Overview of High Performance Computing

Jan Thorbecke
Cray
Application Engineer
janth@xs4all.nl

26 September 2002

Contents

- HPC architectures [1]
- Vector architectures
- Clusters and their interconnects
- HPC software tools
- Classification of applications
- Cray HPC project

[1] Most of this material is based on the overview of Aad van der Steen.

HPC architectures

HPC definitions

HPC architectures

HPC definitions

"High Performance Computing encompasses a collection of hardware systems, software tools, languages and generic programming approaches which make previously un feasible applications possible and which is available at an appropriate price."

HPC architectures

HPC definitions

"High Performance Computing encompasses a collection of hardware systems, software tools, languages and generic programming approaches which make previously un feasible applications possible and which is available at an appropriate price."

"A branch of computer science that concentrates on developing sup
software to run on supercomputers. A main area of this disci
parallel processing algorithms and software: programs that can
pieces so that each piece can be executed simultaneously by

HPC architectures

HPC definitions

"High Performance Computing encompasses a collection of hardware systems, software tools, languages and generic programming approaches which make previously un feasible applications possible and which is available at an appropriate price."

"A branch of computer science that concentrates on developing sup
software to run on supercomputers. A main area of this disci
parallel processing algorithms and software: programs that can
pieces so that each piece can be executed simultaneously by

„Encompasses advanced computing, communications, and information technologies, including scientific workstations, supercomputer systems, high speed networks, special purpose and experimental systems, the new generation o
large scale parallel systems, and applications and syste
software with all components well integrated and linked
high speed network

HPC architectures

HPC definitions

HPC architectures

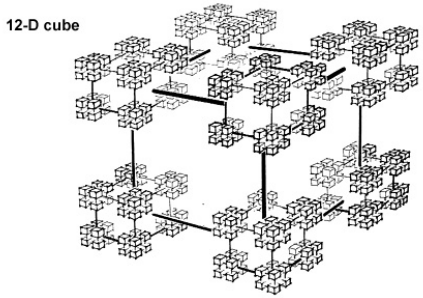
Flynn's classification of High Performance Computers is based on instruction- and data-streams into a processor:

- SISD: Single Instruction Single Data Stream
  most workstations
- SIMD: Single Instruction Multiple Data Stream
  vector-processors act on arrays of data, and processors array
- MISD: Multiple Instruction Single Data Stream
  not yet commercially build
- MIMD: Multiple Instruction Multiple Data Stream
  each processors fetches its own instructions and operates on its own data
  - ⋆ Shared Memory[2]: address-space is shared, (N)UMA
  - ⋆ Distributed Memory: every CPU has its own address-space

[2] virtual shared-memory programming models are able to address the whole collective address space on physically distributed memory systems.

HPC architectures: SIMD

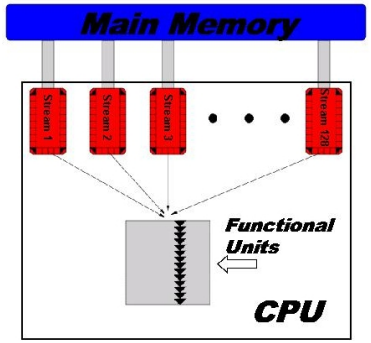
Thinking Machines CM 2



- 12-D hypercube is a cube of 9-D hypercubes: 4096 chips
- processors were grouped 16 to a chip
- 65,536 1-bit processors that simultaneously perform the same calculation
- Operations on larger data elements (32-bit floats) required one cycle per bit
- local memory of 4K bits

HPC architectures: SIMD

Cray MTA



- multithreaded machine (eliminate useless cycles)
- 1-256 CPU's
- memory is physically distributed and globally addressable
- each processor is capable of storing the state of 128 separate threads.

announced that a 40-processor, all-CMOS Cray MTA-2(TM) system with a revolutionary multithreaded architecture has completed customer acceptance

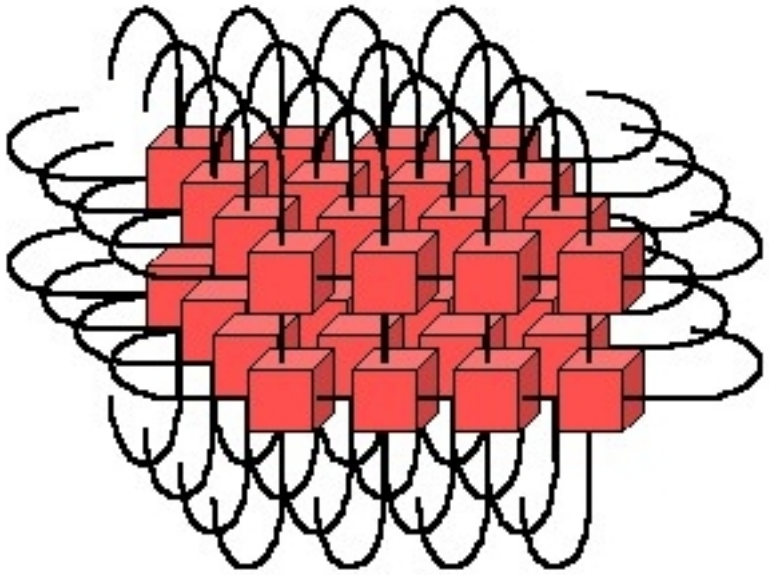
testing at the Washington, D.C. facilities of the Naval Research Laboratory (NRL).

Cray provided the Cray MTA-2 system to NRL under a contract with Northrop Grumman Information Technology, a premier provider of advanced IT solutions, engineering and business services for government and commercial businesses.

NRL is a leading-edge Distributed Center within the Department of Defense's High Performance Computing Modernization Program. NRL's mission is to explore and evaluate innovative computing and networking technologies.

HPC architectures: DM-MIMD

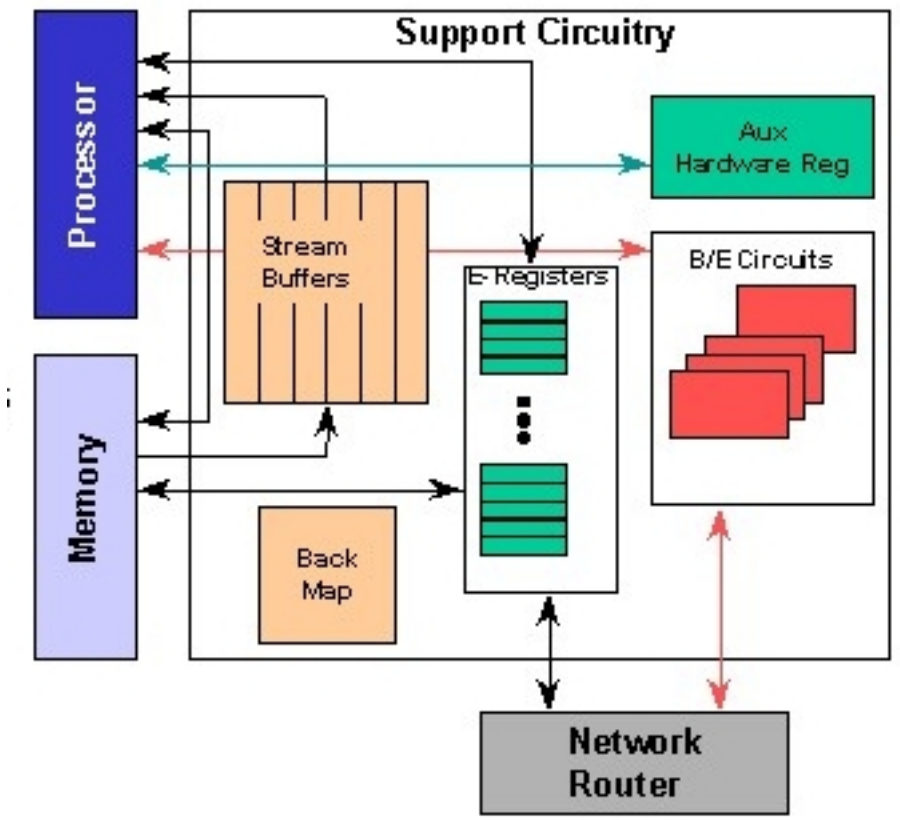
Cray T3E:



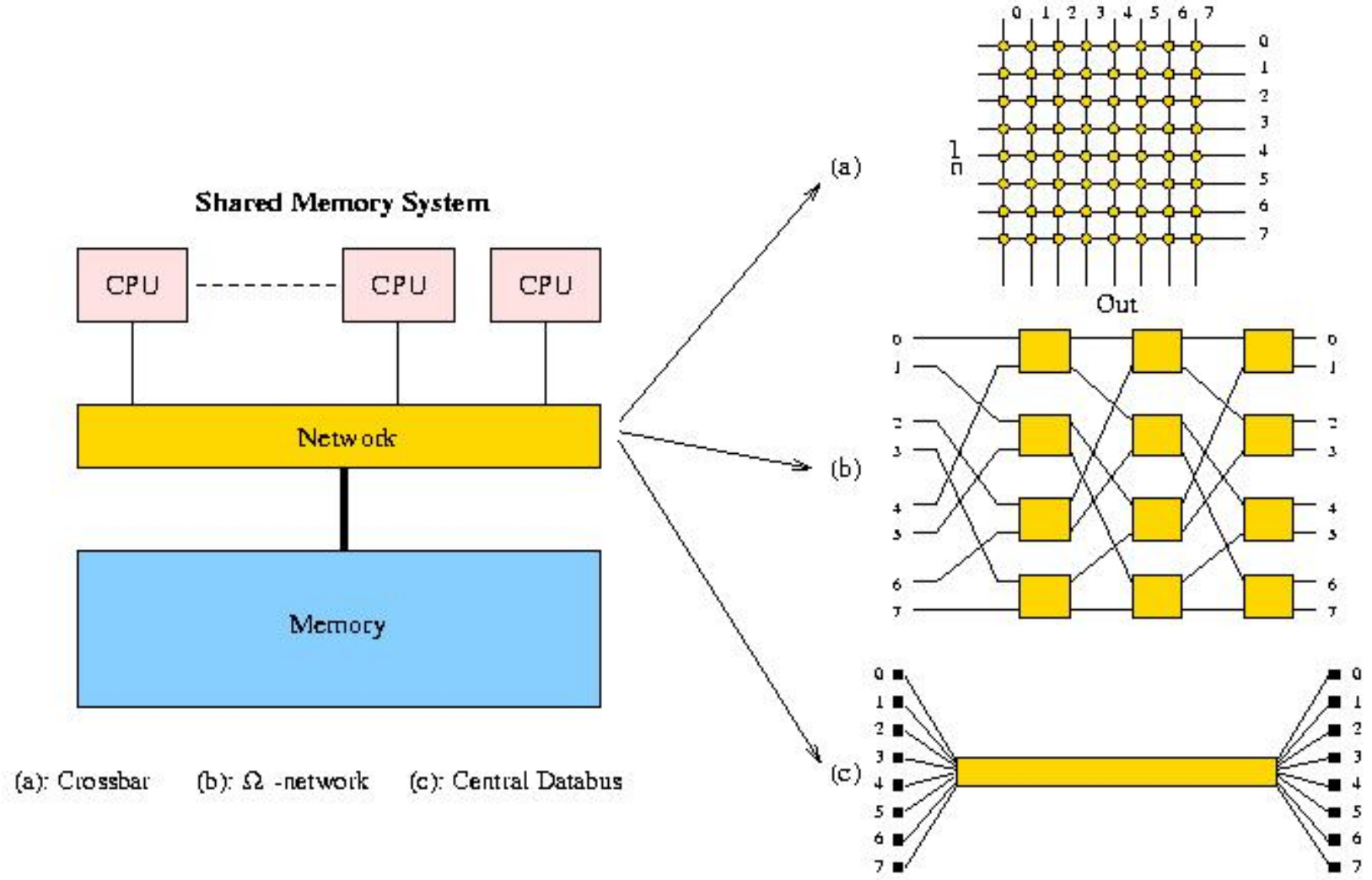
- bi-directional 3D torus (480 MB/s each direction)
- E-registers
- adaptive routing: source and destination node coordinates

HPC architectures: DM-MIMD

Cray T3E Processor (EV-5, EV-6)



