

Advanced Computer Architecture

Lecture 12

Exploiting DLP: Vector & GPU Architectures



NPTEL Online Course : Advanced Computer Architecture : Dr. John Jose, IIT Guwahati

Dr. John Jose

Assistant Professor

**Department of Computer Science & Engineering
Indian Institute of Technology Guwahati, Assam.**

SIMD Architectures

- ❖ **SIMD architectures exploit data-level parallelism**

- ❖ matrix-oriented scientific computing

- ❖ media-oriented image and sound processing

- ❖ **SIMD is more energy efficient than MIMD**

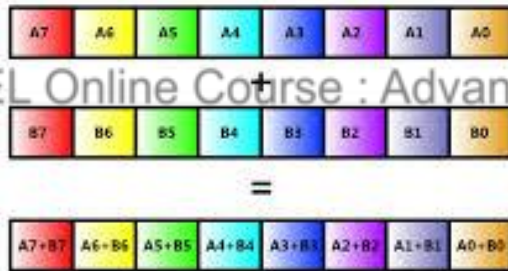
- ❖ Only needs to fetch one instruction per data operation

- ❖ Makes SIMD attractive for personal mobile devices

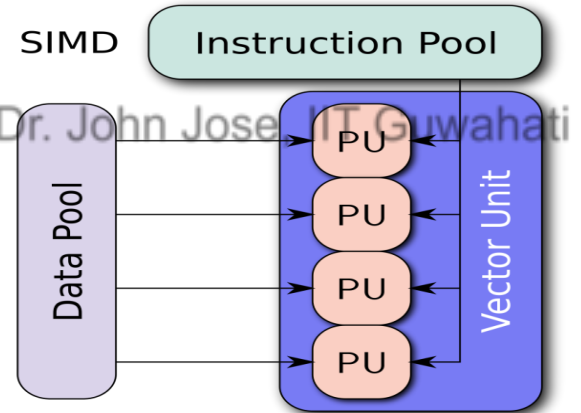
SIMD Architectures

- ❖ Vector Processors (VPs)
- ❖ Graphics Processor Units (GPUs)

SIMD Mode



Scalar Mode



Vector Architectures

- ❖ Read set of data elements into **vector registers**
- ❖ Operate on **vector registers**
- ❖ Disperse the results back into memory
- ❖ Single instruction operate on a vectors of data
- ❖ Vector load and store are deeply pipelined
- ❖ High memory latency once per load & store
- ❖ Amortize the latency over load size

Introduction to VMIPS Architecture

❖ Vector registers

- ❖ 8 vector registers; each register holds 64-elements
- ❖ 64 bits/vector element **[1 VR=512 Bytes]**
- ❖ Register file has 16 read ports and 8 write ports

NPTL Online Course: Advanced Computer Architecture : Dr. John Jose, IIT Guwahati

❖ Scalar registers

- ❖ 32 GPRs, 32 FPRs : Used by VFU / VLSU

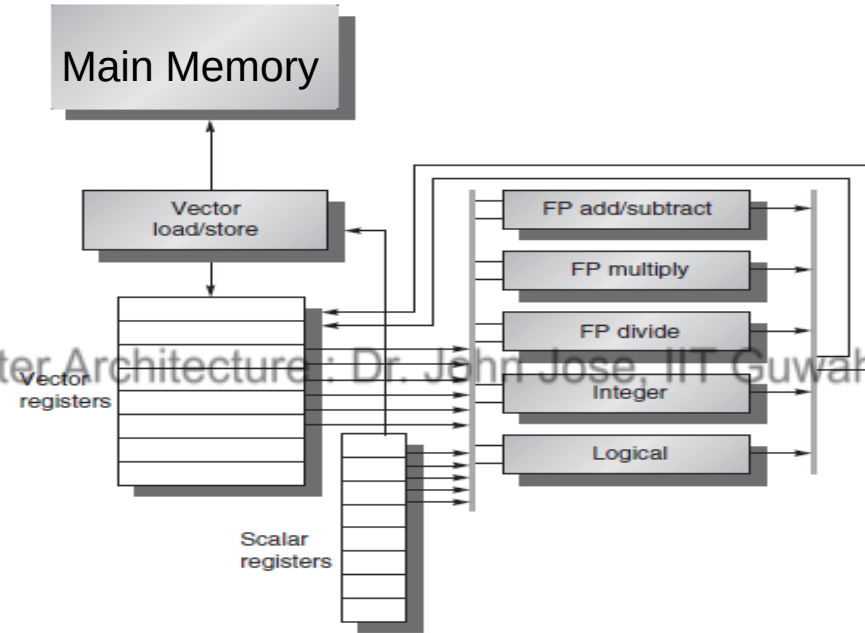
❖ Vector Functional Units **[VFU]**

- ❖ Fully pipelined, can start new operation on every cycle
- ❖ Capability for data and control hazards detection

