Overview of State-of-the-art High-Performance Computer Systems

Fridolin Hofmann
Chair of Computer Science (Operating Systems)
University of Erlangen-Nuremberg, Germany

email: fhofmann@informatik.uni-erlangen.de

1 Introduction

2 Computer models (hardware)

3 Execution models (operating systems)

4 Programming models (languages, libraries)

5 Conclusions

1 General Remarks

❑ High performance computing aims at solving problems that need with today's algorithms

- higher computing power
- much more primary storage
- higher transfer rates between primary and secondary storage

than offered by traditional single computer systems.

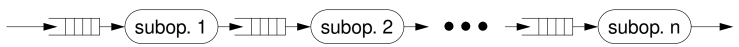
❑ Possibilities for solving such problems:

- developing new algorithms (for example multigrid methods and adaptive grids)
- computing subtasks in parallel.

1 Forms of Parallelism

❑ Main forms of parallelism

- pipelining
  - same operation on a **series** of data values
    operation divided into a series of suboperations



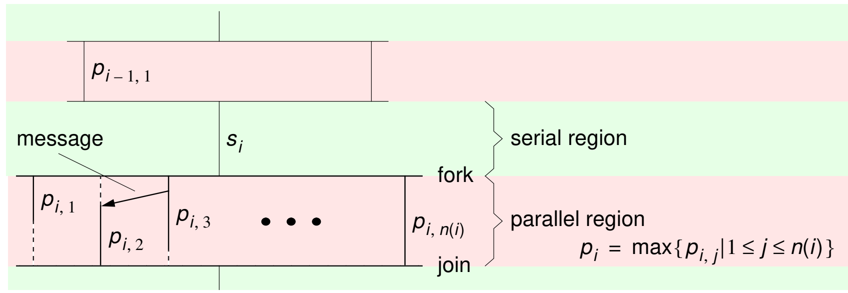
  - ideally all suboperations have the same execution time so that no queues are needed
  - example: vector operations
- concurrency
  - several (possibly different) operations are executed at the same time
  - special forms

    data parallel: same operation on a **set** of data values

    single program multiple data (SPMD): same program on a set of data values
  - example: ray tracing

1 Parallelism and Amdahl's Law

Typical structure of a program using concurrency



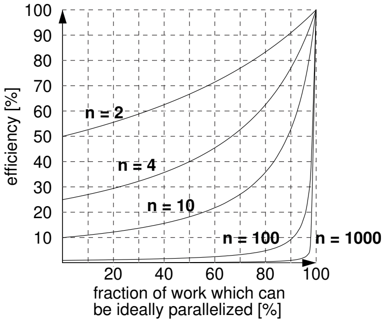
Ideally $n(i)$ equals the number of available processing elements and $p_{i,j}$ are equal for $1 \le j \le n(i)$.

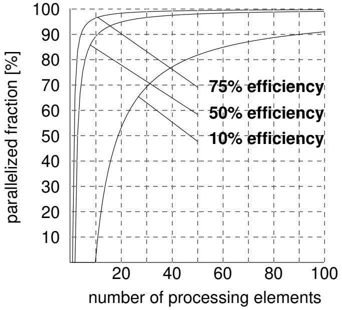
$p_i \le t_{fork} + t_{join}$ fine grained parallelism

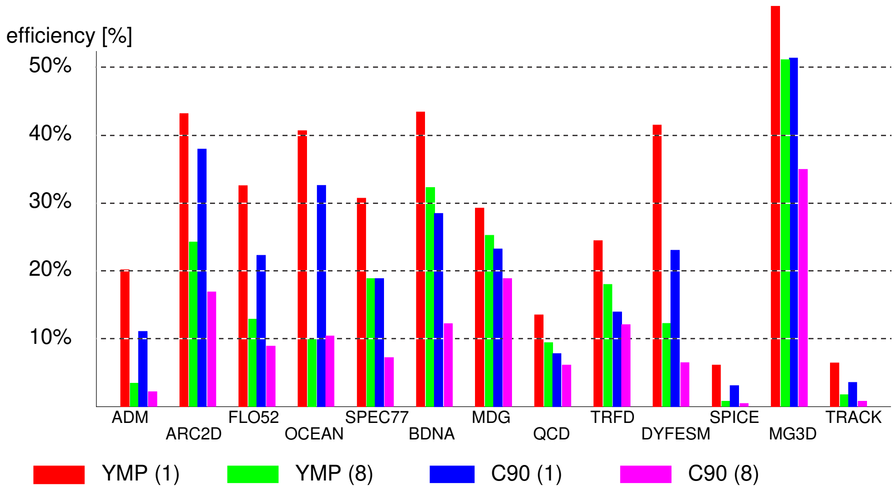
$p_i >> t_{fork} + t_{join}$ coarse grained parallelism

