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,

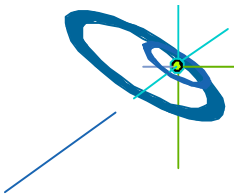
rabenseifner@hlrs.de *www.hlrs.de/people/rabenseifner*

Invited Talk in the Lecture

"Hochleistungsrechnen"

Prof. Dr. habil Thomas Ludwig, Deutsches Klimarechenzentrum (DKRZ), Hamburg

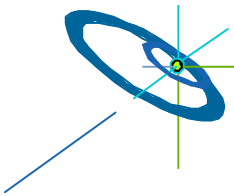
June 28, 2010



H L R S

Outline

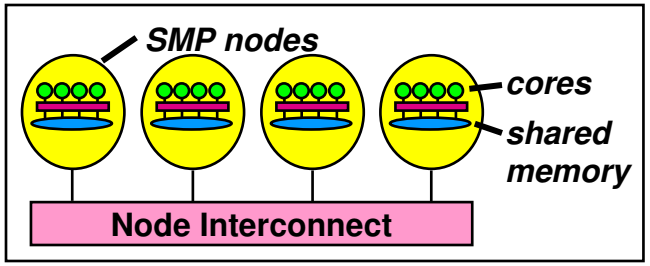
	slide number
• Introduction / Motivation	2
• Programming models on clusters of SMP nodes	6
• Case Studies / Benchmark results	13
• Mismatch Problems	23
• Opportunities: Application categories that can benefit from hybrid parallelization	52
• Thread-safety quality of MPI libraries	62
• Other options on clusters of SMP nodes	67
• Summary	81
• Appendix	89



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**

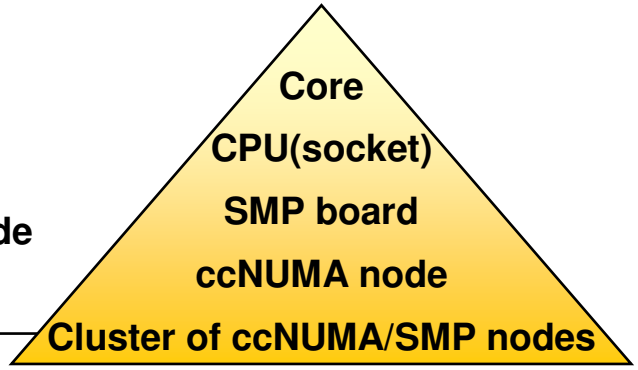


- Hardware range
  - **mini-cluster with dual-core CPUs**
  - **…**
  - **large constellations with large SMP nodes**

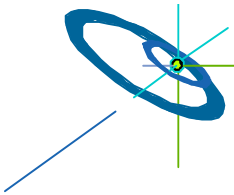
    … with several sockets (CPUs) per SMP node

    … with several cores per socket

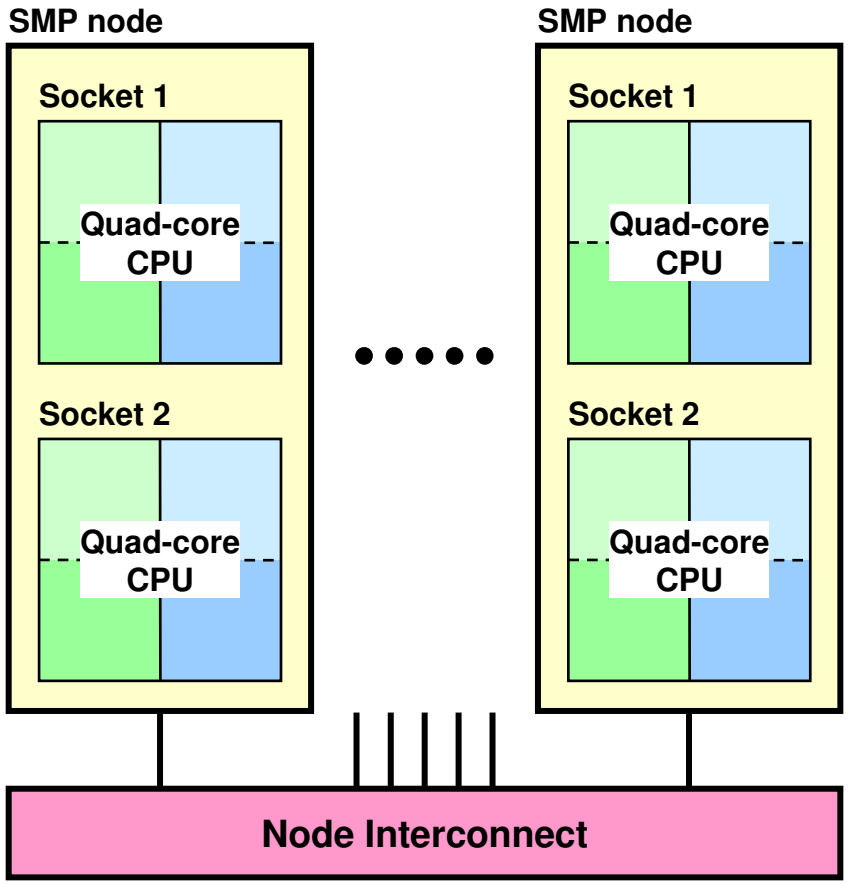
    → **Hierarchical system layout**



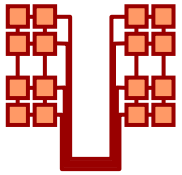
- Hybrid MPI/OpenMP programming seems natural
  - **MPI between the nodes**
  - **OpenMP inside of each SMP node**



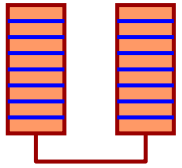
Motivation



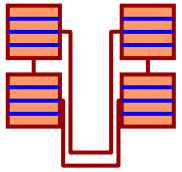
- Which programming model is fastest?
- MPI everywhere?



- Fully hybrid MPI & OpenMP?



- Something between? (Mixed model)



- Often hybrid programming **slower** than pure MPI
  - Examples, Reasons, …

Goals of this tutorial

- Sensitize to problems on clusters of SMP nodes

  see sections → Case studies
  → Mismatch problems

- Technical aspects of hybrid programming

  see sections → Programming models on clusters
  → Examples on hybrid programming

- Opportunities with hybrid programming

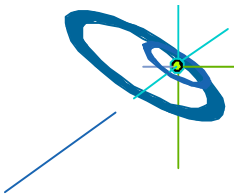
  see section → Opportunities: Application categories that can benefit from hybrid paralleliz.

- Issues and their Solutions

  with sections → Thread-safety quality of MPI libraries
  → Tools for debugging and profiling for MPI+OpenMP

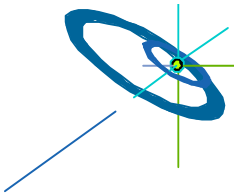
- **Less frustration &**
- **More success**

with your parallel program on clusters of SMP nodes



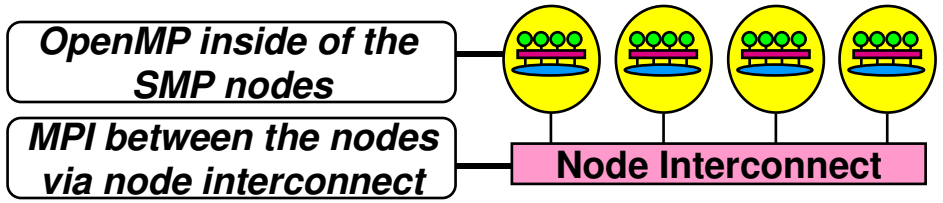
Outline

- Introduction / Motivation
- **Programming models on clusters of SMP nodes**
- Case Studies / Benchmark results
- Mismatch Problems
- Opportunities:
  Application categories that can benefit from hybrid parallelization
- Thread-safety quality of MPI libraries
- Other options on clusters of SMP nodes
- Summary

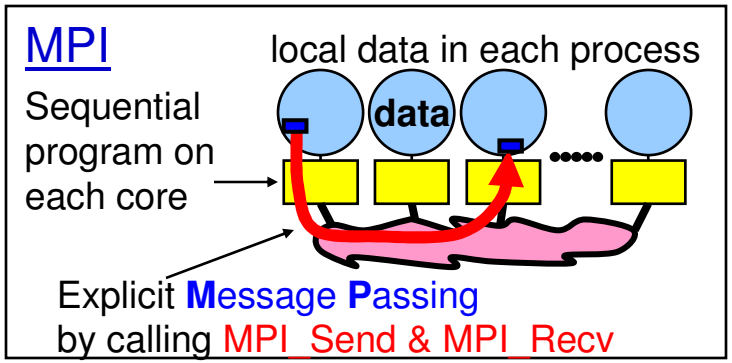


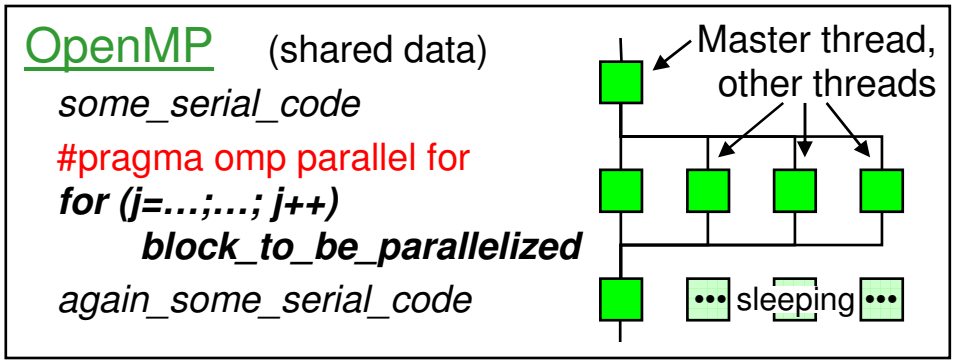
Major Programming models on hybrid systems

- Pure MPI (one MPI process on each core)
- Hybrid MPI+OpenMP
  - shared memory OpenMP
  - distributed memory MPI

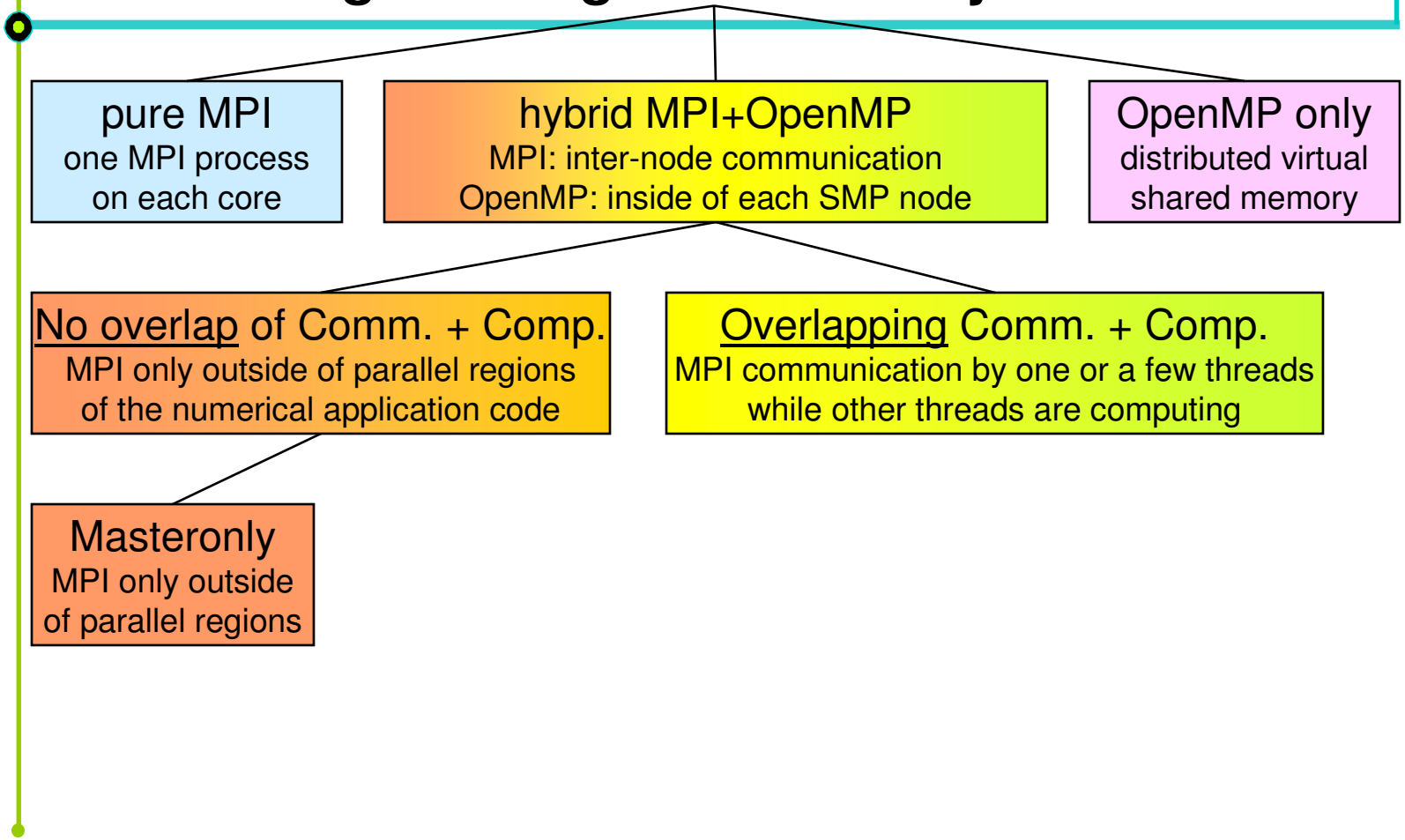


- Other: Virtual shared memory systems, PGAS, HPF, …
- Often **hybrid programming (MPI+OpenMP)** slower than **pure MPI**
  - why?





Parallel Programming Models on Hybrid Platforms



Pure MPI

pure MPI
one MPI process
on each core

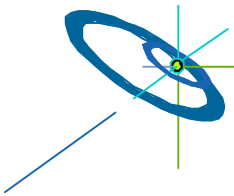
Advantages

- 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!

