Overview of HPC

Jack Dongarra

Computer Science Department
University of Tennessee

and

Mathematical Sciences Section
Oak Ridge National Laboratory

(http://www.netlib.org/utk/people/JackDongarra.html)

Three Basic Ways of Improving Computer Productivity

- **Increasing component operating speed**
- **Parallelizing the process of calculation**
- **Making computers that are specialized for a certain class of problem**

Why Parallel Computers?

- **Desire to solve bigger, more realistic applications problems.**
- **Fundamental limits are being approached.**
- **More cost effective solution**

Example: Weather Prediction (Navier-Stokes) with 3D Grid around the Earth

$$6 \quad variables \begin{cases} temperature \\ pressure \\ humidity \\ 3 - wind \quad velocity \end{cases}$$

- **1 Kilometer Cells**
- **10 slices** $\rightarrow 5 \times 10^9$ **cells**
- **each cell is 8 bytes,** 2×10^{11} **Bytes** $= 200$ **GBytes**
- **at each cell will perform 100 ops/cell**
- **1 minute time step**
- $\frac{100 ops/cell \times 5 \times 10^9 cells}{1 min \times 60 sec/min} = 8 GFlop/s$

Goals of a Parallel Architecture

- **High Performance:**
  - **Performance $>$ fastest uniprocessors**
- **Competitive cost-performance:**
  - **Cost-performance is competitive with workstations**
- **Scalable performance and cost-performance**
- **General-purpose: cost-effective for a wide range of applications**

Scalable Multiprocessors

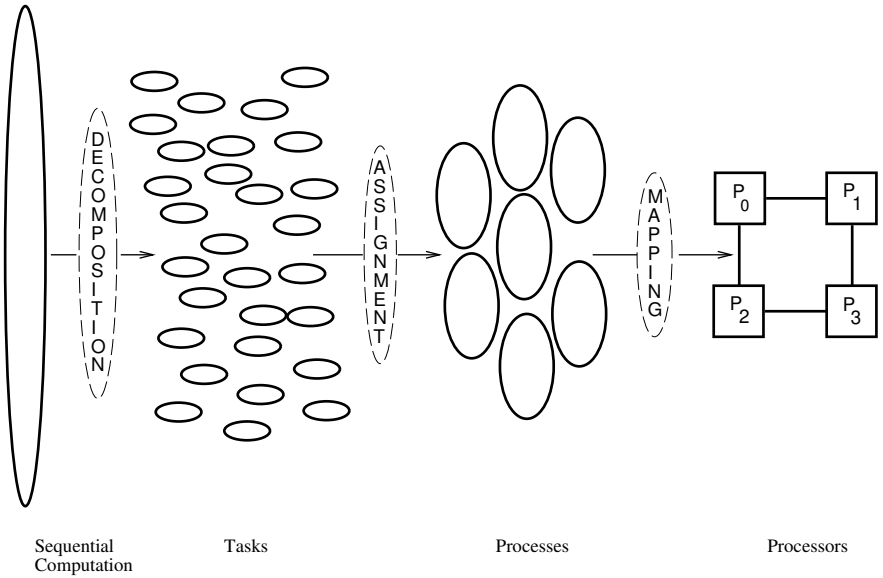
What is Required?

- Must scale the local memory bandwidth linearly.
- Must scale the global interprocessor communication bandwidth.
- Scaling memory bandwidth cost-effectively requires separate, distributed memories.
- Cost-effectiveness also requires best price-performance in individual processors.

What we get

- Compelling Price/Performance
- Tremendous scalability
- Tolerable entry price
- Tackle intractable problems

Creating a parallel program



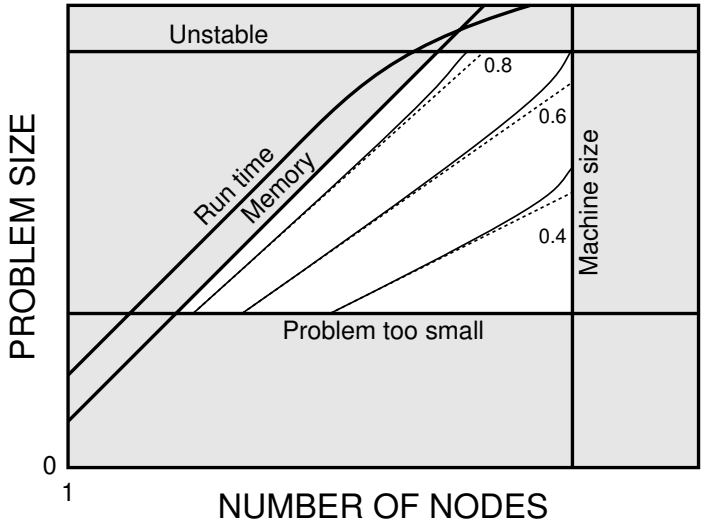
- Tasks : pieces of work that constitute the overall computation
- Processes (threads) : abstract entity that performs tasks
  - portion of address space private
  - portion shared
- 4 primary parts of the programming process
  - decomposition, assignment (partitioning), orchestration, and mapping

