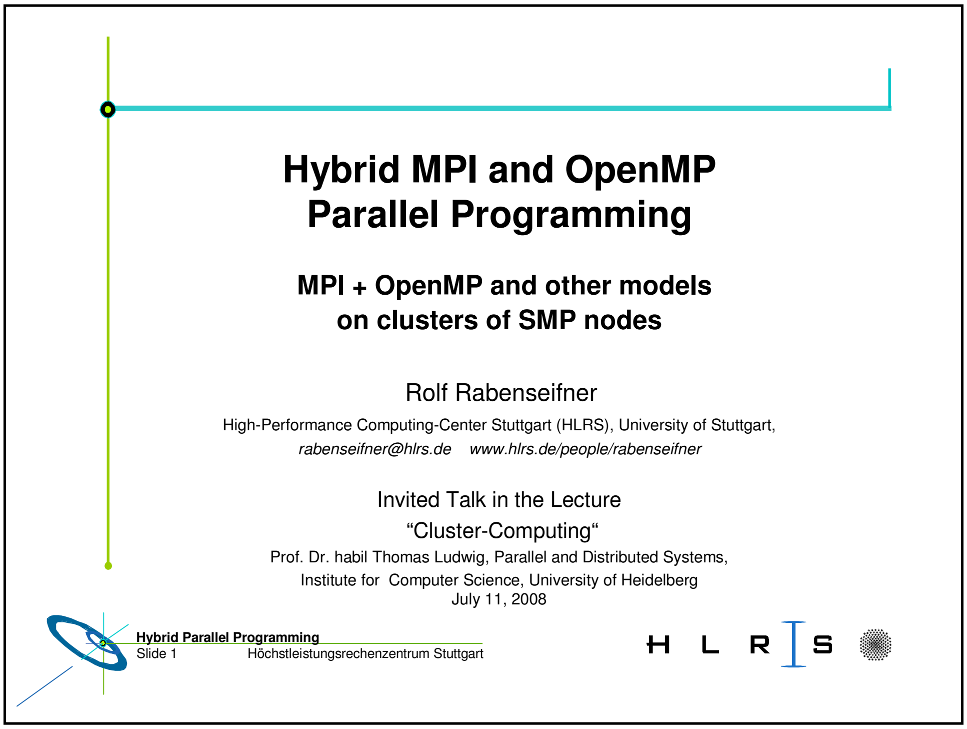


Hybrid MPI and OpenMP Parallel Programming

MPI + OpenMP and other models on clusters of SMP nodes

Rolf Rabenseifner

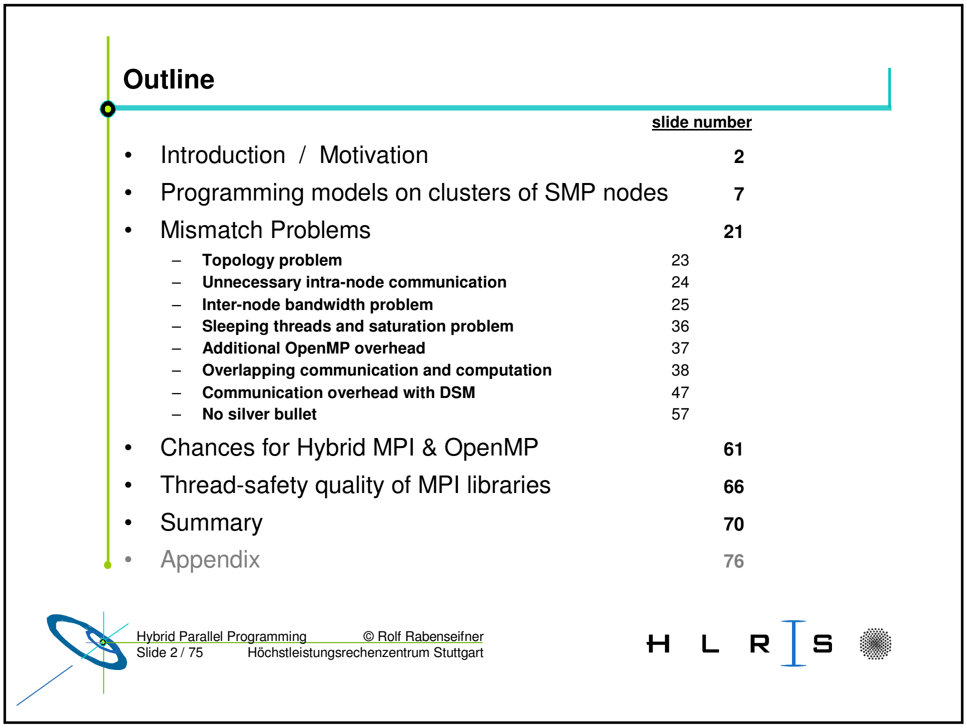
High-Performance Computing-Center Stuttgart (HLRS), University of Stuttgart,

rabenseifner@hlrs.de *www.hlrs.de/people/rabenseifner*

Invited Talk in the Lecture

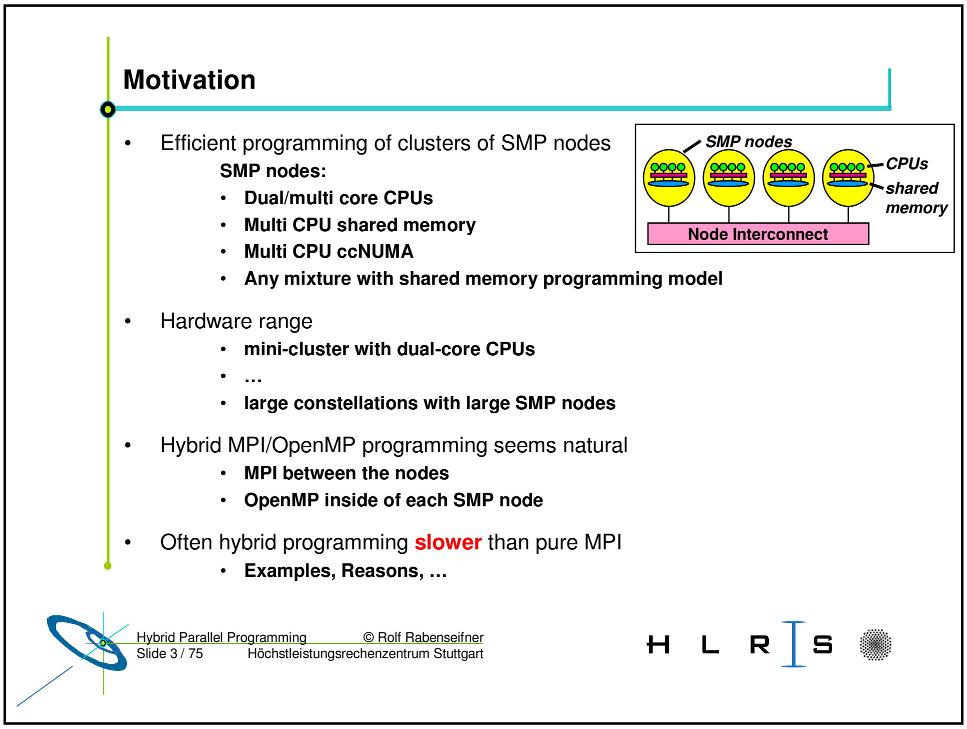
“Cluster-Computing“

Prof. Dr. habil Thomas Ludwig, Parallel and Distributed Systems,
Institute for Computer Science, University of Heidelberg
July 11, 2008



Outline

	slide number
• Introduction / Motivation	**2**
• Programming models on clusters of SMP nodes	**7**
• Mismatch Problems	**21**
– **Topology problem**	23
– **Unnecessary intra-node communication**	24
– **Inter-node bandwidth problem**	25
– **Sleeping threads and saturation problem**	36
– **Additional OpenMP overhead**	37
– **Overlapping communication and computation**	38
– **Communication overhead with DSM**	47
– **No silver bullet**	57
• Chances for Hybrid MPI & OpenMP	**61**
• Thread-safety quality of MPI libraries	**66**
• Summary	**70**
• Appendix	**76**

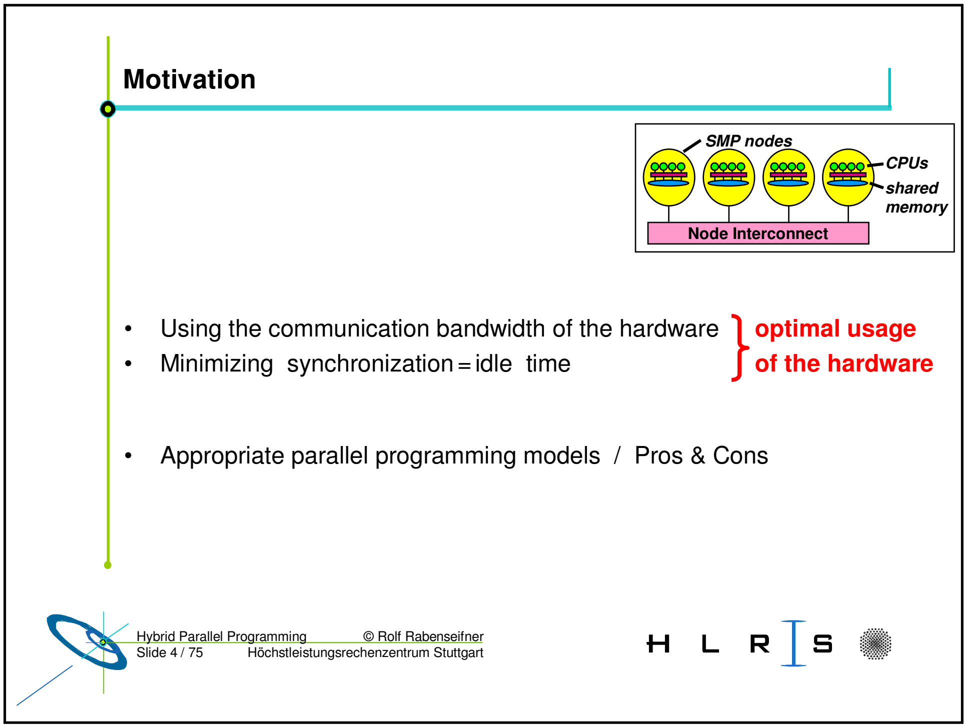


Motivation

- Efficient programming of clusters of SMP nodes
  SMP nodes:
  - **Dual/multi core CPUs**
  - **Multi CPU shared memory**
  - **Multi CPU ccNUMA**
  - **Any mixture with shared memory programming model**

SMP nodes
CPUs
shared memory
Node Interconnect

- Hardware range
  - **mini-cluster with dual-core CPUs**
  - **…**
  - **large constellations with large SMP nodes**
- Hybrid MPI/OpenMP programming seems natural
  - **MPI between the nodes**
  - **OpenMP inside of each SMP node**
- Often hybrid programming **slower** than pure MPI
  - **Examples, Reasons, …**



Motivation

SMP nodes
CPUs
shared memory
Node Interconnect

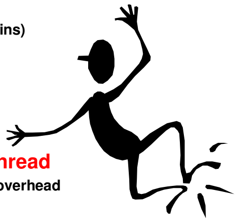
- Using the communication bandwidth of the hardware
- Minimizing synchronization = idle time

} **optimal usage of the hardware**

- Appropriate parallel programming models / Pros & Cons

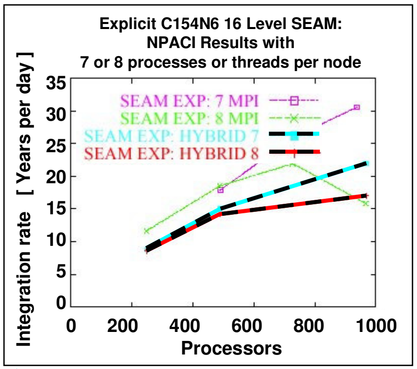
But results may surprise!

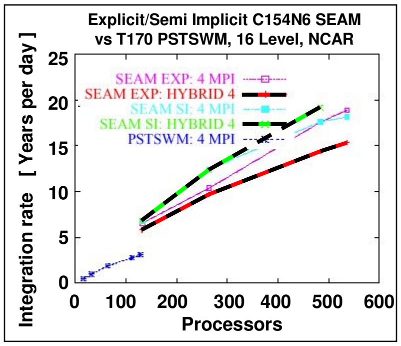
- Example code - HYDRA
- Domain-decomposed hydrodynamics
  - (almost) independent mesh domains with ghost cells on boundaries
  - ghost cells communicate boundary information ~40-50 times per cycle
- Parallelism model: single level
  - MPI divides domains among compute nodes
  - OpenMP further subdivides domains among processors
  - domain size set for cache efficiency
    - **minimizes memory usage, maximizes efficiency**
    - **scales to very large problem sizes (>10^7 zones, >10^3 domains)**
- Results:
  - **MPI** (256 proc.) **~20% faster than MPI / OpenMP** (64 nodes x 4 proc./node)
  - domain-domain communication not threaded, i.e., **MPI communication is done only by main thread**
    - **accounts for ~10% speed difference, remainder in thread overhead**



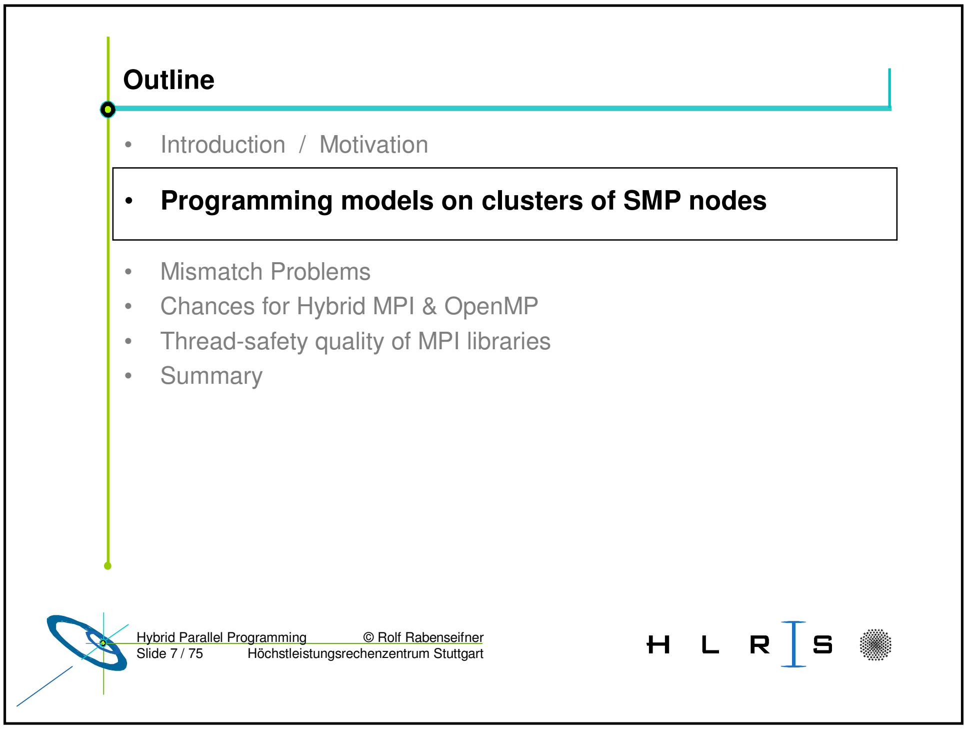
Example from SC

- Pure MPI versus Hybrid MPI+OpenMP (Masteronly)
- What‘s better?
  → it depends on?



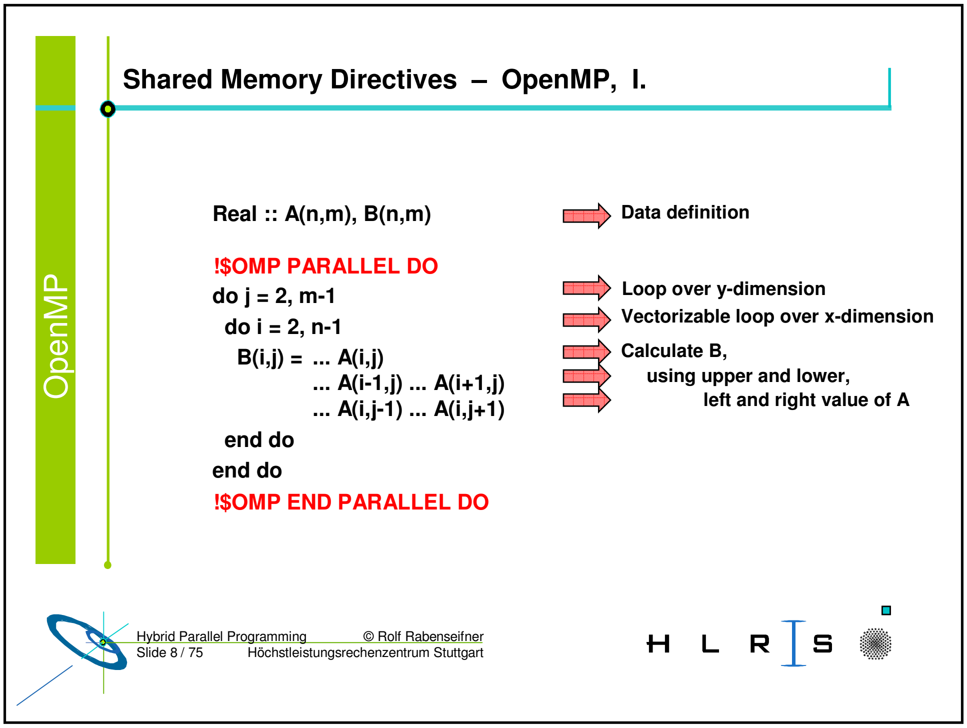


Figures: Richard D. Loft, Stephen J. Thomas, John M. Dennis:
Terascale Spectral Element Dynamical Core for Atmospheric General Circulation Models.
Proceedings of SC2001, Denver, USA, Nov. 2001.
http://www.sc2001.org/papers/pap.pap189.pdf
Fig. 9 and 10.



Outline

- Introduction / Motivation
- **Programming models on clusters of SMP nodes**
- Mismatch Problems
- Chances for Hybrid MPI & OpenMP
- Thread-safety quality of MPI libraries
- Summary



OpenMP

Shared Memory Directives – OpenMP, I.

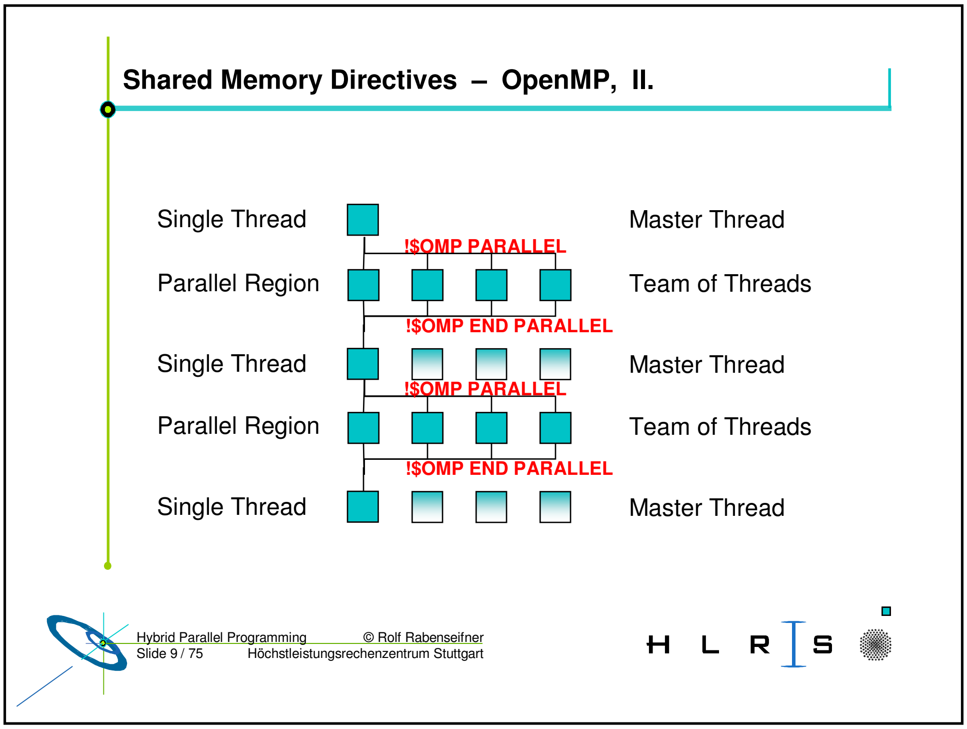
```
Real :: A(n,m), B(n,m)

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

Single Thread — Master Thread

!$OMP PARALLEL

Parallel Region — Team of Threads

!$OMP END PARALLEL

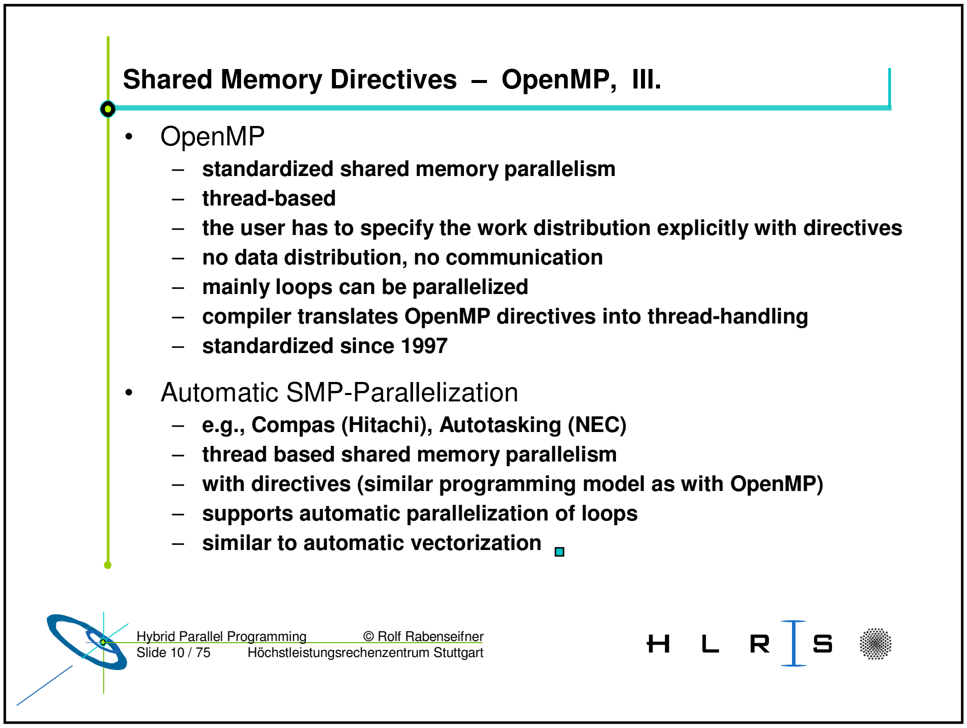
Single Thread — Master Thread

!$OMP PARALLEL

Parallel Region — Team of Threads

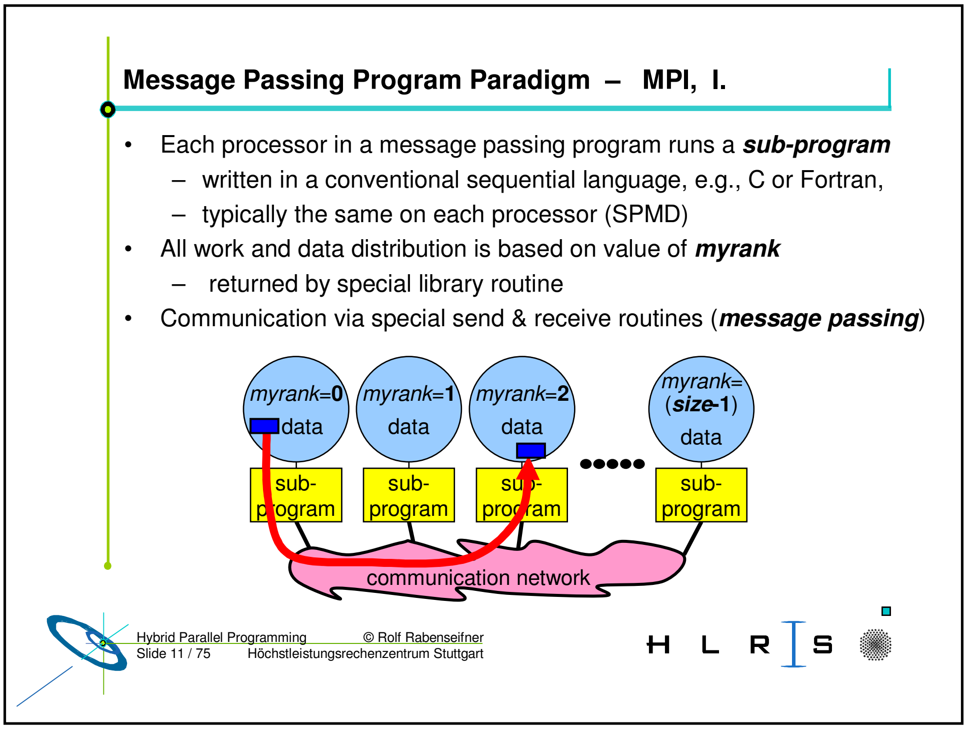
!$OMP END PARALLEL

Single Thread — Master Thread



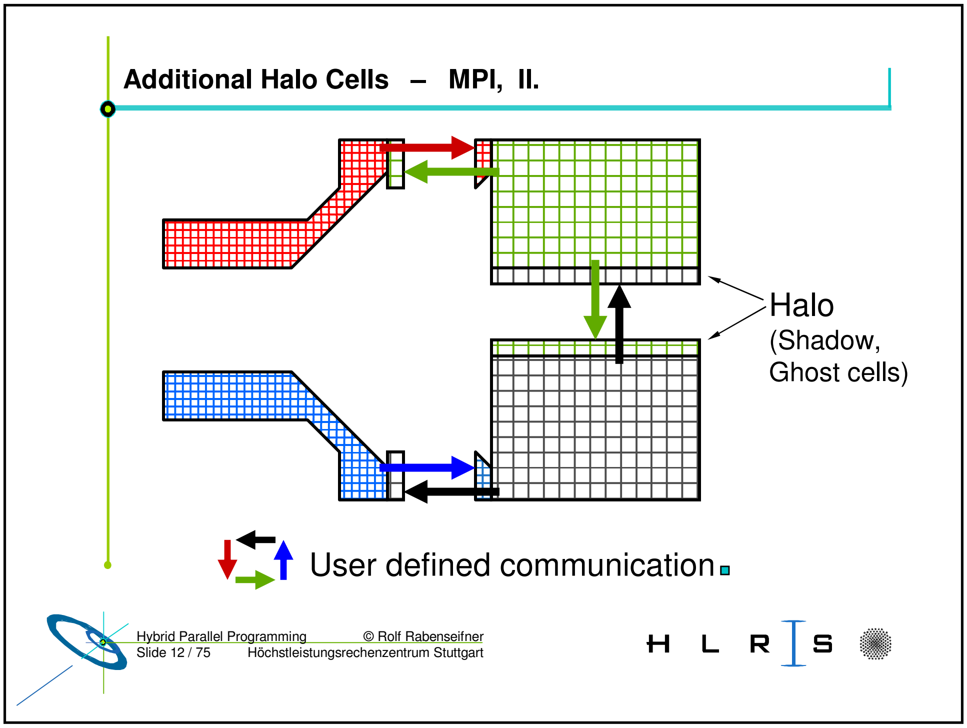
Shared Memory Directives – OpenMP, III.

- 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**



Message Passing Program Paradigm – MPI, I.

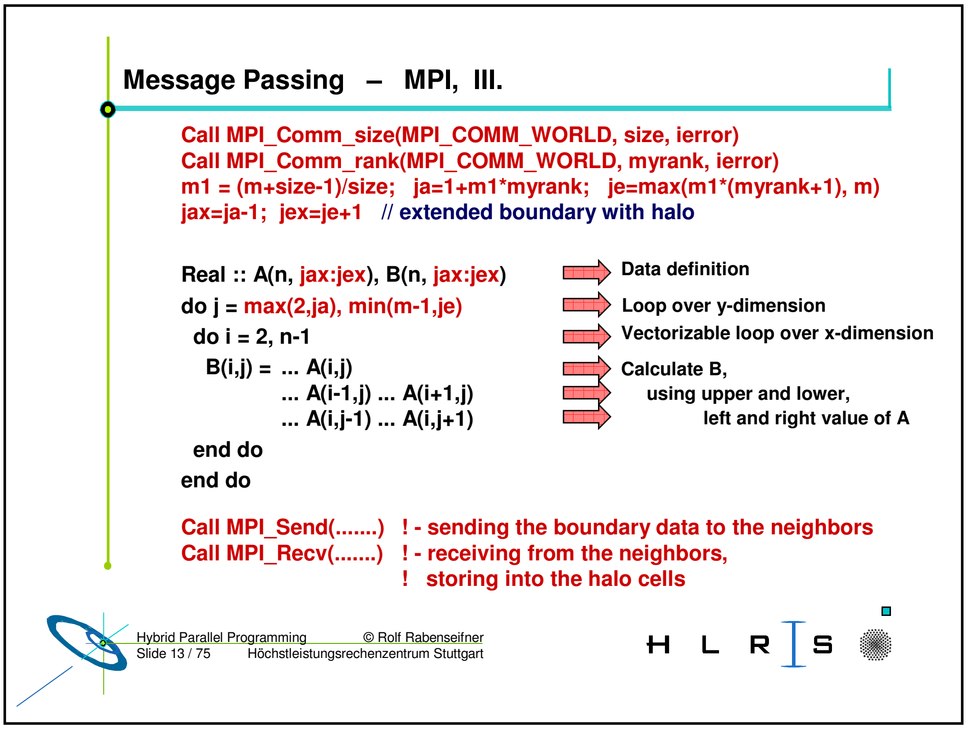
- Each processor in a message passing program runs a ***sub-program***
  - written in a conventional sequential language, e.g., C or Fortran,
  - typically the same on each processor (SPMD)
- All work and data distribution is based on value of ***myrank***
  - returned by special library routine
- Communication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



Additional Halo Cells – MPI, II.

Halo
(Shadow,
Ghost cells)

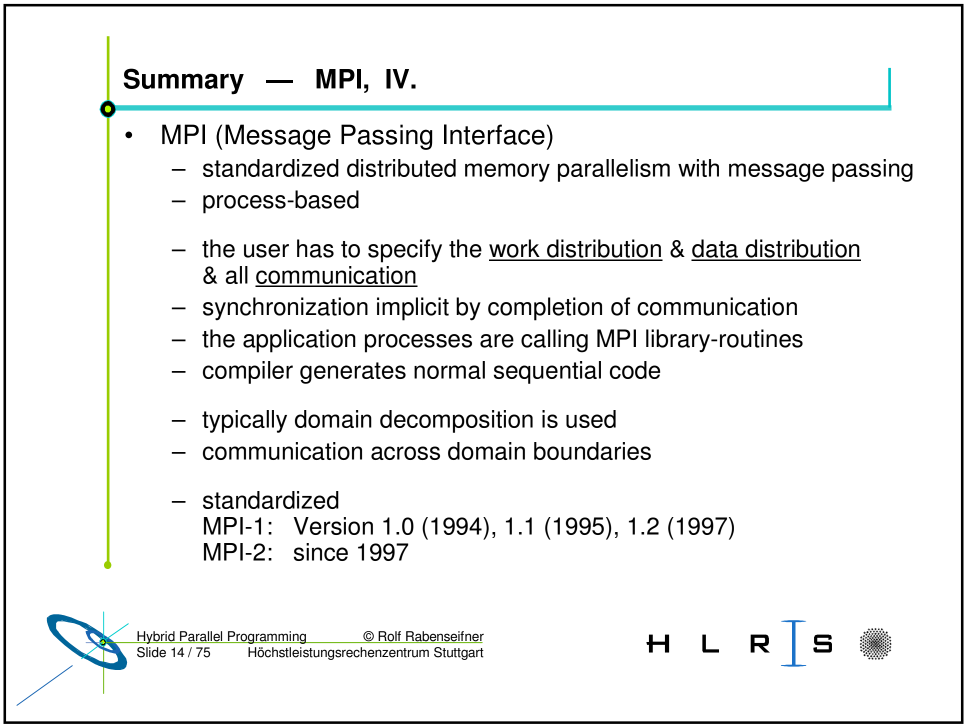
User defined communication



Message Passing – MPI, III.

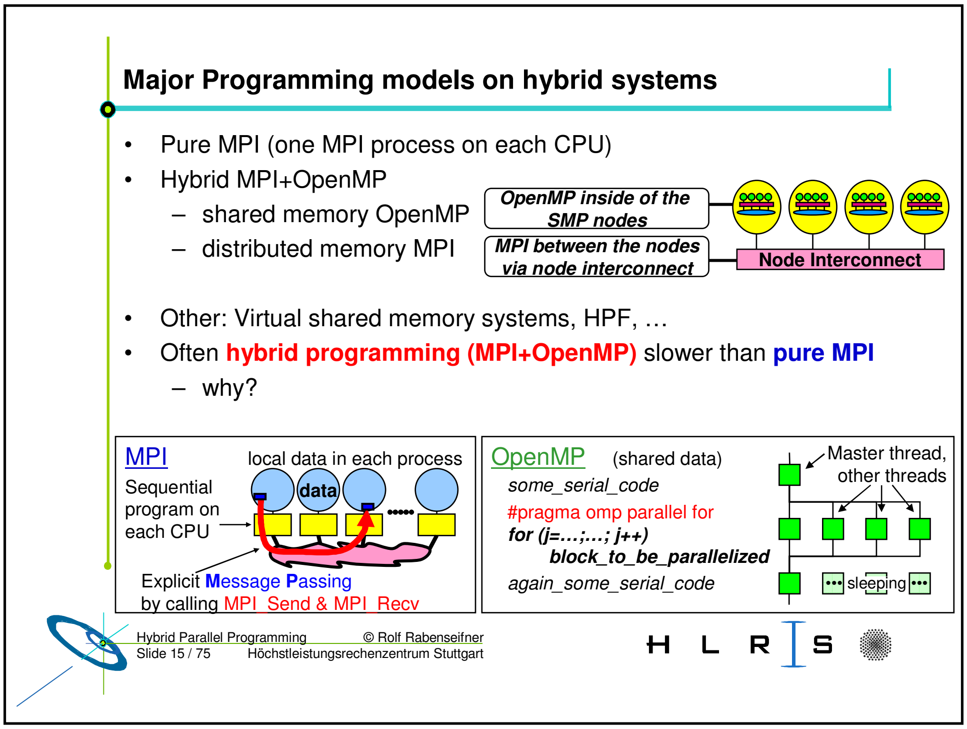
```
Call MPI_Comm_size(MPI_COMM_WORLD, size, ierror)
Call MPI_Comm_rank(MPI_COMM_WORLD, myrank, ierror)
m1 = (m+size-1)/size;   ja=1+m1*myrank;   je=max(m1*(myrank+1), m)
jax=ja-1;  jex=je+1   // extended boundary with halo

