Parallel computing — Motivation (1)

Supercomputing / High-Performance Computing

Large-scale computational problems in science and engineering:
– very large number of (floatingpoint) operations (FLOPs),
– would take too long time on a desktop PC or uniprocessor server

- Example: Weather prediction
  Navier-Stokes equations
  e.g. global weather prediction with cell size 1 mile $\times$ 1 mile $\times$ 1 mile, 200 FLOPs/cell/time step,
  10-day forecast with 10-minute time step takes ca. 10^{15} FLOPs
  $\rightarrow$ takes about 10 days on a 1-GigaFLOP/s computer
  $\rightarrow$ takes a few minutes on a 1-TeraFLOP/s computer

- Example: Quantum chromodynamics,
  e.g., computing the mass of the proton: 10^{17} FLOPs
  $\rightarrow$ about 1 year on a uniprocessor server
  $\rightarrow$ 1 day on a 1-TeraFLOP/s computer

Parallel computing — Motivation (2)

More “Grand Challenges” in large-scale scientific computing:

- Computational chemistry
  e.g., molecular dynamics simulations
- Computational physics, particle physics, astrophysics
  e.g., galaxy simulation with 10^{11} stars $\rightarrow$ ca. 1 CPU year per iteration
- Virtual Reality, graphics rendering, special effects
- Bioinformatics [CACM 47(11), 2004]
  e.g., simulating 100μs of protein folding $\approx 10^{25}$ machine instructions
  $\rightarrow$ 3 years on a PetaFLOPS system or 10^6 centuries on a 3.2 GHz PC
  IBM BlueGene, [IBM Systems J. **40**(2), 2001]
- ...

Parallel computing — Motivation (3)

Moore’s Law (technology observation, holds since the 1970’s):
#gates / chip area grow exponentially (ca. 40...50% / year)

BUT we observe ($\approx$ 2003):
- current processor design, clock rate approaching physical limits
  complexity, production cost, leakage currents, heat/power problem
- limited instruction-level parallelism in programs

The only way to high-performance computing is parallel computing!
- exploit additional (thread-level) parallelism

Will even affect mainstream computing platforms (e.g., desktop processors)
Dual-core, Quad-core, ..., Multi-core
Hardware-multithreading

So, what are parallel computers? How to program them?

Parallel computer

A parallel computer is a computer consisting of

+ two or more processors
  that can cooperate and communicate
  to solve a large problem faster,

+ one or more memory modules,

+ an interconnection network
  that connects processors with each other
  and/or with the memory modules.

Multiprocessor: tightly connected processors, e.g. shared memory

Multicomputer: more loosely connected, e.g. distributed memory

Parallel architecture concepts

Classification of parallel computer architectures:

- by control structure
- by memory organization
- by interconnection topology
- by degree of synchronous execution

Parallel architecture concepts (1)

Classification by control structure [Flynn'72]

SISD single instruction stream, single data stream

SIMD single instruction stream, multiple data streams

One clock, one program memory, one program counter.

+ vector processors
+ VLIW processors
+ array computers

MIMD multiple instruction streams, multiple data streams

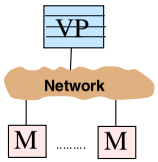
Each processor has its own program counter.

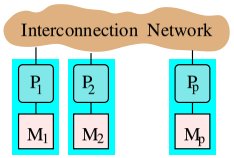
+ synchronous: central clock, central program memory (PRAM)
+ asynchronous: local clock, local program memory

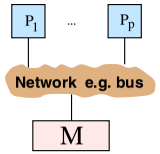
SPMD single program, multiple data

Run the same program on each processor but on different data.

Parallel architecture concepts (2)





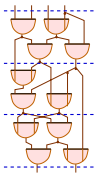


Classification by memory structure

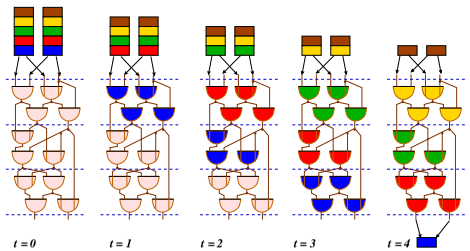
1. Vector computers	SIMD	(shared) memory (interleaved memory banks)
2. Distributed memory systems		
(a) Array computers	SIMD	distributed memory
(b) Multicomputer	MIMD	(b1) distributed (local) memory
		(b2) distributed m. with global address space (NCC-NUMA)
3. Shared memory systems		
(a) UMA (SMP)	MIMD	shared memory
(b) CC-NUMA	MIMD	distributed shared memory

Vector processor architecture (1)

Principle: SIMD + pipelining
cf. assembly line manufacturing of cars etc.



+ Idea: partition "deep" arithmetic circuits (e.g., floatingpoint-adder) into $d > 1$ horizontal layers, called stages, of about equal depth. Reduce clock cycle time such that each stage needs one cycle.
+ Intermediate results of stage k are forwarded to stage $k+1$
+ The operands and result(s) are vectors, sequences (arrays) of floats
+ All stages work simultaneously, but on different components of the vectors
+ Stage k works on l-th vector component in cycle $k+l$
+ First result available after d cycles, a startup phase of $d-1$ cycles is needed to fill the pipeline



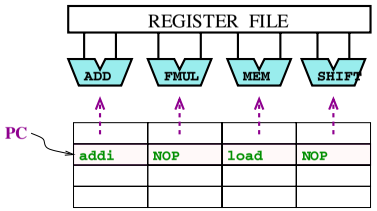
Vector processor architecture (2)

+ A vector operation, e.g. $C[1:N] \leftarrow A[1:N] + B[1:N]$ (elementwise addition) takes $N + d - 1$ cycles (compared to $N \times d$ cycles without pipelining)
+ Condition: All component computations of a vector operation must be of same operation type and independent of each other
+ Scalar operations take d cycles — no improvement.
+ Programs must be vectorized (by the programmer or compiler)
+ Some vector supercomputers: Cray 1, CDC Cyber 205, Fujitsu VP 100
+ Vector nodes in massively parallel supercomputers: Suprenum VX, NEC SX, Intel i860, Cray SV1 node
+ Problem: Memory bandwidth must match the high CPU speed.
+ In modern microprocessors, pipelining is applied at sub-instruction level.