What limits the performance of a parallel program?

- Available Parallelism
- Load Balance
  - some processors work while others wait
- Extra work
  - management of parallelism
  - redundant computation
- Communication

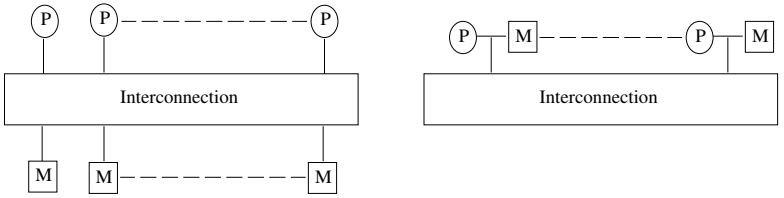
Scalability

- A parallel algorithm is scalable if the concurrent efficiency can be held constant as the number of processors increases by increasing the problem size.
- The *isoefficiency function*, $\rho_\epsilon(N_p)$, **gives the problem size necessary to maintain some fixed efficiency, ϵ, as N_p increases.**



Shared Memory Architectures

- Key Feature : All processors in the system can directly access all memory locations in the system, thus providing a convenient and fast (?) mechanism for processors to communicate.
  - convenient : (i) location transparancy; (ii) abstraction supported is same as that on current day uniprocessors.
- Memory can be centrally placed or distributed in such machines.

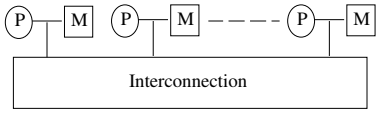


PVP - Parallel Vector Processors
SMP - Symmetic Multi-Processors

Cray X,Y,C,T	Convex Exempler
NEC SX	(KSR)
Fujitsu CP 2600	
Hitachi S-3800	
SGI Power Chall	
SUN	
Sequent	
DEC Tubro Laser	
Chen	

Message Passing Architectures

- Processors can directly access only local memory and all communication and synchronization happens via messages.



MPP - Massively Parallel Processors

IBM SP-2
Intel Paragon
Cray T3D/E
NCube 3
(Thinking Machines CM-5)

Comparison

- Shared Memory
  - Explicit global data structure
  - Decomposition of work is independent of data layout
  - Communication is implicit
  - Explicit synchronization
    - need to avoid race condition and over writing
- Message Passing
  - Implicit global data structure
  - Decomposition of data determines assignment of work
  - Communication is explicit
  - Synchronization is implicit
    - data buffered till received

Survey of Current Machines

MIMD, global memory

- Cray Y–MP C916, T32
- NEC SX–3, SX–4, Hitachi S820, Fujitsu VP2600

MIMD, distributed memory

- Cray T3D, T3E
- Fujitsu VPP500
- IBM SP–2
- Intel Paragon
- Meiko CS–2
- nCUBE 2s, nCUBE3s
- Parsytec GigaCube GC–2
- Thinking Machines CM–5