Real :: A(n, jax:jex), B(n, jax:jex)
do j = max(2,ja), min(m-1,je)
  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

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

⇨ Data definition
⇨ Loop over y-dimension
⇨ Vectorizable loop over x-dimension
⇨ Calculate B,
⇨ using upper and lower,
⇨ left and right value of A



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



Major Programming models on hybrid systems

- Pure MPI (one MPI process on each CPU)
- Hybrid MPI+OpenMP
  - shared memory OpenMP
  - distributed memory MPI
- Other: Virtual shared memory systems, HPF, …
- Often **hybrid programming (MPI+OpenMP)** slower than **pure MPI**
  - why?

OpenMP inside of the SMP nodes
MPI between the nodes via node interconnect
Node Interconnect

MPI

local data in each process

Sequential program on each CPU

Explicit **M**essage **P**assing by calling MPI_Send & MPI_Recv

OpenMP (shared data)

some_serial_code

#pragma omp parallel for

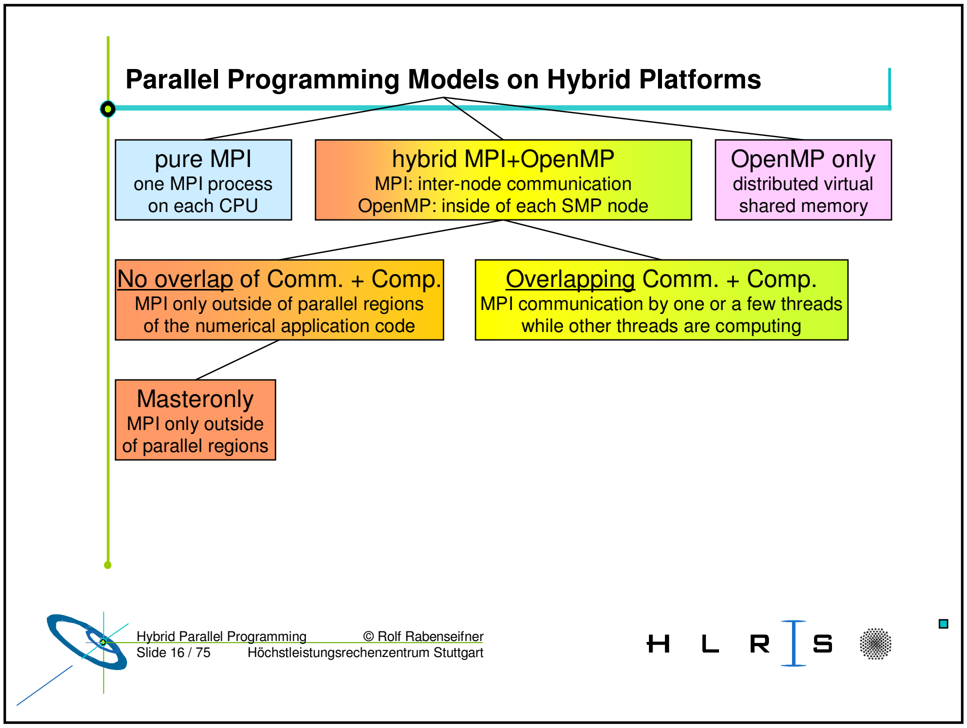
for (j=…;…; j++)

block_to_be_parallelized

again_some_serial_code

Hybrid Parallel Programming © Rolf Rabenseifner
Slide 15 / 75 Höchstleistungsrechenzentrum Stuttgart

H L R S



Parallel Programming Models on Hybrid Platforms

pure MPI
one MPI process on each CPU

hybrid MPI+OpenMP
MPI: inter-node communication
OpenMP: inside of each SMP node

OpenMP only
distributed virtual shared memory

No overlap of Comm. + Comp.
MPI only outside of parallel regions of the numerical application code

Overlapping Comm. + Comp.
MPI communication by one or a few threads while other threads are computing

Masteronly
MPI only outside of parallel regions

Hybrid Parallel Programming © Rolf Rabenseifner
Slide 16 / 75 Höchstleistungsrechenzentrum Stuttgart

H L R S

Pure MPI

pure MPI
one MPI process
on each CPU

Advantages

- No modifications on existing MPI codes
- MPI library need not to support multiple threads

Major problems

- Does MPI library uses internally different protocols?
  - **Shared memory inside of the SMP nodes**
  - **Network communication between the nodes**
- Does application topology fit on hardware topology?
- Unnecessary MPI-communication inside of SMP nodes!

Hybrid Masteronly

Masteronly
MPI only outside
of parallel regions

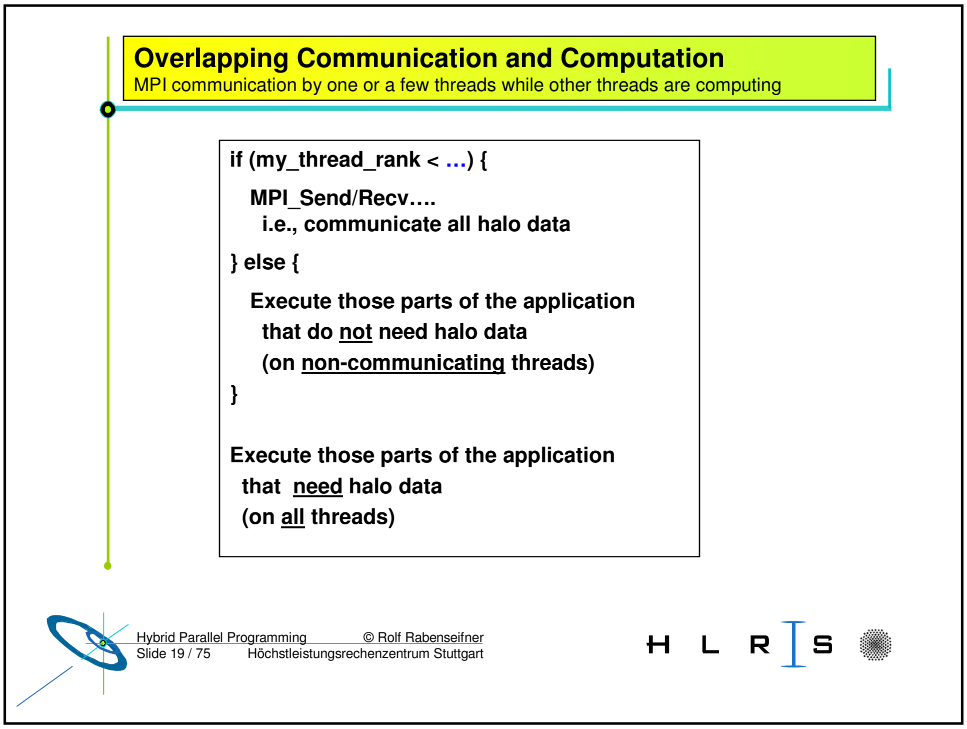
Advantages

- No message passing inside of the SMP nodes
- No topology problem

```
for (iteration ....)
{
 #pragma omp parallel
   numerical code
 /*end omp parallel */

 /* on master thread only */
   MPI_Send (original data
     to halo areas
     in other SMP nodes)
   MPI_Recv (halo data
     from the neighbors)
} /*end for loop
```

Major Problems

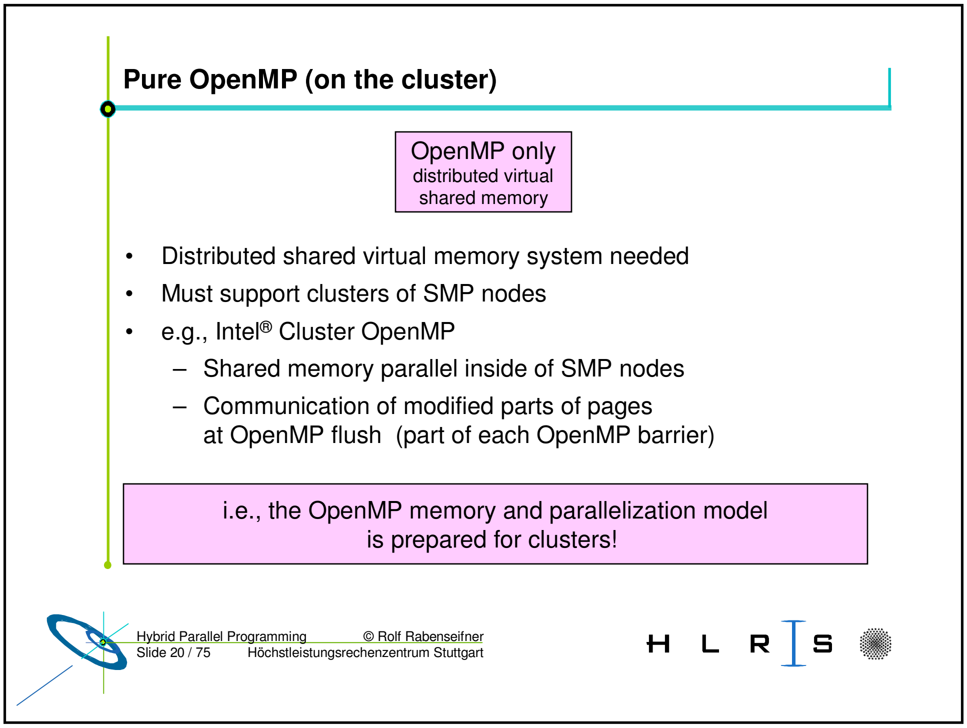
- MPI-lib must support at least MPI_THREAD_FUNNELED
- Which inter-node bandwidth?
- All other threads are sleeping while master thread communicates!



Overlapping Communication and Computation

MPI communication by one or a few threads while other threads are computing

```
if (my_thread_rank < ...) {
   MPI_Send/Recv....
     i.e., communicate all halo data
} else {
   Execute those parts of the application
     that do not need halo data
     (on non-communicating threads)
}

Execute those parts of the application
  that  need halo data
  (on all threads)
```

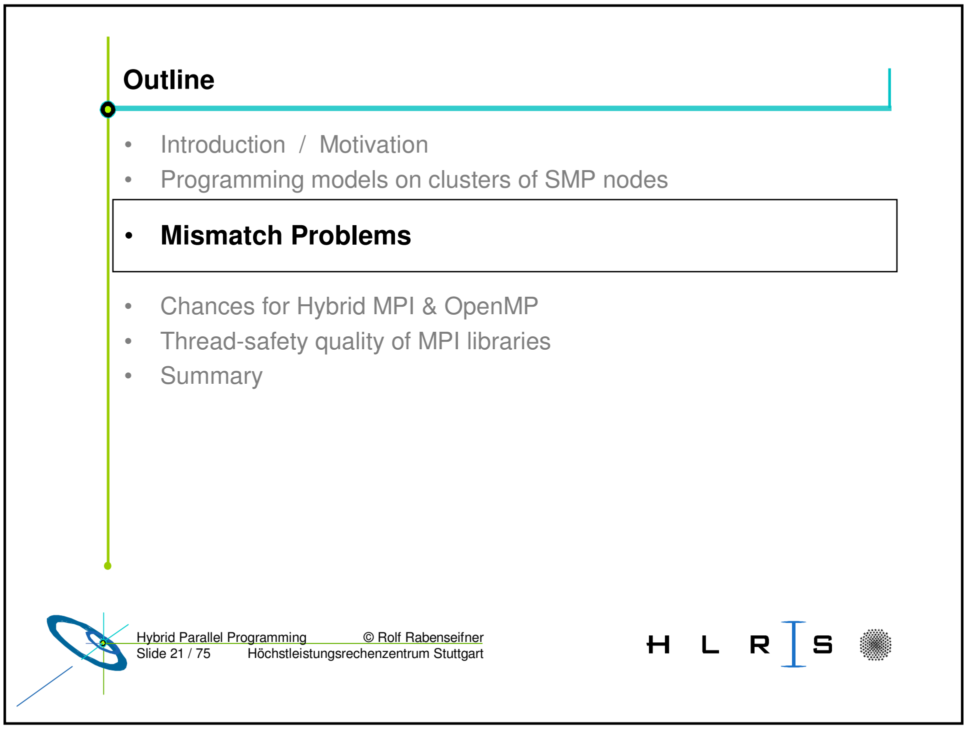


Pure OpenMP (on the cluster)

OpenMP only
distributed virtual
shared memory

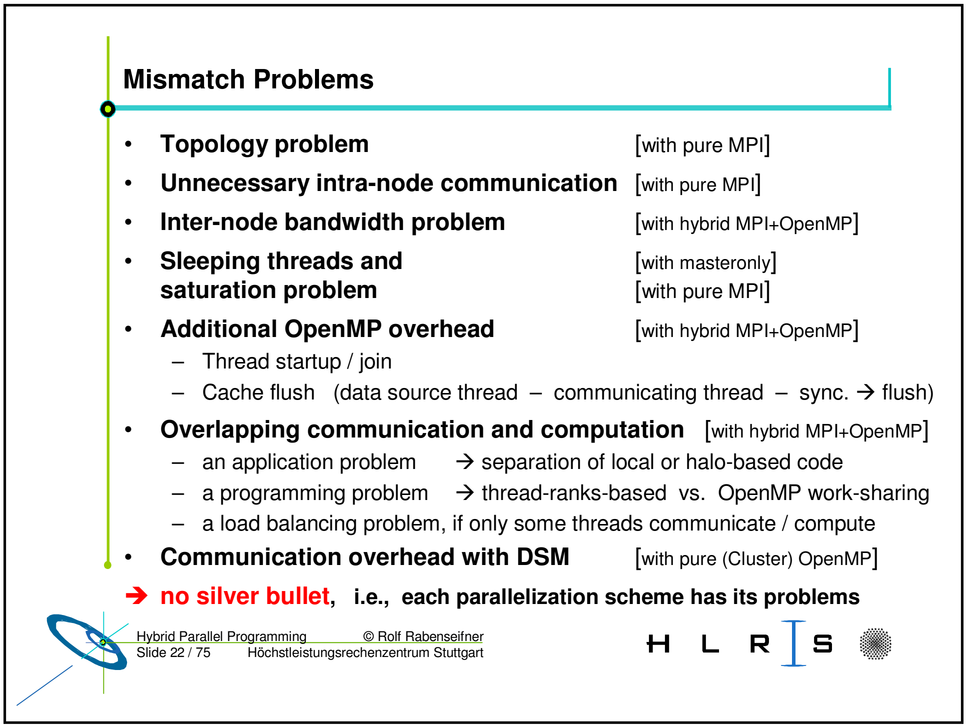
- Distributed shared virtual memory system needed
- Must support clusters of SMP nodes
- e.g., Intel® Cluster OpenMP
  - Shared memory parallel inside of SMP nodes
  - Communication of modified parts of pages at OpenMP flush (part of each OpenMP barrier)

i.e., the OpenMP memory and parallelization model
is prepared for clusters!



Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- **Mismatch Problems**
- Chances for Hybrid MPI & OpenMP
- Thread-safety quality of MPI libraries
- Summary



Mismatch Problems

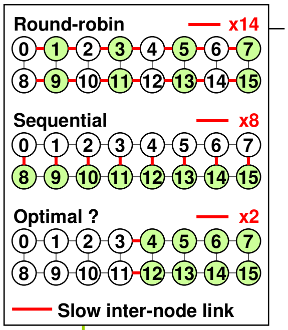
- **Topology problem** [with pure MPI]
- **Unnecessary intra-node communication** [with pure MPI]
- **Inter-node bandwidth problem** [with hybrid MPI+OpenMP]
- **Sleeping threads and** [with masteronly]
  saturation problem [with pure MPI]
- **Additional OpenMP overhead** [with hybrid MPI+OpenMP]
  - Thread startup / join
  - Cache flush (data source thread – communicating thread – sync. → flush)
- **Overlapping communication and computation** [with hybrid MPI+OpenMP]
  - an application problem → separation of local or halo-based code
  - a programming problem → thread-ranks-based vs. OpenMP work-sharing
  - a load balancing problem, if only some threads communicate / compute
- **Communication overhead with DSM** [with pure (Cluster) OpenMP]

➔ **no silver bullet, i.e., each parallelization scheme has its problems**

Mismatch Problems
- ➢ Topology problem
- Unnecessary intra-node comm.
- Inter-node bandwidth problem
- Sleeping threads and saturation problem
- Additional OpenMP overhead
- Overlapping comm. and comp.
- Communication overhead w. DSM

The Topology Problem with **pure MPI** one MPI process on each CPU

Exa.: 2 SMP nodes, 8 CPUs/node



- Problems
  - To fit application topology on hardware topology

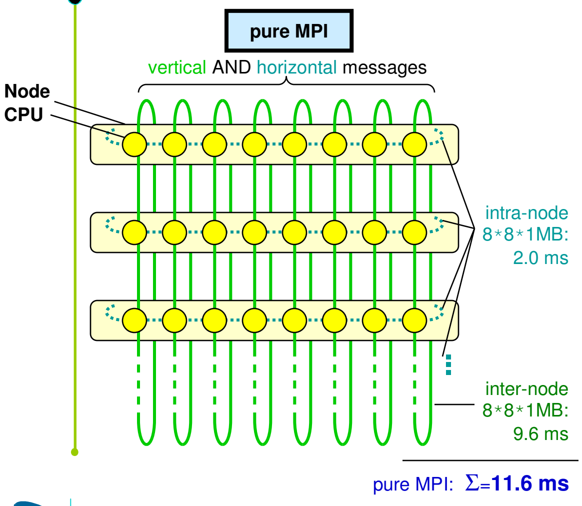
Solutions for Cartesian grids:
- E.g. choosing ranks in MPI_COMM_WORLD ???
  - **round robin (rank 0 on node 0, rank 1 on node 1, ...)**
  - **Sequential (ranks 0-7 on 1st node, ranks 8-15 on 2nd …)**

… in general
- load balancing in two steps:
  - **all cells among the SMP nodes (e.g. with ParMetis)**
  - **inside of each node: distributing the cells among the CPUs**
- or …

→ using hybrid programming models

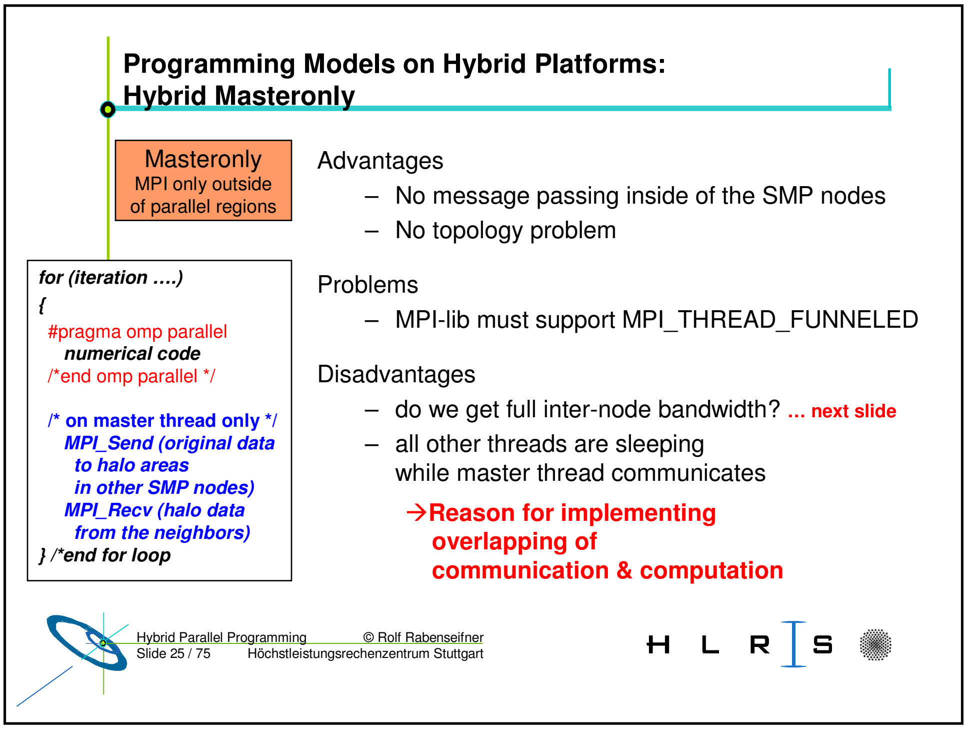
Mismatch Problems
- Topology problem
- ➢ Unnecessary intra-node comm.
- Inter-node bandwidth problem
- Sleeping threads and saturation problem
- Additional OpenMP overhead
- Overlapping comm. and comp.
- Communication overhead w. DSM

Unnecessary intra-node communication



Alternative:
- Hybrid MPI+OpenMP
- No intra-node messages
- Longer inter-node messages
- **Really faster ???????**
  (… wait 2 slides)

Timing:
Hitachi SR8000, MPI_Sendrecv
8 nodes, each node with 8 CPUs



Programming Models on Hybrid Platforms: Hybrid Masteronly

Masteronly
MPI only outside of parallel regions

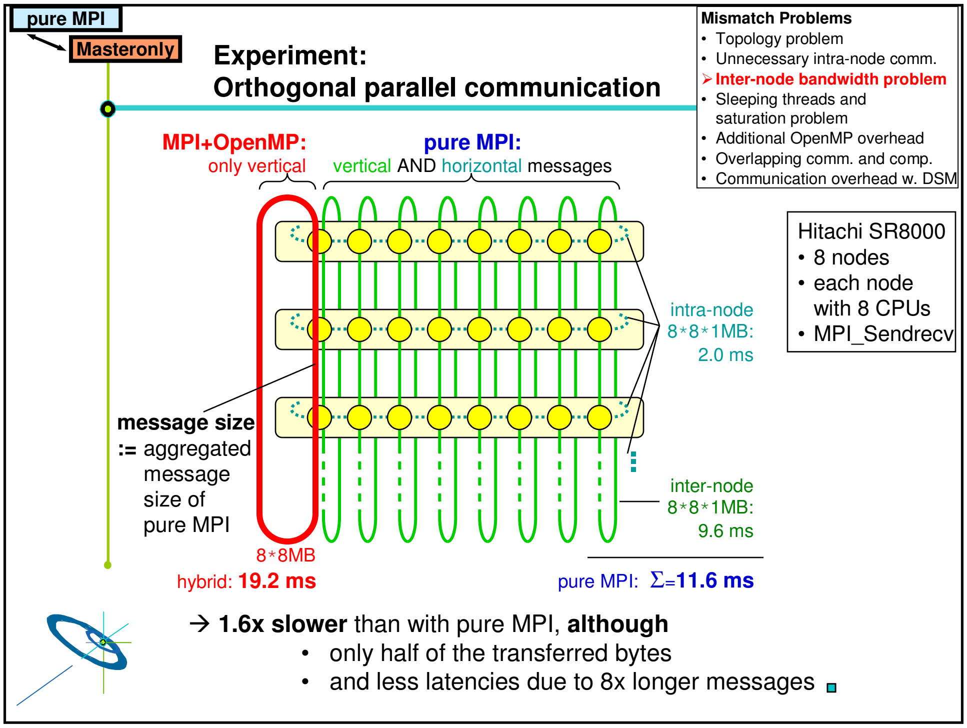
```
for (iteration ….)
{
  #pragma omp parallel
     numerical code
  /*end omp parallel */

  /* on master thread only */
     MPI_Send (original data
       to halo areas
       in other SMP nodes)
     MPI_Recv (halo data
       from the neighbors)
} /*end for loop
```