VLIW processors

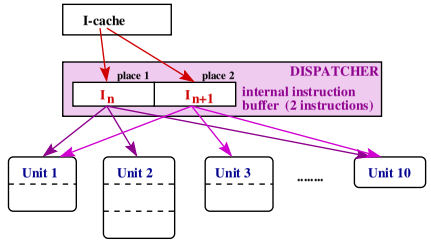
VLIW: (very long instruction word)

+ multiple functional units working in parallel
+ single stream of long instruction words with explicitly pararallel subinstructions
+ requires static scheduling ($\rightarrow$ compiler)



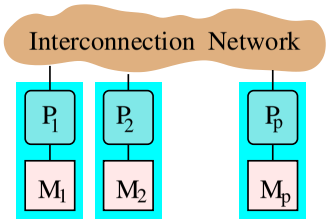
Compare to a superscalar processor:

+ multiple functional units
+ single stream of ordinary instructions (sequential program code, SISD)
+ multiple issue: k-way superscalar = up to $k > 1$ subsequent instructions may start execution simultaneously



+ dynamic instruction dispatcher maps instructions to available units

Distributed memory architectures



(a) Array computers (Systolic arrays)
+ SIMD, synchronous (common clock signal)
+ channel-based, synchronous communication to neighbor PEs
+ e.g., MasPar MP1/2, Transputer networks
+ now out of fashion

(b) Multicomputer
+ MIMD, asynchronous
+ may consist of standard PC components
+ very successful
+ e.g., Cray T3E, IBM SP, PC clusters

+ scalable $\rightarrow$ "MPP"
+ fast, exclusive access to own local memory
+ access to remote variables only via explicit message passing
+ data structures of the user program (large arrays!) must be distributed across the processors

Interconnection networks (1)

network
= physical interconnection medium (wires, switches)
+ communication protocol

(a) connecting processors with each other (DMS)

(b) connecting processors with memory modules (SMS)

direct / static interconnection networks

direct networks with hardware routers
$\rightarrow$ offload processors from most communication work

switched / dynamic interconnection networks

Interconnection networks (2): simple topologies

bus — 1 wire - bus saturation with many processors
e.g. Ethernet

linear array — **ring** — e.g. Token Ring

2D grid — **torus:**

3D grid — **3D torus** — (e.g. Cray T3E)

tree

root processor is bottleneck

Fat Tree = bandwidth-reinforced tree (e.g., TMC CM-5, SGI Altix 3700 Bx2)

Interconnection networks (3): fat tree

Example:
SGI NUMAlink^{TM}

dual fat tree

Source: M. Woodacre et al.: *The SGI Altix^{TM} 3000 Global Shared-Memory Architecture.* White Paper, SGI, 2003.
www.sgi.com

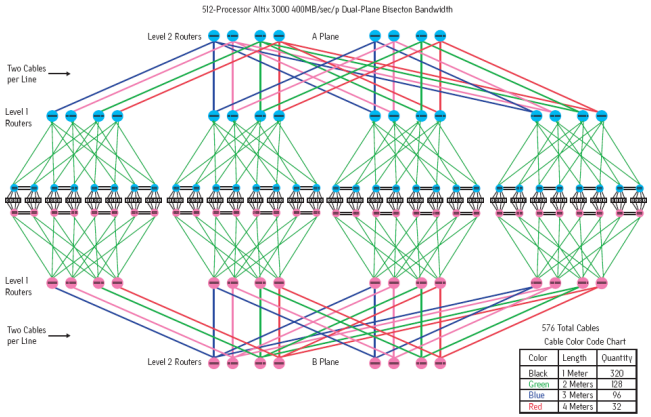
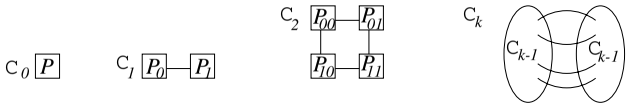


Fig. 3. 512-processor dual "fat tree" interconnect topology

Interconnection networks (4): Hypercube

k-dimensional hypercube C_k has 2^k nodes



connect corresponding (= same index) nodes of the subhypercubes C_{k-1} prefix 0 resp. 1 to node indices

– each node has degree k (= number of direct neighbors)

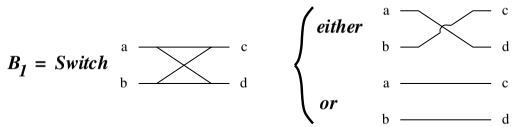
+ $\forall (p_i, p_j)\ \exists$ path of length $\leq k$ hops

shortest path $p_i \rightarrow^* p_j$ has length $HD(p_i, p_j) =$ # 1-bits in $i \oplus j$
(Hamming-distance)

+ routing: indices of neighbor nodes differ in 1 bit

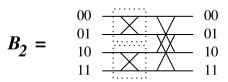
Examples: Intel iPSC (1990); extended hypercube in SGI Origin 3000

