

Chapter 6

Parallel Processors from Client to Cloud

Introduction

- Goal: connecting multiple computers to get higher performance
 - Multiprocessors
 - Scalability, availability, power efficiency
- Task-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)

Hardware and Software

- Hardware
 - Serial: e.g., Pentium 4
 - Parallel: e.g., quad-core Xeon e5345
- Software
 - Sequential: e.g., matrix multiplication
 - Concurrent: e.g., operating system
- Sequential/concurrent software can run on serial/parallel hardware
 - Challenge: making effective use of parallel hardware

What We've Already Covered

- §2.11: Parallelism and Instructions
 - Synchronization
- §3.6: Parallelism and Computer Arithmetic
 - Subword Parallelism
- §4.10: Parallelism and Advanced Instruction-Level Parallelism
- §5.10: Parallelism and Memory Hierarchies
 - Cache Coherence

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

- Sequential part can limit speedup
- Example: 100 processors, 90× speedup?
 - $T_{\text{new}} = T_{\text{parallelizable}}/100 + T_{\text{sequential}}$
 - $$\text{Speedup} = \frac{1}{(1 - F_{\text{parallelizable}}) + F_{\text{parallelizable}}/100} = 90$$
 - Solving: $F_{\text{parallelizable}} = 0.999$
- Need sequential part to be 0.1% of original time

Scaling Example

- Workload: sum of 10 scalars, and 10×10 matrix sum
 - Speed up from 10 to 100 processors
- Single processor: Time = $(10 + 100) \times t_{\text{add}}$
- 10 processors
 - Time = $10 \times t_{\text{add}} + 100/10 \times t_{\text{add}} = 20 \times t_{\text{add}}$
 - Speedup = $110/20 = 5.5$ (55% of potential)
- 100 processors
 - Time = $10 \times t_{\text{add}} + 100/100 \times t_{\text{add}} = 11 \times t_{\text{add}}$
 - Speedup = $110/11 = 10$ (10% of potential)
- Assumes load can be balanced across processors

Scaling Example (cont)

- What if matrix size is 100×100 ?
- Single processor: Time = $(10 + 10000) \times t_{\text{add}}$
- 10 processors
 - Time = $10 \times t_{\text{add}} + 10000/10 \times t_{\text{add}} = 1010 \times t_{\text{add}}$
 - Speedup = $10010/1010 = 9.9$ (99% of potential)
- 100 processors
 - Time = $10 \times t_{\text{add}} + 10000/100 \times t_{\text{add}} = 110 \times t_{\text{add}}$
 - Speedup = $10010/110 = 91$ (91% of potential)
- Assuming load balanced

Instruction and Data Streams

■ An alternate classification

		Data Streams	
		Single	Multiple
Instruction Streams	Single	SISD: Intel Pentium 4	SIMD: SSE instructions of x86
	Multiple	MISD: No examples today	MIMD: Intel Xeon e5345

■ SPMD: Single Program Multiple Data

- A parallel program on a MIMD computer
- Conditional code for different processors

Vector Processors

- Highly pipelined function units
- Stream data from/to vector registers to units
 - Data collected from memory into registers
 - Results stored from registers to memory
- Example: Vector extension to MIPS
 - 32×64 -element registers (64-bit elements)
 - Vector instructions
 - `lv, sv`: load/store vector
 - `addv.d`: add vectors of double
 - `addvs.d`: add scalar to each element of vector of double
- Significantly reduces instruction-fetch bandwidth

