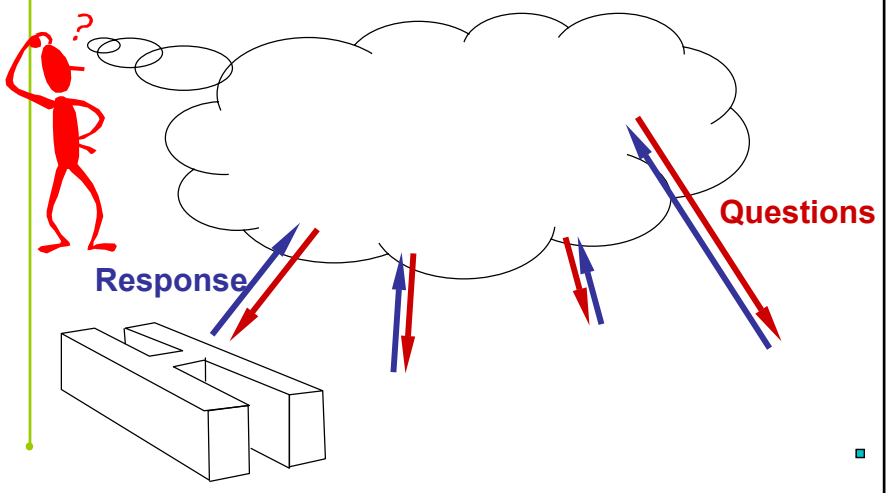
Parallel Programming and Computing

Rolf Rabenseifner
rabenseifner@hlrs.de
www.hlrs.de/people/rabenseifner

University of Stuttgart
High-Performance Computing-Center Stuttgart (HLRS)
www.hlrs.de

Lecture at Hanoi University of Science,
Faculty of Mathematics, Mechanics and Computer Science, March 17, 2003

Motivation



Outline

- Parallel High Performance Computing Platforms in Germany
- Parallel Hardware Architectures
- Parallel Programming Models – an Overview
- Message Passing Interface (MPI) – more Details
- Parallel Programming Models – a Comparison
- Online Parallel Programming Workshop

High Performance Computing in Germany

Federal

- HLRS/HWW Stuttgart – www.hlrs.de
- NIC Jülich – www.kfa-juelich.de/nic
- LRZ/HLRB Munich – www.lrz-muenchen.de

Thematic (federal)

- RZ Garching – www.rzg.mpg.de
- DKRZ/HLRE Hamburg – www.dkrz.de
- DWD Offenbach – www.dwd.de/de/Technik/Datenverarbeitung/

Regional

- ZIB Berlin – www.zib.de
- RRZN Hannover – www.rrzn.uni-hannover.de

See also

www.hlrs.de/hpc/centers.html and www.top500.org

TOP 500 — Nov. 2002 List — German Sites, 1st part

federal
thematic
regional

Rank	Installation Site	Location	R_{max} Gflop/s	Peak Gflop/s	Manufacturer Computer / Procs	Year of Installation
21	Max-Planck-Gesellschaft MPI/IPP	Garching	**2010** =51%	3910	IBM pSeries 690 Turbo 1.3GHz/ **752**	2002
27	Leibniz Rechenzentrum	München	**1653** =82%	2016	Hitachi SR8000-F1/168/ **168**	2000 / 2002
31	Deutscher Wetterdienst	Offenbach	**1293** =67%	1920	IBM SP Power3 375MHz 16way/ **1280**	2001
44	HLRN at Universitaet Hannover / RRZN	Hannover	**1038** =52%	1996	IBM pSeries 690 Turbo 1.3GHz/ **384**	2002
45	HLRN at ZIB/Konrad Zuse-Zentrum fuer Informationstechnik	Berlin	**1038** =52%	1996	IBM pSeries 690 Turbo 1.3GHz/ **384**	2002
49	DKRZ - Deutsches Klimarechenzentrum	Hamburg	**982** =96%	1024	NEC SX-6/128M16/ **128**	2002
64	Universitaet Heidelberg - IWR	Heidelberg	**825** =58%	1430	Megware HELICS AMD1.4GHz Myrinet/ **512**	2002
84	Deutscher Wetterdienst	Offenbach	**671** =69%	974	Cray Inc. T3E1200/ **812**	1999
135	Forschungszentrum Juelich	Jülich	**447** =69%	648	Cray Inc. T3E1200/ **540**	1999
173	Universitaet Aachen/RWTH	Aachen	**357** =69%	518	Sun Fire 15K/ **288**	2002

TOP 500 — Nov. 2002 List — German Sites, 2nd part

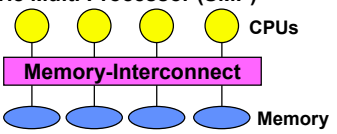
federal
thematic
regional

Rank	Installation Site	Location	R_{max} Gflop/s	Peak Gflop/s	Manufacturer Computer / Procs	Year of Installation
176	Max-Planck-Gesellschaft MPI/IPP	Garching	**355** =73%	487	Cray Inc. T3E/ **812**	1997
181	HWW/Universitaet Stuttgart	Stuttgart	**341** =70%	486	Cray Inc. T3E900/ **540**	1996
183	ZIB/Konrad Zuse-Zentrum fuer Informationstechnik	Berlin	**341** =70%	486	Cray Inc. T3E900/ **540**	1999
302	Theoretical Chemistry, Ruhr-University Bochum	Bochum	**235.8** =66%	358.4	Megware AMD Athlon MP1600+ - SCI 2D-Torus/ **128**	2002
313	Universitaet Darmstadt	Darmstadt	**234.0** =47%	499.2	IBM pSeries 690 Turbo 1.3GHz GigEth/ **96**	2002
315	Forschungszentrum Juelich	Jülich	**234** =72%	324	Cray Inc. T3E600/ **540**	1996
333	GWDG	Göttingen	**226** =67%	336	IBM SP Power3 375 MHz/ **224**	2001
339	Technische Universitaet Chemnitz	Chemnitz	**221.6** =52%	424.0	Self-made CLIC PIII 800 MHz/ **530**	2000
373	GWDG	Göttingen	**198** =45%	442.4	IBM pSeries 690 1.1GHz GigEth/ **96**	2002
454	Universitaet Aachen/RWTH	Aachen	**197.3**	259.2	Sun Fire 15K/ **144**	2002
455	Universitaet Aachen/RWTH	Aachen	**197.3**	259.2	Sun Fire 15K/ **144**	2002

Architectures

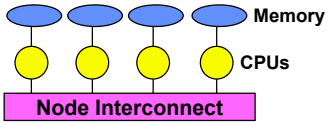
Parallel Vector Processor (PVP) & Symmetric Multi Processor (SMP)

- NEC SX-6, SX-5 and SX-4
- Cray SV1, X1



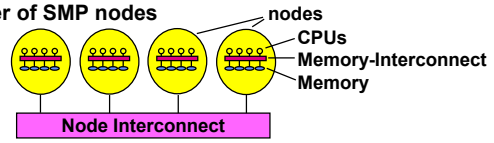
MPP, massively parallel processor achieve

- Cray T3E-1200, 900, 600
- Linux cluster



Hierarchical: Cluster of SMP nodes

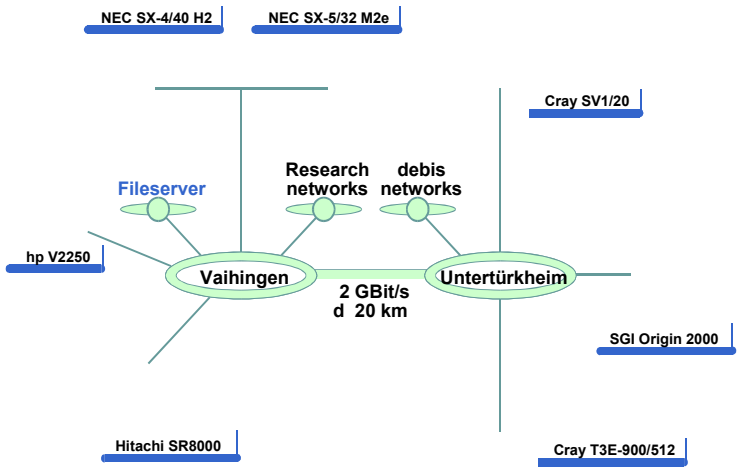
- Hitachi SR8000
- IBM SP
- NEC SX cluster
- Linux cluster



Vector Processing

- some platforms perform vector processing
  - vector processor
  - mainly a question of
    - memory bandwidth
      - how many load/stores per floating point operation?
    - memory-interconnect
  - for which memory access pattern
    - the compiler supports the vector access
    - the hardware supports the vector access

HLRS Stuttgart: hww configuration



HLRS at Stuttgart

- High-Performance Computing-Center Stuttgart
  Höchstleistungsrechenzentrum Stuttgart (HLRS)
  at Rechenzentrum Universität Stuttgart (RUS)
- Platforms:
  - NEC SX-4/40 and SX-5/32
  - Hitachi SR 8000
  - Cray T3E 900-512
- Online-Proposals:
  - www.hlrs.de
    —> HW&Access —> Access
    —> proposal

Cray T3E 900-512
512 PEs, 461 Gflop/s
64 GByte

NEC SX-5/32 M2e
2*16 PEs, 128 Gflop/s
80 GByte

NEC SX-4/40 H2
32+8 PEs, 80 Gflop/s
8+8 GByte

Hitachi SR8000
16*8 PEs, 128 Gflop/s
128 GByte

HLRS Structure — Parallel Computing

- Visualisation
- **Parallel Computing**
- Numerical Methods and Libraries
- Applications
- Parallel and Distributed Systems
- Technical and Scientific Computing
- HPCN Production