Advantages
- – No message passing inside of the SMP nodes
- – No topology problem

Problems
- – MPI-lib must support MPI_THREAD_FUNNELED

Disadvantages
- – do we get full inter-node bandwidth? **… next slide**
- – all other threads are sleeping while master thread communicates

→Reason for implementing overlapping of communication & computation



pure MPI

Masteronly

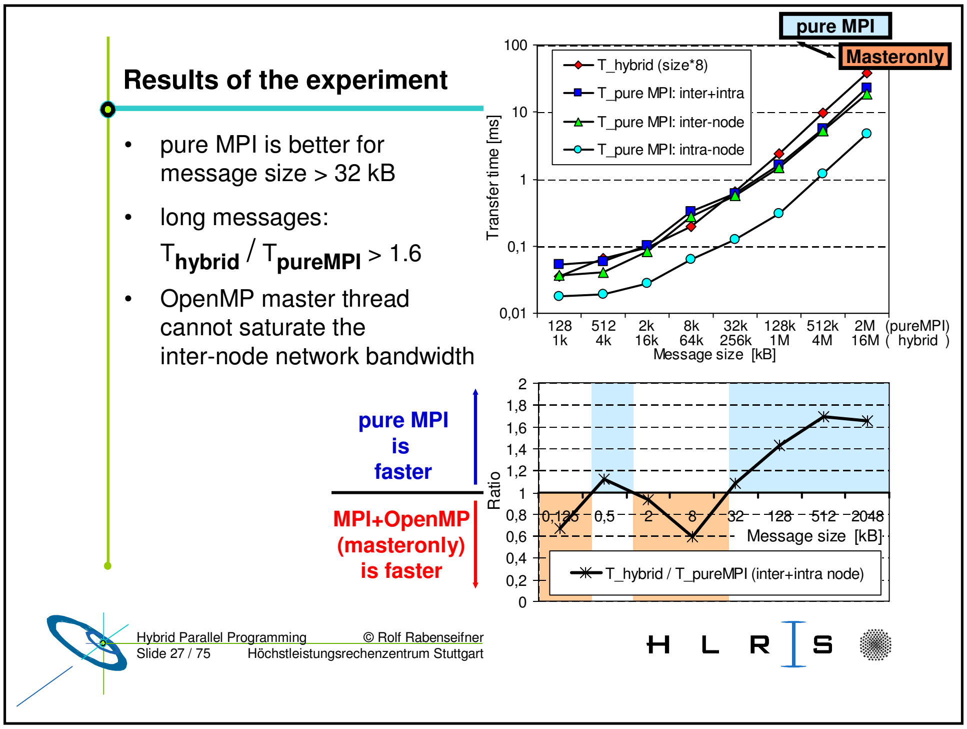
Experiment: Orthogonal parallel communication

Mismatch Problems
- Topology problem
- Unnecessary intra-node comm.
- ➢ **Inter-node bandwidth problem**
- Sleeping threads and saturation problem
- Additional OpenMP overhead
- Overlapping comm. and comp.
- Communication overhead w. DSM

Hitachi SR8000
- 8 nodes
- each node with 8 CPUs
- MPI_Sendrecv

→ **1.6x slower** than with pure MPI, **although**
- only half of the transferred bytes
- and less latencies due to 8x longer messages

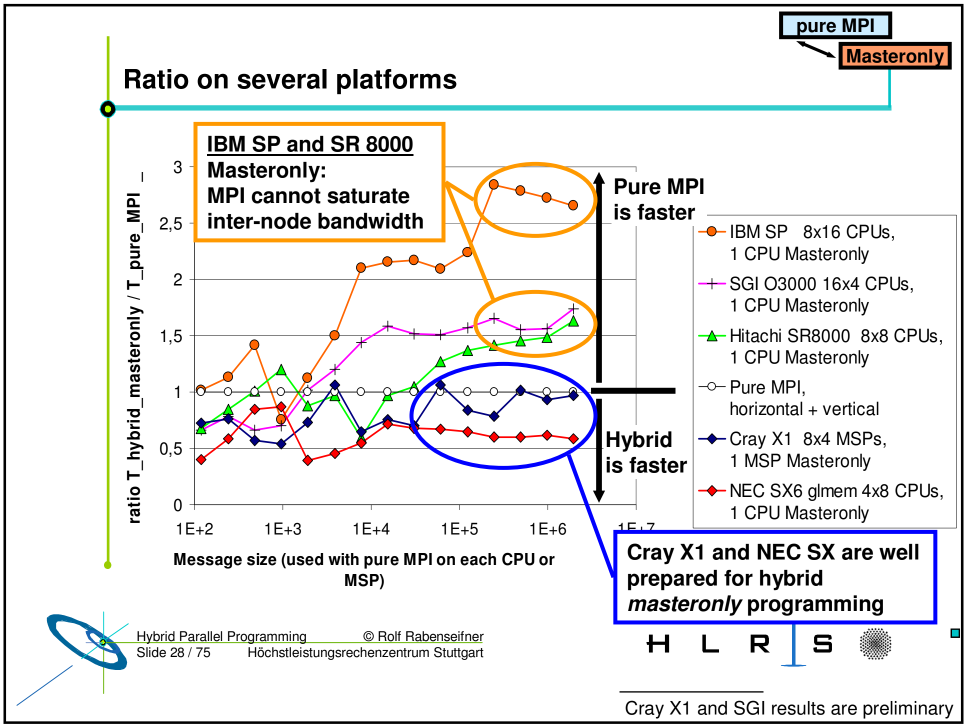
— Hybrid MPI and OpenMP Parallel Programming —
Lecture at IWR, Heidelberg, July 11, 2008



Results of the experiment

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

Hybrid Parallel Programming — © Rolf Rabenseifner
Slide 27 / 75 — Höchstleistungsrechenzentrum Stuttgart



Ratio on several platforms

IBM SP and SR 8000
Masteronly:
MPI cannot saturate inter-node bandwidth

Cray X1 and NEC SX are well prepared for hybrid *masteronly* programming

Hybrid Parallel Programming — © Rolf Rabenseifner
Slide 28 / 75 — Höchstleistungsrechenzentrum Stuttgart

Cray X1 and SGI results are preliminary

pure MPI
Masteronly

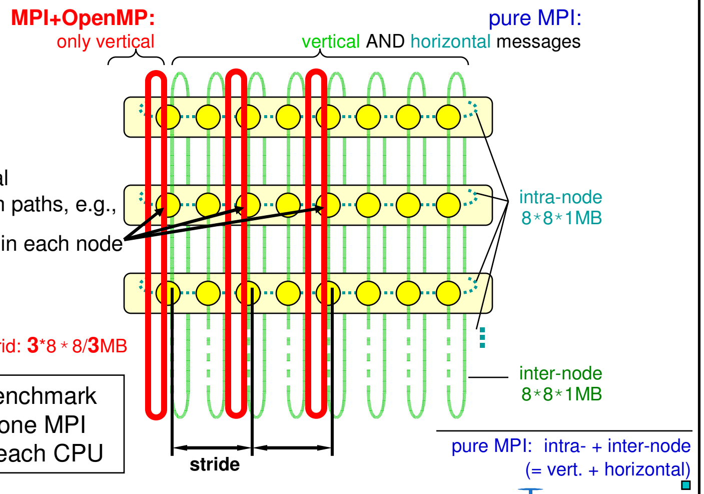
Possible Reasons

- Hardware:
  - is one CPU able to saturate the inter-node network?
- Software:
  - internal MPI buffering may cause additional memory traffic
    → memory bandwidth may be the real restricting factor?

➔ Let’s look at parallel bandwidth results

pure MPI
Masteronly

Multiple inter-node communication paths



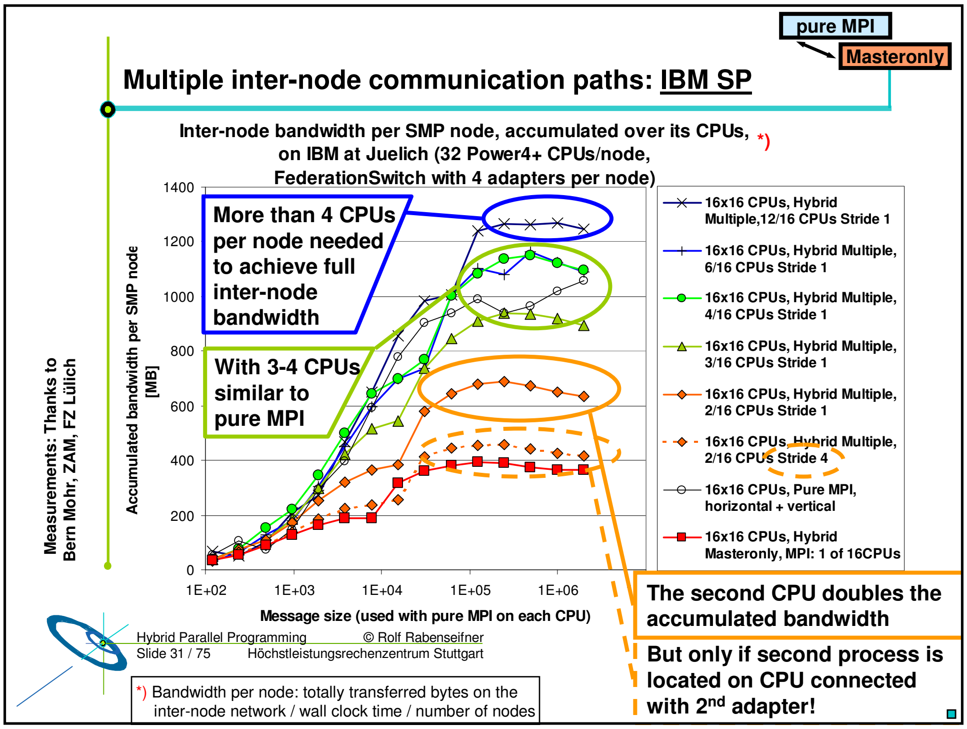
Multiple vertical communication paths, e.g.,

- **3** of 8 CPUs in each node
- **stride 2**

hybrid: **3***8 * 8/**3**MB

Following benchmark results with one MPI process on each CPU

pure MPI: intra- + inter-node
(= vert. + horizontal)



pure MPI → Masteronly

Multiple inter-node communication paths: IBM SP

Inter-node bandwidth per SMP node, accumulated over its CPUs, *)
on IBM at Juelich (32 Power4+ CPUs/node,
FederationSwitch with 4 adapters per node)

More than 4 CPUs per node needed to achieve full inter-node bandwidth

With 3-4 CPUs similar to pure MPI

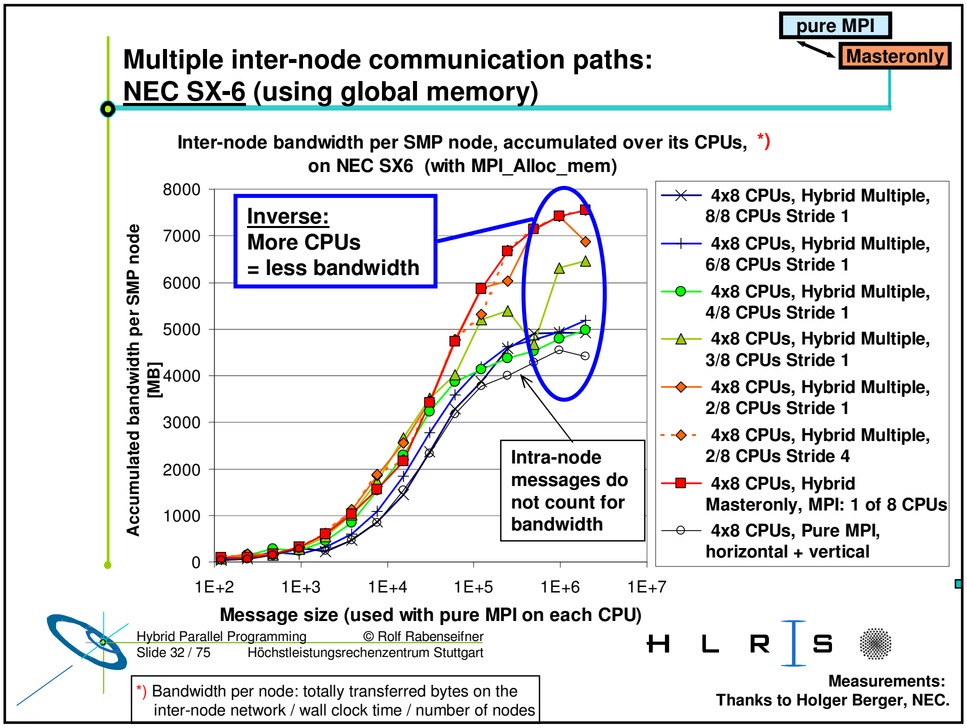
The second CPU doubles the accumulated bandwidth

But only if second process is located on CPU connected with 2nd adapter!

Measurements: Thanks to Bern Mohr, ZAM, FZ Lülich

Hybrid Parallel Programming © Rolf Rabenseifner
Slide 31 / 75 Höchstleistungsrechenzentrum Stuttgart

*) Bandwidth per node: totally transferred bytes on the inter-node network / wall clock time / number of nodes



pure MPI → Masteronly

Multiple inter-node communication paths: NEC SX-6 (using global memory)

Inter-node bandwidth per SMP node, accumulated over its CPUs, *)
on NEC SX6 (with MPI_Alloc_mem)

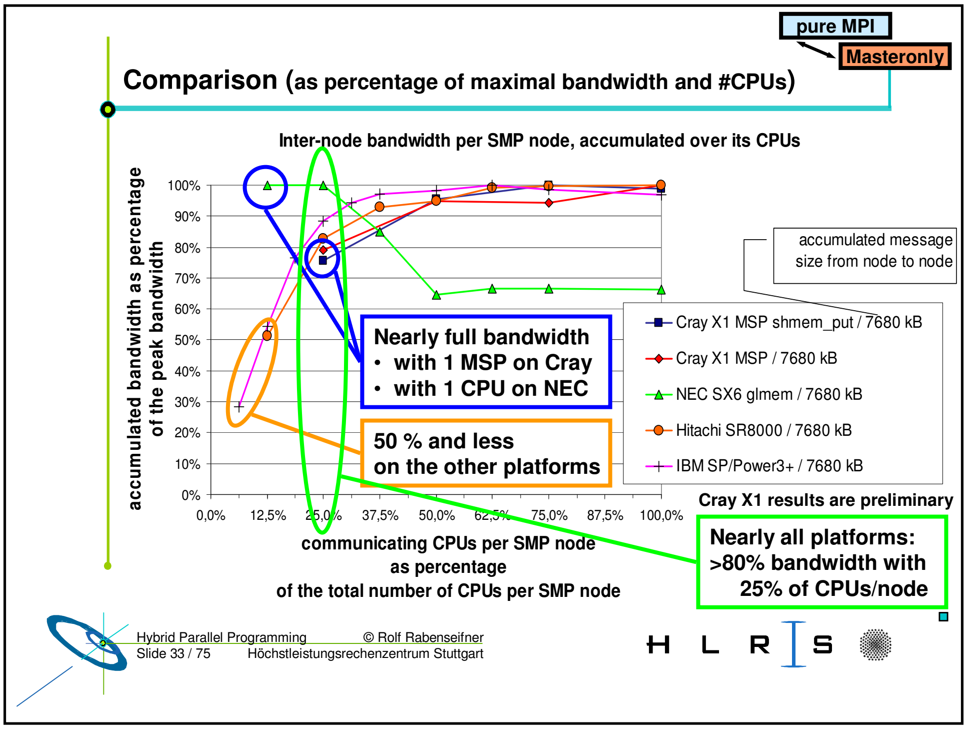
Inverse:
More CPUs
= less bandwidth

Intra-node messages do not count for bandwidth

Hybrid Parallel Programming © Rolf Rabenseifner
Slide 32 / 75 Höchstleistungsrechenzentrum Stuttgart

*) Bandwidth per node: totally transferred bytes on the inter-node network / wall clock time / number of nodes

Measurements:
Thanks to Holger Berger, NEC.



pure MPI → Masteronly

Comparison (as percentage of maximal bandwidth and #CPUs)

Nearly full bandwidth
- with 1 MSP on Cray
- with 1 CPU on NEC

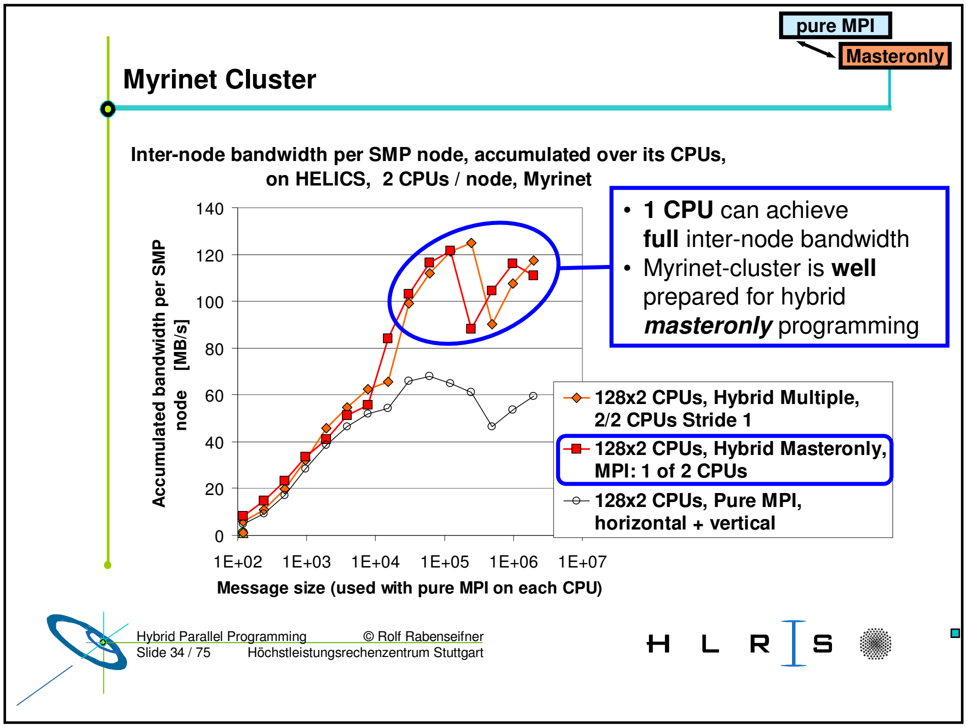
50 % and less on the other platforms

Cray X1 results are preliminary

Nearly all platforms: >80% bandwidth with 25% of CPUs/node

Hybrid Parallel Programming © Rolf Rabenseifner
Slide 33 / 75 Höchstleistungsrechenzentrum Stuttgart

H L R I S



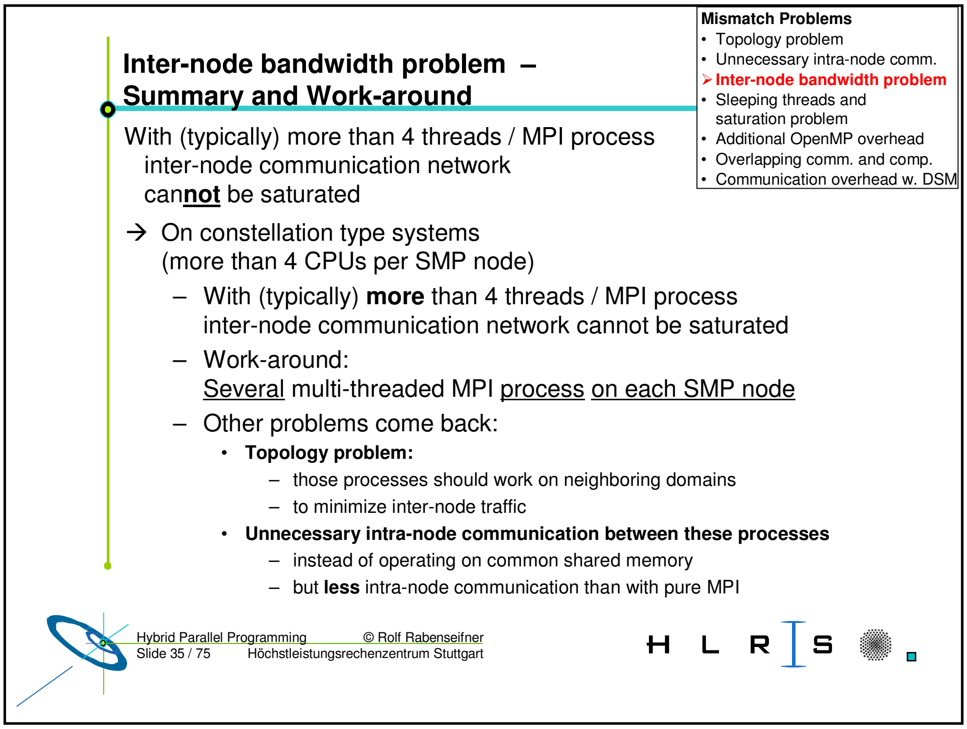
pure MPI → Masteronly

Myrinet Cluster

- **1 CPU** can achieve **full** inter-node bandwidth
- Myrinet-cluster is **well** prepared for hybrid ***masteronly*** programming

Hybrid Parallel Programming © Rolf Rabenseifner
Slide 34 / 75 Höchstleistungsrechenzentrum Stuttgart

H L R I S



Mismatch Problems
- Topology problem
- Unnecessary intra-node comm.
- ➢ **Inter-node bandwidth problem**
- Sleeping threads and saturation problem
- Additional OpenMP overhead
- Overlapping comm. and comp.
- Communication overhead w. DSM

Inter-node bandwidth problem – Summary and Work-around

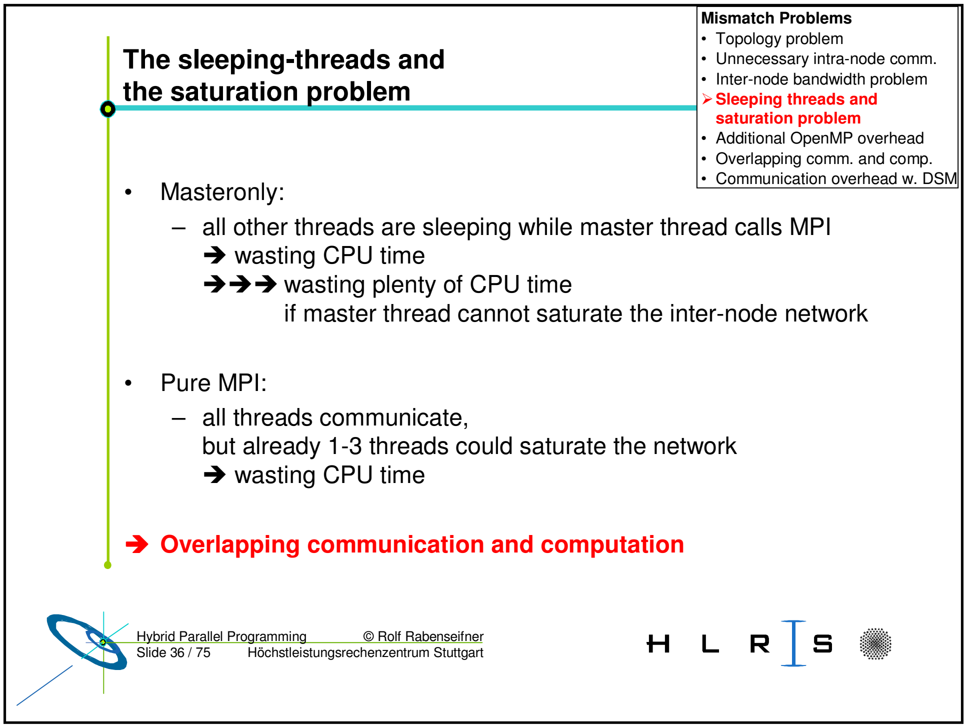
With (typically) more than 4 threads / MPI process
inter-node communication network
can**not** be saturated

→ On constellation type systems
(more than 4 CPUs per SMP node)

- With (typically) **more** than 4 threads / MPI process inter-node communication network cannot be saturated
- Work-around:
  Several multi-threaded MPI process on each SMP node
- Other problems come back:
  - **Topology problem:**
    - those processes should work on neighboring domains
    - to minimize inter-node traffic
  - **Unnecessary intra-node communication between these processes**
    - instead of operating on common shared memory
    - but **less** intra-node communication than with pure MPI

Hybrid Parallel Programming © Rolf Rabenseifner
Slide 35 / 75 Höchstleistungsrechenzentrum Stuttgart

H L R S



Mismatch Problems
- Topology problem
- Unnecessary intra-node comm.
- Inter-node bandwidth problem
- ➢ **Sleeping threads and saturation problem**
- Additional OpenMP overhead
- Overlapping comm. and comp.
- Communication overhead w. DSM

The sleeping-threads and the saturation problem

- Masteronly:
  - all other threads are sleeping while master thread calls MPI
    ➔ wasting CPU time
    ➔➔➔ wasting plenty of CPU time
    if master thread cannot saturate the inter-node network
- Pure MPI:
  - all threads communicate,
    but already 1-3 threads could saturate the network
    ➔ wasting CPU time

➔ Overlapping communication and computation

Hybrid Parallel Programming © Rolf Rabenseifner
Slide 36 / 75 Höchstleistungsrechenzentrum Stuttgart

H L R S

Additional OpenMP Overhead

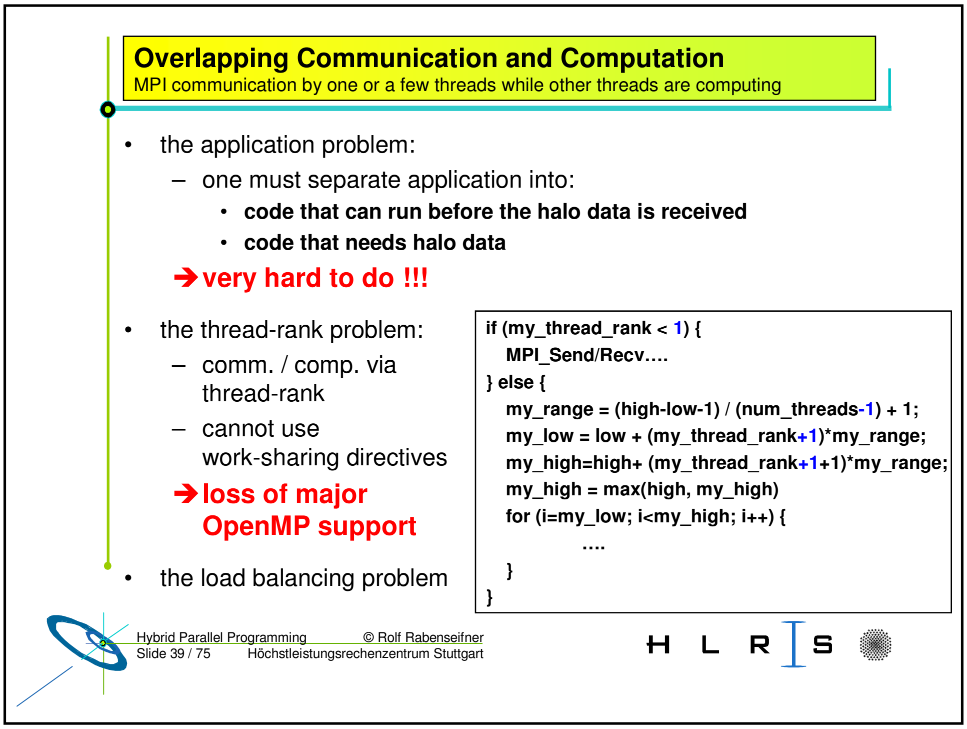
Mismatch Problems
- Topology problem
- Unnecessary intra-node comm.
- Inter-node bandwidth problem
- Sleeping threads and saturation problem
- ➢ **Additional OpenMP overhead**
- Overlapping comm. and comp.
- Communication overhead w. DSM

- Thread fork / join
- Cache flush
  - synchronization between *data source thread* and *communicating thread* implies → a cache flush
- Amdahl's law for each level of parallelism

Mismatch Problems

• Topology problem	[with pure MPI]
• Unnecessary intra-node communication	[with pure MPI]
• Inter-node bandwidth problem	[with hybrid MPI+OpenMP]
• Sleeping threads and saturation problem	[with masteronly] [with pure MPI]
• Additional OpenMP overhead	[with hybrid MPI+OpenMP]
– Thread fork / join	
– Cache flush (data source thread – communicating thread – sync. → flush)	
• **Overlapping communication and computation**	[with hybrid MPI+OpenMP]
– an application problem → separation of local or halo-based code	
– a programming problem → thread-ranks-based vs. OpenMP work-sharing	
– a load balancing problem, if only some threads communicate / compute	
• Communication overhead with DSM	[with pure (Cluster) OpenMP]

→ no silver bullet, **i.e., each parallelization scheme has its problems**



Overlapping Communication and Computation

MPI communication by one or a few threads while other threads are computing

- the application problem:
  - one must separate application into:
    - **code that can run before the halo data is received**
    - **code that needs halo data**

