Hybrid MPI & OpenMP Parallel Programming

MPI + OpenMP and other models on clusters of SMP nodes

Rolf Rabenseifner[1)]
Rabenseifner@hlrs.de

Georg Hager[2)]
Georg.Hager@rrze.uni-erlangen.de

Gabriele Jost[3)]
gjost@tacc.utexas.edu

1) High Performance Computing Center (HLRS), University of Stuttgart, Germany
2) Regional Computing Center (RRZE), University of Erlangen, Germany
3) Texas Advanced Computing Center, The University of Texas at Austin, USA

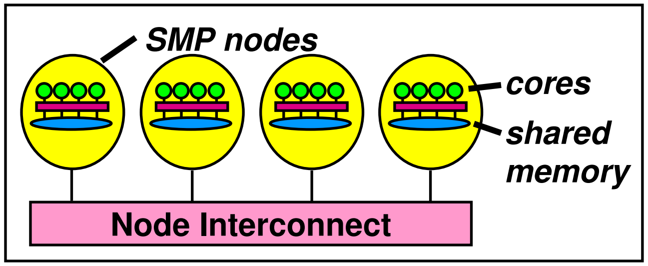
Tutorial S01 at SC11,
November 13, 2011, New Orleans, LA, USA

Outline

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**

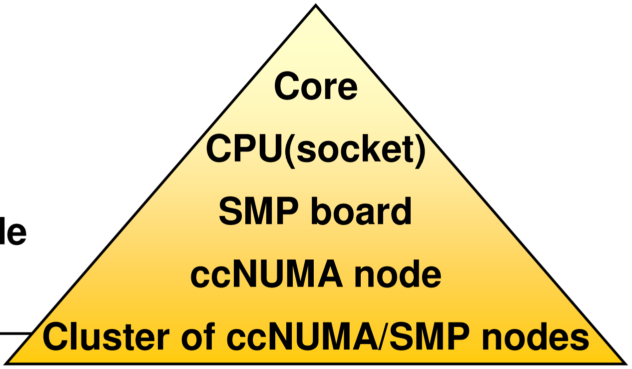


- Hardware range
  - **mini-cluster with dual-core CPUs**
  - **...**
  - **large constellations with large SMP nodes**

    ... with several sockets (CPUs) per SMP node

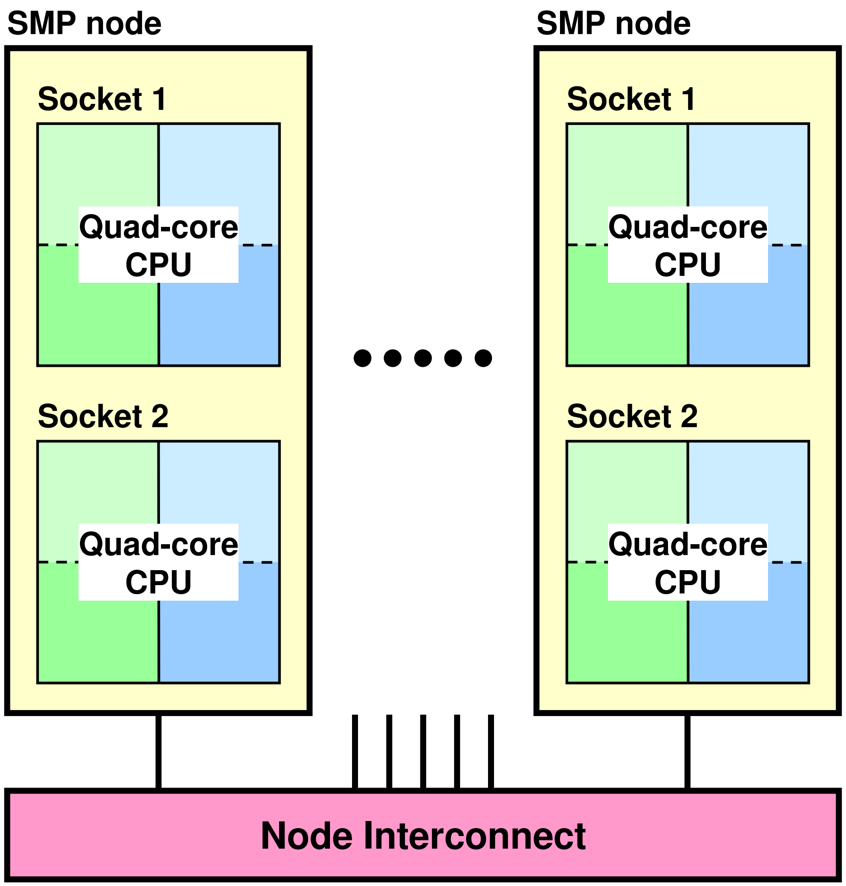
    ... with several cores per socket

  → Hierarchical system layout

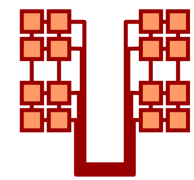


- Hybrid MPI/OpenMP programming seems natural
  - **MPI between the nodes**
  - **OpenMP inside of each SMP node**

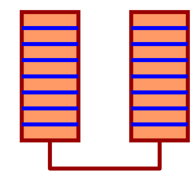
Motivation



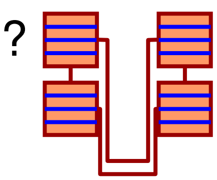
- Which programming model is fastest?
- MPI everywhere?



- 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

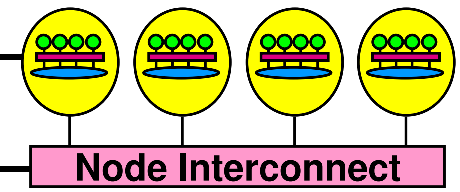
- Introduction / Motivation
- **Programming models on clusters of SMP nodes**
- Case Studies / pure MPI vs hybrid MPI+OpenMP
- Practical “How-To” on hybrid programming
- Mismatch Problems
- Opportunities:
  Application categories that can benefit from hybrid parallelization
- Thread-safety quality of MPI libraries
- Tools for debugging and profiling MPI+OpenMP
- 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

OpenMP inside of the SMP nodes

MPI between the nodes via node interconnect

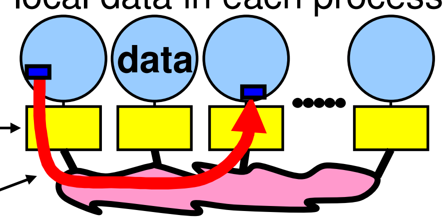


- Other: Virtual shared memory systems, PGAS, HPF, …
- Often **hybrid programming (MPI+OpenMP)** slower than **pure MPI**
  - why?

MPI

local data in each process

Sequential program on each core



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

OpenMP (shared data)

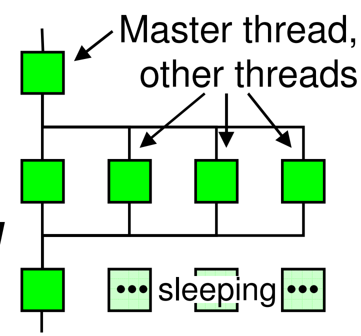
some_serial_code

#pragma omp parallel for

for (j=…;…; j++)

block_to_be_parallelized

again_some_serial_code



Parallel Programming Models on Hybrid Platforms

- **pure MPI**
  one MPI process
  on each core
- **hybrid MPI+OpenMP**
  MPI: inter-node communication
  OpenMP: inside of each SMP node
  - **<u>No overlap</u> of Comm. + Comp.**
    MPI only outside of parallel regions
    of the numerical application code
    - **Masteronly**
      MPI only outside
      of parallel regions
  - **<u>Overlapping</u> Comm. + Comp.**
    MPI communication by one or a few threads
    while other threads are computing
- **OpenMP only**
  distributed virtual
  shared memory

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

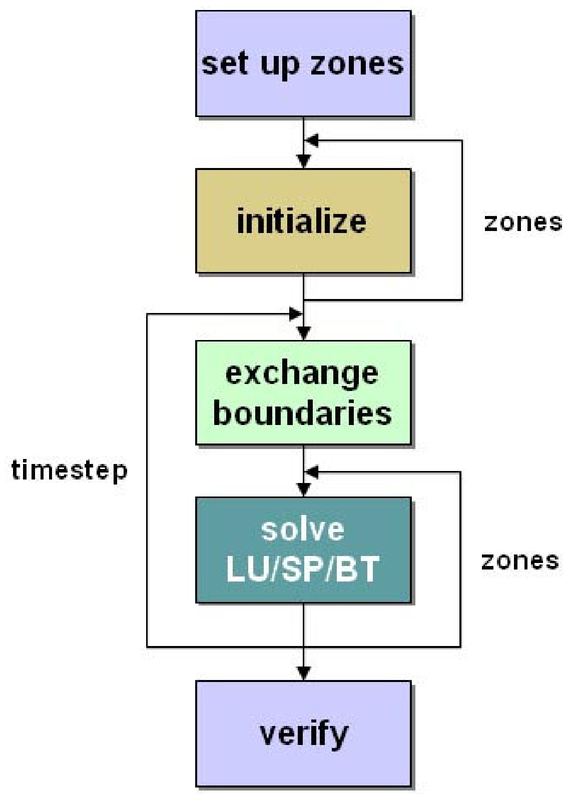
i.e., the OpenMP memory and parallelization model
is prepared for clusters!

Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- **Case Studies / pure MPI vs hybrid MPI+OpenMP**
  - The Multi-Zone NAS Parallel Benchmarks
  - For each application we discuss:
    - **Benchmark implementations based on different strategies and programming paradigms**
    - **Performance results and analysis on different hardware architectures**
  - Compilation and Execution Summary

  Gabriele Jost (University of Texas,TACC/Naval Postgraduate School, Monterey CA)
- Practical “How-To” on hybrid programming
- Mismatch Problems
- Opportunities: Application categories that can benefit from hybrid paralleli.
- Thread-safety quality of MPI libraries
- Tools for debugging and profiling MPI+OpenMP
- Other options on clusters of SMP nodes
- Summary

The Multi-Zone NAS Parallel Benchmarks



	MPI/OpenMP	MLP	Nested OpenMP
Time step	sequential	sequential	sequential
inter-zones	MPI Processes	MLP Processes	OpenMP
exchange boundaries	Call MPI	data copy+ sync.	OpenMP
intra-zones	OpenMP	OpenMP	OpenMP

- Multi-zone versions of the NAS Parallel Benchmarks LU,SP, and BT
- Two hybrid sample implementations
- Load balance heuristics part of sample codes
- www.nas.nasa.gov/Resources/Software/software.html

MPI/OpenMP BT-MZ

```
call omp_set_numthreads (weight)
do step = 1, itmax
  call exch_qbc(u, qbc, nx,…)

        call mpi_send/recv

do zone = 1, num_zones
    if (iam .eq. pzone_id(zone)) then
        call zsolve(u,rsd,…)
      end if
    end do

end do
  ...
```

```
 subroutine zsolve(u, rsd,…)
  ...
!$OMP PARALLEL DEFAUL(SHARED)
!$OMP& PRIVATE(m,i,j,k...)
  do k = 2, nz-1
!$OMP DO
    do j = 2, ny-1
      do i = 2, nx-1
        do m = 1, 5
           u(m,i,j,k)=
             dt*rsd(m,i,j,k-1)
        end do
      end do
    end do
!$OMP END DO nowait
  end do
  ...
!$OMP END PARALLEL
```

MPI/OpenMP LU-MZ

```
call omp_set_numthreads (weight)
do step = 1, itmax
  call exch_qbc(u, qbc, nx,…)



do zone = 1, num_zones
    if (iam .eq. pzone_id(zone)) then
        call ssor
      end if
    end do

end do
  ...
```

Pipelined Thread Execution in SSOR

```
  subroutine  ssor
!$OMP PARALLEL DEFAULT(SHARED)
!$OMP& PRIVATE(m,i,j,k...)
   call sync1 ()
   do k = 2, nz-1
!$OMP DO
     do j = 2, ny-1
       do i = 2, nx-1
         do m = 1, 5
      rsd(m,i,j,k)=
        dt*rsd(m,i,j,k-1) + …
         end do
       end do
     end do
!$OMP END DO nowait
   end do
   call sync2 ()
   ...
!$OMP END PARALLEL
   ...
```

```
subbroutine sync1
 …neigh = iam -1
 do while (isync(neigh) .eq. 0)
!$OMP FLUSH(isync)
 end do
 isync(neigh) = 0
!$OMP FLUSH(isync)
 …
 subroutine sync2
 …
 neigh = iam -1
 do while (isync(neigh) .eq. 1)
!$OMP FLUSH(isync)
 end do
 isync(neigh) = 1
!$OMP FLUSH(isync)
```

Golden Rule for ccNUMA: “First touch”

- A memory page gets mapped into the local memory of the processor that first touches it!
- Caveats:
  - possibly not enough local memory
  - "touch" means "write", not "allocate"

```
c-----------------------------------------------------------------------
c      do one time step to touch all data c----------------------------
-----------------------------------------
       do iz = 1, proc_num_zones
         zone = proc_zone_id(iz)
         call adi(rho_i(start1(iz)), us(start1(iz)),
     $             vs(start1(iz)), ws(start1(iz)
            ….. 
     $ end do
       do iz = 1, proc_num_zones
         zone = proc_zone_id(iz)
         call initialize(u(start5(iz)),…
     $ end do
```

All benchmarks use first-touch policy to achieve good memory placement!

Benchmark Characteristics

- 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

Benchmark Architectures

- Dell Linux Cluster Lonestar
- Cray XE6
- IBM Power 6

Hybrid code on cc-NUMA architectures

- **OpenMP**:
  - Support only per MPI process
  - Version 3.0 does not provide support to control to map threads onto CPUs. Support to specify thread placement is still under discussion.
  - Version 3.1 has support for binding of threads via OMP_PROC_BIND environmentvariable
- **MPI:**
  - Initially not designed for NUMA architectures or mixing of threads and processes, MPI-2 supports threads in MPI
  - API does not provide support for memory/thread placement
- **Vendor specific APIs to control thread and memory placement:**
  - Environment variables
  - System commands like *numactl,taskset,dplace,omplace etc*

  → *http://www.halobates.de/numaapi3.pdf*

  → *More in "How-to's"*

Dell Linux Cluster Lonestar

- Located at the Texas Advanced Computing Center (TACC), University of Texas at Austin (http://www.tacc.utexas.edu)
- 1888 nodes, 2 Xeon Intel 6-Core 64-bit Westmere processors, 3.33 GHz, 24 GB memory per node, Peak Performance 160 Gflops per node, 3 channels from each processor's memory controller to 3 DDR3 ECC DIMMS, 1333 MHz,
- Processor interconnect, QPI, 6.4GT/s
- Node Interconnect: InfiniBand Mellanox Switches, fat-tree topology, 40Gbit/sec point-to-point bandwidth
- More details: http://www.tacc.utexas.edu/user-services/user-guides/lonestar-user-guide
- Compiling the benchmarks: I
  - fort 11.1, Options: -O3 –ipo –openmp –mcmodel=medium
- Running the benchmarks:
  - MVAPICH 2
  - setenv OMP_NUM_THREADS=
  - ibrun tacc_affinity ./bt-mz.x

Example run script

```
#!/bin/csh
#$  -cwd
#$  -j y
#$  -q systest
#$  -pe  12way  24
#$  -V
#$  -l  h_rt=00:10:00
setenv OMP_NUM_THREADS 1
setenv MY_NSLOTS 16
ibrun tacc_affinity ./bin/sp-mz.D.
```

Run 12 MPI processes per node, allocate 24 cores (2nodes) alltogether

1 thread per MPI process

Only use 16 of the 24 cores for MPI. NOTE: 8 cores unused!!!

numactl script for process/thread placement

Command to run mpi job

NUMA Operations

	cmd	option	arguments	description
Socket Affinity	numactl	-c	{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).