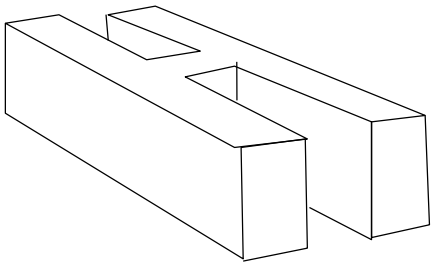
- Parallel programming models
  - MPI, MPI+OpenMP, HPF, UPC, ...
- Tools
  - Vampir, automatic counter profiling
- Parallel benchmarking
  - communication *b_eff,* MPI-I/O *b_eff_io*
  - application benchmarking with PARAPYR, URANUS
- Parallelization consulting
- Consulting for federal projects on hww systems
- Parallelization workshops and courses
- Application Projects (DECAST, SFB 259, Parapyr)

Research & Teaching & Consulting

NIC at Jülich

- John von Neumann-Institut für Computing (NIC) at Forschungszentrum Jülich
- Platforms:
  - IBM Regatta (2x32 CPUs, Installation of 5 Tflop/s in 2003)
  - Cray T3E 1200-512
  - Cray T3E 600-512
  - Cray SV1ex
- Online-Proposals:
  - www.fz-juelich.de/nic/nic-e.html —> Computing Time
- Deadlines
  - ~ April 30, for accounts valid July – June (next year)
  - ~ Oct. 31, for accounts valid Jan. – Dec. (next year)

LRZ / HLRB at Munich: Hitachi SR8000



- System:
  - 168 nodes,
  - 2.02 TFLOP/s peak
  - 1.65 TFLOP/s Linpack
  - 1.3 TB memory
- Node:
  - 8 CPUs, 12 GFLOP/s
  - 8 GB, SMP
  - pseudo-vector
  - ext. b/w: 950 MB/s
- CPU:
  - 1.5 GFLOP/s, 375 MHz
  - 4 GB/s memory b/w
- Installed: 1.Q 2000 at LRZ (112 nodes)

Hitachi SR 8000-F1/112 (Rank 5 in TOP 500 / June 2000)

- Höchstleistungsrechner in Bayern (HLRB) at Leibniz-Rechenzentrum (LRZ)
- Platform:
  - Hitachi SR8000-F1 (168 nodes, each with 8 processors)
- Online-Proposals:
  - www.lrz.de/services/compute/hlrb/
    —> Online Project Proposal Submission

High Performance Computing in Germany — Summary

Federal
- HLRS/HWW Stuttgart
- NIC Jülich
- LRZ/HLRB Munich

Thematic (federal)
- RZ Garching
- DKRZ Hamburg
- DWD Offenbach

Regional
- ZIB Berlin
- RRZN Hannover

See also
www.hlrs.de/hpc/centers.html

H L R S www.hlrs.de

NIC www.fz-juelich.de/nic/nic-e.html

lrz www.lrz.de/services/compute/hlrb/

Outline

- Parallel High Performance Computing Platforms in Germany
- **Parallel Hardware Architectures**
- Parallel Programming Models – an Overview
- Message Passing Interface (MPI) – more Details
- Parallel Programming Models – a Comparison
- Online Parallel Programming Workshop

Concepts

Parallel Processing concepts:

- Pipelining -> vector computing
- Functional Parallelism -> modern processor technology
- Combined instructions -> e.g. multiply-add as one instruction
- Multithreading
- Array-Processing
- Multiprocessors (strongly coupled) -> Shared memory
- Multicomputers (weakly coupled) -> Distributed memory

} Hybrid architectures

Memory access concepts:

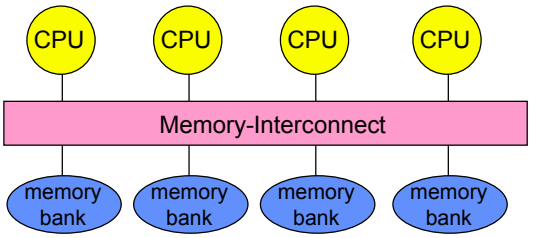
- Cache based
- Vector access via several memory banks
- Pre-load, pre-fetch

—> MFLOP/s performance **and** MB/s or Mword/s memory bandwidth

Major Parallel Hardware Architectures

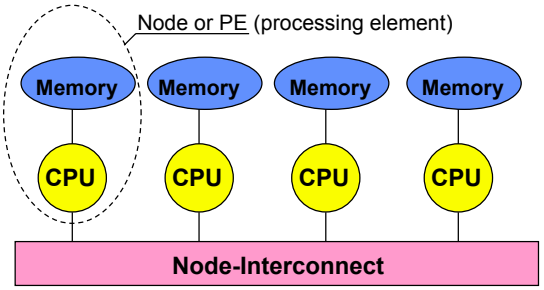
- Shared Memory
  - SMP = symmetric multiprocessing
- Distributed Memory
  - DMP = distributed memory parallel
- Hierarchical memory systems
  - combining both concepts

Multiprocessor - shared memory



- All CPUs are connected to all memory banks with same speed
- **Uniform Memory Access (UMA)**
- **Symmetric Multi-Processing (SMP)**
- Network types, e.g.
  - Crossbar → independent access from each CPU
  - BUS → one CPU can *block* the memory access of the other CPUs

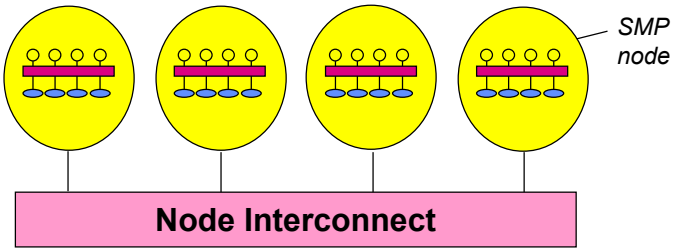
Multicomputer - distributed memory



- Nodes are coupled by a node-interconnect
- Each CPU: – Fast access to its own memory
  – but slower access to other CPU's memories
- **Non-Uniform memory Access (NUMA)**
- Different network types, e.g. BUS, torus, crossbar

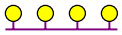
Hybrid architectures

- Most modern high-performance computing (HPC) systems are clusters of SMP nodes



- SMP (symmetric multi-processing) inside of each node
- DMP (distributed memory parallelization) on the node interconnect

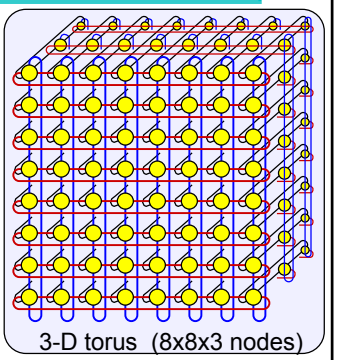
Interconnects

- Memory interconnect
  - bus
  - cross-bar
- Node interconnect
  - bus based networks
    - 
  - multi-link networks, **e.g.,**
    - **ring with independent connections** 
  - 2-D or 3-D torus
    - **each processor is connected by a link with 4 or 6 neighbors**
  - hierarchical networks
    - **multi-level cross-bars**
  - cross-bar (single level)
  - full interconnect

cheap, but poor inter-connect

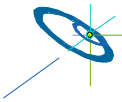
scalable network costs, high accumulated bandwidth

not scalable! — n*(n-1)/2 links


3-D torus (8x8x3 nodes)

Other Architectures

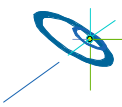
- ccNUMA (cache coherent non-uniform memory access)
  - a distributed (hybrid) architecture
  - looks like one big SMP
  - programmable like one big SMP
  - but cluster of several small SMPs in reality
  - cache coherent
  - programming:
    - **global access with same load/store instruction as local**
    - **parallelization, e.g., with OpenMP**
- ccNUMA with >500 CPUs and multi-level network
    - **parallelization, e.g., with Multi Level Parallelism (MLP)**
- DMP with RDMA (remote direct memory access)
  - programming:
    - **global memory access with special instructions, but without OS**
    - **e.g. Co-array Fortran, UPC (Universal Parallel C), shmem**
- MTA (multi-threaded architecture)



Why?

- Why should I use parallel hardware architectures?

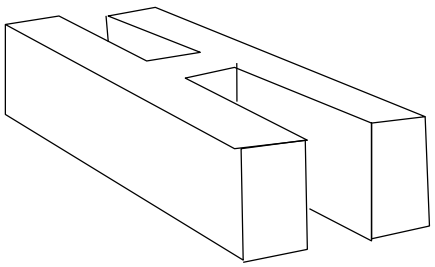
- Possible answers:
  - The response of only one processor is **not** just in time
  - Moore's Law:
    - **The number of transistors on a chip will double approximately every 18 month**
    - **➔ in the future, the number of processors on a chip will grow**
  - You own a
    - **network of workstations (NOW)**
    - **Beowulf-class systems**
      = Clusters of Commercial Off-The-Shelf (COTS) PCs
    - **a dual-board or quad-board PC**
  - Huge application with huge memory needs



Parallel Hardware Architectures

- Ranging from
  - a cluster with PCs
  - or a dual-processor PC
  - …

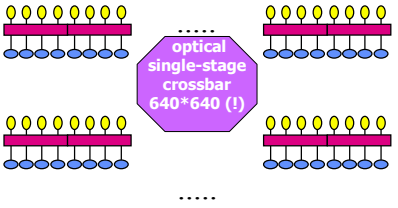
Hitachi SR 8000-F1/112 (Rank 5 in TOP 500 / June 2000)



- System:
  - 168 nodes,
  - 2.016 TFLOP/s peak
  - 1.65 TFLOP/s Linpack
  - 1.3 TB memory
- Node:
  - 8 CPUs, 12 GFLOP/s
  - 8 GB, SMP
  - pseudo-vector
  - ext. b/w: 950 MB/s
- CPU:
  - 1.5 GFLOP/s, 375 MHz
  - 4 GB/s memory b/w
- Installed: 1.Q 2000 at LRZ
- Extended: 1.Q. 2002
  (from 112 to 168 nodes)