HPC architectures: SM-MIMD



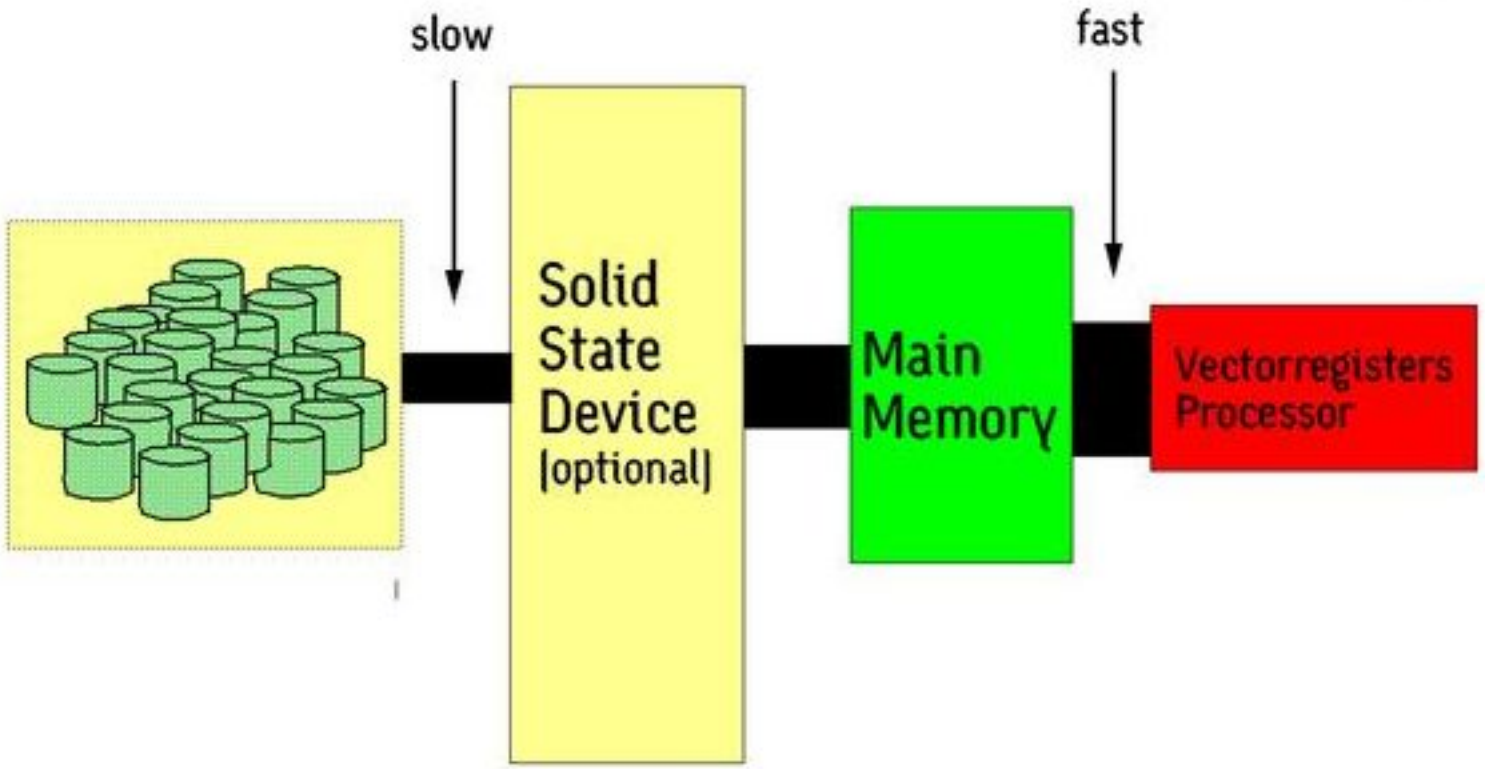
HPC architectures: SM-MIMD

Most important differences in organization of memory

- Uniform Memory Access
- Non-Uniform Memory Access
- Cache Coherency

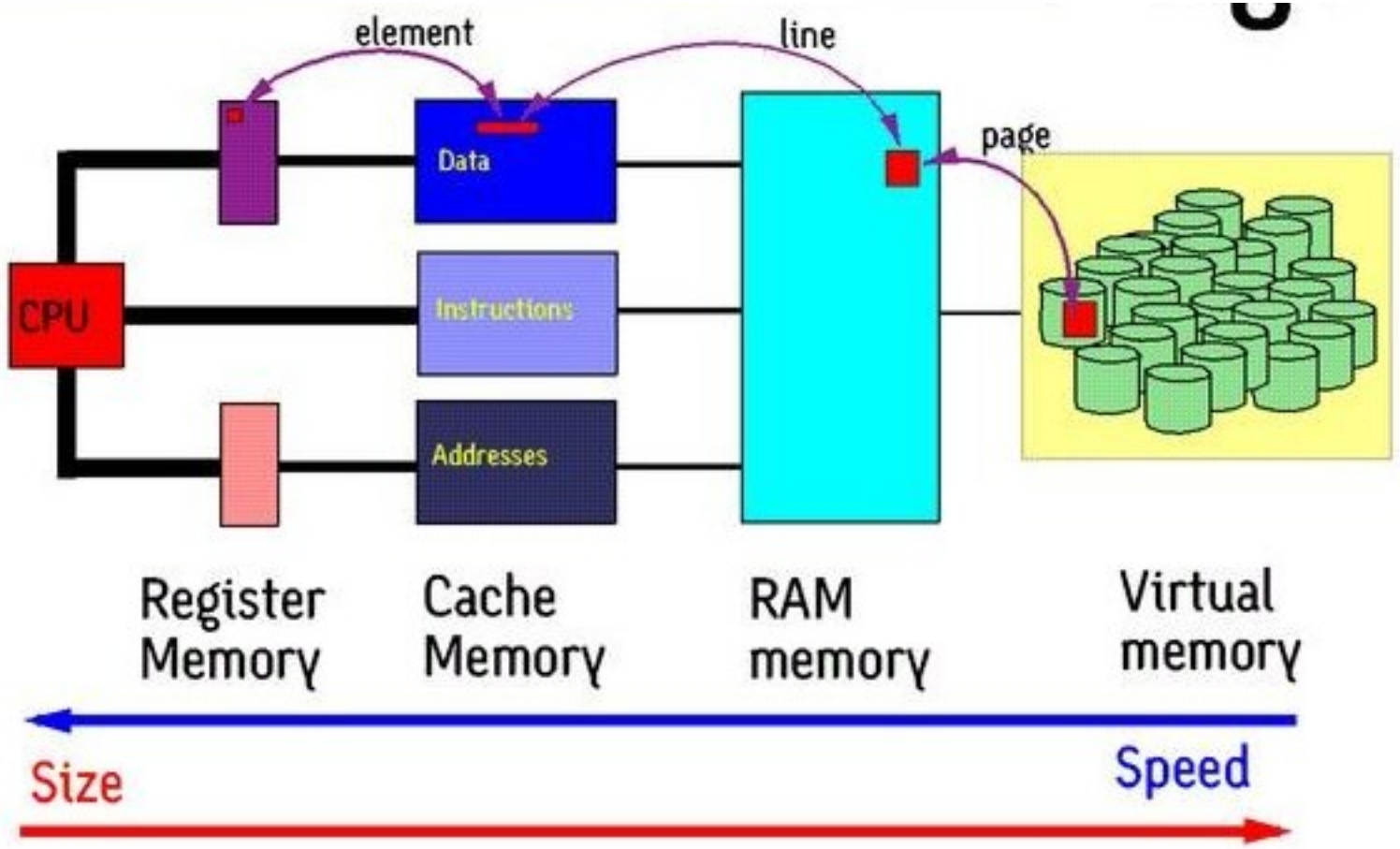
HPC architectures: SM-MIMD UMA

Uniform Memory Access



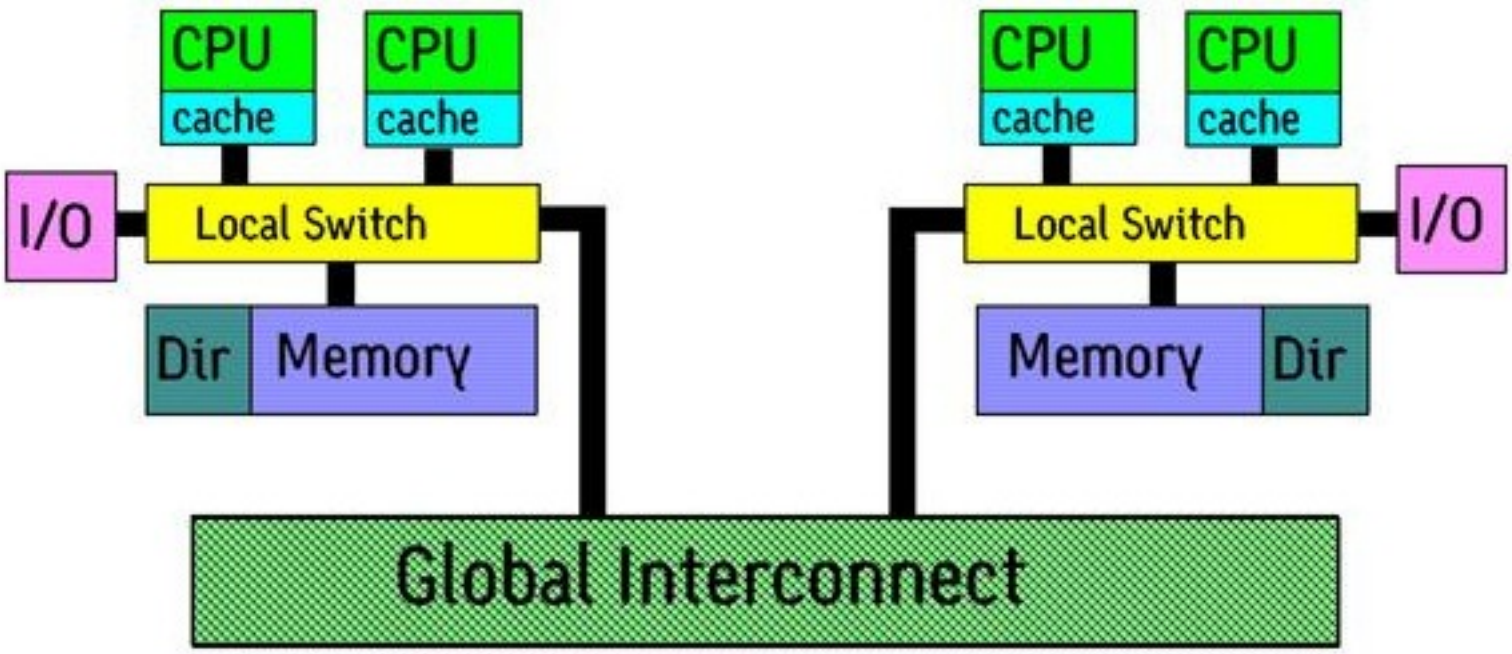
HPC architectures: SM-MIMD NUMA

Non-Uniform Memory Access



HPC architectures: SM-MIMD NUMA

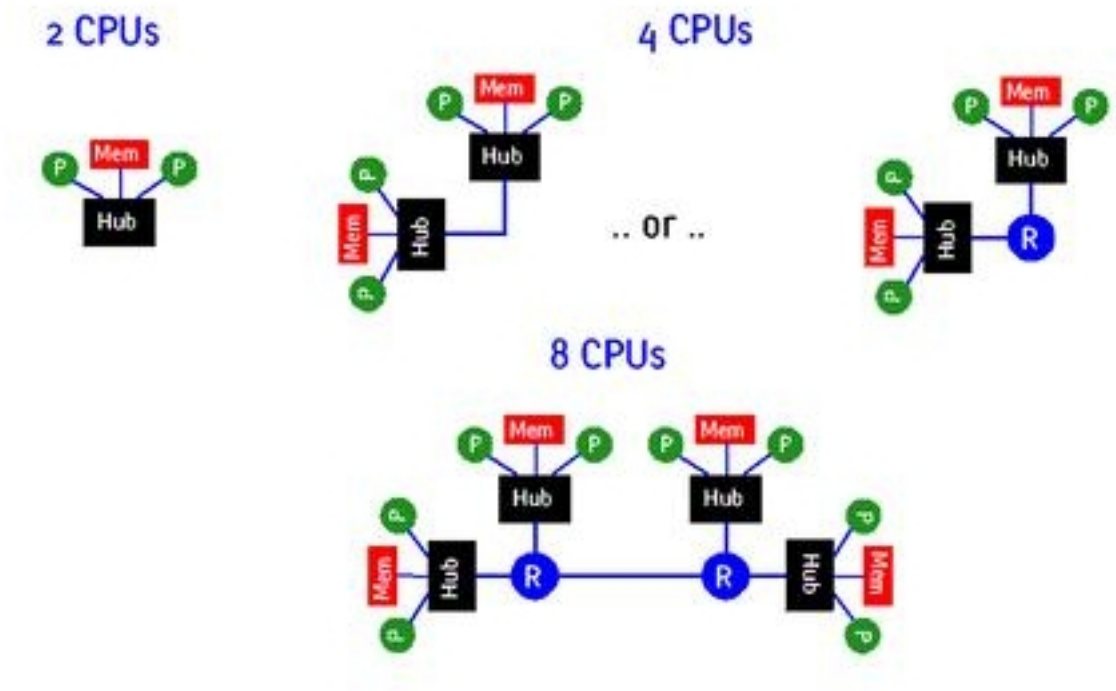
Origin 3000 building block



Distrubuted shared memory is a model, for the user this is a shared memory system.

