Computer Organization and Structure

Bing-Yu Chen
National Taiwan University
Multiprocessors and Clusters

- Parallel Processing Programs
- Graphics Processing Units
- Multiprocessor Network Topologies
Why parallel computing?

- Moore's law is dead (for CPU frequency)
Top500 Supercomputers

1. Titan (Cray + NVIDIA)
   - 560,640 cores, 710 TB memory

2. Sequoia (IBM)
   - 1,572,864 cores, 1,573 TB memory

3. K computer (Fujitsu)
   - 705,024 cores, 1,410 TB memory

4. Mira (IBM)
   - 786,432 cores

5. JUQUEEN (IBM)
   - 393,216 cores, 393 TB memory
Introduction

- Goal: connecting multiple computers to get higher performance
  - Multiprocessors
  - Scalability, availability, power efficiency
- Job-level (process-level) parallelism
  - High throughput for independent jobs
- Parallel processing program
  - Single program run on multiple processors
- Multicore microprocessors
  - Chips with multiple processors (cores)
Parallel Programming

- Parallel software is the problem
- Need to get significant performance improvement
  - Otherwise, just use a faster uniprocessor, since it’s easier!

- Difficulties
  - Partitioning
  - Coordination
  - Communications overhead
Amdahl’s Law

- Serial Parallelizable work
- Serial 2 processors
- Serial 4 processors
- Serial many processors
Amdahl’s Law

Total speedup = \frac{1}{(1-P) + \frac{P}{S}}

Parallelizable work

Speedup for Parallelizable work

Example:
P: 0.8 (80% work is parallelizable)
S: 8 (8 processors)
Total speedup: 3.33x
Amdahl’s Law
Amdahl’s Law

- Sequential part can limit speedup
- Example: 100 processors, 90x speedup?

\[ \text{Speedup} = \frac{1}{(1 - F_{\text{parallelizable}}) + F_{\text{parallelizable}} / 100} = 90 \]

- Solving: \( F_{\text{parallelizable}} = 0.999 \)

- Need sequential part to be 0.1% of original time
Scaling Example

- Workload: sum of 10 scalars, and 10x10 matrix sum
  - Speed up from 10 to 100 processors
- Single processor: Time = (10 + 100) x \( t_{\text{add}} \)
  - 10 processors
    - Time = 10 x \( t_{\text{add}} \) + 100/10 x \( t_{\text{add}} \) = 20 x \( t_{\text{add}} \)
    - Speedup = 110/20 = 5.5x (5.5% of potential)
  - 100 processors
    - Time = 10 x \( t_{\text{add}} \) + 100/100 x \( t_{\text{add}} \) = 11 x \( t_{\text{add}} \)
    - Speedup = 110/11 = 10x (10% of potential)
- Assumes load can be balanced across processors
Scaling Example

- What if matrix size is 100 × 100?
- Single processor: Time = (10 + 10000) x t_{add}
- 10 processors
  - Time = 10 x t_{add} + 10000/10 x t_{add} = 1010 x t_{add}
  - Speedup = 10010/1010 = 9.9x (9.9% of potential)
- 100 processors
  - Time = 10 x t_{add} + 10000/100 x t_{add} = 110 x t_{add}
  - Speedup = 10010/110 = 91x (91% of potential)
- Assuming load balanced
Strong vs Weak Scaling

- **Strong scaling**: problem size fixed
  - As in example

- **Weak scaling**: problem size proportional to number of processors
  - 10 processors, 10x10 matrix
    - Time = \(20 \times t_{\text{add}}\)
  - 100 processors, 32x32 matrix
    - Time = \(10 \times t_{\text{add}} + \frac{1000}{100} \times t_{\text{add}} = 20 \times t_{\text{add}}\)
  - Constant performance in this example
Parallelization design for processors

- **Instruction level parallelism**
  
  add $t0, $t1, $t2
  add $t3, $t4, $t5

- **Data level parallelism**
  
  add 0($t1), 0($t2), $t3
  add 4($t1), 4($t2), $t3
  add 8($t1), 8($t2), $t3
  ...

Computer Graphics Rendering

WORLD

VRCS
view frustum

NDC

SCREEN
viewport

MODEL

PROJECTED
window

CLIPPED
window

RASTERIZED
viewport
History of GPUs

- Early video cards
  - Frame buffer memory with address generation for video output

- 3D graphics processing
  - Originally high-end computers (e.g., SGI)
  - Moore’s Law $\Rightarrow$ lower cost, higher density
  - 3D graphics cards for PCs and game consoles

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

- Processing is highly data-parallel
  - GPUs are highly multithreaded
  - Use thread switching to hide memory latency
    - Less reliance on multi-level caches
  - Graphics memory is wide and high-bandwidth

- Trend toward general purpose GPUs
  - Heterogeneous CPU/GPU systems
  - CPU for sequential code, GPU for parallel code

- Programming languages/APIs
  - DirectX, OpenGL
  - C for Graphics (Cg), High Level Shader Language (HLSL)
  - Compute Unified Device Architecture (CUDA)
Graphics pipeline

Instructions
States
Data

Transforms
Lighting, etc

Rasterize
Pixel shading

Z-buffer
Transparency

Input processor

Do geometry stuff

Do pixel stuff

Accumulate pixel result
Make it faster

Input processor

Do geometry stuff

Do pixel stuff

Accumulate pixel result
Add framebuffer support

Input processor

Do geometry stuff

Do pixel stuff

Accumulate pixel result

FB (memory)
Add programmability

Get data
Process data
Output data

Front end

Geometry shader ALU

Pixel shader ALU

Raster operations

FB (memory)
Uniform shader

Buffer

Front end

Uniform shader ALU

FB (memory)

Raster operations
Scaling it up again
Example: NVIDIA Tesla

Interconnection Networks

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

- Bus
- Ring
- 2D Mesh
- N-cube (N = 3)
- Fully connected
Network Characteristics

- Performance
  - Latency per message (unloaded network)
  - Throughput
    - Link bandwidth
    - Total network bandwidth
    - Bisection bandwidth
  - Congestion delays (depending on traffic)

- Cost
- Power
- Routability in silicon