Earth Simulator Project ESRDC / GS 40 (NEC)

- Virtual Earth - simulating
  - Climate change (global warming)
  - El Niño, hurricanes, droughts
  - Air pollution (acid rain, ozone hole)
  - Diastrophism (earthquake, volcanism)
- Installation: 2002
  http://www.es.jamstec.go.jp/

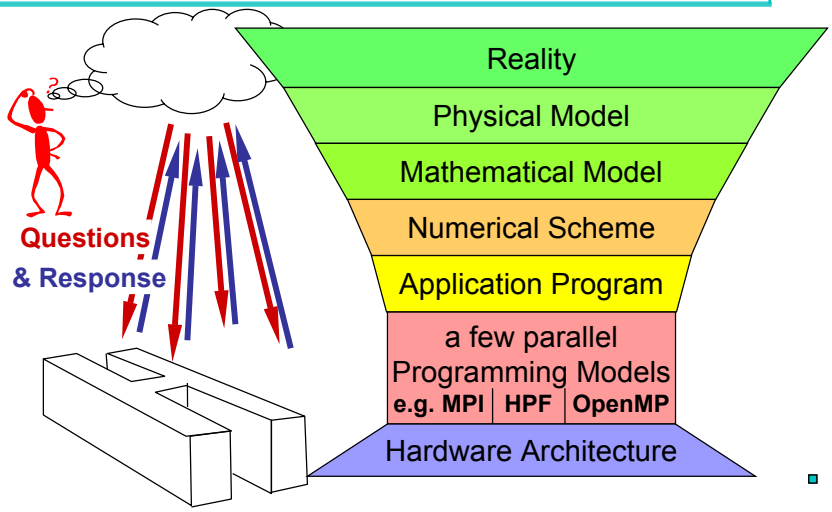


- System: 640 nodes, 40 TFLOP/s
  10 TB memory
  optical 640x640 crossbar
  50m x 20m without peripherals
- Node: 8 CPUs, 64 GFLOP/s
  16 GB, SMP
  ext. b/w: 2x16 GB/s
- CPU: Vector
  8 GFLOP/s, 500 MHz
  Single-Chip, 0.15 µs
  32 GB/s memory b/w

Outline

- Parallel High Performance Computing Platforms in Germany
- Parallel Hardware Architectures
- **Parallel Programming Models – an Overview**
- Message Passing Interface (MPI) – more Details
- Parallel Programming Models – a Comparison
- Online Parallel Programming Workshop

Abstract Model



Parallelization strategies — hardware resources

- Two major resources of computation:
  - processor
  - memory
- Parallelization means
  - **distributing work** to processors
  - **distributing data** (if memory is distributed)

  and
  - **synchronization** of the distributed work
  - **communication** of *remote* data to *local* processor (if memory is distr.)
- Programming models offer a combined method for
  - distribution of work & data, synchronization and communication

Distributing Work & Data

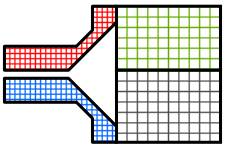
Work decomposition
- based on loop decomposition

do i=1,100
➔ i=1,25
i=26,50
i=51,75
i=76,100

Data decomposition
- all work for a local portion of the data is done by the local processor

A(1:20, 1: 50)
A(1:20, 51:100)
A(21:40, 1: 50)
A(21:40, 51:100)

Domain decomposition
- decomposition of work and data is done in a higher model, e.g. in the reality



Synchronization

```
Do i=1,100                    i=1..25 | 26..50 | 51..75 | 76..100
    a(i) = b(i)+c(i)              execute on the 4 processors
Enddo
                              BARRIER synchronization
Do i=1,100
    d(i) = 2*a(101-i)         i=1..25 | 26..50 | 51..75 | 76..100
Enddo                             execute on the 4 processors
```

- **Synchronization**
  - is necessary
  - may cause
    - **idle time on some processors**
    - **overhead to execute the synchronization primitive**

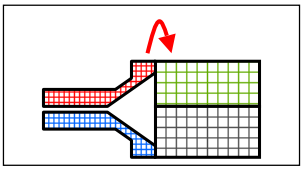
Communication

```
Do i=2,99
    b(i) = a(i) + f*(a(i-1)+a(i+1)-2*a(i))
Enddo
```

- **Communication** is necessary on the boundaries
  - e.g. b(26) = a(26) + f*(a(25)+a(27)-2*a(26))

a(1:25),	b(1:25)
a(26,50),	b(51,50)
a(51,75),	b(51,75)
a(76,100),	b(76,100)

  - e.g. at domain boundaries



Major Programming Models

1. **OpenMP**
   - Shared Memory **Directives**
   - to define the work decomposition
   - no data decomposition
   - synchronization is implicit (can be also user-defined)
- **HPF (High Performance Fortran)**
  - Data Parallelism
  - User specifies data decomposition with **directives**
  - Communication (and synchronization) is implicit
- **MPI (Message Passing Interface)**
  - User specifies how work & data is distributed
  - User specifies how and when communication has to be done
  - by calling MPI communication **library-routines**

Shared Memory Directives – OpenMP, I.

```
Real :: A(n,m), B(n,m)

!$OMP PARALLEL DO
do j = 2, m-1
  do i = 2, n-1
    B(i,j) =  ... A(i,j)
              ... A(i-1,j) ... A(i+1,j)
              ... A(i,j-1) ... A(i,j+1)
  end do
end do
!$OMP END PARALLEL DO
```

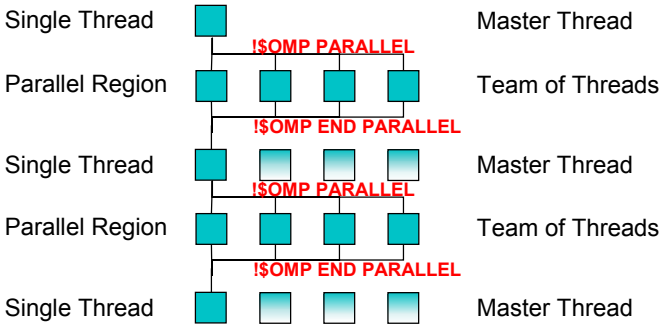
Data definition

Loop over y-dimension

Vectorizable loop over x-dimension

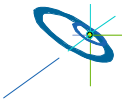
Calculate B,
using upper and lower,
left and right value of A

Shared Memory Directives – OpenMP, II.



Shared Memory Directives – OpenMP, III.

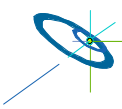
- OpenMP
  - **standardized shared memory parallelism**
  - **thread-based**
  - **the user has to specify the work distribution explicitly with directives**
  - **no data distribution, no communication**
  - **mainly loops can be parallelized**
  - **compiler translates OpenMP directives into thread-handling**
  - **standardized since 1997**
- Automatic SMP-Parallelization
  - **e.g., Compas (Hitachi), Autotasking (NEC)**
  - **thread based shared memory parallelism**
  - **with directives (similar programming model as with OpenMP)**
  - **supports automatic parallelization of loops**
  - **similar to automatic vectorization**



Major Programming Models – HPF

1. **OpenMP**
   - Shared Memory **Directives**
   - to define the work decomposition
   - no data decomposition
   - synchronization is implicit (can be also user-defined)
2. **HPF (High Performance Fortran)**
   - Data Parallelism
   - User specifies data decomposition with **directives**
   - Communication (and synchronization) is implicit

- **MPI (Message Passing Interface)**
  - User specifies how work & data is distributed
  - User specifies how and when communication has to be done
  - by calling MPI communication **library-routines**



Data Parallelism – HPF, I.

```
Real :: A(n,m), B(n,m)                    ➡ Data definition
!HPF$ DISTRIBUTE A(block,block), B(...)

do j = 2, m-1                             ➡ Loop over y-dimension
  do i = 2, n-1                           ➡ Vectorizable loop over x-dimension
    B(i,j) = ... A(i,j)                   ➡ Calculate B,
             ... A(i-1,j) ... A(i+1,j)        using upper and lower,
             ... A(i,j-1) ... A(i,j+1)            left and right value of A
  end do
end do
```


Data Parallelism – HPF, II.

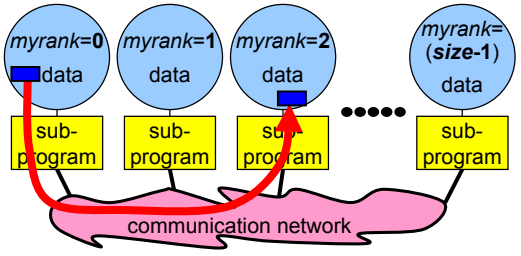
- HPF (High Performance Fortran)
  - standardized data distribution model
  - the user has to specify the data distribution explicitly
  - Fortran with language extensions and directives
  - compiler generates message passing or shared memory parallel code
  - work distribution & communication is implicit
  - set-compute-rule:
    the owner of the left-hand-side object computes the right-hand-side
  - typically arrays and vectors are distributed
  - draft HPF-1 in 1993, standardized since 1996 (HPF-2)
  - JaHPF since 1999

Major Programming Models – MPI

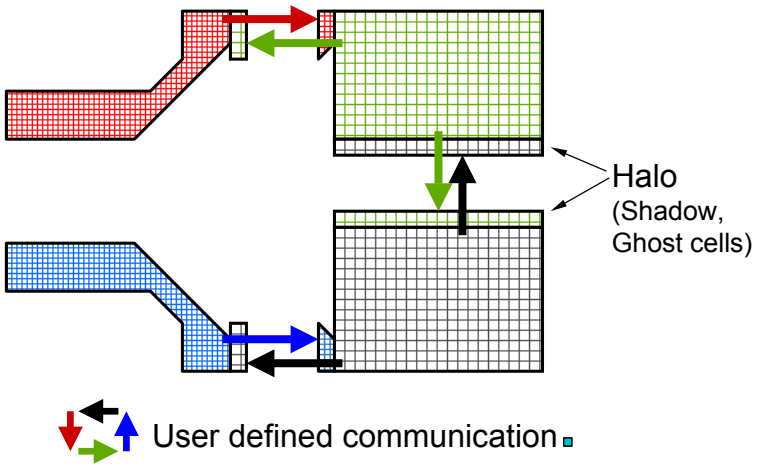
1. **OpenMP**
   - Shared Memory **Directives**
   - to define the work decomposition
   - no data decomposition
   - synchronization is implicit (can be also user-defined)
2. **HPF (High Performance Fortran)**
   - Data Parallelism
   - User specifies data decomposition with **directives**
   - Communication (and synchronization) is implicit
3. **MPI (Message Passing Interface)**
   - User specifies how work & data is distributed
   - User specifies how and when communication has to be done
   - by calling MPI communication **library-routines**

Message Passing Program Paradigm – MPI, I.