HPC architectures: SM-MIMD NUMA

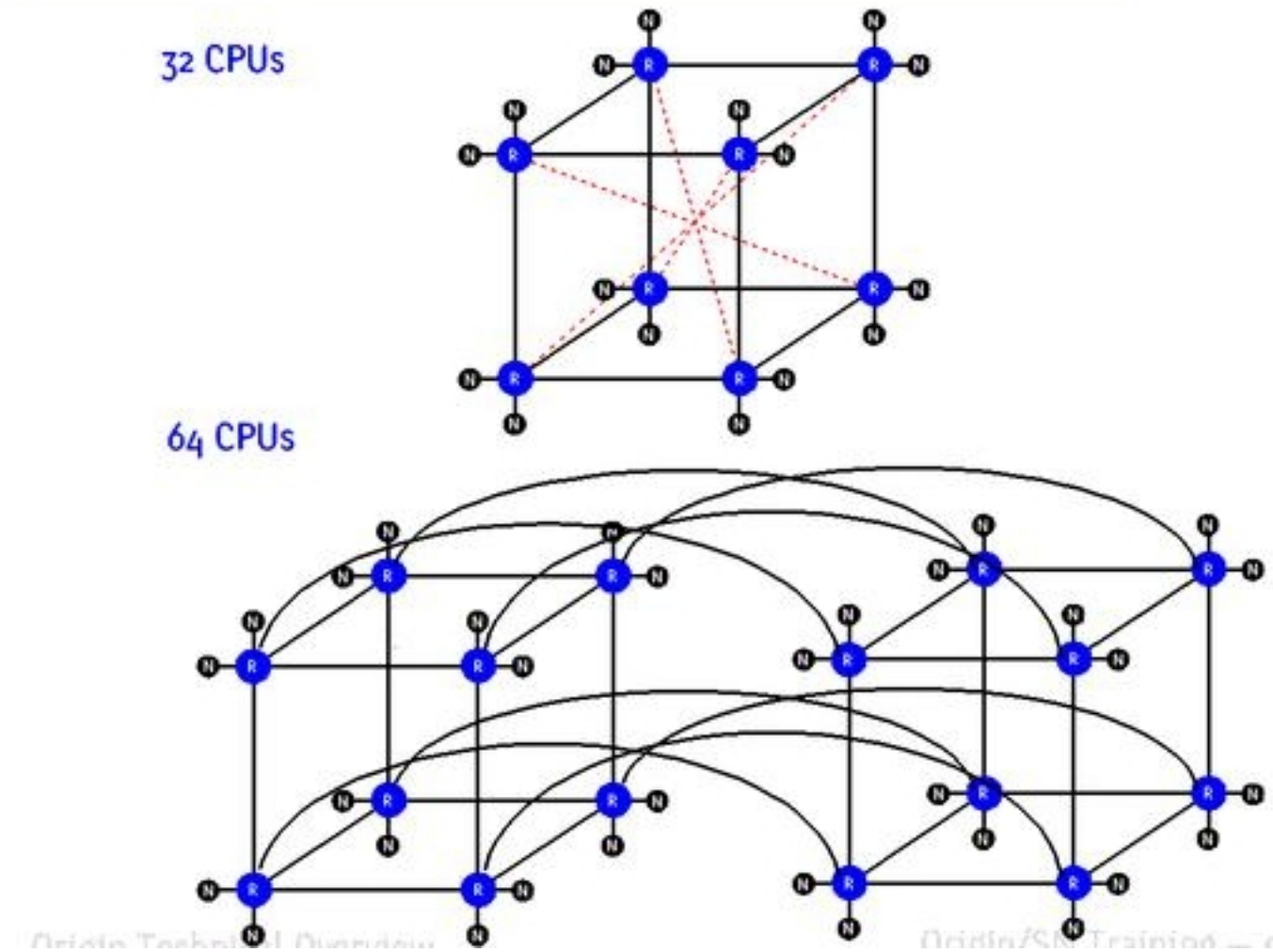
Origin 3000 building systems

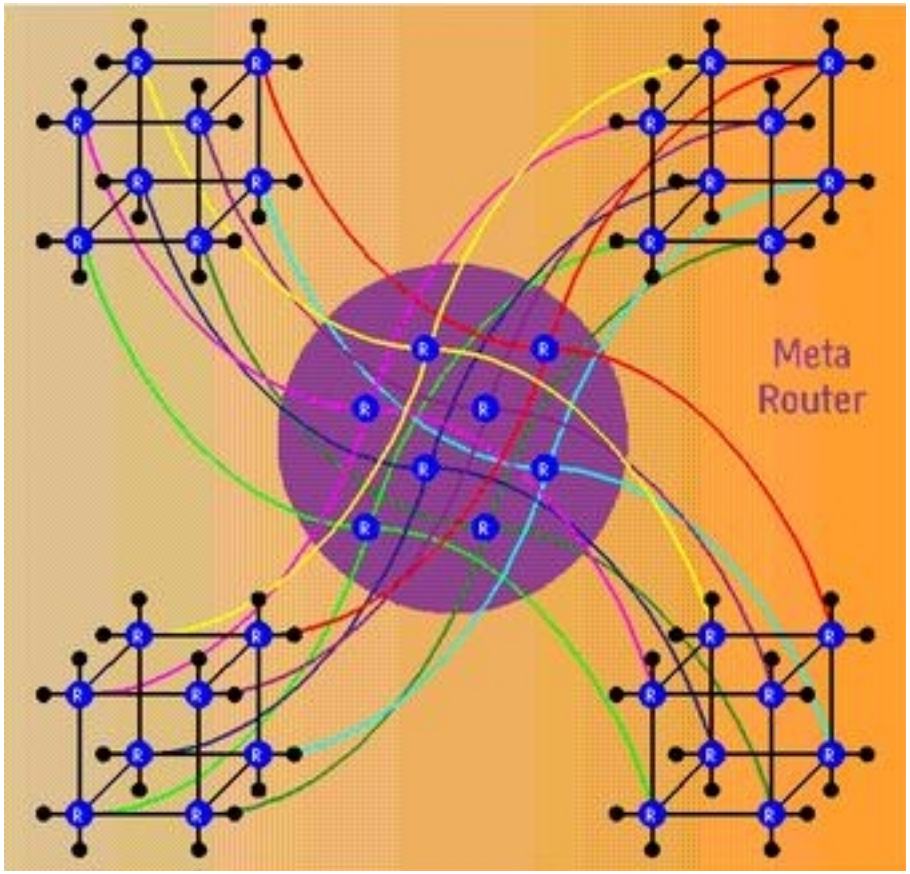


Evert Hub and Router add bandwidth.