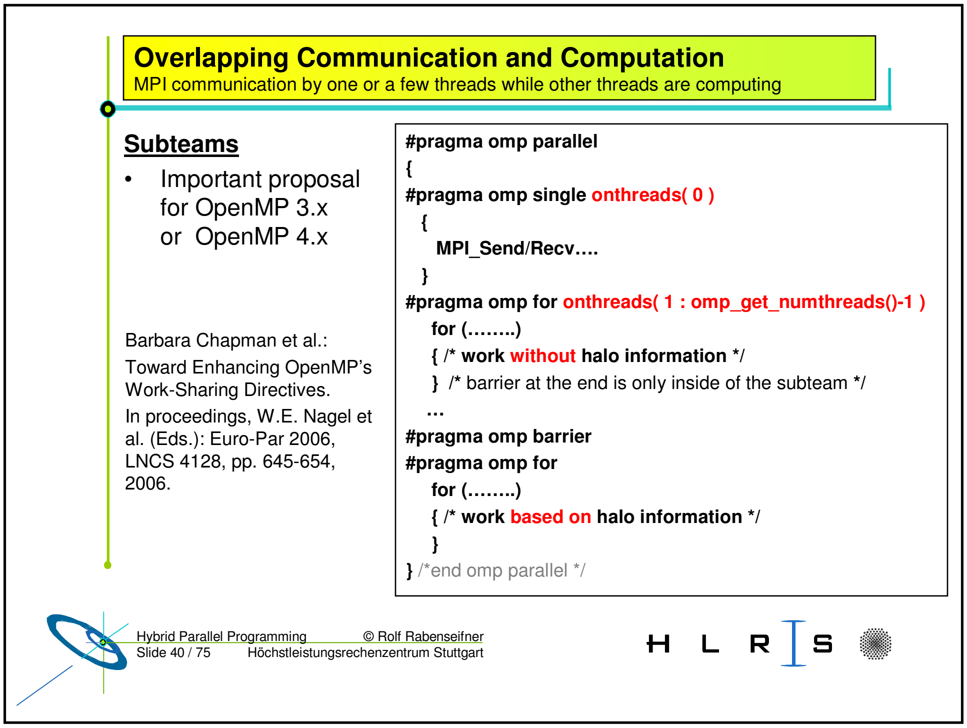
  ➔ **very hard to do !!!**

- the thread-rank problem:
  - comm. / comp. via thread-rank
  - cannot use work-sharing directives

  ➔ **loss of major OpenMP support**

- the load balancing problem

```
if (my_thread_rank < 1) {
   MPI_Send/Recv….
} else {
   my_range = (high-low-1) / (num_threads-1) + 1;
   my_low = low + (my_thread_rank+1)*my_range;
   my_high=high+ (my_thread_rank+1+1)*my_range;
   my_high = max(high, my_high)
   for (i=my_low; i<my_high; i++) {
            ….
   }
}
```



Overlapping Communication and Computation

MPI communication by one or a few threads while other threads are computing

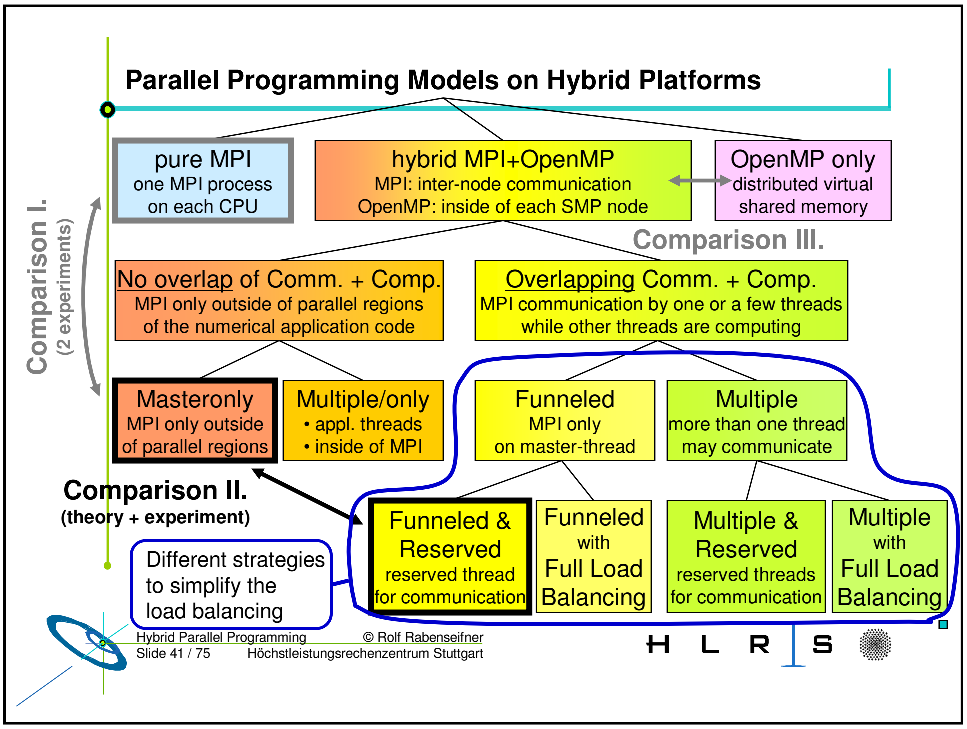
Subteams

- Important proposal for OpenMP 3.x or OpenMP 4.x

Barbara Chapman et al.: Toward Enhancing OpenMP's Work-Sharing Directives.

In proceedings, W.E. Nagel et al. (Eds.): Euro-Par 2006, LNCS 4128, pp. 645-654, 2006.

```
#pragma omp parallel
{
#pragma omp single onthreads( 0 )
  {
    MPI_Send/Recv….
  }
#pragma omp for onthreads( 1 : omp_get_numthreads()-1 )
     for (……..)
     { /* work without halo information */
     }  /* barrier at the end is only inside of the subteam */
   …
#pragma omp barrier
#pragma omp for
     for (……..)
     { /* work based on halo information */
     }
} /*end omp parallel */
```



Parallel Programming Models on Hybrid Platforms

pure MPI
one MPI process
on each CPU

hybrid MPI+OpenMP
MPI: inter-node communication
OpenMP: inside of each SMP node

OpenMP only
distributed virtual
shared memory

Comparison III.

No overlap of Comm. + Comp.
MPI only outside of parallel regions
of the numerical application code

Overlapping Comm. + Comp.
MPI communication by one or a few threads
while other threads are computing

Masteronly
MPI only outside
of parallel regions

Multiple/only
• appl. threads
• inside of MPI

Funneled
MPI only
on master-thread

Multiple
more than one thread
may communicate

Funneled & Reserved
reserved thread
for communication

Funneled with Full Load Balancing

Multiple & Reserved
reserved threads
for communication

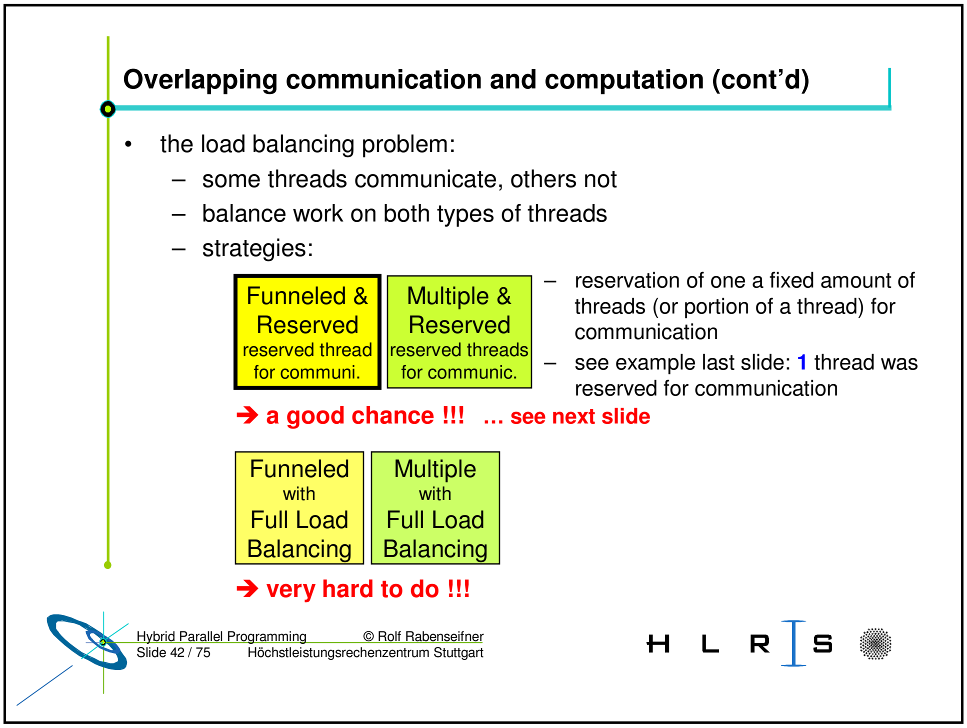
Multiple with Full Load Balancing

Comparison I.
(2 experiments)

Comparison II.
(theory + experiment)

Different strategies
to simplify the
load balancing

Hybrid Parallel Programming © Rolf Rabenseifner
Slide 41 / 75 Höchstleistungsrechenzentrum Stuttgart



Overlapping communication and computation (cont'd)

- the load balancing problem:
  - some threads communicate, others not
  - balance work on both types of threads
  - strategies:

Funneled & Reserved reserved thread for communi.	Multiple & Reserved reserved threads for communic.

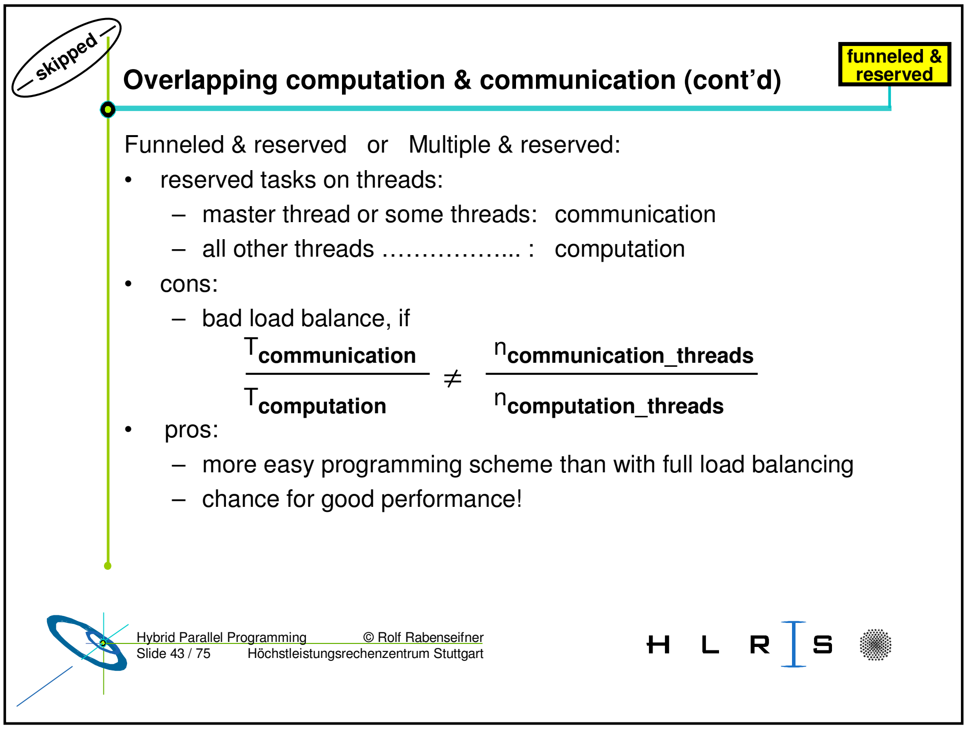
- reservation of one a fixed amount of threads (or portion of a thread) for communication
- see example last slide: **1** thread was reserved for communication

➔ a good chance !!!   ... see next slide

Funneled with Full Load Balancing	Multiple with Full Load Balancing

➔ very hard to do !!!

Hybrid Parallel Programming © Rolf Rabenseifner
Slide 42 / 75 Höchstleistungsrechenzentrum Stuttgart



skipped

funneled & reserved

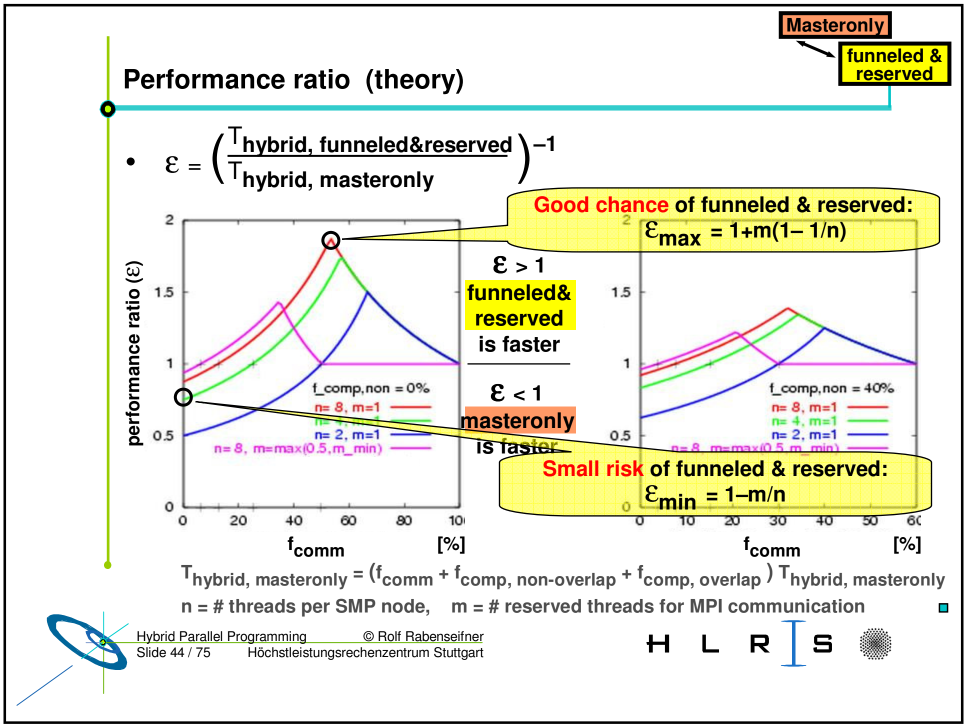
Overlapping computation & communication (cont'd)

Funneled & reserved or Multiple & reserved:

- reserved tasks on threads:
  - master thread or some threads: communication
  - all other threads ……………... : computation
- cons:
  - bad load balance, if

$$\frac{T_{\text{communication}}}{T_{\text{computation}}} \neq \frac{n_{\text{communication\_threads}}}{n_{\text{computation\_threads}}}$$

- pros:
  - more easy programming scheme than with full load balancing
  - chance for good performance!



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:
$\varepsilon_{\max} = 1+m(1-1/n)$

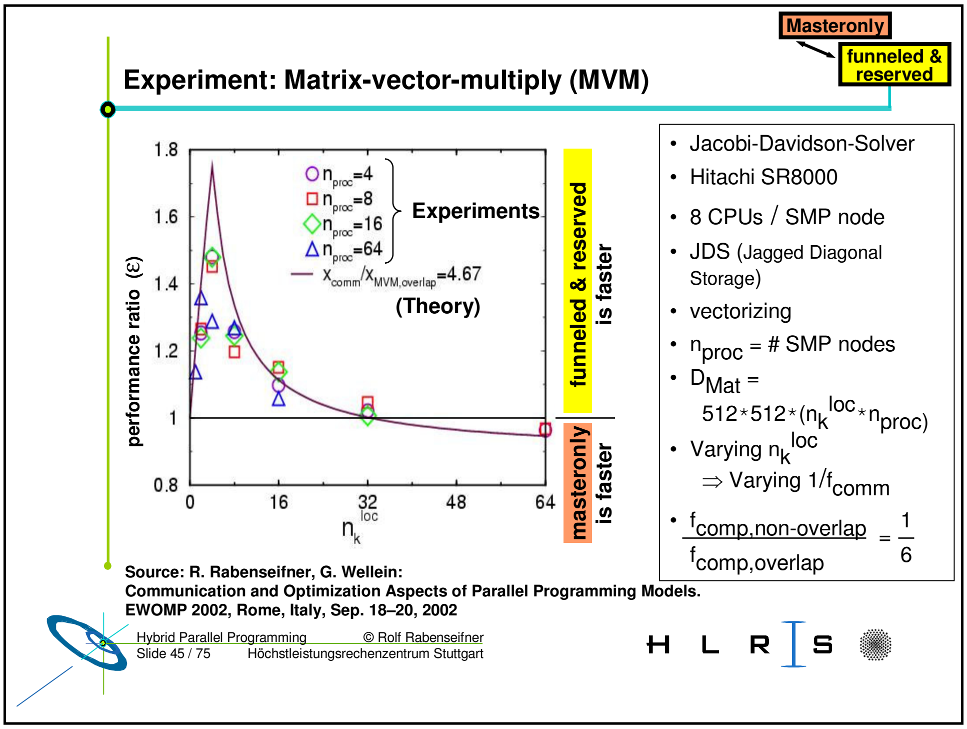
$\varepsilon > 1$ **funneled& reserved is faster**

$\varepsilon < 1$ **masteronly is faster**

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

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

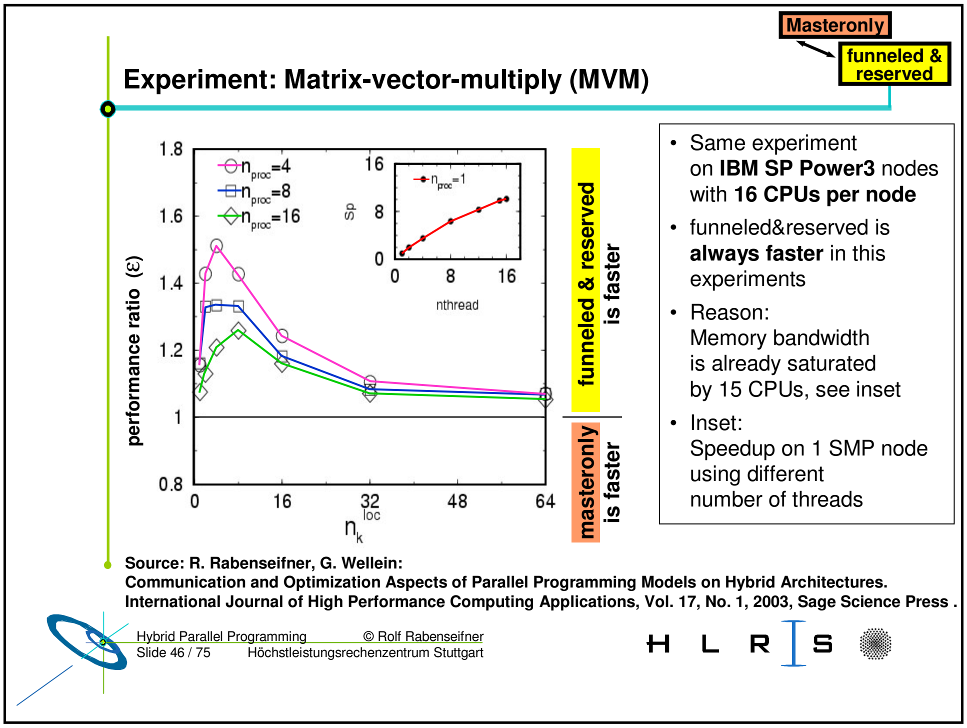
n = # threads per SMP node, m = # reserved threads for MPI communication



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} $\Rightarrow$ Varying $1/f_{comm}$
- $\dfrac{f_{comm,non-overlap}}{f_{comm,overlap}} = \dfrac{1}{6}$

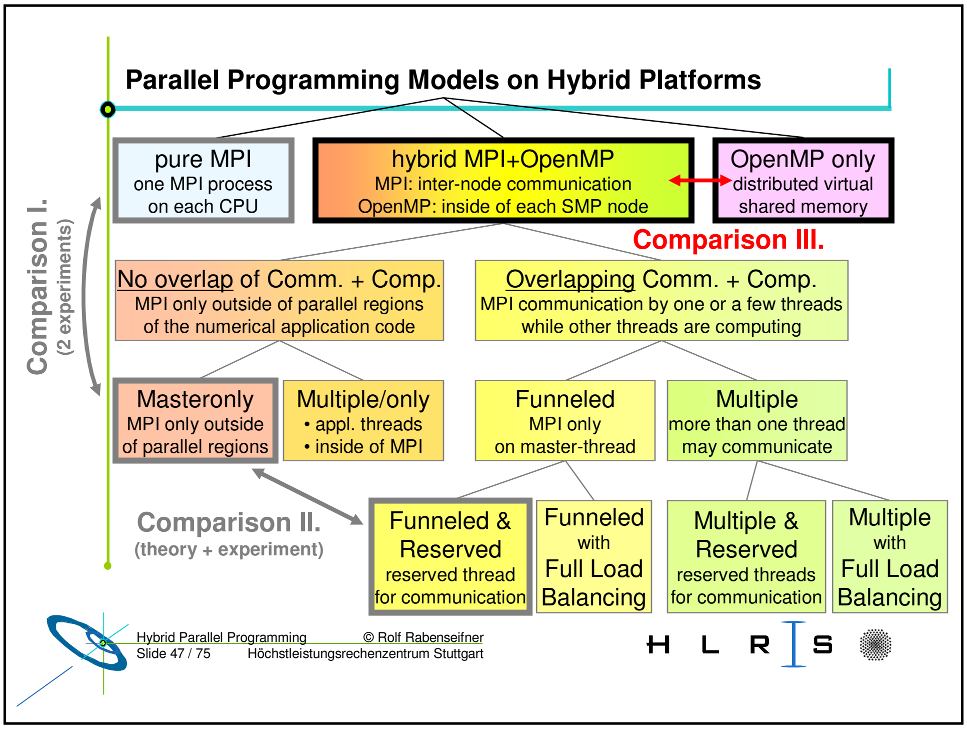
Source: R. Rabenseifner, G. Wellein:
Communication and Optimization Aspects of Parallel Programming Models.
EWOMP 2002, Rome, Italy, Sep. 18–20, 2002



Experiment: Matrix-vector-multiply (MVM)

- Same experiment on **IBM SP Power3** nodes with **16 CPUs per node**
- funneled&reserved is **always faster** in this experiments
- Reason: Memory bandwidth is already saturated by 15 CPUs, see inset
- Inset: Speedup on 1 SMP node using different number of threads

Source: R. Rabenseifner, G. Wellein:
Communication and Optimization Aspects of Parallel Programming Models on Hybrid Architectures.
International Journal of High Performance Computing Applications, Vol. 17, No. 1, 2003, Sage Science Press .



Parallel Programming Models on Hybrid Platforms

pure MPI
one MPI process on each CPU

hybrid MPI+OpenMP
MPI: inter-node communication
OpenMP: inside of each SMP node

OpenMP only
distributed virtual shared memory

Comparison III.

No overlap of Comm. + Comp.
MPI only outside of parallel regions of the numerical application code

Overlapping Comm. + Comp.
MPI communication by one or a few threads while other threads are computing

Masteronly
MPI only outside of parallel regions

Multiple/only
- appl. threads
- inside of MPI

Funneled
MPI only on master-thread

Multiple
more than one thread may communicate

Funneled & Reserved
reserved thread for communication

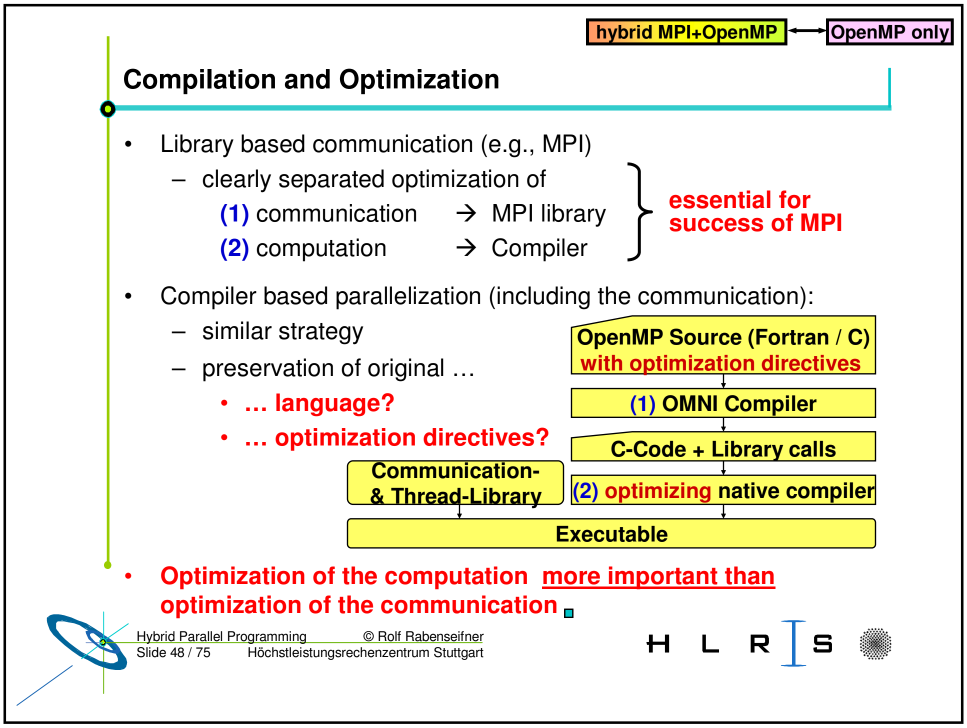
Funneled with Full Load Balancing

Multiple & Reserved
reserved threads for communication

Multiple with Full Load Balancing

Comparison I.
(2 experiments)

Comparison II.
(theory + experiment)



hybrid MPI+OpenMP ⟷ OpenMP only

Compilation and Optimization

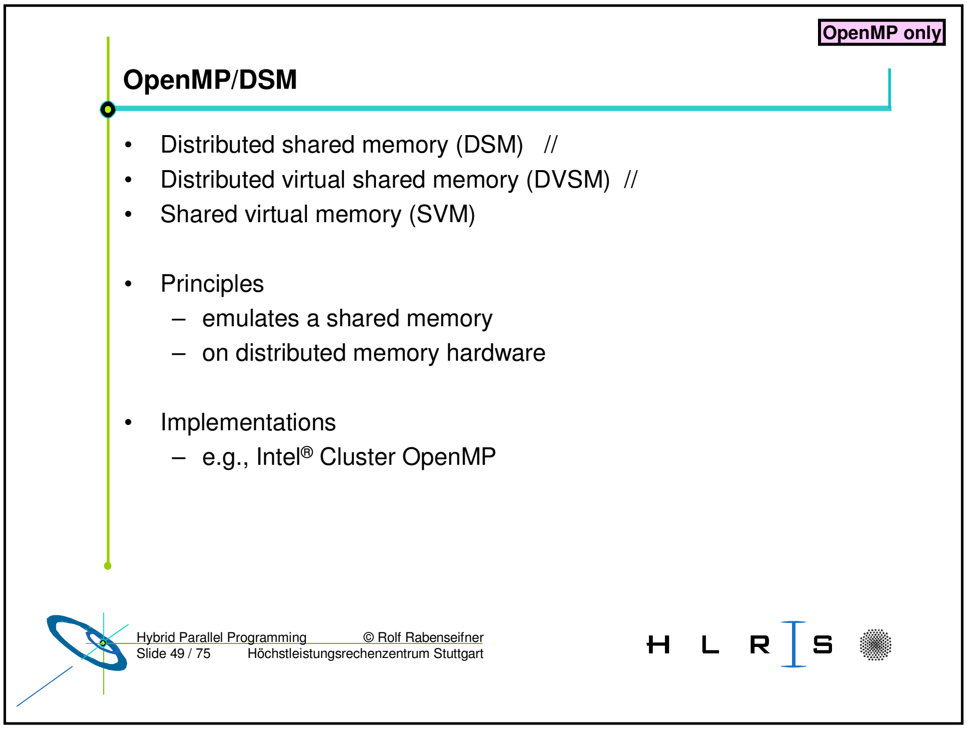
- Library based communication (e.g., MPI)
  - clearly separated optimization of
    - (1) communication → MPI library
    - (2) computation → Compiler

  } **essential for success of MPI**

- Compiler based parallelization (including the communication):
  - similar strategy
  - preservation of original …
    - **… language?**
    - **… optimization directives?**

OpenMP Source (Fortran / C) with optimization directives
↓
(1) OMNI Compiler
↓
C-Code + Library calls
↓
(2) optimizing native compiler ← **Communication- & Thread-Library**
↓
Executable

- **Optimization of the computation more important than optimization of the communication**