- Each processor in a message passing program runs a ***sub-program***
  - written in a conventional sequential language, e.g., C or Fortran,
  - typically the same on each processor (SPMD)
- All work and data distribution is based on value of ***myrank***
  - returned by special library routine
- Communication via special send & receive routines (***message passing***)



Additional Halo Cells – MPI, II.



Message Passing – MPI, III.

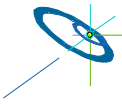
```
Call MPI_Comm_size(MPI_COMM_WORLD, size, ierror)
Call MPI_Comm_rank(MPI_COMM_WORLD, myrank, ierror)
m1 = (m+size-1)/size;   ja=1+m1*myrank;   je=max(m1*(myrank+1), m)
jax=ja-1;  jex=je+1   // extended boundary with halo

Real :: A(n, jax:jex), B(n, jax:jex)          ➡ Data definition
do j = max(2,ja), min(m-1,je)                 ➡ Loop over y-dimension
  do i = 2, n-1                               ➡ Vectorizable loop over x-dimension
    B(i,j) =  ... A(i,j)                      ➡ Calculate B,
              ... A(i-1,j) ... A(i+1,j)       ➡   using upper and lower,
              ... A(i,j-1) ... A(i,j+1)       ➡         left and right value of A
  end do
end do

Call MPI_Send(.....)  ! - sending the boundary data to the neighbors
Call MPI_Recv(.....)  ! - receiving from the neighbors,
                      !   storing into the halo cells
```

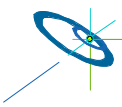
Summary — MPI, IV.

- MPI (Message Passing Interface)
  - standardized distributed memory parallelism with message passing
  - process-based
  - the user has to specify the work distribution & data distribution & all communication
  - synchronization implicit by completion of communication
  - the application processes are calling MPI library-routines
  - compiler generates normal sequential code
  - typically domain decomposition is used
  - communication across domain boundaries
  - standardized
    MPI-1: Version 1.0 (1994), 1.1 (1995), 1.2 (1997)
    MPI-2: since 1997



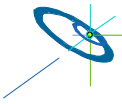
Distribution methods

Decom-position	easiest Model	Memory distribution & communication	Work distribution
• Work	OpenMP	– none –	→ explicit by program (with directives)
• Data	HPF	→ explicit by program (with directives) & implicit comm.	→ implicit by set-compute-rule or explicit with ON directive
• Domain	MPI	→ explicit by program (via process' ranks) & explicit communication (with MPI library routines)	→ explicit by program (via process' ranks)



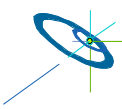
Limitations, I.

- Automatic Parallelization
  - the compiler
    - **has no global view**
    - **cannot detect independencies, e.g., of loop iterations**
    - **➔ parallelizes only parts of the code**
  - only for shared memory and ccNUMA systems, see OpenMP
- OpenMP
  - only for shared memory and ccNUMA systems
  - mainly for loop parallelization with directives
  - only for medium number of processors
  - explicit domain decomposition also via rank of the threads



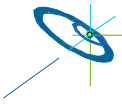
Limitations, II.

- HPF
  - set-compute-rule may cause a lot of communication
  - HPF-1 (and 2) not suitable for irregular and dynamic data
  - JaHPF may solve these problems, but with additional programming costs
  - can be used on any platform
- MPI
  - the amount of your hours available for MPI programming
  - can be used on any platform, but communication overhead on shared memory systems



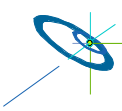
Other Concepts

- shmem and MPI-2 one-sided communication
- Distributed memory programming (DMP) language extensions
  - Co-array Fortran
  - UPC (Unified Parallel C)
- Multi level parallelism (MLP)
- Threads: A single process having multiple execution paths
- Remote Memory Operation: A set of processes in which one process can access the memory of another process without its participation
- Shared Virtual Memory (SVM)
  Software based Distributed Shared Memory (SoftDSM)
  Distributed Virtual Shared Memory (DVSM)



SHMEM - Shared Memory Interface

- SHMEM allows a user to access remote memory locations with shmem_..._put() and shmem_..._get() routines.
- For parallel machines with global address space, this means no OS intervention => high bandwidth and low latency.
- Targeted for SPMD programs.
- No forced syncs: User has control of (and responsibility for) integrity of data from remote transfers.
- High BW, low latency and minimal syncs make SHMEM very fast, but dangerous if not carefully used.
- Cache coherency must be programmed explicitly.
- Example Cray T3E:
  - MPI and SHMEM bandwidth ~ the same
  - MPI latency about 10x longer than SHMEM latency

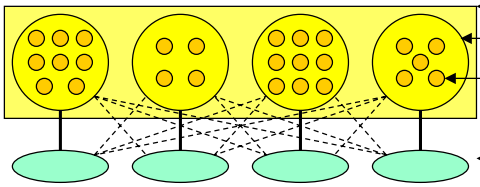


DMP Language Extensions, I.

- Programmable access to the memory of the other processes
- Language bindings:
  - Co-array Fortran
  - UPC (Unified Parallel C)
- Special additional array index to explicitly address the process
- Examples (Co-array Fortran):

```
integer  a[*], b[*]                ! Replicate a and b on all processes
a[1] = b[6]                        ! a on process 1  :=  b on process 6
-------------------------------------------------------------------------
dimension (n,n) :: u[3,*]          ! Allocates the n×n array u
                                   !              on each of  the 3x* processes
p = THIS_IMAGE(u,1)                ! first co-subscript of local process
q = THIS_IMAGE(u,1)                ! second co-subscript of local process
u(1:n,1)[p+1,q] = u(1:n,n)[p,q]    ! Copy right boundary u(1,) on process [p,]
                                   ! to right neighbor [p+1,] into left boundary u(n,)
```

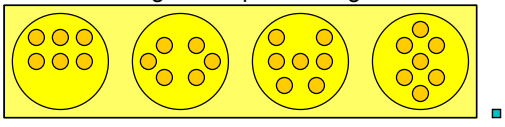

Multi Level Parallelism (MLP)



- program
- processes
- multiple threads inside of each process (**OpenMP**)
- data associated with each process
- **but shared (ccNUMA) access to other processes' data**

Cheap load balancing

- by changing the number of threads per process
- before starting a new parallel region

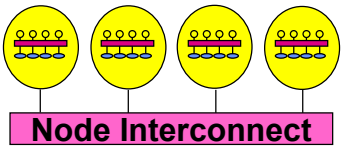


Programming Models on Hardware Platforms

Hardware allows:	→	Usable programming model:
• only reliable message transfer	→	MPI, HPF
• remote DMA **(direct memory access)**	→	´´ , ´´ + SVM, shmem, UPC, Co-array Fortran
• SMP and PVP, MTA, ccNUMA	→	´´ , ´´ + ´´ , ´´ , ´´ , ´´ + OpenMP

Programming Models on Hybrid Systems

- MPI based:
  - the MPP model
    - **massively parallel processing**
    - **each CPU = one MPI process**
  - MPI + OpenMP
    - **each SMP node = one MPI process**
    - **MPI communication on the node interconnect**
    - **OpenMP inside of each SMP node**
    - **DMP with MPI & SMP with OpenMP**
  - MPI + automatic parallelization
    - **Compas on Hitachi, Autotasking on NEC, ...**
    - **same model as MPI+OpenMP**
- Other models:
  - HPF, MLP, ... ▫

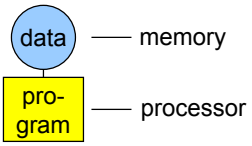


Outline – [Message Passing Interface (MPI)]

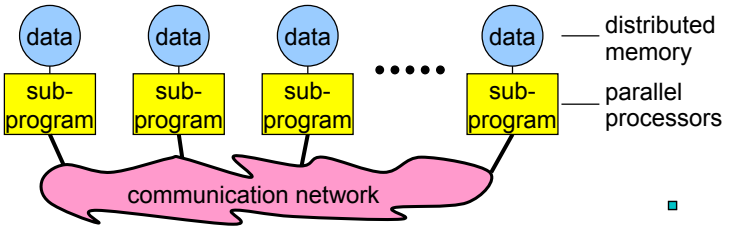
- Parallel High Performance Computing Platforms in Germany
- Parallel Hardware Architectures
- Parallel Programming Models – an Overview
- **Message Passing Interface (MPI) – more Details**
- Parallel Programming Models – a Comparison
- Online Parallel Programming Workshop