Let n be the number of processing units and p be the fraction of runtime that can be ideally parallelized. Then (neglecting t_{fork} and t_{join}) the overall efficiency is approximated by

$$\text{efficiency} = \frac{1/n}{(1-p)+p/n} = \frac{1}{n(1-p)+p} \qquad \text{(Amdahl's Law)}$$

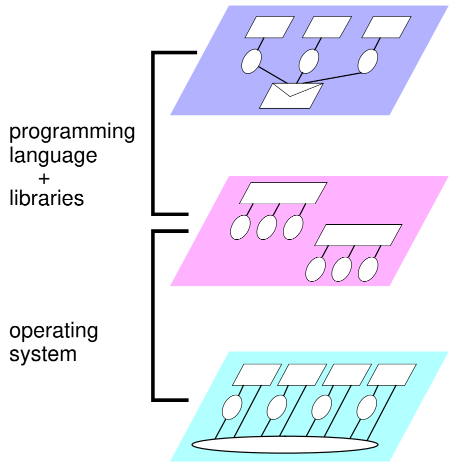






Execution times for an SCF example using different kinds of algorithms to approximate the incomplete gamma function

System	Taylor series	Schaad e. a.	Früchtl; Otto
Cray Y MP, scalar	90.34	121.81	122.66
Cray Y MP, vectorized	24.77	89.70	9.35
Convex C220-256S	115.55	90.83	62.70



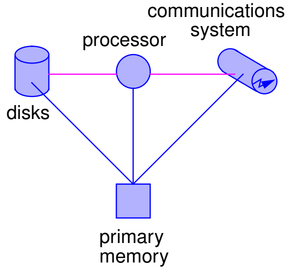
programming model:
communicating sequential processes
thread
address space
mailbox

execution model:
set of multiprocessors
processor
memory (primary and secondary)

computer model:
workstation cluster
processor
memory (primary and secondary)
communications network

❑ Essential components of a classical monoprocessor

- the processor, executing the instructions in the order given by a program
- the primary memory containing program text and data, it is connected to the processor by a bidirectional data transport system
- the peripherals controlled by the processor and connected to the primary memory
- eventually access to a communications system



❑ Architectural extensions

- Pipelining of instruction and/or function execution, multiple functional units
  - ➥ single instruction multiple data (SIMD)
- Replication of components like processors, disks, and whole computers
  - ➥ multiple instruction multiple data (MIMD)

❑ Possibilities for replication of parts internal to a processor are

- pipelining of instruction fetching, decoding, executing, and back writing
- pipelining of functional units operating on sequences of operands (vector processors)
- chaining of several functional units
- replication of functional units (SIMD)