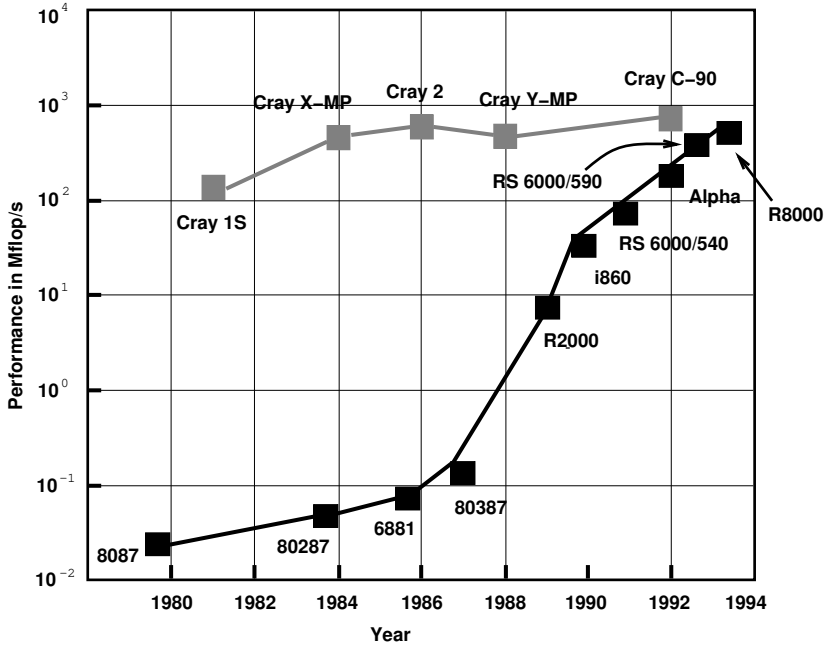
SIMD

- Thinking Machines CM–2
- Maspar MP–2

MIMD, global: Comparison

Computer	cycle time	single CPU peak Mflops/s	#CPUs	peak Mflops/s
Cray–1	12.5 ns	160	1	160
Cray X–MP	8.5 ns	233	4	932
Cray 2	4.1 ns	488	4	1,952
Cray Y–MP	6.0 ns	333	8	2,666
Cray C90	4.2 ns	952	16	15,238
Cray T90	2.2 ns	1,800	32	57,600
NEC SX–3	2.9 ns	5,500	4	22,000
NEC SX–4	4 ns	2,000	32	64,000
Fujitsu VP2600	4.0 ns	4,000	1	4,000
Hitachi S–3800/480		8,000	4	32,000

Growth in Microprocessor Performance in 1980's



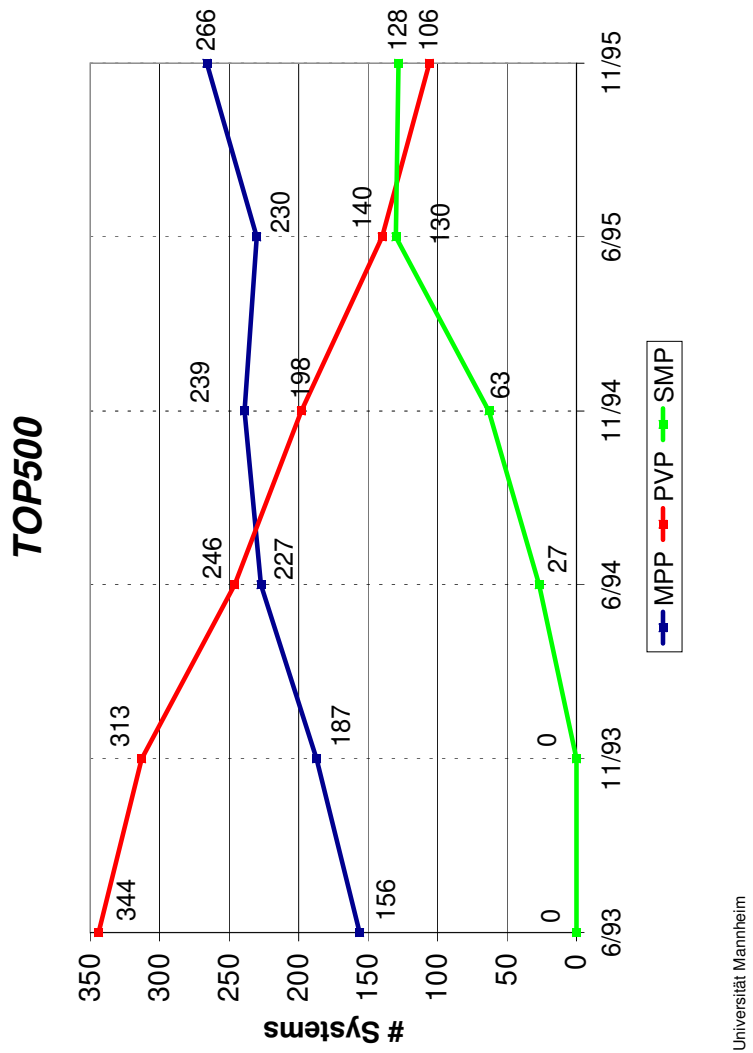
Performance Numbers Comparing Various RISC Processors

Using the Linpack Benchmark

(All based on actual runs.)
Linpack n=100 based on Fortran code only.