The Message-Passing Programming Paradigm

- **Sequential Programming Paradigm**

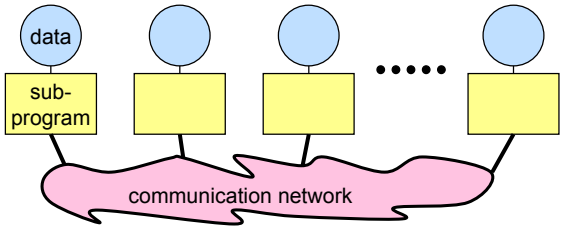


- **Message-Passing Programming Paradigm**



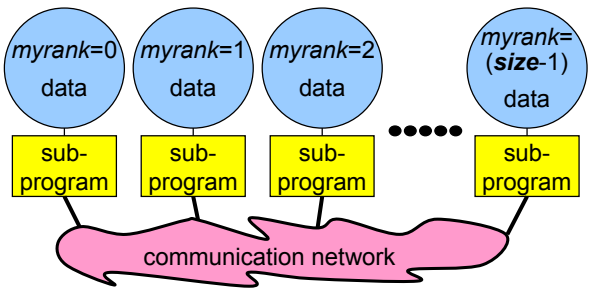
The Message-Passing Programming Paradigm

- Each processor in a message passing program runs a ***sub-program:***
  - written in a conventional sequential language, e.g., C or Fortran,
  - typically the same on each processor (SPMD),
  - the variables of each sub-program have
    - **the same name**
    - **but different locations (distributed memory) and different data!**
    - **i.e., all variables are private**
  - communicate via special send & receive routines (***message passing***)



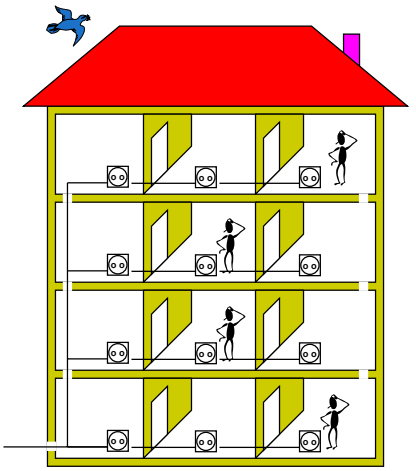
Data and Work Distribution

- the value of ***myrank*** is returned by special library routine
- the system of ***size*** processes is started by special MPI initialization program (mpirun or mpiexec)
- all distribution decisions are based on ***myrank***
- i.e., which process works on which data



Analogy: Electric Installations in Parallel

- MPI sub-program
  = work of one electrician
  on one floor
- data
  = the electric installation
- MPI communication
  = real communication
  to guarantee that the wires
  are coming at the same
  position through the floor



What is SPMD?

- **S**ingle **P**rogram, **M**ultiple **D**ata
- Same (sub-)program runs on each processor

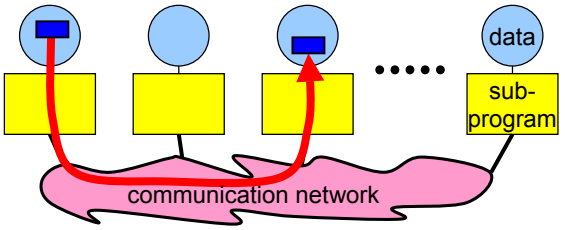
- MPI allows also MPMD, i.e., **Multiple** Program, ...
- but some vendors may be restricted to SPMD
- MPMD can be emulated with SPMD

Emulation of Multiple Program (MPMD), Example

- main(int argc, char **argv)
  {
  if (myrank < .... /* process should run the ocean model */)
  {
  ocean(/* arguments */);
  }else{
  weather(/* arguments */);
  }
  }

- PROGRAM
  IF (myrank < ...) THEN !! process should run the ocean model
  CALL ocean (some arguments)
  ELSE
  CALL weather (some arguments)
  ENDIF
  END

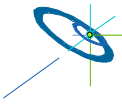
Messages



- Messages are packets of data moving between sub-programs
- Necessary information for the message passing system:
  - sending process – receiving process } i.e., the ranks
  - source location – destination location
  - source data type – destination data type
  - source data size – destination buffer size

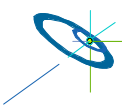
Access

- A sub-program needs to be connected to a message passing system
- A message passing system is similar to:
  - mail box
  - phone line
  - fax machine
  - etc.
- MPI:
  - sub-program must be linked with an MPI library
  - the total program (i.e., all sub-programs of the program) must be started with the MPI startup tool



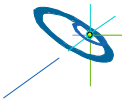
Addressing

- Messages need to have addresses to be sent to.
- Addresses are similar to:
  - mail addresses
  - phone number
  - fax number
  - etc.
- MPI: addresses are ranks of the MPI processes (sub-programs)



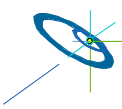
Reception

- All messages must be received.



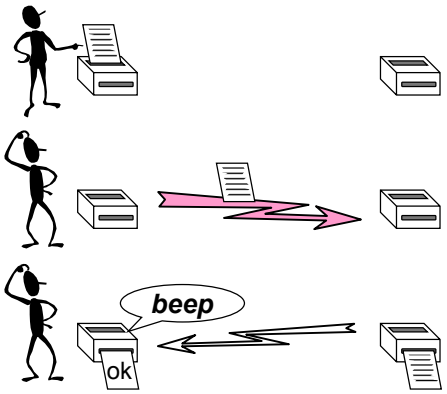
Point-to-Point Communication

- Simplest form of message passing.
- One process sends a message to another.
- Different types of point-to-point communication:
  - synchronous send
  - buffered = asynchronous send



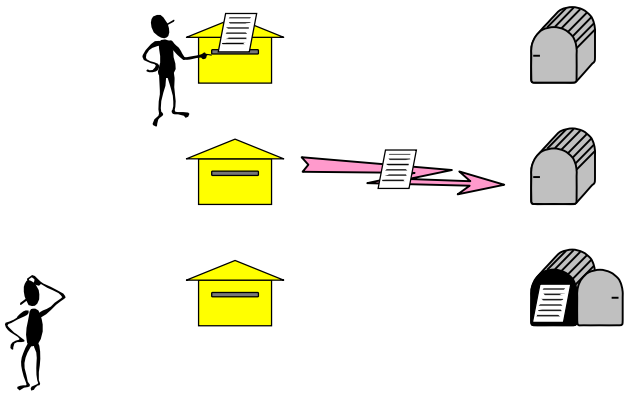
Synchronous Sends

- The sender gets an information that the message is received.
- Analogue to the *beep* or *okay-sheet* of a fax.



Buffered = Asynchronous Sends

- Only know when the message has left.

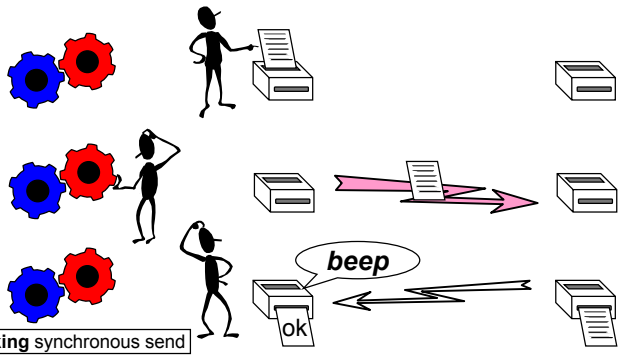


Blocking Operations

- Operations are local activities, e.g.,
  - sending (a message)
  - receiving (a message)
- Some operations may **block** until another process acts:
  - synchronous send operation **blocks until** receive is posted;
  - receive operation **blocks until** message is sent.
- Relates to the completion of an operation.
- Blocking subroutine returns only when the operation has completed.

Non-Blocking Operations

- Non-blocking operation: returns immediately and allow the sub-program to perform other work.
- At some later time the sub-program must ***test*** or ***wait*** for the completion of the non-blocking operation.



non-blocking synchronous send

Non-Blocking Operations (cont'd)

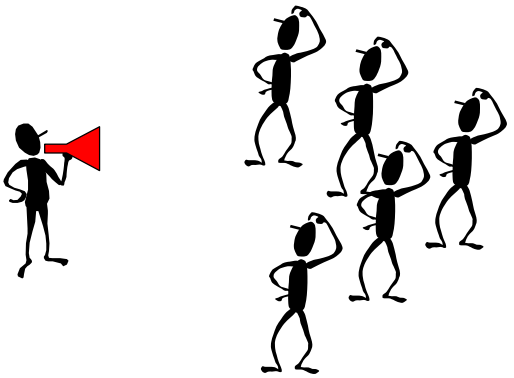
- All non-blocking operations must have matching wait (or test) operations. (Some system or application resources can be freed only when the non-blocking operation is completed.)
- A non-blocking operation immediately followed by a matching wait is equivalent to a blocking operation.
- Non-blocking operations are not the same as sequential subroutine calls:
  - the operation may continue while the application executes the next statements!

Collective Communications

- Collective communication routines are higher level routines.
- Several processes are involved at a time.
- May allow optimized internal implementations, e.g., tree based algorithms
- Can be built out of point-to-point communications.

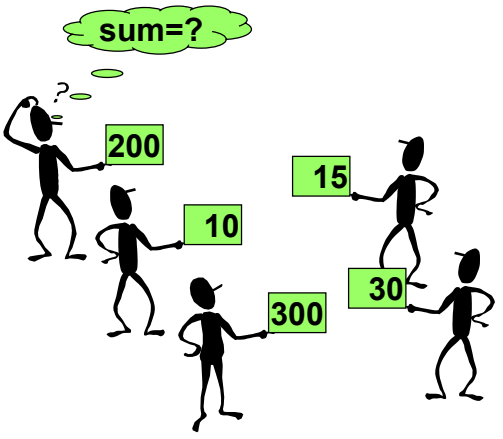
Broadcast

- A one-to-many communication.