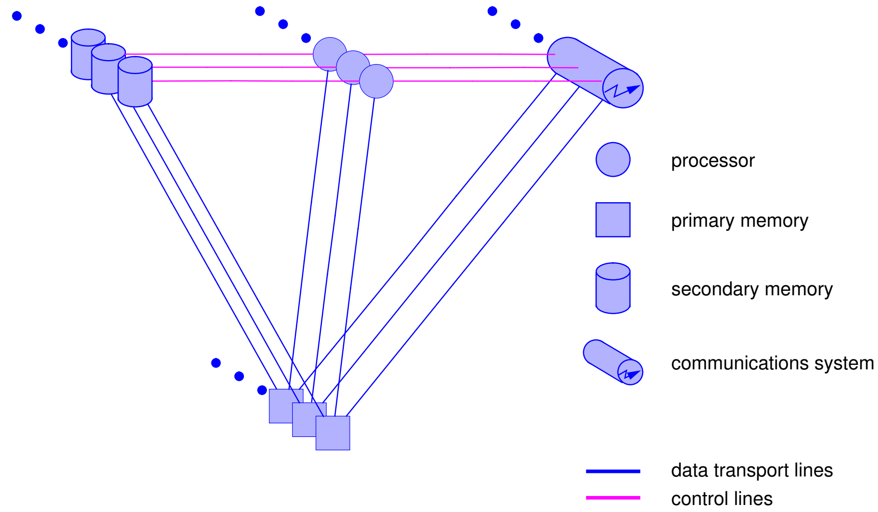
❑ General problems of data transport

- set-up time: scheduling the request
  transmitting request
  preparing to send required data
- transport time (proportional to number of bytes)
- close-down time

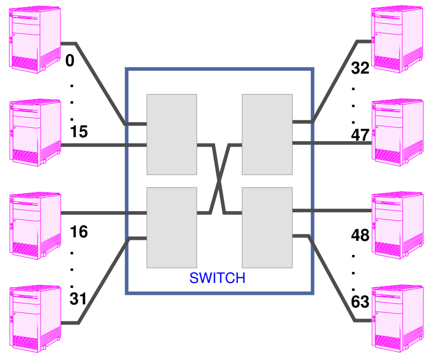
❑ Increased processor performance needs higher effective transport rate between processor and primary memory

- bulk transfer: reduces number of requests to be transmitted
- banking: overlaps set-up and transport times of different requests
- interleaving: allows higher transport rates
- caching: reducing latency through shorter cycle times

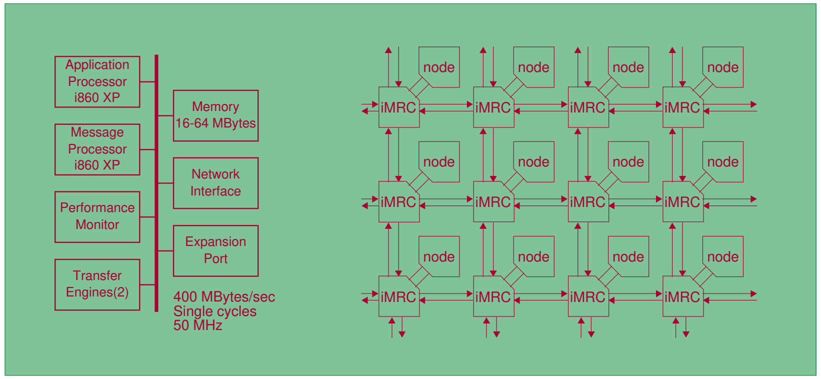
System	Pipes	Peak performance MFLOP/s	comp. LINPACK (100 x 100) MFLOP/s	best effort order : MFLOP/s
Cray C90 (4.2 ns)	2 ADD, 2 MULT	952	387	1000 : 902
Convex C4/XA-1 (7.41 ns)	6 pipes	810	99	1000 : 705
NEC SX-3/14R (2.5 ns)	8 ADD, 8 MULT	6400	368	1000 : 5199
MasPar MP-2204 (80 ns)	4096 PE’s	2400		11264 : 1600 1920 : 800



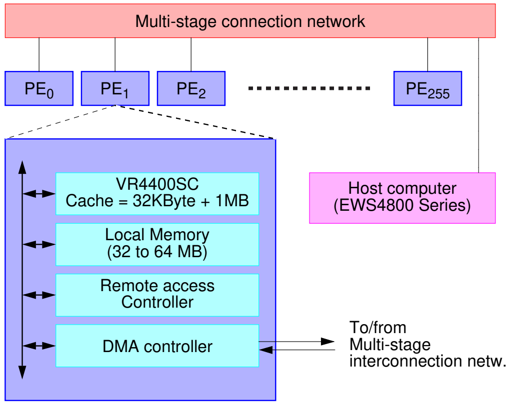
- IBM SP-2 (max. 128 processors)