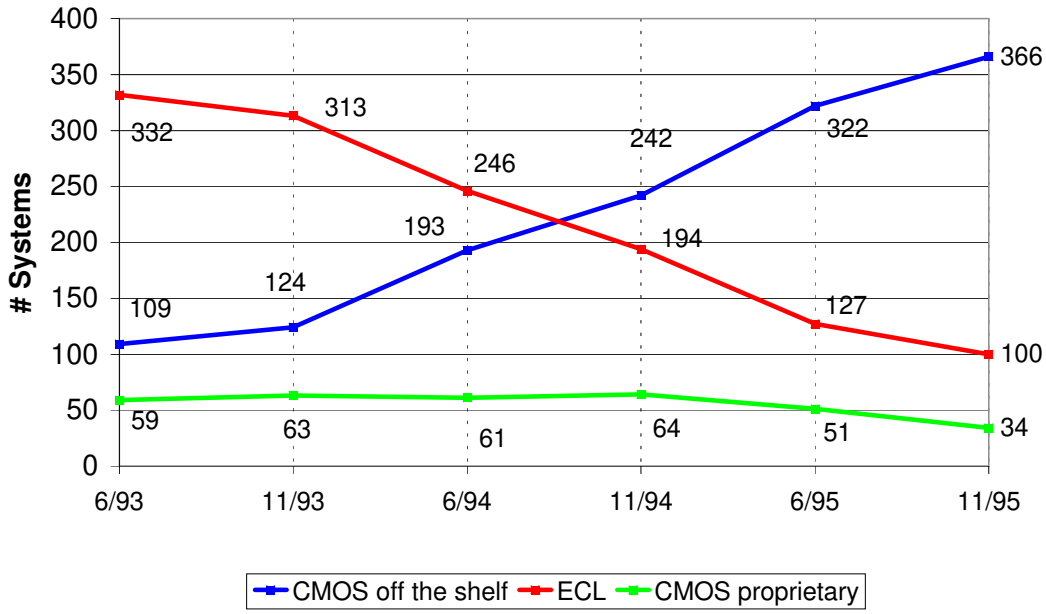
Ax=b n=1000 is based on a blocked algorithm (LAPACK routine)
using the Level 3 BLAS as provided by the vendor.

Machine	MHz	Cycle time nsec	Linpack n=100 Mflops	% peak	Ax = b n=1000 Mflops	% peak	Theor Peak Mflops
DEC Alpha	300	3.3	140	23%	411	69%	600
IBM Power2	66	15	130	48%	236	89%	266
SGI Power	75	13.3	101	33%	260	87%	300
HP 735	100	10	61	31%	107	54%	198
DEC Alpha	200	5	43	22%	155	78%	200
DEC Alpha	182	5	39	21%	141	77%	182
IBM RS-580	66	15	38	30%	104	83%	125
DEC Alpha	160	6	36	23%	114	71%	160
DEC Alpha	150	7	30	20%	107	71%	150
IBM RS-550	42	24	26	31%	70	83%	84
HP 730/750	66	15	24	36%	47	71%	66
SGI Indy2/Crim	100	10	15	30%	32	64%	50
IBM RS-530	25	40	15	30%	42	84%	50
KSR (1 proc)	40	25	15	38%	31	78%	40
Intel i860	40	25	10	25%	34	85%	40
CRAY T90	454	2.2	522	29%	1576	88%	1800
CRAY C90	238	4.2	387	41%	902	95%	952
CRAY J90	100	10	115	51%	193	96%	200
CRAY Y-MP	166	6	161	48%	324	97%	333
CRAY X-MP	118	8.5	121	51%	218	93%	235
CRAY 3	481	2.1	241	25%			962
CRAY 2	244	4.1	120	25%	384	79%	488
CRAY 1	80	12.5	27	17%	110	69%	160