Rabenseifner, Hager, Jost

NUMA Operations: Memory Placement

Memory allocation:

- MPI
  - Pure MPI: socket local allocation is best
  - Hybrid: Depending on #threads per process remote socket memory may be required
- OpenMP
  - Regular structured access patter that does not change: Allocate close to core where thread runs
  - Irregular, unpredictable access: Round-robin placement of pages
- Once allocated, a memory-structure is fixed

Example: `numactl –c 1 -l ./a.out`
Use socket 1, allocate memory on current socket

Example numactl script

```
myway=`echo $PE | sed s/way//`
export MV2_USE_AFFINITY=0
export MV2_ENABLE_AFFINITY=0
my_rank=$PMI_RANK
local_rank=$(( my_rank % myway ))
if [ $myway -eq 12 ]; then
  numnode=$(( local_rank / 6 ))
fi
exec numactl -c $numnode -m $numnode $*
```

Pitfall (1): Running 2 threads on the same core

Running NPB BT-MZ Class D 128 MPI Procs, 12 threads each, 1 MPI per node (1way)

Pinning A:

```
exec numactl –c 0  –m 0 $*
```

Only use cores and memory on socket 0, 12 threads on 6 cores

Running 128 MPI Procs, 12 threads each

Pinning B:

```
exec numactl –c 0,1 –m 0,1 $*
```

N

Use cores and memory on 2 sockets

Dell Linux Cluster Lonestar Topology

```
*********************************************************************
Socket 0:
+-----------------------------------------------------------------------+
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| |    1    | |    3    | |    5    | |    7    | |    9    | |   11    | |
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| |   32kB  | |   32kB  | |   32kB  | |   32kB  | |   32kB  | |   32kB  | |
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| |  256kB  | |  256kB  | |  256kB  | |  256kB  | |  256kB  | |  256kB  | |
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| +---------------------------------------------------------------------+ |
| |                                12MB                                 | |
| +---------------------------------------------------------------------+ |
+-----------------------------------------------------------------------+
Socket 1:
+-----------------------------------------------------------------------+
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| |    0    | |    2    | |    4    | |    6    | |    8    | |   10    | |
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| |   32kB  | |   32kB  | |   32kB  | |   32kB  | |   32kB  | |   32kB  | |
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| |  256kB  | |  256kB  | |  256kB  | |  256kB  | |  256kB  | |  256kB  | |
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| +---------------------------------------------------------------------+ |
| |                                12MB                                 | |
| +---------------------------------------------------------------------+ |
+-----------------------------------------------------------------------+
```

Pitfall (2): Cause remote memory access

Running NPB BT-MZ Class D 128 MPI Procs, 6 threads each 2 MPI per node

Pinning A:

```
if [ $localrank == 0 ]; then
exec numactl --physcpubind=0,1,2,3,4,5 -m 0 $*
elif [ $localrank == 1 ]; then
exec numactl --physcpubind=6,7,8,9,10,11 -m 1 $*
fi
```

600 Gflops

Half of the threads access remote memory

Running 128 MPI Procs, 6 threads each

Pinning B:

```
if [ $localrank == 0 ]; then
exec numactl --physcpubind=0,2,4,6,8,10 -m 0 $*
elif [ $localrank == 1 ]; then
exec numactl -physcpubind=1,3,5,7,9,11 -m 1 $*
fi
```

900 Gflops

Only local memory access

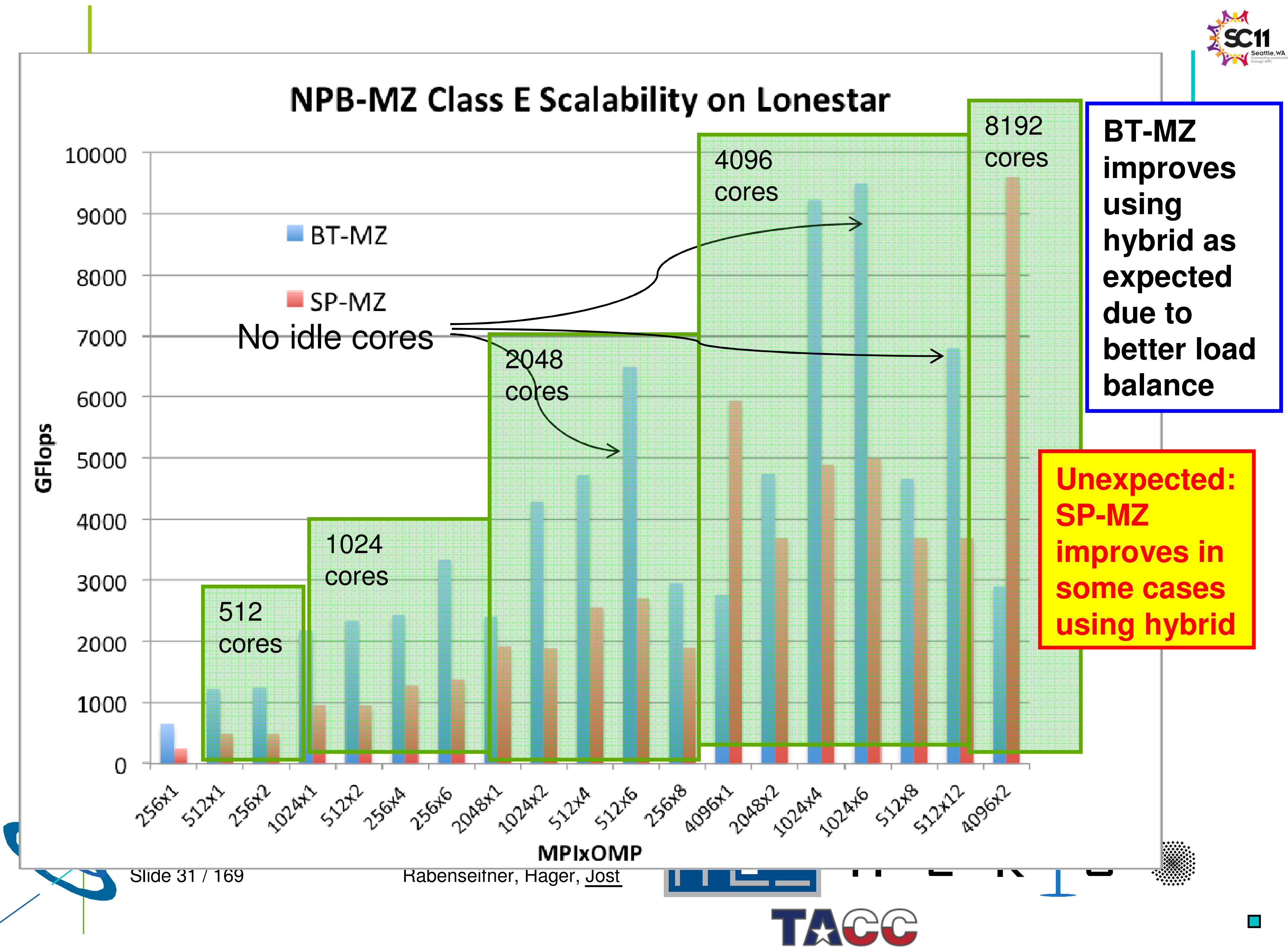
N

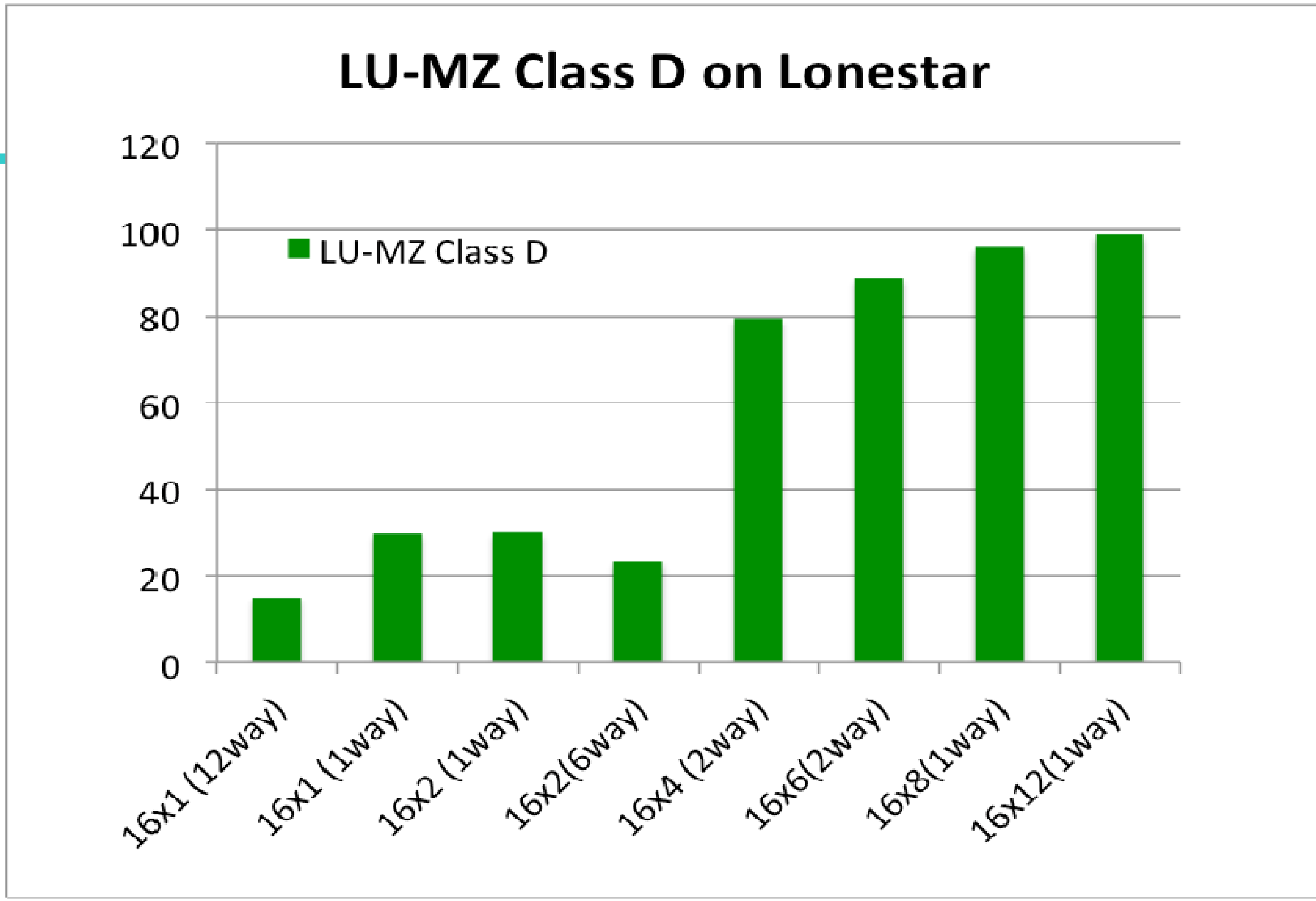
r:
st

Dell Linux Cluster Lonestar Topology

```
CPU type: Intel Core
   Westmere processor
***************************
   *********
Hardware Thread Topology
***************************
   *********
Sockets:                  2
Cores per socket:         6
Threads per core:         1

---------------------------------
Socket 0: ( 1 3 5 7 9 11 )
Socket 1: ( 0 2 4 6 8 10 )
---------------------------------
```

Careful!
Numbering scheme of cores is system dependent


NPB-MZ Class E Scalability on Lonestar
GFlops
MPIxOMP
BT-MZ
SP-MZ
No idle cores
512 cores
1024 cores
2048 cores
4096 cores
8192 cores
256x1
512x1
256x2
1024x1
512x2
256x4
256x6
2048x1
1024x2
512x4
512x6
256x8
4096x1
2048x2
1024x4
1024x6
512x8
512x12
4096x2
BT-MZ improves using hybrid as expected due to better load balance
Unexpected: SP-MZ improves in some cases using hybrid
TACC



- LU-MZ significantly benefits from hybrid mode:
  - Pure MPI limited to 16 cores, due to #zones = 16
- Decrease of resource contention large contribution to improvement

Cray XE6 Hector

- Located at EPCC, Edinburgh, Scotland, UK National Supercomputing Services, Hector Phase 2b (http://www.hector.ac.uk)
- 1856 XE6 compute nodes.
- Each node contains two AMD 2.1 GHz 12-core processors giving a total of 44,544 cores
- Around 373 Tflops theoretical peak performance
- 32 GB of main memory available per node
- 24-way shared memory system.
- High-bandwidth interconnect using Cray Gemini communication chips.

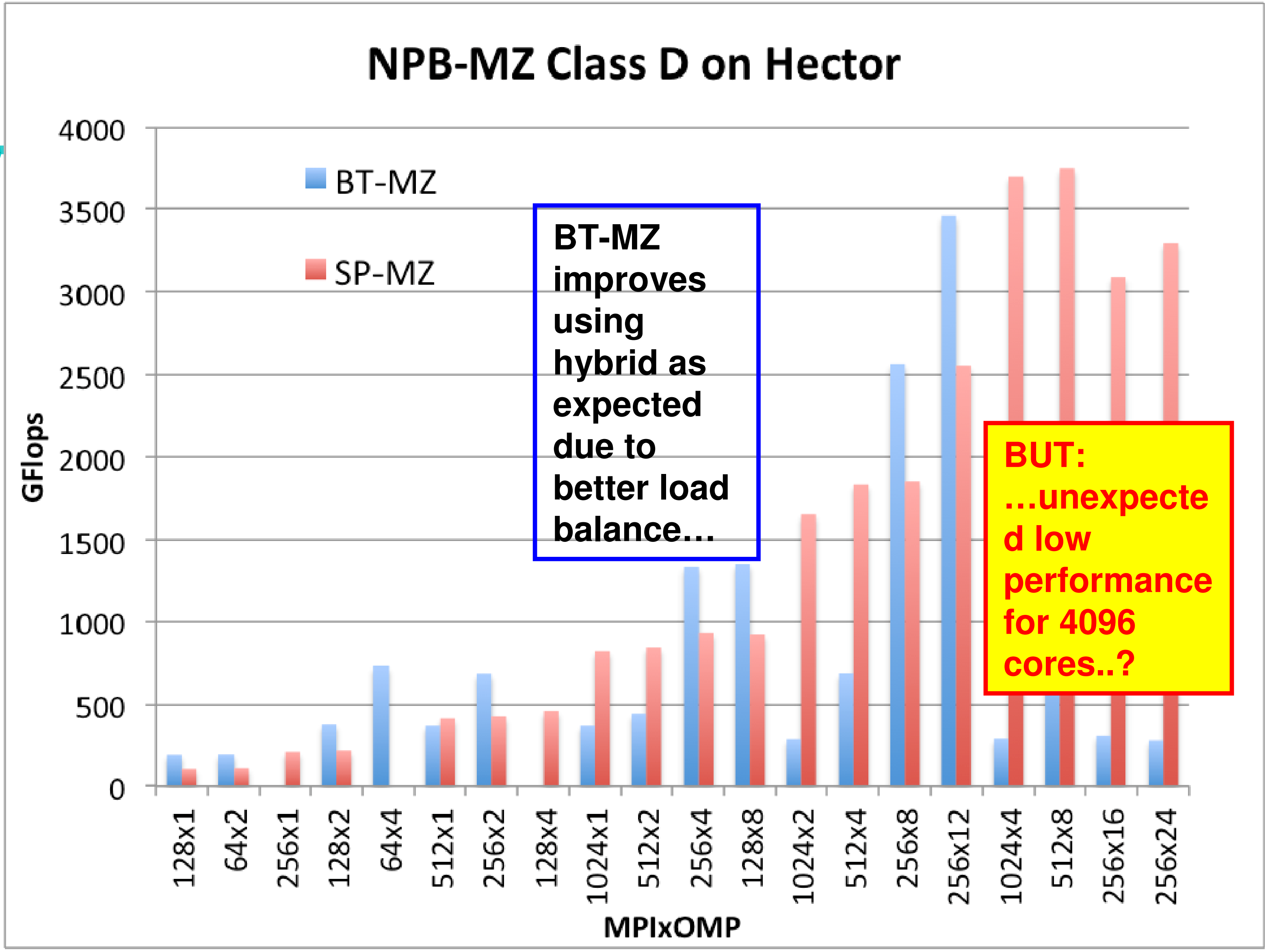
CPU type: AMD Magny Cours processor
Hardware Thread Topology
Sockets: 2
Cores per socket: 12
Threads per core: 1

no SMT

Cray XE6 Hector Node Topology

4 NUMA domains

```
********************************************************************
Graphical:
********************************************************************
Socket 0:
+-------------------------------------------------------------------------------------------------------------------+
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| |   0    | |   1    | |   2    | |   3    | |   4    | |   5    | |   6    | |   7    | |   8    | |   9    | |   10   | |   11   | |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| |  64kB  | |  64kB  | |  64kB  | |  64kB  | |  64kB  | |  64kB  | |  64kB  | |  64kB  | |  64kB  | |  64kB  | |  64kB  | |  64kB  | |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| | 512kB  | | 512kB  | | 512kB  | | 512kB  | | 512kB  | | 512kB  | | 512kB  | | 512kB  | | 512kB  | | 512kB  | | 512kB  | | 512kB  | |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| +-------------------------------------------------------------+ +-------------------------------------------------------------+ |
| |                             5MB                             | |                             5MB                             | |
| +-------------------------------------------------------------+ +-------------------------------------------------------------+ |
+-------------------------------------------------------------------------------------------------------------------+
Socket 1:
+-------------------------------------------------------------------------------------------------------------------+
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| |   12   | |   18   | |   19   | |   20   | |   21   | |   22   | |   23   | |   13   | |   14   | |   15   | |   16   | |   17   | |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| |  64kB  | |  64kB  | |  64kB  | |  64kB  | |  64kB  | |  64kB  | |  64kB  | |  64kB  | |  64kB  | |  64kB  | |  64kB  | |  64kB  | |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| | 512kB  | | 512kB  | | 512kB  | | 512kB  | | 512kB  | | 512kB  | | 512kB  | | 512kB  | | 512kB  | | 512kB  | | 512kB  | | 512kB  | |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| +-------------------------------------------------------------+ +-------------------------------------------------------------+ |
| |                             5MB                             | |                             5MB                             | |
| +-------------------------------------------------------------+ +-------------------------------------------------------------+ |
+-------------------------------------------------------------------------------------------------------------------+
```