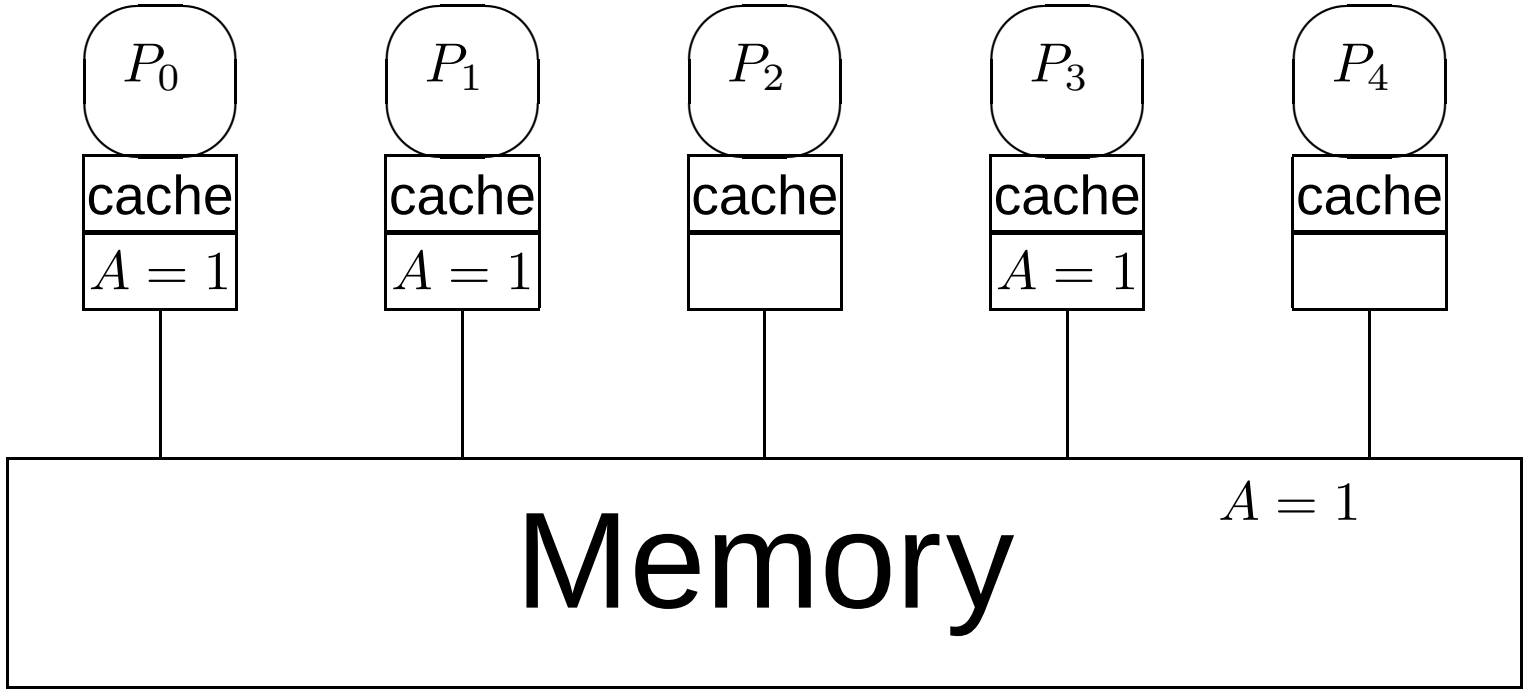
HPC architectures: SM-MIMD NUMA

Origin 3000 building systems

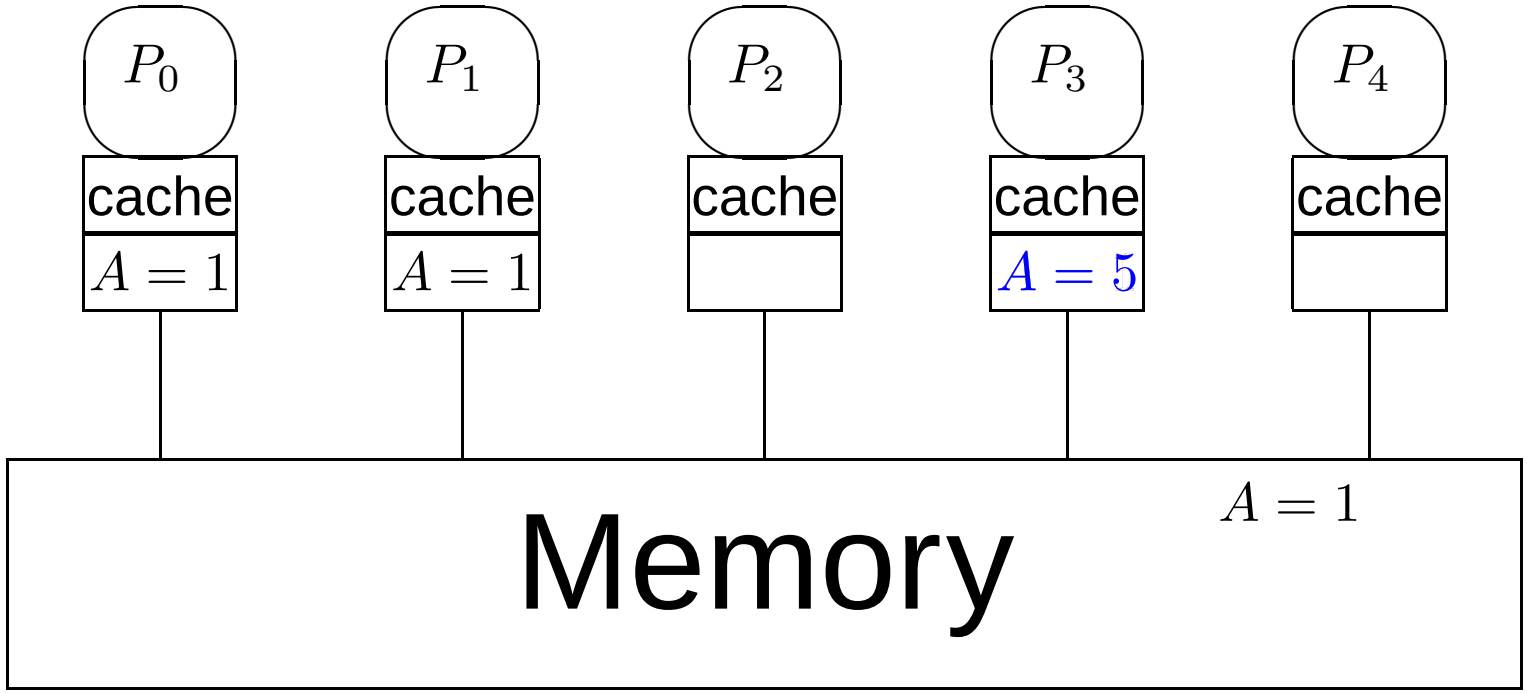



Meta
Router

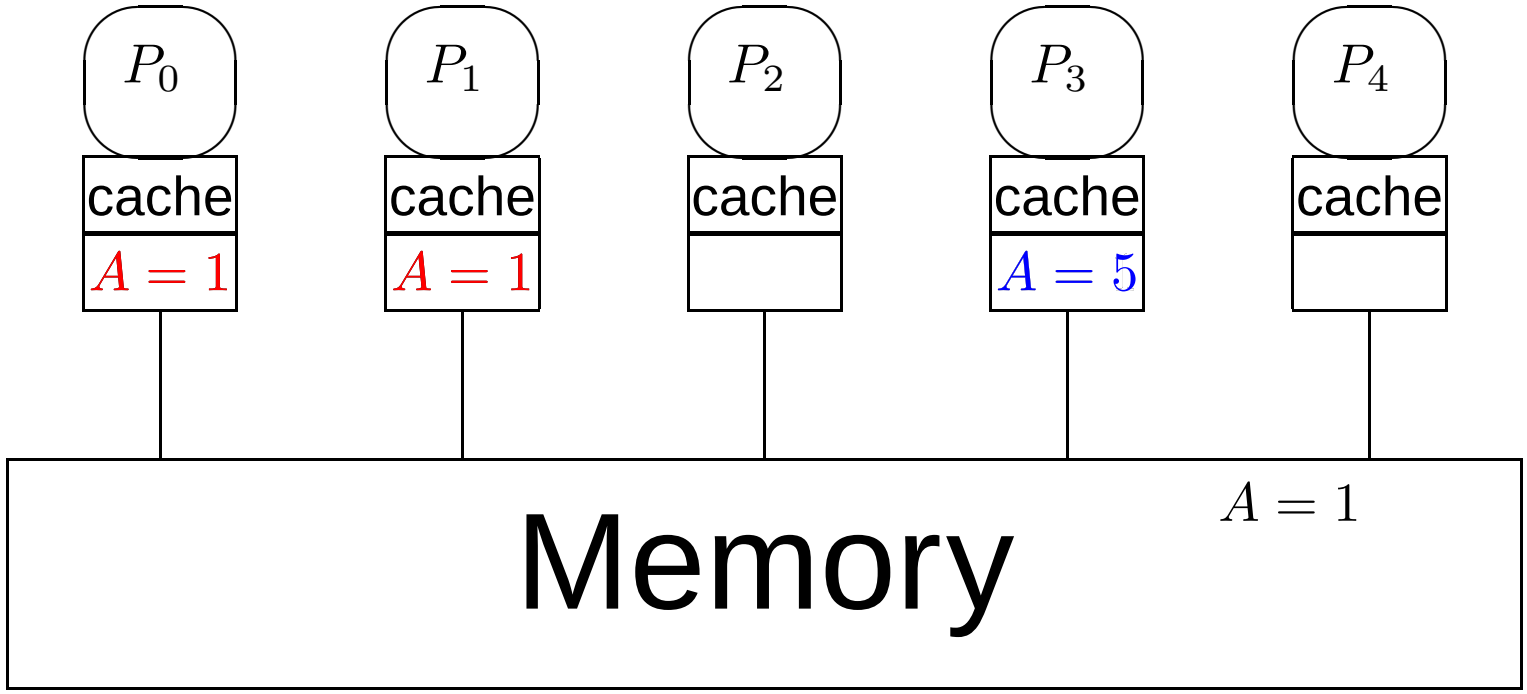
HPC architectures: Cache Coherency



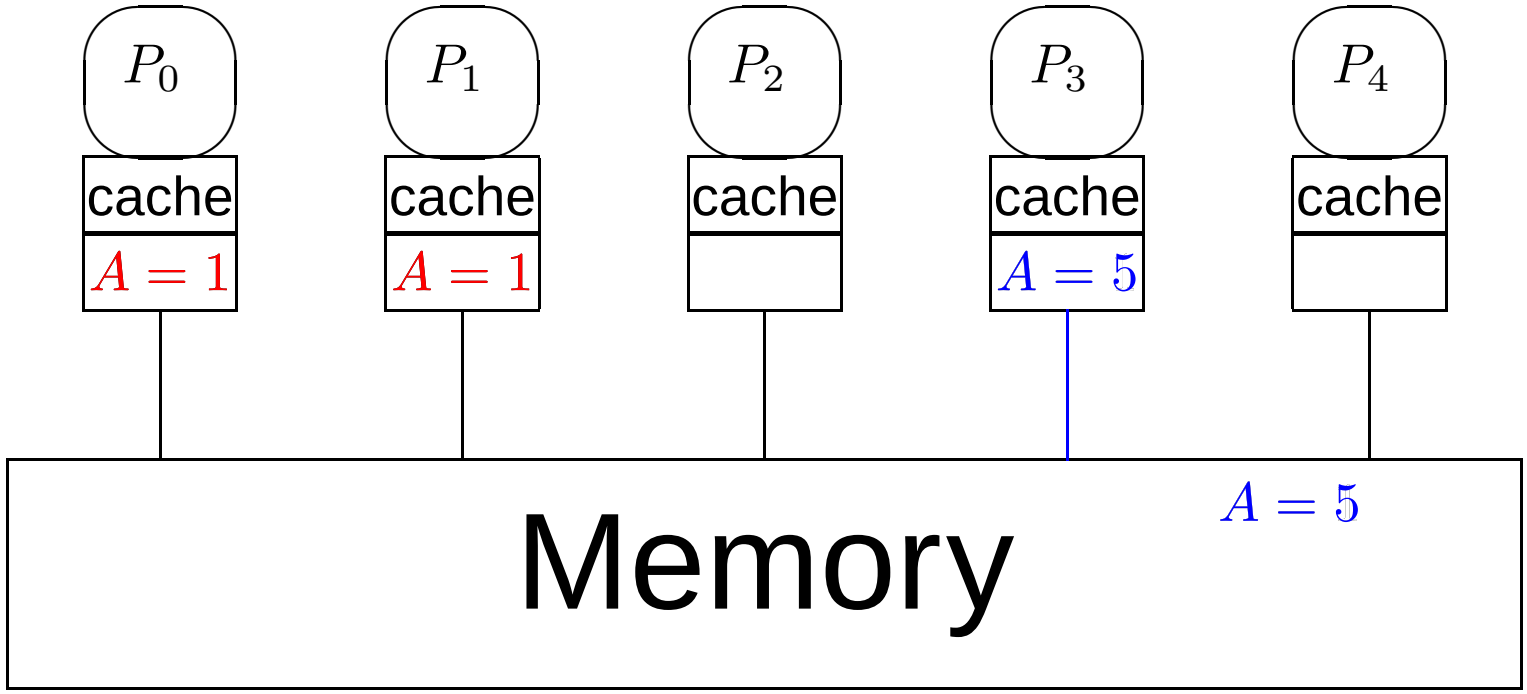
HPC architectures: Cache Coherency



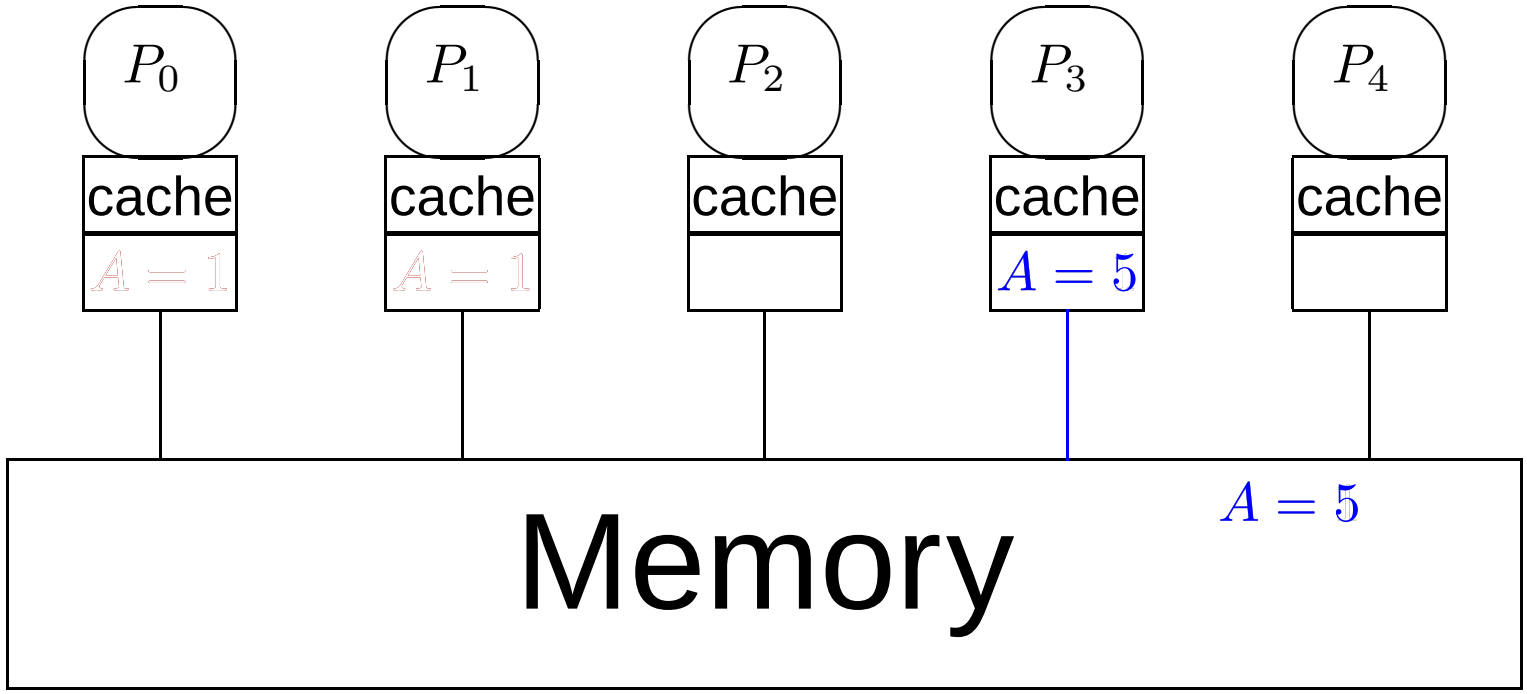
HPC architectures: Cache Coherency



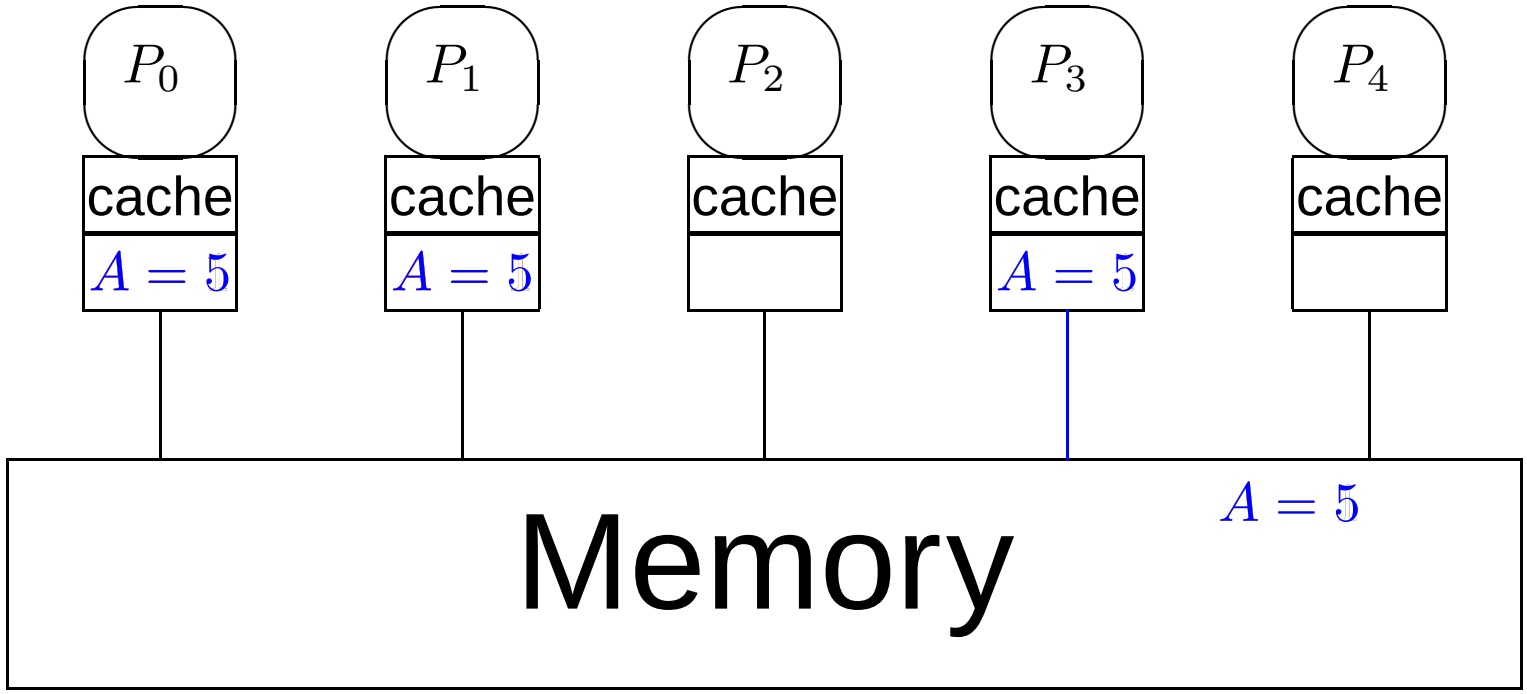
HPC architectures: Cache Coherency



HPC architectures: Cache Coherency



HPC architectures: Cache Coherency



If multiple processors cache and update data concurrently, then inconsistency may arise between the ”same” data in different caches

HPC architectures: Cache Coherency

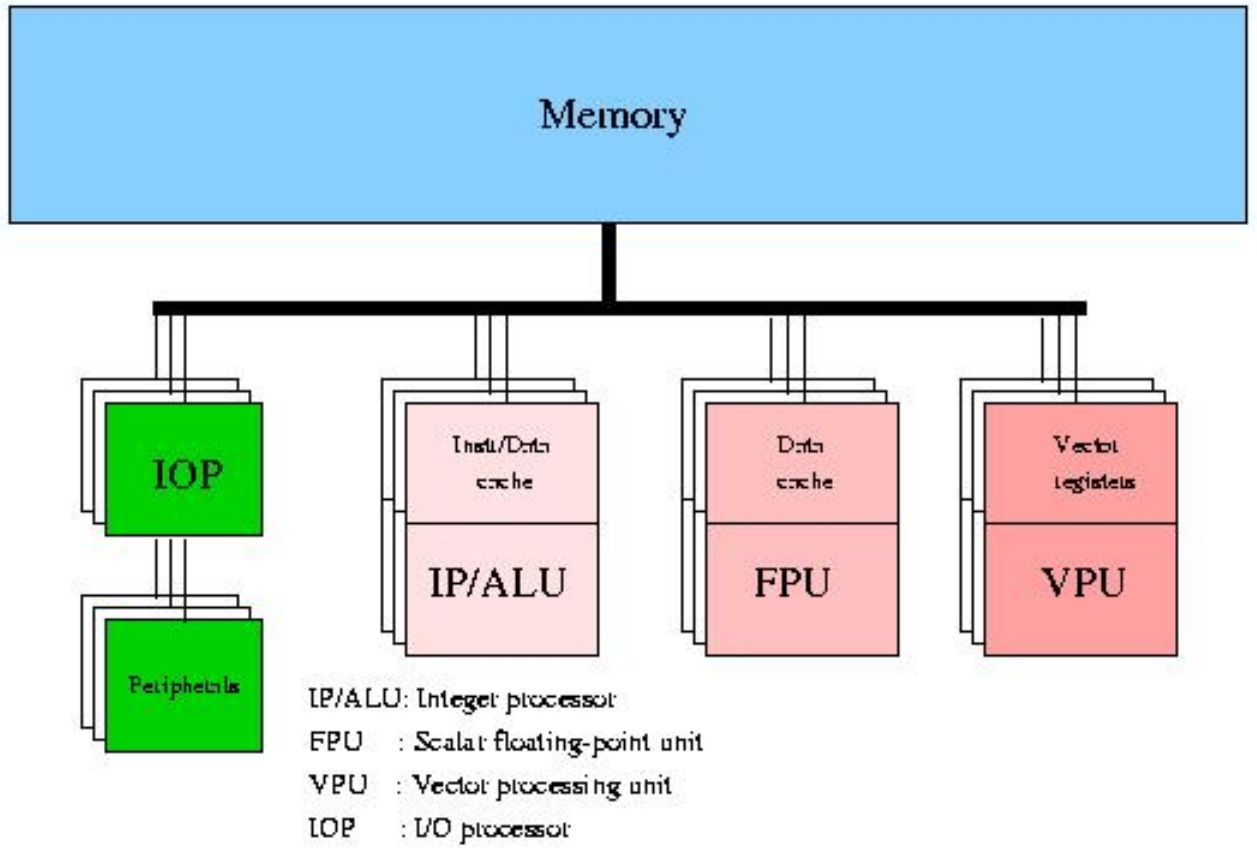
There are two classes of cache coherency protocols

- snoopy protocol, monitoring all requests to memory, bus based architecture (SUN)
- directory memory, cache state maintained in each memory block(SGI)

and two schemes:

- updates: write though can consume more system bandwidth (Cray)
- invalidates: can lead to false sharing optimized for uniprocessors performance (SGI)

HPC architectures: SIMD/MIMD: Vector Processor



Vector supercomputers load data into vector registers, to hide latency, and perform vector operations, to produce high computation rates.