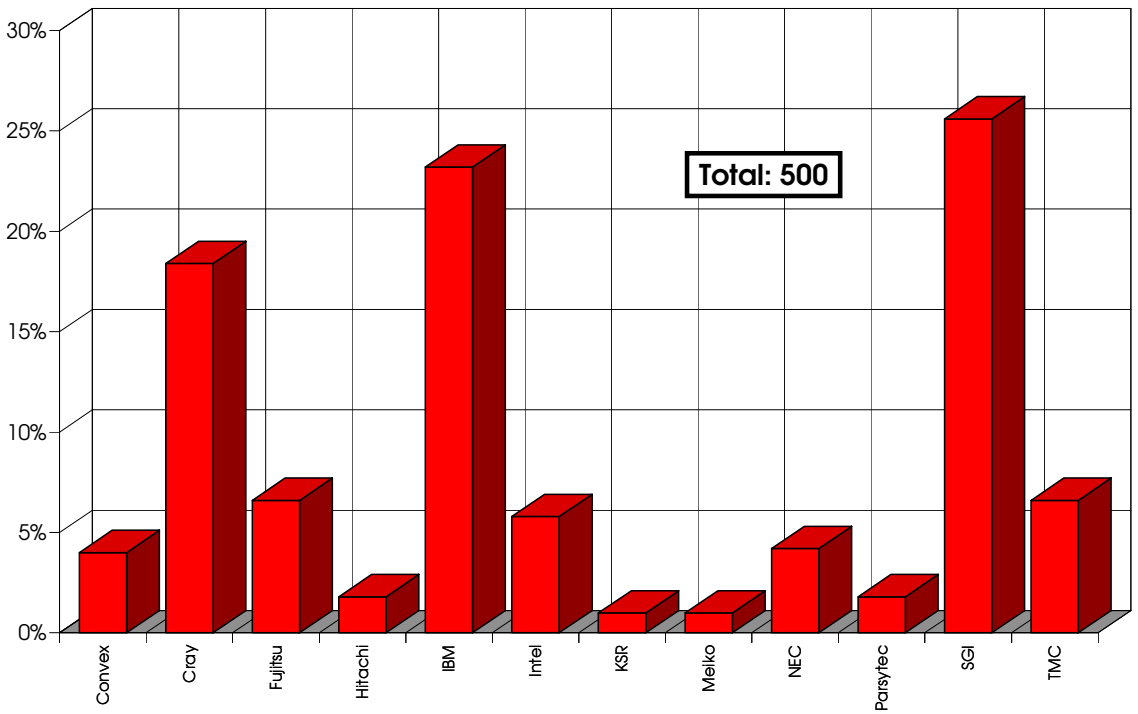
© Universität Mannheim

TOP500 - CPU Technology



© Universität Mannheim

Number of systems installed



Divide

- **Important on Tridiagonal Solver for ADI Schemes.**
- **$9n$ floating point operations in the inner loop, n are floating-point divide instructions.**
- **Also important on block tridiagonal and pentadiagonal solvers.**
- **Newton's iteration usually used**
- **Not typically pieplined**

Processor	Clock (Mhz)	Divide (CP)	Cache BW (MW/s)	Memory BW (MW/s)	Peak Perf. (MF/s)	Linpack (MF/s)
Cray C-90/1	240	4	-	1440	960	387
Cray Y-MP/1	166	4	-	500	333	161
RS/6000-590	66	19	266	266	264	130
DEC Alpha	150	63	150	37	150	30
HP PA-RISC	99	20	99	33	198	41
Intel i860	40	190	80	16	60	10
SGI (R4400)	100	36	100	50	50	17
Super SPARC	40	7	40	40	40	7

Fundamental Architectural Issues

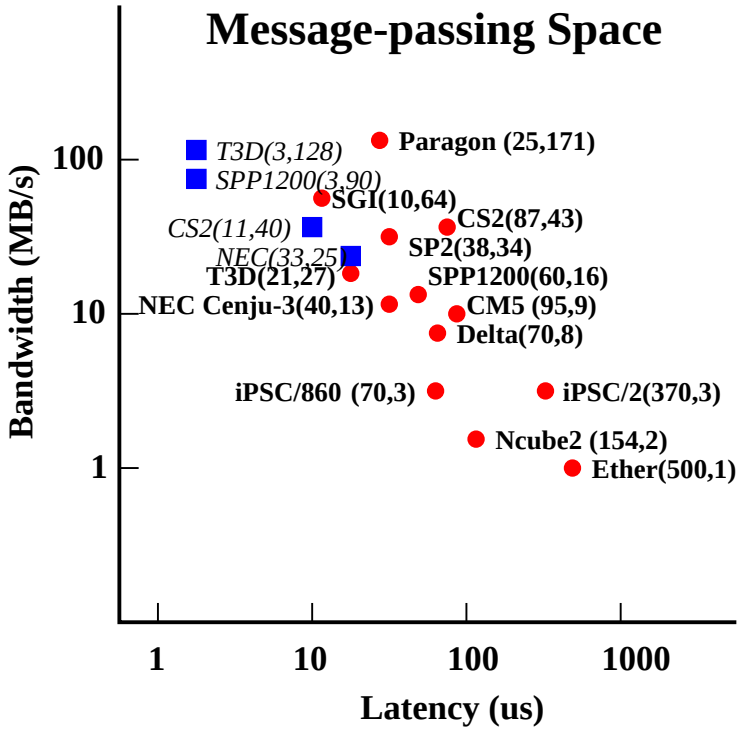
- Naming: How is communicated data and/or partner referenced?
- Synchronization: How can producers and consumers of data coordinate their activities?
- Latency: How long does it take to communicate data in a protected fashion?
- Bandwidth: How much data can be communicated per second? How many operations per second?

