Computer Organization and Architecture

Lecture – 27

Nov 21st , 2022

Vector Processors

- Vector processors abstract operations on vectors, e.g. replace the following loop

```
for (i=0; i<n; i++) {
    a[i] = b[i] + c[i];
}
```

by

```
a = b + c;  ⟶  ADDV.D V10, V8, V6
```

- Some languages offer high-level support for these operations (e.g. Fortran90, Python numpy, etc.)

Main concepts

- Advantages of vector instructions
  - A single instruction specifies a great deal of work
  - Since each loop iteration **must not** contain data dependence to other loop iterations
    - No need to check for data hazards between loop iterations
    - Only one check required between two vector instructions
    - Loop branches eliminated

Basic vector architecture

- A modern vector processor contains
  - Regular, pipelined scalar units
  - Regular scalar registers
  - Vector units
  - Vector register: can hold a fixed number of entries (e.g. 64)
  - Vector load-store units

DAXPY loop

Double precision *a* × X plus Y

$$Y = a \times X + Y$$

X and Y are vectors with 64 dimensions.
Assume that a in X28, and that the starting addresses of *X* and *Y* are, X19 and X20, respectively.

DAXPY loop

Double precision *a* × X plus Y

$$Y = a \times X + Y$$

Assume that the starting addresses of *X* and *Y* are in X19 and X20, respectively.

```
LDURD   D0,[X28,a]    //load scalar a
```

DAXPY loop

Double precision *a* × X plus Y

$$Y = a \times X + Y$$

Assume that the starting addresses of *X* and *Y* are in X19 and X20, respectively.

```
LDURD   D0,[X28,a]    //load scalar a
ADDI    X0,X19,512    //upper bound of what to load
```

Iterate over 64 elements (64 * 8)

DAXPY loop

Double precision *a* × X plus Y

$$Y = a \times X + Y$$

Assume that the starting addresses of *X* and *Y* are in X19 and X20, respectively.

```
        LDURD   D0,[X28,a]    //load scalar a
        ADDI    X0,X19,512    //upper bound of what to load
loop:   LDURD   D2,[X19,#0]   //load x(i)
```

DAXPY loop

Double precision $a \times X$ plus Y

$$Y = a \times X + Y$$

Assume that the starting addresses of *X* and *Y* are in X19 and X20, respectively.

```
        LDURD   D0,[X28,a]    //load scalar a
        ADDI    X0,X19,512    //upper bound of what to load
loop:   LDURD   D2,[X19,#0]   //load x(i)
        FMULD   D2,D2,D0      //a x x(i)
```

DAXPY loop

Double precision $a \times$ X plus Y

$$Y = a \times X + Y$$

Assume that the starting addresses of *X* and *Y* are in X19 and X20, respectively.

```
        LDURD   D0,[X28,a]    //load scalar a
        ADDI    X0,X19,512    //upper bound of what to load
loop:   LDURD   D2,[X19,#0]   //load x(i)
        FMULD   D2,D2,D0      //a x x(i)
        LDURD   D4,[X20,#0]   //load y(i)
```

DAXPY loop

Double precision $a \times X$ plus Y

$$Y = a \times X + Y$$

Assume that the starting addresses of *X* and *Y* are in X19 and X20, respectively.

```
        LDURD   D0,[X28,a]    //load scalar a
        ADDI    X0,X19,512    //upper bound of what to load
loop:   LDURD   D2,[X19,#0]   //load x(i)
        FMULD   D2,D2,D0      //a x x(i)
        LDURD   D4,[X20,#0]   //load y(i)
        FADDD   D4,D4,D2      //a x x(i) + y(i)
```

DAXPY loop

Double precision *a* × X plus Y

$$Y = a \times X + Y$$

Assume that the starting addresses of *X* and *Y* are in X19 and X20, respectively.

```
        LDURD   D0,[X28,a]    //load scalar a
        ADDI    X0,X19,512    //upper bound of what to load
loop:   LDURD   D2,[X19,#0]   //load x(i)
        FMULD   D2,D2,D0      //a x x(i)
        LDURD   D4,[X20,#0]   //load y(i)
        FADDD   D4,D4,D2      //a x x(i) + y(i)
        STURD   D4,[X20,#0]   //store into y(i)
```