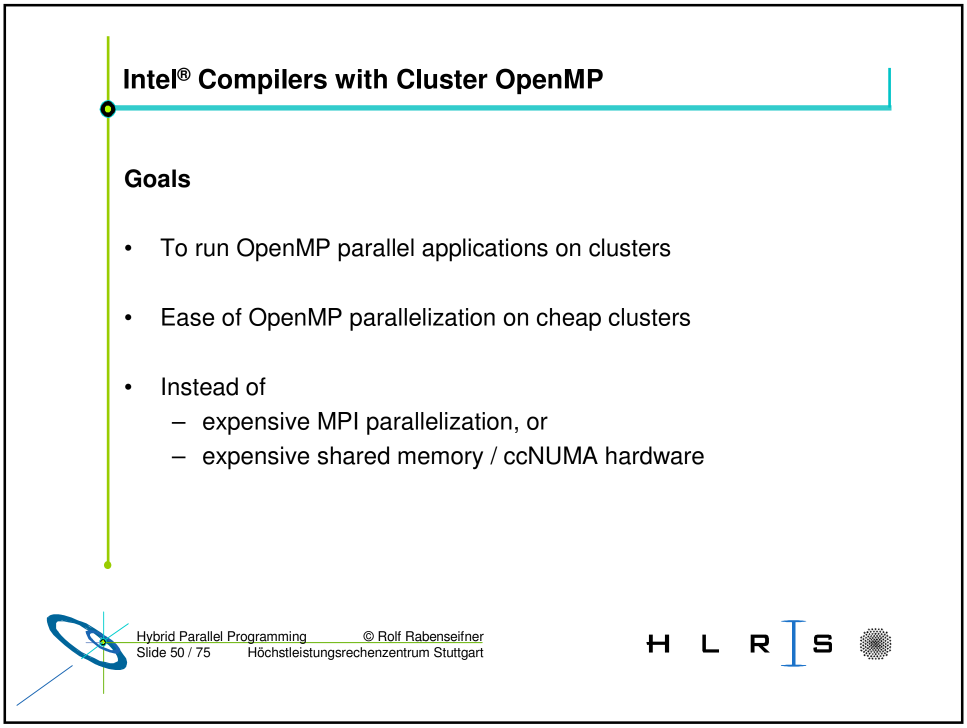
OpenMP only

OpenMP/DSM

- Distributed shared memory (DSM) //
- Distributed virtual shared memory (DVSM) //
- Shared virtual memory (SVM)

- Principles
  - emulates a shared memory
  - on distributed memory hardware

- Implementations
  - e.g., Intel® Cluster OpenMP



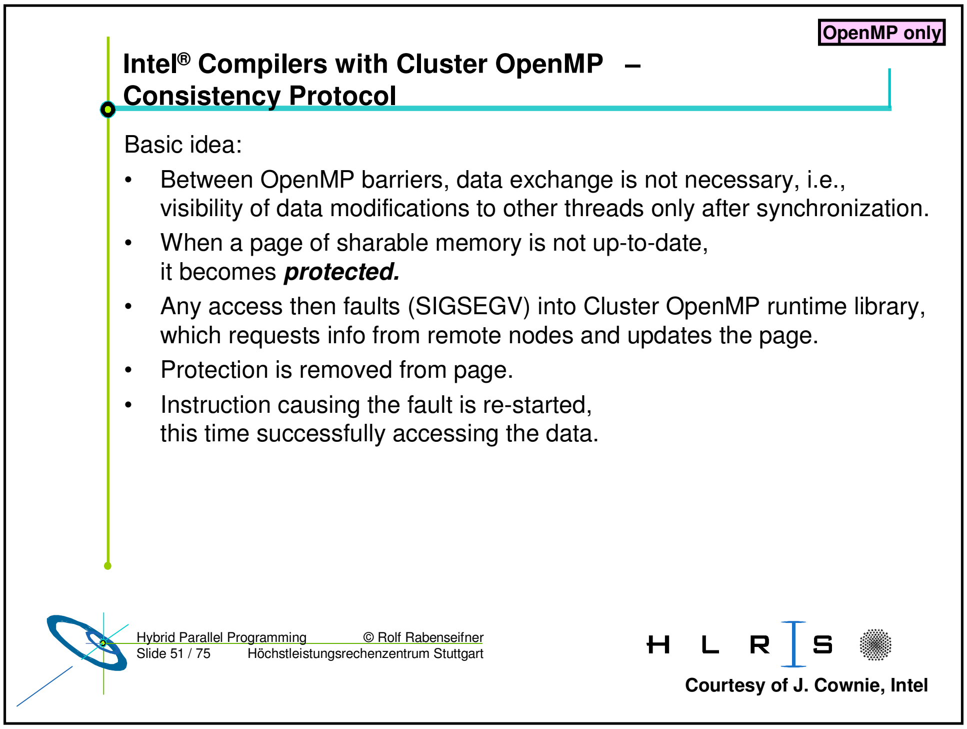
Intel® Compilers with Cluster OpenMP

Goals

- To run OpenMP parallel applications on clusters

- Ease of OpenMP parallelization on cheap clusters

- Instead of
  - expensive MPI parallelization, or
  - expensive shared memory / ccNUMA hardware



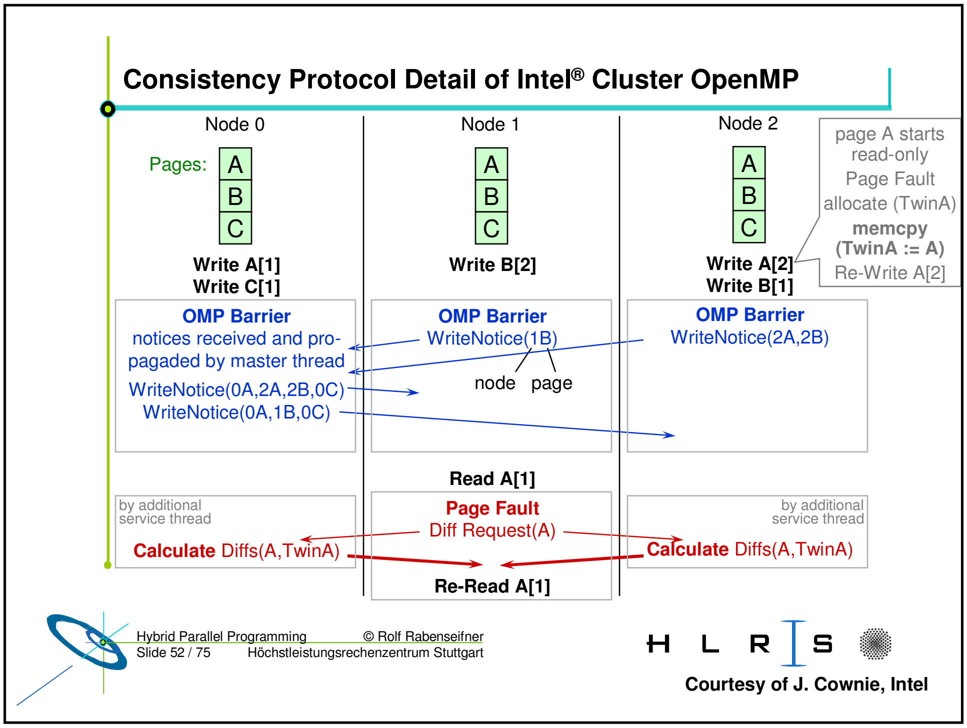
OpenMP only

Intel® Compilers with Cluster OpenMP – Consistency Protocol

Basic idea:

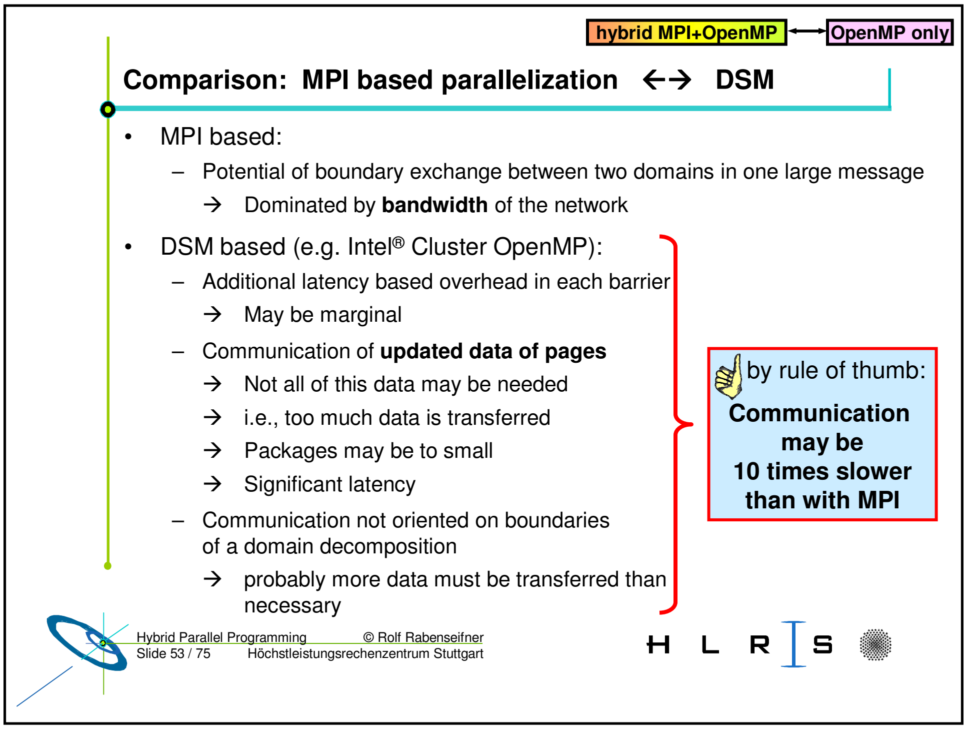
- Between OpenMP barriers, data exchange is not necessary, i.e., visibility of data modifications to other threads only after synchronization.
- When a page of sharable memory is not up-to-date, it becomes ***protected.***
- Any access then faults (SIGSEGV) into Cluster OpenMP runtime library, which requests info from remote nodes and updates the page.
- Protection is removed from page.
- Instruction causing the fault is re-started, this time successfully accessing the data.

Courtesy of J. Cownie, Intel



Consistency Protocol Detail of Intel® Cluster OpenMP

Courtesy of J. Cownie, Intel

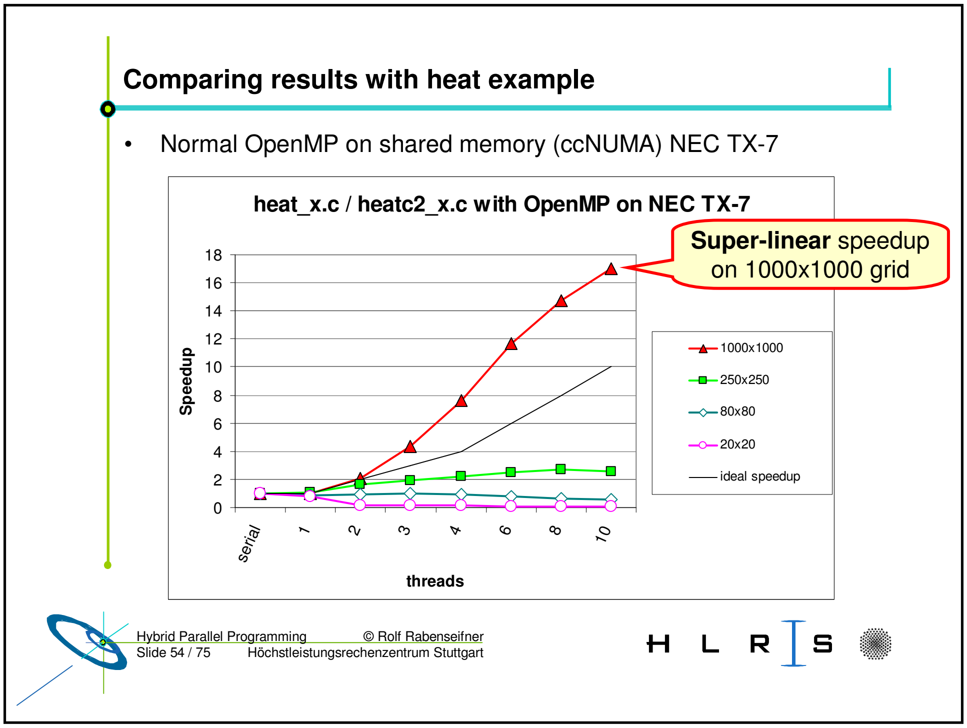


hybrid MPI+OpenMP
OpenMP only

Comparison: MPI based parallelization ←→ DSM

- MPI based:
  - Potential of boundary exchange between two domains in one large message
    → Dominated by **bandwidth** of the network
- DSM based (e.g. Intel® Cluster OpenMP):
  - Additional latency based overhead in each barrier
    → May be marginal
  - Communication of **updated data of pages**
    → Not all of this data may be needed
    → i.e., too much data is transferred
    → Packages may be to small
    → Significant latency
  - Communication not oriented on boundaries of a domain decomposition
    → probably more data must be transferred than necessary

by rule of thumb:
Communication may be 10 times slower than with MPI



Comparing results with heat example

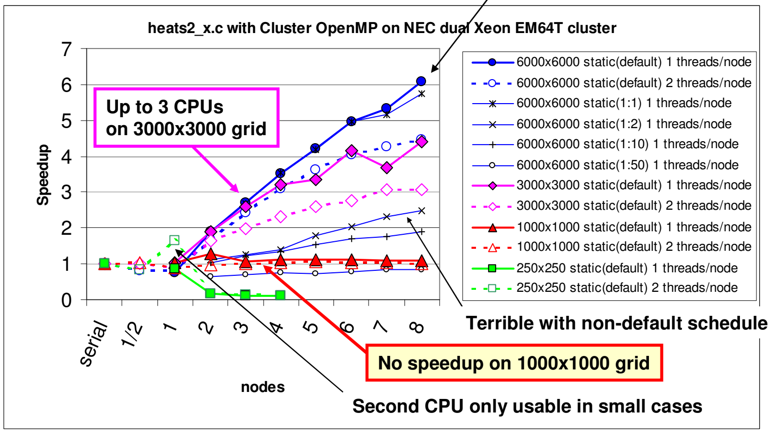
- Normal OpenMP on shared memory (ccNUMA) NEC TX-7

heat_x.c / heatc2_x.c with OpenMP on NEC TX-7

Super-linear speedup on 1000x1000 grid

Heat example: Cluster OpenMP Efficiency

- Cluster OpenMP on a Dual-Xeon cluster



Efficiency only with small communication foot-print

Up to 3 CPUs on 3000x3000 grid

Terrible with non-default schedule

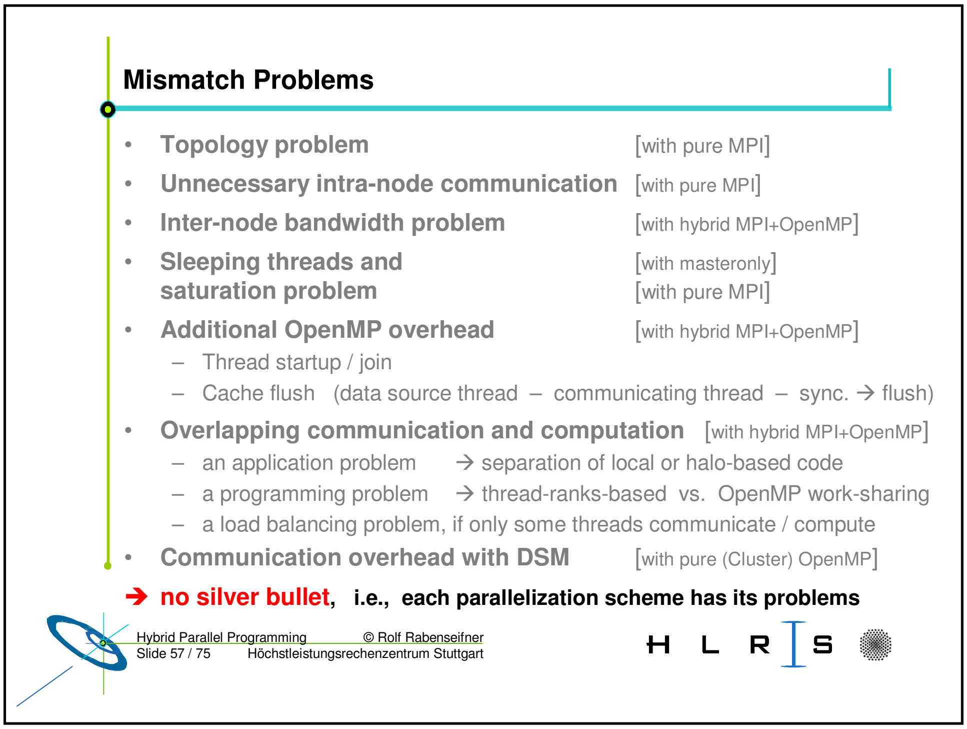
No speedup on 1000x1000 grid

Second CPU only usable in small cases

Cluster OpenMP – a summary

- **Intel® Cluster OpenMP** can be used for programs with small communication foot-print!
- Source code modification needed: shared variables must be allocated in ***sharable*** memory
- It works!
- But efficiency strongly depends on type of application!

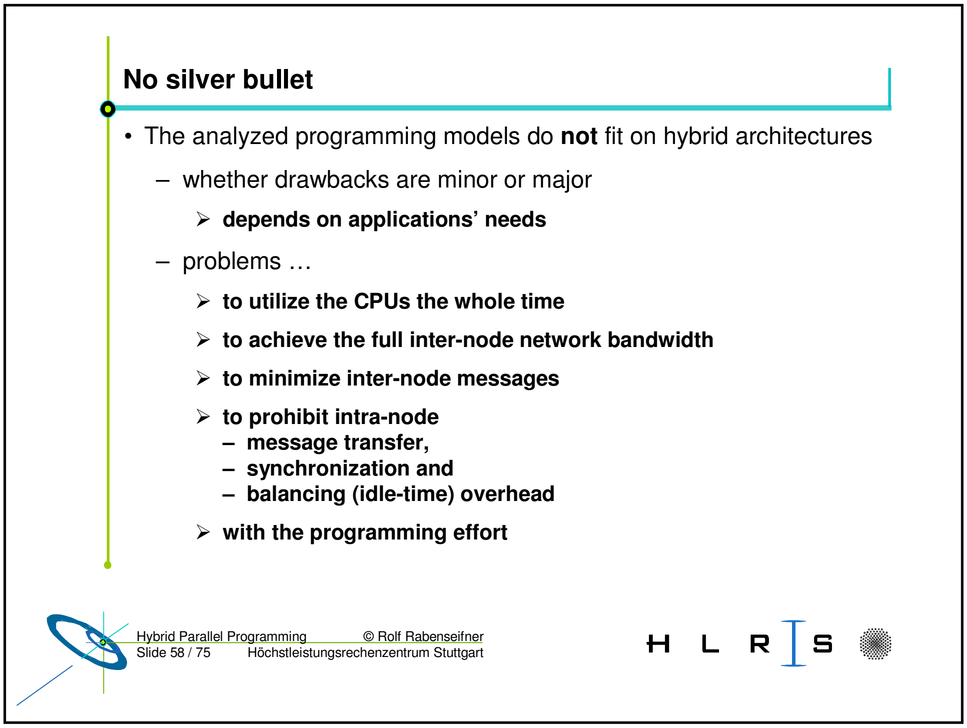
For the appropriate application a suitable tool!



Mismatch Problems

- **Topology problem** [with pure MPI]
- **Unnecessary intra-node communication** [with pure MPI]
- **Inter-node bandwidth problem** [with hybrid MPI+OpenMP]
- **Sleeping threads and** [with masteronly]
  saturation problem [with pure MPI]
- **Additional OpenMP overhead** [with hybrid MPI+OpenMP]
  - Thread startup / join
  - Cache flush (data source thread – communicating thread – sync. → flush)
- **Overlapping communication and computation** [with hybrid MPI+OpenMP]
  - an application problem → separation of local or halo-based code
  - a programming problem → thread-ranks-based vs. OpenMP work-sharing
  - a load balancing problem, if only some threads communicate / compute
- **Communication overhead with DSM** [with pure (Cluster) OpenMP]

➔ **no silver bullet**, **i.e., each parallelization scheme has its problems**



No silver bullet

- The analyzed programming models do **not** fit on hybrid architectures
  - whether drawbacks are minor or major
    - ➢ **depends on applications' needs**
  - problems …
    - ➢ **to utilize the CPUs the whole time**
    - ➢ **to achieve the full inter-node network bandwidth**
    - ➢ **to minimize inter-node messages**
    - ➢ **to prohibit intra-node**
      - **message transfer,**
      - **synchronization and**
      - **balancing (idle-time) overhead**
    - ➢ **with the programming effort**

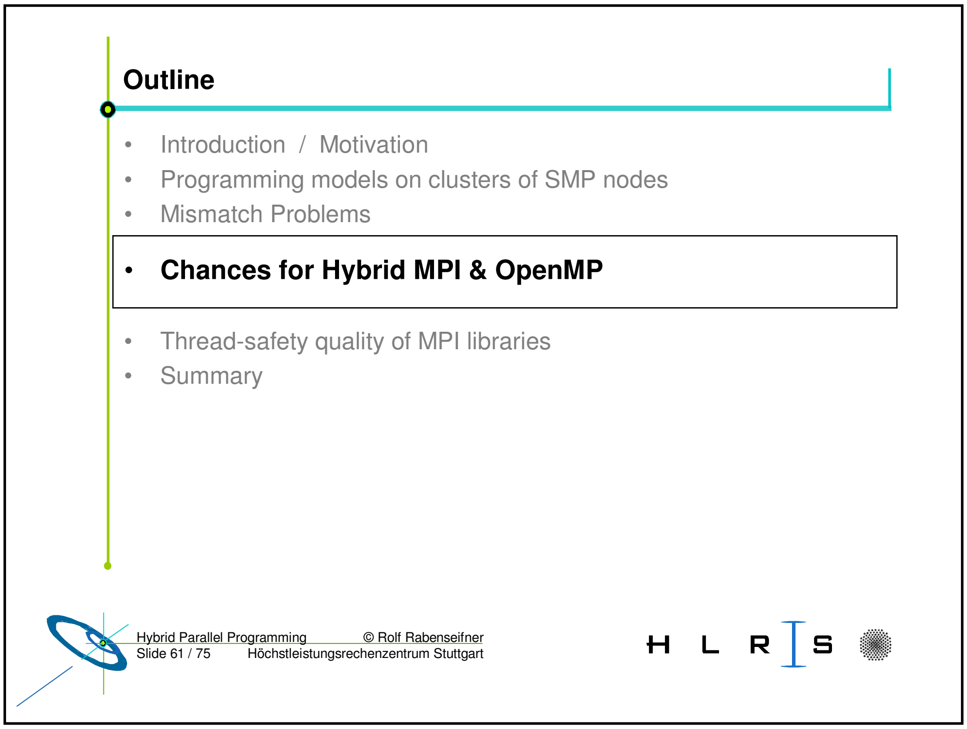
Chances for optimization

- with hybrid masteronly (**MPI only outside of parallel OpenMP regions**), e.g.,
  - **Minimize work of MPI routines, e.g.,**
    - application can copy non-contiguous data into contiguous scratch arrays (instead of using derived datatypes)
  - **MPI communication parallelized with multiple threads to saturate the inter-node network**
    - by internal parallel regions inside of the MPI library
    - by the user application
  - **Use only hardware that can saturate inter-node network with 1 thread**
  - **Optimal throughput:**
    - reuse of idling CPUs by other applications
- On constellations:
  - **Hybrid Masteronly with several MPI multi-threaded processes on each SMP node**

skipped

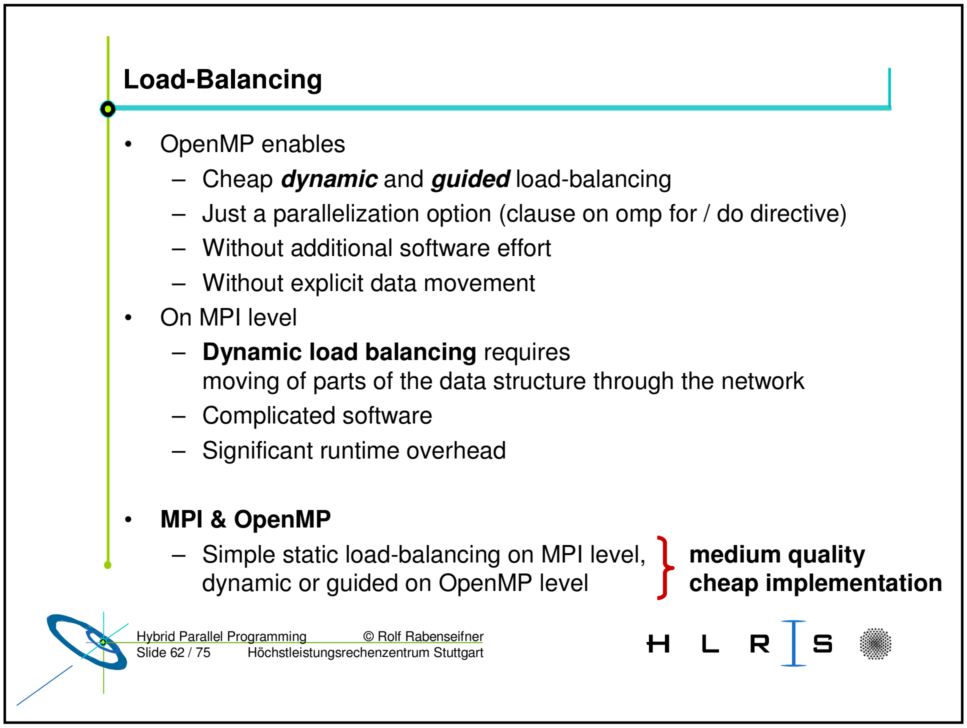
Summary of mismatch problems

Performance and Programming Problems with ...	**Pure MPI**	**Master-only** 1 process per node	**Master-only** several processes per node	**Over-lapping** 1 process per node	**Over-lapping** several processes per node	**Pure OpenMP:** e.g., Intel Cluster OpenMP
Application topology problem (neighbor domains inside of SMP node)	↯		↯		↯	↯
Additional MPI communication inside of SMP nodes	↯		↯		↯	
Do we achieve full inter-node bandwidth on constellations?		↯↯↯		↯		↯↯↯
Sleeping CPUs while MPI communication	(↯)	↯↯	↯			↯
Additional OpenMP overhead		↯	↯	↯	↯	
Separation of (a) halo data and (b) inner data based calculations				↯↯	↯↯	
OpenMP work sharing only partially usable				↯↯	↯↯	
Load balancing problem due to hybrid programming model				↯	↯	



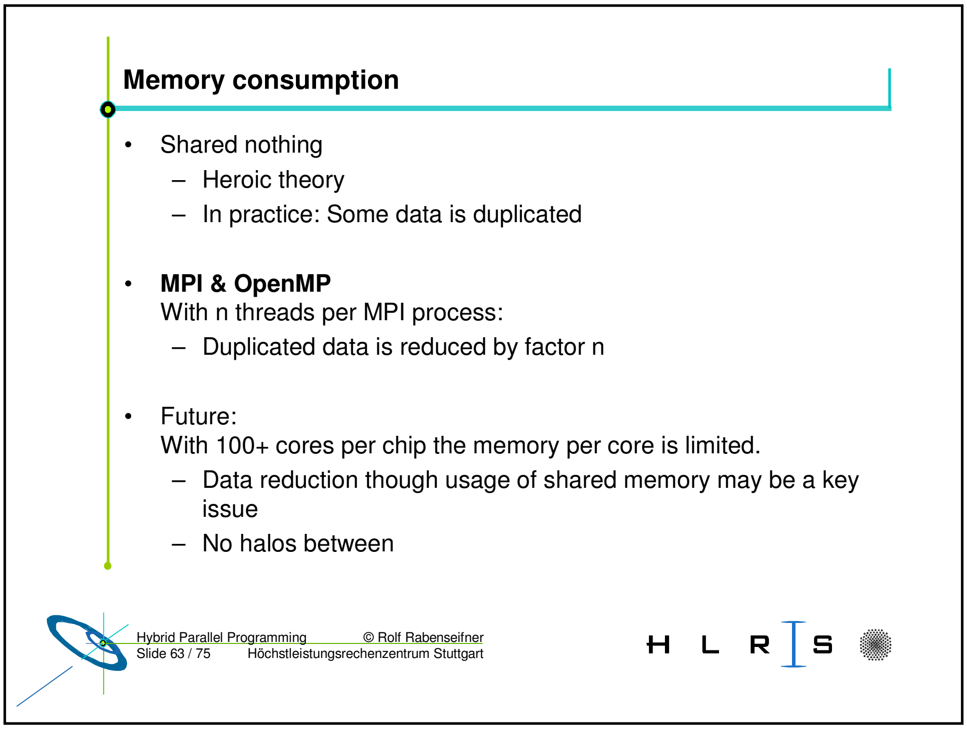
Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Mismatch Problems
- **Chances for Hybrid MPI & OpenMP**
- Thread-safety quality of MPI libraries
- Summary