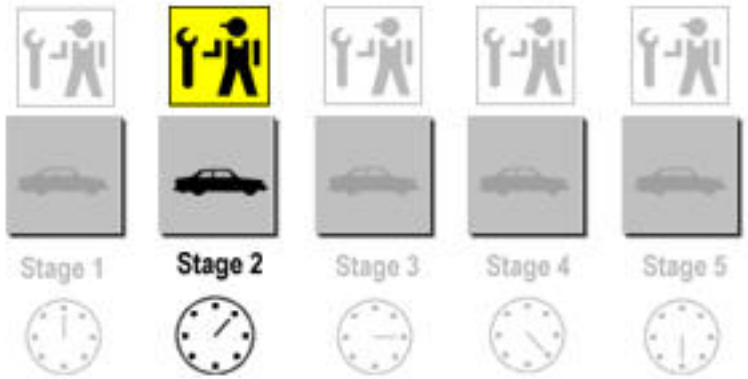
HPC architectures: Vector

- Overcomes limitations of pipelining in scalar processors:
  - ⋆ deep pipelining can slow down a processor,
  - ⋆ limits on instruction fetch and decode.
- One single vector instruction executes an entire loop: control unit overhead is greatly reduced.
- Vector instructions have a known access pattern to memory
- High memory bandwidth (interleaved) is needed to feed the vector processors.
- memory-memory (70's) and memory-register (Cray 1 ... C-90, Convex, NEC) memory-cache-register (Cray SV1, X1)

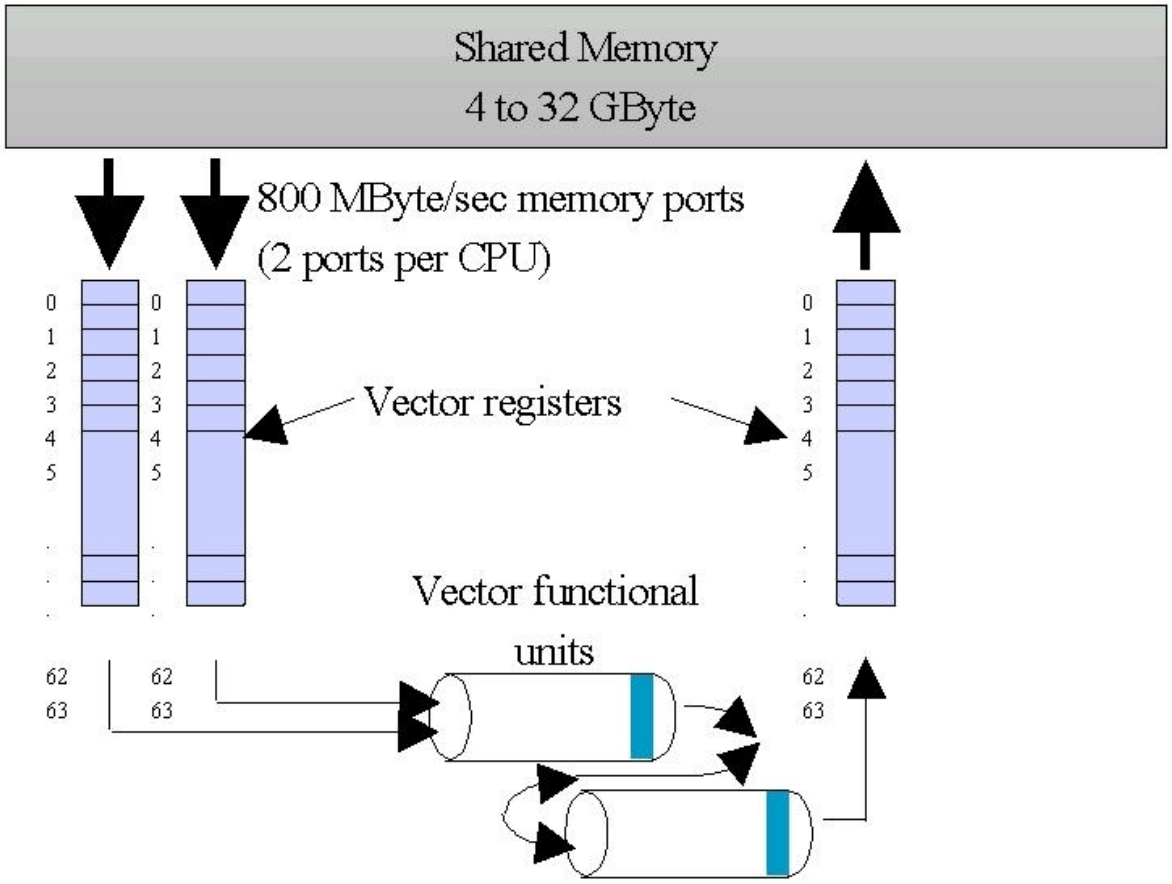
HPC architectures: Pipelining

Clock Cycle	Fetch	Decode	Execute	Store
1	Inst. 1			
2	Inst. 2	Inst. 1		
3	Inst. 3	Inst. 2	Inst. 1	
4	Inst. 4	Inst. 3	Inst. 2	Inst. 1
5	Inst. 5	Inst. 4	Inst. 3	Inst. 2

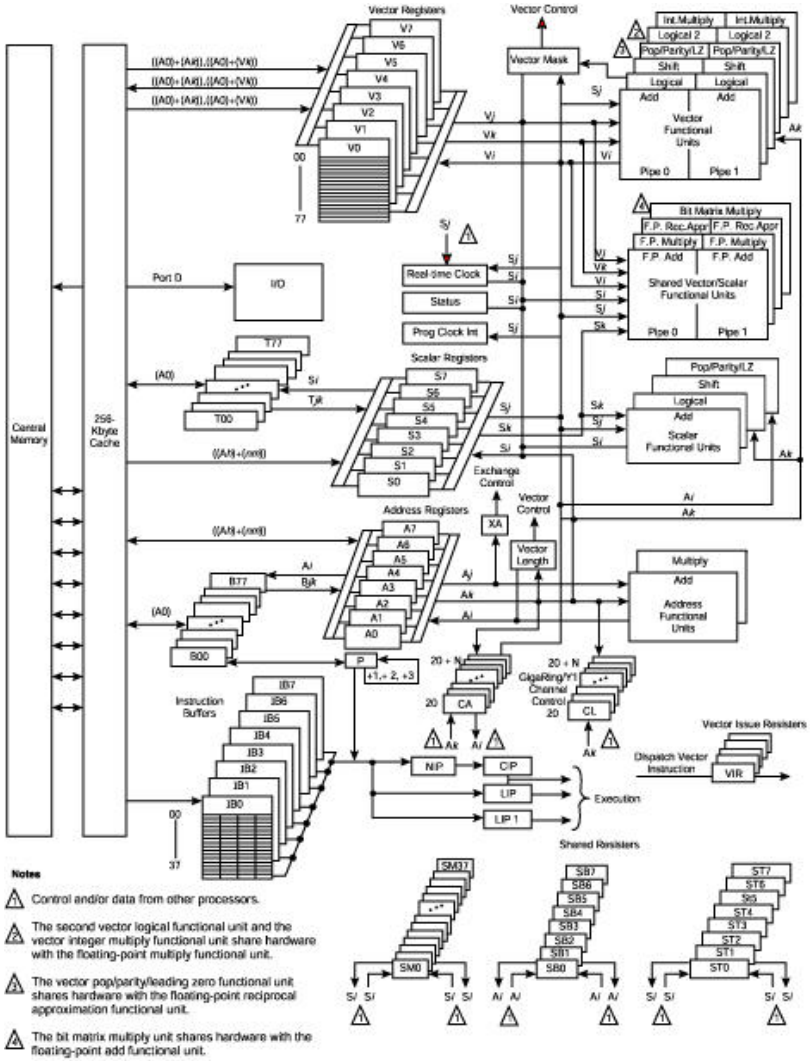
4-cycle instructions; using all the available hardware (ALU, Register, ..) within one cycle. Balance between clocks/instructions and pipeline depth.



HPC architectures: Vector



HPC architectures: Vector



HPC architectures: Vector code

Vectorization inhibitors within loops are:

- function/subroutine calls
- I/O statements
- Backward branches
- Statement numbers with references from outside the loop
- References to character variables
- External functions that do not vectorize
- RETURN, STOP, or PAUSE statements
- Dependencies (recursive members)

HPC architectures: Vector code

Dependency within loop cannot be vectorized:

```
for (i=2; i<=5; i++) {
   A[i-1] = B[i];
   C[i]   = A[i];
}
```

i	b[i]		A[i]		c[i]
1					
2		↗		→	
3		↗		→	
4		↗		→	
5		↗		→	

The correct computation requires the values of A_i to be stored in C_i before they are updated in the next iteration.

HPC architectures: Vector code

```
for (i=2; i<=5; i++) {
   A[i-1] = B[i];
   C[i]   = A[i];
}
```

Scalar mode: sequential through the loop

i	B		A		C
1			B[2]		
2		↗	B[3]	→	A[2]
3		↗	B[4]	→	A[3]
4		↗	B[5]	→	A[4]
5		↗		→	A[5]

HPC architectures: Vector code

```
for (i=2; i<=5; i++) {
   A[i-1] = B[i];
   C[i]   = A[i];
}
```

Vector mode: parallel through the loop

i	B		A		C
1			B[2]		
2	2	↗	B[3]	→	B[3]
3	3	↗	B[4]	→	B[4]
4	4	↗	B[5]	→	B[5]
5	5	↗		→	A[5]

All the new values of A_i are fetched by the vector instruction before and stored in C_i. This generates incorrect results.

HPC architectures: Vector code

Loop-linked Dependencies

```
      do i=1,100
1        A(i+1) = A(i) + B(i);
2        B(i+1) = B(i) + A(i+1)
      enddo
```

In the actual Iteration i+1, three Data Dependencies can be distinguished:

a Both Instructions 1 and 2 use values, which have been calculated by themselves in an earlier Iteration.

b The Instruction 1 uses a value, which has been calculated by the following Instruction 2 in an earlier Iteration.

c The Instruction 2 uses a value, which has been calculated by the preceeding Instruction 1 in the same Iteration.

HPC architectures: Vector code

For example, the compiler will not vectorize the following loop

```
do i = 1, n
   A(index(i)) = A(index(i)) + b(i)
enddo
```

potential dependency on the array A

HPC architectures: Vector code

For example, the compiler will not vectorize the following loop

```
do i = 1, n
   A(index(i)) = A(index(i)) + b(i)
enddo
```

potential dependency on the array A

```
!dir$ ivdep
      do i = 1, n
         A(index(i)) = A(index(i)) + b(i)
      enddo
```

Adding this directive to a safe loop can improve the performance up to a factor of 10.

HPC architectures: Vector code

In the following loop there is a vector dependency in both the inner and outer loops:

```
      do j = 1, n
         do i = 1, m
            temp = .25*(x(i,j-1)+x(i-1,j)
     *           + x(i+1,j)+x(i,j+1))-x(i,j)
            x(i,j) = x(i,j) + omega * temp
            if (abs(temp).gt.err1) err1=abs(temp)
         enddo
      enddo
```

x(i-1,j) is needed to update x(i,j). This loops runs at about 20 Mflops on the Cray SV1.

HPC architectures: Vector code

A solution to this problem is to change the vectors to run diagonally through the matrix X

```
      do jd=2,n+m
!dir$ ivdep
         do j=max(1,jd-m),min(n,jd-1)
            i = jd - j
            temp = .25*(x(i,j-1) + x(i-1,j)
 *                    + x(i+1,j) + x(i,j+1)) - x(i,j)
            x(i,j) = x(i,j) + omega * temp
            if (abs(temp) .gt. err1) err1 = abs(temp)
         enddo
      enddo
```

This loop now vectorizes and runs at over 260 Mflop/s on a 1000x1000 grid.

HPC architectures: Vector code

To Do: vector compiler examples hpm examples sustained peak performance scalar-vector

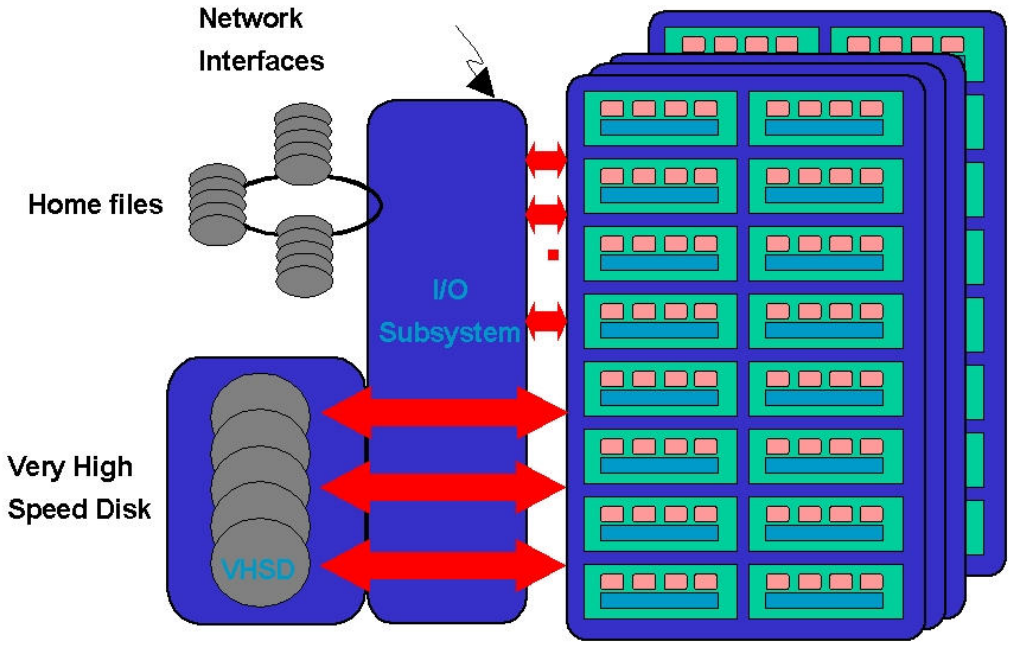
Vector architectures: the Cray X1

Cray X1 = Cray T3E + Cray SV1ex

- 1 CPU: 8 vector pipes: 12.8 (25.6) Gflop/s
- 1 node board contains 4 CPU's
- 200 GB/s from memory to CPU's within a node board
- 8-32 (64) GB Memory per node board
- 1024 nodes $\rightarrow$ 4096 CPU's: peak 50 TeraFlop, max 64 TB memory