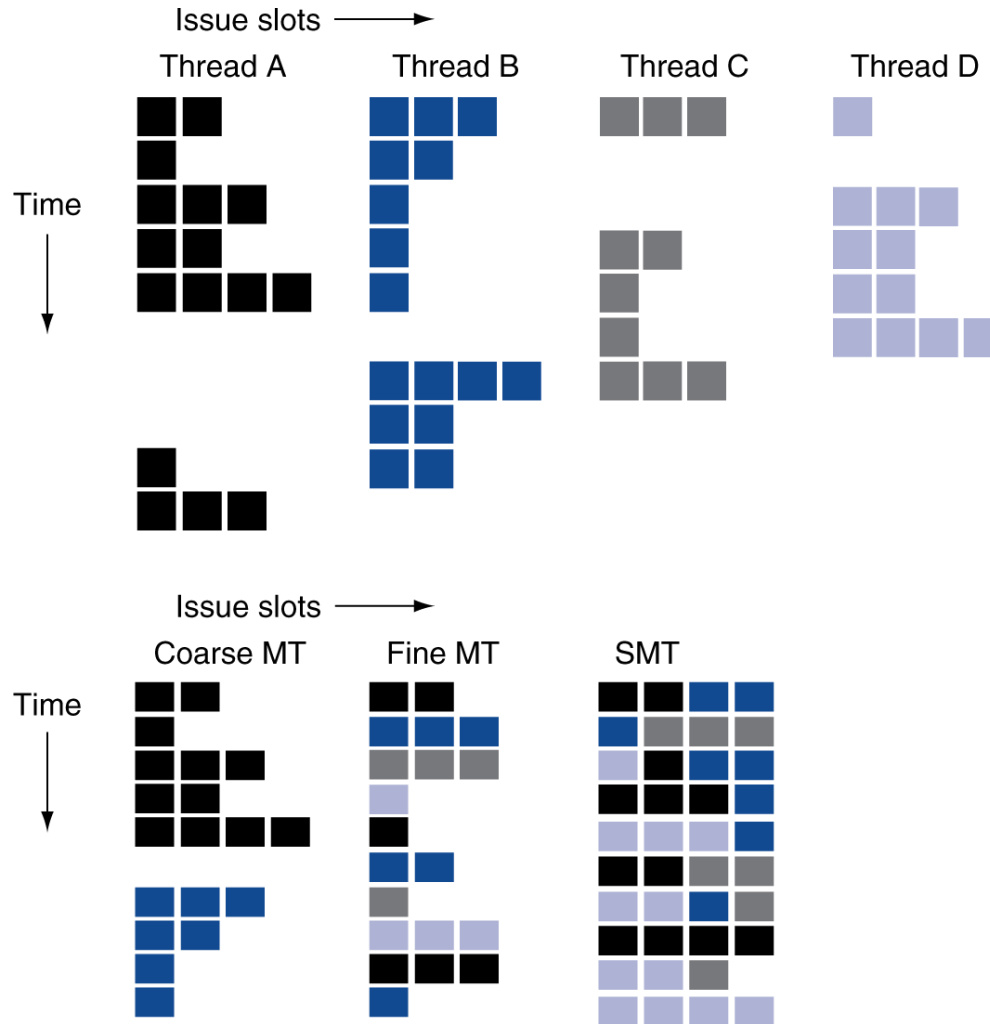
Multithreading

- Performing multiple threads of execution in parallel
 - Replicate registers, PC, etc.
 - Fast switching between threads
- Fine-grain multithreading
 - Switch threads after each cycle
 - Interleave instruction execution
 - If one thread stalls, others are executed
- Coarse-grain multithreading
 - Only switch on long stall (e.g., L2-cache miss)
 - Simplifies hardware, but doesn't hide short stalls (eg, data hazards)

Simultaneous Multithreading

- In multiple-issue dynamically scheduled processor
 - Schedule instructions from multiple threads
 - Instructions from independent threads execute when function units are available
 - Within threads, dependencies handled by scheduling and register renaming
- Example: Intel Pentium-4 HT
 - Two threads: duplicated registers, shared function units and caches