Interconnection networks (5): Butterfly network [PPP] p.86–88

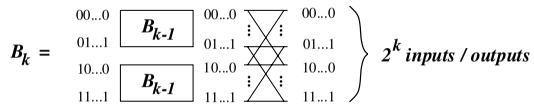


Butterfly-network topology

+ B_k connects 2^k inputs to 2^k outputs using $k2^{k-1}$ switches



+ path from each entry to each exit (and vice versa) is unique



+ routing: j-th stage switches according to target index bit $k - j$

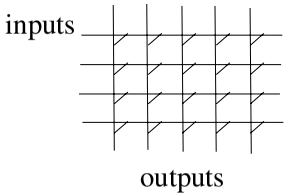
+ scalable: path length is k (logarithmic in # processors)

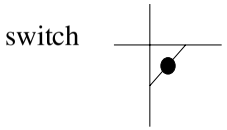
Example: SB-PRAM, Univ. Saarbrücken 2001

Interconnection networks (6): Crossbar

Crossbar

n input lines connected to n output lines by matrix of n^2 switches





status: either connected or open

+ very flexible (permutations, broadcast)

+ high throughput

– expensive (chip area, wiring)

– not scalable (typ. $n \leq 32$)

e.g. Alliant FX-8

Interconnection networks (7): hybrid networks

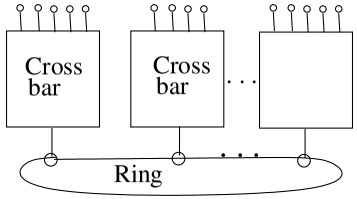
Hybrid networks as combinations of multiple network types

(hierarchical networks)

+ Hyper-Crossbar (e.g. CP-PACS),

+ Ring of crossbars

(e.g. Convex/HP Exemplar)



+ Ring of rings (e.g. KSR-1)

Interconnection network topologies (8): Graph-theoretic properties

node degree

= max. # neighbors

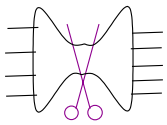
diameter

= max. distance in hops,

where 1 hop = 1 switch-stage oder 1 interprocessor connection

connectivity

node connectivity, edge connectivity $\rightarrow$ fault tolerance



bandwidth

= max. #bytes/sec. for one connection

throughput

= max. #bytes/sec. for all simultaneously active connections

Interconnection network topologies (9): Other issues

Routing deterministic or randomized

usually not accessible / not visible to the programmer

Switching	wire switching	$\leftrightarrow$	packet switching
	permanent interconnection		fixed packet format, variable route
	(cf. telephone network)		(cf. mail system)

fault tolerance

= max. damage if a switch / processor fails

scalability

= asymptotic behaviour of node degree, diameter, ... for $p \rightarrow \infty$

"massively parallel": currently about $p \geq 64$

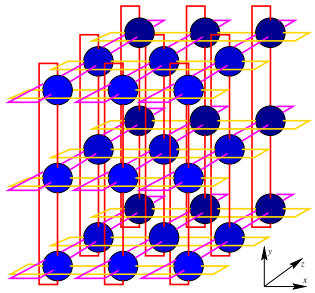
embeddability

= map the communication structure of the program (= virtual network)

suitably to the physical network topology: minimize latency, contention

Example: Cray T3E at NSC (1997-2003)

+ 256 Application PEs
  DEC (Compaq) Alpha 21164 (EV5) RISC microprocessors at 300 MHz
  up to 2 floatingpoint operations per cycle (multiply&add)
  $\rightarrow$ 600 MFLOPS peak performance per proc.



+ 13 Command PEs (log in, compile, edit)
+ 3 OS PEs (used by the operating system)
+ overall peak performance: 160.8 GFLOPS
+ 45.6 GB of memory
+ Interconnection network is a 3-D torus
+ Hardware / system support for shared address space
+ Short connections:
  e.g. 256 nodes as $4 \times 8 \times 8$ torus $\Rightarrow$ largest distance $2+4+4 = 10$ hops
+ low blocking, high bandwidth, fault tolerance

Example: Beowulf-class PC clusters

Characteristics:

- off-the-shelf (PC) nodes
  Pentium, Itanium, Opteron, Alpha;
  also (chip-)SMP ("constellation")
- commodity interconnect
  G-Ethernet, Myrinet, Infiniband, SCI
- Open Source Unix
  Linux, BSD
- Message passing computing
  MPI, PVM
  (HPF)

Advantages:

+ best price-performance ratio
+ low entry-level cost
+ vendor independent
+ scalable (today: 64..8192)
+ rapid technology tracking

T. Sterling: *The scientific workstation of the future may be a pile of PCs*,
Communications of the ACM **39**(9), Sept. 1996

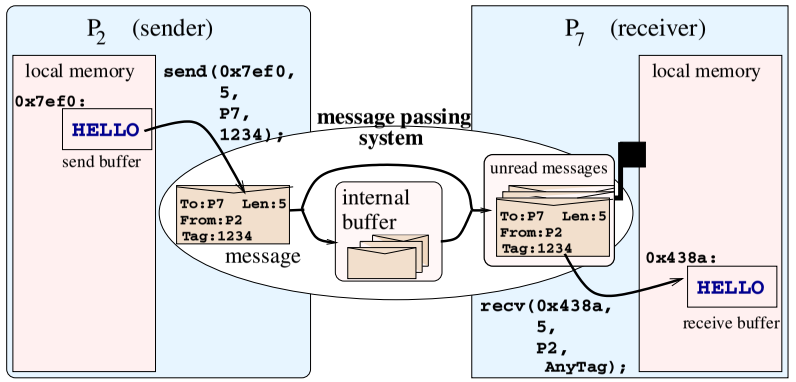
Example: Monolith

Monolith `www.nsc.liu.se/systems/monolith`

- 200 dual-processor PCs mounted in a rack
  each with 2 Intel Xeon 2.2 GHz, 2 GB main memory,
  80 GB disk memory
  $\rightarrow$ 400 processors (396 for users)
  $\rightarrow$ theoretical peak performance: 1.8 TFlops
  (achieved on LINPACK: 0.96 TFlops)