NPB-MZ Class D on Hector
GFlops
4000
3500
3000
2500
2000
1500
1000
500
0
BT-MZ
SP-MZ
BT-MZ improves using hybrid as expected due to better load balance…
BUT: …unexpected low performance for 4096 cores..?
128x1
64x2
256x1
128x2
64x4
512x1
256x2
128x4
1024x1
512x2
256x4
128x8
1024x2
512x4
256x8
256x12
1024x4
512x8
256x16
256x24
MPIxOMP

Craypat BT-MZ 256x16

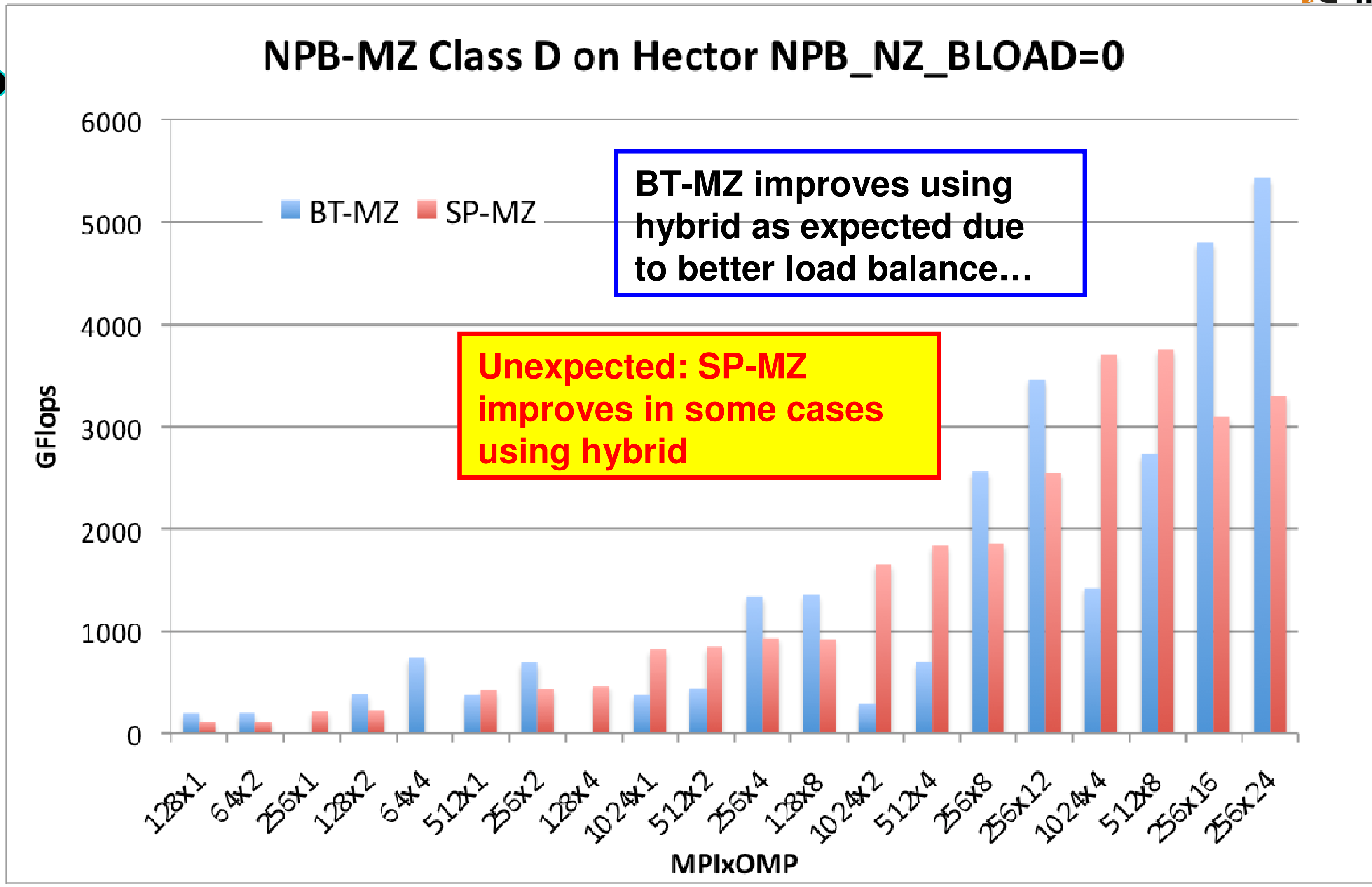
```
Number of PEs (MPI ranks):   256

Numbers of PEs per Node:       1  PE on each of       256  Nodes

Numbers of Threads per PE:    16  threads on each of  248  PEs
                              17  threads on each of    8  PEs

Number of Cores per Socket:   12
```

Benchmark tries to balance load, aprun –d 16 yields multiple threads on same core!

```
export NPB_MZ_BLOAD=0
```

Benchmark will not try to load-balance between threads

```
Number of PEs (MPI ranks):   256

Numbers of PEs per Node:       1  PE on each of  256  Nodes

Numbers of Threads per PE:    16

Number of Cores per Socket:   12
```


NPB-MZ Class D on Hector NPB_NZ_BLOAD=0
6000
5000
4000
3000
2000
1000
0
GFlops
BT-MZ
SP-MZ
BT-MZ improves using hybrid as expected due to better load balance…
Unexpected: SP-MZ improves in some cases using hybrid
128x1
64x2
256x1
128x2
64x4
512x1
256x2
128x4
1024x1
512x2
256x4
128x8
1024x2
512x4
256x8
256x12
1024x4
512x8
256x16
256x24
MPIxOMP

Cray XE6: CrayPat Performance Analysis

- `module load xt-craypat`
- Compilation:
  - ftn –fastsse –r8 –mp[= trace]
- Instrument:
  - pat_build –w –g mpi,omp bt.exe bt.exe.pat
- Execution :
  - (export PAT_RT_HWPC {0,1,2,..})
  - export OMP_NUM_THREADS 4
  - aprun –n *NPROCS* –d 4 ./bt.exe.pat
- Generate report:
  - pat_report –O load_balance,thread_times,program_time,mpi_callers –O profile_pe.th $1

-d *depth* Specifies the number of CPUs for each PE and its threads.

BT-MZ 32x4 Function Profile

```
!$OMP PARALLEL DEFAULT(SHARED) PRIVATE(n,m,k,i,j,ksize)
!$OMP&  SHARED(dz5,dz4,dz3,dz2,dz1,tz2,tz1,dt,c1345,c4,c3,con43,c3c4,c1,
!$OMP&          c2,nx,ny,nz)
      ksize = nz-1

c---------------------------------------------------------------------
c     Compute the indices for storing the block-diagonal matrix;
c     determine c (labeled f) and s jacobians
c---------------------------------------------------------------------
!$OMP DO
      do j = 1, ny-2
         do i = 1, nx-2
            do k = 0, ksize

               tmp1 = 1.d0 / u(1,i,j,k)
               tmp2 = tmp1 * tmp1
               tmp3 = tmp1 * tmp2

               fjac(1,1,k) = 0.d0
               fjac(1,2,k) = 0.d0
               fjac(1,3,k) = 0.d0
               fjac(1,4,k) = 1.d0
               fjac(1,5,k) = 0.d0
```

```
nt=1
n
ead='HIDE'
----------
e_.LOOP@li.43
e_.LOOP@li.43
e_.LOOP@li.46
e_rhs_.MASTER@li.291
e_rhs_.LOOP@li.187
e_rhs_.LOOP@li.53
e_rhs_.LOOP@li.76
e_rhs_.LOOP@li.28
e_rhs_.LOOP@li.297
lize_.LOOP@li.40
e_rhs_.LOOP@li.381
||   1.2% | 0.016753 | 0.003972 |  19.3% |   168 |add_.LOOP@li.22
||=============================================================
|   2.1% | 0.030491 |        -- |      -- |  1040 |MPI
||-------------------------------------------------------------
||   1.8% | 0.026193 | 0.111613 |  81.6% |   105 |mpi_waitall_
||=============================================================
```

BT-MZ Load-Balance 32x4 vs 128x1

```
Table 2:  Load Balance across PE's by FunctionGroup

 Time % |      Time |  Calls |Experiment=1
        |           |        |Group
        |           |        | PE[mmm]
        |           |        |  Thread

 100.0% | 1.782603 | 18662 |Total
|------------------------------------------
|  86.1% | 1.535163 |  7783 |USER
||-----------------------------------------
||   2.7% | 1.535987 |  6813 |pe.0
|||----------------------------------------
3||    0.7% | 1.535987 |  6188 |thread.1
3||    0.7% | 1.535871 |  6188 |thread.3
3||    0.7% | 1.535829 |  6188 |thread.2
3||    0.7% | 1.466954 |  6813 |thread.0
|||========================================
||   2.7% | 1.535147 |  7783 |pe.18
|||----------------------------------------
3||    0.7% | 1.535147 |  7072 |thread.1
3||    0.7% | 1.534995 |  7072 |thread.3
3||    0.7% | 1.534968 |  7072 |thread.2
3||    0.6% | 1.290502 |  7783 |thread.0
|||========================================
||   2.7% | 1.534239 |  7783 |pe.16
|||----------------------------------------
3||    0.7% | 1.534239 |  7072 |thread.1
3||    0.7% | 1.534101 |  7072 |thread.3
3||    0.7% | 1.534076 |  7072 |thread.2
3||    0.6% | 1.268085 |  7783 |thread.0
||=========================================
```

bt-mz-C.32x4

```
Table 2:  Load Balance across PE's by FunctionGroup

 Time % |       Time |  Calls |Group
        |            |        | PE[mmm]

 100.0% | 24.277514 | 38258 |Total
|-------------------------------------
|  54.2% | 13.166225 |   4545 |MPI
||------------------------------------
||    0.5% | 16.454993 |   4846 |pe.91
||    0.5% | 14.058598 |   2434 |pe.29
||    0.0% |  0.289479 |   2434 |pe.0
||====================================
|  44.9% | 10.894808 | 17983 |USER
||------------------------------------
||    0.7% | 23.205797 |   9093 |pe.0
||    0.3% | 10.084200 | 26873 |pe.110
||    0.3% |  8.070997 | 17983 |pe.91
|=====================================
```

bt-mz-C.128x1

- maximum, median, minimum PE are shown
- bt-mz.C.128x1 shows large imbalance in User and MPI time
- bt-mz.C.32x4 shows well balanced times

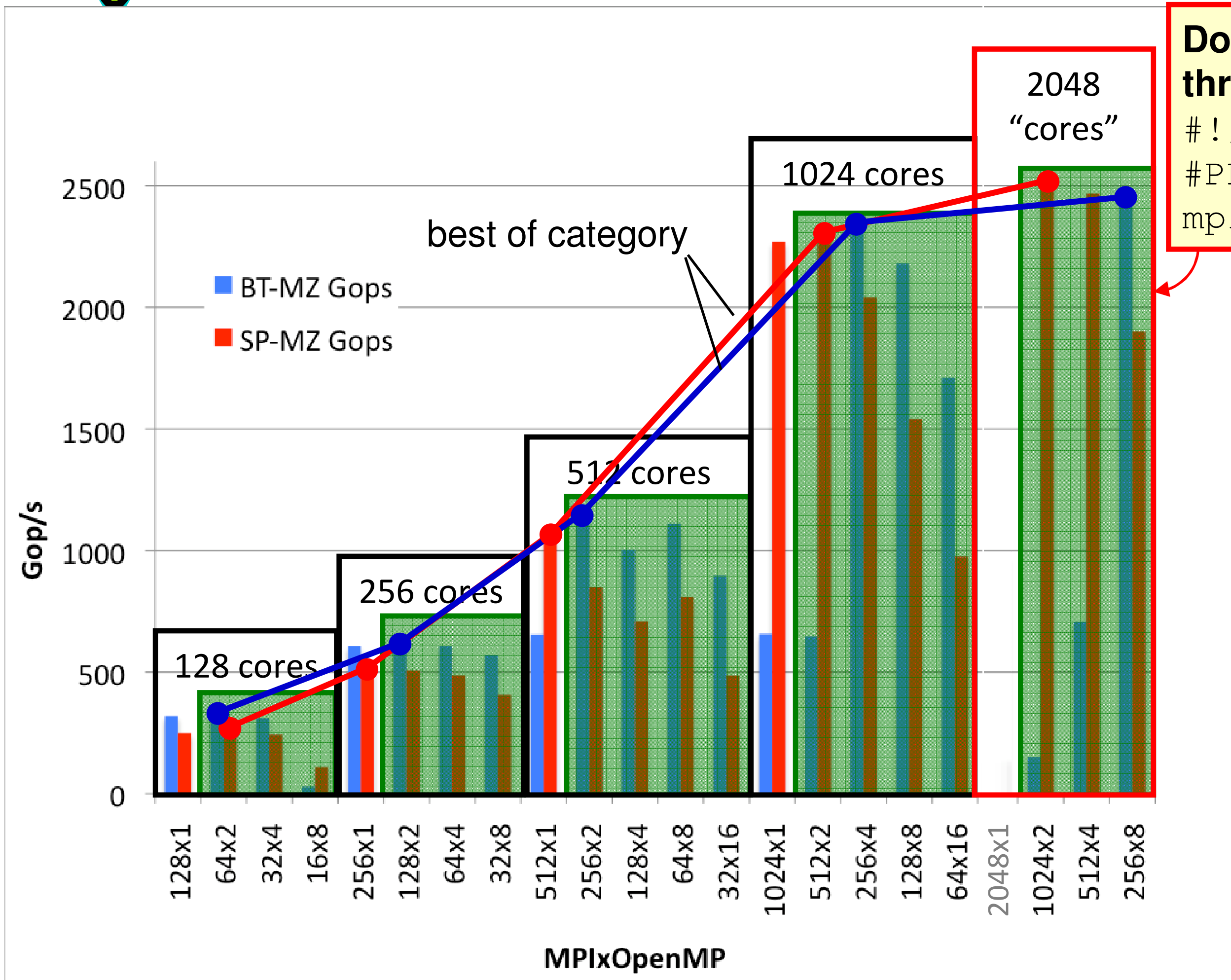
IBM Power 6

- Results obtained by the courtesy of the HPCMO Program and the Engineer Research and Development Center Major Shared Resource Center, Vicksburg, MS (http://www.erdc.hpc.mil/index)
- The IBM Power 6 System is located at (http://www.navo.hpc.mil/davinci_about.html)
- 150 Compute Nodes
- 32 4.7GHz Power6 Cores per Node (4800 cores total)
- 64 GBytes of dedicated memory per node
- QLOGOC Infiniband DDR interconnect
- IBM MPI: MPI 1.2 + MPI-IO
  - mpxlf_r –O4 –qarch=pwr6 –qtune=pwr6 –qsmp=omp

Flag was essential to achieve full compiler optimization in presence of OMP directives!