❖ Vector Load-Store Unit **[VLSU]**

Introduction to VMIPS Architecture

- ❖ Vector registers
- ❖ Scalar registers
- ❖ Vector Functional Units
- ❖ Vector Load-Store Unit



Program with VMIPS Instructions

- ❖ ADDVV.D: add two vectors
- ❖ ADDVS.D: add vector to a scalar
- ❖ LV/SV: vector load and vector store from address

L.D F0,a ; load scalar a

LV V1,Rx ; load vector X

MULVS.D V2,V1,F0 ; vector-scalar multiply

LV V3,Ry ; load vector Y

ADDVV V4,V2,V3 ; add

SV Ry,V4 ; store the result

- ❖ Requires 6 instructions vs. almost 600 for MIPS

VMIPS Instruction Set

Instruction	Operands	Function
ADDVV.D	V1, V2, V3	Add elements of V2 and V3, then put each result in V1.
ADDVS.D	V1, V2, F0	Add F0 to each element of V2, then put each result in V1.
SUBVV.D	V1, V2, V3	Subtract elements of V3 from V2, then put each result in V1.
SUBVS.D	V1, V2, F0	Subtract F0 from elements of V2, then put each result in V1.
SUBSV.D	V1, F0, V2	Subtract elements of V2 from F0, then put each result in V1.
MULVV.D	V1, V2, V3	Multiply elements of V2 and V3, then put each result in V1.
MULVS.D	V1, V2, F0	Multiply each element of V2 by F0, then put each result in V1.
DIVVV.D	V1, V2, V3	Divide elements of V2 by V3, then put each result in V1.
DIVVS.D	V1, V2, F0	Divide elements of V2 by F0, then put each result in V1.
DIVSV.D	V1, F0, V2	Divide F0 by elements of V2, then put each result in V1.
LV	V1, R1	Load vector register V1 from memory starting at address R1.
SV	R1, V1	Store vector register V1 into memory starting at address R1.
LVWS	V1, (R1, R2)	Load V1 from address at R1 with stride in R2 (i.e., $R1 + i \times R2$).
SVWS	(R1, R2), V1	Store V1 to address at R1 with stride in R2 (i.e., $R1 + i \times R2$).
LVI	V1, (R1+V2)	Load V1 with vector whose elements are at $R1 + V2(i)$ (i.e., V2 is an index).
SVI	(R1+V2), V1	Store V1 to vector whose elements are at $R1 + V2(i)$ (i.e., V2 is an index).
CVI	V1, R1	Create an index vector by storing the values $0, 1 \times R1, 2 \times R1, \dots, 63 \times R1$ into V1.
S--VV.D	V1, V2	Compare the elements (EQ, NE, GT, LT, GE, LE) in V1 and V2. If condition is true, put a 1 in the corresponding bit vector; otherwise put 0. Put resulting bit vector in vector-mask register (VM). The instruction S--VS.D performs the same compare but using a scalar value as one operand.
S--VS.D	V1, F0	
POP	R1, VM	Count the 1s in vector-mask register VM and store count in R1.
CVM		Set the vector-mask register to all 1s.
MTC1	VLR, R1	Move contents of R1 to vector-length register VL.
MFC1	R1, VLR	Move the contents of vector-length register VL to R1.
MVTM	VM, F0	Move contents of F0 to vector-mask register VM.
MVFM	F0, VM	Move contents of vector-mask register VM to F0.