Discussed in detail later on in the section **Mismatch Problems**



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

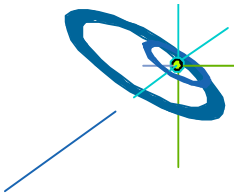
- All other threads are sleeping while master thread communicates!
- Which inter-node bandwidth?
- MPI-lib must support at least MPI_THREAD_FUNNELED

→ Section **Thread-safety quality of MPI libraries**

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)
```



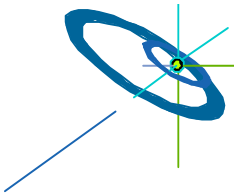
Pure OpenMP (on the cluster)

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)

Experience:
→ **Mismatch** section

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

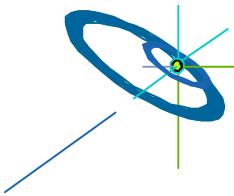


Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- **Case Studies / Benchmark results**
  - The Multi-Zone NAS Parallel Benchmarks

    Gabriele Jost (University of Texas,TACC/Naval Postgraduate School, Monterey CA)
  - Micro Benchmarks

    Georg Hager (Regionales Rechenzentrum Erlangen, RRZE)
- Mismatch Problems
- Opportunities: Application categories that can benefit from hybrid paralleli.
- Thread-safety quality of MPI libraries
- Other options on clusters of SMP nodes
- Summary



The Multi-Zone NAS Parallel Benchmarks

- Aggregate sizes:
  - Class D: 1632 x 1216 x 34 grid points
  - Class E: 4224 x 3456 x 92 grid points
- **BT-MZ: (Block tridiagonal simulated CFD application)**
  - Alternative Directions Implicit (ADI) method
  - #Zones: 1024 (D), 4096 (E)
  - Size of the zones varies widely:
    - large/small about 20
    - requires multi-level parallelism to achieve a good load-balance
- **LU-MZ: (LU decomposition simulated CFD application)**
  - SSOR method (2D pipelined method)
  - #Zones: 16 (all Classes)
  - Size of the zones identical:
    - no load-balancing required
    - limited parallelism on outer level
- **SP-MZ: (Scalar Pentadiagonal simulated CFD application)**
  - #Zones: 1024 (D), 4096 (E)
  - Size of zones identical
    - no load-balancing required

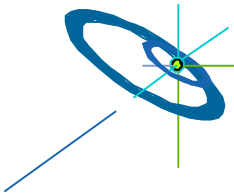
Expectations:

Pure MPI: Load-balancing problems!

Good candidate for MPI+OpenMP

Limitted MPI Parallelism:
→ MPI+OpenMP increases Parallelism

Load-balanced on MPI level: Pure MPI should perform best



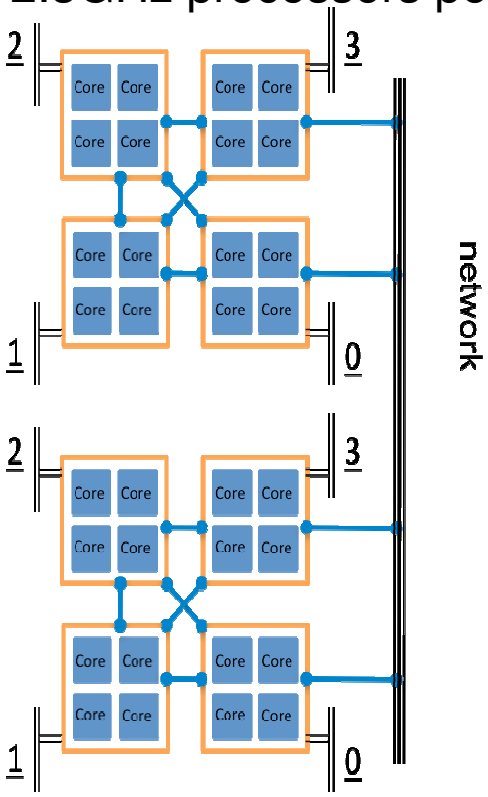
HLRS

TACC

Slides, courtesy of Gabriele Jost, TACC, Austin, USA

Sun Constellation Cluster Ranger (1)

- Located at the Texas Advanced Computing Center (TACC), University of Texas at Austin (http://www.tacc.utexas.edu)
- 3936 Sun Blades, 4 AMD Quad-core 64bit 2.3GHz processors per node (blade), 62976 cores total
- 123TB aggregrate memory
- Peak Performance 579 Tflops
- InfiniBand Switch interconnect
- Sun Blade x6420 Compute Node:
  - 4 Sockets per node
  - 4 cores per socket
  - HyperTransport System Bus
  - 32GB memory

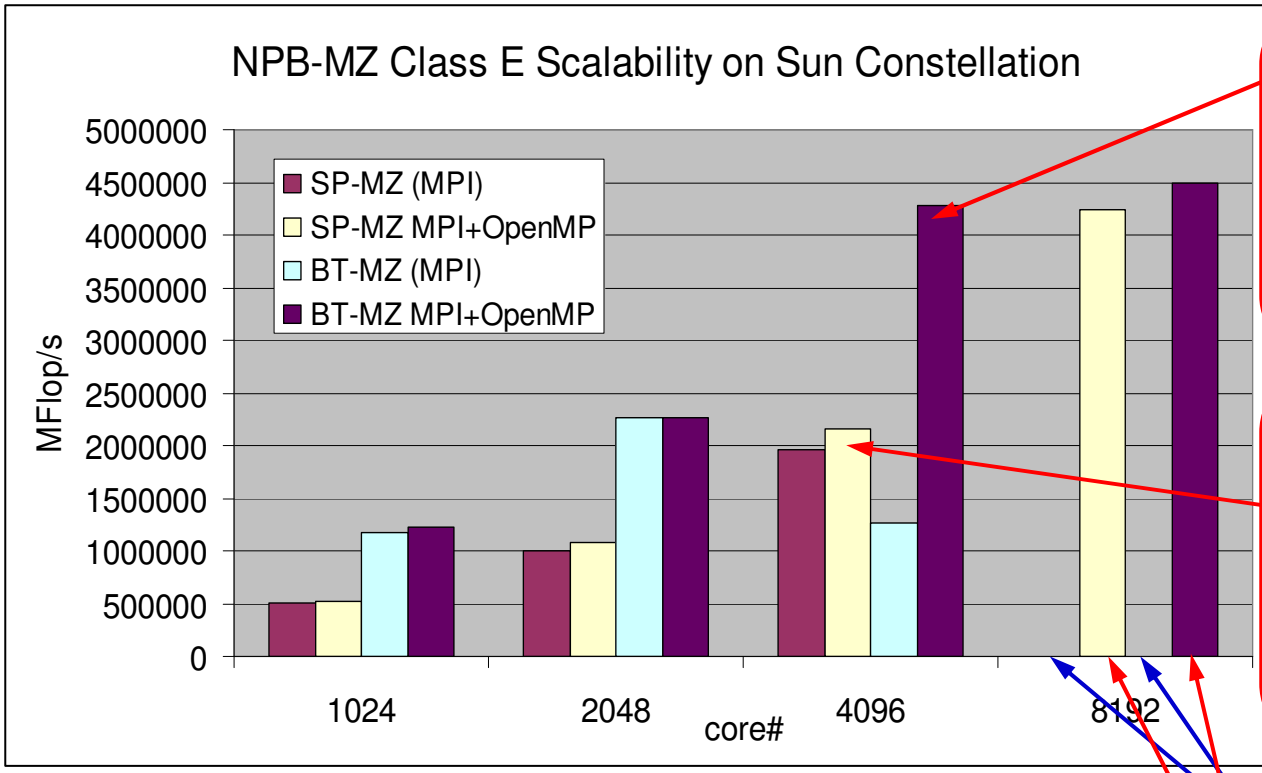


H L R S

TACC

Slides, courtesy of Gabriele Jost, TACC, Austin, USA

SUN: NPB-MZ Class E Scalability on Ranger



BT

Significant improvement (235%):

Load-balancing issues solved with MPI+OpenMP

SP

Pure MPI is already load-balanced.

But hybrid 9.6% faster, due to smaller message rate at NIC

Cannot be build for 8192 processes!

Hybrid:

SP: still scales

BT: does not scale

- Scalability in Mflops
- MPI/OpenMP outperforms pure MPI
- Use of numactl essential to achieve scalability

H L

TACC

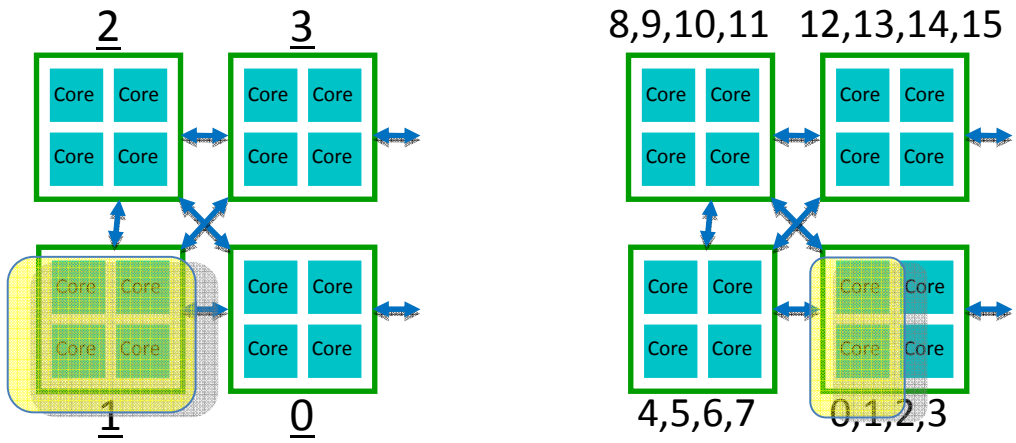
Slides, courtesy of Gabriele Jost, TACC, Austin, USA

skipped

NUMA Control: Process Placement

- Affinity and Policy can be changed externally through numactl at the socket and core level.

Command: numactl <options> ./a.out



Socket References

Example: numactl -N 1 ./a.out

Core References

Example: numactl –c 0,1 ./a.out

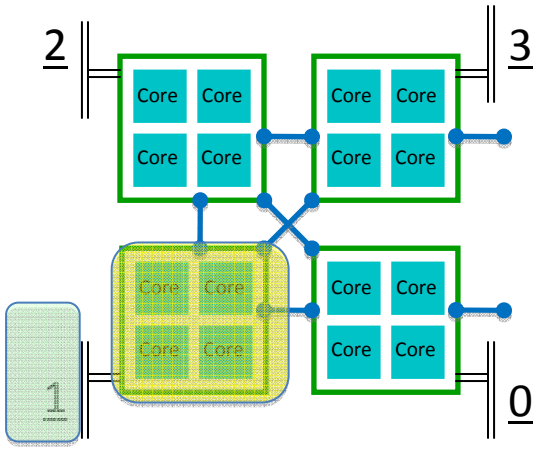
H L R S

TACC

Slides, courtesy of Gabriele Jost, TACC, Austin, USA

skipped

NUMA Operations: Memory Placement



Memory: Socket References

Memory allocation:

- MPI
  - local allocation is best
- OpenMP
  - Interleave best for large, completely shared arrays that are randomly accessed by different threads
  - local best for private arrays
- Once allocated, a memory-structure is fixed

Example: numactl -N 1 -l ./a.out

H L R S

TACC

Slides, courtesy of Gabriele Jost, TACC, Austin, USA

skipped

NUMA Operations (cont. 3)

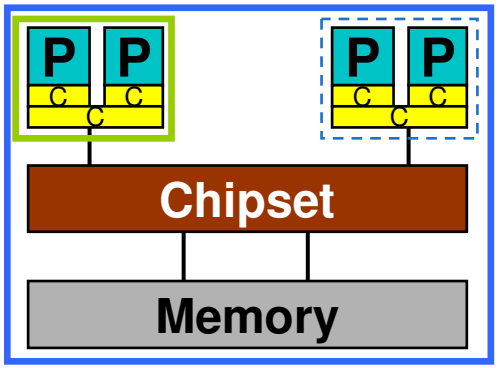
	cmd	option	arguments	description
Socket Affinity	numactl	-N	{0,1,2,3}	Only execute process on cores of this (these) socket(s).
Memory Policy	numactl	-l	{no argument}	Allocate on current socket.
Memory Policy	numactl	-i	{0,1,2,3}	Allocate round robin (interleave) on these sockets.
Memory Policy	numactl	--preferred=	{0,1,2,3} select only one	Allocate on this socket; fallback to any other if full .
Memory Policy	numactl	-m	{0,1,2,3}	Only allocate on this (these) socket(s).
Core Affinity	numactl	-C	{0,1,2,3, 4,5,6,7, 8,9,10,11, 12,13,14,15}	Only execute process on this (these) Core(s).

Slides, courtesy of Gabriele Jost, TACC, Austin, USA

Intra-node MPI characteristics: IMB Ping-Pong benchmark

- Code (to be run on 2 processors):

```
wc = MPI_WTIME()

do i=1,NREPEAT

  if(rank.eq.0) then
   MPI_SEND(buffer,N,MPI_BYTE,1,0,MPI_COMM_WORLD,ierr)
   MPI_RECV(buffer,N,MPI_BYTE,1,0,MPI_COMM_WORLD, &
                 status,ierr)
  else
   MPI_RECV(…)
   MPI_SEND(…)
  endif