- Execution:
  - poe launch $PBS_O_WORKDIR./sp.C.16x4.exe

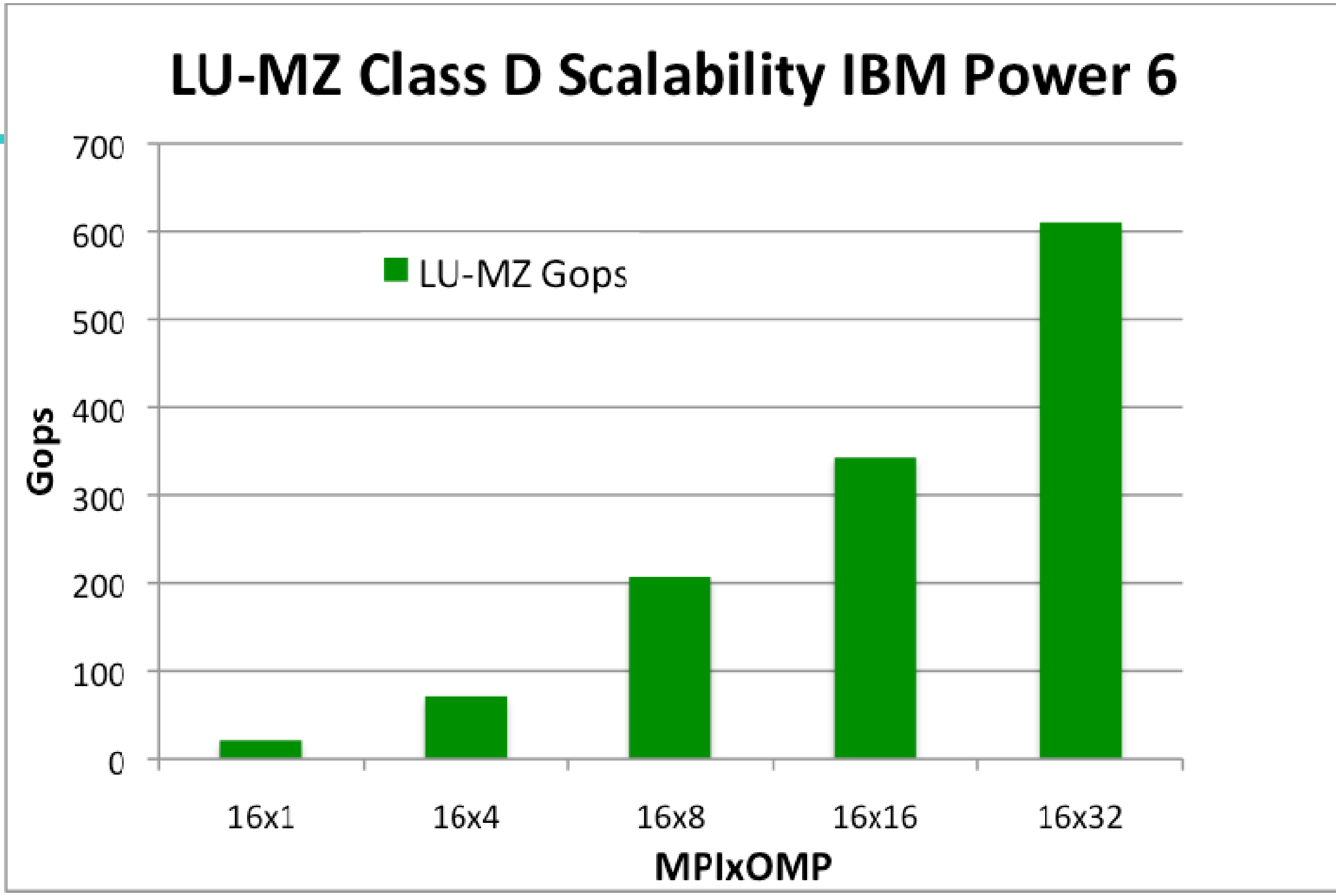
NPB-MZ Class D on IBM Power 6: Exploiting SMT for 2048 Core Results



Doubling the number of threads through hyperthreading (SMT):

```
#!/bin/csh
#PBS -l select=32:ncpus=64:
mpiprocs=NP:ompthreads=NT
```

- Results for 128-2048 cores
- Only 1024 cores were available for the experiments
- BT-MZ and SP-MZ show benefit from **Simultaneous Multithreading (SMT)**: 2048 threads on 1024 cores



- LU-MZ significantly benefits from hybrid mode:
  - ➢ Pure MPI limited to 16 cores, due to #zones = 16

Performance Analysis on IBM Power 6

- Compilation:
  - ➢ mpxlf_r –O4 –qarch=pwr6 –qtune=pwr6 –qsmp=omp –pg
- Execution :
  - ➢ export OMP_NUM_THREADS 4
  - ➢ poe launch $PBS_O_WORKDIR./sp.C.16x4.exe
  - ➢ Generates a file gmount.MPI_RANK.out for each MPI Process
- Generate report:
  - ➢ gprof sp.C.16x4.exe gmon*

```
%    cumulative   self              self     total
time   seconds   seconds    calls  ms/call  ms/call  name
16.7     117.94   117.94   205245     0.57     0.57  .@10@x_solve@OL@1 [2]
14.6     221.14   103.20   205064     0.50     0.50  .@15@z_solve@OL@1 [3]
12.1     307.14    86.00   205200     0.42     0.42  .@12@y_solve@OL@1 [4]
 6.2     350.83    43.69   205300     0.21     0.21  .@8@compute_rhs@OL@1@OL@6 [5]
```

Conclusions:

- **BT-MZ:**
  - Inherent workload imbalance on MPI level
  - #nprocs = #nzones yields poor performance
  - #nprocs < #zones => better workload balance, but decreases parallelism
  - Hybrid MPI/OpenMP yields better load-balance, maintains amount of parallelism
- **SP-MZ:**
  - No workload imbalance on MPI level, pure MPI should perform best
  - MPI/OpenMP outperforms MPI on some platforms due contention to network access within a node
- **LU-MZ:**
  - Hybrid MPI/OpenMP increases level of parallelism
- **"Best of category" depends on many factors**
  - Depends on many factors
  - Hard to predict
  - Good thread affinity is essential

Conclusions:

- **BT-MZ:**
  - Inherent workload imbalance on MPI level
  - #nprocs = #nzones yields poor performance
  - #nprocs < #zones => better workload balance, but decreases parallelism
  - Hybrid MPI/OpenMP yields better load-balance, maintains amount of parallelism
- **SP-MZ:**
  - No workload imbalance on MPI level, pure MPI should perform best
  - MPI/OpenMP outperforms MPI on some platforms due contention to network access within a node
- **LU-MZ:**
  - Hybrid MPI/OpenMP increases level of parallelism
- **All Benchmarks:**
  - Decrease network pressure
  - Lower memory requirements
  - Good process/thread affinity essential

Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / pure MPI vs hybrid MPI+OpenMP
- **Practical "How-To" on hybrid programming**

  Georg Hager, Regionales Rechenzentrum Erlangen (RRZE)
- Mismatch Problems
- Application categories that can benefit from hybrid parallelization
- Thread-safety quality of MPI libraries
- Tools for debugging and profiling MPI+OpenMP
- Other options on clusters of SMP nodes
- Summary

Hybrid Programming How-To: Overview

- A practical introduction to hybrid programming
  - How to compile and link
  - Getting a hybrid program to run on a cluster

- Running hybrid programs efficiently on multi-core clusters
  - Affinity issues
    - **ccNUMA**
    - **Bandwidth bottlenecks**
  - Intra-node MPI/OpenMP anisotropy
    - **MPI communication characteristics**
    - **OpenMP loop startup overhead**
  - Thread/process binding

How to compile, link and run

- Use appropriate OpenMP compiler switch (-openmp, -xopenmp, -mp, -qsmp=openmp, …) and MPI compiler script (if available)
- Link with MPI library
  - Usually wrapped in MPI compiler script
  - If required, specify to link against thread-safe MPI library
    - **Often automatic when OpenMP or auto-parallelization is switched on**
- Running the code
  - Highly non-portable! Consult system docs! (if available…)
  - If you are on your own, consider the following points
  - Make sure OMP_NUM_THREADS etc. is available on all MPI processes
    - **Start "env VAR=VALUE … <YOUR BINARY>" instead of your binary alone**
    - **Use Pete Wyckoff's *mpiexec* MPI launcher (see below): http://www.osc.edu/~pw/mpiexec**
  - Figure out how to start less MPI processes than cores on your nodes

Some examples for compilation and execution (1)

- **NEC SX9**
  - NEC SX9 compiler
  - `mpif90 -C hopt -P openmp … # -ftrace for profiling info`
  - Execution:

```
$ export OMP_NUM_THREADS=<num_threads>
$ MPIEXPORT="OMP_NUM_THREADS"
$ mpirun -nn <# MPI procs per node> -nnp <# of nodes> a.out
```

- **Standard Intel Xeon cluster (e.g. @HLRS):**
  - Intel Compiler
  - `mpif90 -openmp …`
  - Execution (**handling of OMP_NUM_THREADS, see next slide**):

```
$ mpirun_ssh -np <num MPI procs> -hostfile machines a.out
```

Some examples for compilation and execution (2)

Handling of OMP_NUM_THREADS

- **<u>without</u> any support by mpirun:**
  - E.g. with mpich-1
  - Problem:
    mpirun has no features to export environment variables to the via ssh automatically started MPI processes
  - Solution: Set
    `export OMP_NUM_THREADS=``<# threads per MPI process>`
    in ~/.bashrc (if a bash is used as login shell)
  - If you want to set OMP_NUM_THREADS individually when starting the MPI processes:
    - **Add**
      `test -s ~/myexports && . ~/myexports`
      in your ~/.bashrc
    - **Add**
      `echo '$OMP_NUM_THREADS=``<# threads per MPI process>`**`' > ~/myexports`**
      before invoking mpirun
    - **Caution: Several invocations of mpirun cannot be executed at the same time with this trick!**

Some examples for compilation and execution (3)

Handling of OMP_NUM_THREADS (continued)

- **with support by OpenMPI –x option:**

```
export OMP_NUM_THREADS= <# threads per MPI process>
mpiexec –x OMP_NUM_THREADS –n <# MPI processes> ./executable
```

Some examples for compilation and execution (4)

- **Sun Constellation Cluster:**
  - `mpif90 -fastsse -tp barcelona-64 -mp …`
  - SGE Batch System
  - `setenv OMP_NUM_THREADS`
  - `ibrun numactl.sh a.out`
  - Details see TACC Ranger User Guide (www.tacc.utexas.edu/services/userguides/ranger/#numactl)
- **Cray XT5:**
  - `ftn -fastsse -tp barcelona-64 -mp=nonuma …`
  - **`aprun -n`** ***`nprocs`*** **`-N`** ***`nprocs_per_node`*** **`a.out`**

Interlude: Advantages of mpiexec or similar mechanisms

- Uses PBS/Torque Task Manager ("TM") interface to spawn MPI processes on nodes
  - As opposed to starting remote processes with ssh/rsh:
    - **Correct CPU time accounting in batch system**
    - **Faster startup**
    - **Safe process termination**
    - **Understands PBS per-job nodefile**
    - **Allowing password-less user login not required between nodes**
  - Support for many different types of MPI
    - **All MPICHs, MVAPICHs, Intel MPI, …**
  - Interfaces directly with batch system to determine number of procs
  - Downside: If you don't use PBS or Torque, you're out of luck…
- Provisions for starting less processes per node than available cores
  - Required for hybrid programming
  - "-pernode" and "-npernode #" options – does not require messing around with nodefiles

Running the code

Examples with mpiexec

- Example for using mpiexec on a dual-socket quad-core cluster:

```
$ export OMP_NUM_THREADS=8
$ mpiexec -pernode ./a.out
```

- Same but 2 MPI processes per node:

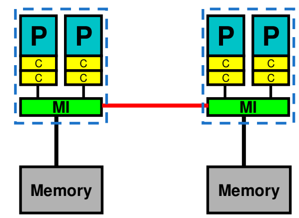
```
$ export OMP_NUM_THREADS=4
$ mpiexec -npernode 2 ./a.out
```

- Pure MPI:

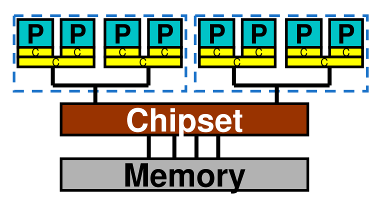
```
$ export OMP_NUM_THREADS=1 # or nothing if serial code
$ mpiexec ./a.out
```

Running the code *efficiently?*

- Symmetric, UMA-type compute nodes have become rare animals
  - NEC SX
  - Intel 1-socket (“Port Townsend/Melstone/Lynnfield”) – see case studies
  - Hitachi SR8000, IBM SP2, single-core multi-socket Intel Xeon… (all dead)
- Instead, systems have become “non-isotropic” on the node level
  - ccNUMA (AMD Opteron, SGI Altix, IBM Power6 (p575), Intel Nehalem)



  - Multi-core, multi-socket
    - **Shared vs. separate caches**
    - **Multi-chip vs. single-chip**
    - **Separate/shared buses**



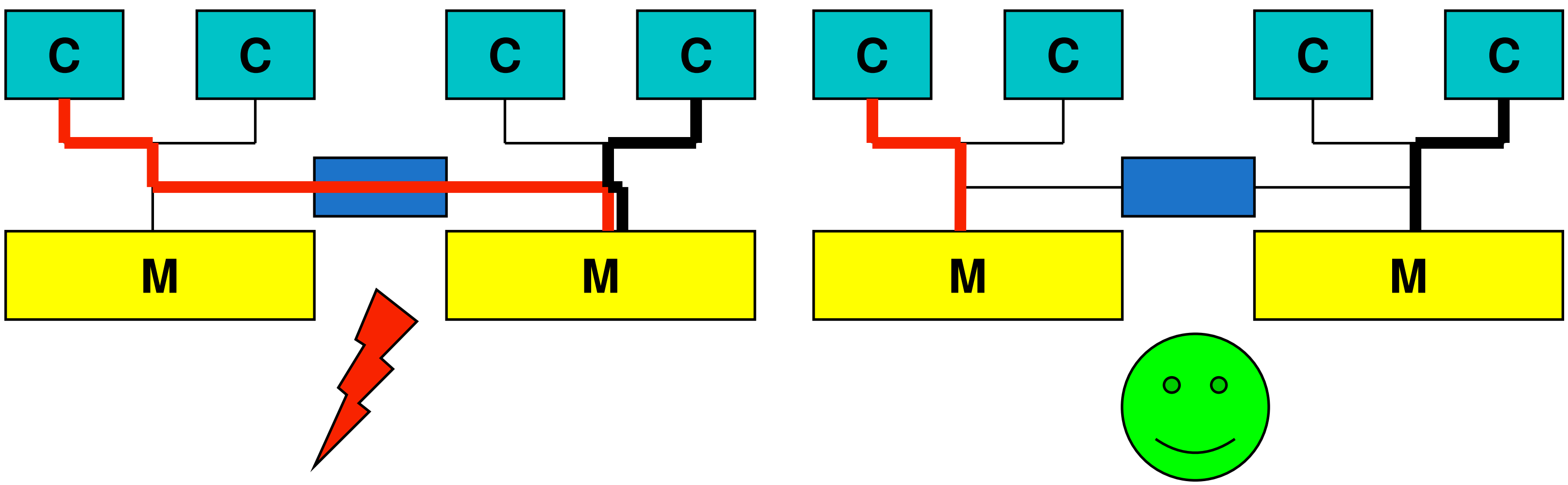
Issues for running code efficiently on "non-isotropic" nodes

- ccNUMA locality effects
  - Penalties for inter-LD access
  - Impact of contention
  - Consequences of file I/O for page placement
  - Placement of MPI buffers

- Multi-core / multi-socket anisotropy effects
  - Bandwidth bottlenecks, shared caches
  - Intra-node MPI performance
    - **Core ↔ core vs. socket ↔ socket**
  - OpenMP loop overhead depends on mutual position of threads in team