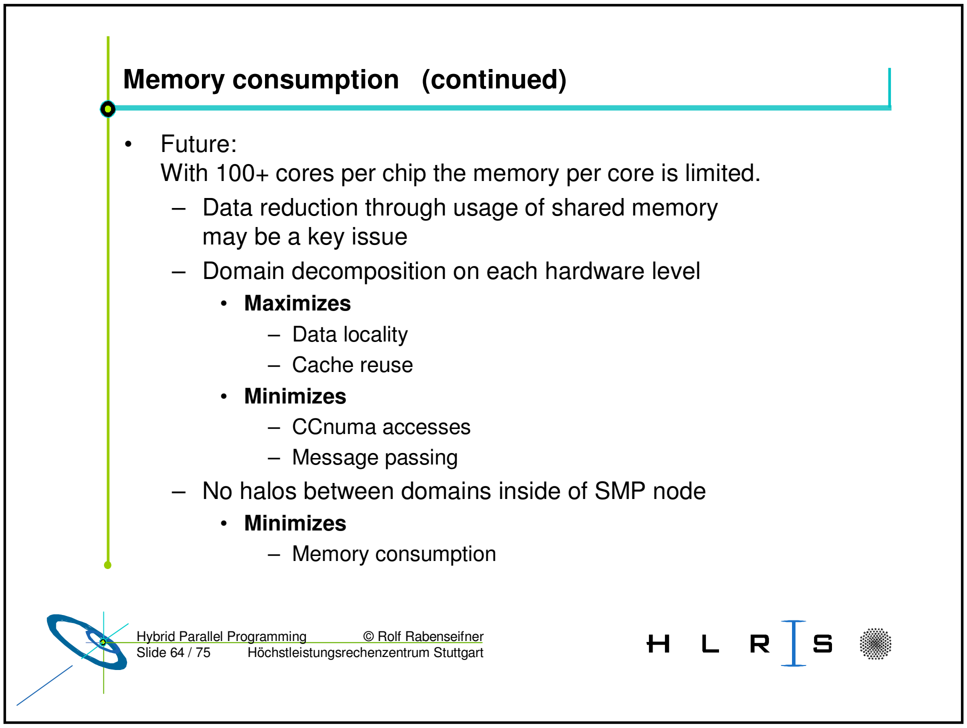
Load-Balancing

- OpenMP enables
  - Cheap ***dynamic*** and ***guided*** load-balancing
  - Just a parallelization option (clause on omp for / do directive)
  - Without additional software effort
  - Without explicit data movement
- On MPI level
  - **Dynamic load balancing** requires moving of parts of the data structure through the network
  - Complicated software
  - Significant runtime overhead
- **MPI & OpenMP**
  - Simple static load-balancing on MPI level, dynamic or guided on OpenMP level } **medium quality cheap implementation**



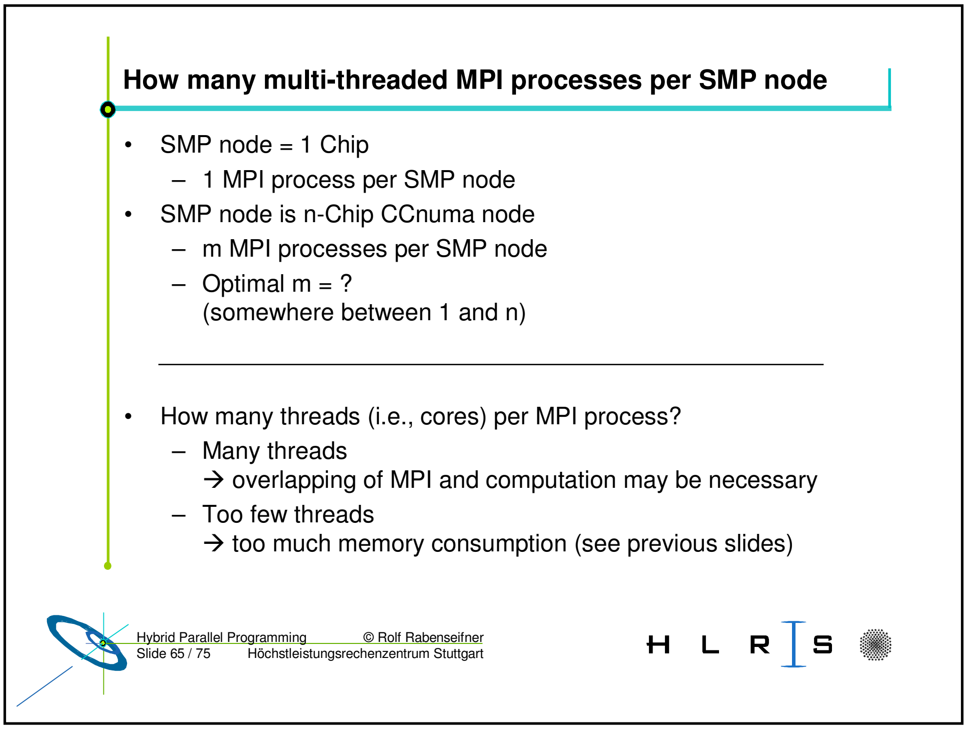
Memory consumption

- Shared nothing
  - Heroic theory
  - In practice: Some data is duplicated
- **MPI & OpenMP**
  With n threads per MPI process:
  - Duplicated data is reduced by factor n
- Future:
  With 100+ cores per chip the memory per core is limited.
  - Data reduction though usage of shared memory may be a key issue
  - No halos between



Memory consumption (continued)

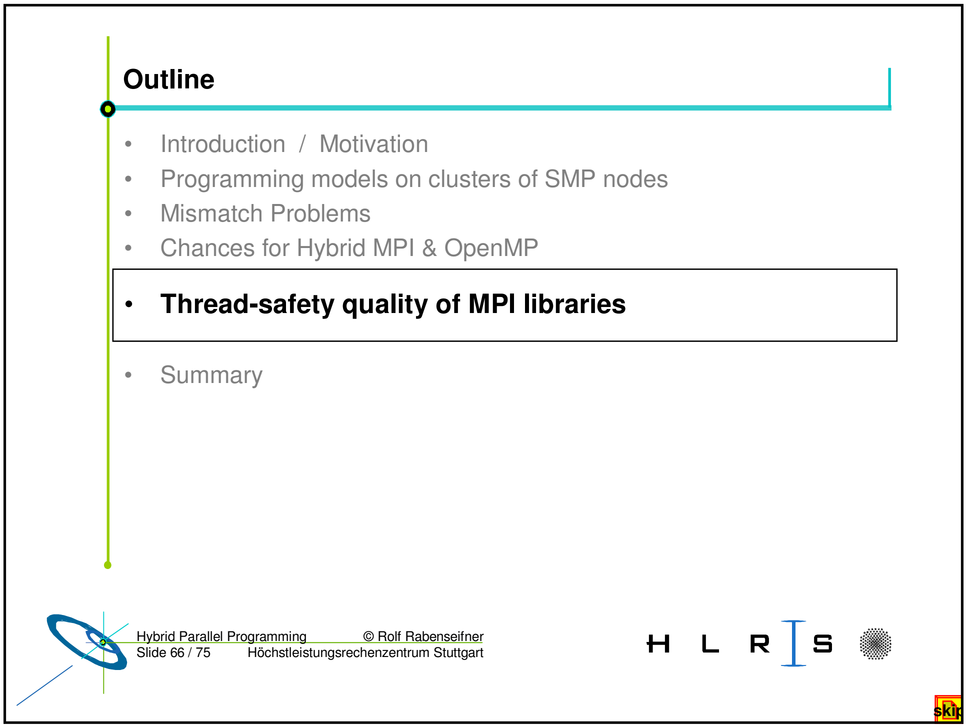
- Future:
  With 100+ cores per chip the memory per core is limited.
  - Data reduction through usage of shared memory may be a key issue
  - Domain decomposition on each hardware level
    - **Maximizes**
      - Data locality
      - Cache reuse
    - **Minimizes**
      - CCnuma accesses
      - Message passing
  - No halos between domains inside of SMP node
    - **Minimizes**
      - Memory consumption



How many multi-threaded MPI processes per SMP node

- SMP node = 1 Chip
  - 1 MPI process per SMP node
- SMP node is n-Chip CCnuma node
  - m MPI processes per SMP node
  - Optimal m = ?
    (somewhere between 1 and n)

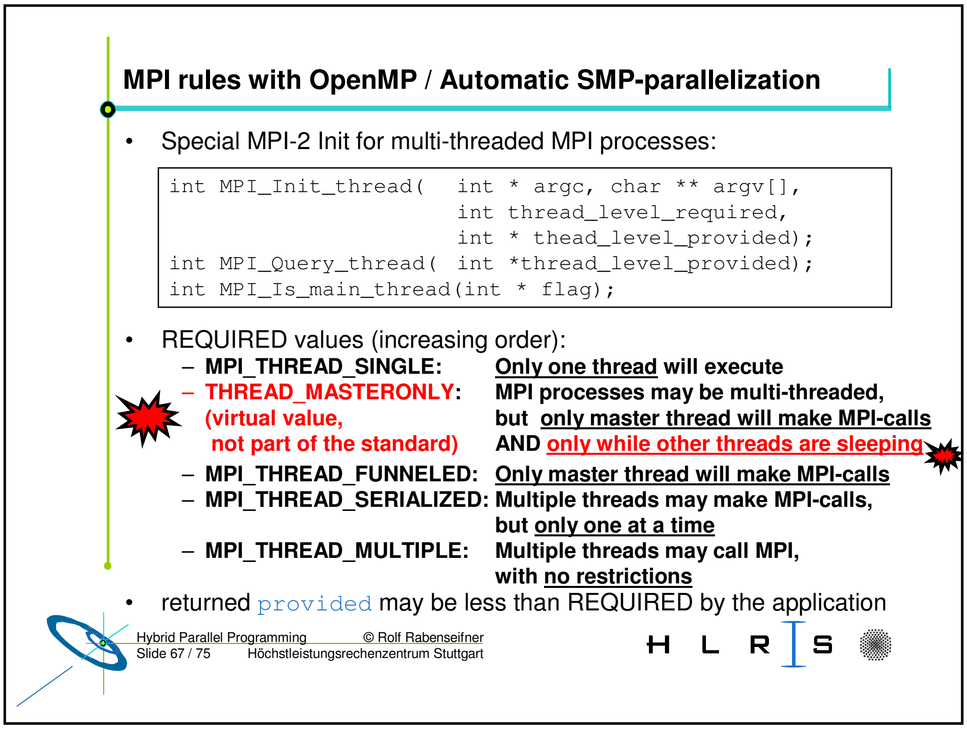
- How many threads (i.e., cores) per MPI process?
  - Many threads
    → overlapping of MPI and computation may be necessary
  - Too few threads
    → too much memory consumption (see previous slides)



Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Mismatch Problems
- Chances for Hybrid MPI & OpenMP
- **Thread-safety quality of MPI libraries**
- Summary

skip



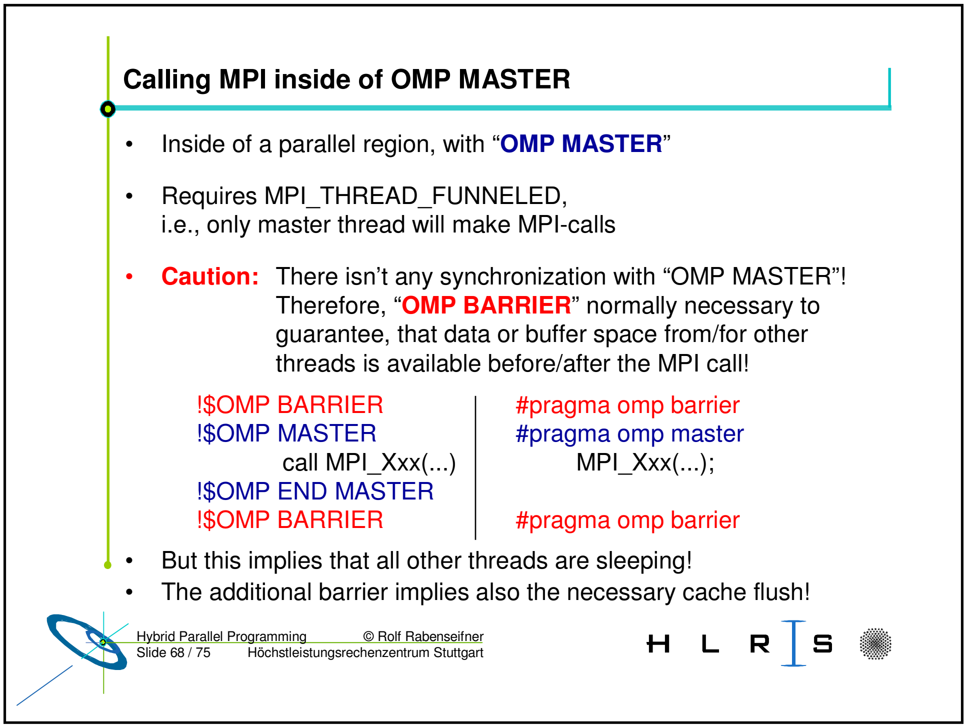
MPI rules with OpenMP / Automatic SMP-parallelization

- Special MPI-2 Init for multi-threaded MPI processes:

```
int MPI_Init_thread(   int * argc, char ** argv[],
                       int thread_level_required,
                       int * thead_level_provided);
int MPI_Query_thread(  int *thread_level_provided);
int MPI_Is_main_thread(int * flag);
```

- REQUIRED values (increasing order):
  - **MPI_THREAD_SINGLE:** **Only one thread will execute**
  - **THREAD_MASTERONLY: (virtual value, not part of the standard)** **MPI processes may be multi-threaded, but only master thread will make MPI-calls AND only while other threads are sleeping**
  - **MPI_THREAD_FUNNELED:** **Only master thread will make MPI-calls**
  - **MPI_THREAD_SERIALIZED:** **Multiple threads may make MPI-calls, but only one at a time**
  - **MPI_THREAD_MULTIPLE:** **Multiple threads may call MPI, with no restrictions**
- returned `provided` may be less than REQUIRED by the application

Hybrid Parallel Programming © Rolf Rabenseifner
Slide 67 / 75 Höchstleistungsrechenzentrum Stuttgart



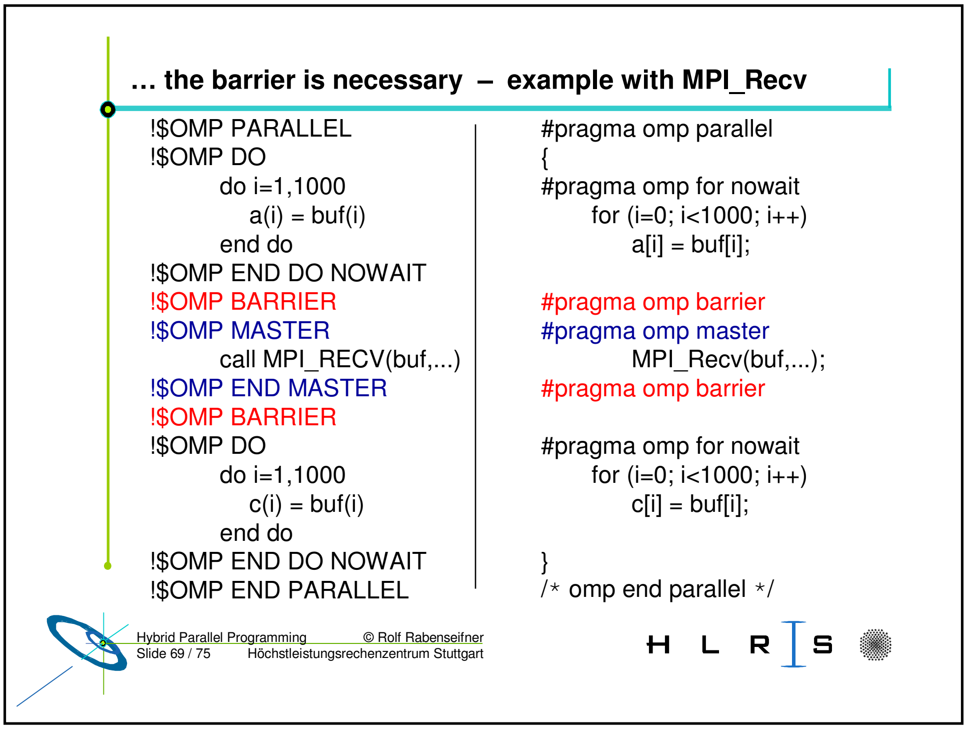
Calling MPI inside of OMP MASTER

- Inside of a parallel region, with "**OMP MASTER**"
- Requires MPI_THREAD_FUNNELED, i.e., only master thread will make MPI-calls
- **Caution:** There isn't any synchronization with "OMP MASTER"! Therefore, "**OMP BARRIER**" normally necessary to guarantee, that data or buffer space from/for other threads is available before/after the MPI call!

```
!$OMP BARRIER                 #pragma omp barrier
!$OMP MASTER                  #pragma omp master
        call MPI_Xxx(...)             MPI_Xxx(...);
!$OMP END MASTER
!$OMP BARRIER                 #pragma omp barrier
```

- But this implies that all other threads are sleeping!
- The additional barrier implies also the necessary cache flush!

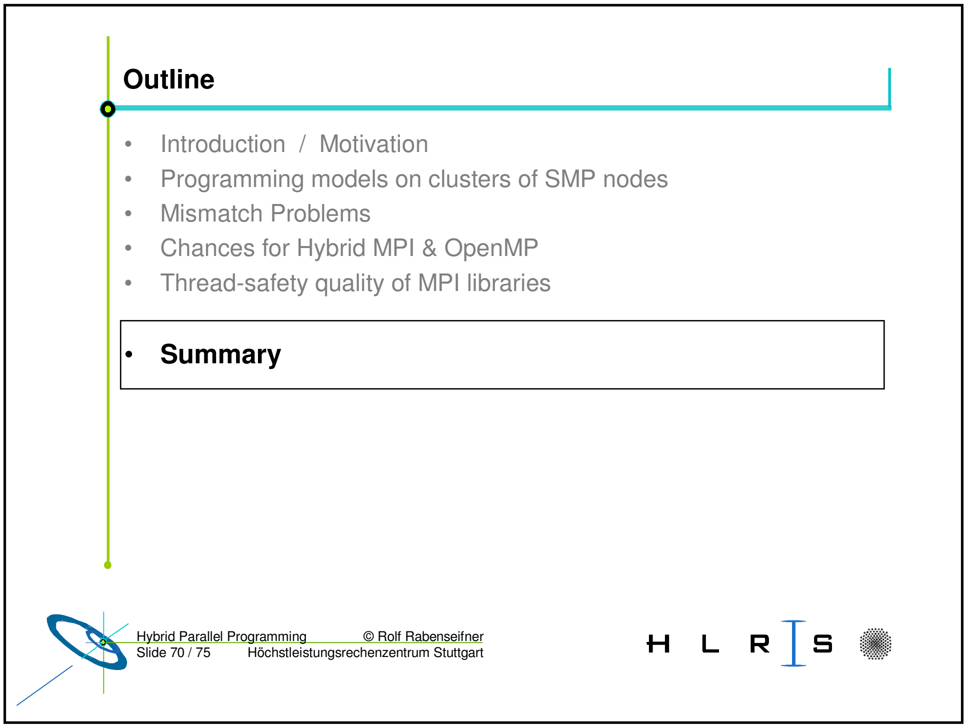
Hybrid Parallel Programming © Rolf Rabenseifner
Slide 68 / 75 Höchstleistungsrechenzentrum Stuttgart



… the barrier is necessary – example with MPI_Recv

```
!$OMP PARALLEL
!$OMP DO
        do i=1,1000
            a(i) = buf(i)
        end do
!$OMP END DO NOWAIT
!$OMP BARRIER
!$OMP MASTER
        call MPI_RECV(buf,...)
!$OMP END MASTER
!$OMP BARRIER
!$OMP DO
        do i=1,1000
            c(i) = buf(i)
        end do
!$OMP END DO NOWAIT
!$OMP END PARALLEL
```

```
#pragma omp parallel
{
#pragma omp for nowait
      for (i=0; i<1000; i++)
           a[i] = buf[i];

#pragma omp barrier
#pragma omp master
           MPI_Recv(buf,...);
#pragma omp barrier

#pragma omp for nowait
      for (i=0; i<1000; i++)
           c[i] = buf[i];

}
/* omp end parallel */
```



Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Mismatch Problems
- Chances for Hybrid MPI & OpenMP
- Thread-safety quality of MPI libraries
- **Summary**

Acknowledgements

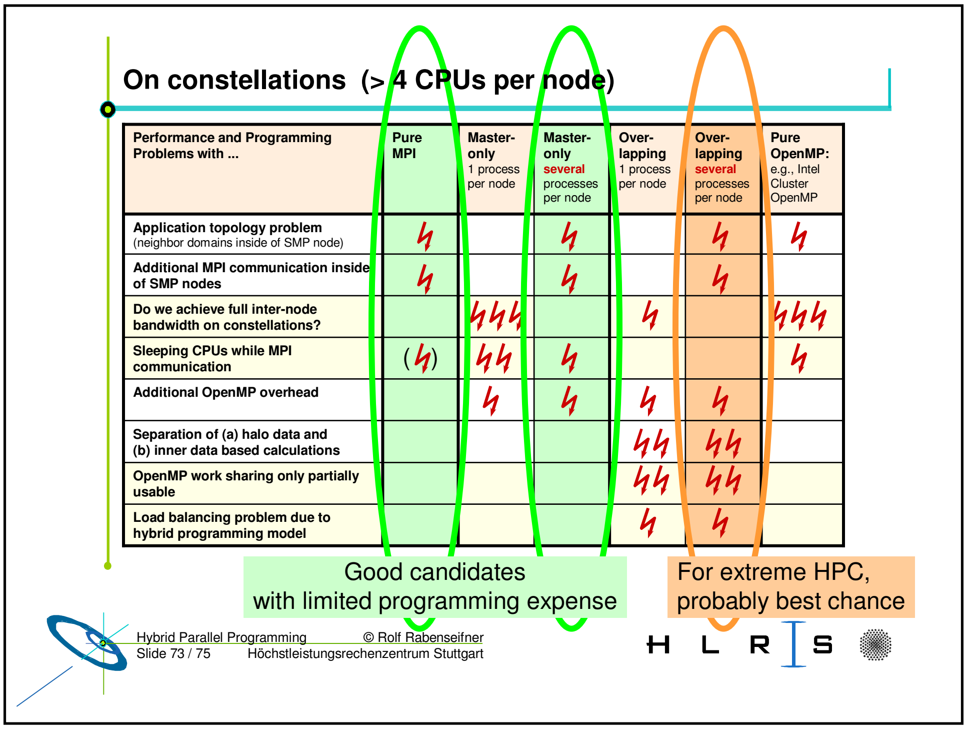
- I want to thank
  - Gerhard Wellein, RRZE
  - Monika Wierse, Wilfried Oed, and Tom Goozen, CRAY
  - Holger Berger, NEC
  - Reiner Vogelsang, SGI
  - Gabriele Jost, NASA
  - Dieter an Mey, RZ Aachen
  - Horst Simon, NERSC
  - Matthias Müller, HLRS
  - my colleges at HLRS

On clusters with small nodes (≤ 4 CPUs)

Performance and Programming Problems with ...	**Pure MPI**	**Master-only** 1 process per node	**Master-only** several processes per node	**Over-lapping** 1 process per node	**Over-lapping** several processes per node	**Pure OpenMP:** e.g., Intel Cluster OpenMP
Application topology problem (neighbor domains inside of SMP node)	↯		↯		↯	↯
Additional MPI communication inside of SMP nodes	↯		↯		↯	
~~**Do we achieve full inter-node bandwidth on constellations?**~~		~~↯↯↯~~		~~↯~~		↯↯↯
Sleeping CPUs while MPI communication	(↯)	(~~↯~~↯)	↯			↯
Additional OpenMP overhead		↯	↯	↯	↯	
Separation of (a) halo data and (b) inner data based calculations				↯↯	↯↯	
OpenMP work sharing only partially usable				↯↯	↯↯	
Load balancing problem due to hybrid programming model				↯	↯	

Row should not be relevant due to nodes with **≤ 4 CPUs**

Good candidates
with limited programming expense



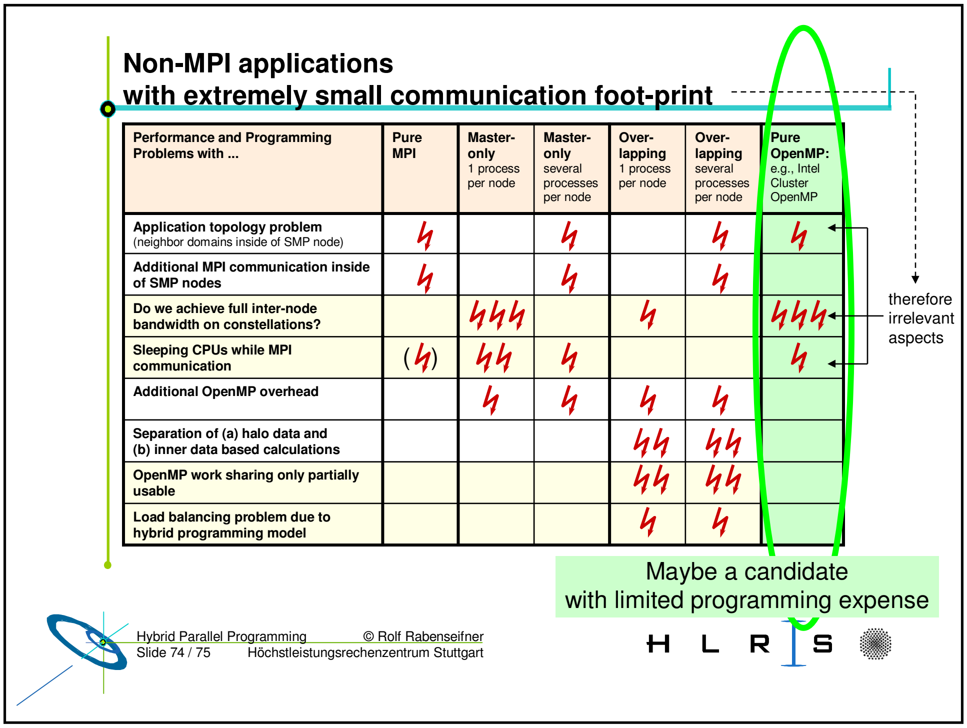
On constellations (> 4 CPUs per node)

Performance and Programming Problems with ...	Pure MPI	Master-only 1 process per node	Master-only several processes per node	Over-lapping 1 process per node	Over-lapping several processes per node	Pure OpenMP: e.g., Intel Cluster OpenMP
Application topology problem (neighbor domains inside of SMP node)	ϟ		ϟ		ϟ	ϟ
Additional MPI communication inside of SMP nodes	ϟ		ϟ		ϟ	
Do we achieve full inter-node bandwidth on constellations?		ϟϟϟ		ϟ		ϟϟϟ
Sleeping CPUs while MPI communication	(ϟ)	ϟϟ	ϟ			ϟ
Additional OpenMP overhead		ϟ	ϟ	ϟ	ϟ	
Separation of (a) halo data and (b) inner data based calculations				ϟϟ	ϟϟ	
OpenMP work sharing only partially usable				ϟϟ	ϟϟ	
Load balancing problem due to hybrid programming model				ϟ	ϟ	

Good candidates
with limited programming expense

For extreme HPC,
probably best chance

Hybrid Parallel Programming © Rolf Rabenseifner
Slide 73 / 75 Höchstleistungsrechenzentrum Stuttgart



Non-MPI applications with extremely small communication foot-print

Performance and Programming Problems with ...	Pure MPI	Master-only 1 process per node	Master-only several processes per node	Over-lapping 1 process per node	Over-lapping several processes per node	Pure OpenMP: e.g., Intel Cluster OpenMP
Application topology problem (neighbor domains inside of SMP node)	ϟ		ϟ		ϟ	ϟ
Additional MPI communication inside of SMP nodes	ϟ		ϟ		ϟ	
Do we achieve full inter-node bandwidth on constellations?		ϟϟϟ		ϟ		ϟϟϟ
Sleeping CPUs while MPI communication	(ϟ)	ϟϟ	ϟ			ϟ
Additional OpenMP overhead		ϟ	ϟ	ϟ	ϟ	
Separation of (a) halo data and (b) inner data based calculations				ϟϟ	ϟϟ	
OpenMP work sharing only partially usable				ϟϟ	ϟϟ	
Load balancing problem due to hybrid programming model				ϟ	ϟ	

therefore irrelevant aspects

Maybe a candidate
with limited programming expense

Hybrid Parallel Programming © Rolf Rabenseifner
Slide 74 / 75 Höchstleistungsrechenzentrum Stuttgart