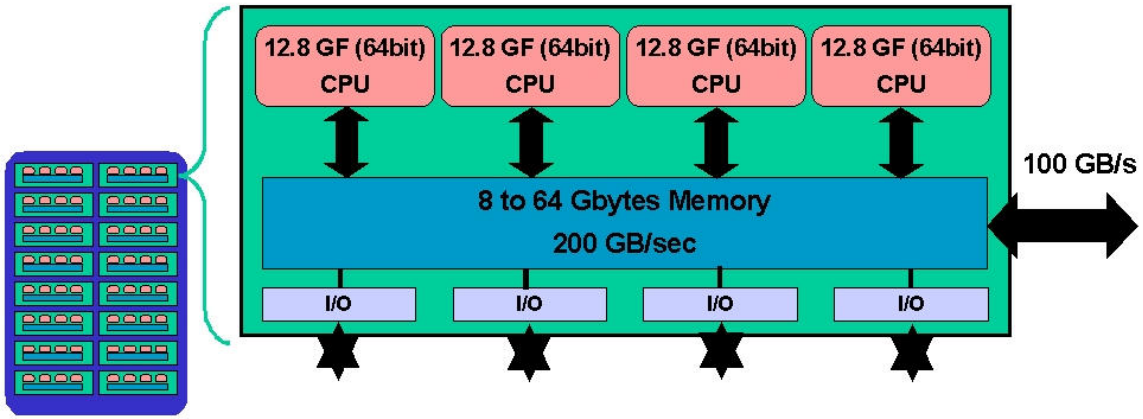
Vector architectures: the Cray X1

Frame



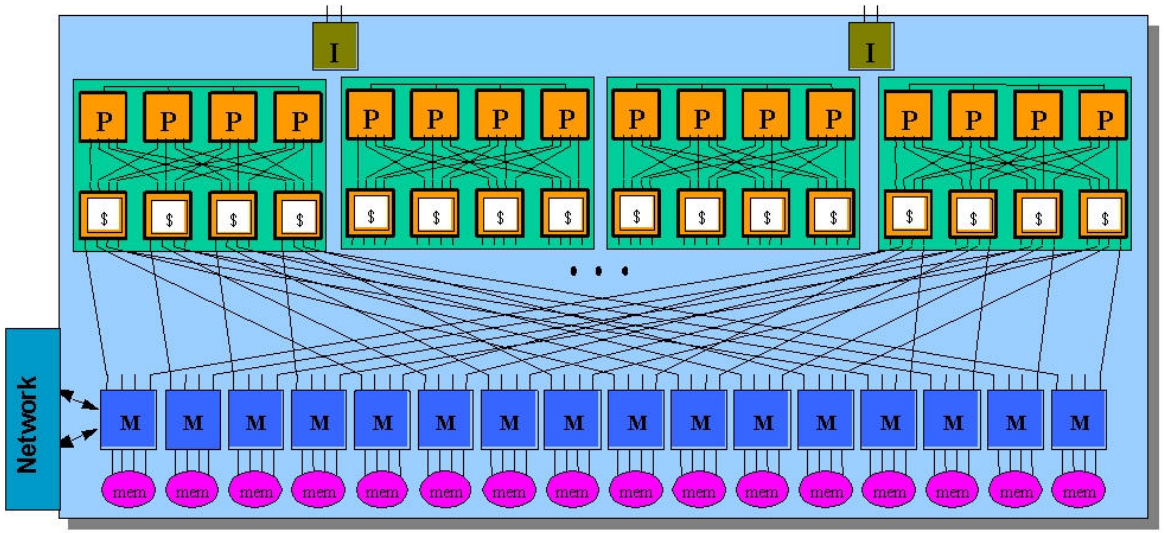
Vector architectures: the Cray X1

Node



Vector architectures: the Cray X1

Detailed node

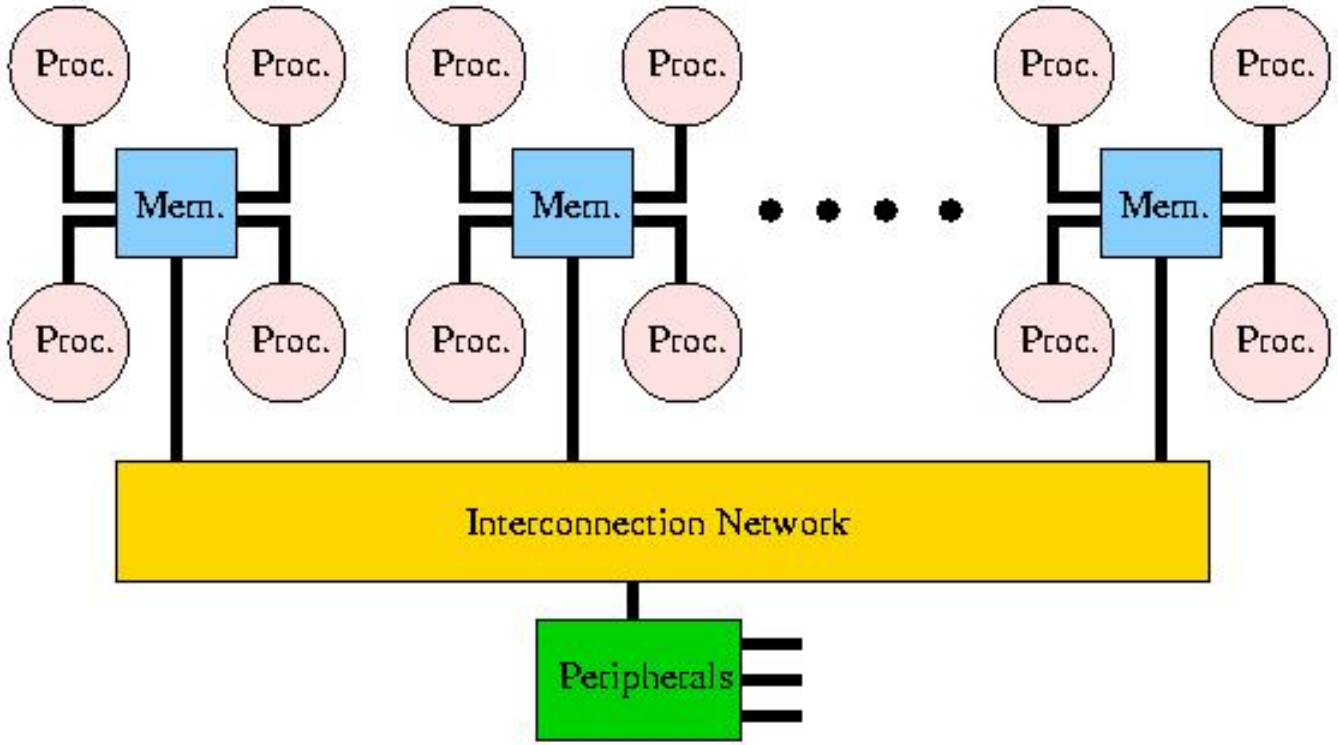


architectures: future CPU's

- Field Programmable Gate Array
  FPGA's allow computer users to tailor microprocessors to meet their own individual needs.

- Intelligent RAM: processors and memory are merged onto a single chip.
  narrow or remove the processor-memory performance gap

Clusters and their interconnects



A Beowulf-class system is a cluster with nodes that are personal computers (PC) or small symmetric multiprocessors (SM) of PCs integrated by COTS local area networks (LAN) or system area networks (SAN), and hosting an open source Unix-like node operating system.

Clusters and their interconnects

Cluster Node Hardware

- Processor: P3, P4, Athlon, EV7, Itanium 2, Power4 , Ultra Sparc ...
- memory: SDRAM, DDR, RAMBUS
- secondary storage IDE, SCSI, CD-ROM
- external interface PCI, USB

Clusters and their interconnects

Cluster Network Hardware

- hubs: inexpensive limited amount of communication
- switches: increasing system throughput and reducing network contention
- Myrinet: custom network control processor that provides high bandwidth and low latencies
- Fast Ethernet / Gigabit Ethernet: high latencies
- SCI (scalable coherent interface): high bandwidth
- VIA (Virtual Interface Architecture): very low latencies. moving data between application processes without requiring the intervening copying of the data to the node operating systems.
- Infiniband: latency even lower, directly connecting to the processor's memory channel interface.

Clusters and their interconnects

Four types of interconnections:

Connection	*Communication*	
	Message Based	Shared Storage
I/O Attached	Most common type, includes most high-speed networks	Shared disk subsystems
Memory Attached	Usually implemented in software as optimization	Global and distributed shared memory

I/O attached message-based systems are by far the most common. The I/O bus provides a

Memory attached systems are less common, since the memory bus of an individual computer generally has a design that is unique.

As network hardware became faster during the 1990's, the overhead of the *communication protocols* became significantly larger than the actual hardware transmission time for messages,

HPC architectures Summary

High performance computers are achieved either by increasing the level of parallelism in the CPU and in memory, or by using multiple CPUs. In practice, there are 3 classes of HPC today:

- Vector
  - Instructions with vector operands
  - Memory is banked to increase the bandwidth between CPU and memory
  - Compilers are able to transform certain loop structures into series of vector operations
- SM-MIMD
  - Memory is segmented, with each segment being attached to every processor
  - Ensure that the memory is consistent between processors.
  - For large SMP systems, the cost of memory far outweighs the cost of processors
  - Programs can be incrementally parallelized (OpenMP)

- Clusters
  - clusters are the ultimate in scalability – need more CPU power – just add another PC
  - the number of (expensive) networking components per CPU will increase logarithmically with CPU number
  - exploitation of parallelism must be handled explicitly by the programmer and data needs to be distributed at the outset.
  - can be used for overflow computing.
  - apart from the speed of the processors and the software provided by the vendors of DM-MIMD supercomputers, the distinction between clusters and compute clusters becomes rather small and will undoubtly decrease in the coming years.

HPC software tools

The software components that comprise the environment of a commodity cluster may be described in two major categories:

- resource management
  - system software
- programming tools

HPC software tools

Resource Management Software:

- Installation and Configuration
- Scheduling and Allocation
- System Administration
- Monitoring and Diagnosis
- Distributed Secondary Storage
- Availability

HPC software tools

Cluster Operating System:

- Stability
- Performance
- Extensibility
- Scalability Enabling low-overhead calls to access the interconnect (inter-node scalability).
- Heterogeneity
- High availability
- Checkpoint restart

HPC software tools

Application Programming Tools:

- Numerical Libraries (BLAS, Lapack, Atlas, NAG) Mathematical software
- Compilers and Preprocessors
- MPI/PVM Implementations MPICH
- Development Environments
- Performance Analyzers
- Debuggers
- Middleware; RPC, CORBA, JAVA

HPC software tools

Single System Image (SSI) can be realized either using hardware or software techniques.

- hardware-level
  highest level of transparency, but due to its rigid architecture, it does not offer the flexibility required during the extension and enhancement of the system.

- software-level
  - ⋆ kernel-level: expensive to develop and maintain
  - ⋆ application-level approach helps realize SSI partially and requires that each application be developed as SSI-aware separately. HPF, PVM, MPI
  - ⋆ middleware-level appears to offer an economy of scale compared to other approaches although it cannot offer full SSI like the OS approach.
    MOSIX is a set of kernel extensions for Linux that provides support for seamless process migration.

HPC software tools

File I/O is crucial for the performance of many types of applications, scientific codes as well as databases.

- dedicated I/O nodes
  Every shared resource represents a potential bottleneck in a system that has to be scalable.

- node-local I/O
  Reduces inter-node communication and I/O contention, while maintaining a consistent global view of the I/O space. Examples of this approach are PVFS (ROMIO).

HPC software tools

Suppose that a program running on a cluster needs 1 MByte of data to be accessible by all the processors concurrently. Several strategies can be considered:

- Put a copy of that data on each disk (replication);
- Read the data from a single disk, broadcast it to all the units and keep it in the memory of each processor (caching)
- Distribute the data among the available disks. Whenever a part of the data is needed during program run the processor owning the data reads that part and broadcasts it to the other units (distribution);
- Various combinations of the above.

It is not easy to determine which solution is the best. Everything depends on what the program actually is doing. Are there some other disk accesses too, is there much communication over the network .....

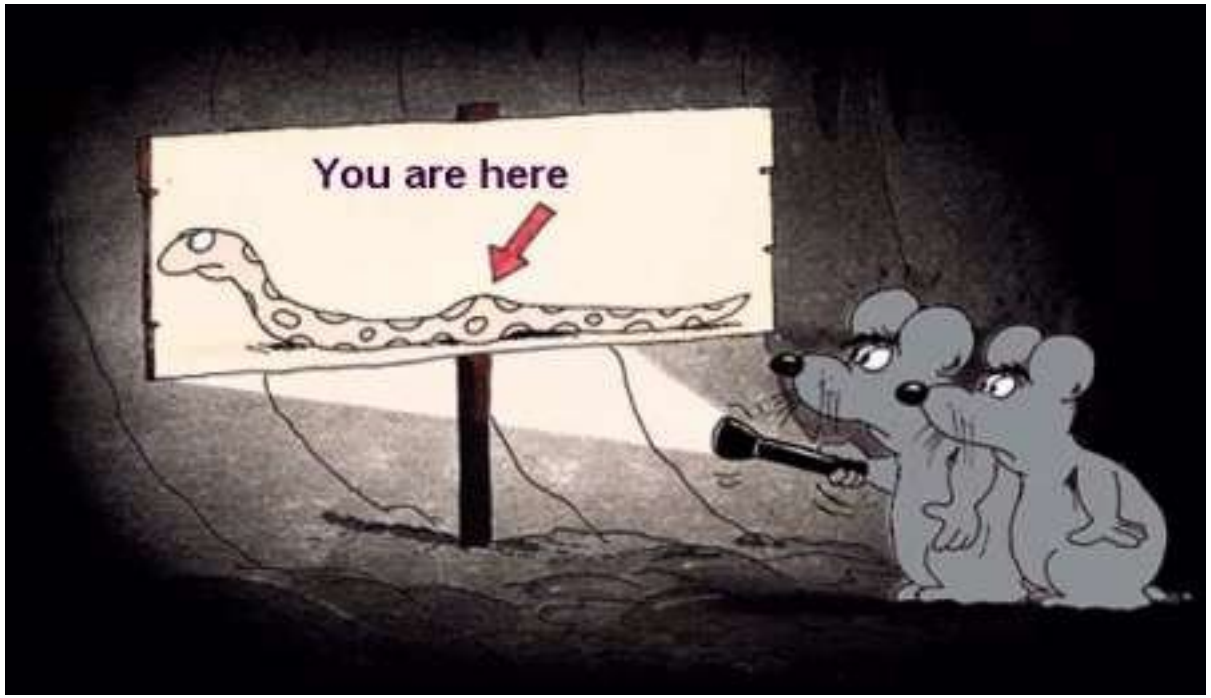
PVFS tested by Dell

http://www.dell.com/us/en/esg/topics/power_ps4q02-kashyap.htm

Conclusions and points for improvement from Dell test:

- Effectively provides a global scratch space for HPC clusters
- Dell cluster running PVFS efficiently uses the Myrinet network
- Limited by use of TCP
- Other interfaces can improve PVFS: Virtual Interface Architecture (VIA), GM
- Fault tolerance needs improvement