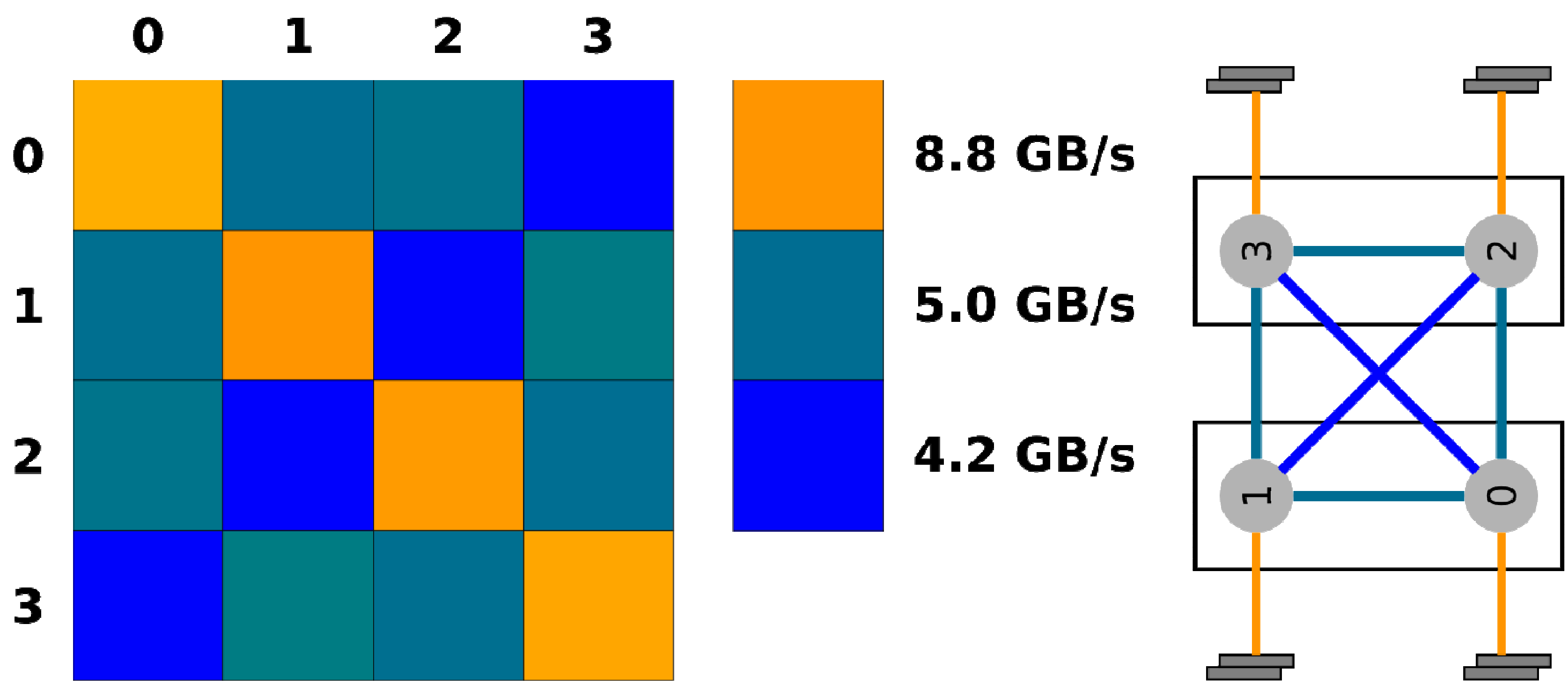
A short introduction to ccNUMA

- ccNUMA:
  - whole memory is transparently accessible by all processors
  - but physically distributed
  - with varying bandwidth and latency
  - and potential contention (shared memory paths)



How much does non-local access cost?

- Example: AMD Magny Cours 2-socket system (4 chips, 2 sockets)
  STREAM bandwidth measurements



ccNUMA Memory Locality Problems

- Locality of reference is key to scalable performance on ccNUMA
  - Less of a problem with pure MPI, but see below
- What factors can destroy locality?
- MPI programming:
  - processes lose their association with the CPU the mapping took place on originally
  - OS kernel tries to maintain strong affinity, but sometimes fails
- Shared Memory Programming (OpenMP, hybrid):
  - threads losing association with the CPU the mapping took place on originally
  - improper initialization of distributed data
  - Lots of extra threads are running on a node, especially for hybrid
- All cases:
  - Other agents (e.g., OS kernel) may fill memory with data that prevents optimal placement of user data

Avoiding locality problems

- How can we make sure that memory ends up where it is close to the CPU that uses it?
  - See the following slides
- How can we make sure that it stays that way throughout program execution?
  - See end of section

Solving Memory Locality Problems: First Touch

Important

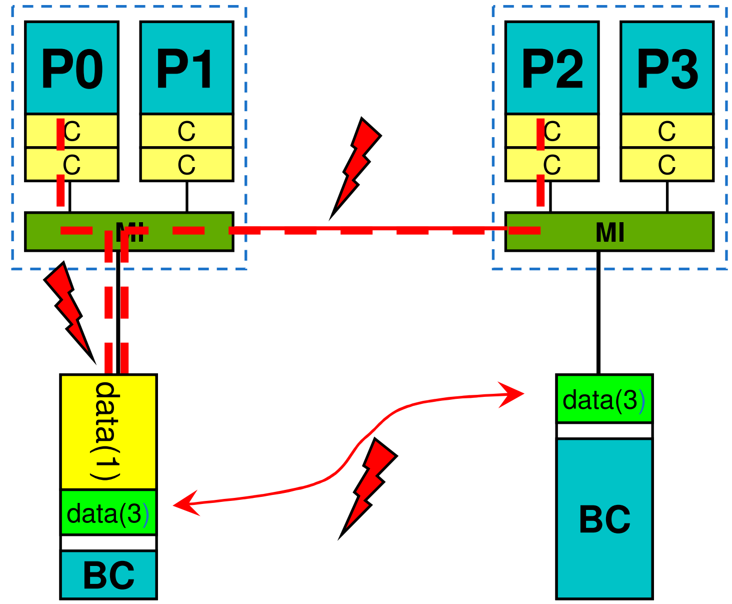
- "Golden Rule" of ccNUMA:
  A memory page gets mapped into the local memory of the processor that first touches it!
  - Except if there is not enough local memory available
  - this might be a problem, see later
  - Some OSs allow to influence placement in more direct ways
    - **cf. libnuma (Linux), MPO (Solaris), …**
- Caveat: "touch" means "write", not "allocate"
- Example:

```
double *huge = (double*)malloc(N*sizeof(double));
// memory not mapped yet
for(i=0; i<N; i++) // or i+=PAGE_SIZE
   huge[i] = 0.0;  // mapping takes place here!
```

- It is sufficient to touch a single item to map the entire page

ccNUMA problems beyond first touch

- OS uses part of main memory for disk buffer (FS) cache
  - If FS cache fills part of memory, apps will probably allocate from foreign domains
  - → non-local access!
  - Locality problem even on hybrid and pure MPI with "asymmetric" file I/O, i.e. if not all MPI processes perform I/O



- Remedies
  - Drop FS cache pages after user job has run (admin's job)
    - **Only prevents cross-job buffer cache "heritage"**
  - "Sweeper" code (run by user)
  - Flush buffer cache after I/O if necessary ("sync" is not sufficient!)

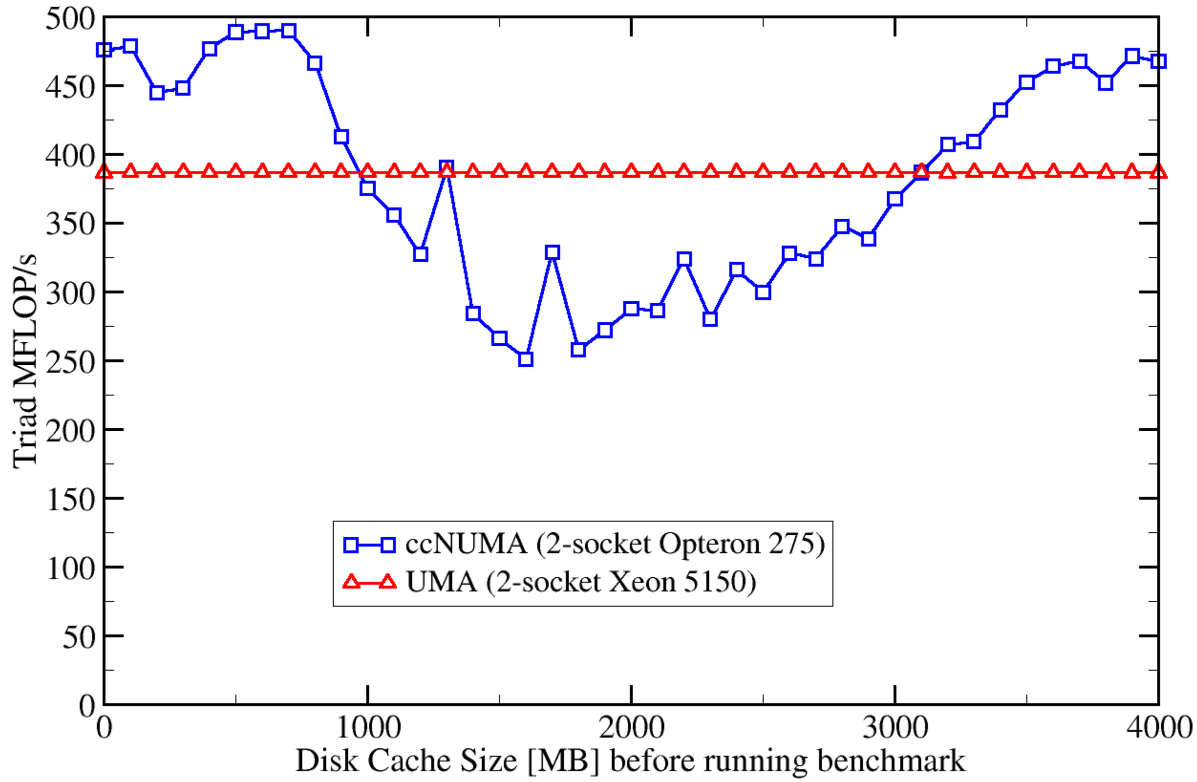
ccNUMA problems beyond first touch

- Real-world example: ccNUMA vs. UMA and the Linux buffer cache
- Compare two 4-way systems: AMD Opteron ccNUMA vs. Intel UMA, 4 GB main memory

- Run 4 concurrent array copy loops (512 MB each) after writing a large file

- Report perfor-mance vs. file size

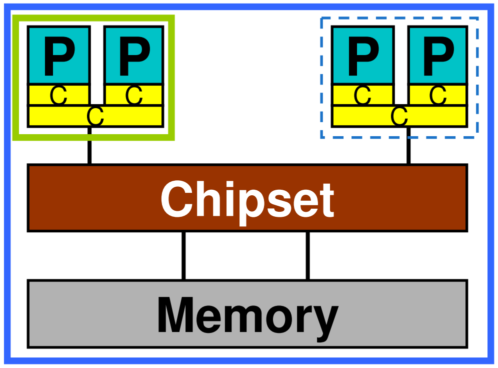
- Drop FS cache after each data point



Intra-node MPI characteristics: IMB Ping-Pong benchmark

- Code (to be run on 2 processors):

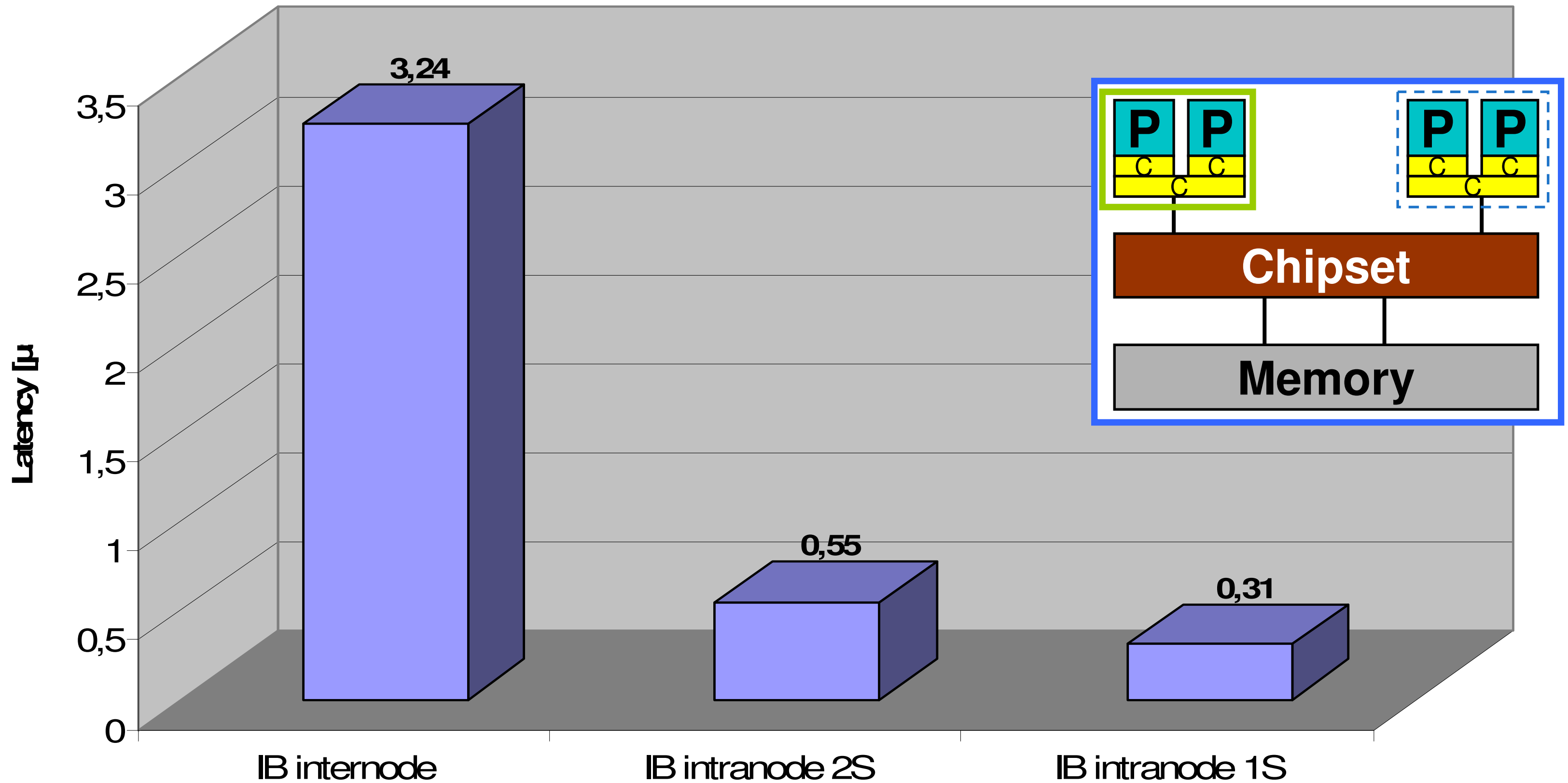
```
wc = MPI_WTIME()

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`

IMB Ping-Pong: Latency

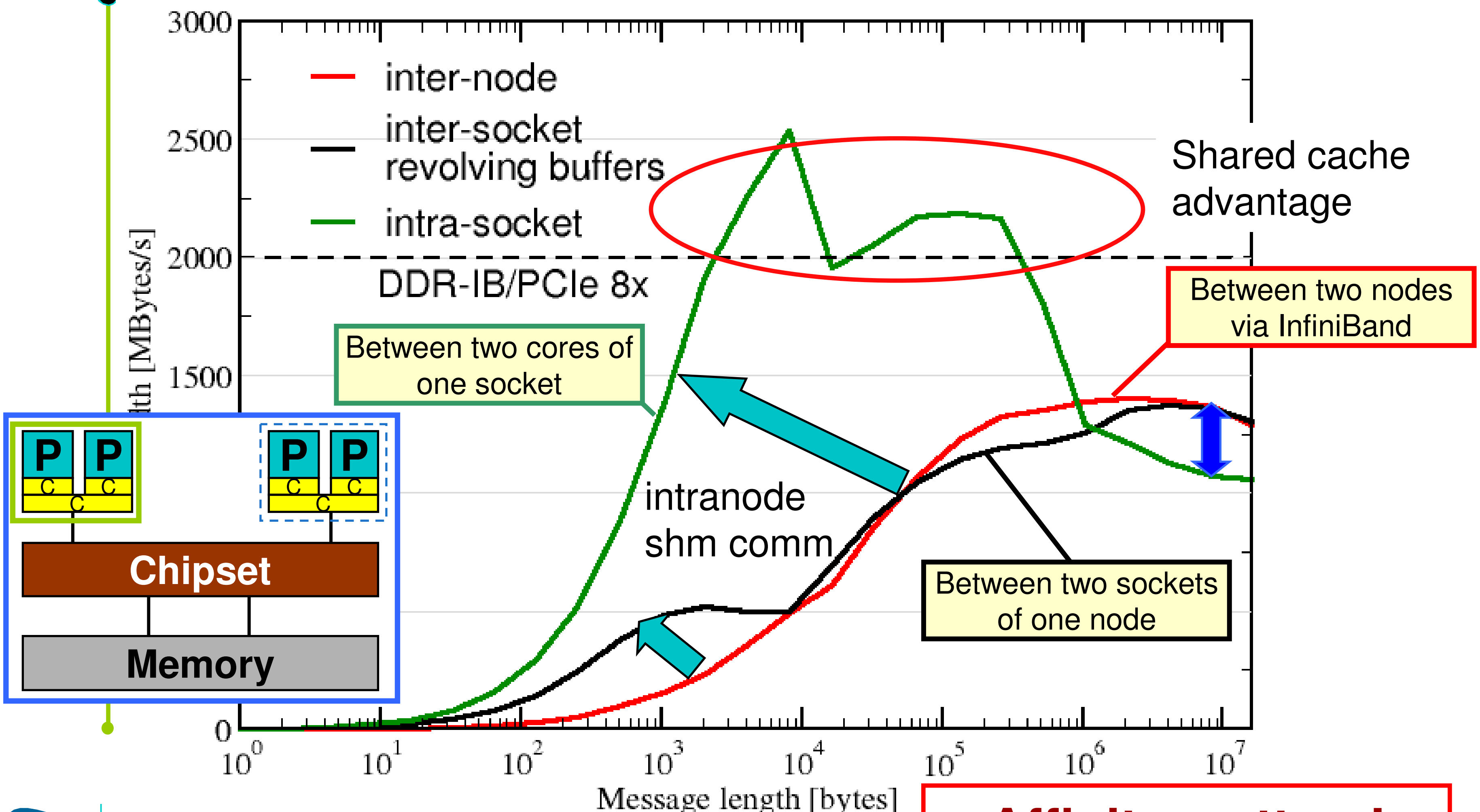
Intra-node vs. Inter-node on Woodcrest DDR-IB cluster *(Intel MPI 3.1)*



Affinity matters!

IMB Ping-Pong: Bandwidth Characteristics

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



Affinity matters!

The parallel vector triad benchmark

A "swiss army knife" for microbenchmarking

- What about OpenMP overhead?
- Simple streaming benchmark:

```
for(int j=0; j < NITER; j++){
#pragma omp parallel for
  for(i=0; i < N; ++i)
    a[i]=b[i]+c[i]*d[i];
    if(OBSCURE)
      dummy(a,b,c,d);
}
```

- Report performance for different N
- Choose NITER so that accurate time measurement is possible
- Triad results lead to a deep understanding of multicore architecture and OpenMP performance overhead

The parallel vector triad benchmark

Optimal code on x86 machines

```
timing(&wct_start, &cput_start);
#pragma omp parallel private(j)
{
  for(j=0; j<niter; j++){
    if(size > CACHE_SIZE>>5) {
#pragma omp parallel for
#pragma vector always
#pragma vector aligned
#pragma vector nontemporal
      for(i=0; i<size; ++i)
        a[i]=b[i]+c[i]*d[i];
    } else {
#pragma omp parallel for
#pragma vector always
#pragma vector aligned
      for(i=0; i<size; ++i)
        a[i]=b[i]+c[i]*d[i];
    }
    if(a[5]<0.0)
      cout << a[3] << b[5] << c[10] << d[6];
  }
}
timing(&wct_end, &cput_end);
```

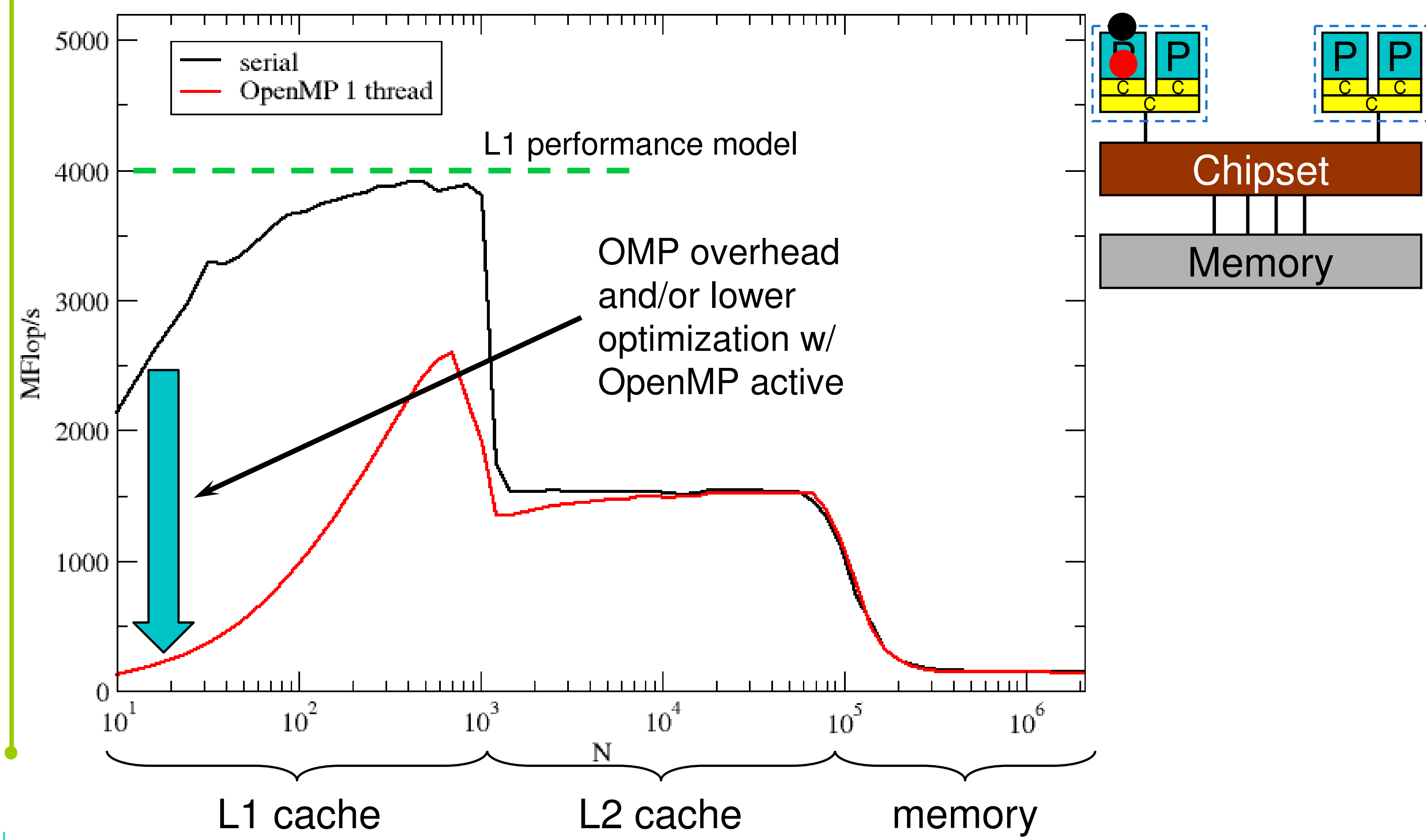
```
// size = multiple of 8
int vector_size(int n){
  return int(pow(1.3,n))&(-8);
}
```

Large-N version (NT)

Small-N version (noNT)

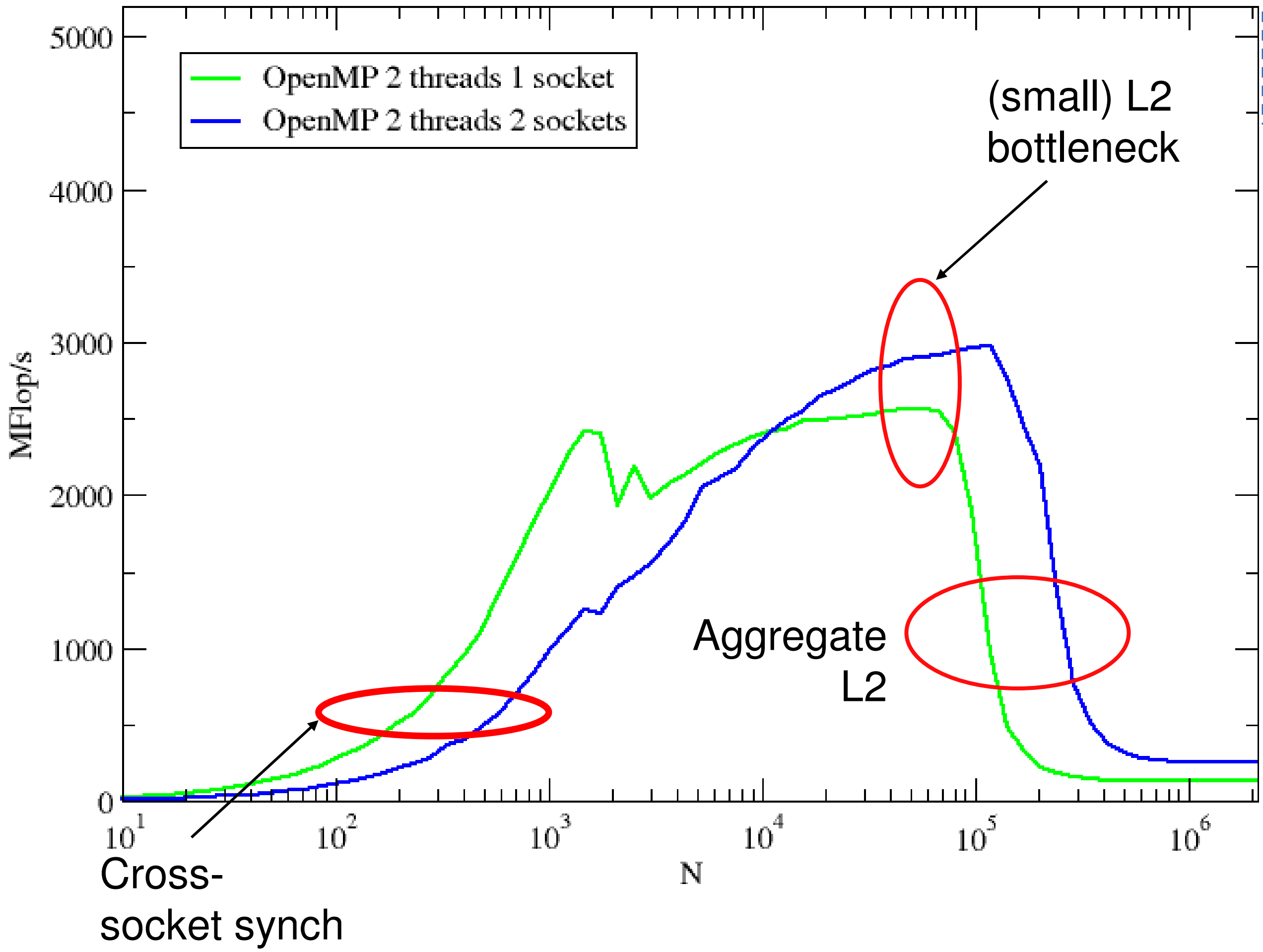
The parallel vector triad benchmark

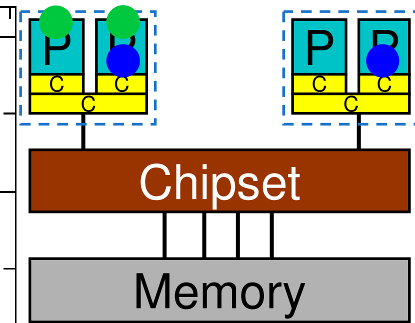
Performance results on Xeon 5160 node



The parallel vector triad benchmark

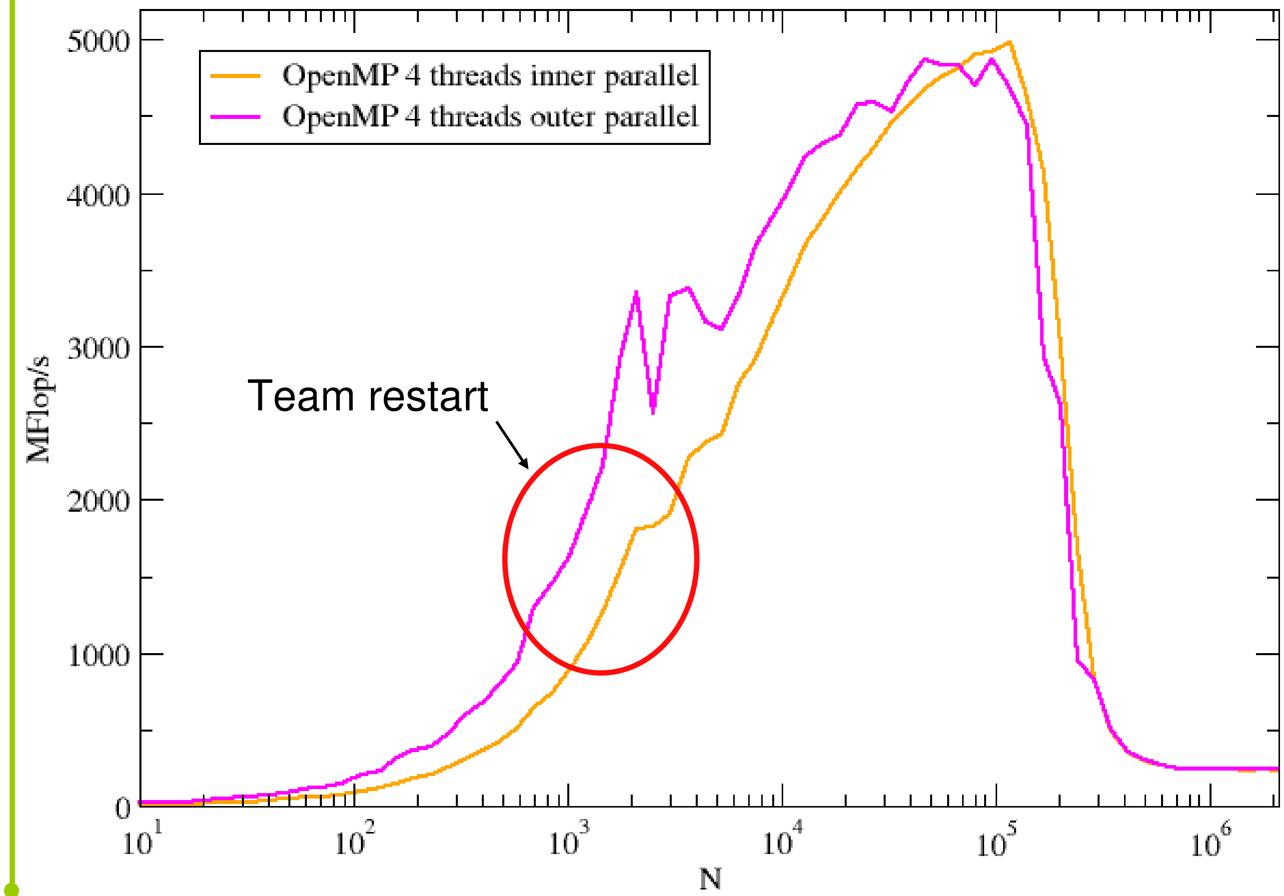
Performance results on Xeon 5160 node

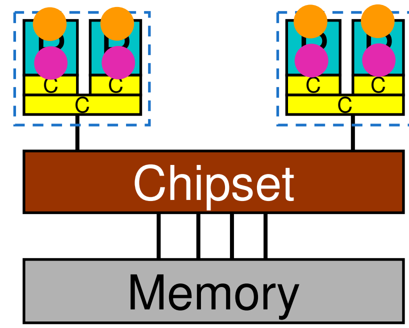




The parallel vector triad benchmark

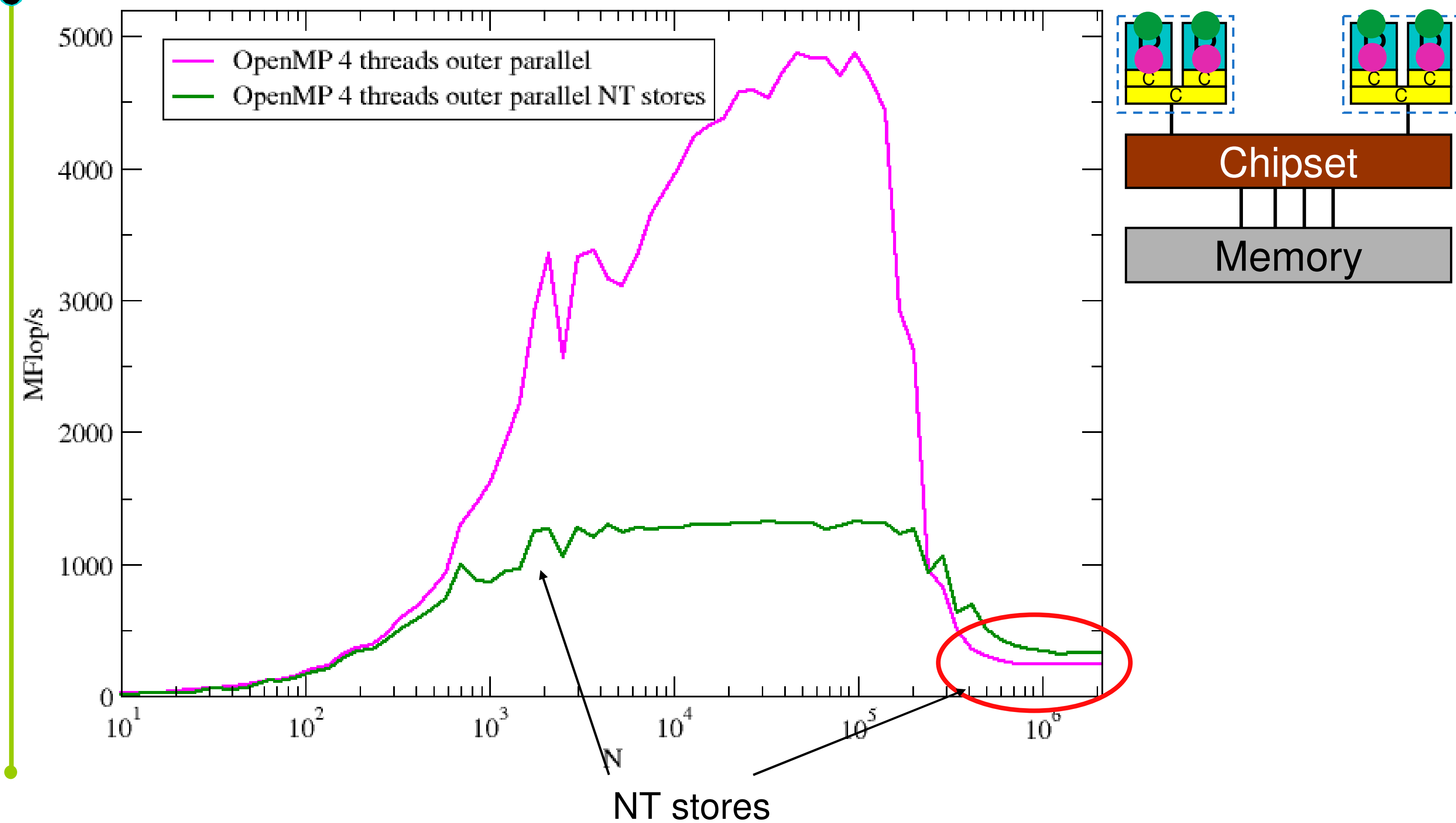
Performance results on Xeon 5160 node





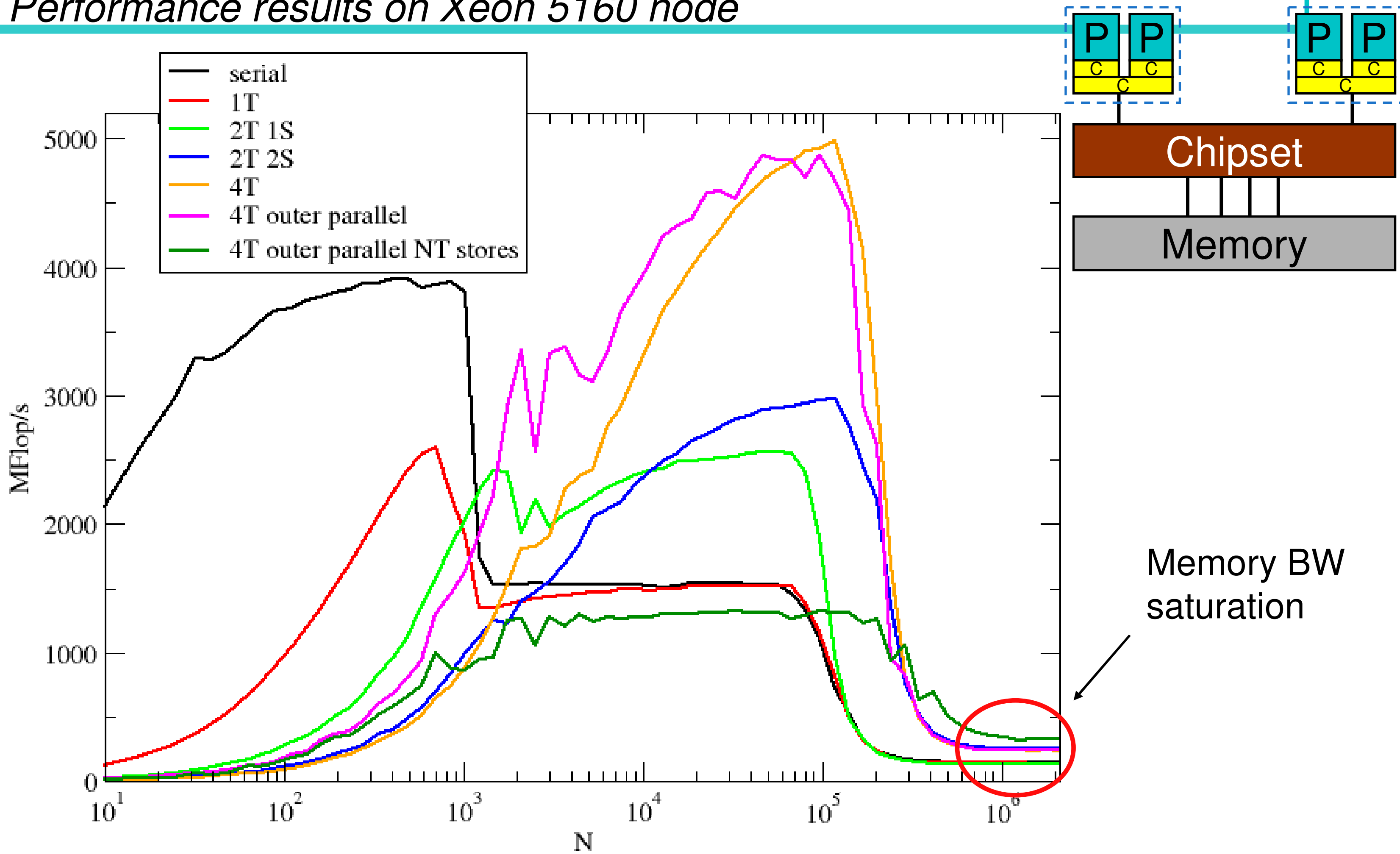
The parallel vector triad benchmark

Performance results on Xeon 5160 node



The parallel vector triad benchmark

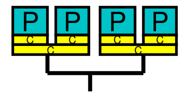
Performance results on Xeon 5160 node

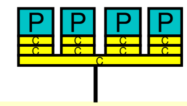


Most of the OpenMP overhead is barrier sync!

But how much is it exactly, and does it depend on the topology?

Overhead in cycles:



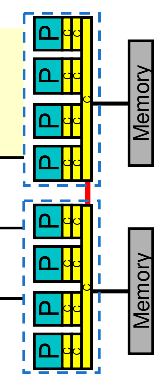


4 Threads	Q9550	i7 920 (shared L3)
(pthreads_barrier_wait)	42533	9820
omp barrier (icc 11.0)	977	814
gcc 4.4.3	41154	8075

pthreads/gcc → OS kernel call

OpenMP & Intel compiler

Nehalem 2 Threads	Shared SMT threads	shared L3	different socket
(pthreads_barrier_wait)	23352	4796	49237
omp barrier (icc 11.0)	2761	479	1206



- SMT can be a performance problem for synchronizing threads
- Topology has an influence on overhead!

Thread/Process Affinity ("Pinning")

- Highly OS-dependent system calls
  - But available on all systems

    Linux: `sched_setaffinity()`, PLPA (see below) → hwloc
    Solaris: `processor_bind()`
    Windows: `SetThreadAffinityMask()`
    …
- Support for "semi-automatic" pinning in some compilers/environments
  - Intel compilers > V9.1 (**`KMP_AFFINITY`** environment variable)
  - Pathscale
  - SGI Altix **`dplace`** (works with logical CPU numbers!)
  - Generic Linux: **`taskset`**, **`numactl`**, **`likwid-pin`** (see below)
- Affinity awareness in MPI libraries
  - SGI MPT
  - OpenMPI
  - Intel MPI
  - …

Seen on SUN Ranger slides

Widely usable example: Using PLPA under Linux!

Explicit Process/Thread Binding With PLPA on Linux: `http://www.open-mpi.org/software/plpa/`

- Portable Linux Processor Affinity
- Wrapper library for `sched_*affinity()` functions
  - Robust against changes in kernel API
- Example for pure OpenMP: Pinning of threads