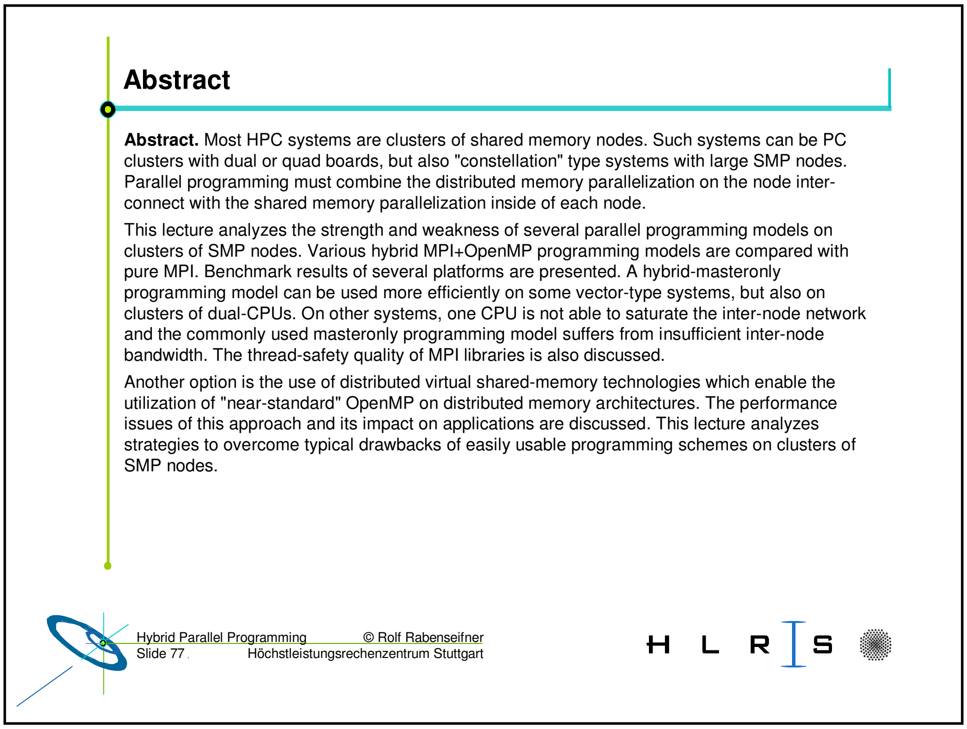
Conclusions

- **Constellations** (>4 CPUs per SMP node)**:**
  - **Most platforms**
    - masteronly style cannot saturate inter-node bandwidth
    - **Several multi-threaded MPI processes** per SMP node may help
- **Clusters with small SMP nodes:**
  - **Simple masteronly style** is a good candidate
  - although some CPU idle **(while one is communicating)**
- **DSM systems** (pure OpenMP, e.g Intel Cluster OpenMP)**:**
  - may help for **some** (**but only some**) applications
- **Optimal performance:**
  - overlapping of communication & computation → extreme programming effort
- **Pure MPI:**
  - often the cheapest and (nearly) best solution
- **MPI & OpenMP:**
  - Necessary, if load-balancing and memory consumption issues must be solved

See also www.hlrs.de/people/rabenseifner → list of publications → Teaching in Germany

Appendix

- Abstract
- Intel® Compilers with Cluster OpenMP – Consistency Protocol – Examples
- Author
- References (with direct relation to the content of this tutorial)
- Further references

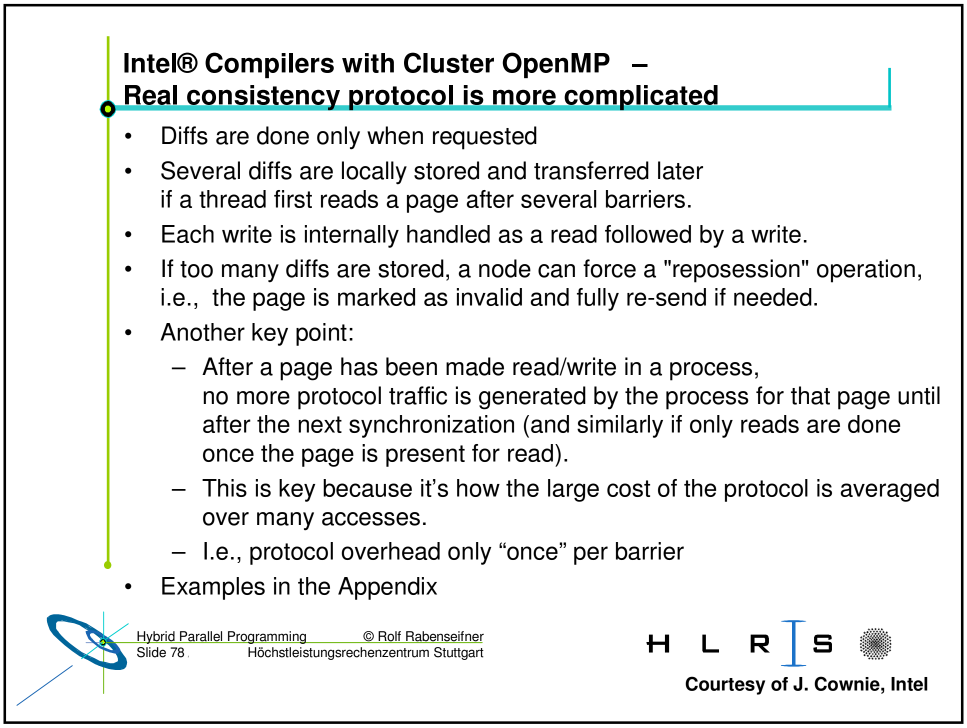


Abstract

Abstract. Most HPC systems are clusters of shared memory nodes. Such systems can be PC clusters with dual or quad boards, but also "constellation" type systems with large SMP nodes. Parallel programming must combine the distributed memory parallelization on the node inter-connect with the shared memory parallelization inside of each node.

This lecture analyzes the strength and weakness of several parallel programming models on clusters of SMP nodes. Various hybrid MPI+OpenMP programming models are compared with pure MPI. Benchmark results of several platforms are presented. A hybrid-masteronly programming model can be used more efficiently on some vector-type systems, but also on clusters of dual-CPUs. On other systems, one CPU is not able to saturate the inter-node network and the commonly used masteronly programming model suffers from insufficient inter-node bandwidth. The thread-safety quality of MPI libraries is also discussed.

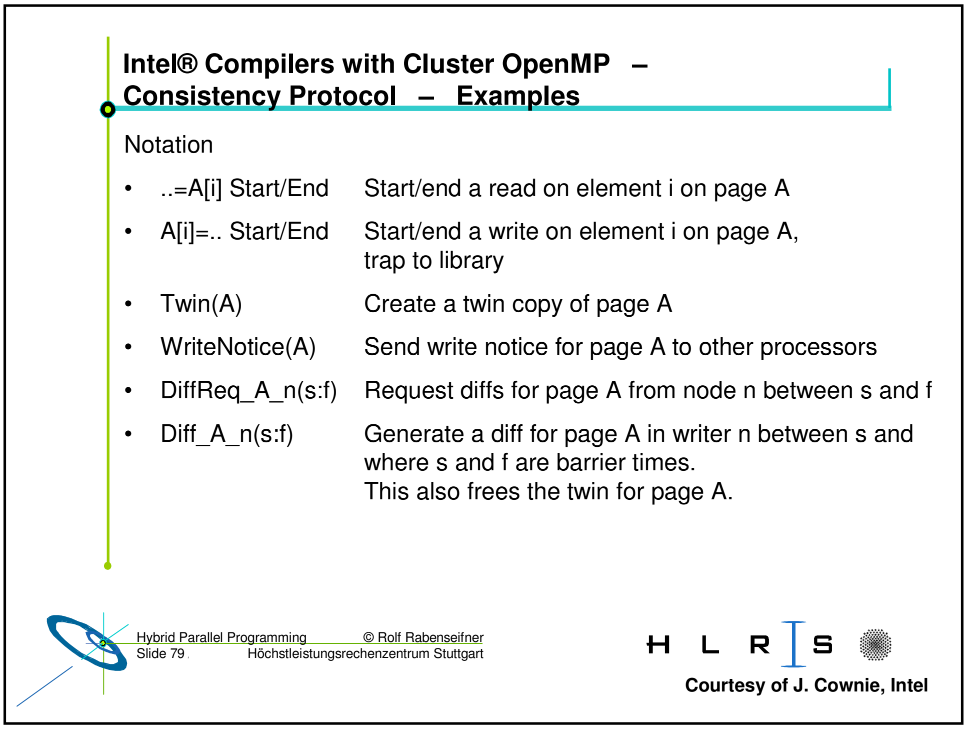
Another option is the use of distributed virtual shared-memory technologies which enable the utilization of "near-standard" OpenMP on distributed memory architectures. The performance issues of this approach and its impact on applications are discussed. This lecture analyzes strategies to overcome typical drawbacks of easily usable programming schemes on clusters of SMP nodes.



Intel® Compilers with Cluster OpenMP – Real consistency protocol is more complicated

- Diffs are done only when requested
- Several diffs are locally stored and transferred later if a thread first reads a page after several barriers.
- Each write is internally handled as a read followed by a write.
- If too many diffs are stored, a node can force a "reposession" operation, i.e., the page is marked as invalid and fully re-send if needed.
- Another key point:
  - After a page has been made read/write in a process, no more protocol traffic is generated by the process for that page until after the next synchronization (and similarly if only reads are done once the page is present for read).
  - This is key because it's how the large cost of the protocol is averaged over many accesses.
  - I.e., protocol overhead only "once" per barrier
- Examples in the Appendix

Courtesy of J. Cownie, Intel

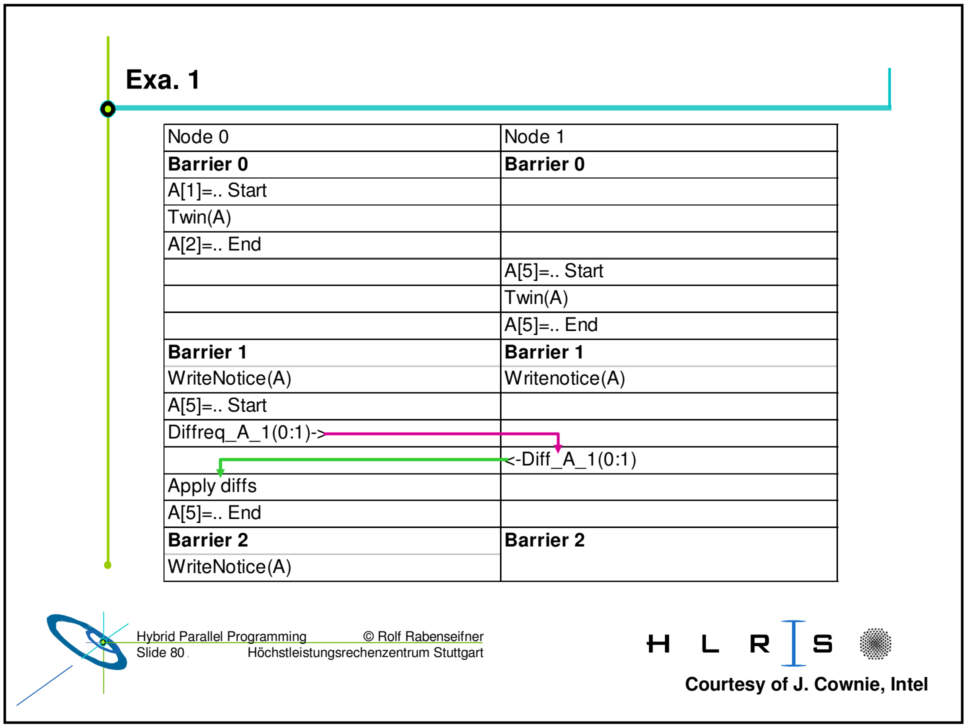


Intel® Compilers with Cluster OpenMP – Consistency Protocol – Examples

Notation

..=A[i] Start/End	Start/end a read on element i on page A
A[i]=.. Start/End	Start/end a write on element i on page A, trap to library
Twin(A)	Create a twin copy of page A
WriteNotice(A)	Send write notice for page A to other processors
DiffReq_A_n(s:f)	Request diffs for page A from node n between s and f
Diff_A_n(s:f)	Generate a diff for page A in writer n between s and where s and f are barrier times. This also frees the twin for page A.

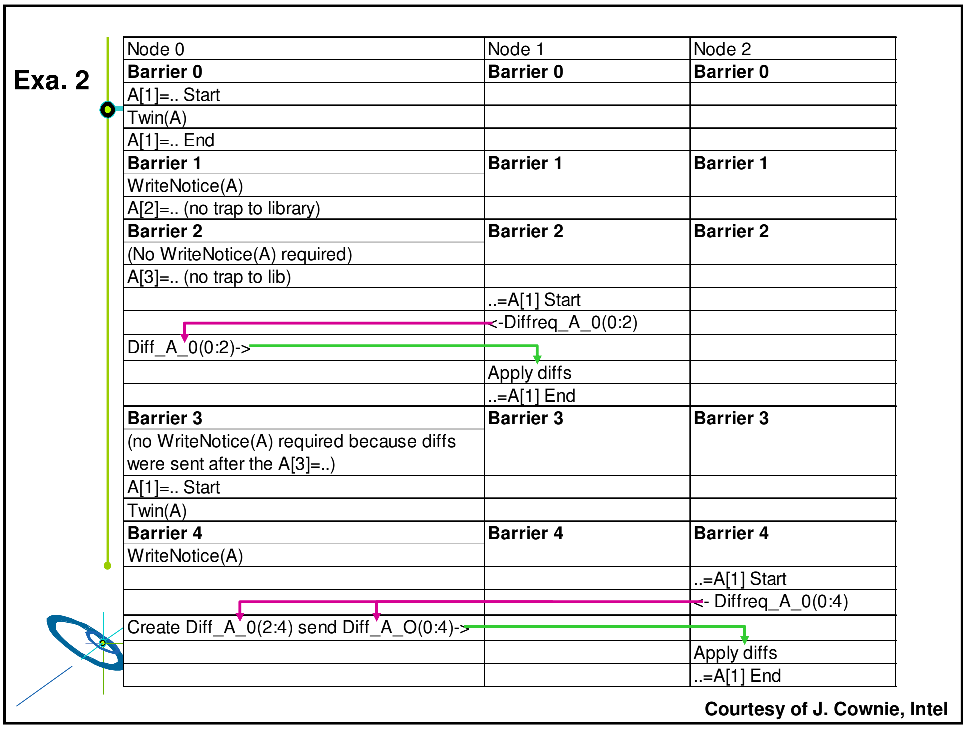
Courtesy of J. Cownie, Intel



Exa. 1

Node 0	Node 1
Barrier 0	**Barrier 0**
A[1]=.. Start	
Twin(A)	
A[2]=.. End	
	A[5]=.. Start
	Twin(A)
	A[5]=.. End
Barrier 1	**Barrier 1**
WriteNotice(A)	Writenotice(A)
A[5]=.. Start	
Diffreq_A_1(0:1)->	
	<-Diff_A_1(0:1)
Apply diffs	
A[5]=.. End	
Barrier 2	**Barrier 2**
WriteNotice(A)	

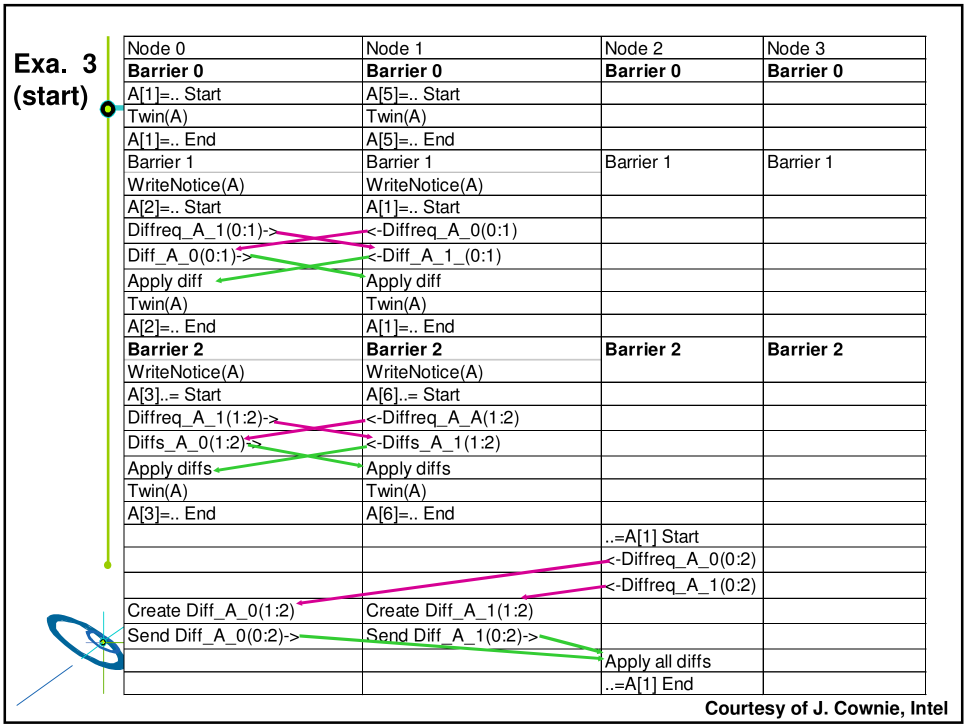
Courtesy of J. Cownie, Intel



Exa. 2

Node 0	Node 1	Node 2
Barrier 0	**Barrier 0**	**Barrier 0**
A[1]=.. Start		
Twin(A)		
A[1]=.. End		
Barrier 1	**Barrier 1**	**Barrier 1**
WriteNotice(A)		
A[2]=.. (no trap to library)		
Barrier 2	**Barrier 2**	**Barrier 2**
(No WriteNotice(A) required)		
A[3]=.. (no trap to lib)		
	..=A[1] Start	
	<-Diffreq_A_0(0:2)	
Diff_A_0(0:2)->		
	Apply diffs	
	..=A[1] End	
Barrier 3	**Barrier 3**	**Barrier 3**
(no WriteNotice(A) required because diffs were sent after the A[3]=..)		
A[1]=.. Start		
Twin(A)		
Barrier 4	**Barrier 4**	**Barrier 4**
WriteNotice(A)		
		..=A[1] Start
		<- Diffreq_A_0(0:4)
Create Diff_A_0(2:4) send Diff_A_O(0:4)->		
		Apply diffs
		..=A[1] End

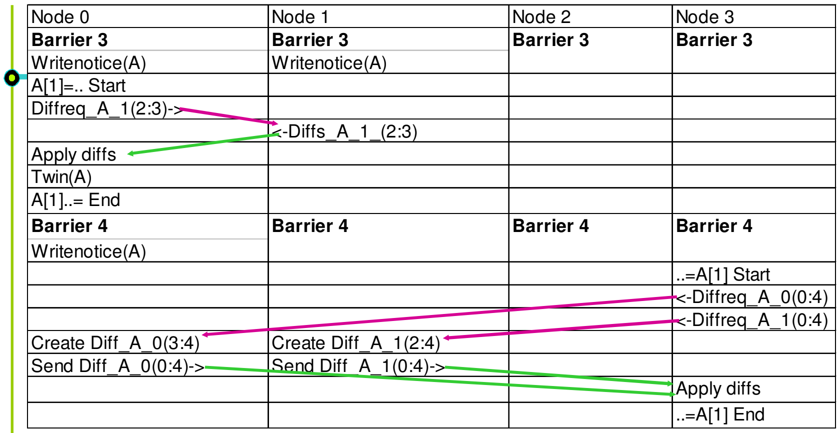
Courtesy of J. Cownie, Intel



Exa. 3 (start)

Node 0	Node 1	Node 2	Node 3
Barrier 0	**Barrier 0**	**Barrier 0**	**Barrier 0**
A[1]=.. Start	A[5]=.. Start		
Twin(A)	Twin(A)		
A[1]=.. End	A[5]=.. End		
Barrier 1	Barrier 1	Barrier 1	Barrier 1
WriteNotice(A)	WriteNotice(A)		
A[2]=.. Start	A[1]=.. Start		
Diffreq_A_1(0:1)->	<-Diffreq_A_0(0:1)		
Diff_A_0(0:1)->	<-Diff_A_1_(0:1)		
Apply diff	Apply diff		
Twin(A)	Twin(A)		
A[2]=.. End	A[1]=.. End		
Barrier 2	**Barrier 2**	**Barrier 2**	**Barrier 2**
WriteNotice(A)	WriteNotice(A)		
A[3]..= Start	A[6]..= Start		
Diffreq_A_1(1:2)->	<-Diffreq_A_A(1:2)		
Diffs_A_0(1:2)->	<-Diffs_A_1(1:2)		
Apply diffs	Apply diffs		
Twin(A)	Twin(A)		
A[3]=.. End	A[6]=.. End		
		..=A[1] Start	
		<-Diffreq_A_0(0:2)	
		<-Diffreq_A_1(0:2)	
Create Diff_A_0(1:2)	Create Diff_A_1(1:2)		
Send Diff_A_0(0:2)->	Send Diff_A_1(0:2)->		
		Apply all diffs	
		..=A[1] End	

Courtesy of J. Cownie, Intel



Node 0	Node 1	Node 2	Node 3
Barrier 3	**Barrier 3**	**Barrier 3**	**Barrier 3**
Writenotice(A)	Writenotice(A)		
A[1]=.. Start			
Diffreq_A_1(2:3)->			
	<-Diffs_A_1_(2:3)		
Apply diffs			
Twin(A)			
A[1]..= End			
Barrier 4	**Barrier 4**	**Barrier 4**	**Barrier 4**
Writenotice(A)			
			..=A[1] Start
			<-Diffreq_A_0(0:4)
			<-Diffreq_A_1(0:4)
Create Diff_A_0(3:4)	Create Diff_A_1(2:4)		
Send Diff_A_0(0:4)->	Send Diff_A_1(0:4)->		
			Apply diffs
			..=A[1] End

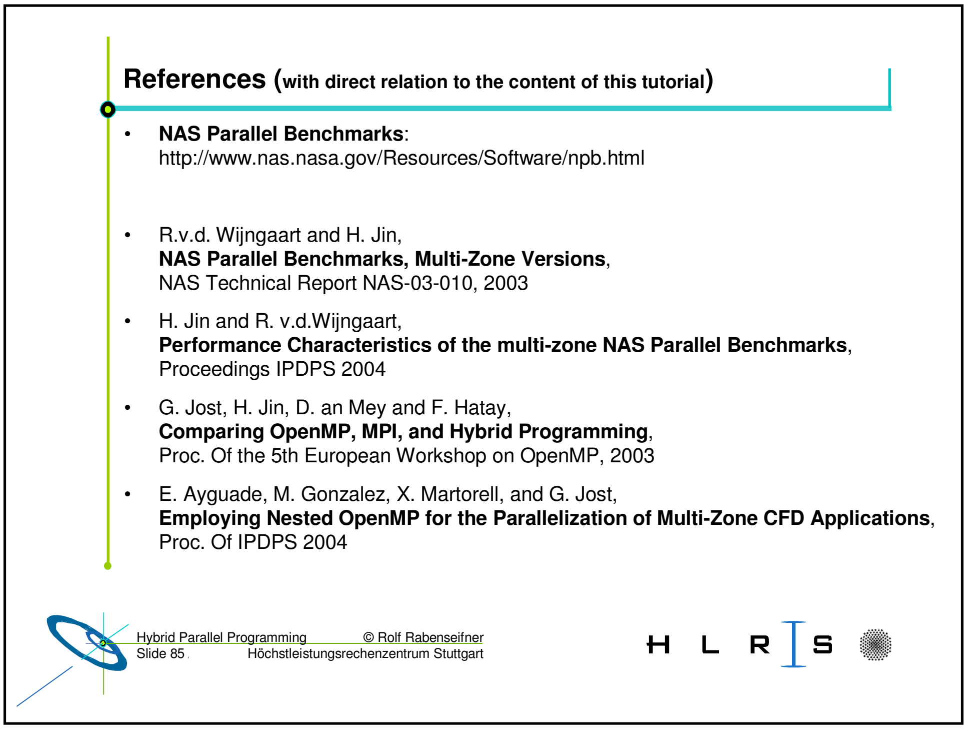
These examples may give an impression of the overhead induced by the Cluster OpenMP consistency protocol.

Courtesy of J. Cownie, Intel

Rolf Rabenseifner

Dr. Rolf Rabenseifner studied mathematics and physics at the University of Stuttgart. Since 1984, he has worked at the High-Performance Computing-Center Stuttgart (HLRS). He led the projects DFN-RPC, a remote procedure call tool, and MPI-GLUE, the first metacomputing MPI combining different vendor's MPIs without loosing the full MPI interface. In his dissertation, he developed a controlled logical clock as global time for trace-based profiling of parallel and distributed applications. Since 1996, he has been a member of the MPI-2 Forum. From January to April 1999, he was an invited researcher at the Center for High-Performance Computing at Dresden University of Technology.

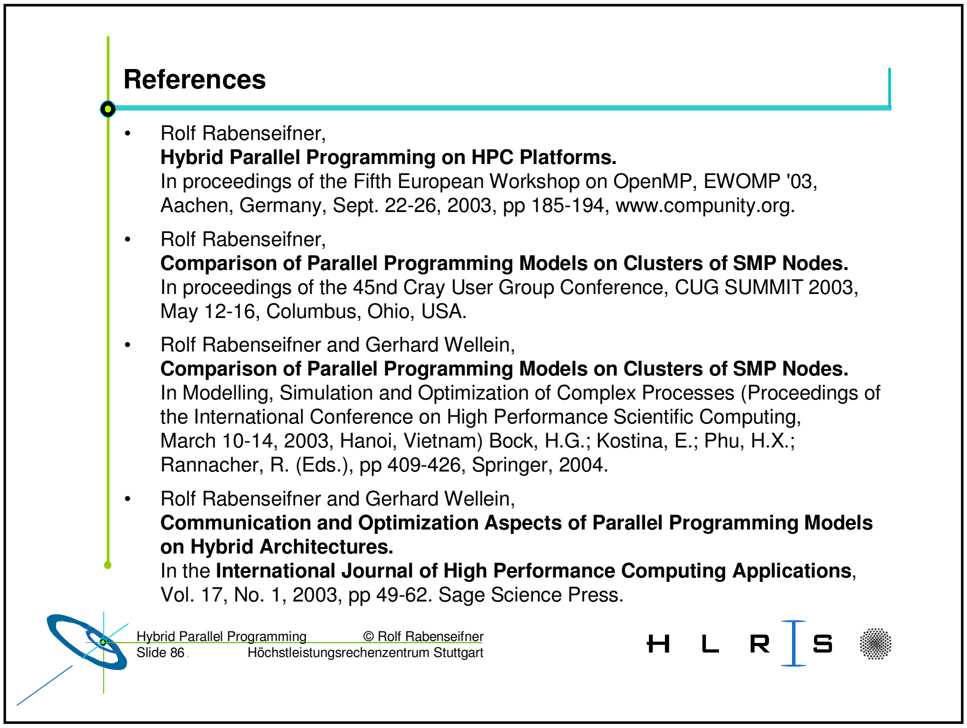
Currently, he is head of Parallel Computing - Training and Application Services at HLRS. He is involved in MPI profiling and benchmarking, e.g., in the HPC Challenge Benchmark Suite. In recent projects, he studied parallel I/O, parallel programming models for clusters of SMP nodes, and optimization of MPI collective routines. In workshops and summer schools, he teaches parallel programming models in many universities and labs in Germany.