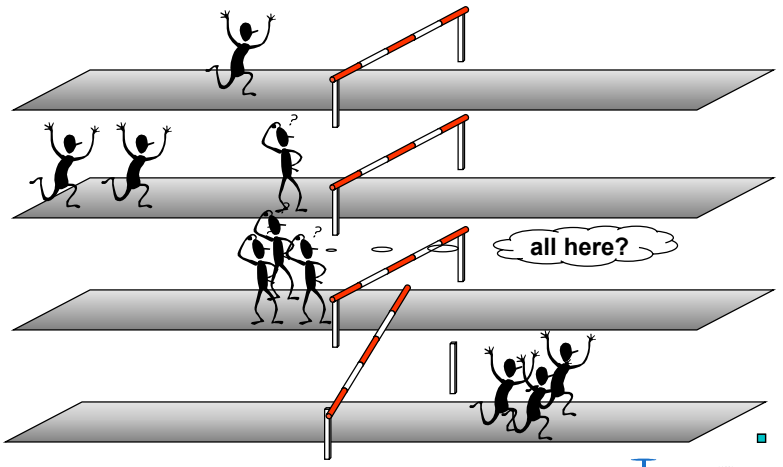
Reduction Operations

- Combine data from several processes to produce a single result.



Barriers

- Synchronize processes.

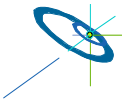


MPI Forum

- MPI-1 Forum
  - First message-passing interface standard.
  - Sixty people from forty different organizations.
  - Users and vendors represented, from US and Europe.
  - Two-year process of proposals, meetings and review.
  - *Message-Passing Interface* document produced.
  - MPI 1.0 — June,1994.
  - MPI 1.1 — June 12, 1995.

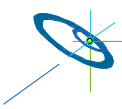
MPI-2 Forum

- MPI-2 Forum
  - Same procedure.
  - *MPI-2: Extensions to the Message-Passing Interface* document.
  - MPI 1.2 — mainly clarifications.
  - MPI 2.0 — extensions to MPI 1.2.



Goals and Scope of MPI

- MPI's prime goals
  - To provide a message-passing interface.
  - To provide source-code portability.
  - To allow efficient implementations.
- It also offers:
  - A great deal of functionality.
  - Support for heterogeneous parallel architectures.
- With MPI-2:
  - Important additional functionality.
  - No changes to MPI-1.



A Heat-Transfer-Eample: Communication Efficiency

Communication-time (number of PEs)	T3E (16)	Hitachi (16)	HP-V (8)	
MPI non-blocking	3.4	5.3	2	[sec]
MPI_SENDRECV	1.9	5.8	2	[sec]
MPI_ALLTOALLV	0.8 *)	15.4	2	[sec]
Computation-Time	0.65	1.9	2	[sec]

MPI targets portable and efficient message-passing programming
but
efficiency of MPI application-programming is **not portable**!

*) **up 128 PEs: ALLTOALLV is better than SENDRECV**

(measured April 29, 1999, with heat-mpi1-big.f and stride 179,
CRAY T3E: sn6715 hwwt3e.hww.de 2.0.4.48 unicosmk CRAY T3E mpt.1.3.0.0.6,
Hitachi: HI-UX/MPP hitachi.rus.uni-stuttgart.de 02-03 0 SR2201,
HP: HP-UX hp-v.hww.de B.11.00 A 9000/800 75859,
Time-step iteration on a 79x79 grid)

Informations about MPI

- **MPI: A Message-Passing Interface Standard** (1.1, June 12, 1995)
- **MPI-2: Extensions to the Message-Passing Interface** (July18,1997)
- Marc Snir and William Gropp et al.:
  MPI: The Complete Reference. (2-volume set).The MIT Press, 1998.
  (excellent catching up of the standard MPI-1.2 and MPI-2 in a readable form)
- William Gropp, Ewing Lusk and Rajeev Thakur:
  Using MPI: Portable Parallel Programming With the Message-Passing Interface. MIT Press, Nov. 1999. And
  Using MPI-2: Advanced Features of the Message-Passing Interface. MIT Press, Aug. 1999.
  (or both in one volume, 725 pages, ISBN 026257134X)
- Peter S. Pacheco: **Parallel Programming with MPI**.
  Morgen Kaufmann Publishers, 1997.
  (very good introduction, can be used as accompanying text for MPI lectures)
- http://www.hlrs.de/mpi/

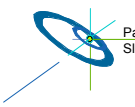
Outline

- Parallel High Performance Computing Platforms in Germany
- Parallel Hardware Architectures
- Parallel Programming Models – an Overview
- Message Passing Interface (MPI) – more Details
- **Parallel Programming Models – a Comparison**
- Online Parallel Programming Workshop

H L R S

Which parallel programming model is the best for my application?

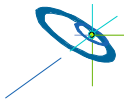
- no absolute answer
- only hints on the next slides



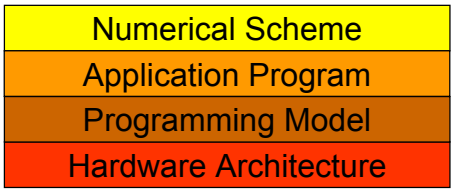
H L R S

Why do I want to parallelize my code?

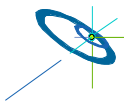
- time
  - parallelization of the work on many CPUs
  - to speedup the execution
- memory space
  - the application does not fit to the available memory of one CPU
  - parallelization on distributed memory



Additional questions



- Available parallelization strategies in my numerical scheme?
  - loop parallelism
  - domain decomposition
- Which hardware architecture?
  - today
  - in the future
- How many working hours do I want to spend for parallelizing the code?



Execution time

- My application runs to slow
  - (Floating point) operations / second on each CPU?
    - **vectorization** (memory→CPU→memory)
      - expensive hardware / cheap programming effort
    - **cache oriented optimization**
      - cheap hardware / expensive programming effort
    - **such optimization is impossible for me**
  - Parallelization
    - **shared memory** → **OpenMP**, HPF, MPI
      - expensive & limited hardware / cheap programming effort
    - **distributed memory** → **MPI**, **HPF**, shmem, MLP, CoArrayFort., ...
      - cheap hardware / expensive programming effort

→ **Today / in the future**

Speedup, Efficiency, and Scaleup

- Definition:
  - T(p,N) = **time** to solve **problem of size N** on **p processors**
- Speedup:
  - S(p,N) = T(1,N) / T(p,N)
  - compute **same problem** with more processors in **shorter time**
- Efficiency:
  - E(p,N) = S(p,N) / p
- Scaleup:
  - Sc(p,N) = N / n with T(1,n) = T(p,N)
  - compute **larger problem** with more processors in **same time**
- Problems:
  - Absolute MFLOPS rate **/** hardware peak performance?
  - S(p,N) close to **p** or far less? —> see Amdahls Law on next slide
  - Or super-scalar speedup: S(p,N)>p, e.g., due to cache usage

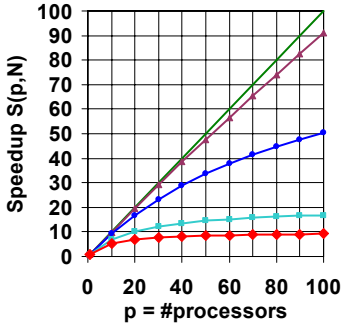
Amdahls Law

$T(p,N) = f \cdot T(1,N) + (1-f) \cdot T(1,N) / p$

f ... sequential part of code that can not be done in parallel

$S(p,N) = T(1,N) / T(p,N) = 1 / (f + (1-f) / p)$

For p —> infinity, speedup is limited by $S(p,N) < 1 / f$



- S(p,N) = p (ideal speedup)
- f=0.1% => S(p,N) < 1000
- f= 1% => S(p,N) < 100
- f= 5% => S(p,N) < 20
- f= 10% => S(p,N) < 10

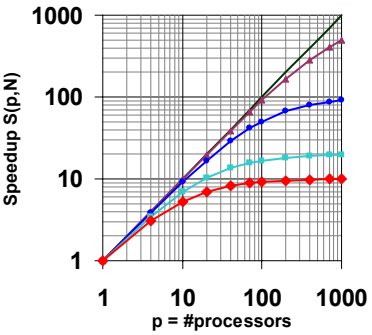
Amdahls Law (double-logarithmic)

$T(p,N) = f \cdot T(1,N) + (1-f) \cdot T(1,N) / p$

f ... sequential part of code that can not be done in parallel

$S(p,N) = T(1,N) / T(p,N) = 1 / (f + (1-f) / p)$

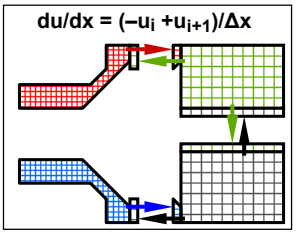
For p —> infinity, speedup is limited by $S(p,N) < 1 / f$



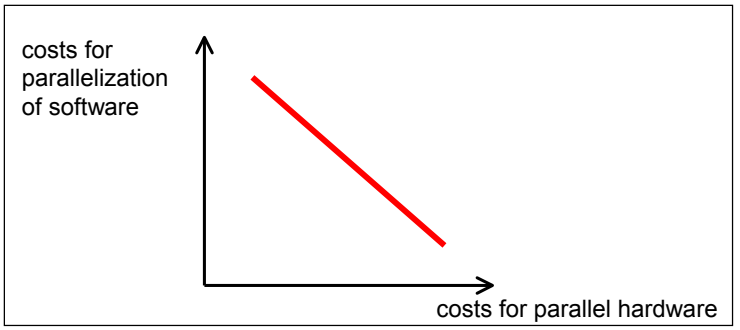
- S(p,N) = p (ideal speedup)
- f=0.1% => S(p,N) < 1000
- f= 1% => S(p,N) < 100
- f= 5% => S(p,N) < 20
- f= 10% => S(p,N) < 10