```
#include <plpa.h>
...
#pragma omp parallel
  {
#pragma omp critical
  {
    if(PLPA_NAME(api_probe)()!=PLPA_PROBE_OK) {
        cerr << "PLPA failed!" << endl; exit(1);
  }
    plpa_cpu_set_t msk;
    PLPA_CPU_ZERO(&msk);
    int cpu = omp_get_thread_num();
    PLPA_CPU_SET(cpu,&msk);
    PLPA_NAME(sched_setaffinity)((pid_t)0, sizeof(cpu_set_t), &msk);
  }
```

Pinning available?

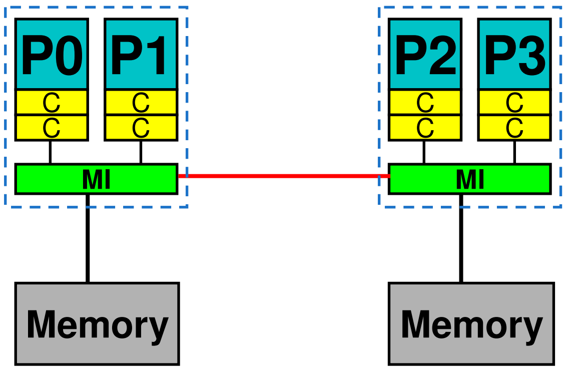
Which CPU to run on?

Pin "me"

Care about correct core numbering! 0…N-1 is not always contiguous! If required, reorder by a map:
`cpu = map[cpu];`

Process/Thread Binding With PLPA

- Example for pure MPI: Process pinning
  - Bind MPI processes to cores in a cluster of 2x2-core machines




```
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
int mask = (rank % 4);
PLPA_CPU_SET(mask,&msk);
PLPA_NAME(sched_setaffinity)((pid_t)0,
                             sizeof(cpu_set_t), &msk);
```

- Hybrid case:

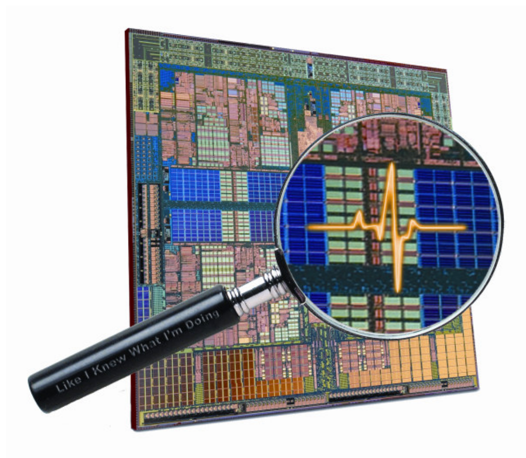
```
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
#pragma omp parallel
  {
    plpa_cpu_set_t msk;
    PLPA_CPU_ZERO(&msk);
    int cpu = (rank % MPI_PROCESSES_PER_NODE)*omp_num_threads
              + omp_get_thread_num();
    PLPA_CPU_SET(cpu,&msk);
    PLPA_NAME(sched_setaffinity)((pid_t)0, sizeof(cpu_set_t), &msk);
  }
```

How do we figure out the topology?

- … and how do we enforce the mapping without changing the code?
- Compilers and MPI libs may still give you ways to do that

- But LIKWID supports all sorts of combinations:

Like
I
Knew
What
I'm
Doing



- Open source tool collection (developed at RRZE):

http://code.google.com/p/likwid

Likwid Tool Suite

- Command line tools for Linux:
  - works with standard linux 2.6 kernel
  - supports Intel and AMD CPUs
  - Supports all compilers whose OpenMP implementation is based on pthreads

- Current tools:
  - **likwid-topology:** Print thread and cache topology (similar to lstopo from the hwloc package)
  - **likwid-pin:** Pin threaded application without touching code
  - likwid-perfctr: Measure performance counters
  - likwid-perfscope: Performance oscilloscope w/ real-time display
  - likwid-powermeter: Current power consumption of chip (alpha stage)
  - likwid-features: View and enable/disable hardware prefetchers
  - likwid-bench: Low-level bandwidth benchmark generator tool
  - likwid-mpirun: mpirun wrapper script for easy LIKWID integration

likwid-topology – Topology information

- Based on cpuid information
- Functionality:
  - Measured clock frequency
  - Thread topology
  - Cache topology
  - Cache parameters (-c command line switch)
  - ASCII art output (-g command line switch)
- Currently supported:
  - Intel Core 2 (45nm + 65 nm)
  - Intel Nehalem, Westmere, Sandy Bridge (alpha)
  - AMD K10 (Quadcore and Hexacore)
  - AMD K8

Output of likwid-topology

```
CPU name:          Intel Core i7 processor
CPU clock:         2666683826 Hz
*************************************************************
Hardware Thread Topology
*************************************************************
Sockets:                   2
Cores per socket:          4
Threads per core:          2
-------------------------------------------------------------
HWThread        Thread          Core            Socket
0               0               0               0
1               1               0               0
2               0               1               0
3               1               1               0
4               0               2               0
5               1               2               0
6               0               3               0
7               1               3               0
8               0               0               1
9               1               0               1
10              0               1               1
11              1               1               1
12              0               2               1
13              1               2               1
14              0               3               1
15              1               3               1
-------------------------------------------------------------
```

likwid-topology continued

```
Socket 0: ( 0 1 2 3 4 5 6 7 )
Socket 1: ( 8 9 10 11 12 13 14 15 )
-------------------------------------------------------------