MIPS Vs VMIPS Comparison

MIPS

```
Loop:  L.D      F0,a
        DADDIU   R4,Rx,#512
        L.D      F2,0(Rx)
        MUL.D    F2,F2,F0
        L.D      F4,0(Ry)
        ADD.D    F4,F4,F2
        S.D      F4,9(Ry)
        DADDIU   Rx,Rx,#8
        DADDIU   Ry,Ry,#8
        DSUBU    R20,R4,Rx
        BNEZ     R20,Loop
```

VMIPS

```
L.D      F0,a
LV        V1,Rx
MULVS.D  V2,V1,F0
LV        V3,Ry
ADDVV.D  V4,V2,V3
SV        V4,Ry
```

NPTEL Online Course : Advanced Computer Architecture : Dr. John Joseph, IIT Guwahati

- ❖ 600 vs 6 instructions,
- ❖ Avoiding loop overhead instructions
- ❖ Loop Vectorization (if no dependency between iterations)

Vector Execution Time

- ❖ **Execution time depends on three factors:**

- ❖ Length of operand vectors

- ❖ Structural hazards among the operations

- ❖ Data dependencies

NPTEL Online Course : Advanced Computer Architecture : Dr. John Jose, IIT Guwahati

- ❖ **VMIPS functional units consume 1 element/clock cycle**

- ❖ VFUs have an initiation rate of one

- ❖ Execution time is approximately the vector length

Vector Execution Time

❖ Convoy

- ❖ Set of vector instructions that could potentially execute together
- ❖ A convoy of instructions must complete execution before any other instructions can begin execution
- ❖ Sequences with RAW dependency hazards can be in the same convoy via chaining

Vector Execution Time

❖ Chaining

- ❖ Allows a vector operation to start as soon as the individual elements of its vector source operand is available
- ❖ Result from the first functional unit in the chain is forwarding to second functional unit
- ❖ A vector instruction is chained to another vector instruction

Vector Execution Time

❖ Chime

- ❖ Unit of time to execute one convoy
- ❖ m convoys execute in m chimes
- ❖ For vector length of n , it requires $m \times n$ clock cycles

NPTEL Online Course : Advanced Computer Architecture : Dr. John Jose, IIT Guwahati

Start-up overhead

- ❖ **Start-up time:** Time between initialization of the instruction and time the first result emerges from pipeline
- ❖ Once pipeline is full, result is produced every cycle.
- ❖ If vector lengths were infinite, start-up overhead is amortized
- ❖ But for finite vector lengths, it adds significant overhead

Vector Execution Time

LV	V1,Rx	;load vector X	
MULVS.D	V2,V1,F0	;vector-scalar multiply	
LV	V3,Ry	;load vector Y	Convoys: // one LS unit
ADDVV.D	V4,V2,V3	;add two vectors	1. LV MULVS.D
SV	Ry,V4	;store the sum	2. LV ADDVV.D
			3. SV

NPTEL Online Course 4: Advanced Computer Architecture : Dr. John Jose, IIT Guwahati

- ❖ Amortize execution time of 1 convoy to 1 chime
- ❖ 3 chimes, 2 FP ops per result; Neglect load overhead
- ❖ Number of cycles /FLOP=1.5
- ❖ A 64 element vector require $64 \times 3 = 192$ chimes or CC

Vector Execution - Challenges

❖ Vector Start up time

- ❖ Functional units are pipelined

- ❖ Latency of vector functional unit is large

 - ❖ Floating-point add → 6 clock cycles

 - ❖ Floating-point multiply → 7 clock cycles

 - ❖ Floating-point divide → 20 clock cycles

 - ❖ Vector load → 12 clock cycles

Vector Mask Registers for Conditionals

- ❖ Control dependencies inside a loop make it non-vectorizable

```
for (i = 0; i < 64; i=i+1)
```

```
    if (X[i] != 0) { X[i] = X[i] - Y[i]; }
```

- ❖ Use vector mask register to disable elements:

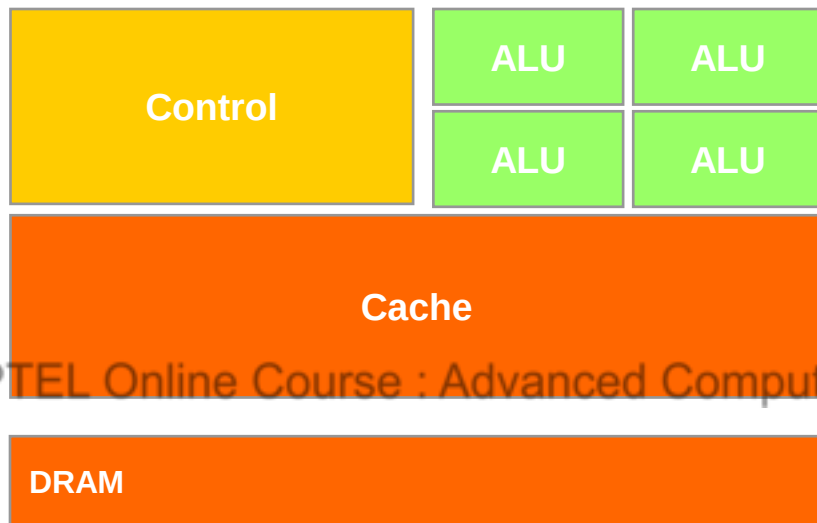
NPTEL Online Course : Advanced Computer Architecture : Dr. John Jose, IIT Guwahati

LV	V1,Rx	;load vector X into V1
LV	V2,Ry	;load vector Y
L.D	F0,#0	;load FP zero into F0
SNEVS.D	V1,F0	;sets VM(i) to 1 if V1(i)≠F0
SUBVV.D	V1,V1,V2	;subtract under vector mask
SV	Rx,V1	;store the result in X

GPU - A Multithreaded Coprocessor

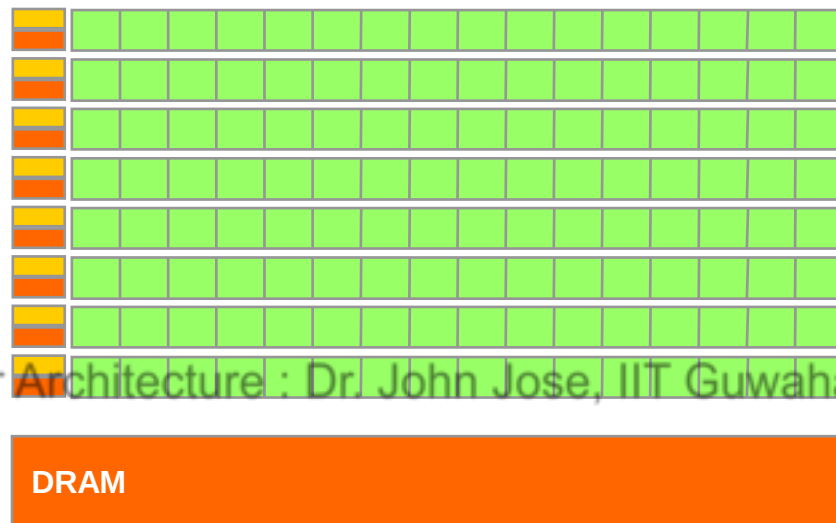
- ❖ The GPU is viewed as a compute **device** that:
 - ❖ Is a coprocessor to the CPU or **host**
 - ❖ Has its own DRAM (**device memory**)
 - ❖ Runs many **threads in parallel**
- ❖ Data-parallel portions of an application are executed on the device as **kernels** which run in parallel on many light weight threads
- ❖ GPU gives best performance if the application is:
 - ❖ Computation intensive
 - ❖ Many independent computations
 - ❖ Many similar computations

CPUs vs GPUs



CPU

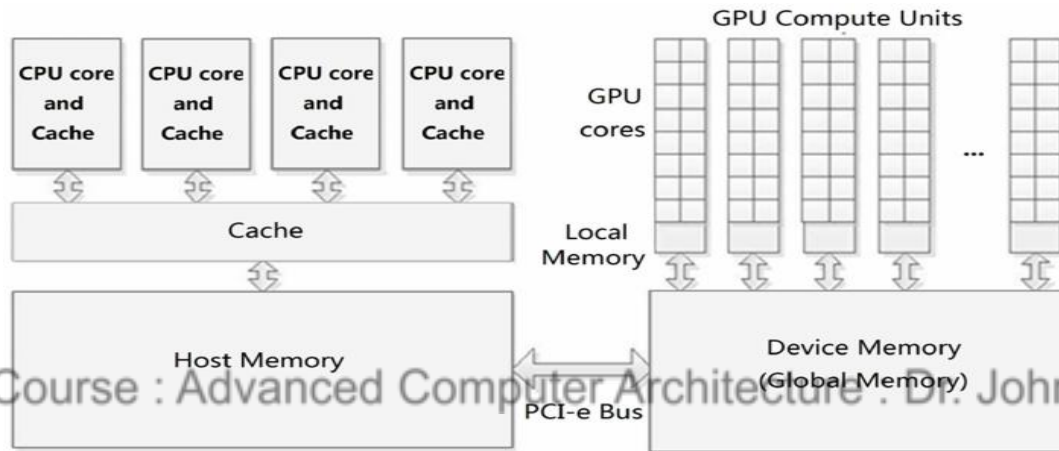
- ❖ Few, complex cores
- ❖ Perform irregular operations
- ❖ Caching and prefetching