Table 1: Multiprocessor Latency and Bandwidth.

Machine	OS	Latency $n = 0$ (μs)	Bandwidth $n = 10^6$ (MB/s)	$n_{1/2}$ bytes	Theoretical Bandwidth (MB/s)
Convex SPP1200 (PVM)	SPP-UX 3.0.4.1	63	15	1000	250
Convex SPP1200 (sm m-n)	SPP-UX 3.0.4.1	11	71	1000	250
Cray T3D (sm)	MAX 1.2.0.2	3	128	363	300
Cray T3D (PVM)	MAX 1.2.0.2	21	27	1502	300
Intel Paragon	OSF 1.0.4	29	154	7236	175
Intel Paragon	SUNMOS 1.6.2	25	171	5856	175
Intel Delta	NX 3.3.10	77	8	900	22
Intel iPSC/860	NX 3.3.2	65	3	340	3
Intel iPSC/2	NX 3.3.2	370	2.8	1742	3
IBM SP-1	MPL	270	7	1904	40
IBM SP-2	MPI	35	35	3263	40
KSR-1	OSF R1.2.2	73	8	635	32
Meiko CS2 (sm)	Solaris 2.3	11	40	285	50
Meiko CS2	Solaris 2.3	83	43	3559	50
nCUBE 2	Vertex 2.0	154	1.7	333	2.5
nCUBE 1	Vertex 2.3	384	0.4	148	1
NEC Cenju-3	Env. Rel 1.5d	40	13	900	40
NEC Cenju-3 (sm)	Env. Rel 1.5d	34	25	400	40
SGI	IRIX 6.1	10	64	799	1200
TMC CM-5	CMMD 2.0	95	9	962	10
Ethernet	TCP/IP	500	0.9		1.2
FDDI	TCP/IP	900	9.7		12
ATM-100	TCP/IP	900	3.5		12

Table 2: Computation Performance.

Machine	OS	Clock cycle MHz	(nsec)	Linpack 100 Mfls	(ops/cl)	Linpack 1000 Mfls	(ops/cl)	Latency us	(cl)
Convex SPP1200 (PVM)	SPP-UX 3.0.4.1	100	(8.33)	65	(.54)	123	(1.02)	63	(7560)
Convex SPP1200 (sm m-n)								11	(1260)
Cray T3D (sm)	MAX 1.2.0.2	150	(6.67)	38	(.25)	94	(.62)	3	(450)
Cray T3D (PVM)								21	(3150)
Intel Paragon	OSF 1.0.4	50	(20)	10	(.20)	34	(.68)	29	(1450)
Intel Paragon	SUNMOS 1.6.2							25	(1250)
Intel Delta	NX 3.3.10	40	(25)	9.8	(.25)	34	(.85)	77	(3080)
Intel iPSC/860	NX 3.3.2	40	(25)	9.8	(.25)	34	(.85)	65	(2600)
Intel iPSC/2	NX 3.3.2	16	(63)	.37	(.01)	–	(–)	370	(5920)
IBM SP-1	MPL	62.5	(16)	38	(.61)	104	(1.66)	270	(16875)
IBM SP-2	MPI	66	(15.15)	130	(1.97)	236	(3.58)	35	(2310)
KSR-1	OSF R1.2.2	40	(25)	15	(.38)	31	(.78)	73	(2920)
Meiko CS2 (MPI)	Solaris 2.3	90	(11.11)	24	(.27)	97	(1.08)	83	(7470)
Meiko CS2 (sm)								11	(990)
nCUBE 2	Vertex 2.0	20	(50)	.78	(.04)	2	(.10)	154	(3080)
nCUBE 1	Vertex 2.3	8	(125)	.10	(.01)	–	(–)	384	(3072)
NEC Cenju-3	Env Rev 1.5d	75	(13.3)	23	(.31)	39	(.52)	40	(3000)
NEC Cenju-3(sm)	Env Rev 1.5d	75	(13.3)	23	(.31)	39	(.52)	34	(2550)
SGI Power Challenge	IRIX 6.1	90	(11.11)	126	(1.4)	308	(3.42)	10	(900)
TMC CM-5	CMMD 2.0	32	(31.25)	–	(–)	–	(–)	95	(3040)