  7 TB common main disk storage
- SCI (scalable coherent interface) network with own MPI impl. (ScaMPI).
  point-to-point message latency: 4.5μs for small messages,
  point-to-point message bandwidth: 260 MB/s for large messages
- allocate processors using PBS batch queue system
- fastest supercomputer in Sweden 2002–2004:
  first entry in TOP500 in Nov 2002: rank 51, Nov 2003: rank 103

Message passing



two-sided: sender executes `send`, receiver executes `recv`
  usually: non-blocking send, blocking receive $\rightarrow$ partial synchronization

one-sided (DRMA): direct remote memory access
  sender sends access request, receiver's DMA handler executes it

Message passing (software interface): Classification

Addressing:

- two-sided, direct (sender names receiver explicitly; receiver may name sender)
- two-sided, indirect (via named channels/ports or mailboxes)
- one-sided (no explicit receive operation)
  (DRMA = direct remote memory access)

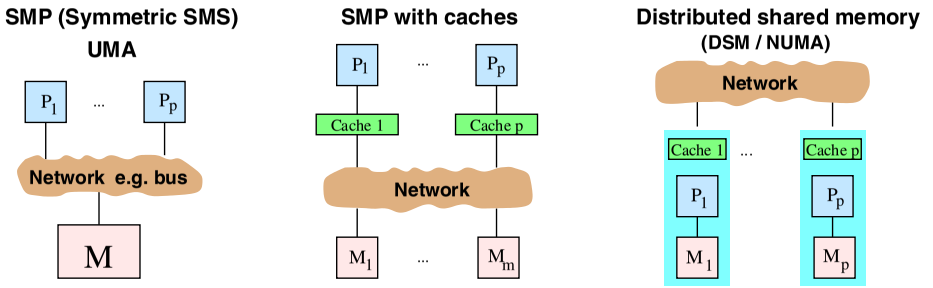
Blocking:

- asynchronous (sender injects msg into comm.-system and continues)
- synchronous (sender waits in send() until receiver receives)
- rendezvous (sender waits in send() until receive() completed)

Buffering:

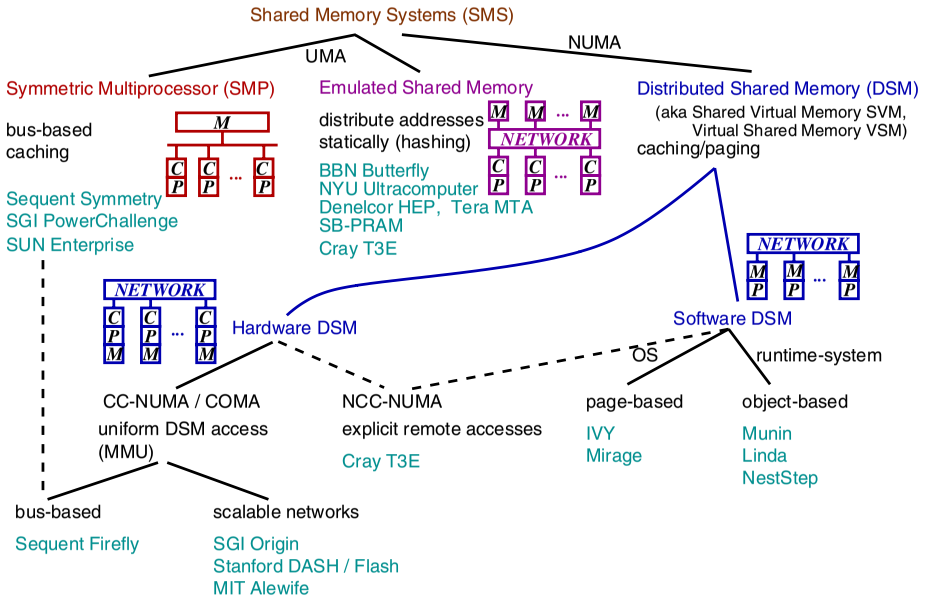
- buffered (sender writes message to system buffer, continues)
- unbuffered (sender responsible for buffering in a user data structure)

Shared memory architectures



UMA = uniform memory access time e.g., Alliant FX8, SB-PRAM, Tera MTA
NUMA = non-uniform memory access time e.g., Cray T3E (SHMEM)
CC-NUMA = cache-coherent NUMA e.g., Stanford DASH, SGI Origin
COMA = cache-only memory architecture e.g., KSR1, KSR2
DSM = distributed shared memory
VSM = virtual shared memory

Overview of shared and virtual shared memory systems



Cache issues (1)

Cache = small, fast memory (SRAM) between processor and main memory
contains copies of main memory words

cache hit = accessed word already in cache, get it fast.

cache miss = not in cache, load from main memory (slower)

Cache line size: from 16 bytes (Dash) ...
Memory page size: ... up to 8 KB (Mermaid)

Cache-based systems profit from
+ spatial access locality (access also other data in same cache line)
+ temporal access locality (access same location multiple times)
+ dynamic adaptivity of cache contents

→ suitable for applications with high (also dynamic) data locality

Cache issues (2)

Mapping memory blocks $\rightarrow$ cache lines / page frames:

- direct mapped: $\forall j \; \exists! i : \; B_j \mapsto C_i$, namely where $i \equiv j \bmod m$.
- fully-associative: any memory block may be placed in any cache line
- set-associative

Replacement strategies (for fully- and set-associative caches)

- LRU least-recently used
- LFU least-frequently used
- ...