DAXPY loop

Double precision $a \times X$ plus Y

$$Y = a \times X + Y$$

Assume that the starting addresses of *X* and *Y* are in X19 and X20, respectively.

```
        LDURD   D0,[X28,a]    //load scalar a
        ADDI    X0,X19,512    //upper bound of what to load
loop:   LDURD   D2,[X19,#0]   //load x(i)
        FMULD   D2,D2,D0      //a x x(i)
        LDURD   D4,[X20,#0]   //load y(i)
        FADDD   D4,D4,D2      //a x x(i) + y(i)
        STURD   D4,[X20,#0]   //store into y(i)
        ADDI    X19,X19,#8    //increment index to x
        ADDI    X20,X20,#8    //increment index to y
```

DAXPY loop

Double precision *a* × X plus Y

$$Y = a \times X + Y$$

Assume that the starting addresses of *X* and *Y* are in X19 and X20, respectively.

```
        LDURD   D0,[X28,a]    //load scalar a
        ADDI    X0,X19,512    //upper bound of what to load
loop:   LDURD   D2,[X19,#0]   //load x(i)
        FMULD   D2,D2,D0      //a x x(i)
        LDURD   D4,[X20,#0]   //load y(i)
        FADDD   D4,D4,D2      //a x x(i) + y(i)
        STURD   D4,[X20,#0]   //store into y(i)
        ADDI    X19,X19,#8    //increment index to x
        ADDI    X20,X20,#8    //increment index to y
        CMPB    X0,X19        //compute bound
        B.NE    loop          //check if done
```

vector LEGv8 code for DAXPY

```
LDURD       D0,[X28,a]        //load scalar a
```

vector LEGv8 code for DAXPY

```
LDURD       D0,[X28,a]        //load scalar a
LDURDV      V1,[X19,#0]       //load vector x
```

vector LEGv8 code for DAXPY

```
LDURD     D0,[X28,a]      //load scalar a
LDURDV    V1,[X19,#0]     //load vector x
FMULDVS   V2,V1,D0        //vector-scalar multiply
```

vector LEGv8 code for DAXPY

```
LDURD     D0,[X28,a]      //load scalar a
LDURDV    V1,[X19,#0]     //load vector x
FMULDVS   V2,V1,D0        //vector-scalar multiply
LDURDV    V3,[X20,#0]     //load vector y
```

vector LEGv8 code for DAXPY

```
LDURD     D0,[X28,a]      //load scalar a
LDURDV    V1,[X19,#0]     //load vector x
FMULDVS   V2,V1,D0        //vector-scalar multiply
LDURDV    V3,[X20,#0]     //load vector y
FADDDV    V4,V2,V3        //add y to product
```

vector LEGv8 code for DAXPY

```
LDURD     D0,[X28,a]      //load scalar a
LDURDV    V1,[X19,#0]     //load vector x
FMULDVS   V2,V1,D0        //vector-scalar multiply
LDURDV    V3,[X20,#0]     //load vector y
FADDDV    V4,V2,V3        //add y to product
STURDV    V4,[X20,#0]     //store the result
```

SIMD Instructions

- Originally developed for Multimedia applications
- Same operation executed for multiple data items

History of GPUs

- Graphics Processing Units
  - Processors oriented to 3D graphics tasks
  - Vertex/pixel processing, shading, texture mapping, rasterization

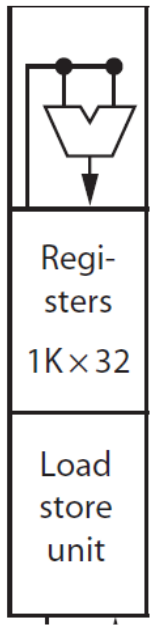
Graphics Processing Units (GPU)

- Hardware in Graphics Units similar to Vector Processors
  - Works well with data-level parallel problems