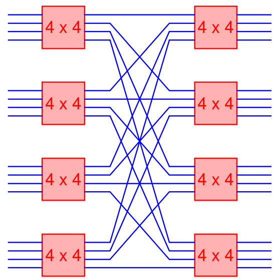


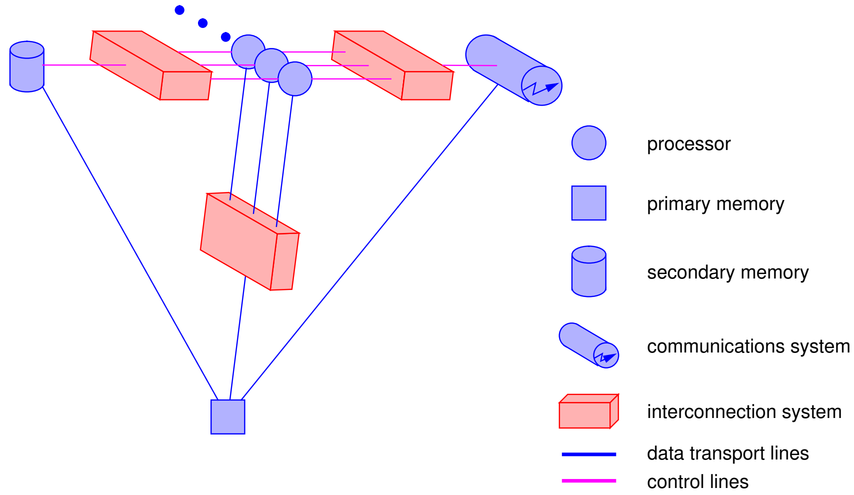
- Paragon



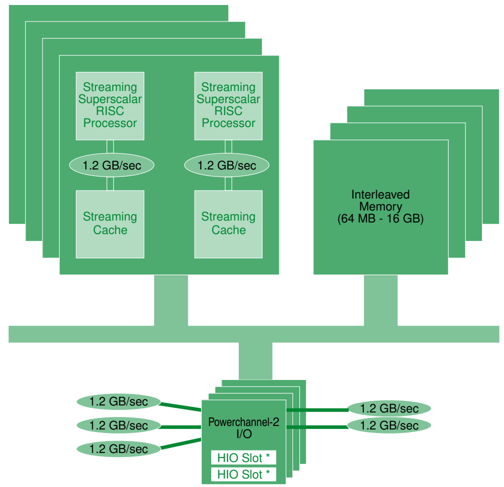
NEC Cenju-3

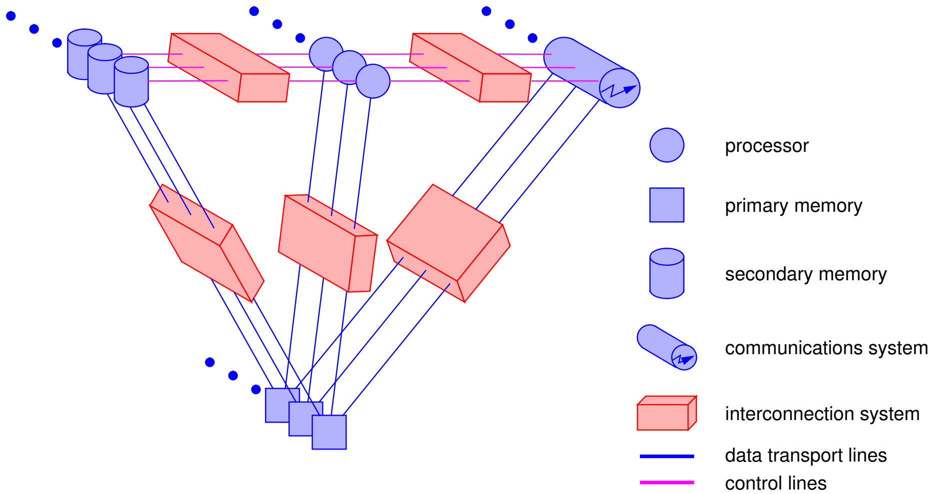




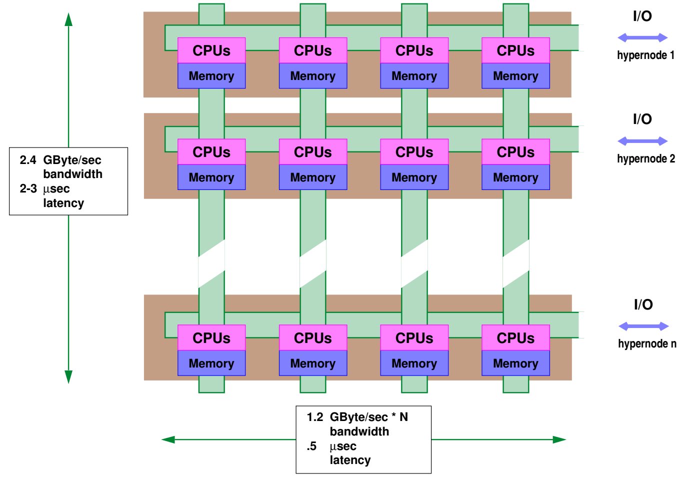


- SGI -Challenge

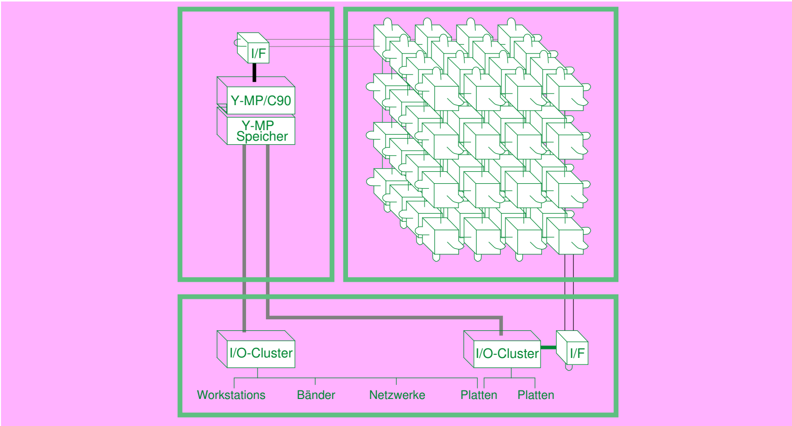




- Convex SPP1 - storage hierarchy



- Cray T3D - as a subsystem attached to an Y-MP



❑ Most important for scientific computing:
connection network between processors and main memory

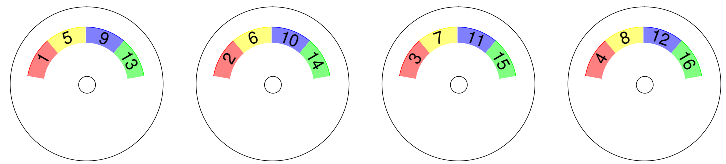
❑ rough classification
- one to one connection