Multithreading Example

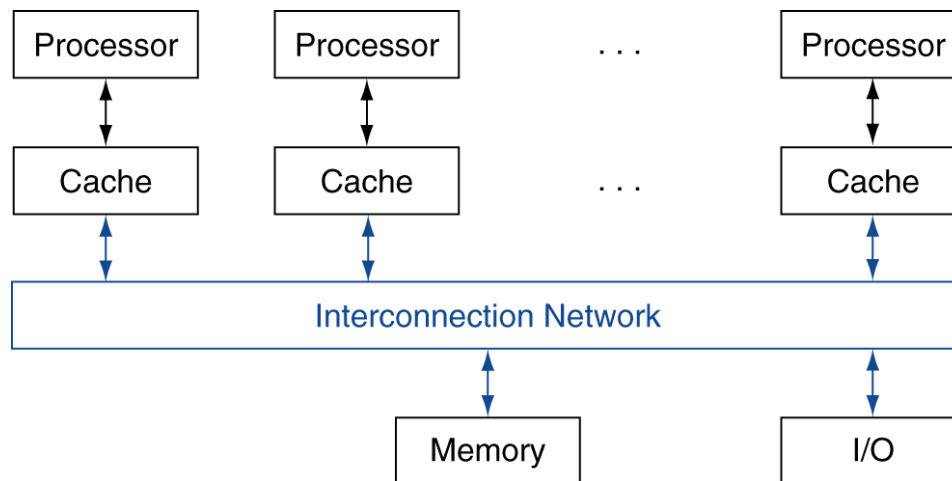


Future of Multithreading

- Will it survive? In what form?
- Power considerations \Rightarrow simplified microarchitectures
 - Simpler forms of multithreading
- Tolerating cache-miss latency
 - Thread switch may be most effective
- Multiple simple cores might share resources more effectively

Shared Memory

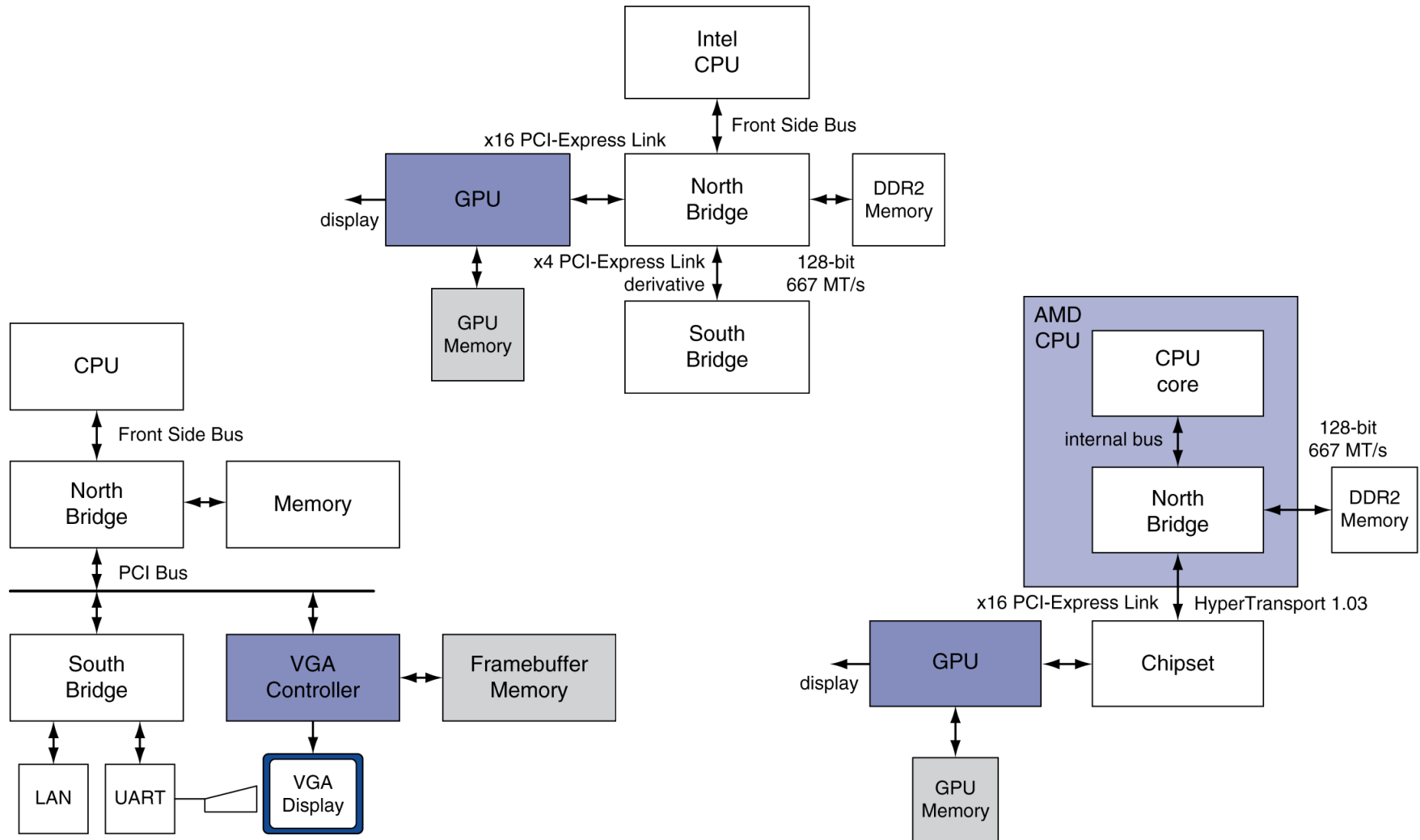
- SMP: shared memory multiprocessor
 - Hardware provides single physical address space for all processors
 - Synchronize shared variables using locks
 - Memory access time
 - UMA (uniform) vs. NUMA (nonuniform)