- Differences:
  - No scalar processor
  - Uses multithreading extensively
  - Has many functional units, as opposed to a few deeply pipelined units like a vector processor

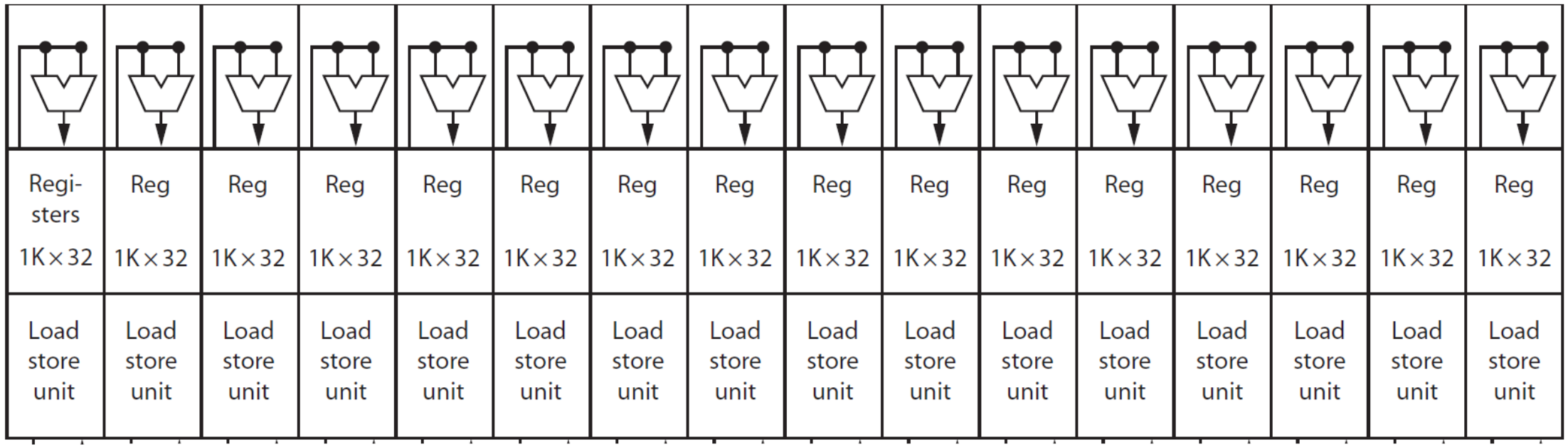
Graphics Processing Units (GPU)