Parallelization problems

- Two major resources of computation:
  - processor
  - memory
- Parallelization means
  - distributing work to processors
    - —> load balancing necessary
    - —> synchronization overhead should be minimized
      - —> to achieve optimal speedup
  - distributing data (if memory is distributed)
    - —> implies communication to bring data to processor
      - —> communication is overhead
        - —> is reducing the speedup



Parallelization costs



- low costs for parallel hardware → high parallelization costs
- high costs for parallel hardware → low parallelization costs

Advantages and Challenges

	OpenMP	HPF	MPI
Maturity of programming model	++	+	++
Maturity of standardization	+	+	++
Migration of serial programs	++	0	– –
Ease of programming (new progr.)	++	+	–
Correctness of parallelization	–	++	– –
Portability to any hardware architecture	–	++	++
Availability of implementations of the stand.	+	+	++
Availability of parallel libraries	0	0	0
Scalability to hundreds/thousands of processors	– –	0	++
Efficiency	–	0	++
Flexibility – dynamic program structures	–	–	++
– irregular grids, triangles, tetra-hedrons, load balancing, redistribut.	–	–	++

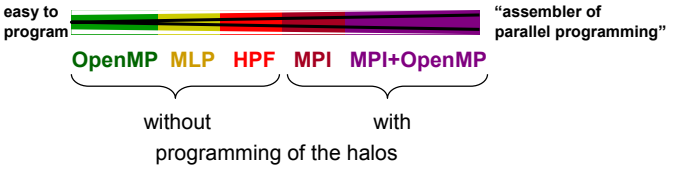
Implications on Hybrid Systems

- Hybrid system = cluster of SMPs, e.g., with vector CPUs
  - MPP (massively parallel processing) model (pure MPI):
    - one MPI process on each CPU
  - hybrid model: MPI+OpenMP or MPI+automatic parallelization
    - each MPI process is multi-threaded with OpenMP/…
    - lousy communication speed, if MPI is done only by master thread (all other threads are sleeping)
    - highest costs for parallelizing the software
    - Amdahl's law with reduced number of CPUs on several levels

  - HPF may also fit,
    - e.g., on the Earth Simulator in Japan

Which Model is the Best for Me?

- Depends on
  - your application
  - your platform
  - which efficiency do you need on your platform
  - how much time do you want to spent on parallelization



Acknowledgements

- Thanks to Alfred Geiger and Michael Resch (HLRS)
  - pictures and slides from their *Parallel Programming* lectures
- Thanks to EPCC Training and Education Centre, Edinburgh Parallel Computing Centre, University of Edinburgh
  - MPI-overview slides from their MPI course

Outline

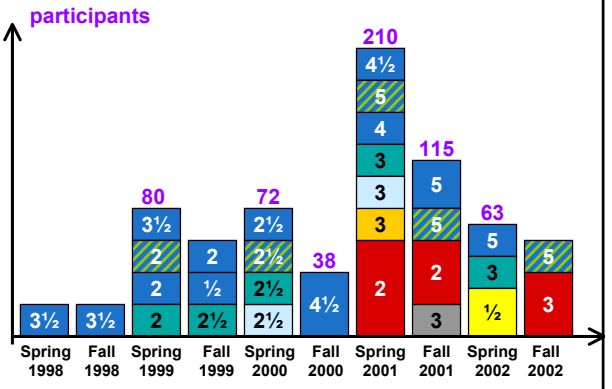
- Parallel High Performance Computing Platforms in Germany
- Parallel Hardware Architectures
- Parallel Programming Models – an Overview
- Message Passing Interface (MPI) – more Details
- Parallel Programming Models – a Comparison
- **Online Parallel Programming Workshop**
  - http://www.hlrs.de/organization/par/par_prog_ws/
  - or on CD-ROM
  - see also class-room courses

Parallel Programming Workshops — Courses

Locations:

H L R S
ZHR
NIC
LZ
UH and RRZN
IAG

= in English
5 = 5 day course with 18-25 participants



Outline – [Research Activities of Parallel Computing Dptm.]

- Parallel High Performance Computing Platforms in Germany
- Parallel Hardware Architectures
- Parallel Programming Models – an Overview
- Message Passing Interface (MPI) – more Details
- Parallel Programming Models – a Comparison
- Online Parallel Programming Workshop
  - http://www.hlrs.de/organization/par/par_prog_ws/
  - or on CD-ROM
  - see also class-room courses

Research Activities of the Parallel Computing Department at HLRS

Enhanced **D**esign **E**nvironment for Industrial **CAST**ing Processes on Parallel Computing Platforms (**DECAST**)

- Simulation of a complete casting process using two Finite Element programs:
  - a CFD code for the filling up the mould and
  - a CTM (Coupled Thermal-Mechanical) code for the solidification and cooling of the part
- Target Parallel Platforms:
  - clusters of PCs and Workstations

Panagiotis Adamidis

URANUS Navier-Stokes Code for High-Temperature Nonequilibrium flows

Upwind **R**elaxation **A**lgorithm for **N**onequilibrium Flows of the **U**niversity of **S**tuttgart

Features:

- CVCV multiple temperature gasphase model
- Chapman-Cowling transport coefficients models
- Gaskinetic gas-surface model with different surface reaction models
- PARADE/HERTA gas-radiation coupling
- Sequential 3-D solver
- Adaptive parallel multiblock-solver

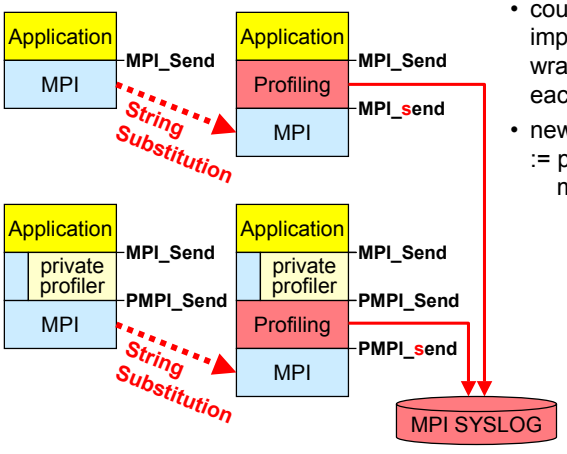
Thomas P. Bönisch

Marc Lange - Parapyr

Weitere Forschungsbereiche in der Abteilung

- Automatic Counter Profiling of MPI Applications on Cray T3E
- Effective Parallel File I/O Bandwidth Benchmark – b_eff_io
- Effective Communication Bandwidth Benchmark – b_eff
- Optimierung Kollektiver MPI-Routinen – MPI_(All)Reduce Butterfly-Algor.
- Parallele Programmiermodelle für hybride Plattformen

Automatic MPI Profiling: Software design for libmpi.a



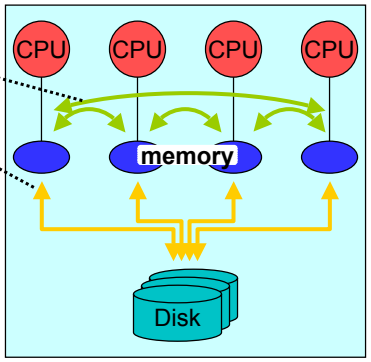
- counter profiling is implemented as wrapper routines to each MPI routine
- new libmpi.a := profiling + modified libmpi.a

Effective Communication & I/O Bandwidth Benchmarks

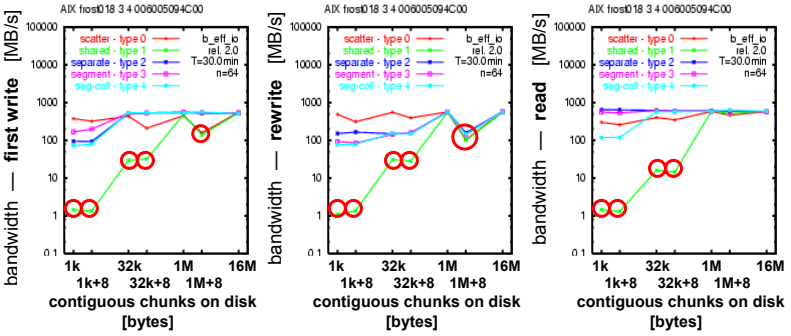
Goals

- Parallel **Communication** Benchmark
- Parallel **File-I/O** Benchmark

  each process is involved!
- Detailed insight
  - bandwidth experiments of several
    - I/O or communication patterns
    - chunk or message sizes
- One characteristic value
  - based on experiments above
  - averaging
- Appropriate execution time for rapid benchmarking



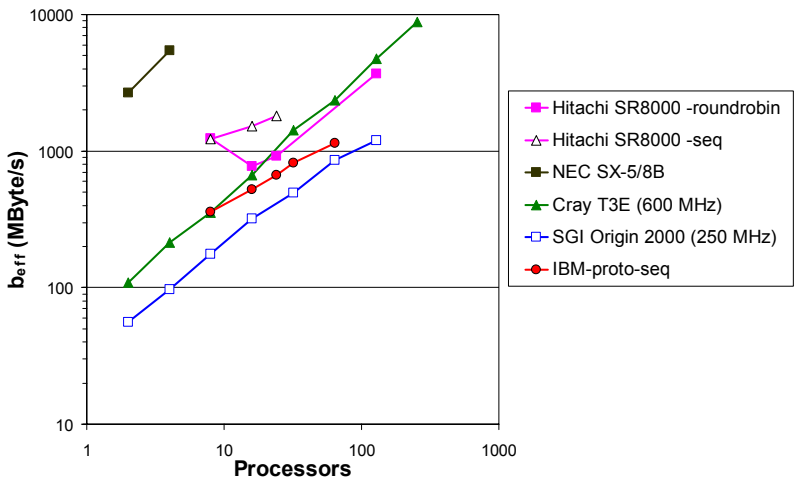
b_eff_io: Results on ASCI White testbed with optimal hints