Cache issues (3): Memory update strategies

(still considering single-processor system with cache)

Write-through

+ consistency
– slow, write stall ($\rightarrow$ write buffer)

Write-back

+ update only cache entry
+ write back to memory only when replacing cache line
+ write only if modified, marked by “dirty” bit for each C_i
– not consistent,
DMA access (I/O, other procs) may access stale values
$\rightarrow$ must be protected by OS, write back on request

Cache coherence and Memory consistency

Caching of shared variables leads to consistency problems.

A cache management system is called coherent
if a read access to a (shared) memory location x reproduces always the value corresponding to the most recent write access to x.

$\rightarrow$ no access to stale values

A memory system is consistent (at a certain time)
if all copies of shared variables in the main memory and in the caches are identical.

Permanent cache-consistency implies cache-coherence.

For performance reasons, weaker consistency models have been developed [Gharachorloo/Adve’96]

Cache coherence protocols

Inconsistencies occur when modifying only the copy of a shared variable in a cache, not in the main memory and all other caches where it is held.

Write-update protocol
At a write access, all other copies in the system must be updated as well.
Updating must be finished before the next access.

Write-invalidate protocol
Before modifying a copy in a cache,
all other copies in the system must be declared as “invalid”.

Most cache-based SMPs use a write-invalidate protocol.

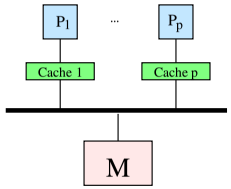
Updating / invalidating straightforward in bus-based systems (bus-snooping)

otherwise, a directory mechanism is necessary

Cache-based SMP: Bus-Snooping

For bus-based SMP with caches and write-through strategy.

All relevant memory accesses go via the central bus.



Cache-controller of each processor listens to addresses on the bus:

write access to main memory is recognized
and committed to the own cache.

– bus is performance bottleneck → poor scalability

CC-NUMA: Directory-protocols for non-bus-based systems

No central medium:

(a) → no cache coherence (e.g. Cray T3E)

(b) → directory lookup

Directory keeps the copy set for each cache line / memory block

e.g. stored as bitvectors

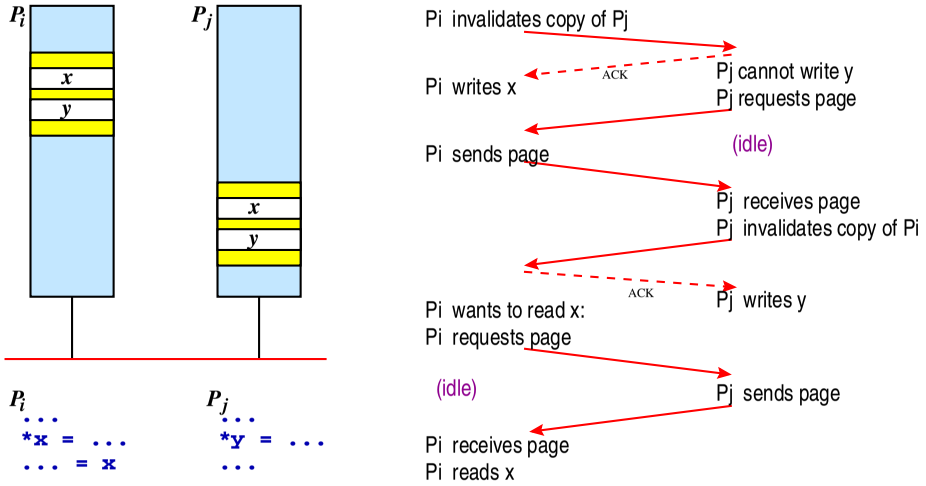
- 1 presence bit per processor
- status bits

  e.g. dirty-bit for the status of the main memory copy

See e.g. [Culler'98, Ch. 8]

DSM problem: False sharing

False sharing in cache- or page-based DSM systems



Cache lines / pages treated as units → sequentialization, thrashing

DSM problem: False sharing (cont.)

How to avoid false sharing?

Smaller cache lines / pages

→ false sharing less probable, but

→ more administrative effort

Programmer or compiler gives hints for data placement

→ more complicated

Time slices for exclusive use:

each page stays for $\geq d$ time units at one processor — Mirage

How to reduce performance penalty of false sharing?

Use weaker consistency models

Example: SGI 3800 – Hardware structure

CC-NUMA architecture

- 128 MIPS 14000 RISC processors:
  - 500 MHz, 4-way superscalar,
  - 2 floatingpoint units

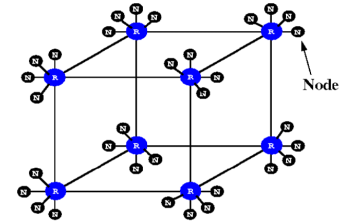
  → 1 GFLOPS peak performance

  → total: 128 GFLOPS peak performance
  - 1 GByte main memory

  → aggregate: 128 GB main memory
  - 8 MByte cache memory (Level-2, off-chip)