enddo

wc = MPI_WTIME() - wc
```



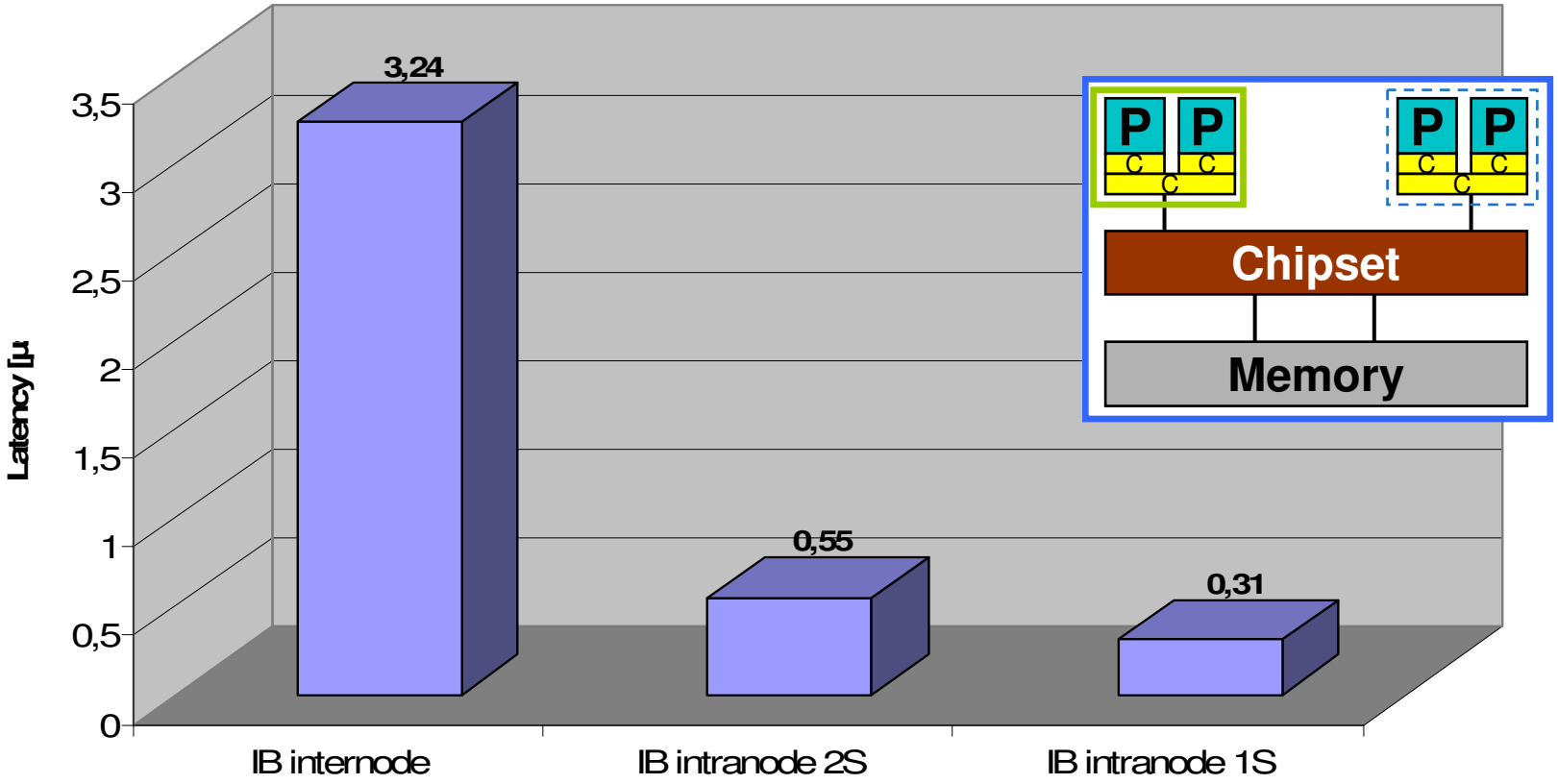
- Intranode (1S): `mpirun -np 2 -pin "1 3" ./a.out`
- Intranode (2S): `mpirun -np 2 -pin "2 3" ./a.out`
- Internode: `mpirun -np 2 -pernode ./a.out`

H L R S

Slides, courtesy of Georg Hager, RRZE, Erlangen

IMB Ping-Pong: Latency

Intra-node vs. Inter-node on Woodcrest DDR-IB cluster (Intel MPI 3.1)



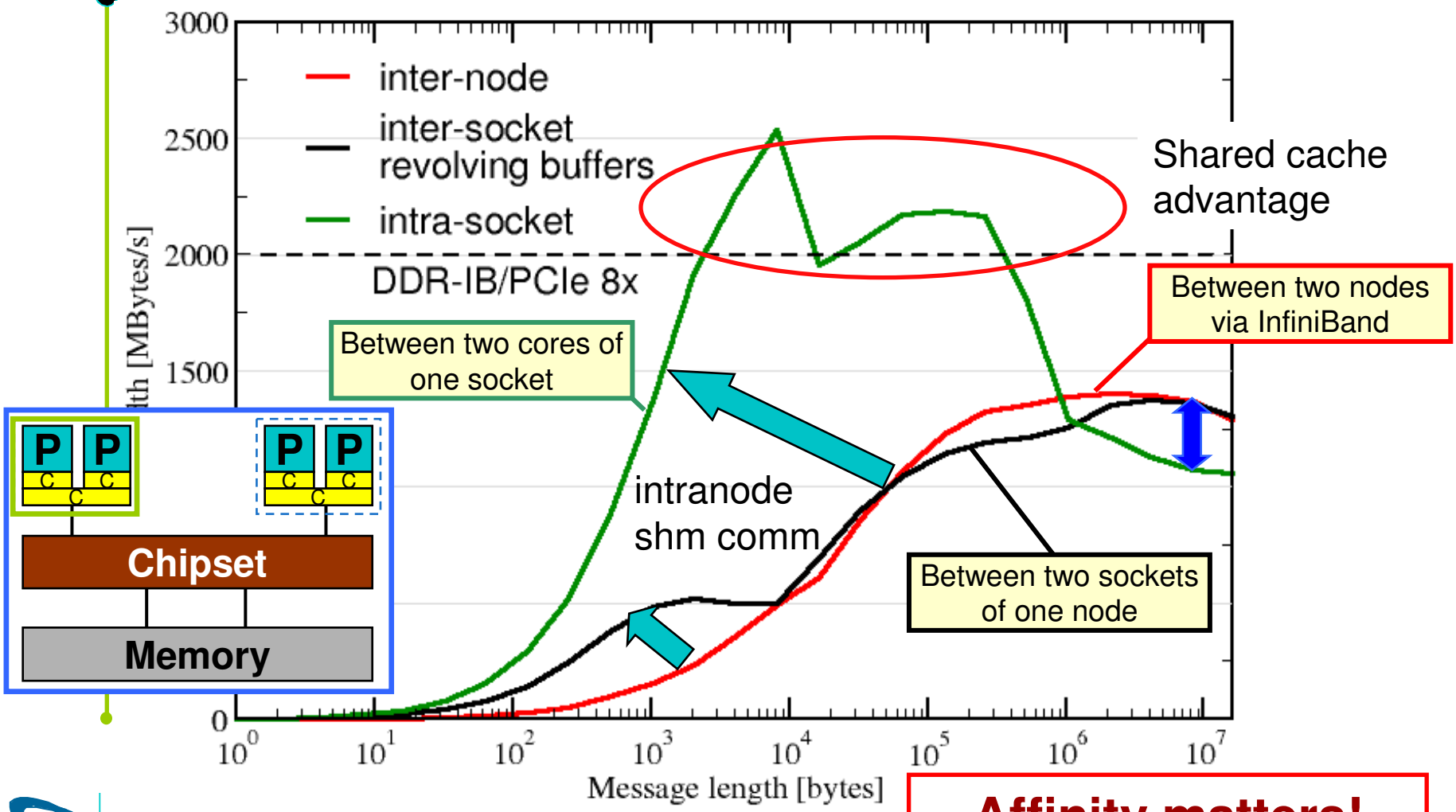
Affinity matters!

H L R S

Slides, courtesy of Georg Hager, RRZE, Erlangen

IMB Ping-Pong: Bandwidth Characteristics

Intra-node vs. Inter-node on Woodcrest DDR-IB cluster (Intel MPI 3.1)

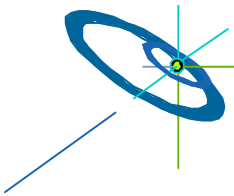


Affinity matters!

Slides, courtesy of Georg Hager, RRZE, Erlangen

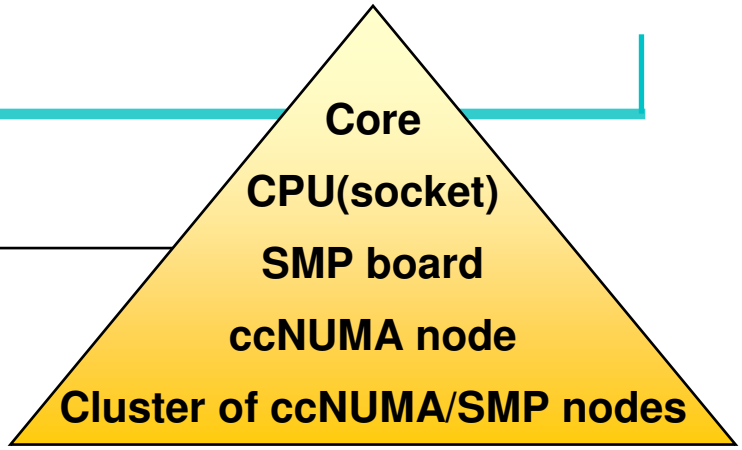
Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / Benchmark results
- **Mismatch Problems**
- Opportunities:
  Application categories that can benefit from hybrid parallelization
- Thread-safety quality of MPI libraries
- Other options on clusters of SMP nodes
- Summary



Mismatch Problems

- None of the programming models fits to the hierarchical hardware (cluster of SMP nodes)
- Several mismatch problems
  → following slides
- Benefit through hybrid programming
  → Opportunities, see next section
- Quantitative implications
  → depends on you application



Examples:	No.1	No.2
Benefit through hybrid (see next section)	30%	10%
Loss by mismatch problems	–10%	–25%
Total	+20%	–15%

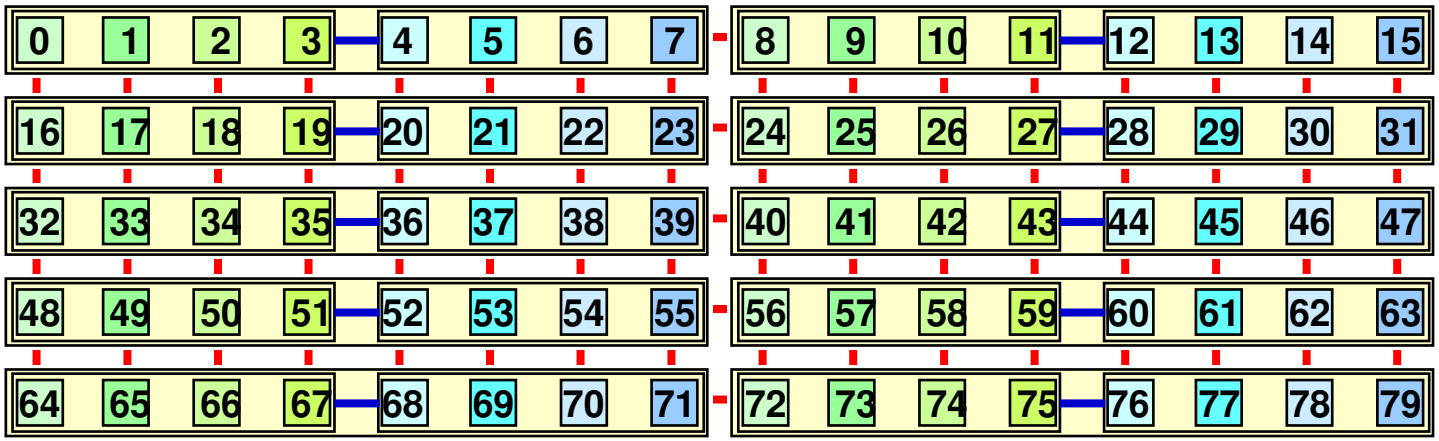
In most cases: **Both categories!**

The Topology Problem with **pure MPI**

one MPI process on each core

Application example on 80 cores:

- Cartesian application with 5 x 16 = 80 sub-domains
- On system with 10 x dual socket x quad-core



\+ 17 x inter-node connections per node

− 1 x inter-socket connection per node

Does it matter?

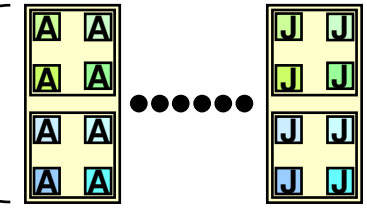
Sequential ranking of MPI_COMM_WORLD

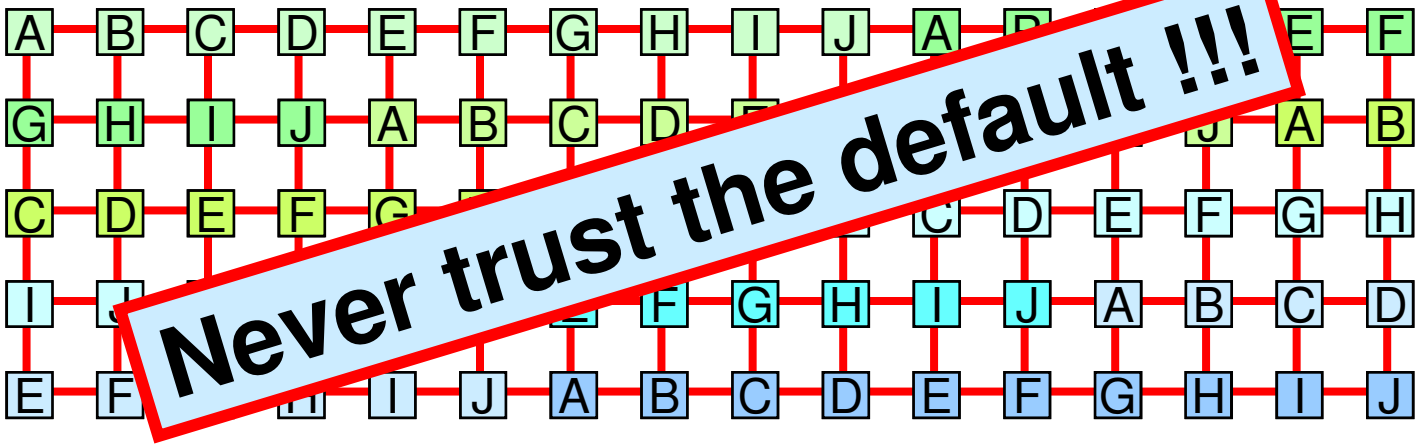
The Topology Problem with **pure MPI**

one MPI process on each core

Application example on 80 cores:

- Cartesian application with 5 x 16 = 80 sub-domains
- On system with 10 x dual socket x quad-core





Never trust the default !!!

\+ 32 x inter-node connections per node

− 0 x inter-socket connection per node

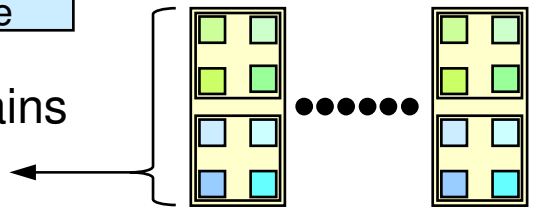
Round robin ranking of MPI_COMM_WORLD

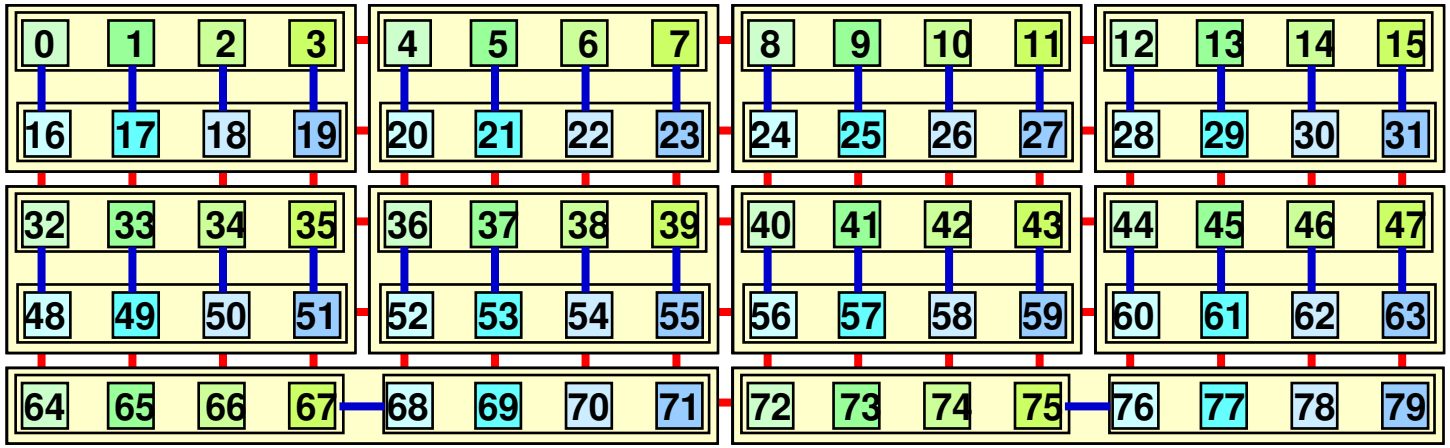
The Topology Problem with **pure MPI**

one MPI process on each core

Application example on 80 cores:

- Cartesian application with 5 x 16 = 80 sub-domains
- On system with 10 x dual socket x quad-core





\+ 12 x inter-node connections per node

\+ 4 x inter-socket connection per node

Two levels of domain decomposition

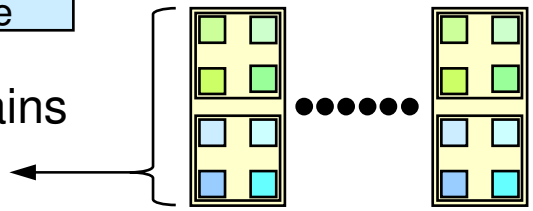
Bad affinity of cores to thread ranks