Break



Applications: Amdahl's Law

Time to run a parallel program can be subdivided in 3 components

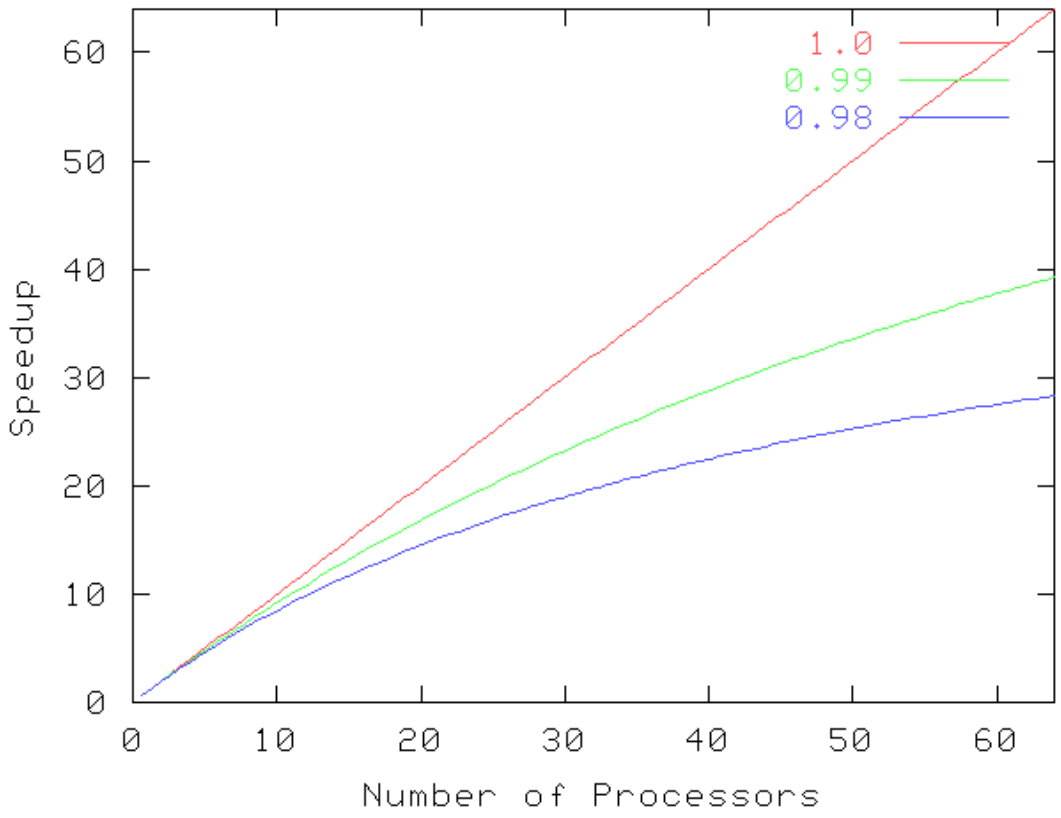
$$T_p = \frac{\alpha T_1}{P} + \frac{(1-\alpha)T_1}{P} + T_c(P)$$

Fraction $1-\alpha$ is 'essentially serial', for most problems the serial fraction decreases as the problem size grows. Note that T_1 is the run time on 1 processor and $T_c(1) = 0$.

Speedup is defined as:

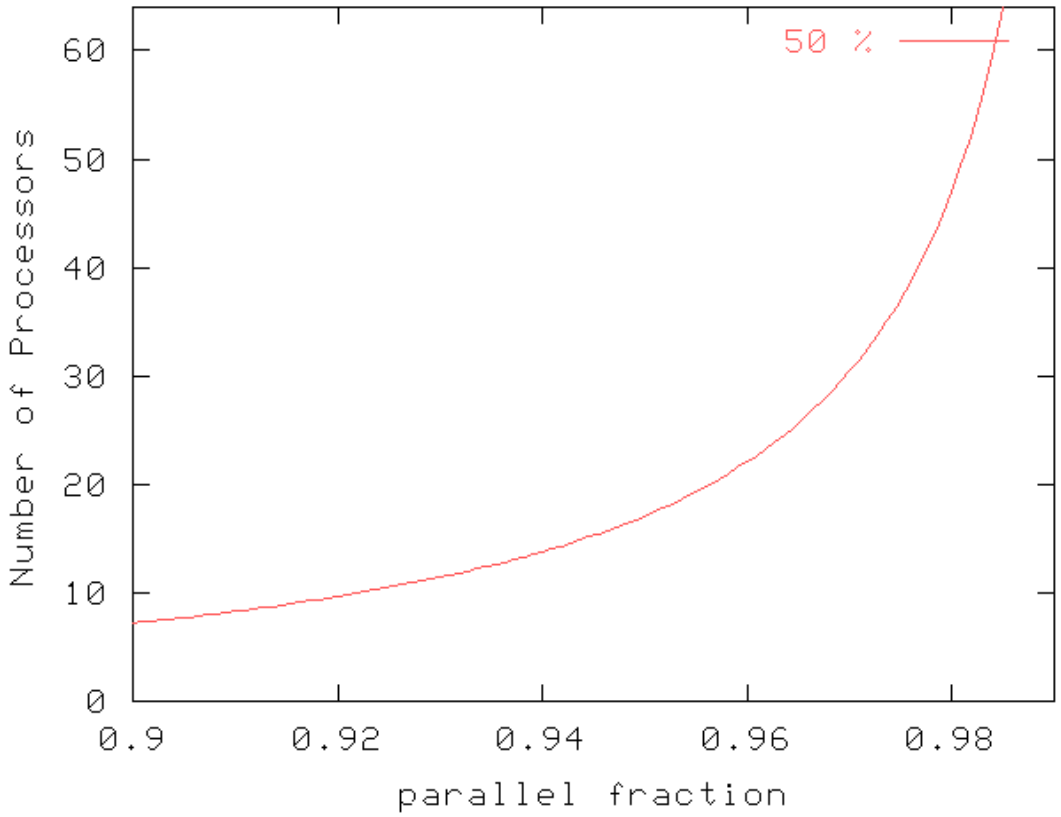
$$S_p = \frac{T_1}{T_p} = \frac{1}{1-\alpha+\frac{\alpha}{P}+\frac{T_c(P)}{T_1}} \leq \frac{1}{1-\alpha}$$

Applications: Amdahl's Law



Speedup changes rapidly with decreasing parallel fraction.

Applications: Amdahl's Law



50% efficiency $\frac{S_p}{P} = 0.5$ as function of the parallel fraction.

$$\alpha = \frac{2P^2 - P}{1 - P}$$

Applications: Amdahl's Law

Some conclusions from Amdahl's law:

Applications: Amdahl's Law

Some conclusions from Amdahl's law:

- For better scalability use a slower CPU (T_c remains constant)

Applications: Amdahl's Law

Some conclusions from Amdahl's law:

- For better scalability use a slower CPU (T_c remains constant)
- Single CPU optimizations reduce the scalability (α larger)

Applications: Amdahl's Law

Some conclusions from Amdahl's law:

- For better scalability use a slower CPU (T_c remains constant)
- Single CPU optimizations reduce the scalability (α larger)
- Scalability is most of all a function of the application not of the hardware.

HPC architectures

Two types of large scale computing:

- Capability computing
  The system is employed for one or a few programs for which no alternative is available in terms of computational capabilities. The system are not always used with the greatest efficiency

- Capacity computing
  Use the system with the highest possible throughput capacity using the machine resources as efficient as possible. This may have adverse effects on the performance of individual computing tasks.

Classification of applications

Application Area's:

- Government-Classified
- Government-Research
- Aerospace
- Automotive
- Bioinformatics
- Chemical/Pharmaceutical
- Petroleum
- Weather/Environmental
- Academic Research

Classification of applications

Application Types:

- Forward Modeling
- Inversion
- Signal Processing
- Searching/Comparing

Classification of applications

Area	Modeling	Inversion	Processing	Searching
Government-Classified				X
Government-Research	X		X	
Aerospace	X			
Automotive	X			
Bioinformatics	X			X
Chemical/Pharmaceutical	X			
Petroleum	X	X	X	
Weather/Environmental	X	X	X	
Academic Research	X	X	X	X

Application: Forward Modeling

- Grid based
- Finite element
- Finite difference methods
- Parallelization by domain decomposition
- many small messages: low latency network

Application: Forward Modeling

Finite Differences solution of (partial) differential equations

$$\frac{\partial^2 G(x)}{\partial x^2} + A(x)\frac{\partial G(x)}{\partial x} + B(x)G(x) = C(x)$$

$$\frac{G_{i-1} - 2G_i + G_{i+1}}{h^2} + A(x_i)\frac{G_{i-1} + G_{i+1}}{2h} + B(x_i)G_i = C(x_i)$$

solve for G_i

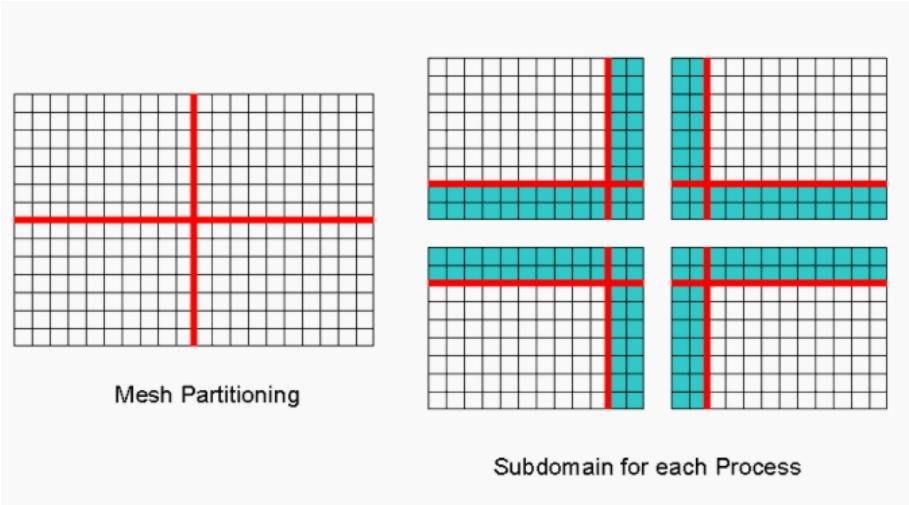
$$\begin{pmatrix} d_1 & e_1 & 0 & \dots & 0 \\ c_2 & d_2 & e_2 & \dots & 0 \\ 0 & c_3 & d_3 & e_3 \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & c_{n-1} & d_{n-1} & e_{n-1} \\ 0 & \dots & & d_n & e_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \\ a_n \end{pmatrix}$$

Solve direct or by iterative methods.

Application: Forward Modeling

For 2D differential equations convolution with stencil

$$\begin{array}{ccc} & 1 & \\ 1 & -2 & 1 \\ & 1 & \end{array}$$



Examples: Car crash simulation, Weather Prediction, Reservoir Simulation (Fluid Flow in Porous and Fractured Media), Computational Fluid Dynamics, Molecular Dynamics, n-body problem (Barnes Hut), ray-tracing, ...

Application: Forward Modeling

Numerical Simulation of Turbulence.

Reynolds number(Re) is used in momentum, heat, and mass transfer to account for dynamic similarity.

Object	Re	grid	flop	Gflop/s[3]	Memory GB	GB/s
square cylinder	10^4	10^7	10^{14}	0.3	1	
golf ball	10^5	3.10^9	10^{17}	300	300	100
car	10^{12}	10^{12}	10^{20}	3.10^5	1.10^5	1.10^5

Moore's Law 2 x faster per 18 months = 100 x faster in 10 years.

Reynolds number $10^6 - 10^7$ in 2017 !

[3] CPU runs for 300 hours

Application: Forward Modeling
Bio Informatics

- Molecular Dynamics: simulation of protein structures (folding/docking)
- Biophysics: prediction of structure from experimental data (X-ray)
- Structural BioInformatics: sequence comparison algorithms (BLAST)

Protein folding is fundamental to life and is of the utmost practical importance for developing new medicines and getting insight in gene functions.

Application: Forward Modeling
Bio Informatics

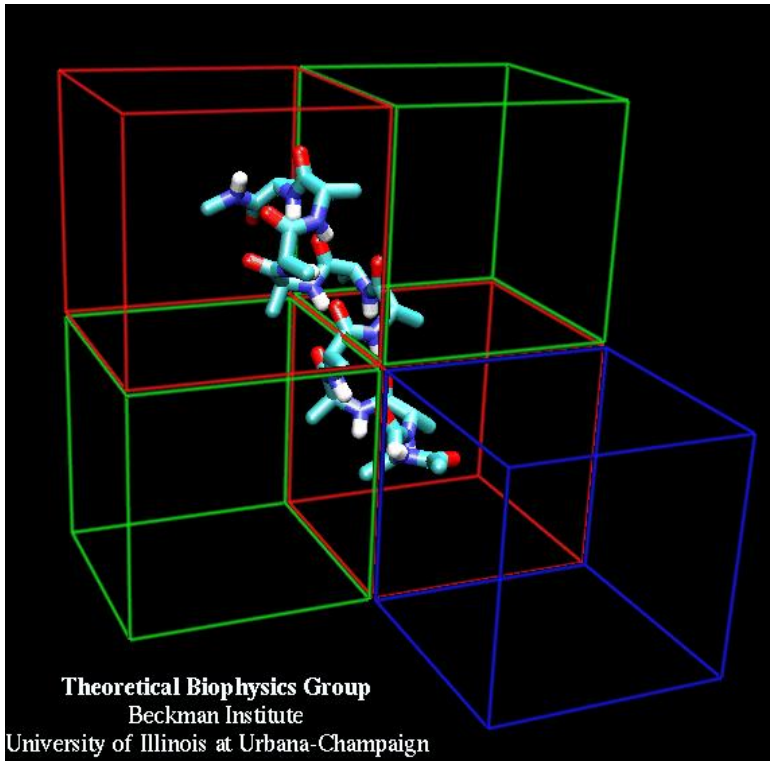
The typical composition of potential energy functions for protein folding is

$$E(X) = \sum_{bonds\ i} S_i^b(b_i - \bar{b}_i) + \sum_{bond\ angles\ i} S_i^\phi(\phi_i - \bar{\phi}_i)$$
$$+ \sum_{dihedral\ angles\ i} \sum_n \left(\frac{V n_i^\tau}{2} [1 \pm \cos(n\tau_i)] \right) + \sum_{atoms\ i<j} -\left(A_{ij}/r_{ij}^6 + B_{ij}/r_{ij}\right)$$
$$+ \sum_{atoms\ i<j} (Q_i Q_j / D(r_{ij}) r_{ij}) + \ldots$$

In these expressions, the symbols b, ϕ, τ, and r represent, respectively, bond lengths, bond angles, dihedral angles, and interatomic distances. All are functions of the collective Cartesian positions X. The bar symbols represent equilibrium, or target values.

Application: Forward Modeling
Bio Informatics

NAMD: Domain decomposition of 3D structure block/domain replaced by notional atom which interact with atoms in other blocks.