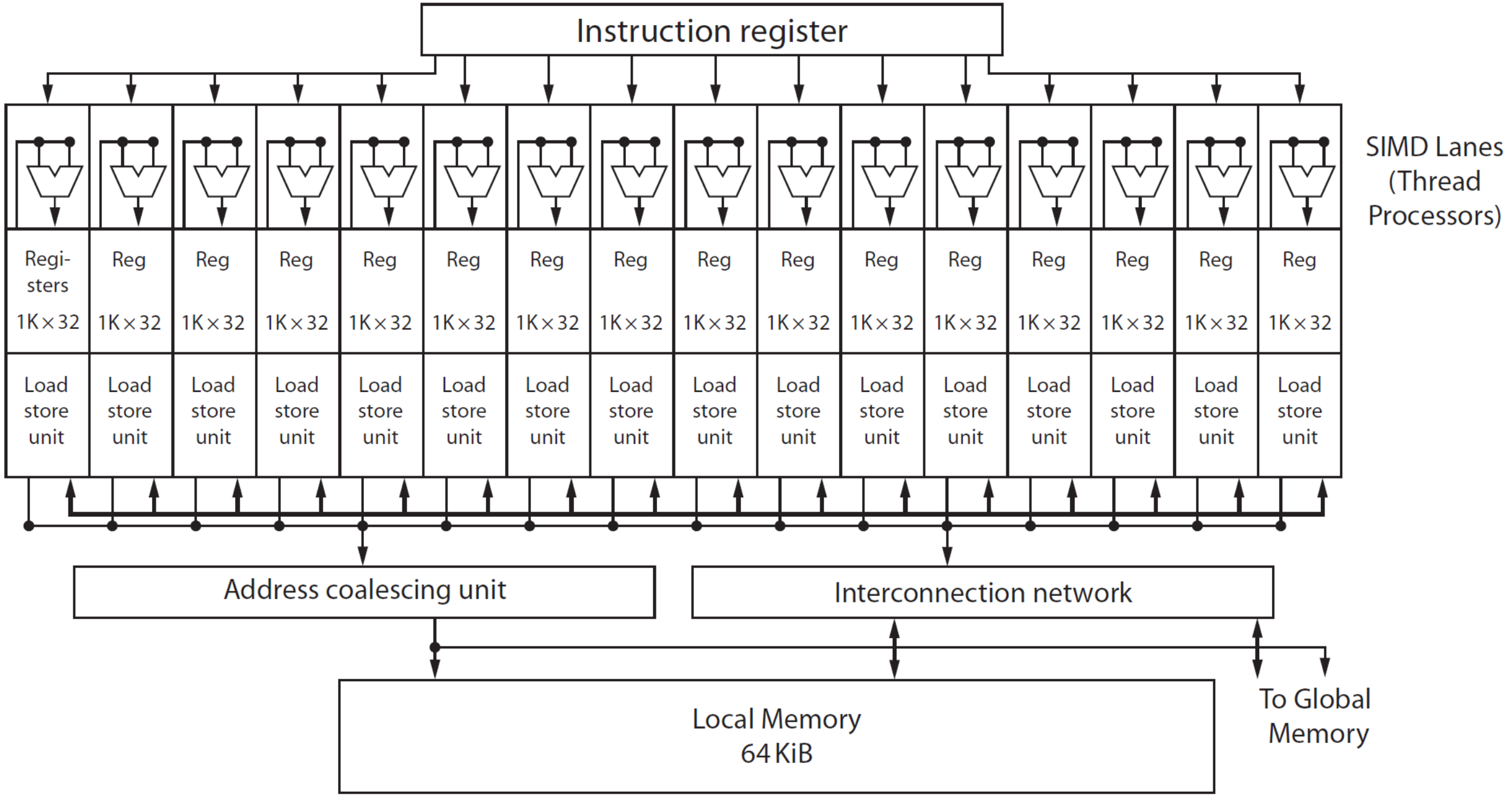
- Vector processors are SIMD architectures
  - Instructions are pipelined
- GPUs contain numerous multithreaded SIMD processors
  - Rely on hardware multithreading
  - Collection of SIMD, so GPU is MIMD


Regi-
sters
1K × 32
Load
store
unit

Example: NVIDIA Fermi

- SIMD Processor: 16 SIMD lanes
- SIMD instruction
  - Operates on 32 element wide threads
  - Dynamically scheduled on 16-wide processor over 2 cycles
- 16K x 32-bit registers spread across lanes
  - 64 registers per thread context


Regi-sters 1K×32
Reg 1K×32
Load store unit
SIMD Lanes (Thread Processors)