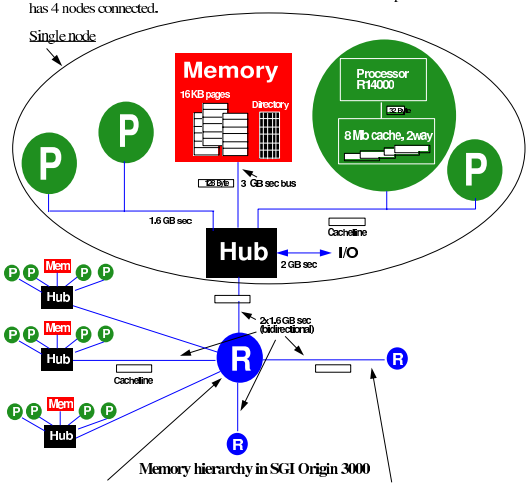
  → aggregate: 1 GB cache
- Cache line size 128 bytes
- Latency from memory to cache: 290...440 ns (dep. on where memory is)

SGI 3800 – Node structure



128–processor (32 nodes) Origin 3000

A single node consists of 4 processors, memory and a device called the hub. The hub manages each processor's access to the memory and I/O of the local as well as of a remote node. Routers control the information flow between multiple hubs. Each router has 4 nodes connected.



Memory hierarchy in SGI Origin 3000

Routers control the information flow between multiple hubs.

Hubs and routers communicate via a bidirectional link with a raw bandwidth of 1.6 GB/s in each direction.

Memory is accessed in units of *cache–lines* = 128 bytes. Directories on each memory keeps track of all cache–lines from that memory. The CPU operates on cache lines stored in its cache. The 'cache coherence protocol' ensures that the caches of all CPUs reflect the true state of the memory.

Memory latency for a local memory access is 180 nsec; for remote access it varies between 290–440 nsec.

SGI 3800 – Programming environment

+ Trusted IRIX MLS (Multi Level Security) operating system
+ LSF batch queue system
+ MPI, PVM, SHMEM and OpenMP for parallelization
+ C, C++, Fortran 77, Fortran 90
+ Linear algebra packages

  BLAS 1, 2, 3, EISPACK, LINPACK, LAPACK, FFT, ...
+ *lsload*, *lsmon*, *xlsmon* tools display load information
+ *perfex* performance analysis tool, *speedshop* profiling
+ *vampir* MPI program execution and performance visualizer
+ *dbx*, (*totalview*) debuggers
+ SecurID cards required for access

Chip multiprocessors (1)

Superscalar/VLIW processor technology has reached its limits:

- limited instruction-level parallelism in applications

  → no gain in adding even more functional units
- limited clock frequency, due to power consumption / heat dissipation
- limited chip area that can be reached within one clock cycle
- but "Moore's Law" will continue for exponential growth of chip area

Consequence:

→ increase throughput by multiprocessing on chip

→ put (L2-cache) memory on chip

Strategies for chip multiprocessors:

- Hardware multithreading SMP
- Multi-core processors SMP
- Processor-in-memory (PIM) DMS/NUMA

Multithreading and Simultaneous Multithreading / Hyperthreading

Hardware multithreading divides a physical CPU in 2 (or more) virtual CPUs

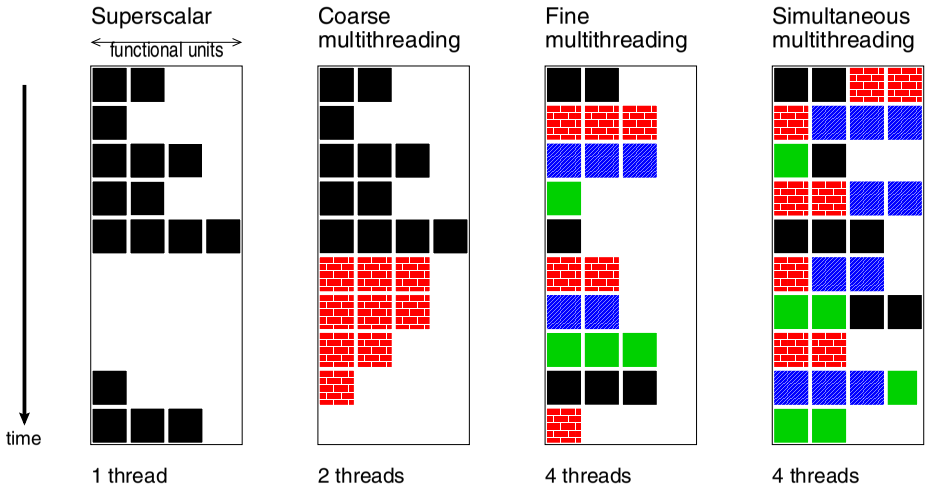
- OS and application see a dual-processor SMP system
- Each thread (virtual CPU) has its own context:
  private register set, PC, status register
- The virtual CPUs share the functional units

Multithreading techniques:

- Coarse-grain multithreading (e.g., switch on LOAD / cache miss)
- Cycle-by-cycle interleaving (e.g. SBPRAM)
- SMT/Hyperthreading (e.g. Intel Xeon (2002), later Pentium 4)

Multithreading and Simultaneous Multithreading / Hyperthreading

Multiple threads share the functional units of a single processor



Multi-Core Processors

Two or more complete processor cores (plus caches) fit on one chip.