Note, Amber uses Force decomposition, replication of spatial data on every node (Memory intensive)

Application: Forward Modeling
Bio Informatics

60.000 atoms NAMD 5 days on 64 CPU T3E to simulate 1 nanosecond, Typical protein folding takes microseconds (1000 more !)

36.000 atoms of DNA proetin interaction femtoseconds modeling 6 seconds 1 workstation

The development of more efficient simulation algorithms and computational strategies is continuously motivated by the remaining temporal gap needed to describe protein folding or ligand binding more than three orders of magnitude.

Application: Inversion

From measurements (F) compute models (M) representing properties (d) of the measured object(s).

$$F = Md + \varepsilon$$

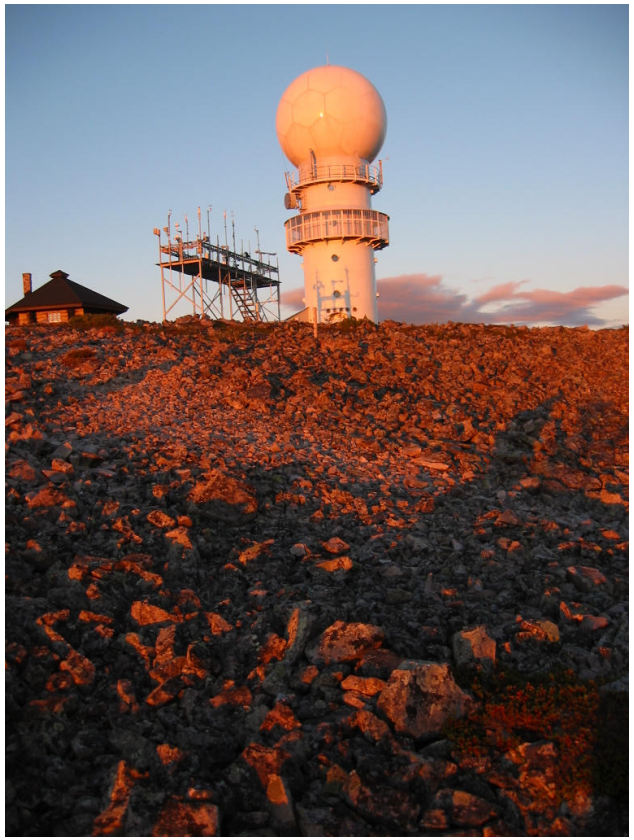
$$M^{-1}F = d + \varepsilon'$$

- Deterministic
  - ⋆ matrix inversions (hence the name inversion)
  - ⋆ conjugate gradient
- Stochastic
  - ⋆ Monte Carlo, Markov Chain
  - ⋆ Simulated annealing
  - ⋆ Genetic algorithms
- Requires large amounts of shared memory: low latency and high bandwidth

Examples: weather radar, seismic inversion, ...

Application: Inversion

Tomographic Statistical Inversion of Radar data



Memory usage is : $((xyz)^2)/2$ floating point numbers
Number of Flop : $(xyz)^3$
2D 200x200 (3D 39x39x39) requires 3.2 GB of memory and about 64 Tflop.

Application: Signal Processing

- Convolution model (stencil)
- FFT's http://www.fftw.org
- Matrix computations (eigenvalues, ...)
- Conjugate gradient methods
- not very demanding on latency and bandwidth
- some algorithms are embarrassingly parallel

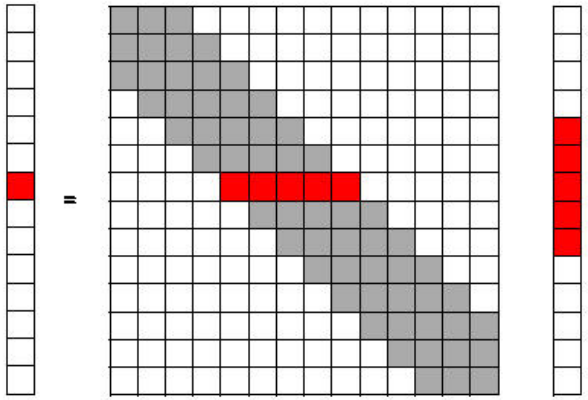
Examples: seismic migration/processing, medical imaging, SETI, streaming media

Application: Signal Processing

A 1D convolution is represented by:

$$P(x, z_{m+1}) = \int_{\partial D} W(x - x')P(x', z_m)\mathrm{d}x'$$

$$\begin{pmatrix} P(0) \\ P(1) \\ \vdots \\ P(j\Delta x) \\ \vdots \\ P((N-2)\Delta x) \\ P((N-1)\Delta x) \end{pmatrix} = \Delta x \begin{pmatrix} W(0) & W(1) & \dots & \dots & 0 \\ W(-1) & W(0) & W(1) & \dots & 0 \\ 0 & \ddots & \vdots & \dots & 0 \\ 0 & \dots & W(0) & \dots & 0 \\ 0 & \dots & \vdots & \ddots & 0 \\ 0 & \dots & W(-1) & W(0) & W(1) \\ 0 & \dots & \dots & W(-1) & W(0) \end{pmatrix} \begin{pmatrix} P(0) \\ P(1) \\ \vdots \\ P(j\Delta x) \\ \vdots \\ P((N-2)\Delta x) \\ P((N-1)\Delta x) \end{pmatrix}$$



Application: Signal Processing

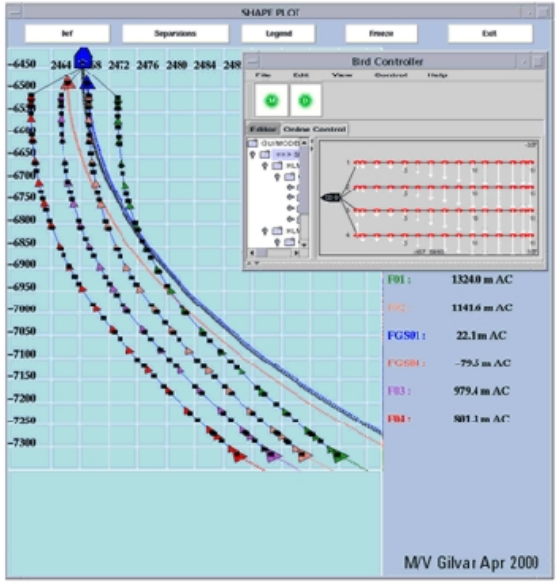
Simple algorithm

```
hoplx = (oplx+1)/2;
hoply = (oply+1)/2;
for (iy = 0; iy < ny; iy++) {
        starty = MAX(iy-hoply+1, 0);
        endy   = MIN(iy+hoply, ny);
        for (ix = 0; ix < nx; ix++) {
                startx = MAX(ix-hoplx+1, 0);
                endx   = MIN(ix+hoplx, nx);
                dumr = dumi = 0.0;
                k = MAX(hoply-1-iy, 0);
                for (i = starty; i < endy; i++) {
                        l = MAX(hoplx-1-ix, 0);
                        for (j = startx; j < endx; j++) {
                                dumr += data[i*nx+j].r*opx[k*oplx+l].r;
                                dumr += data[i*nx+j].i*opx[k*oplx+l].i;
                                dumi += data[i*nx+j].i*opx[k*oplx+l].r;
                                dumi -= data[i*nx+j].r*opx[k*oplx+l].i;
                                l++;
                        }
                        k++;
                }
                convr[iy*nx+ix].r = dumr;
                convr[iy*nx+ix].i = dumi;
        }
}
```

Cache optimized algorithm is 2.5 times faster on RISC processors than simple algorithm.

Application: Signal Processing

Seismic pre-stack depth migration.



Seismic shot:	12 * 6000/25 * 1024 * 4 = 11 MB/shot
Area:	2500 km^2 = 400.10^6 shots
Total Data:	4.5 TB
Processing one shot:	5-60 minutes

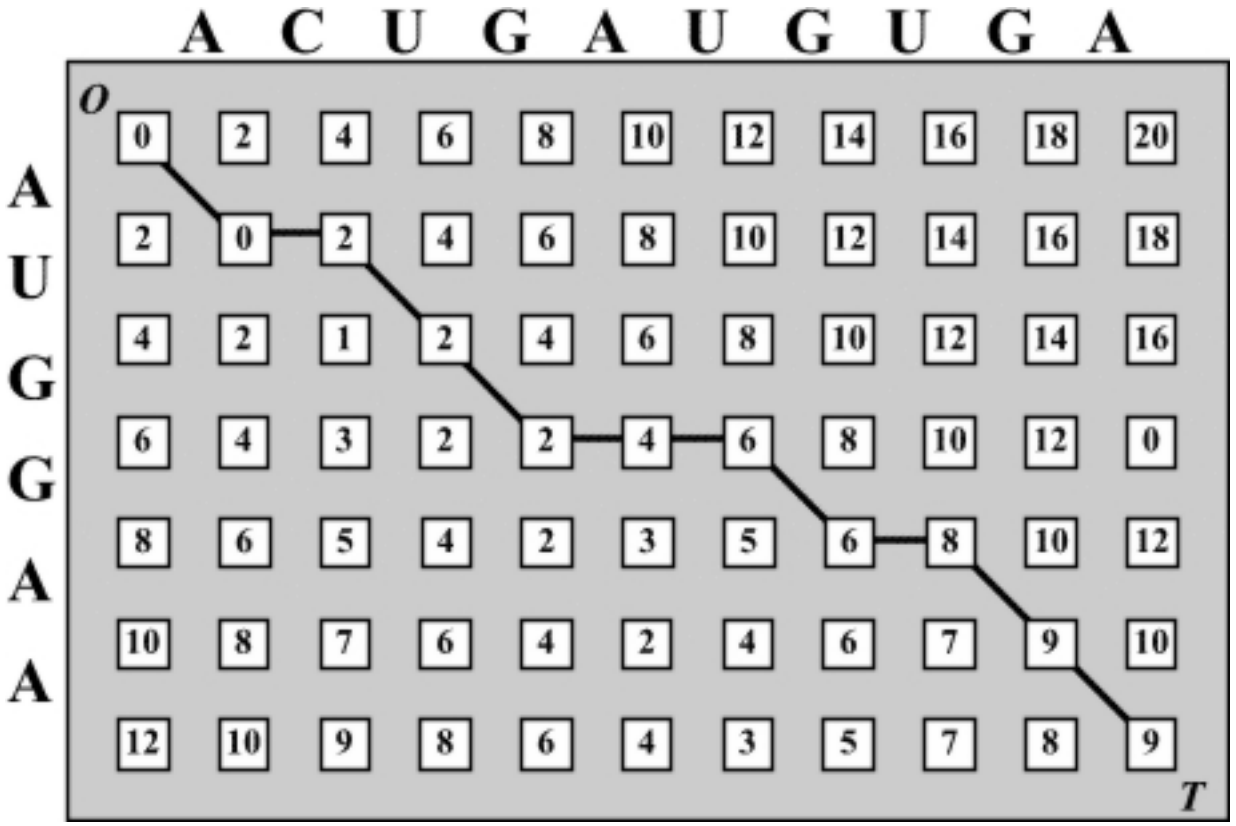
Application: Searching/Comparing

- integer operations are more dominant than floating point
- IO intensive
- pattern matching
- embarrassingly parallel, suitable for grid computing

Examples: code encrypting, bio-informatics, data-mining

Application: Searching/Comparing

Bio-informatics



The D-matrix for the alignment of AUGGAA to ACUGAUGUGA.

Cray's Bit Matrix Multiply can be used in the search algorithms.

Applications: Parallelization on cluster

Some guidelines:

- Applications with a minimum amount of communication will always run well
- Achieving low latency and high bandwidth requires efficient communication protocols that minimize communication software overhead, and fast hardware.
- Low-level details such as memory bus speed and PCI bus interfaces are just as important as processor speed or cache sizes:
  - On a code, which is memory-intensive, a system with a lower core frequency but higher bus speed is probably the best choice.
  - Single- or multiprocessor building blocks.
    Memory-intensive applications are better of with single processor blocks or crossbar-based memory connections (expensive).

- Network interconnects
  - Communication-intensive applications require efficient data transfers even at low processor scales. Fine-grain algorithms that are sensitive to short message *latency*, and coarse grain algorithms that are sensitive to peak *bandwidth* of bulk transfers require balanced and efficient communication systems.
  - Technical applications often possess the ability to overlap communication and computation. Network architectures with software stacks that support user-to-user overlap of communication and computation can reduce application run-times significantly.
  - Large-scale clusters with number of processors exceeding 1,000 requires highly optimized network interconnects and applications.

- The effective benefit of the increase in microprocessors speed may be significantly diminished if the cluster is interconnected with an inadequate network that may become a performance bottleneck, in latency, bandwidth, and/or consumption of CPU cycles to deliver needed performance.

Cray HPC project

Steps

Cray HPC project

Steps

- Application is central

Cray HPC project

Steps

- Application is central
- Intensive benchmarking and tuning

Cray HPC project

Steps

- Application is central
- Intensive benchmarking and tuning
- System Hardware design

Differentiator is the application knowledge coupled to the hardware.

References

- This presentation: HPC Overview
  http://www.xs4all.nl/~janth/Presentations/HPC_Overview.pdf
- Aad van der Steen Overview of recent Supercomputers
  http://www.phys.uu.nl/~steen/web02/overview02.html
- Cray hardware and software manuals
  http://www.cray.com/craydoc/
- Cluster Computing white paper
  http://www.dcs.port.ac.uk/~mab/tfcc/WhitePaper/
- Single System Image
  http://www.mosix.org/
  http://www.cs.rice.edu/~willy/TreadMarks/overview.html
- Fast Fourier Transform
  http://www.fftw.org
  http://aurora.phys.utk.edu/~forrest/papers/fourier/index.html

- Origin 3000 architecture
  http://www.sara.nl/Customer/systems/sgisn_1/qr/NCF_cursus/index.html

- Molecular Dynamics
  http://www.ks.uiuc.edu/Publications/Papers/PDF/SCHL99/SCHL99.pdf

- Hennessy, J. L. and Patterson, D. A. (2001) *Computer Architecture: A Quantitative Approach.*
  Morgan Kaufmann Publishers, Inc. San Mateo, CA. 3^{rd} edition.

- Flynn, M.F. (1972). *Some computer organizations and their effectiveness.*
  IEEE Trans. on Comp., C/21, 9, 948–960.

- Alan V. Oppenheim, Alan S. Willsky, S. Nawab Nawab, Syed ham Nawab Signals and Systems, Prentice Hall, Hardcover, 2nd edition, Published August 1996.