KB or 48KB/16KB
    - L2 Unified
      - 768KB
  - 64 bit addressing
    - All memories

DLP
Data Level Parallelism

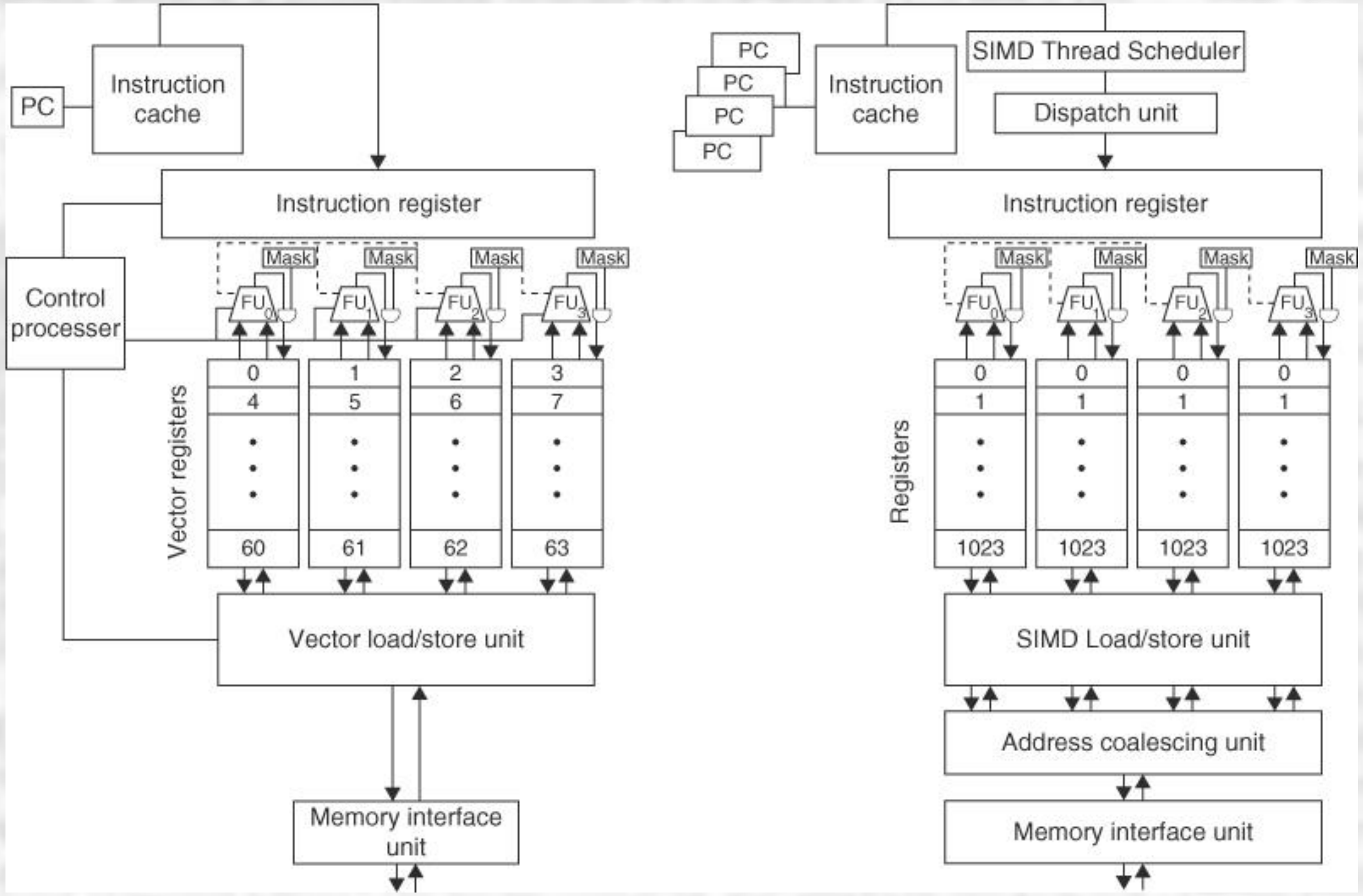
- FERMI – multithreaded SIMD processor core



DLP

Data Level Parallelism

- GPU Vs. Vector Processor


PC
Instruction cache
Instruction register
Control processer
Mask
FU 0
FU 1
FU 2
FU 3
Vector registers
0 4 60
1 5 61
2 6 62
3 7 63
Vector load/store unit
Memory interface unit
PC
PC
PC
PC
Instruction cache
SIMD Thread Scheduler
Dispatch unit
Instruction register
Registers
0 1 1023
SIMD Load/store unit
Address coalescing unit
Memory interface unit