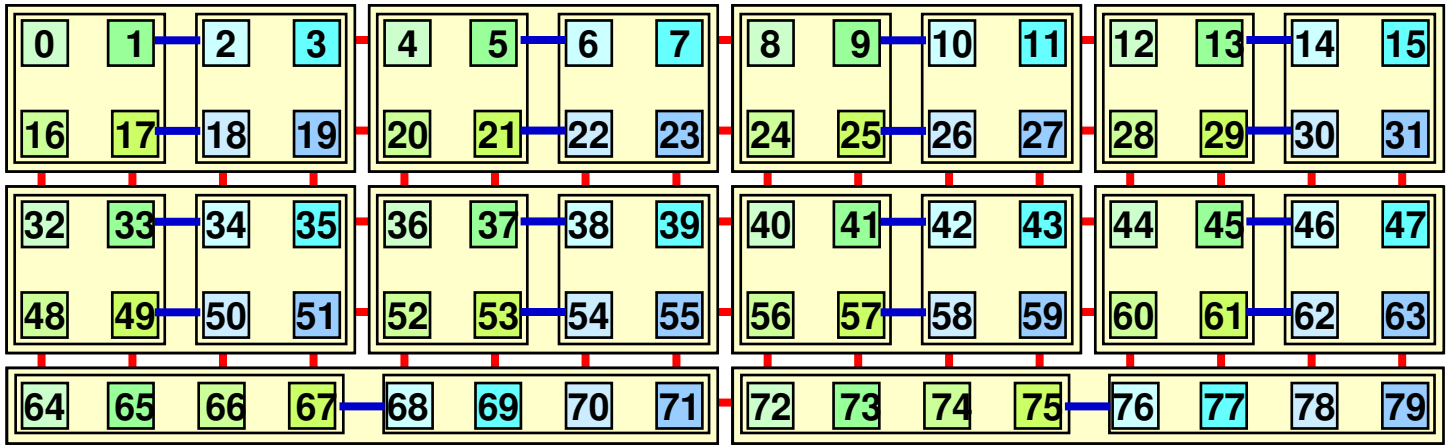
The Topology Problem with **pure MPI**

one MPI process on each core

Application example on 80 cores:

- Cartesian application with 5 x 16 = 80 sub-domains
- On system with 10 x dual socket x quad-core





\+ 12 x inter-node connections per node

\+ 2 x inter-socket connection per node

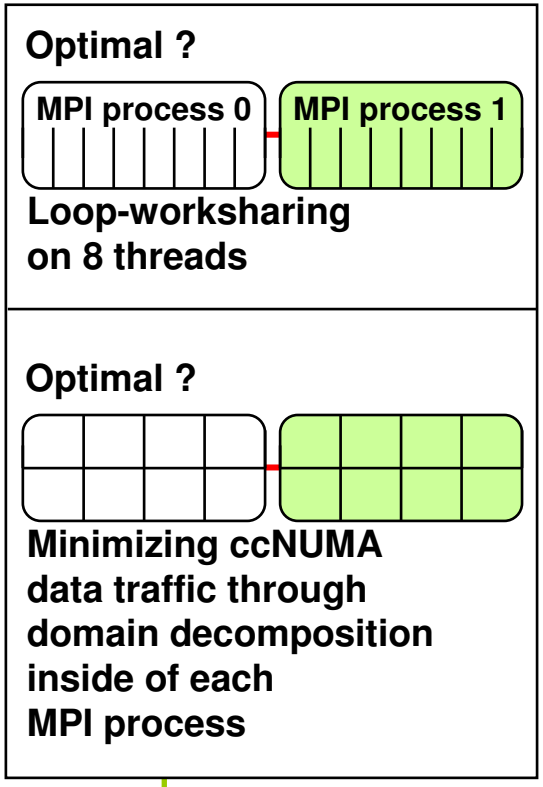
Two levels of domain decomposition

Good affinity of cores to thread ranks

The Topology Problem with hybrid MPI+OpenMP

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

Exa.: 2 SMP nodes, 8 cores/node



Problem

- Does application topology inside of SMP parallelization fit on inner hardware topology of each SMP node?

Solutions:

- Domain decomposition inside of each thread-parallel MPI process, and
- first touch strategy with OpenMP

Successful examples:

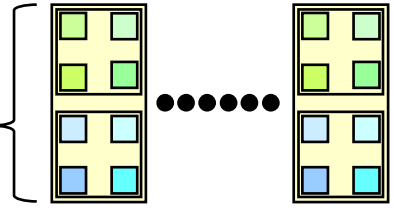
- Multi-Zone NAS Parallel Benchmarks (MZ-NPB)

The Topology Problem with hybrid MPI+OpenMP

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

Application example:

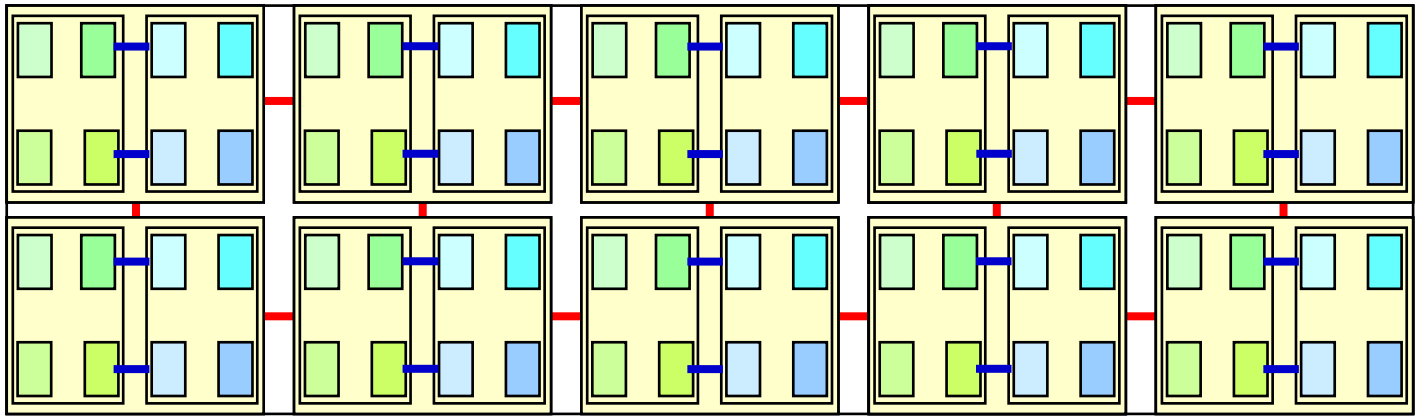
- Same Cartesian application aspect ratio: 5 x 16
- On system with 10 x dual socket x quad-core
- 2 x 5 domain decomposition



Application

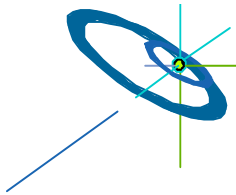
MPI Level

OpenMP



\+ 3 x inter-node connections per node, but ~ 4 x more traffic

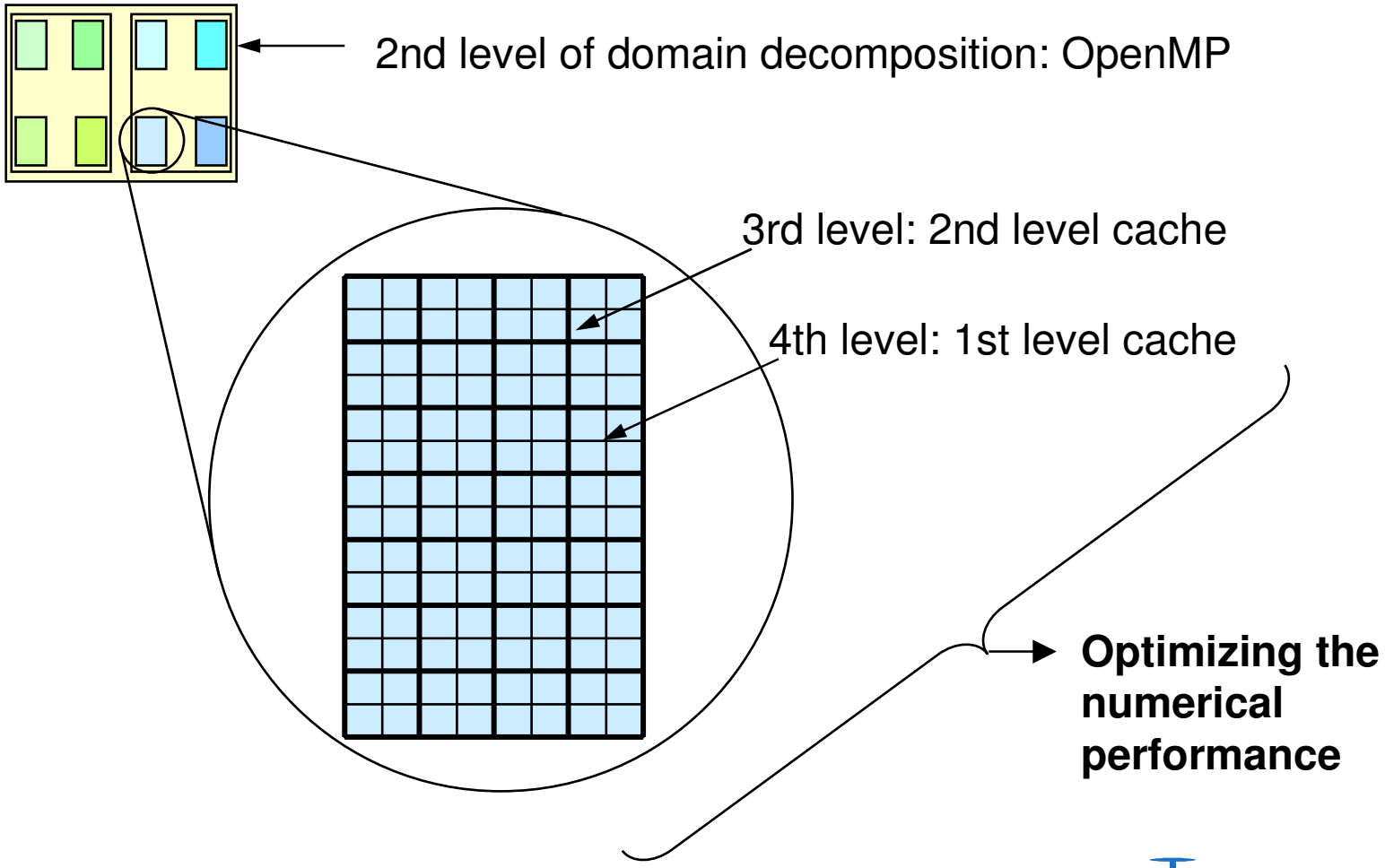
\+ 2 x inter-socket connection per node



Affinity of cores to thread ranks !!!

skipped

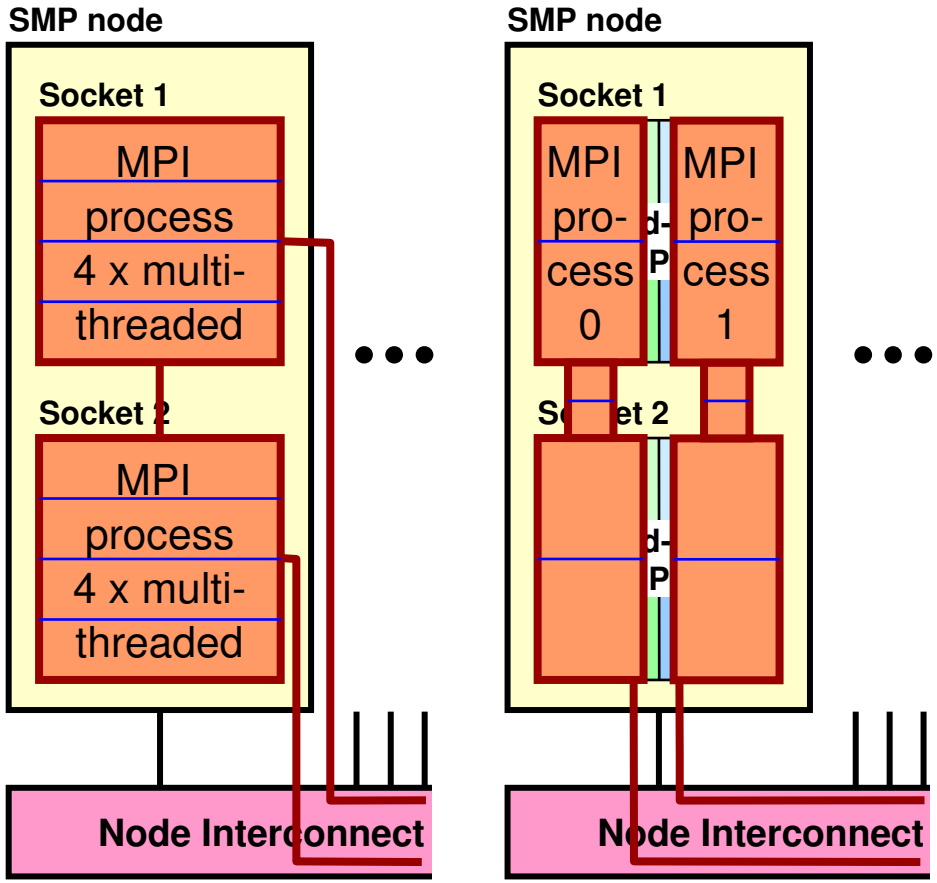
Inside of an SMP node



The Mapping Problem with mixed model

pure MPI
&
hybrid MPI+OpenMP

Do we have this? … or that?



Several multi-threaded MPI process per SMP node:

Problem

- Where are your processes and threads really located?

Solutions:

- Depends on your platform,
- e.g., with **numactl**

→ Case study on Sun Constellation Cluster Ranger with BT-MZ and SP-MZ

Further questions:

- Where is the NIC[1)] located?
- Which cores share caches?

HLRS

[1)] NIC = Network Interface Card

Unnecessary intra-node communication

pure MPI

Mixed model
(several multi-threaded MPI processes per SMP node)

Problem:

- If several MPI process on each SMP node
  → unnecessary intra-node communication

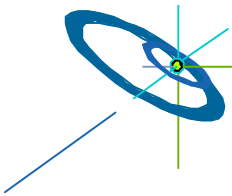
Solution:

- Only one MPI process per SMP node

Remarks:

- MPI library must use appropriate fabrics / protocol for intra-node communication
- Intra-node bandwidth higher than inter-node bandwidth
  → problem may be small
- MPI implementation may cause unnecessary data copying
  → waste of memory bandwidth

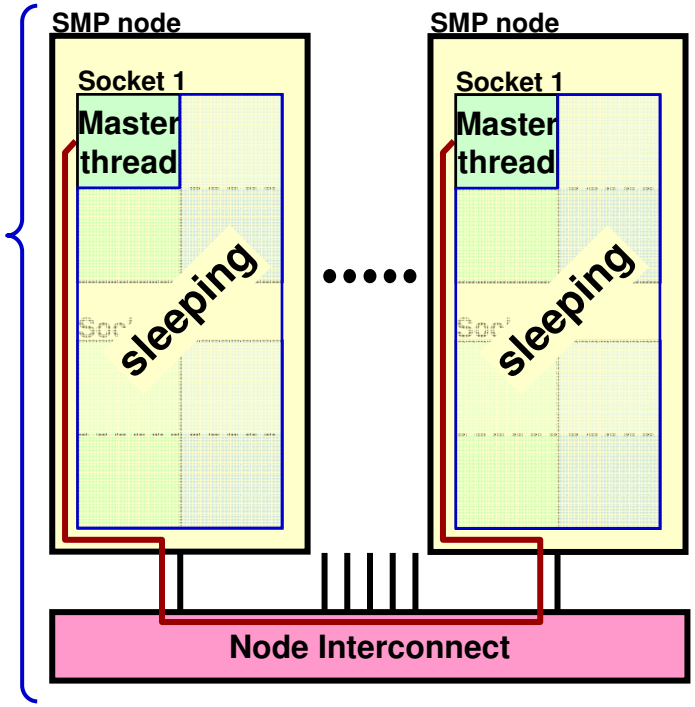
Quality aspects of the MPI library



Sleeping threads and network saturation with 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
```