  computer cluster
  no remote memory access (NORMA)
- n to n
  - access time equal for all processor memory pairs (UMA)
  - processors own 'local memory'
    direct access to remote memory possible but slower (NUMA)

❑ Problem
- Some scientific and especially commercial programs are I/O bound
- Over the last 15 years almost no improvement in latency (only about a factor of 2)

❑ Aims
- Reducing mean value of latency
- Increasing transfer rates

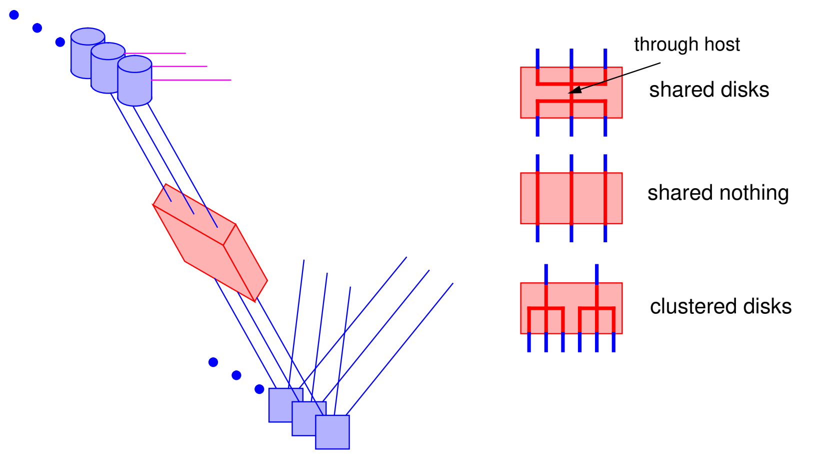
❑ Possible solutions
- Improvements within disk systems
- Independent disk systems working in parallel