References (with direct relation to the content of this tutorial)

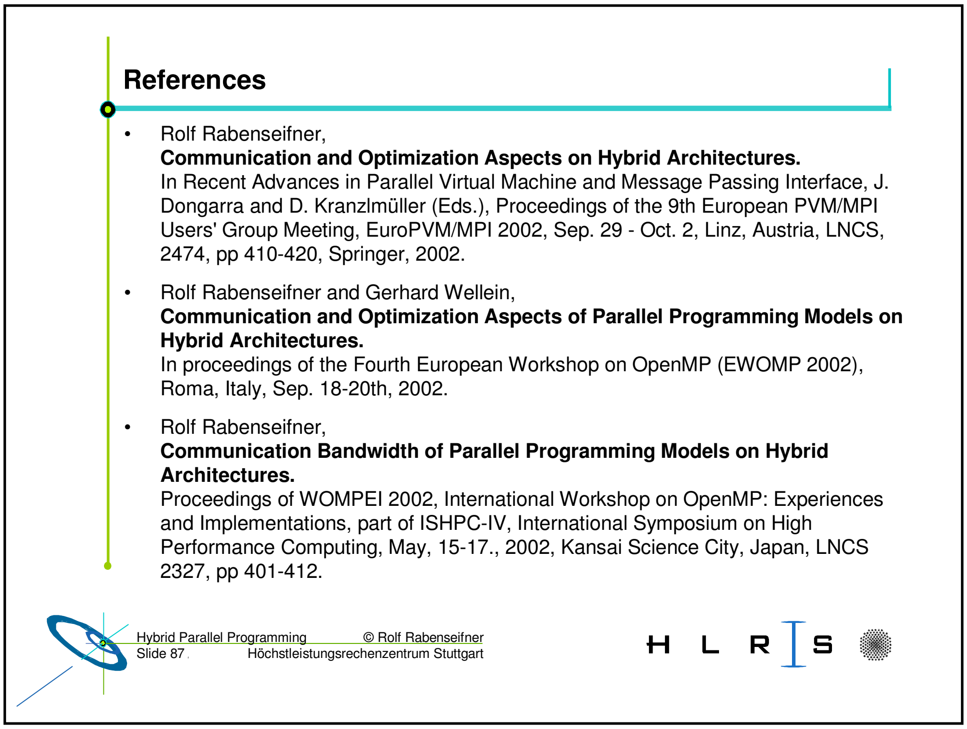
- **NAS Parallel Benchmarks**:
  http://www.nas.nasa.gov/Resources/Software/npb.html

- R.v.d. Wijngaart and H. Jin,
  NAS Parallel Benchmarks, Multi-Zone Versions,
  NAS Technical Report NAS-03-010, 2003
- H. Jin and R. v.d.Wijngaart,
  Performance Characteristics of the multi-zone NAS Parallel Benchmarks,
  Proceedings IPDPS 2004
- G. Jost, H. Jin, D. an Mey and F. Hatay,
  Comparing OpenMP, MPI, and Hybrid Programming,
  Proc. Of the 5th European Workshop on OpenMP, 2003
- E. Ayguade, M. Gonzalez, X. Martorell, and G. Jost,
  Employing Nested OpenMP for the Parallelization of Multi-Zone CFD Applications,
  Proc. Of IPDPS 2004



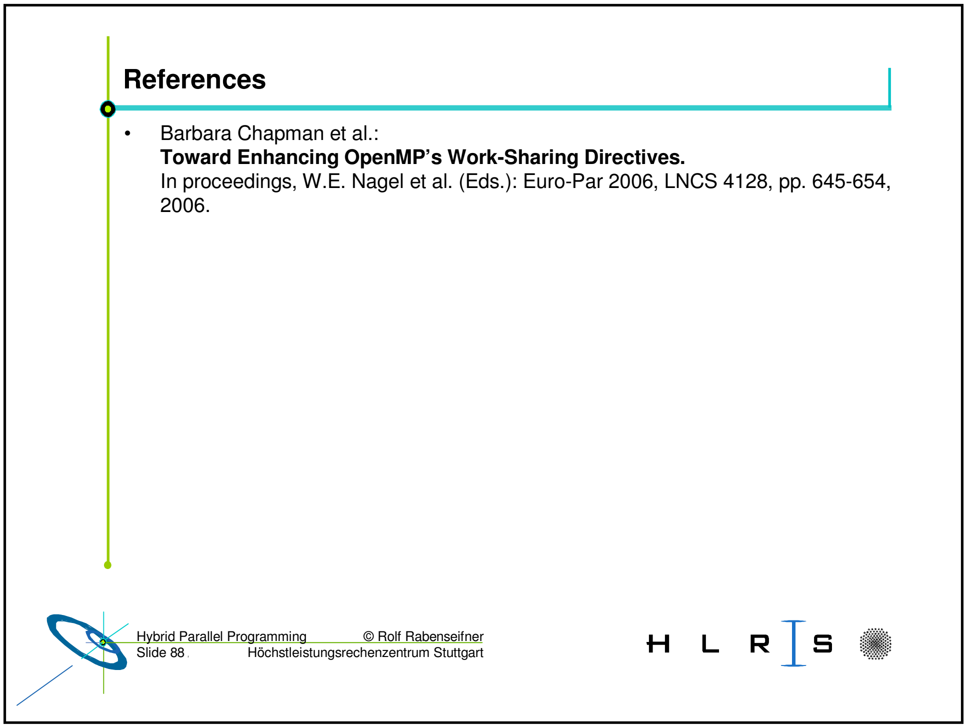
References

- Rolf Rabenseifner,
  Hybrid Parallel Programming on HPC Platforms.
  In proceedings of the Fifth European Workshop on OpenMP, EWOMP '03, Aachen, Germany, Sept. 22-26, 2003, pp 185-194, www.compunity.org.
- Rolf Rabenseifner,
  Comparison of Parallel Programming Models on Clusters of SMP Nodes.
  In proceedings of the 45nd Cray User Group Conference, CUG SUMMIT 2003, May 12-16, Columbus, Ohio, USA.
- Rolf Rabenseifner and Gerhard Wellein,
  Comparison of Parallel Programming Models on Clusters of SMP Nodes.
  In Modelling, Simulation and Optimization of Complex Processes (Proceedings of the International Conference on High Performance Scientific Computing, March 10-14, 2003, Hanoi, Vietnam) Bock, H.G.; Kostina, E.; Phu, H.X.; Rannacher, R. (Eds.), pp 409-426, Springer, 2004.
- Rolf Rabenseifner and Gerhard Wellein,
  Communication and Optimization Aspects of Parallel Programming Models on Hybrid Architectures.
  In the **International Journal of High Performance Computing Applications**, Vol. 17, No. 1, 2003, pp 49-62. Sage Science Press.



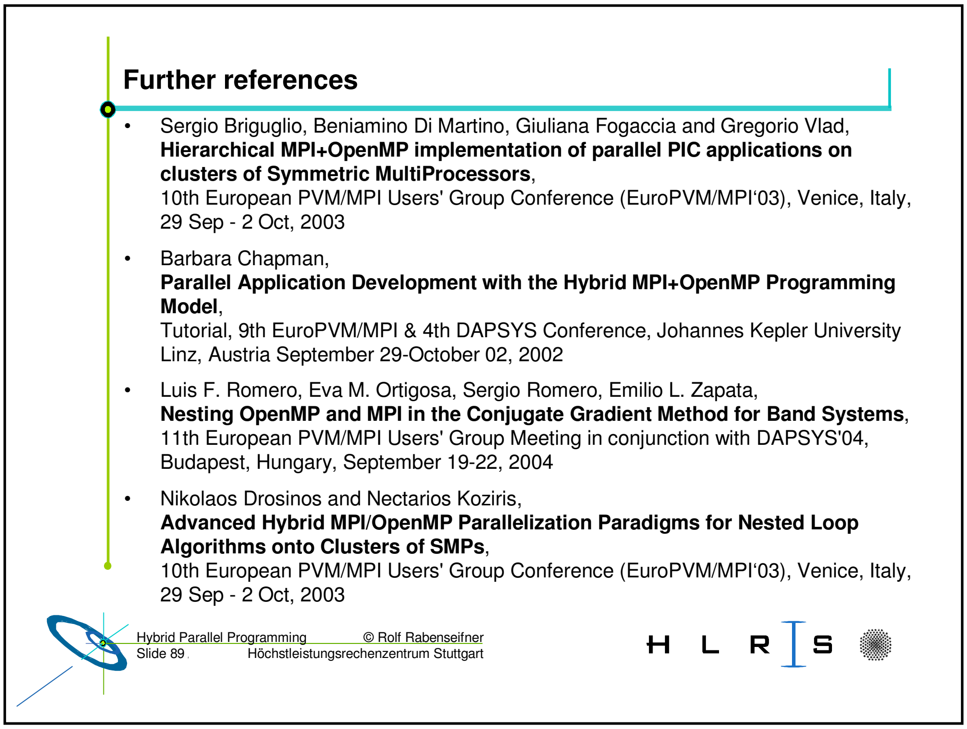
References

- Rolf Rabenseifner,
  Communication and Optimization Aspects on Hybrid Architectures.
  In Recent Advances in Parallel Virtual Machine and Message Passing Interface, J. Dongarra and D. Kranzlmüller (Eds.), Proceedings of the 9th European PVM/MPI Users' Group Meeting, EuroPVM/MPI 2002, Sep. 29 - Oct. 2, Linz, Austria, LNCS, 2474, pp 410-420, Springer, 2002.
- Rolf Rabenseifner and Gerhard Wellein,
  Communication and Optimization Aspects of Parallel Programming Models on Hybrid Architectures.
  In proceedings of the Fourth European Workshop on OpenMP (EWOMP 2002), Roma, Italy, Sep. 18-20th, 2002.
- Rolf Rabenseifner,
  Communication Bandwidth of Parallel Programming Models on Hybrid Architectures.
  Proceedings of WOMPEI 2002, International Workshop on OpenMP: Experiences and Implementations, part of ISHPC-IV, International Symposium on High Performance Computing, May, 15-17., 2002, Kansai Science City, Japan, LNCS 2327, pp 401-412.



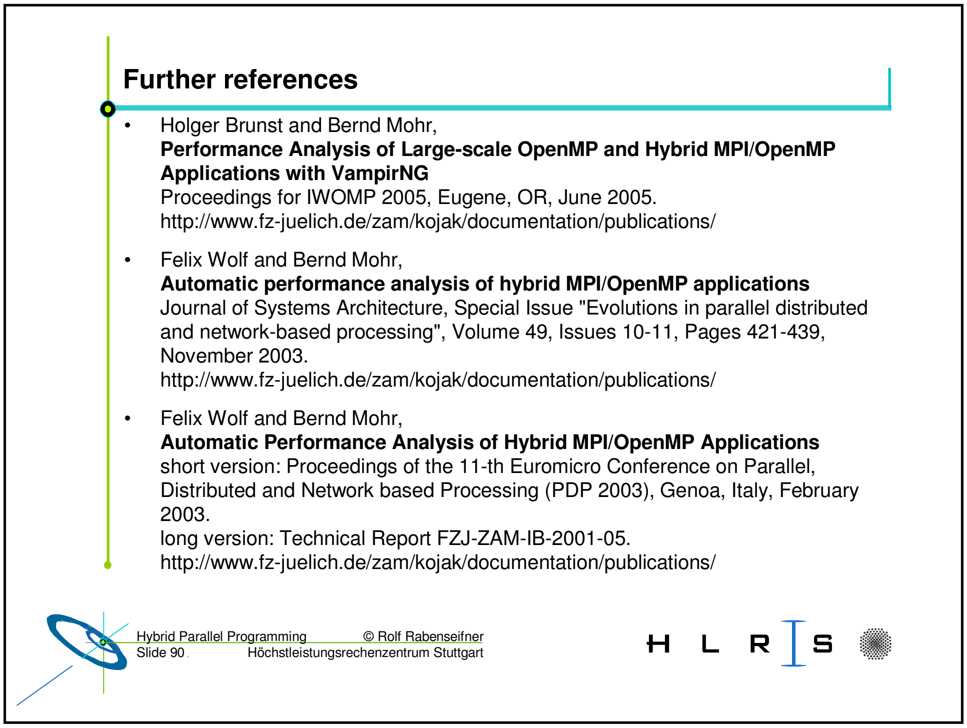
References

- Barbara Chapman et al.:
  Toward Enhancing OpenMP's Work-Sharing Directives.
  In proceedings, W.E. Nagel et al. (Eds.): Euro-Par 2006, LNCS 4128, pp. 645-654, 2006.



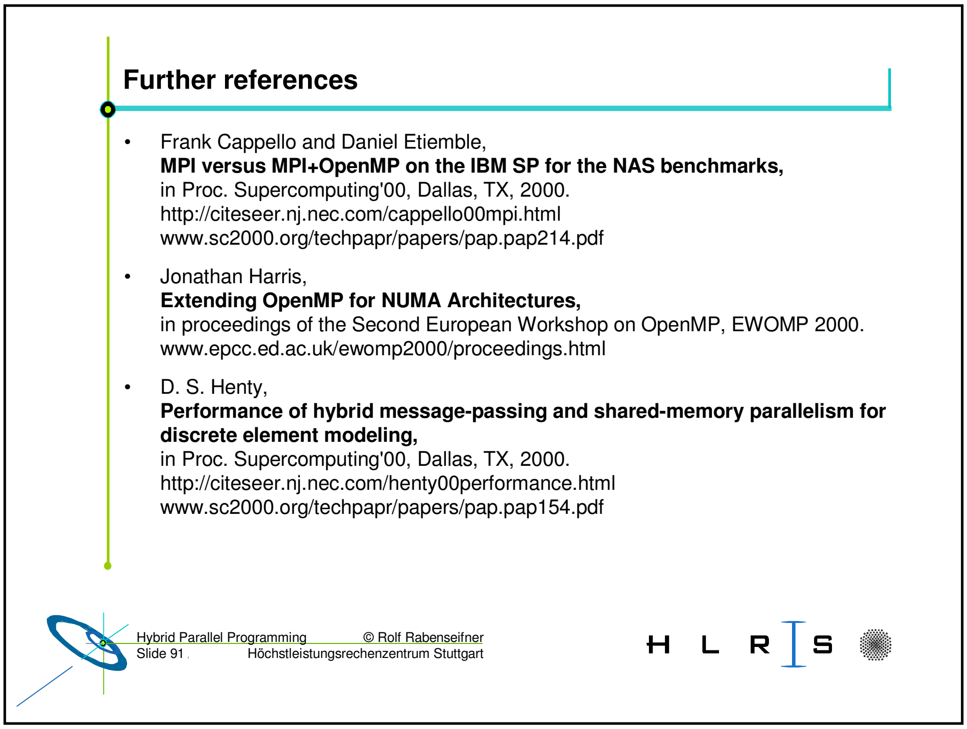
Further references

- Sergio Briguglio, Beniamino Di Martino, Giuliana Fogaccia and Gregorio Vlad,
  Hierarchical MPI+OpenMP implementation of parallel PIC applications on clusters of Symmetric MultiProcessors,
  10th European PVM/MPI Users' Group Conference (EuroPVM/MPI‘03), Venice, Italy, 29 Sep - 2 Oct, 2003
- Barbara Chapman,
  Parallel Application Development with the Hybrid MPI+OpenMP Programming Model,
  Tutorial, 9th EuroPVM/MPI & 4th DAPSYS Conference, Johannes Kepler University Linz, Austria September 29-October 02, 2002
- Luis F. Romero, Eva M. Ortigosa, Sergio Romero, Emilio L. Zapata,
  Nesting OpenMP and MPI in the Conjugate Gradient Method for Band Systems,
  11th European PVM/MPI Users' Group Meeting in conjunction with DAPSYS'04, Budapest, Hungary, September 19-22, 2004
- Nikolaos Drosinos and Nectarios Koziris,
  Advanced Hybrid MPI/OpenMP Parallelization Paradigms for Nested Loop Algorithms onto Clusters of SMPs,
  10th European PVM/MPI Users' Group Conference (EuroPVM/MPI‘03), Venice, Italy, 29 Sep - 2 Oct, 2003



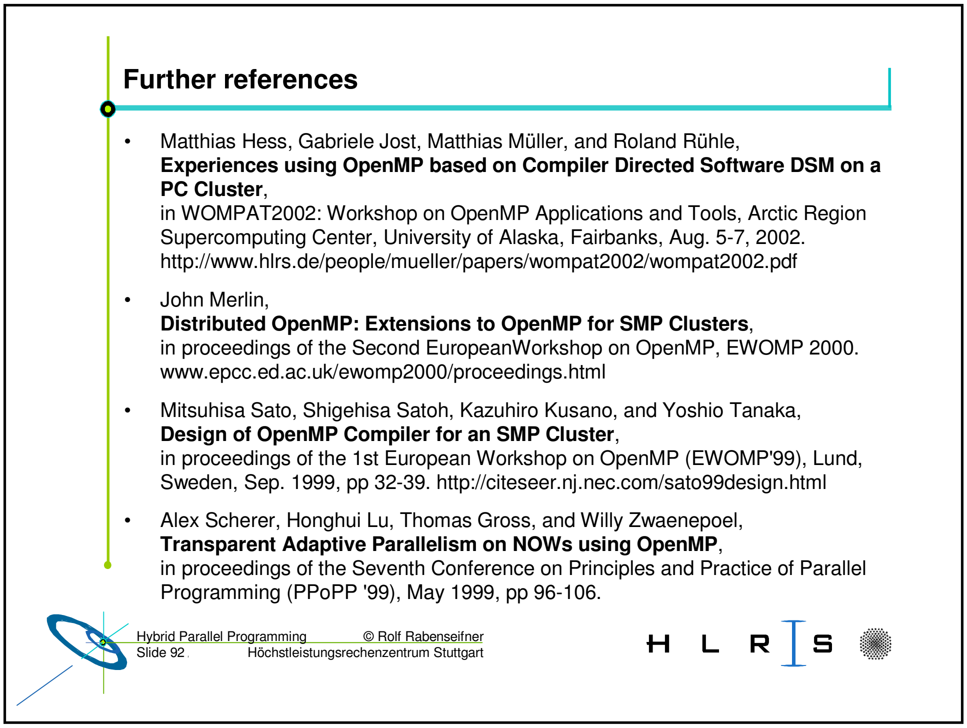
Further references

- Holger Brunst and Bernd Mohr,
  Performance Analysis of Large-scale OpenMP and Hybrid MPI/OpenMP Applications with VampirNG
  Proceedings for IWOMP 2005, Eugene, OR, June 2005.
  http://www.fz-juelich.de/zam/kojak/documentation/publications/
- Felix Wolf and Bernd Mohr,
  Automatic performance analysis of hybrid MPI/OpenMP applications
  Journal of Systems Architecture, Special Issue "Evolutions in parallel distributed and network-based processing", Volume 49, Issues 10-11, Pages 421-439, November 2003.
  http://www.fz-juelich.de/zam/kojak/documentation/publications/
- Felix Wolf and Bernd Mohr,
  Automatic Performance Analysis of Hybrid MPI/OpenMP Applications
  short version: Proceedings of the 11-th Euromicro Conference on Parallel, Distributed and Network based Processing (PDP 2003), Genoa, Italy, February 2003.
  long version: Technical Report FZJ-ZAM-IB-2001-05.
  http://www.fz-juelich.de/zam/kojak/documentation/publications/



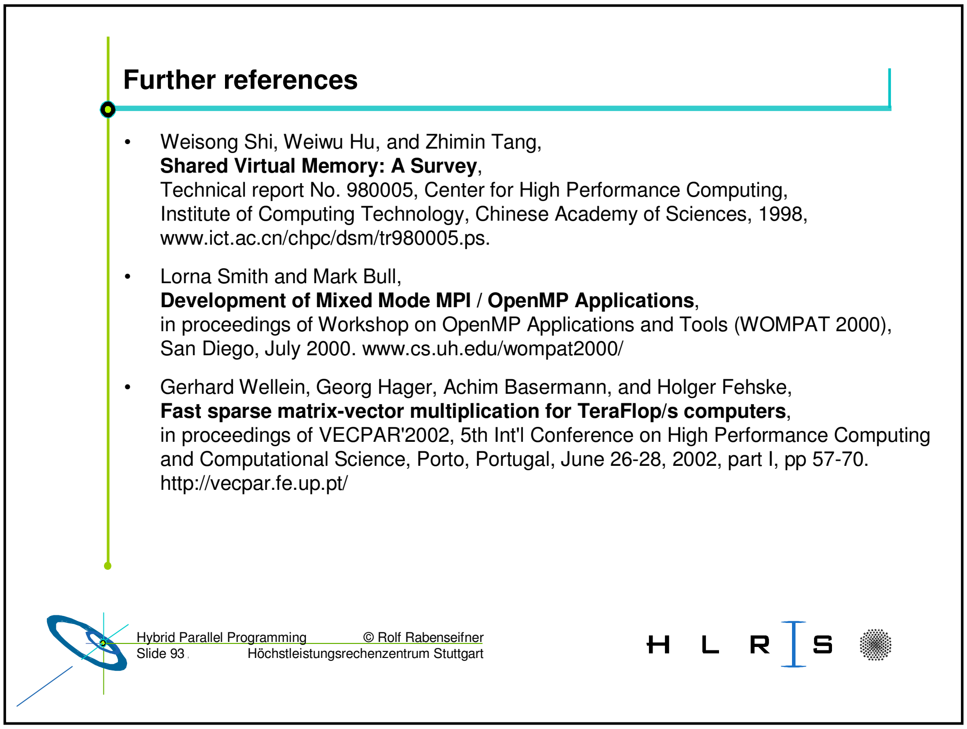
Further references

- Frank Cappello and Daniel Etiemble,
  MPI versus MPI+OpenMP on the IBM SP for the NAS benchmarks,
  in Proc. Supercomputing'00, Dallas, TX, 2000.
  http://citeseer.nj.nec.com/cappello00mpi.html
  www.sc2000.org/techpapr/papers/pap.pap214.pdf
- Jonathan Harris,
  Extending OpenMP for NUMA Architectures,
  in proceedings of the Second European Workshop on OpenMP, EWOMP 2000.
  www.epcc.ed.ac.uk/ewomp2000/proceedings.html
- D. S. Henty,
  Performance of hybrid message-passing and shared-memory parallelism for discrete element modeling,
  in Proc. Supercomputing'00, Dallas, TX, 2000.
  http://citeseer.nj.nec.com/henty00performance.html
  www.sc2000.org/techpapr/papers/pap.pap154.pdf



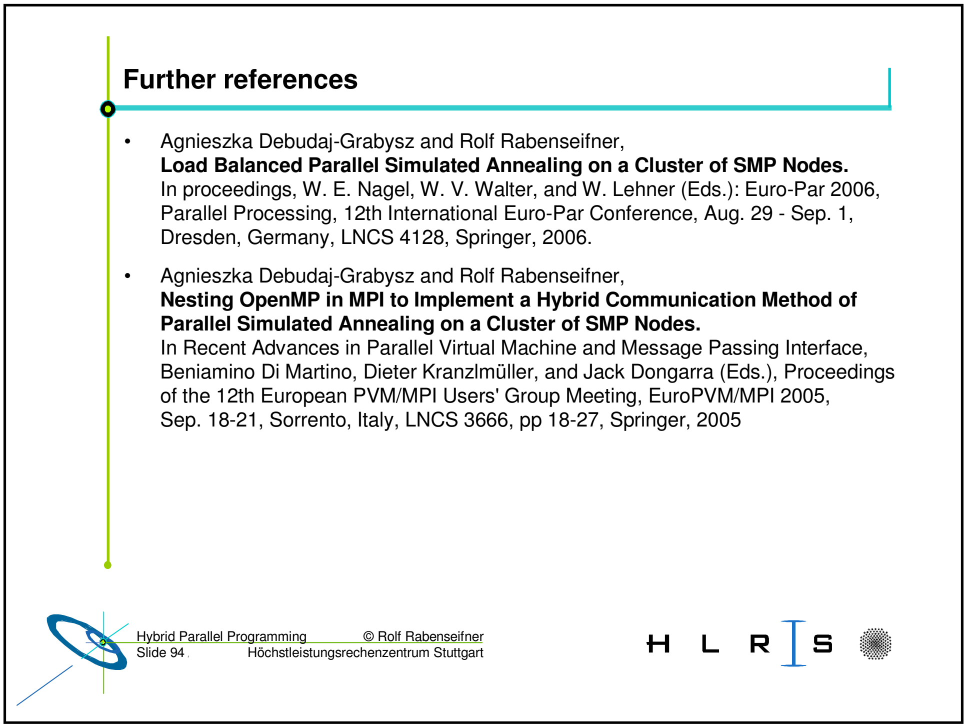
Further references

- Matthias Hess, Gabriele Jost, Matthias Müller, and Roland Rühle,
  Experiences using OpenMP based on Compiler Directed Software DSM on a PC Cluster,
  in WOMPAT2002: Workshop on OpenMP Applications and Tools, Arctic Region Supercomputing Center, University of Alaska, Fairbanks, Aug. 5-7, 2002.
  http://www.hlrs.de/people/mueller/papers/wompat2002/wompat2002.pdf
- John Merlin,
  Distributed OpenMP: Extensions to OpenMP for SMP Clusters,
  in proceedings of the Second EuropeanWorkshop on OpenMP, EWOMP 2000.
  www.epcc.ed.ac.uk/ewomp2000/proceedings.html
- Mitsuhisa Sato, Shigehisa Satoh, Kazuhiro Kusano, and Yoshio Tanaka,
  Design of OpenMP Compiler for an SMP Cluster,
  in proceedings of the 1st European Workshop on OpenMP (EWOMP'99), Lund, Sweden, Sep. 1999, pp 32-39. http://citeseer.nj.nec.com/sato99design.html
- Alex Scherer, Honghui Lu, Thomas Gross, and Willy Zwaenepoel,
  Transparent Adaptive Parallelism on NOWs using OpenMP,
  in proceedings of the Seventh Conference on Principles and Practice of Parallel Programming (PPoPP '99), May 1999, pp 96-106.



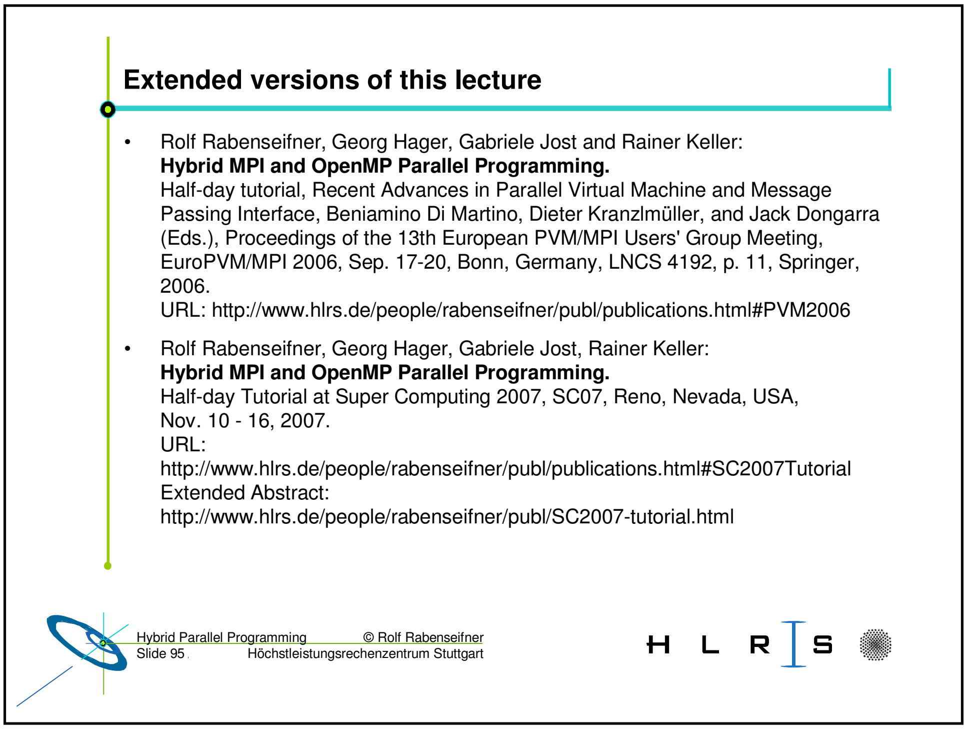
Further references

- Weisong Shi, Weiwu Hu, and Zhimin Tang,
  Shared Virtual Memory: A Survey,
  Technical report No. 980005, Center for High Performance Computing,
  Institute of Computing Technology, Chinese Academy of Sciences, 1998,
  www.ict.ac.cn/chpc/dsm/tr980005.ps.
- Lorna Smith and Mark Bull,
  Development of Mixed Mode MPI / OpenMP Applications,
  in proceedings of Workshop on OpenMP Applications and Tools (WOMPAT 2000),
  San Diego, July 2000. www.cs.uh.edu/wompat2000/
- Gerhard Wellein, Georg Hager, Achim Basermann, and Holger Fehske,
  Fast sparse matrix-vector multiplication for TeraFlop/s computers,
  in proceedings of VECPAR'2002, 5th Int'l Conference on High Performance Computing and Computational Science, Porto, Portugal, June 26-28, 2002, part I, pp 57-70.
  http://vecpar.fe.up.pt/



Further references

- Agnieszka Debudaj-Grabysz and Rolf Rabenseifner,
  Load Balanced Parallel Simulated Annealing on a Cluster of SMP Nodes.
  In proceedings, W. E. Nagel, W. V. Walter, and W. Lehner (Eds.): Euro-Par 2006, Parallel Processing, 12th International Euro-Par Conference, Aug. 29 - Sep. 1, Dresden, Germany, LNCS 4128, Springer, 2006.
- Agnieszka Debudaj-Grabysz and Rolf Rabenseifner,
  Nesting OpenMP in MPI to Implement a Hybrid Communication Method of Parallel Simulated Annealing on a Cluster of SMP Nodes.
  In Recent Advances in Parallel Virtual Machine and Message Passing Interface, Beniamino Di Martino, Dieter Kranzlmüller, and Jack Dongarra (Eds.), Proceedings of the 12th European PVM/MPI Users' Group Meeting, EuroPVM/MPI 2005, Sep. 18-21, Sorrento, Italy, LNCS 3666, pp 18-27, Springer, 2005



Extended versions of this lecture

- Rolf Rabenseifner, Georg Hager, Gabriele Jost and Rainer Keller:
  Hybrid MPI and OpenMP Parallel Programming.
  Half-day tutorial, Recent Advances in Parallel Virtual Machine and Message Passing Interface, Beniamino Di Martino, Dieter Kranzlmüller, and Jack Dongarra (Eds.), Proceedings of the 13th European PVM/MPI Users' Group Meeting, EuroPVM/MPI 2006, Sep. 17-20, Bonn, Germany, LNCS 4192, p. 11, Springer, 2006.
  URL: http://www.hlrs.de/people/rabenseifner/publ/publications.html#PVM2006
- Rolf Rabenseifner, Georg Hager, Gabriele Jost, Rainer Keller:
  Hybrid MPI and OpenMP Parallel Programming.
  Half-day Tutorial at Super Computing 2007, SC07, Reno, Nevada, USA,
  Nov. 10 - 16, 2007.
  URL:
  http://www.hlrs.de/people/rabenseifner/publ/publications.html#SC2007Tutorial
  Extended Abstract:
  http://www.hlrs.de/people/rabenseifner/publ/SC2007-tutorial.html