MIMD, multicomputer: networked workstations

Enabling software technology: PVM (Parallel Virtual Machine) available from netlib@ornl.gov

Enabling software technology: MPI (Message Passing Interface) available from netlib@ornl.gov

very active research area; about 150 software products;
catalog available NHSE

Enabling hardware technology: high bandwidth interconnect is not here yet;

Ethernet: msec latencies and 100's of Kbyte/sec bandwidth insufficient

Other technology is on the verge of becoming available:
HIPPI products, Fibre Channel, ATM.

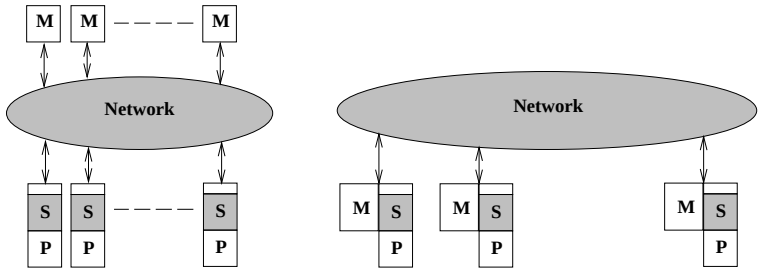
Shared-Memory Implementation

- Shared memory algorithm:
  - Divide cost-array into regions
    - logically assign regions to PEs
  - Assign wires to PEs based on the region in which center lies
  - Do load balancing using stealing when local queue empty
- Good points:
  - Good load balancing
  - Mostly local accesses
  - High cache-hit ratio
- Not so good:
  - non-deterministic
  - potential hot spots
  - amount of parallelism

Message-Passing Implementations

- Solution-1:
  - Distribute wires and cost-array regions as in sh-mem implementation
  - Handle when wire-path crosses to remote region
    - send computation to remote PE, or
    - send messages to access remote data
- Solution-2:
  - Wires distributed as in sh-mem implementation
  - Each PE has copy of full cost array
    - one owned region, plus potentially stale copy of others
    - send frequent updates so that copies not too stale
  - Consequences:
    - waste of memory in replication
    - stale data $\Rightarrow$ poorer quality results or more iterations
- In either case, lots of thinking needed on programmers part

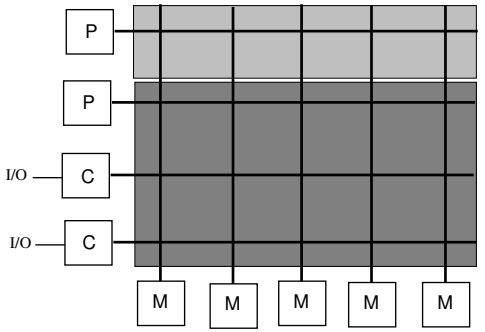
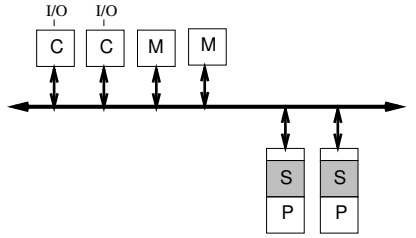
Scalability



- Latency, Bandwidth, and Cost
- Construct shared memory out of simple message transactions across a general–purpose network.
  - read–request, read–response
- Caching?

Hardware Organizations

- "Mainframe" approach
  - motivated by multiprogramming
  - extends xbar
  - latency
  - bandwidth
  - cost



- "Minicomputer" approach
  - motivated by multiprogramming and transaction processing
  - used heavily for parallel comp.
  - latency
  - bandwidth
    >> caching is key
  - cost