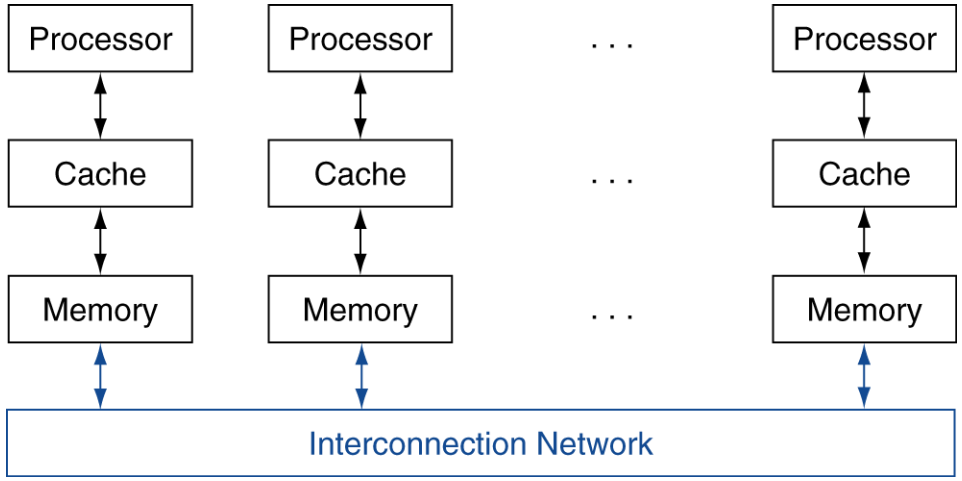

Instruction register
Regi-sters 1K×32
Reg 1K×32
Load store unit
SIMD Lanes (Thread Processors)
Address coalescing unit
Interconnection network
Local Memory 64 KiB
To Global Memory

GPU hardware has two levels of hardware schedulers:

1. The Thread Block Scheduler that assigns blocks of threads to multithreaded SIMD processors, and
2. The SIMD Thread Scheduler within a SIMD processor, which schedules when SIMD threads should run.

Message Passing Microprocessors

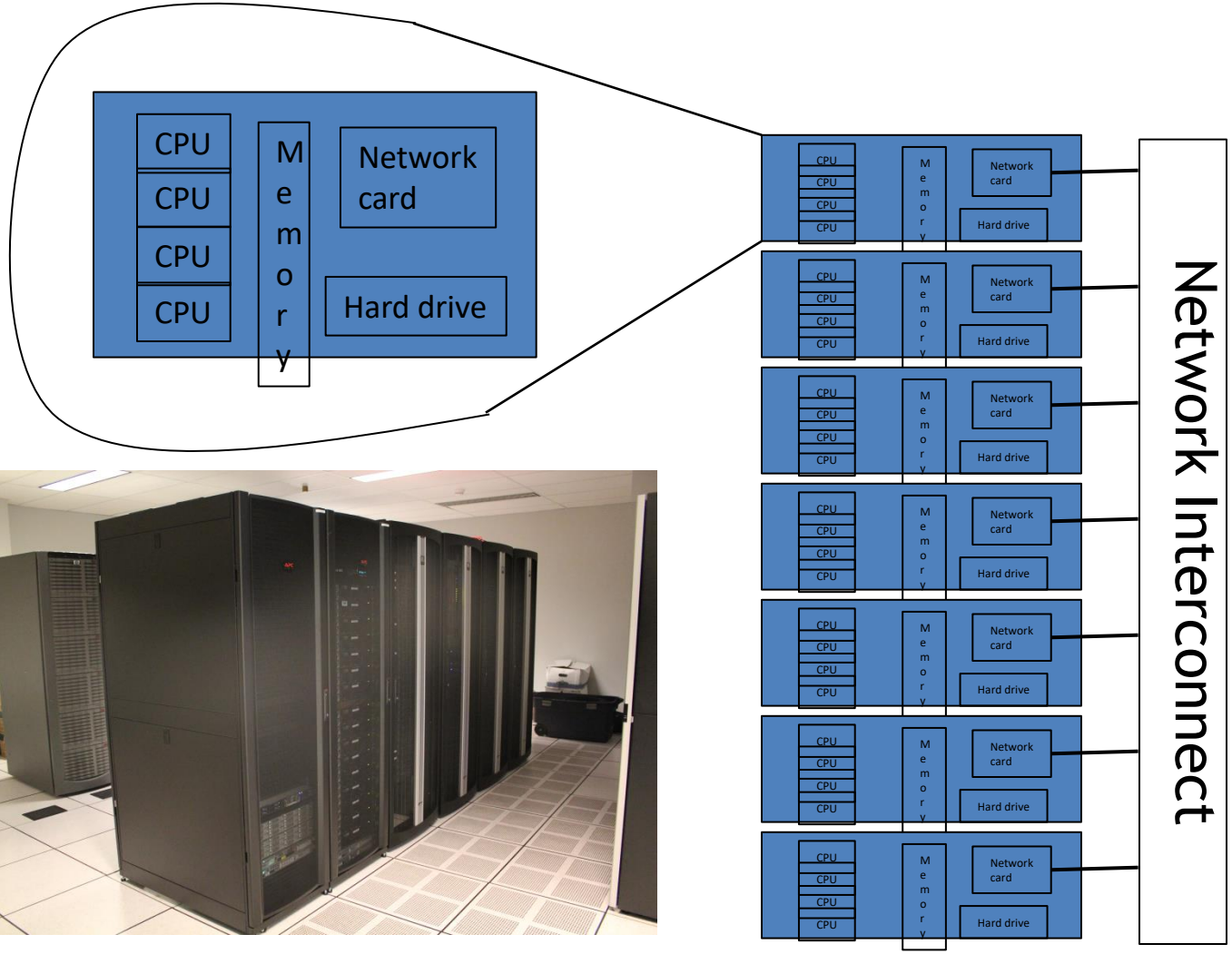
- Each processor has private physical address space
- Hardware sends/receives messages between processors