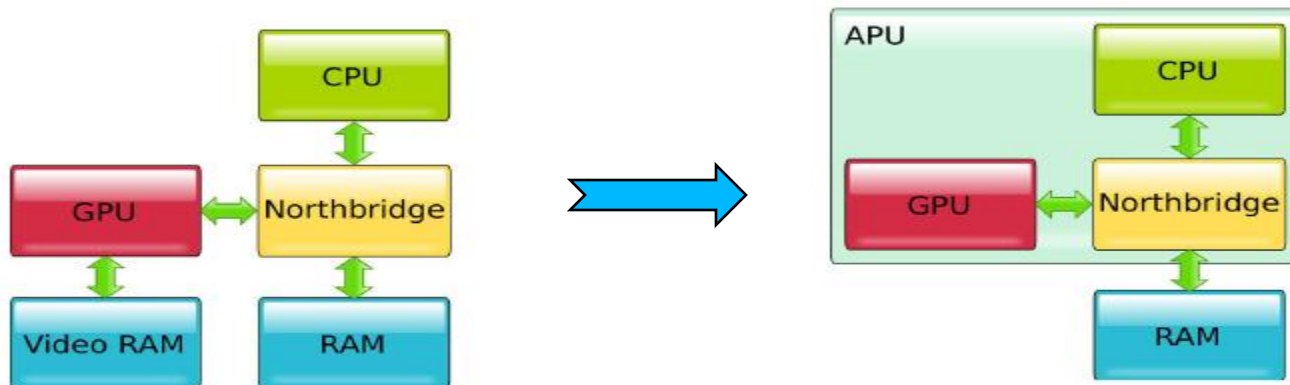
GPU

- ❖ Hundreds of simple cores
- ❖ Same operation on the cores
- ❖ Operating on a common memory
- ❖ No caching, no prefetching

CPUs vs GPUs



NPTEL Online Course : Advanced Computer Architecture : Dr. John Jose, IIT Guwahati



Graphics Processing Units

- ❖ Working on SIMD model
- ❖ Heterogeneous execution model with CPU as the host, GPU as the device
- ❖ CUDA - programming language for GPU
(*CUDA - Compute Unified Device Architecture*)
- ❖ A thread is associated with each data element
- ❖ Threads are organized into blocks, blocks into a grid
- ❖ GPU hardware handles thread management.

Graphics Processing Units

- ❖ The GPU is good at

- ❖ data-parallel processing when the same computation is executed on many data elements in parallel. It has low control flow overhead.

NPTEL Online Course : Advanced Computer Architecture : Dr. John Jose, IIT Guwahati

- ❖ high floating point arithmetic intensity operations that has many calculations per memory access and high floating point to integer ratio.

GPU Architecture – Key Features

❖ Similarities to vector machines:

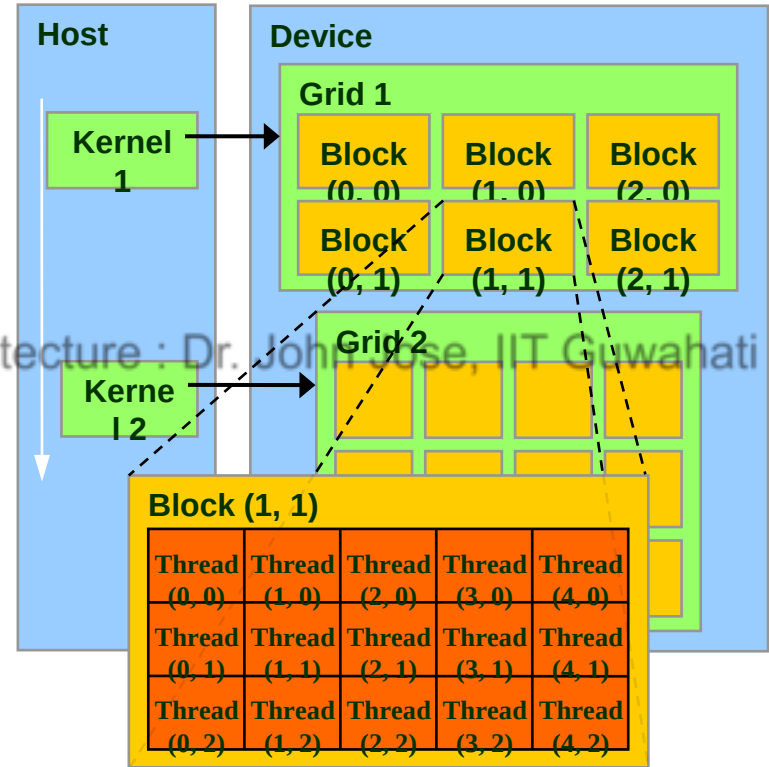
- ❖ Works well with data-level parallel problems
- ❖ Scatter-gather transfers
- ❖ Mask registers
- ❖ Large register files

❖ 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

Thread Batching: Grids and Blocks

- ❖ A kernel is executed as a **grid of thread blocks**
- ❖ Threads share memory space
- ❖ A **thread block** is a batch of threads that can **cooperate** with each other by:
 - ❖ Synchronizing their execution for hazard-free shared memory accesses
 - ❖ Efficiently sharing data through a low latency **shared memory**
- ❖ Two threads from two different blocks cannot cooperate



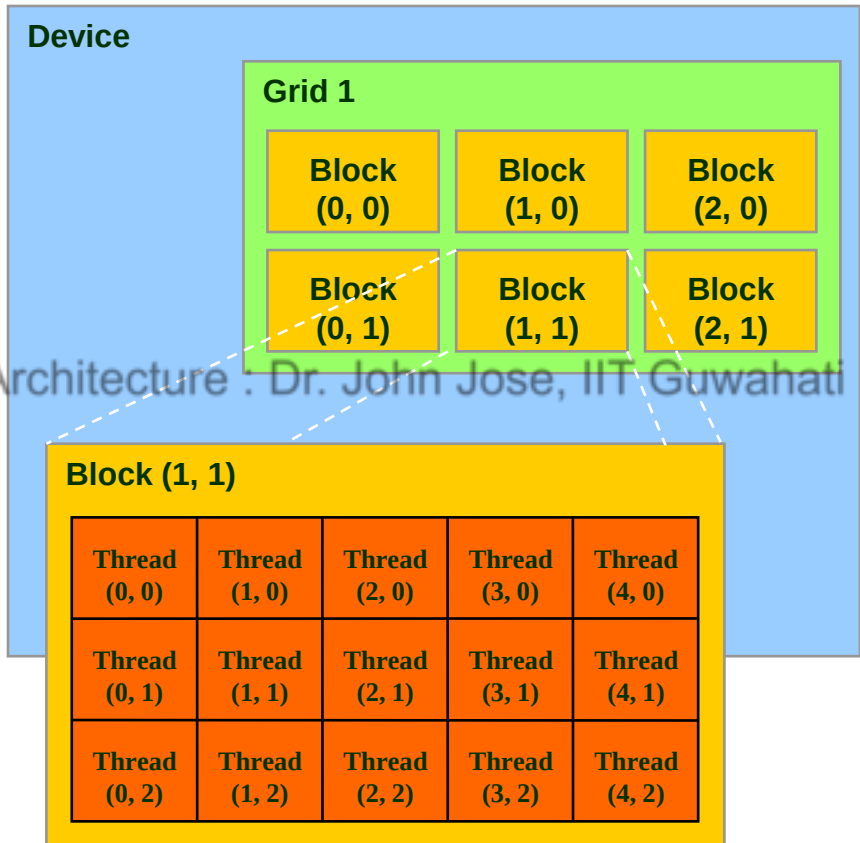
Block and Thread IDs

- ❖ Threads and blocks have IDs

- ❖ Block ID: 1D or 2D
(blockIdx.x, blockIdx.y)

- ❖ Thread ID: 1D, 2D, or 3D
(threadIdx.{x,y,z})