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

Graphics in the System

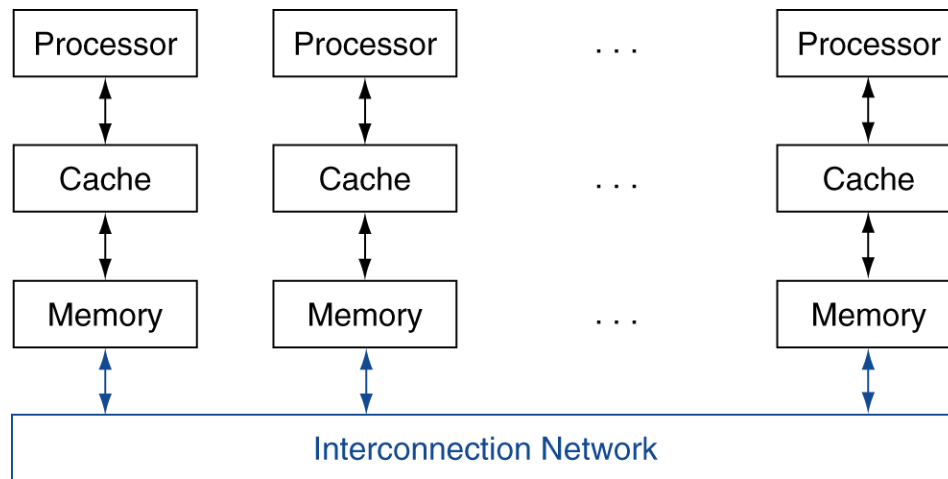


GPU Architectures

- 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)

Message Passing

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



Loosely Coupled Clusters

- Network of independent computers
 - Each has private memory and OS
 - Connected using I/O system
 - E.g., Ethernet/switch, Internet
- 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

Grid Computing

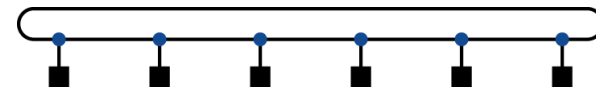
- Separate computers interconnected by long-haul networks
 - E.g., Internet connections
 - Work units farmed out, results sent back
- Can make use of idle time on PCs
 - E.g., SETI@home, World Community Grid

Interconnection Networks

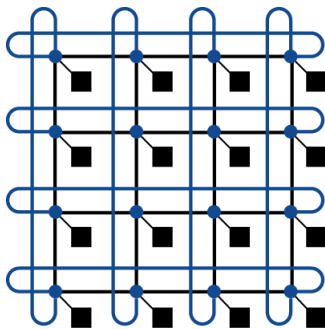
- Network topologies
 - Arrangements of processors, switches, and links



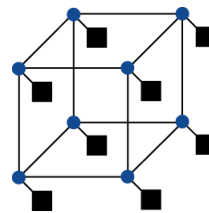
Bus



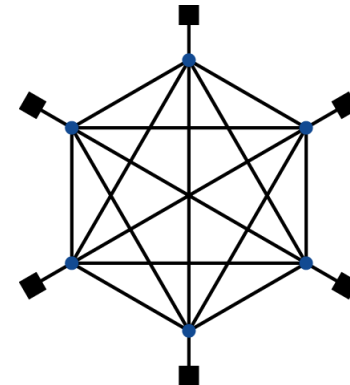
Ring



2D Mesh

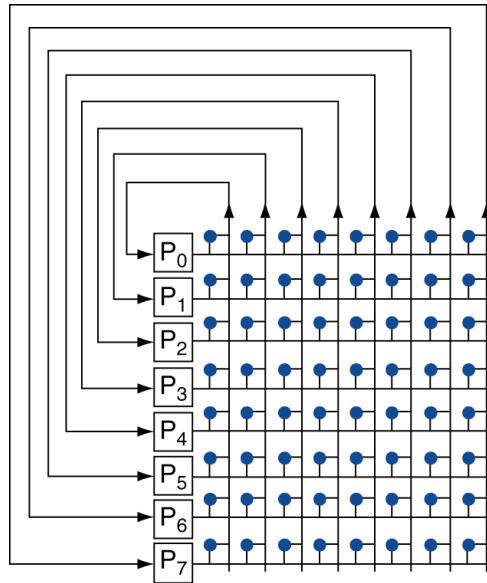


N-cube ($N = 3$)