Cluster Computing

- Cluster: collection of individual PC's (compute nodes) connected by a (high performance) network interconnect
  - Each compute node is an independent entity with its own
    - Processor(s)
    - Main memory
    - One ore multiple networking cards
  - All compute nodes typically have access to a shared file system (e.g. Network File System (NFS))
    - Removes the necessity to replicate programs and data on all compute nodes
    - All accesses to files require communication over the network

Conceptual View



Clusters

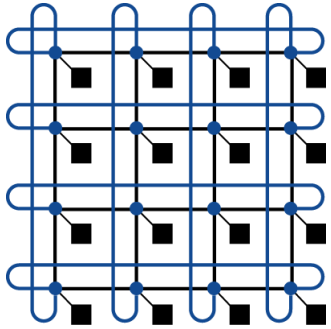
- Suitable for applications with independent tasks
  - Web servers, databases, simulations, …
- High availability, scalable, affordable
- Problems
  - Administration cost (prefer virtual machines)
  - Low interconnect bandwidth
    - c.f. processor/memory bandwidth on an SMP

Interconnection Networks

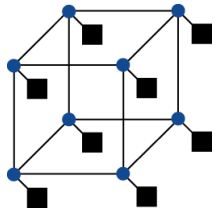
- Network topologies
  - Arrangements of processors, switches, and links

Bus

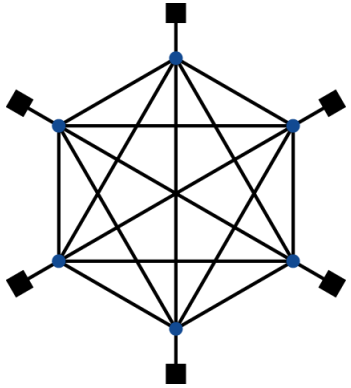
Ring



2D Mesh



N-cube (N = 3)



Fully connected

Cloud Computing

- **Cloud Computing:** general term used to describe a class of network based computing
  - a collection/group of integrated and networked hardware, software and Internet infrastructure (called a platform).
  - Using the Internet for communication and transport provides hardware, software and networking services to clients
- Hides the complexity and details of the underlying infrastructure from users and applications by providing very simple graphical interface or API

Cloud Computing (II)

- The platform provides on demand services, that are always on, anywhere, anytime and any place.
  - Pay for use and as needed
  - Scale up and down in capacity and functionality
- The hardware and software services are available to
  - general public, enterprises, corporations, and businesses markets
- Services or data are hosted on remote infrastructure

Cloud Service Models

- Software as a Service (SaaS):
  - execute a specific application required for business / research
- Platform as a Service (PaaS):
  - deploy customer created applications
- Infrastructure as a Service (IaaS):
  - rent processing and compute capacity, storage, etc.

Concluding Remarks

- Goal: higher performance by using multiple processors
- Difficulties
  - Developing parallel software
  - Devising appropriate architectures
- SaaS importance is growing and clusters are a good match