Problem 1:

- **Can the master thread saturate the network?**

Solution:

- If not, use mixed model
- i.e., several MPI processes per SMP node

Problem 2:

- **Sleeping threads are wasting CPU time**

Solution:

- Overlapping of computation and communication

Problem 1&2 together:

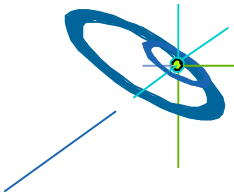
- Producing more idle time through lousy bandwidth of master thread

OpenMP: Additional Overhead & Pitfalls

- Using OpenMP
  - → may prohibit compiler optimization
  - → **may cause significant loss of computational performance**
- Thread fork / join overhead
- On ccNUMA SMP nodes:
  - **Loss of performance due to missing memory page locality or missing first touch strategy**
  - E.g. with the masteronly scheme:
    - One thread produces data
    - Master thread sends the data with MPI

    → data may be internally communicated from one memory to the other one
- Amdahl's law for each level of parallelism
- Using MPI-parallel application libraries? → Are they prepared for hybrid?

See, e.g., the necessary **–O4** flag with mpxlf_r on IBM Power6 systems



Overlapping Communication and Computation

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

Three problems:

- 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
  (see next slide)

- 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

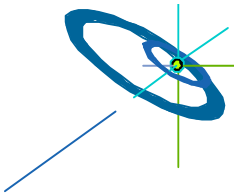
MPI communication by one or a few threads while other threads are computing

Subteams

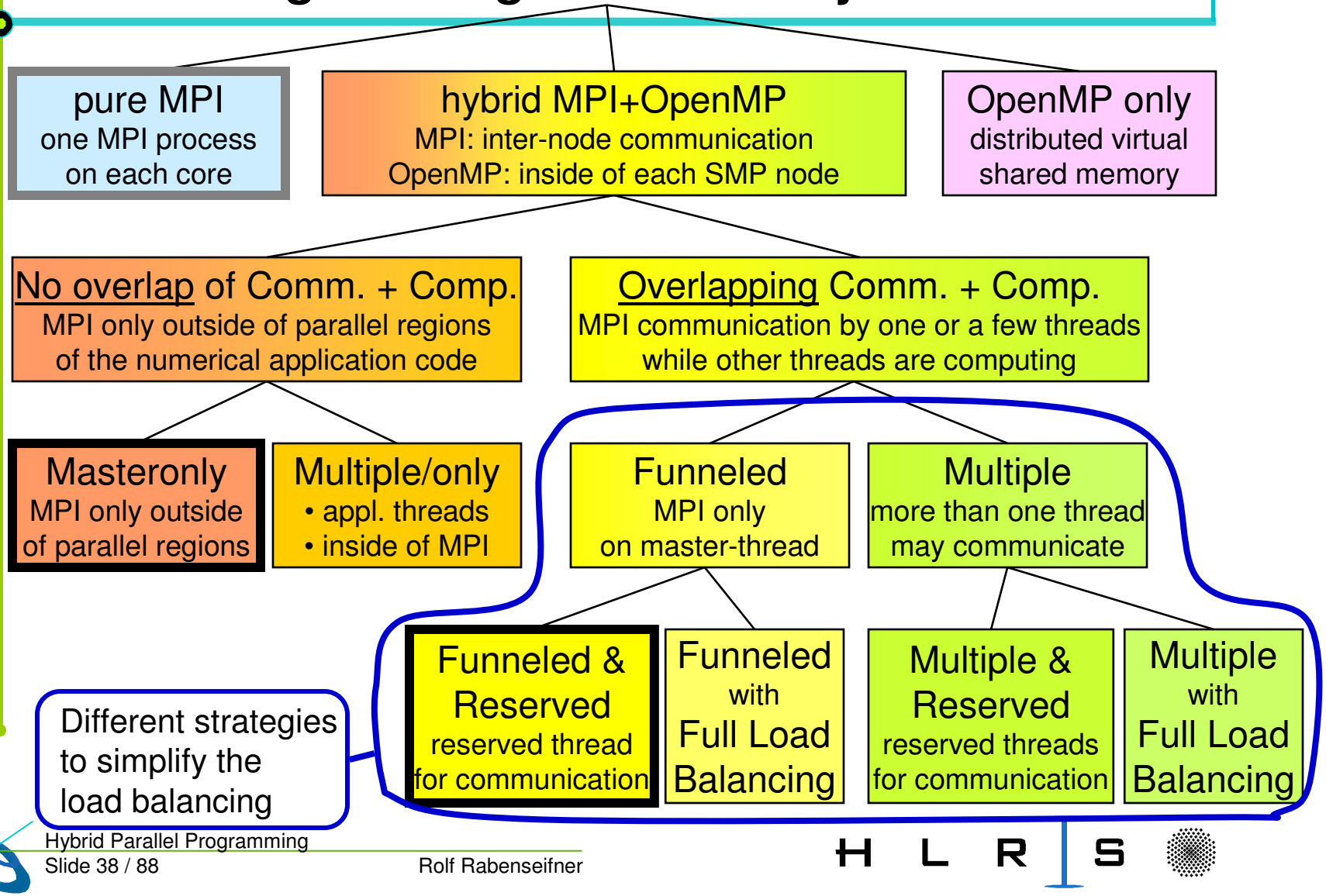
- Important proposal for OpenMP 3.x or OpenMP 4.x

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 */
```



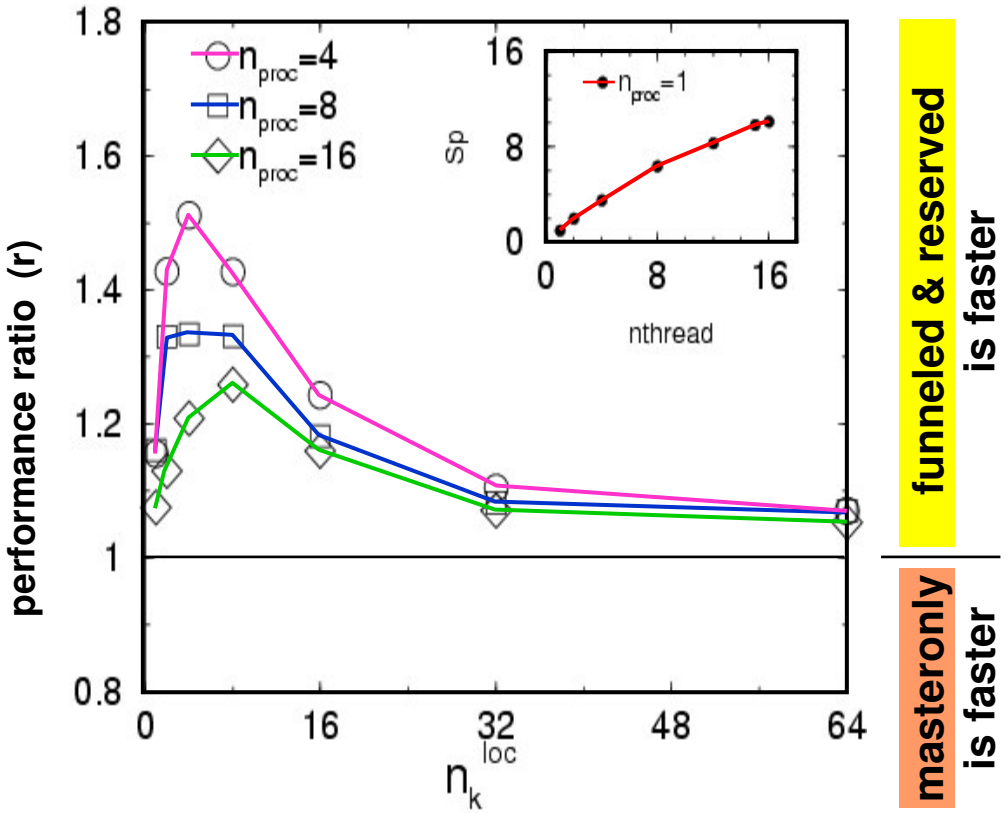
Parallel Programming Models on Hybrid Platforms



Experiment: Matrix-vector-multiply (MVM)

Masteronly

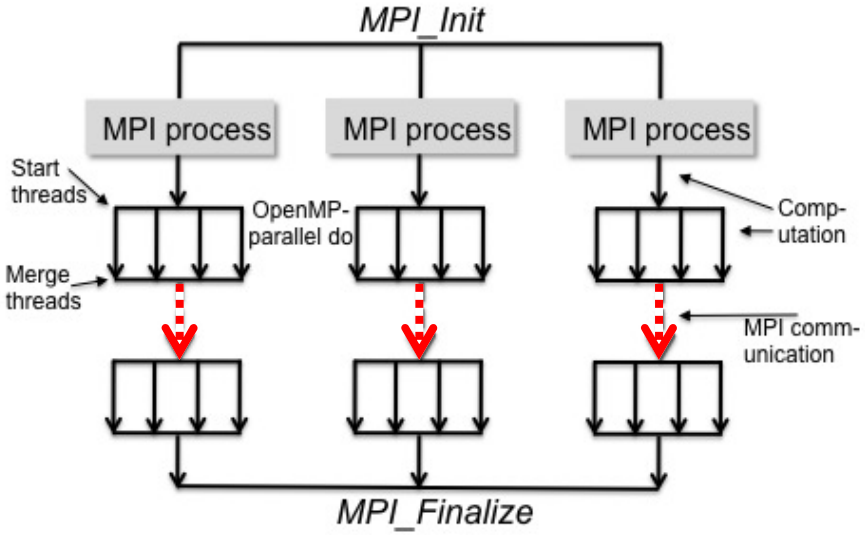
funneled & reserved

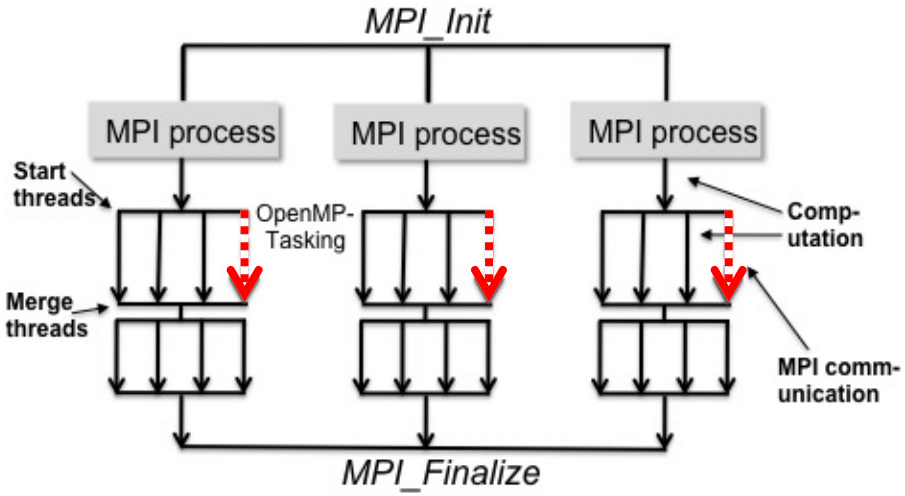


- Jacobi-Davidson-Solver on **IBM SP Power3** nodes with **16 CPUs per node**
- funneled&reserved is **always faster** in this experiments
- Reason: Memory bandwidth is already saturated by 15 CPUs, see inset
- Inset: Speedup on 1 SMP node using different number of threads

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 .

Overlapping: Using OpenMP tasks





NEW OpenMP Tasking Model gives a new way to achieve more parallelism form hybrid computation.

Alice Koniges et al.:
Application Acceleration on Current and Future Cray Platforms.
Proceedings, CUG 2010, Edinburgh, GB, May 24-27, 2010.

Slides, courtesy of Alice Koniges, NERSC, LBNL

Case study: Communication and Computation in Gyrokinetic Tokamak Simulation (GTS) shift routine

```
do iterations=1,N
!compute particles to be shifted
  !$omp parallel do
  shift_p=particles_to_shift(p_array);

!communicate amount of shifted
! particles and return if equal to 0
  shift_p=x+y
  MPI_ALLREDUCE(shift_p,sum_shift_p),
  if(sum_shift_p==0) { return; }

!pack particle to move right and left
  !$omp parallel do
  do m=1,x
    sendright(m)=p_array(f(m));
  enddo
  !$omp parallel do
  do n=1,y
    sendleft(n)=p_array(f(n));
  enddo
```

```
1  !reorder remaining particles: fill holes
     fill_hole(p_array);
3  !send number of particles to move right
     MPI_SENDRECV(x,length=2,..);
5  !send to right and receive from left
     MPI_SENDRECV(sendright,length=g(x),..);
7  !send number of particles to move left
     MPI_SENDRECV(y,length=2,..);
9  !send to left and receive from right
     MPI_SENDRECV(sendleft,length=g(y),..);
11 !adding shifted particles from right
     !$omp parallel do
13   do m=1,x
       p_array(h(m))=sendright(m);
15   enddo
   !adding shifted particles from left
17   !$omp parallel do
     do n=1,y
19     p_array(h(n))=sendleft(n);
     enddo
21 }
```

```
23
25
27
29
31
33
35
37
39
41
43
```

INDEPENDENT

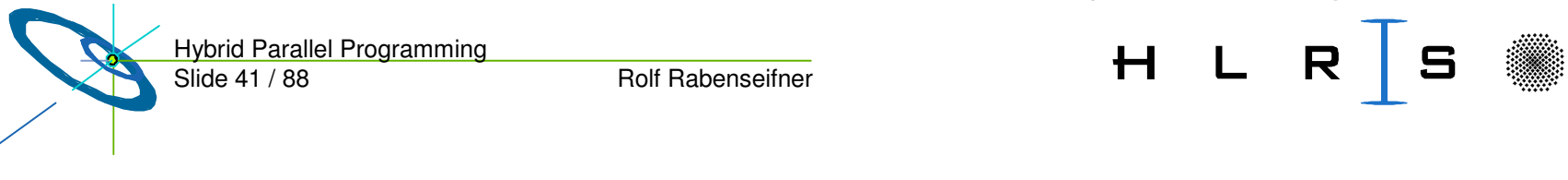
INDEPENDENT

SEMI-INDEPENDENT

GTS shift routine

Work on particle array (packing for sending, reordering, adding after sending) can be overlapped with **data independent** MPI communication using **OpenMP tasks.**

Slides, courtesy of Alice Koniges, NERSC, LBNL



Overlapping can be achieved with OpenMP tasks (1st part)

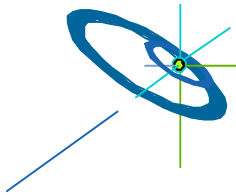
```
integer stride=1000
!$omp parallel
!$omp master
!pack particle to move right
  do m=1,x-stride,stride
    !$omp task
    do mm=0,stride-1,1
      sendright(m+mm)=p_array(f(m+mm));
    enddo
    !$omp end task
  enddo
  !$omp task
  do m=m,x
    sendright(m)=p_array(f(m));
  enddo
  !$omp end task
```

```
!pack particle to move left
  do n=1,y-stride,stride
    !$omp task
    do nn=0,stride-1,1
      sendleft(n+nn)=p_array(f(n+nn));
    enddo
    !$omp end task
  enddo
  !$omp task
  do n=n,y
    sendleft(n)=p_array(f(n));
  enddo
  !$omp end task
  MPI_ALLREDUCE(shift_p,sum_shift_p);
!$omp end master
!$omp end parallel
if(sum_shift_p==0) { return; }
```

Overlapping MPI_Allreduce with particle work

- **Overlap**: Master thread encounters (!$omp master) tasking statements and creates work for the thread team for deferred execution. MPI Allreduce call is immediately executed.
- MPI implementation has to support at least MPI_THREAD_FUNNELED
- Subdividing tasks into smaller chunks to allow better *load balancing* and *scalability* among threads.

Slides, courtesy of Alice Koniges, NERSC, LBNL



HLRS

Overlapping can be achieved with OpenMP tasks (2nd part)

```
!$omp parallel
!$omp master
  !$omp task
  fill_hole(p_array);
  !$omp end task

  MPI_SENDRECV(x,length=2,..);
  MPI_SENDRECV(sendright,length=g(x),..);
  MPI_SENDRECV(y,length=2,..);
!$omp end master
!$omp end parallel
}
```

Overlapping particle reordering

Particle reordering of remaining particles (above) and adding sent particles into array (right) & sending or receiving of shifted particles can be independently executed.

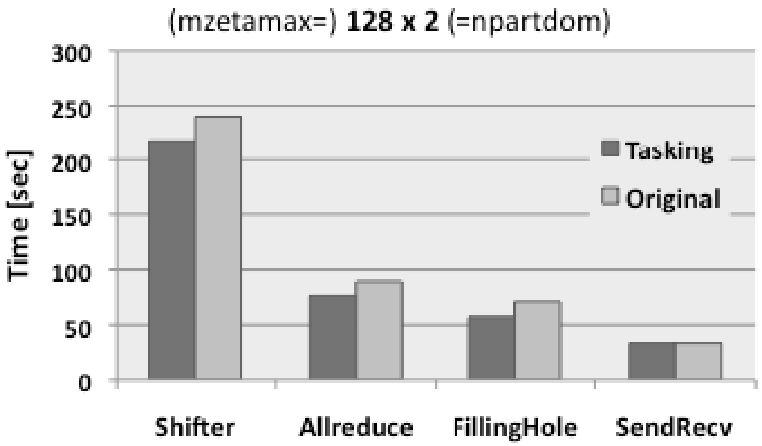
```
!$omp parallel
!$omp master
!adding shifted particles from right
   do m=1,x-stride,stride
     !$omp task
     do mm=0,stride-1,1
       p_array(h(m))=sendright(m);
     enddo
     !$omp end task
  enddo
  !$omp task
  do m=m,x
    p_array(h(m))=sendright(m);
  enddo
  !$omp end task