- typically: separate L1 caches, shared L2 cache and bus interface
- often combined with hardware multithreading in each core
- often several of these on a blade server

Examples:

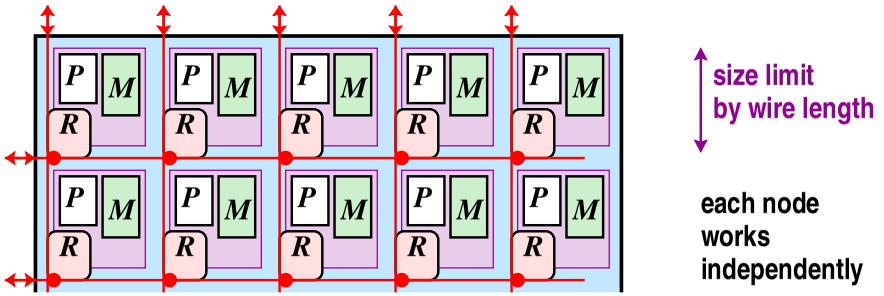
IBM POWER4/POWER5, AMD Opteron/Athlon, Sun Ultrasparc T1 Niagara, Intel CoreDuo/Xeon, HP PA-8800, IBM/Sony/Toshiba CELL, ...
(virtually any new desktop/server processor after 2005)

By ca. 2012, expect hundreds of cores per processor chip!

Chip multiprocessors (3): Processor-in-memory (PIM)

Trend: $\uparrow$ chip density $> 10^9$ gates/mm^2, $\rightarrow$ clock rate, $\uparrow$ wire delays

- Idea: Put also memory modules and a scalable network on a chip
- NUMA architecture



Summary: Current categories of supercomputer architectures

MPP's – massively parallel multiprocessors
typically: MIMD, distributed memory, special hardware, $p \gg 64$
Cray T3E, IBM SP-2, Cray XD1, ... → expensive

Clusters – NOW, COW, Beowulf cluster
MIMD, distributed memory, $p \leq 512$ and more → replaced classical MPPs

SMS – shared memory multiprocessors
typically: MIMD, cache-based, $p \leq 256$
symmetric SMS (SMP): bus / crossbar network, $p \leq 64$
SMP servers
"scalable" CC-NUMA: SGI Origin 3000, SUN Enterprise 4000, ...
Chip multiprocessors (multithreaded, multicore)

Constellations – Clusters with few but heavyweight nodes

Summary: Currently successful concepts in supercomputer architectures

use off-the-shelf components wherever possible!

- standard processors (Itanium, Opteron, PowerPC, ...)
  → participate in the steady improvements in microprocessor technology
  → cheaper than self-designed processors (esp., time-to-market!)
- simple, scalable interconnection networks, standard protocols
  → flexible machine sizes, easy upgrading
- hardware multithreading
- memory hierarchy to speed up memory access (caches)
  exploits spatial and temporal locality in memory accesses of the program
- system support for a shared address space or (virtual) shared memory
- standard languages, standard programming environments
  Fortran, C, PVM / MPI, pthreads, HPF / OpenMP
- standard OS UNIX-derivates, Linux

TOP-500 list (1) http://www.top500.org/

Rank	Site	Computer	Processors	Year	R_{max}	R_{peak}
1	DOE/NNSA/LLNL United States	BlueGene/L - eServer Blue Gene Solution IBM	131072	2005	280600	367000
2	NNSA/Sandia National Laboratories United States	Red Storm - Sandia/ Cray Red Storm, Opteron 2.4 GHz dual core Cray Inc.	26544	2006	101400	127411
3	IBM Thomas J. Watson Research Center United States	BGW - eServer Blue Gene Solution IBM	40960	2005	91290	114688
4	DOE/NNSA/LLNL United States	ASC Purple - eServer pSeries p5 575 1.9 GHz IBM	12208	2006	75760	92781
5	Barcelona Supercomputing Center Spain	MareNostrum - BladeCenter JS21 Cluster, PPC 970, 2.3 GHz, Myrinet IBM	10240	2006	62630	94208
6	NNSA/Sandia National Laboratories United States	Thunderbird - PowerEdge 1850, 3.6 GHz, Infiniband Dell	9024	2006	53000	64972.8
7	Commissariat a l'Energie Atomique (CEA) France	Tera-10 - NovaScale 5160, Itanium2 1.6 GHz, Quadrics Bull SA	9968	2006	52840	63795.2
8	NASA/Ames Research Center/NAS United States	Columbia - SGI Altix 1.5 GHz, Voltaire Infiniband SGI	10160	2004	51870	60960
9	GSIC Center, Tokyo Institute of Technology Japan	TSUBAME Grid Cluster - Sun Fire x4600 Cluster, Opteron 2.4/2.6 GHz and ClearSpeed Accelerator, Infiniband NEC/Sun	11088	2006	47380	82124.8
10	Oak Ridge National Laboratory United States	Jaguar - Cray XT3, 2.6 GHz dual Core Cray Inc.	10424	2006	43480	54204.8

...

List of the world-wide 500 most powerful supercomputer installations

- updated and published twice a year (June and November)
- sorted according to runtime of the LINPACK benchmark program

November 2006:

No swedish supercomputer in the TOP500 :-(

fastest in Linköping: Monolith (Rank 51 when introduced 2002)