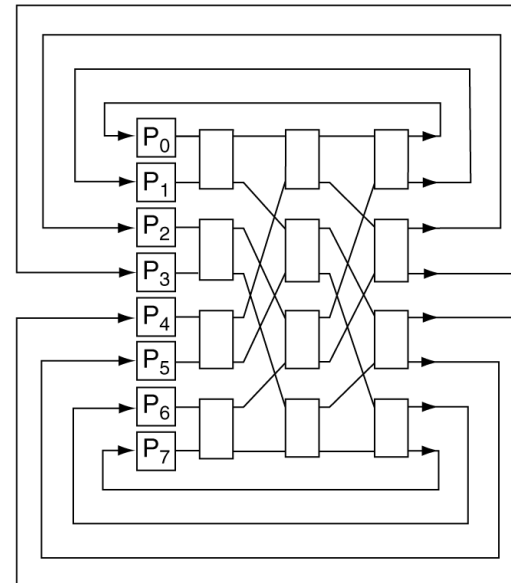


Fully connected

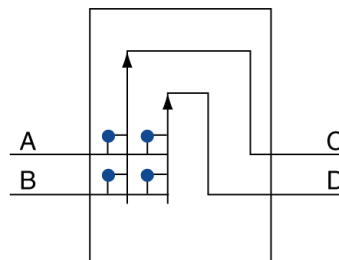
Multistage Networks



a. Crossbar



b. Omega network



c. Omega network switch box

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

Parallel Benchmarks

- Linpack: matrix linear algebra
- SPECrate: parallel run of SPEC CPU programs
 - Job-level parallelism
- SPLASH: Stanford Parallel Applications for Shared Memory
 - Mix of kernels and applications, strong scaling
- NAS (NASA Advanced Supercomputing) suite
 - computational fluid dynamics kernels
- PARSEC (Princeton Application Repository for Shared Memory Computers) suite
 - Multithreaded applications using Pthreads and OpenMP

i7-960 vs. NVIDIA Tesla 280/480

	Core i7-960	GTX 280	GTX 480	Ratio 280/i7	Ratio 480/i7
Number of processing elements (cores or SMs)	4	30	15	7.5	3.8
Clock frequency (GHz)	3.2	1.3	1.4	0.41	0.44
Die size	263	576	520	2.2	2.0
Technology	Intel 45 nm	TCMS 65 nm	TCMS 40 nm	1.6	1.0
Power (chip, not module)	130	130	167	1.0	1.3
Transistors	700 M	1400 M	3100 M	2.0	4.4
Memory bandwidth (GBytes/sec)	32	141	177	4.4	5.5
Single precision SIMD width	4	8	32	2.0	8.0
Double precision SIMD width	2	1	16	0.5	8.0
Peak Single precision scalar FLOPS (GFLOP/sec)	26	117	63	4.6	2.5
Peak Single precision s SIMD FLOPS (GFLOP/Sec)	102	311 to 933	515 to 1344	3.0-9.1	6.6-13.1
(SP 1 add or multiply)	N.A.	(311)	(515)	(3.0)	(6.6)
(SP 1 instruction fused)	N.A.	(622)	(1344)	(6.1)	(13.1)
(face SP dual issue fused)	N.A.	(933)	N.A.	(9.1)	–
Peak double precision SIMD FLOPS (GFLOP/sec)	51	78	515	1.5	10.1

Fallacies

- Amdahl's Law doesn't apply to parallel computers
 - Since we can achieve linear speedup
 - But only on applications with weak scaling
- Peak performance tracks observed performance
 - Marketers like this approach!
 - But compare Xeon with others in example
 - Need to be aware of bottlenecks

Pitfalls

- Not developing the software to take account of a multiprocessor architecture
 - Example: using a single lock for a shared composite resource
 - Serializes accesses, even if they could be done in parallel
 - Use finer-granularity locking

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
- Performance per dollar and performance per Joule drive both mobile and WSC