  MPI_SENDRECV(sendleft,length=g(y),..);
!$omp end master
!$omp end parallel

!adding shifted particles from left
!$omp parallel do
do n=1,y
  p_array(h(n))=sendleft(n);
enddo
```

Overlapping remaining MPI_Sendrecv

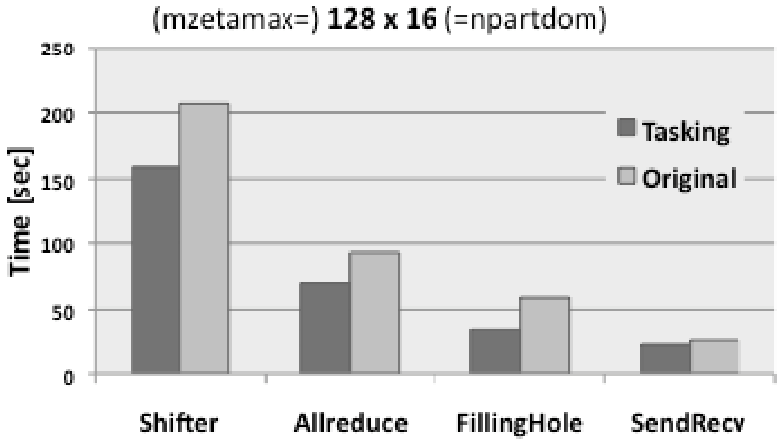
Slides, courtesy of Alice Koniges, NERSC, LBNL

OpenMP tasking version outperforms original shifter, especially in larger poloidal domains

256 size run



2048 size run



- Performance breakdown of GTS shifter routine using 4 OpenMP threads per MPI process with varying domain decomposition and particles per cell on Franklin Cray XT4.
- MPI communication in the shift phase uses a **toroidal MPI communicator** (constantly 128).
- Large performance differences in the 256 MPI run compared to 2048 MPI run!
- Speed-Up is expected to be higher on larger GTS runs with hundreds of thousands CPUs since MPI communication is more expensive.

H L R S

Slides, courtesy of Alice Koniges, NERSC, LBNL

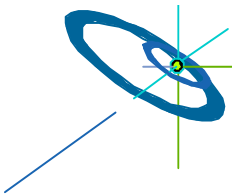
OpenMP only

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

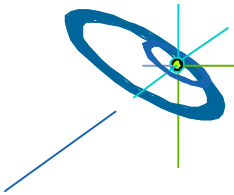


OpenMP only

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.



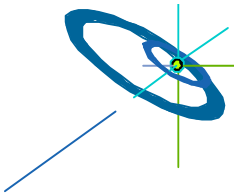
Courtesy of J. Cownie, Intel

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:

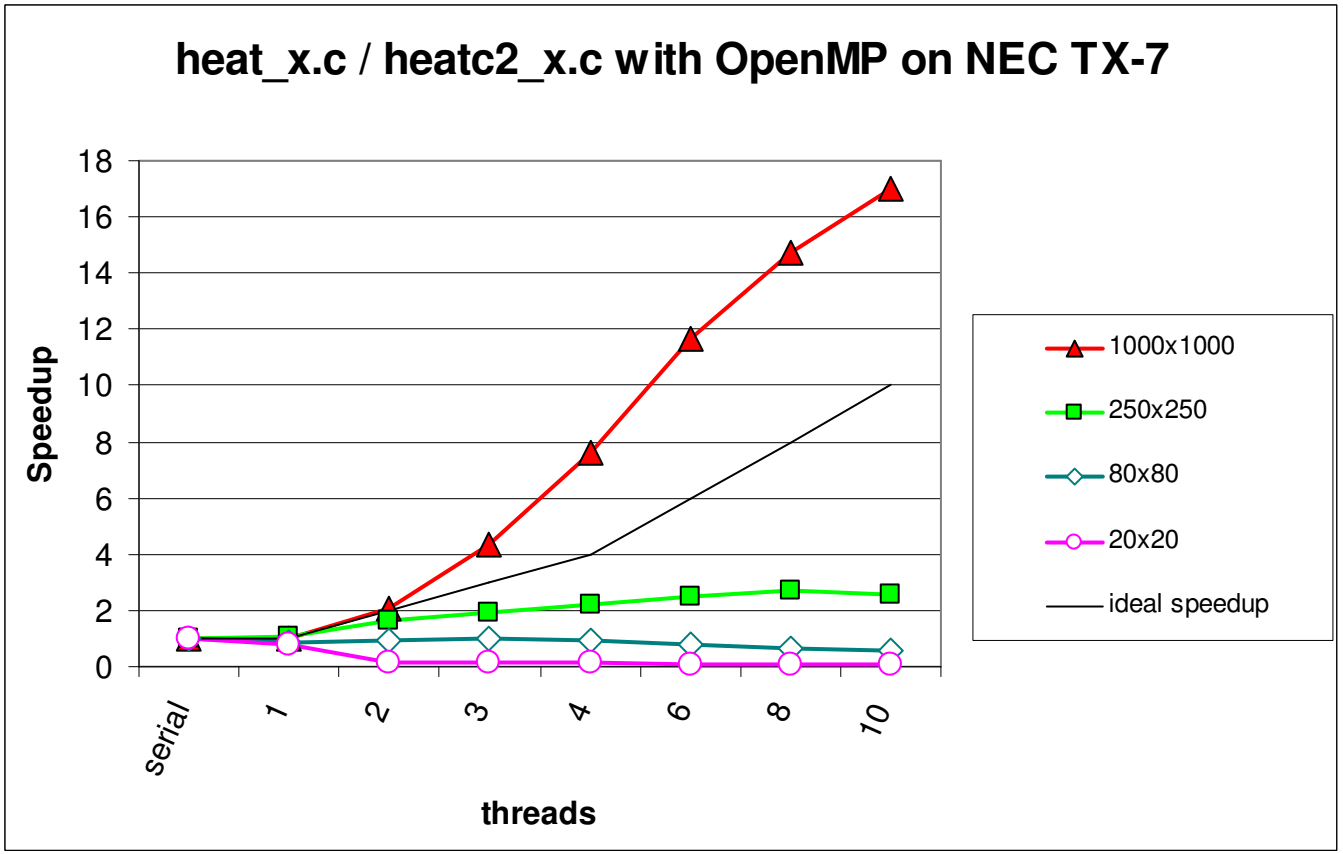
Communication may be 10 times slower than with MPI



skipped

Comparing results with heat example

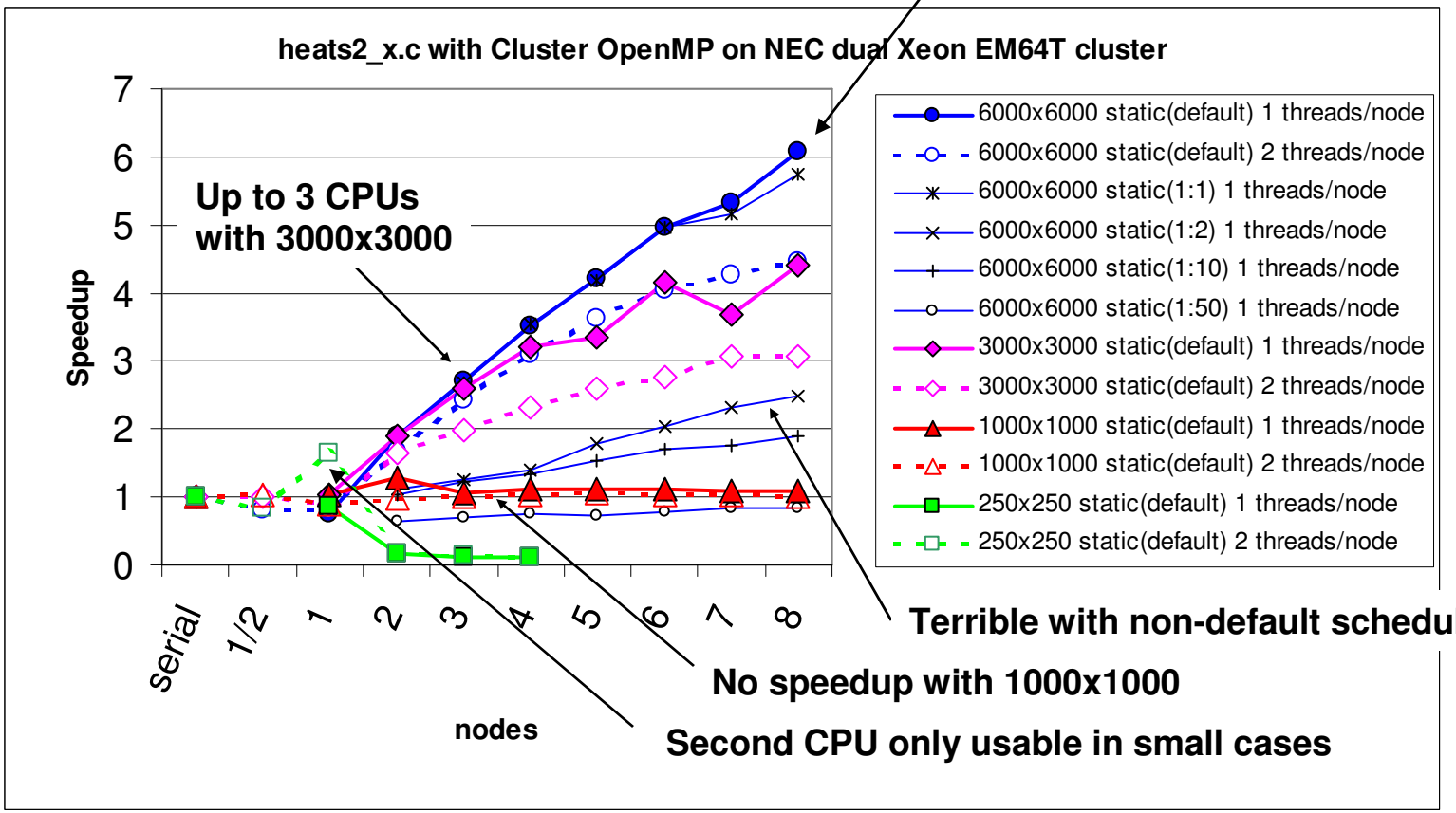
- Normal OpenMP on shared memory (ccNUMA) NEC TX-7



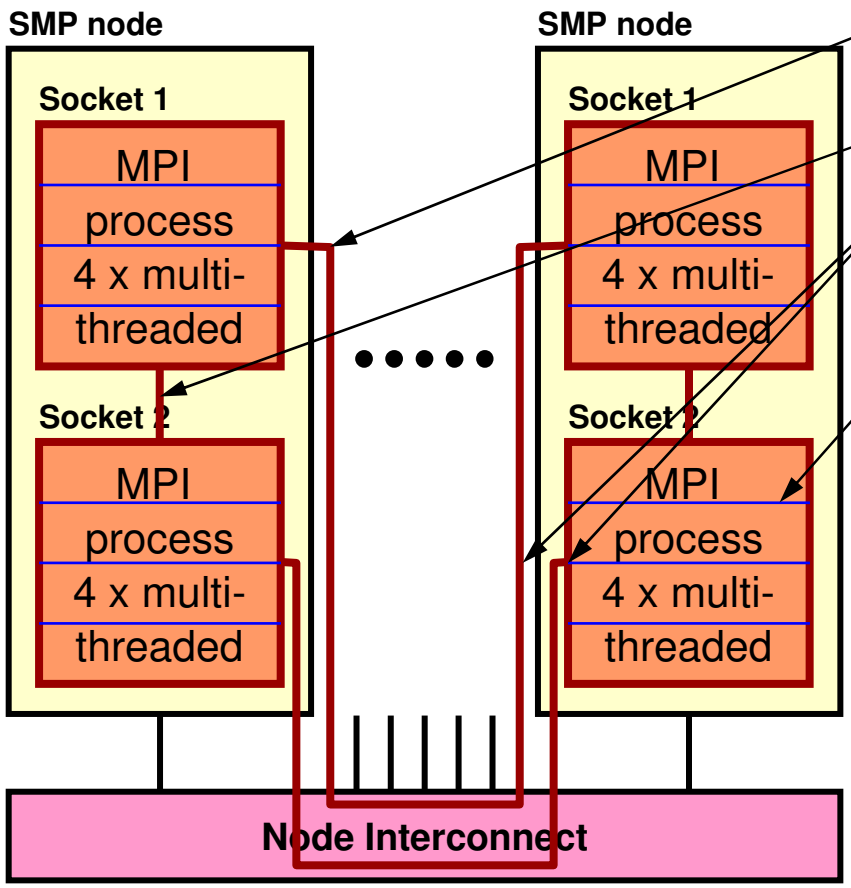
skipped

Heat example: Cluster OpenMP Efficiency

- Cluster OpenMP on a Dual-Xeon cluster



Back to the mixed model – an Example



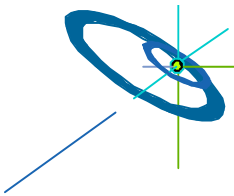
- Topology-problem solved:
  Only horizontal inter-node comm.
- Still intra-node communication
- Several threads per SMP node are communicating in parallel:
  → network saturation is possible
- Additional OpenMP overhead
- With Masteronly style:
  75% of the threads sleep while master thread communicates
- With Overlapping Comm.& Comp.:
  Master thread should be reserved for communication only partially – otherwise too expensive
- MPI library must support
  - Multiple threads
  - Two fabrics (shmem + internode)

No silver bullet

- The analyzed programming models do **not** fit on hybrid architectures
  - whether drawbacks are minor or major
    - **depends on applications' needs**
  - But there are major opportunities → next section

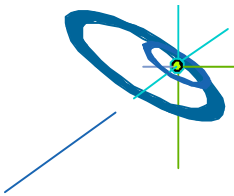
- In the NPB-MZ case-studies
  - We tried to use optimal parallel environment
    - **for pure MPI**
    - **for hybrid MPI+OpenMP**
  - i.e., the developers of the MZ codes and we tried to minimize the mismatch problems

→ the opportunities in next section dominated the comparisons



Outline

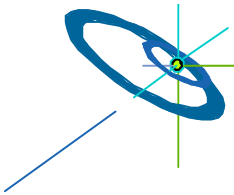
- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / Benchmark results
- Mismatch Problems
- **Opportunities:**
  Application categories that can benefit from hybrid parallelization
- Thread-safety quality of MPI libraries
- Other options on clusters of SMP nodes
- Summary



Nested Parallelism

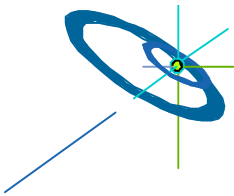
- Example NPB: BT-MZ (Block tridiagonal simulated CFD application)
  - Outer loop:
    - **limited number of zones → limited parallelism**
    - **zones with different workload → speedup < $\frac{\text{Sum of workload of all zones}}{\text{Max workload of a zone}}$**
  - Inner loop:
    - **OpenMP parallelized (static schedule)**
    - **Not suitable for distributed memory parallelization**

- Principles:
  - Limited parallelism on outer level
  - Additional inner level of parallelism
  - Inner level not suitable for MPI
  - Inner level may be suitable for static OpenMP worksharing



Load-Balancing (on same or different level of parallelism)

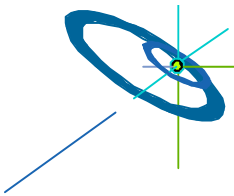
- OpenMP enables
  - Cheap ***dynamic*** and ***guided*** load-balancing
  - Just a parallelization option (clause on omp for / do directive)
  - Without additional software effort
  - Without explicit data movement
- On MPI level
  - **Dynamic load balancing** requires moving of parts of the data structure through the network
  - Significant runtime overhead
  - Complicated software / therefore not implemented
- **MPI & OpenMP**
  - Simple static load-balancing on MPI level, dynamic or guided on OpenMP level } **medium quality cheap implementation**



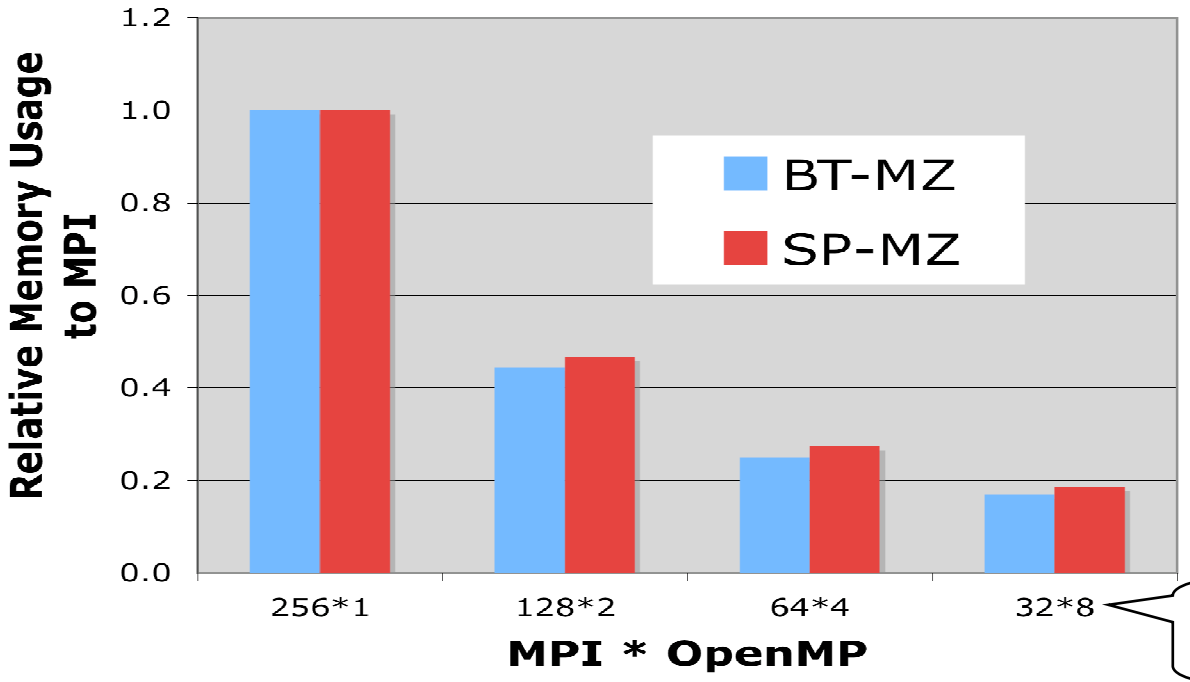
Memory consumption

- Shared nothing
  - Heroic theory
  - In practice: Some data is duplicated

- **MPI & OpenMP**
  With n threads per MPI process:
  - Duplicated data may be reduced by factor n



Case study: MPI+OpenMP memory usage of NPB



Using more OpenMP threads could reduce the memory usage **substantially**, up to **five** times on Hopper Cray XT5 (eight-core nodes).

Always same number of cores

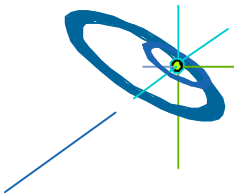
Hongzhang Shan, Haoqiang Jin, Karl Fuerlinger, Alice Koniges, Nicholas J. Wright: Analyzing the Effect of Different Programming Models Upon Performance and Memory Usage on Cray XT5 Platforms. Proceedings, CUG 2010, Edinburgh, GB, May 24-27, 2010.

H L R S

Slide, courtesy of Alice Koniges, NERSC, LBLN

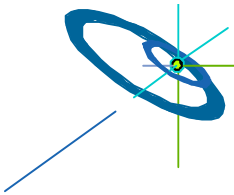
Memory consumption (continued)

- 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



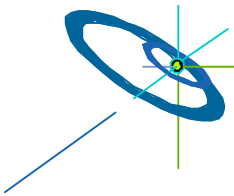
How many threads per MPI process?

- SMP node = with **m sockets** and **n cores/socket**
- How many threads (i.e., cores) per MPI process?
  - Too many threads per MPI process
    → overlapping of MPI and computation may be necessary,
    → some NICs unused?
  - Too few threads
    → too much memory consumption (see previous slides)
- Optimum
  - somewhere between 1 and m x n threads per MPI process,
  - Typically:
    - **Optimum = n, i.e., 1 MPI process per socket**
    - **Sometimes = n/2, i.e., 2 MPI processes per socket**
    - **Seldom = 2n, i.e., each MPI process on 2 sockets**



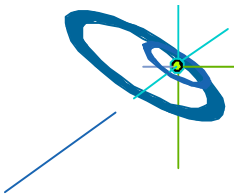
Opportunities, if MPI speedup is limited due to algorithmic problems

- Algorithmic opportunities due to larger physical domains inside of each MPI process
  - → If multigrid algorithm only inside of MPI processes
  - → If separate preconditioning inside of MPI nodes and between MPI nodes
  - → If MPI domain decomposition is based on physical zones



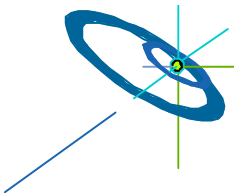
To overcome MPI scaling problems

- Reduced number of MPI messages, reduced aggregated message size } compared to pure MPI
- MPI has a few scaling problems
  - Handling of more than 10,000 MPI processes
  - Irregular Collectives: MPI_....v(), e.g. MPI_Gatherv()
    - **Scaling applications should not use MPI_....v() routines**
  - MPI-2.1 Graph topology (MPI_Graph_create)
    - **MPI-2.2 MPI_Dist_graph_create_adjacent**
  - Creation of sub-communicators with MPI_Comm_create
    - **MPI-2.2 introduces a new scaling meaning of MPI_Comm_create**
  - … see P. Balaji, et al.: **MPI on a Million Processors.** Proceedings EuroPVM/MPI 2009.
- Hybrid programming reduces all these problems (due to a smaller number of processes)



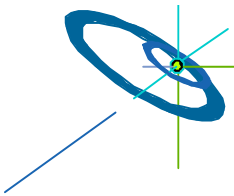
Summary: Opportunities of hybrid parallelization (MPI & OpenMP)

- Nested Parallelism
  → Outer loop with MPI / inner loop with OpenMP
- Load-Balancing
  → Using OpenMP ***dynamic*** and ***guided*** worksharing
- Memory consumption
  → Significantly reduction of replicated data on MPI level
- Opportunities, if MPI speedup is limited due to algorithmic problem
  → Significantly reduced number of MPI processes
- Reduced MPI scaling problems
  → Significantly reduced number of MPI processes



Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / Benchmark results
- Mismatch Problems
- Opportunities:
  Application categories that can benefit from hybrid parallelization
- **Thread-safety quality of MPI libraries**
- Other options on clusters of SMP nodes
- Summary



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:** **MPI processes may be multi-threaded,**
    (virtual value, **but only master thread will make MPI-calls**
    not part of the standard) **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

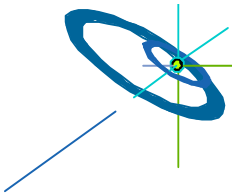
Calling MPI inside of OMP MASTER

- 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!