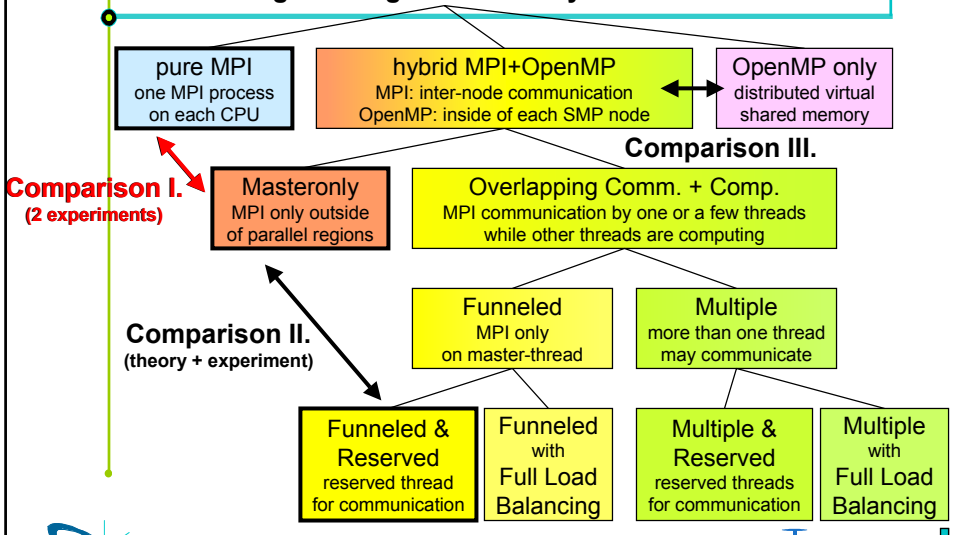
- **each pattern may be optimized with hints** *)
  - **here: IBM_large_block_io=false & IBM_io_buffer_size=2MB (instead of 8MB) for "*scatter*" and "*shared*" pattern type with chunk sizes <= 32k+8**
  - **86 from 105 patterns quite fast ! 19 candidates for further optimization**

*) R. Rabenseifner, A. E. Koniges, J.-P. Prost, and R. Hedges: The Parallel Effective I/O Bandwidth Benchmark: b_eff_io. To be published in Special Issue of Calculateurs Parallèles Journal on Parallel I/O for Cluster Computing.

B_eff Scaling: current systems

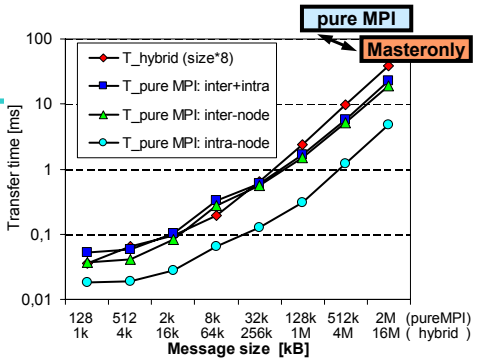


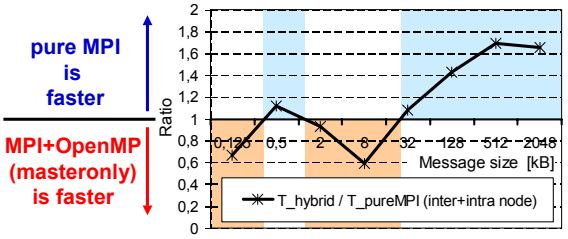
Parallel Programming Models on Hybrid Platforms



Results of the experiment

- pure MPI is better for message size > 32 kB
- long messages: $T_{hybrid} / T_{pureMPI} > 1.6$
- OpenMP master thread cannot saturate the inter-node network bandwidth

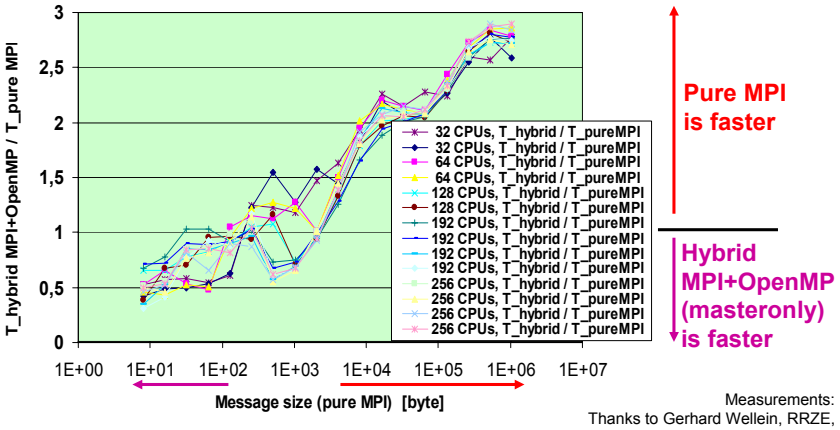




H L R S

Ratio T_hybrid MPI+OpenMP / T_pure MPI

IBM RS/6000 SP, 208 SMP nodes, each SMP node with 16 POWER3+ CPUs, at NERSC



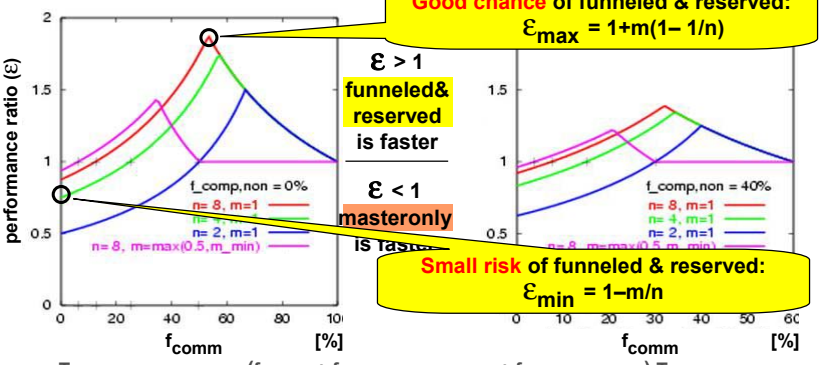
Measurements:
Thanks to Gerhard Wellein, RRZE, and Horst Simon, NERSC.

NERSC H L R S

Masteronly
funneled & reserved

Performance ratio (theory)

- $\varepsilon = \left(\frac{T_{\text{hybrid, funneled\&reserved}}}{T_{\text{hybrid, masteronly}}} \right)^{-1}$



Good chance of funneled & reserved: $\varepsilon_{max} = 1+m(1-1/n)$

$\varepsilon > 1$ funneled& reserved is faster

$\varepsilon < 1$ masteronly is faster

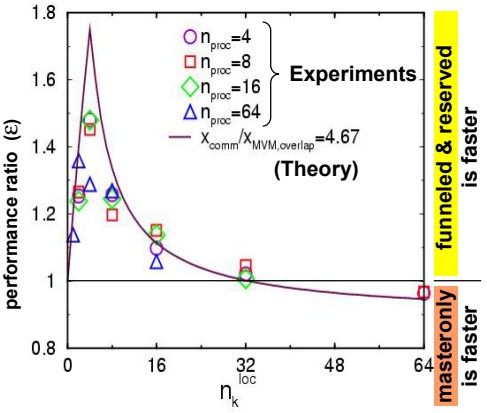
Small risk of funneled & reserved: $\varepsilon_{min} = 1-m/n$

$T_{\text{hybrid, masteronly}} = (f_{\text{comm}} + f_{\text{comp, non-overlap}} + f_{\text{comp, overlap}})\, T_{\text{hybrid, masteronly}}$

n = # threads per SMP node, m = # reserved threads for MPI communication

Masteronly
funneled & reserved

Experiment: Matrix-vector-multiply (MVM)



- Jacobi-Davidson-Solver
- Hitachi SR8000
- 8 CPUs / SMP node
- JDS (Jagged Diagonal Storage)
- vectorizing
- n_{proc} = # SMP nodes
- $D_{Mat} = 512*512*(n_k^{loc}*n_{proc})$
- Varying n_k^{loc} ⇒ Varying $1/f_{comm}$
- $\frac{f_{comp,non\text{-}overlap}}{f_{comp,overlap}} = \frac{1}{6}$

Source: R. Rabenseifner, G. Wellein:
Communication and Optimization Aspects of Parallel Programming Models.
EWOMP 2002, Rome, Italy, Sep. 18–20, 2002

Hybrid Parallel Programming: Conclusions

- ***Pure MPI*** versus ***hybrid masteronly*** model:
  - Topology problem may be hard, e.g., with unstructured grids
  - Communication is bigger with pure MPI, but may be nevertheless faster
  - On the other hand, typically communication is only some percent → relax
- Efficient hybrid programming:
  - one should overlap communication and computation → hard to do!
  - using simple ***hybrid funneled&reserved*** model,
    you maybe up to 50% faster (compared to *masteronly* model)
- → Coming from ***pure MPI***,
  one may try to implement ***hybrid funneled&reserved*** *model*
- If you want to use ***pure OpenMP*** (based on virtual shared memory)
  - try to use still the full bandwidth of the inter-node network
    (keep pressure on your compiler/DSM writer)
  - be sure that you do not lose any computational optimization
    - **e.g., best Fortran compiler & optimization directives should be usable**

Summary

- Hardware architectures
  - hybrid (hierarchical) systems are the future
    - **cluster of dual-board PC**
    - **…**
    - **clusters of PVP-SMP systems**
- Parallel Programming models
  - MPI and OpenMP are dominating
  - HPF still alive (→ JaHPF on Earth Simulator)
- Which model is the best
  - depends on your needs & hardware, today and in future
  - OpenMP is limited to shared memory platforms,
    but may be extended with a data distribution model (like HPF)
  - MPI is the *assembler of parallel programming*
  - invest your working effort into single-CPU-optimization,
    rather than into hybrid programming