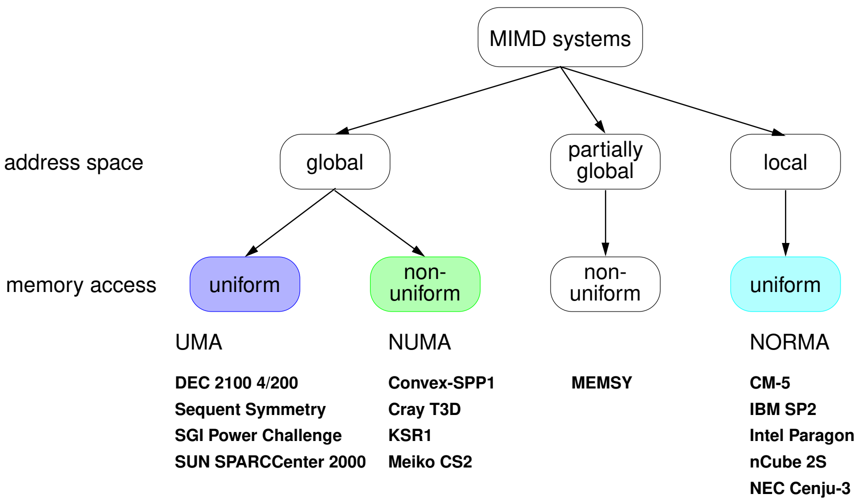
Improvements within disk systems: RAID 0 (Redundant Array of Inexpensive Disks, Level 0)



disk striping
- can reduce meantime of latency, if several I/Os exist at the same time
- can increase transfer rate, if multiple blocks are requested

Using parallel I/O





System	Type	No. proc.	Main memory	Interconnection	Cache i / d KB	Coherence
Digital 2100 4/200	UMA	4	2 GB	shared bus	1 MB	global
Sequent Symmetry 2000	UMA	32	1.5 GB	shared bus	0.256 MB	global
SGI Power Challenge	UMA	18	16 GB	packet oriented bus	2 MB	global
Sun-SPARCCenter 2000	UMA	20	5.12 GB	packet oriented bus	2 MB	global
Convex-SPP1	NUMA	128	32 GB	multi stage network	2 MB	global
Cray T3D	NUMA	2048	128 GB	3-D torus network	8 KB	none
KSR 1 (rip)	NUMA	1088	32 GB	multi stage network	32 MB	global
Meiko CS2	NUMA	256	32 GB	hierarchical crossbar		none
CM-5 (rip)	NORMA	16384	128 MB/proc.	fat tree		
IBM SP2	NORMA	64 (wide)	2 GB/proc.	packet switched multi stage network	8/64 KB	
Intel Paragon	NORMA	4000	128 MB/proc.	2-D mesh	16/16 KB	

Operating systems are serving a twofold purpose:

❏ For a single application they offer an execution model that is easier to handle than the computing model of the underlying hardware.

❏ For better use of the system they multiplex the hardware between different applications by offering so called virtual machines.

Most important criteria to judge new execution models

❏ simplicity

❏ efficiency

❏ portability

❏ compatibility

❏ Unix execution model

- basic concepts
  - processes
  - files
  - interaction mechanisms

    coordination through *semaphores*

    communication through *pipes*, *queues*, *sockets*

    cooperation through *shared memory segments*
- Easily portable
  - to UMA and NUMA
  - to NORMA if processes are used as distribution units and no use is made of shared segments

❏ Mach execution model

- basic concepts
  - tasks
  - threads
  - ports
  - messages
  - memory objects
- Extendable through application level servers
  - Allows providing each node with the features it needs, for example

    Unix server

    File server

The most important criteria new programming models have to fullfil are:

❏ Compatibility

- enormous amount of existing software
- vast amount of knowledge in programming sequential machines
- most important programming models are those underlying Fortran, C and SQL

❏ Conceptual clarity

- few orthogonal concepts
- explainable without recourse to hardware

❏ Portability

- efficient implementable into existing and expected architectures

❏ Efficiency

- If needed, also special features of a system have to be accessible.