```
!$OMP BARRIER
!$OMP MASTER
        call MPI_Xxx(...)
!$OMP END MASTER
!$OMP BARRIER
```

```
#pragma omp barrier
#pragma omp master
        MPI_Xxx(...);

#pragma omp barrier
```

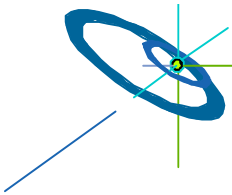
- But this implies that all other threads are sleeping!
- The additional barrier implies also the necessary cache flush!



… 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 */
```

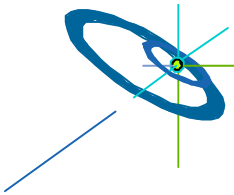


Thread support in MPI libraries

- The following MPI libraries offer thread support:

Implementation	Thread support level
MPIch-1.2.7p1	Always announces `MPI_THREAD_FUNNELED`.
MPIch2-1.0.8	ch3:sock supports `MPI_THREAD_MULTIPLE`
	ch:nemesis has “Initial Thread-support”
MPIch2-1.1.0a**2**	ch3:nemesis (default) has `MPI_THREAD_MULTIPLE`
Intel MPI 3.1	Full `MPI_THREAD_MULTIPLE`
SciCortex MPI	`MPI_THREAD_FUNNELED`
HP MPI-2.2.7	Full `MPI_THREAD_MULTIPLE (with libmtmpi)`
SGI MPT-1.14	Not thread-safe?
IBM MPI	Full `MPI_THREAD_MULTIPLE`
Nec MPI/SX	`MPI_THREAD_SERIALIZED`

- Testsuites for thread-safety may still discover bugs in the MPI libraries

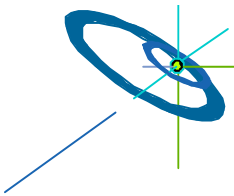


H L R S

Courtesy of Rainer Keller, HLRS and ORNL

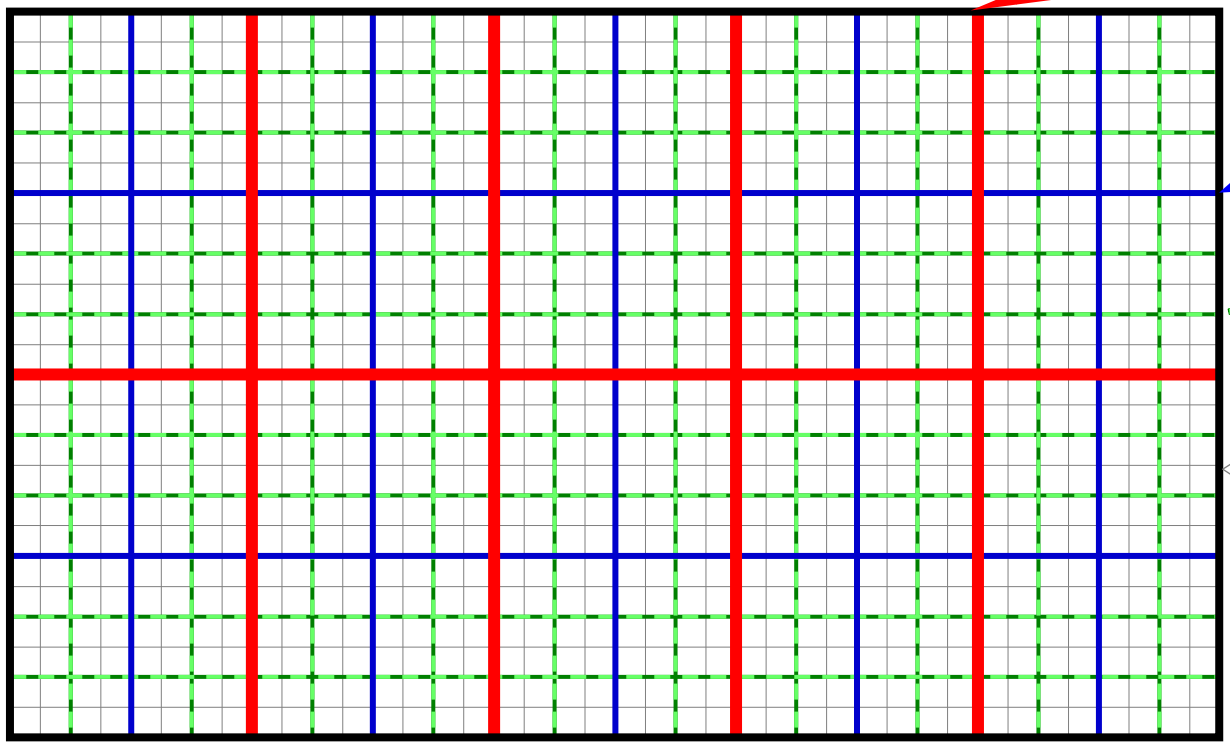
Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / Benchmark results
- Mismatch Problems
- Opportunities:
  Application categories that can benefit from hybrid parallelization
- Thread-safety quality of MPI libraries
- **Other options on clusters of SMP nodes**
- Summary



Pure MPI – multi-core aware

- **Hierarchical domain decomposition**
  (or distribution of Cartesian arrays)

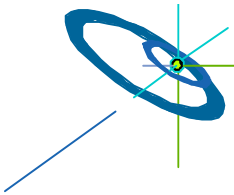


Example on 10 nodes, each with 4 sockets, each with 6 cores.

How to achieve a hierarchical domain decomposition (DD)?

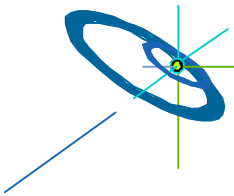
- **Cartesian grids:**
  - Several levels of subdivide
  - Ranking of MPI_COMM_WORLD – two choices:
    - **a) Sequential ranks through original data structure**
      + locating these ranks correctly on the hardware
      - ➢ can be achieved with one-level DD on finest grid
        + special startup (mpiexec) with optimized rank-mapping
    - **b) Sequential ranks in comm_cart (from MPI_CART_CREATE)**
      - ➢ requires optimized MPI_CART_CREATE,
        or special startup (mpiexec) with optimized rank-mapping
    - **c) Sequential ranks in MPI_COMM_WORLD**
      + additional communicator with sequential ranks in the data structure
      + self-written and optimized rank mapping.

- **Unstructured grids:**
  - → next slide



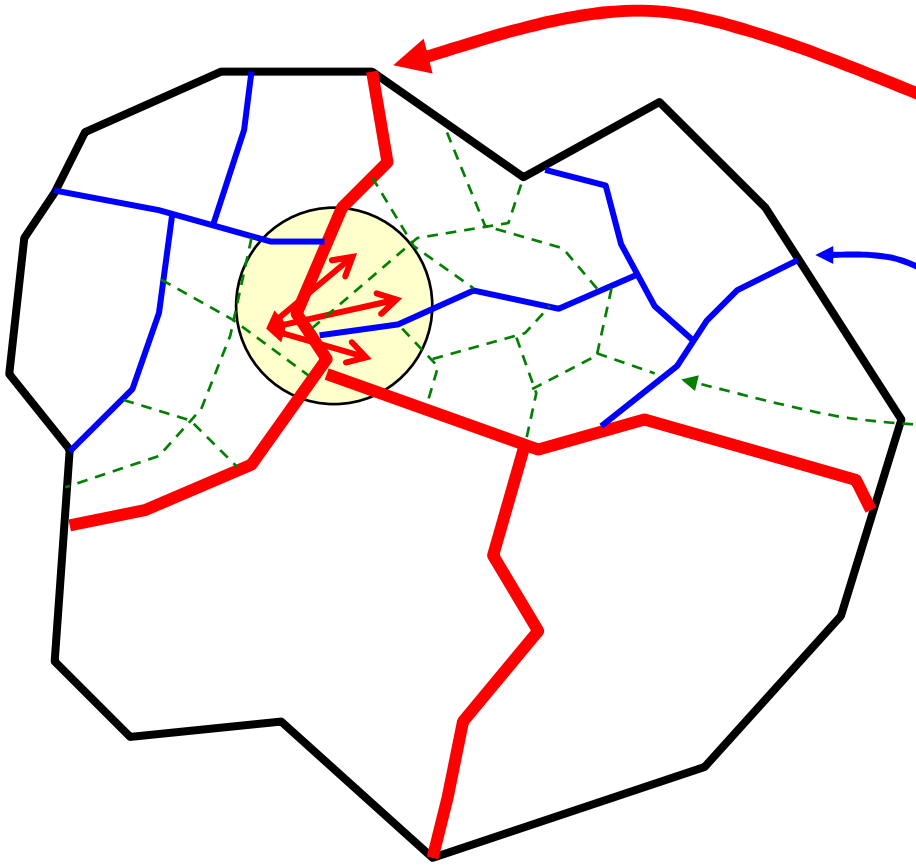
How to achieve a hierarchical domain decomposition (DD)?