- ❖ Simplifies memory addressing when processing multidimensional data



Illustrative Example

- ❖ How a thread works?
 - ❖ Scan elements of array of numbers
 - ❖ How many times does **6** appear in an array of 16 elements
 - ❖ Total 4 threads
 - ❖ Each thread examines 4 elements,
 - ❖ 1 block in grid, 1 grid



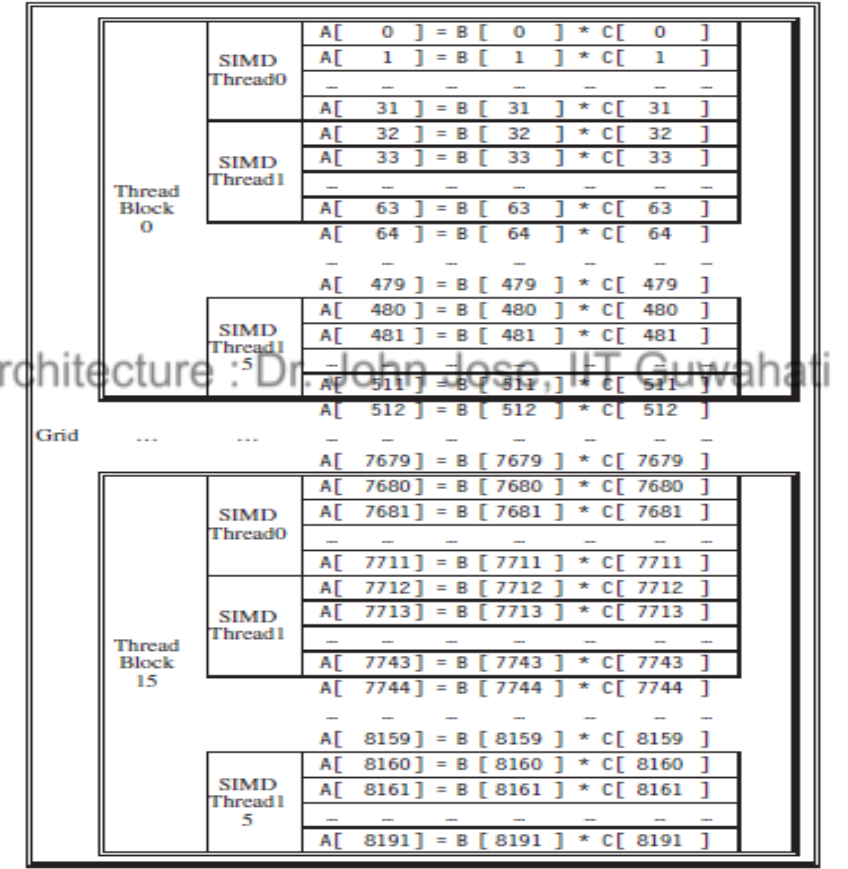
threadIdx.x = 0 examines in_array elements 0, 4, 8, 12
threadIdx.x = 1 examines in_array elements 1, 5, 9, 13
threadIdx.x = 2 examines in_array elements 2, 6, 10, 14
threadIdx.x = 3 examines in_array elements 3, 7, 11, 15



Known as a
cyclic data
distribution

Illustrative Example

- ❖ Multiply two vectors of length 8192
- ❖ Grid – entire code
- ❖ Grid is divided into blocks
- ❖ Grid size = 16 blocks
- ❖ Block is assigned to few multithreaded SIMD processor
- ❖ 16 multithreaded SIMD processor
- ❖ Each SIMD instruction executes 32 elements at a time
- ❖ Here 512 threads per block



Reductions

❖ Reduction Operation:

```
for (i=9999; i>=0; i=i-1)  
    sum = sum + x[i] * y[i];
```

❖ Transform to...

```
for (i=9999; i>=0; i=i-1)  
    sum [i] = x[i] * y[i];
```

NPTEL Online Course : Advanced Computer Architecture : Dr. John Jose IIT Guwahati

❖ Do on p processors:

```
for (i=999; i>=0; i=i-1)
```

```
    finalsum[p] = finalsum[p] + sum[i+1000*p];
```

❖ Add the last serially.

```
finalsum = finalsum + sum[i];
```

**THANK
YOU!**

FOR

YOUR

PRESENCE

NPTEL Online Course : Advanced Computer Architecture : Dr. John Jose, IIT Guwahati

johnjose@iitg.ac.in

<http://www.iitg.ac.in/johnjose/>