❏ Programming models for vector architectures

- aggregates as basic data types: array a(1000), b(1000)
- aggregate operations: a + b, a * b
- examples
  - Vectorizing compilers trying to recognize loops that in fact describe aggregate operations
  - Fortran 90, HPF: language defined aggregate types
  - C++: user definable types

❏ programming models for MIMD architectures

- NORMA
  - components, units for distribution
    - data segments: Single Program Multiple Data (HPF)
    - processes
    - active objects (COOL)
  - interaction
    - message passing (PVM 3, MPI, PARLIB, PARMACS)
- UMA (can be used like NORMA)
  - components: address spaces, threads
  - interaction: shared objects
- NUMA
  - components: address spaces, threads
  - interaction: message passing, shared objects
  - distribution: through defining affinities

❏ Tools

- Interactive debuggers working on source code level
- Performance measuring
  - efficiency of parallel regions
  - load of processors
  - memory access statistics
  - cache hits
  - coordination and communication delays

❏ Unsolved problem: How to support detecting coordination errors?

System	Type	No. proc.	Peak perf. GFLOP/s	Comp. LINPACK order 100 MFLOP/s	Lin. eq. best effort order : GFLOP/s	OS based on	Programming models						
							PVM 3 et al.	SPMD	Fortran	F90	HPF	C	C++
Digital 2100 4/200	UMA	1 4	0.190 0.760	43	1000 : 0.129 1000 : 0.358	Mach	x						
Sequent Symmetry 2000	UMA	32			:	Unix	x		x			x	x
SGI Power Challenge	UMA	1 16	0.300 4.800	101	1000 : 0.260 2000 : 1.812 14000 : 3.624	Unix	x		x			x	x
Sun-SPARCCenter 2000	UMA	1 16	0.050 0.800		1000 : 0.028 1000 : 0.333	Unix	x		x			x	x
Convex-SPP1	NUMA	1 15	0.200 3.000	41	1000 : 0.123 1000 : 0.965	Mach	x		x			x	
Cray T3D	NUMA	1024	152.000		81920 :100.500	Unix	x	x	x	x			
KSR 1 (rip)	NUMA	256	10.240		10240 : 3.380	Mach	x		x			x	x
Meiko CS2	NUMA	64	11.500		18688 : 5.000	Unix	x	x	x	x	x	x	x

System	Type	No. proc.	Peak perf. GFLOP/s	Lin. eq. best effort order : GFLOP/s	OS based on	Programming models						
						PVM 3 et al.	SPMD	Fortran	F90	HPF	C	C++
CM-5 (rip)	NORMA	1024	131.000	52224 : 59.700	Unix			x	x	x	x	x
IBM SP2	NORMA	64 (wide)	17.000	27000 : 12.100								
Intel Paragon	NORMA	6768	338.000	128600 : 281.100	Unix	x		x		x	x	x
nCube 2S	NORMA	1024	3.800	11264 : 1.600	Unix		x	x			x	x
NEC Cenju-3	NORMA	256	12.800		Unix	x		x			x	

- There will always be a need for higher computation power than achieved by monoprocessors, so using parallel systems will be unavoidable.
- In developing new programs right from the beginning one should have in mind to
  - explicitly formulate maximal parallelism
  - achieve a high degree of locality
  - hide latency when accessing remote data
- This may be done by
  - using well established libraries like PVM 3, MPI, and so on
  - using languages like HPF
  - using object oriented methods
- There will remain the problem that in some cases the algorithms have to be chosen in regard to the target computing system.

As message passing is supported by all operating systems it will remain the most common programming model at least for the near future.

Presently the state of the art necessitates parallel programming on a case by case (i. e. application) basis.'

(Polychronopoulos, 1993)

But not necessarily on a machine by machine basis!