- **Unstructured grids:**
  - Multi-level DD:
    - Top-down: Several levels of (Par)Metis
    - Bottom-up: Low level DD + higher level recombination
  - Single-level DD (finest level)
    - Analysis of the communication pattern in a first run (with only a few iterations)
    - Optimized rank mapping to the hardware before production run
    - E.g., with CrayPAT + CrayApprentice



skipped

Top-down – several levels of (Par)Metis



Steps:

- Load-balancing (e.g., with ParMetis) on outer level, i.e., between all SMP nodes
- Independent (Par)Metis inside of each node
- Metis inside of each socket

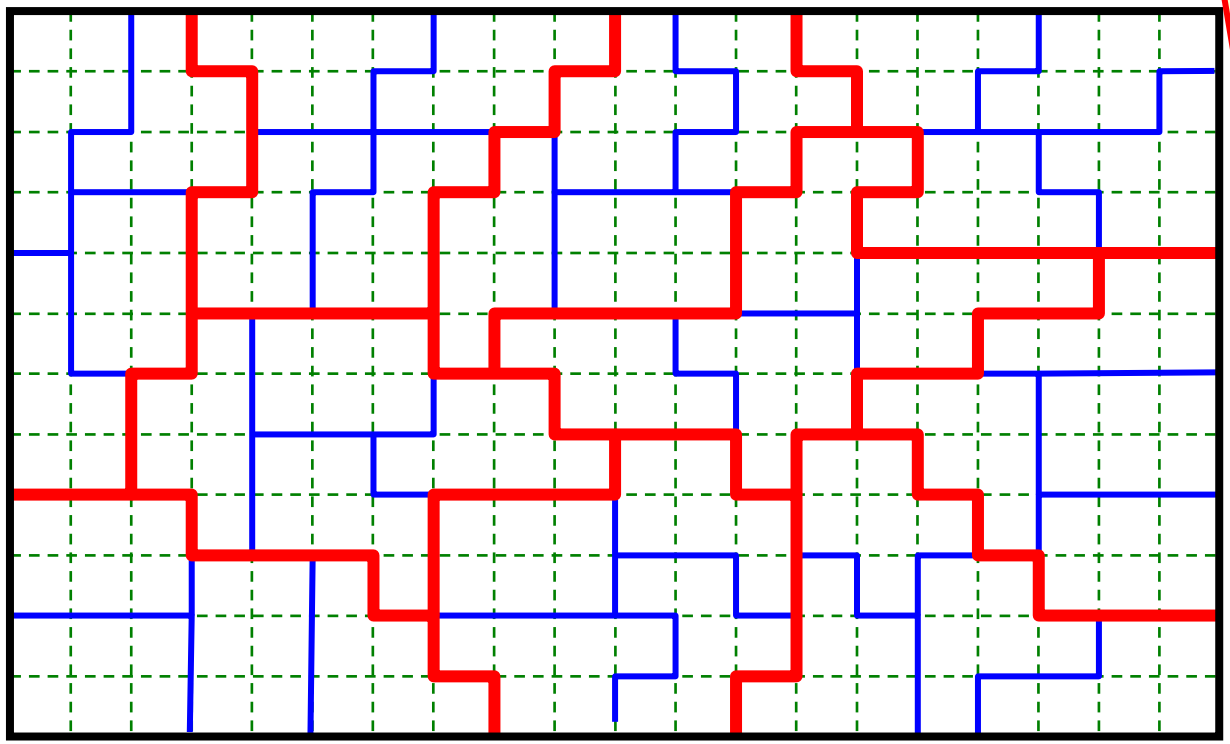
➢ Subdivide does not care on balancing of the outer boundary

➢ processes can get a lot of neighbors with inter-node communication

➢ unbalanced communication

Bottom-up – Multi-level DD through recombination

1. **Core-level DD:** partitioning of application's data grid
2. **Socket-level DD:** recombining of core-domains
3. **SMP node level DD:** recombining of socket-domains

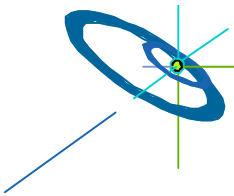


- **Problem:** Recombination must **not** calculate patches that are smaller or larger than the average
- In this example the load-balancer **must** combine always
  - 6 cores, and
  - 4 sockets
- **Advantage:** Communication is balanced!

Profiling solution

- First run with profiling
  - Analysis of the communication pattern
- Optimization step
  - Calculation of an optimal mapping of ranks in MPI_COMM_WORLD to the hardware grid (physical cores / sockets / SMP nodes)
- Restart of the application with this optimized locating of the ranks on the hardware grid

- Example: CrayPat and CrayApprentice



Scalability of MPI to hundreds of thousands …

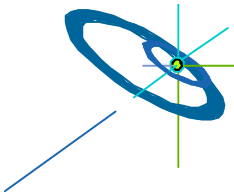
Weak scalability of pure MPI

- As long as the application does not use
  - MPI_ALLTOALL
  - MPI_<collectives>V (i.e., with length arrays)

  and application
  - distributes all data arrays

  one can expect:
  - Significant, but still scalable memory overhead for halo cells.
  - MPI library is internally scalable:
    - **E.g., mapping ranks → hardware grid**
      - Centralized storing in shared memory (OS level)
      - In each MPI process, only used neighbor ranks are stored (cached) in process-local memory.
    - **Tree based algorithm wiith O(log N)**
      - From 1000 to 1000,000 process O(Log N) only doubles!

The vendors will (or must) deliver scalable MPI libraries for their largest systems!



skipped

Remarks on Cache Optimization

- **After** all parallelization domain decompositions (DD, up to 3 levels) are done:
- Additional DD into data blocks
  - that fit to 2nd or 3rd level cache.
  - It is done inside of each MPI process (on each core).
  - Outer loops over these blocks
  - Inner loops inside of a block
  - Cartesian example: 3-dim loop is split into

```
do i_block=1,ni,stride_i
   do j_block=1,nj,stride_j
      do k_block=1,nk,stride_k
         do i=i_block,min(i_block+stride_i-1, ni)
            do j=j_block,min(j_block+stride_j-1, nj)
               do k=k_block,min(k_block+stride_k-1, nk)
                  a(i,j,k) = f( b(i±0,1,2, j±0,1,2, k±0,1,2) )
         … … … end do
end do
```

Access to 13-point stencil

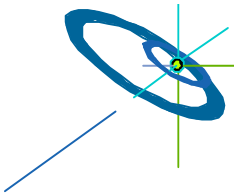
Remarks on Cost-Benefit Calculation

Costs

- for optimization effort
  - e.g., additional OpenMP parallelization
  - e.g., 3 person month x 5,000 € = 15,000 € (full costs)

Benefit

- from reduced CPU utilization
  - e.g., Example 1:
    100,000 € hardware costs of the cluster
    x 20% used by this application over whole lifetime of the cluster
    x 7% performance win through the optimization
    = 1,400 € → **total loss = 13,600 €**
  - e.g., Example 2:
    10 Mio € system x 5% used x 8% performance win
    = 40,000 € → **total win = 25,000 €**

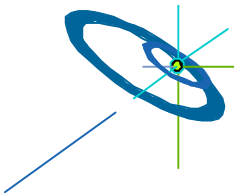


skipped

Remarks on MPI and PGAS (UPC & CAF)

- Parallelization always means
  - expressing locality.
- If the application has no locality,
  - Then the parallelization needs not to model locality

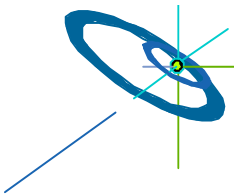
  → UPC with its round robin data distribution may fit
- If the application has locality,
  - then it must be expressed in the parallelization
- Coarray Fortran (CAF) expresses data locality explicitly through “co-dimension”:
  - A(17,15)[3]
    = element A(17,13) in the distributed array A in process with rank 3



skipped

Remarks on MPI and PGAS (UPC & CAF)

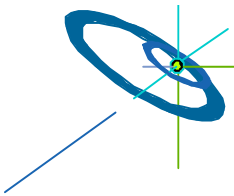
- Future shrinking of memory per core implies
  - Communication time becomes a bottleneck
  - → Computation and communication must be overlapped, i.e., latency hiding is needed
- With PGAS, halos are not needed.
  - But it is hard for the compiler to access data such early that the transfer can be overlapped with enough computation.
- With MPI, typically too large message chunks are transferred.
  - This problem also complicates overlapping.
- Strided transfer is expected to be slower than contiguous transfers
  - Typical packing strategies do not work for PGAS on compiler level
  - Only with MPI, or with explicit application programming with PGAS



skipped

Remarks on MPI and PGAS (UPC & CAF)

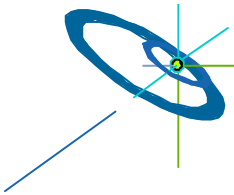
- Point-to-point neighbor communication
  - PGAS or MPI nonblocking may fit
    if message size makes sense for overlapping.
- Collective communication
  - Library routines are best optimized
  - Non-blocking collectives (comes with MPI-3.0)
    versus calling MPI from additional communication thread
  - Only blocking collectives in PGAS library?



skipped

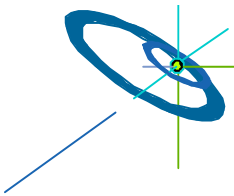
Remarks on MPI and PGAS (UPC & CAF)

- For extreme HPC (many nodes x many cores)
  - Most parallelization may still use MPI
  - Parts are optimized with PGAS, e.g., for better latency hiding
  - PGAS efficiency is less portable than MPI
  - #ifdef … PGAS
  - Requires mixed programming PGAS & MPI
    → will be addressed by MPI-3.0



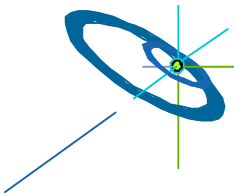
Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / Benchmark results
- Mismatch Problems
- Opportunities:
  Application categories that can benefit from hybrid parallelization
- Thread-safety quality of MPI libraries
- Other options on clusters of SMP nodes
- **Summary**



Acknowledgements

- We want to thank
  - Georg Hager, Gerhard Wellein, RRZE
  - Gabriele Jost, TACC
  - Alice Koniges, NERSC, LBNL
  - Rainer Keller, HLRS and ORNL
  - Jim Cownie, Intel
  - KOJAK project at JSC, Research Center Jülich
  - HPCMO Program and the Engineer Research and Development Center Major Shared Resource Center, Vicksburg, MS (http://www.erdc.hpc.mil/index)



Summary –  the good news

MPI + OpenMP

- Significant opportunity → higher performance on smaller number of threads
- Seen with NPB-MZ examples
  - BT-MZ → strong improvement (as expected)
  - SP-MZ → small improvement (none was expected)
- Usable on higher number of cores
- Advantages
  - Load balancing
  - Memory consumption
  - Two levels of parallelism
    - **Outer → distributed memory → halo data transfer → MPI**
    - **Inner → shared memory → ease of SMP parallelization → OpenMP**
- You can do it → "How To"

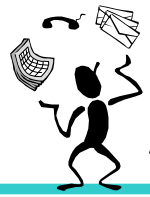
Summary – the bad news

MPI+OpenMP: There is a huge amount of pitfalls

- Pitfalls of MPI
- Pitfalls of OpenMP
  - On ccNUMA → e.g., first touch
  - Pinning of threads on cores
- Pitfalls through combination of MPI & OpenMP
  - E.g., topology and mapping problems
  - Many mismatch problems
- Tools are available ☺
  - It is not easier than analyzing pure MPI programs ☹
- Most hybrid programs → Masteronly style
- Overlapping communication and computation with several threads
  - Requires thread-safety quality of MPI library
  - Loss of OpenMP worksharing support → using OpenMP tasks as workaround

Summary – good and bad

- Optimization
  - 1 MPI process per core ..... mismatch problem ..... 1 MPI process per SMP node
    ^– somewhere between may be the optimum
- ☺ Efficiency of MPI+OpenMP is not for free:
  The efficiency strongly depends on
  ☹ the amount of work in the source code development

Summary –  Alternatives

Pure MPI

- \+ Ease of use
- – Topology and mapping problems may need to be solved
  (depends on loss of efficiency with these problems)
- – Number of cores may be more limited than with MPI+OpenMP
- \+ Good candidate for perfectly load-balanced applications

Pure OpenMP

- \+ Ease of use
- – Limited to problems with tiny communication footprint
- – source code modifications are necessary
  (Variables that are used with "*shared*" data scope must be allocated as "*sharable*")
- ± (Only) for the appropriate application a suitable tool

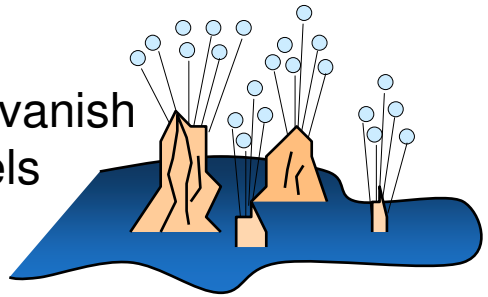
Summary

- This tutorial tried to
  - help to negotiate obstacles with hybrid parallelization,
  - give hints for the design of a hybrid parallelization,
  - and technical hints for the implementation → “How To”,
  - show tools if the application does not work as designed.

- This tutorial was not an introduction into other parallelization models:
  - Partitioned Global Address Space (PGAS) languages **(Unified Parallel C (UPC), Co-array Fortran (CAF), Chapel, Fortress, Titanium, and X10).**
  - High Performance Fortran (HPF)

  → Many rocks in the cluster-of-SMP-sea do not vanish into thin air by using new parallelization models

  → Area of interesting research in next years



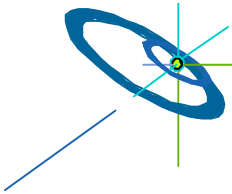
Conclusions

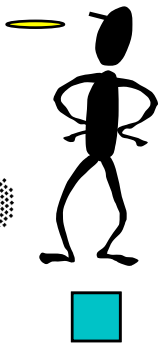
- Future hardware will be more complicated
  - Heterogeneous → GPU, FPGA, …
  - ccNUMA quality may be lost on cluster nodes
  - ….
- High-end programming → more complex
- Medium number of cores → more simple
  (if **#cores / SMP-node** will not shrink)
- MPI+OpenMP → work horse on large systems
- Pure MPI → still on smaller cluster
- OpenMP → on large ccNUMA nodes
  (**not ClusterOpenMP**)

Thank you for your interest

Q & A

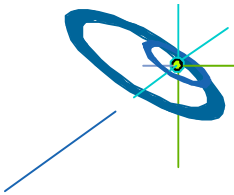
Please fill in the feedback sheet – Thank you





Appendix

- Author
- References (with direct relation to the content of this tutorial)
- Further references



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 and since Dec. 2007, he is in the steering committee of the MPI-3 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)

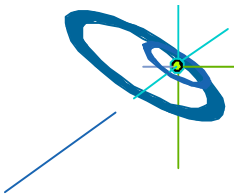
- **NAS Parallel Benchmarks**:
  http://www.nas.nasa.gov/Resources/Software/npb.html

- 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

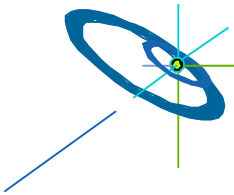
- G. Jost, H. Jin, D. an Mey and F. Hatay,
  Comparing OpenMP, MPI, and Hybrid Programming,
  Proc. Of the 5th European Workshop on OpenMP, 2003

- 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.

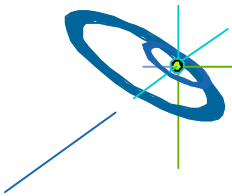


References

- 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.

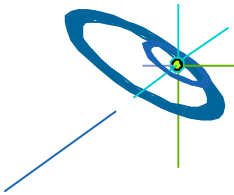
- Rolf Rabenseifner and Gerhard Wellein,
  Communication and Optimization Aspects of Parallel Programming Models on Hybrid Architectures.
  In proceedings of the Fourth European Workshop on OpenMP (EWOMP 2002), Roma, Italy, Sep. 18-20th, 2002.

- 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.



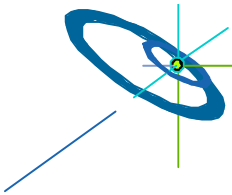
References

- Georg Hager and Gerhard Wellein:
  Introduction to High Performance Computing for Scientists and Engineers.
  CRC Press, to appear in July 2010, ISBN 978-1439811924.
- 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.
- Barbara Chapman, Gabriele Jost, and Ruud van der Pas:
  Using OpenMP.
  The MIT Press, 2008.
- Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, Sameer Kumar, Ewing Lusk, Rajeev Thakur and Jesper Larsson Traeff:
  MPI on a Million Processors.
  EuroPVM/MPI 2009, Springer.
- Alice Koniges et al.: **Application Acceleration on Current and Future Cray Platforms.**
  Proceedings, CUG 2010, Edinburgh, GB, May 24-27, 2010.
- H. Shan, H. Jin, K. Fuerlinger, A. Koniges, N. J. Wright: **Analyzing the Effect of Different Programming Models Upon Performance and Memory Usage on Cray XT5 Platorms**. Proceedings, CUG 2010, Edinburgh, GB, May 24-27, 2010.



Further references

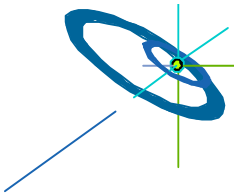
- Sergio Briguglio, Beniamino Di Martino, Giuliana Fogaccia and Gregorio Vlad, **Hierarchical MPI+OpenMP implementation of parallel PIC applications on clusters of Symmetric MultiProcessors**, 10th European PVM/MPI Users' Group Conference (EuroPVM/MPI'03), Venice, Italy, 29 Sep - 2 Oct, 2003
- Barbara 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



H L R S

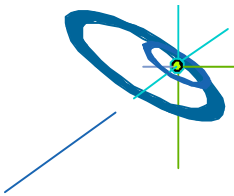
Further references

- 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/



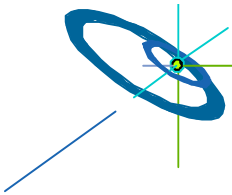
Further references

- 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



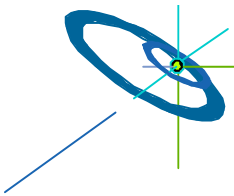
Further references

- 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