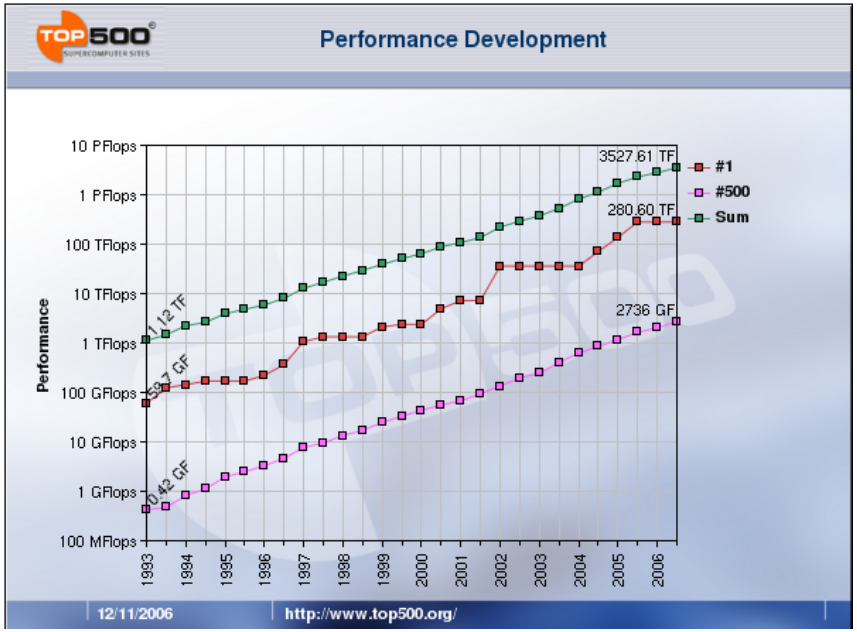
Top-500 list (2)

+ ranking based on system performance on a real application program
takes also memory bandwidth, cache sizes, compiler etc. into account,
is more fair than ranking by pure peak performance

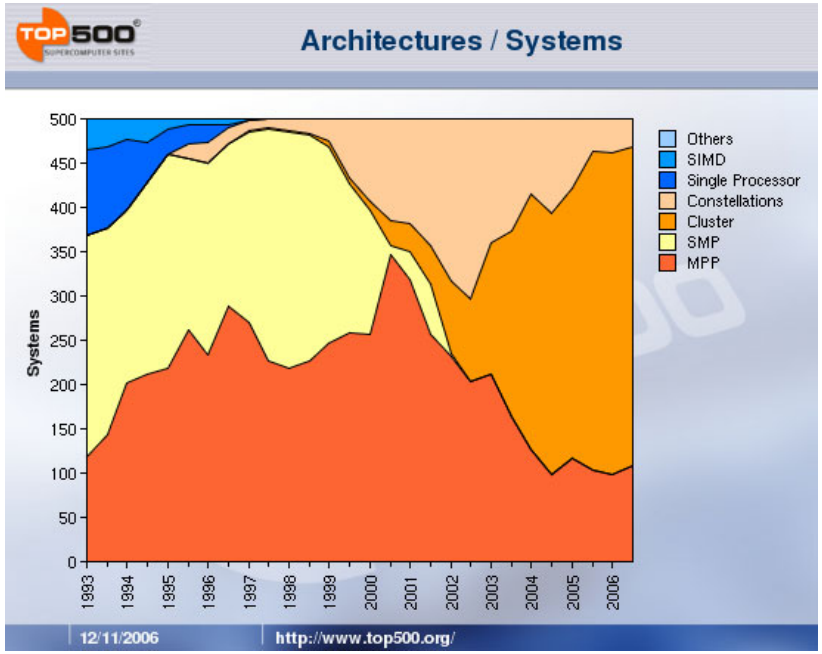
– LINPACK = solving a 1000×1000 dense linear equation system
is a good-natured, regular application:
+ high degree of parallelism,
+ $O(n^3)$ computation on $O(n^2)$ data
+ much more computation than communication
+ high data locality

⇒ ranking says nothing about performance on irregular applications!

TOP-500 list (3): Performance development



TOP-500 list (4): Trends in parallel architectures



Grid computing

\+ distributed computing over a wide-area network
\+ multiple administrative domains participate
\+ often for special-purpose applications

Computational grids

clusters of clusters, VPN backbone → virtual supercomputer center
central access control mechanism, scheduler

Peer-to-Peer computing

self-organizing networks of end-user devices
internet-based, no central administration
file-sharing or cycle-sharing

Data grids

distributed database (usually for special-purpose data)

Web services

Computational grids

- connect multiple parallel machines to a huge virtual supercomputer
- coordinated use of geographically distributed resources
- heterogeneous hardware but uniform software layer e.g. MPI, RPC
- users have access to massive computation power on demand
  pay for service instead of investment
  don't care about where the work is done
  → analogy to electrical power grid
- requires performance-portable parallel programs
  good-natured task farming applications
- actual execution platform (structure, processors, network, parameters...)
  not visible to the end user
- Some projects: Legion, GLOBUS, Condor, NorduGrid, SweGrid, ...
  `www.gridforum.org`

Peer-to-peer computing

- Internet as interconnection network
- gather unused cycles in millions of end-user PCs/workstations:
  run a task of a supercomputing application as screensaver
- Example: SETI@home, 2002: $\approx$ 400000 machines, average 26 TFlops
  (search for extraterrestrial intelligence)
- "Peer-to-peer computing" (P2P)
  comp. equivalent to P2P file sharing systems (Napster, Gnutella, ...)
- highly heterogeneous and dynamic
- commercial aspects versus voluntary cooperation
- security problem (application security vs. host security)

Summary: Parallel computer architecture concepts

For solving large problems in high-performance computing, we need parallel computer systems with thousands to millions of processors.

Parallel computer architecture classification:
- control organization: SIMD, MIMD
- memory organization: shared memory, distributed memory
- interconnection topology

Parallel computer architecture classes considered:
- SIMD and instruction-level parallelism: pipelining, VLIW
- MIMD shared memory (SMP, CC-NUMA, multithreading, CMP)
- MIMD distributed memory, message passing (Beowulf clusters)
- MIMD distributed, loosely coupled (computational grids, comp. P2P)