*************************************************************
Cache Topology
*************************************************************
Level:   1
Size:    32 kB
Cache groups:   ( 0 1 ) ( 2 3 ) ( 4 5 ) ( 6 7 ) ( 8 9 ) ( 10 11 ) ( 12 13 ) ( 14 15 )
-------------------------------------------------------------
Level:   2
Size:    256 kB
Cache groups:   ( 0 1 ) ( 2 3 ) ( 4 5 ) ( 6 7 ) ( 8 9 ) ( 10 11 ) ( 12 13 ) ( 14 15 )
-------------------------------------------------------------
Level:   3
Size:    8 MB
Cache groups:   ( 0 1 2 3 4 5 6 7 ) ( 8 9 10 11 12 13 14 15 )
-------------------------------------------------------------
```

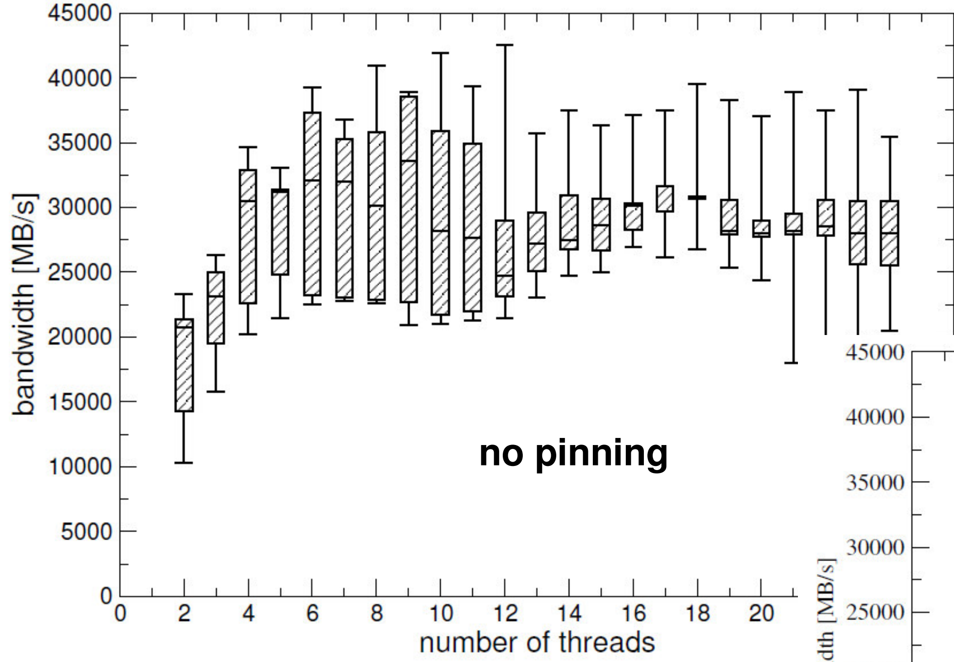
- … and also try the ultra-cool –g option!

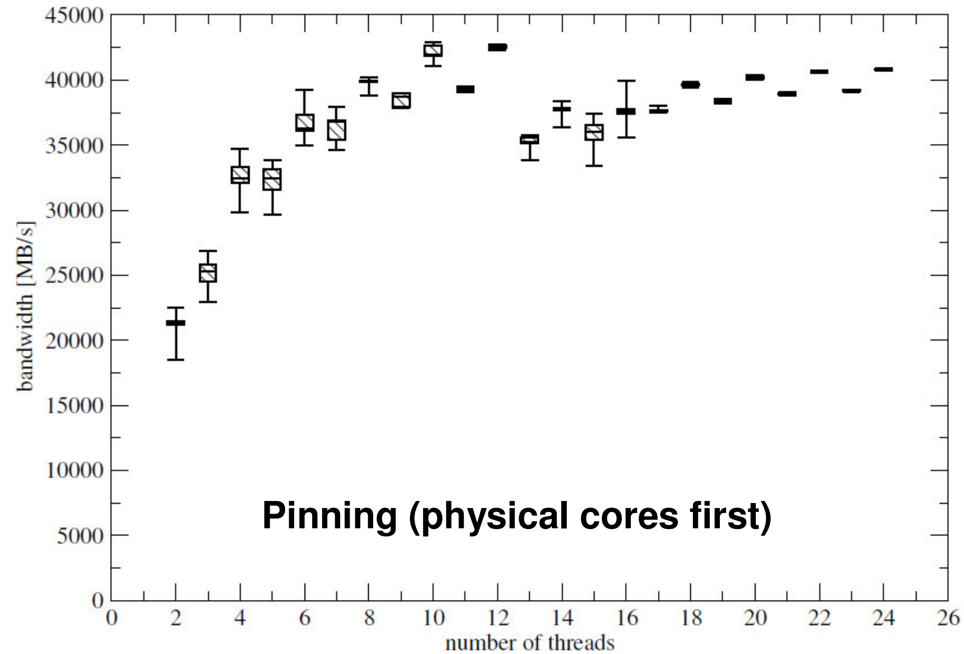
likwid-pin

- Inspired and based on **`ptoverride`** (Michael Meier, RRZE) and **`taskset`**
- Pins process and threads to specific cores without touching code
- Directly supports pthreads, gcc OpenMP, Intel OpenMP
- Allows user to specify skip mask (i.e., supports many different compiler/MPI combinations)
- Can also be used as replacement for **`taskset`**
- Uses logical (contiguous) core numbering when running inside a restricted set of cores
- Supports logical core numbering inside node, socket, core

- Usage examples:
  - `env OMP_NUM_THREADS=6 likwid-pin -t intel -c 0,2,4-6  ./myApp parameters`
  - `env OMP_NUM_THREADS=6 likwid-pin -c S0:0-2@S1:0-2 ./myApp`
  - `env OMP_NUM_THREADS=2 mpirun -npernode 2 \`
    `likwid-pin -s 0x3 -c 0,1  ./myApp parameters`

Example: STREAM benchmark on 12-core Intel Westmere:
Anarchy vs. thread pinning

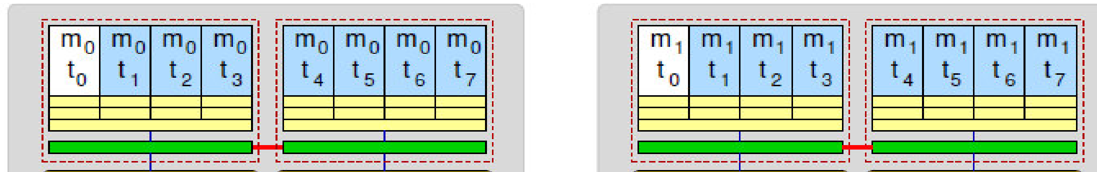




Topology ("mapping") choices with MPI+OpenMP:

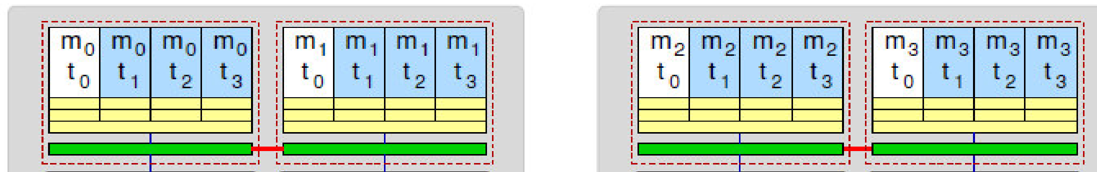
More examples using Intel MPI+compiler & home-grown mpirun

One MPI process per node



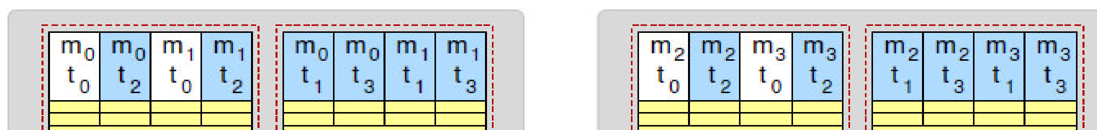

```
env OMP_NUM_THREADS=8 mpirun -pernode \
          likwid-pin -t intel -c 0-7 ./a.out
```

One MPI process per socket



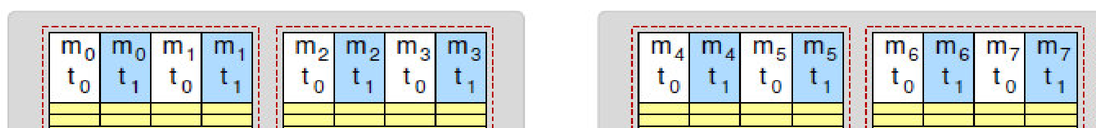

```
env OMP_NUM_THREADS=4 mpirun -npernode 2 \
        -pin "0,1,2,3_4,5,6,7" ./a.out
```

OpenMP threads pinned "round robin" across cores in node




```
env OMP_NUM_THREADS=4 mpirun -npernode 2 \
        -pin "0,1,4,5_2,3,6,7" \
        likwid-pin -t intel -c L:0,2,1,3 ./a.out
```

Two MPI processes per socket




```
env OMP_NUM_THREADS=2 mpirun -npernode 4 \
        -pin "0,1_2,3_4,5_6,7" \
        likwid-pin -t intel -c L:0,1 ./a.out
```

skipped

Case study: 3D Jacobi Solver

Basic implementation (2 arrays; no blocking etc…)

```
do k = 1 , Nk
   do j = 1 , Nj
      do i = 1 , Ni
         y(i,j,k) = a*x(i,j,k) + b*
                        (x(i-1,j,k)+ x(i+1,j,k) + x(i,j-1,k)
                        +x(i,j+1,k)+ x(i,j,k-1) + x(i,j,k+1))
      enddo
   enddo
enddo
```

Performance metric:
Million Lattice Site Updates per second (MLUPs)

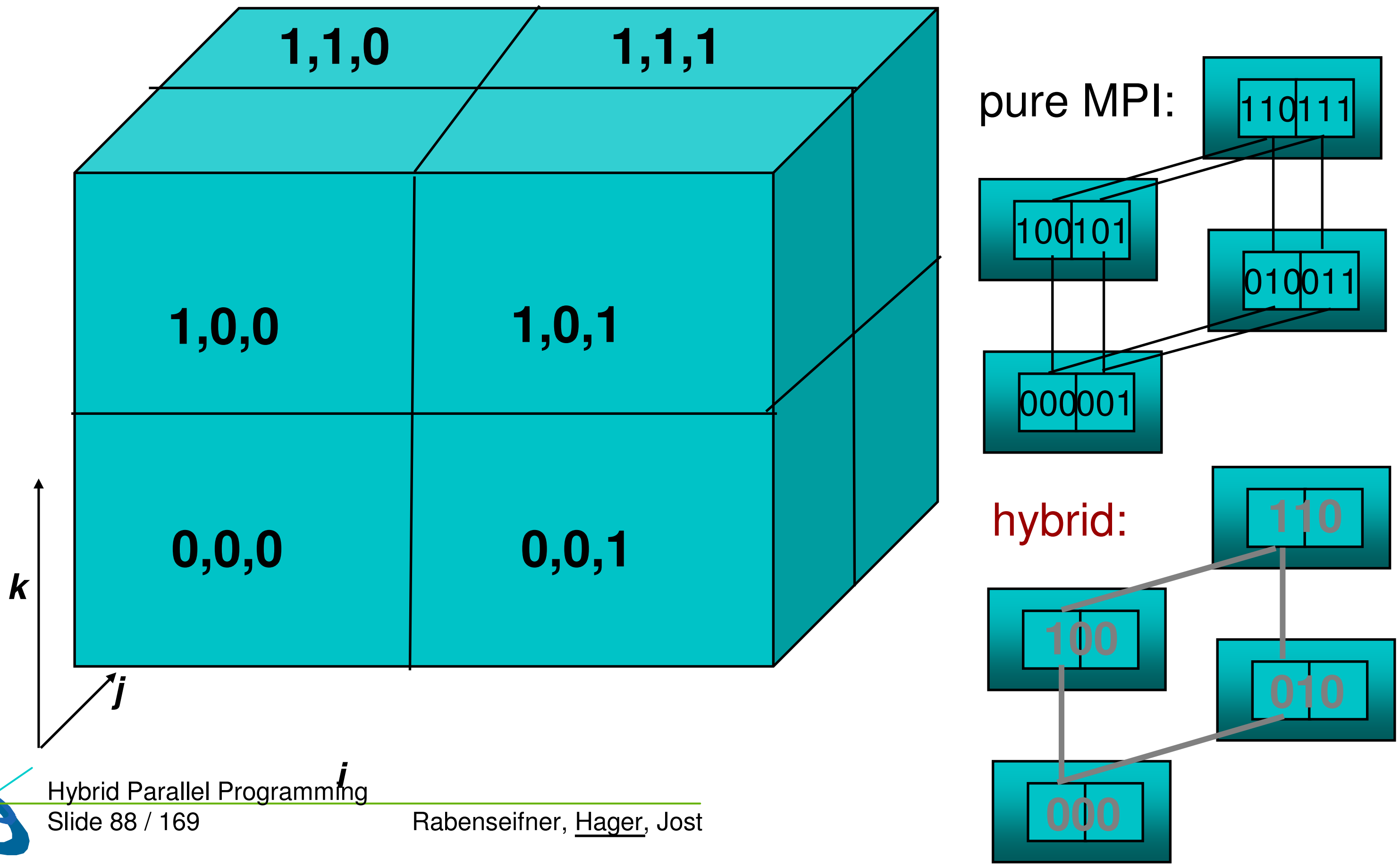
Equivalent MFLOPs:
8 FLOP/LUP * MLUPs

MPI Parallelization by

- Domain Decomposition
- Halo cells
- Data Exchange through cyclic SendReceive operation

MPI/OpenMP Parallelization – 3D Jacobi

- Cubic 3D computational domain with periodic BCs in all directions
- Use single-node IB/GE cluster with one dual-core chip per node
- Homogeneous distribution of workload, e.g. on 8 procs

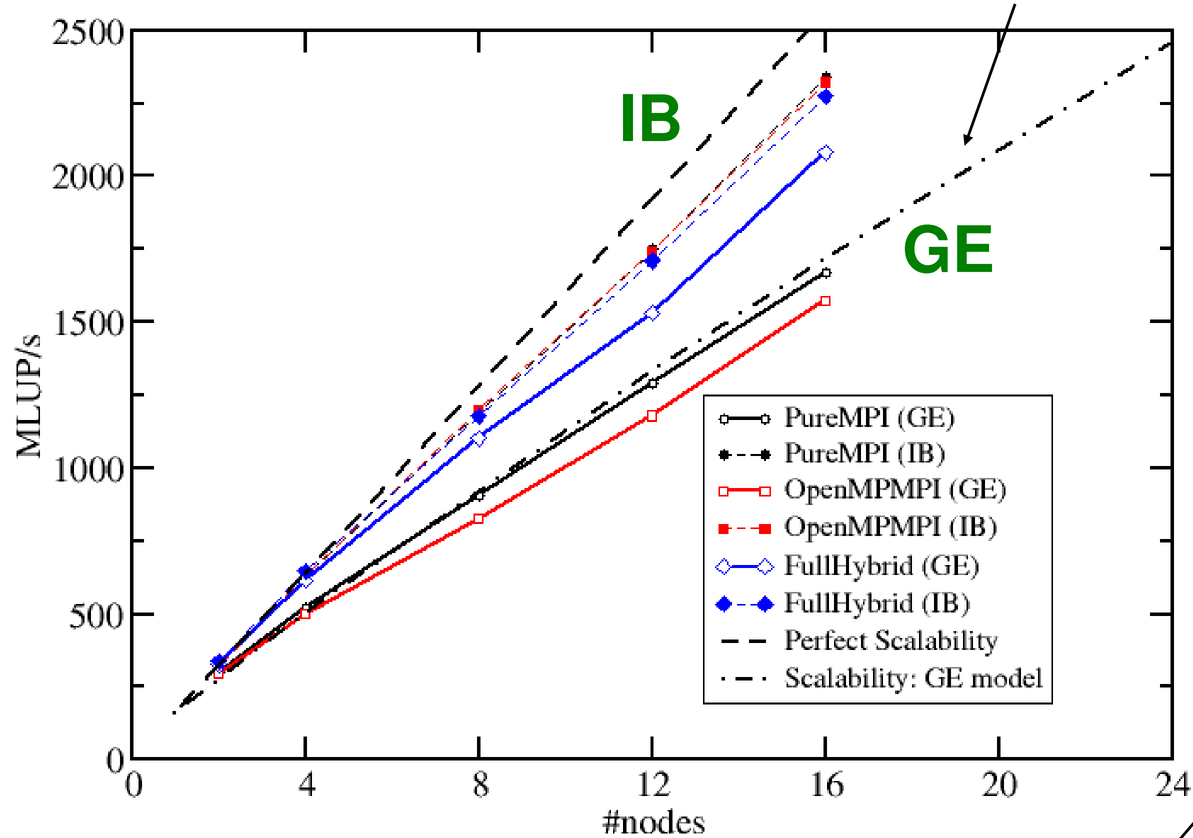


skipped

Performance Data for 3D MPI/hybrid Jacobi

Strong scaling, $N^3 = 480^3$

FullHybrid: Thread 0: Communication + boundary cell updates
Thread 1: Inner cell updates



Performance model

$T = T_{COMM} + T_{COMP}$

$T_{COMP} = N^3 / P_0$

$T_{COMM} = V_{data} / BW$

$P_0 = 150$ MLUP/s
BW(GE) = 100 MByte/s

$\mathbf{V_{data}}$ = Data volume of halo exchange

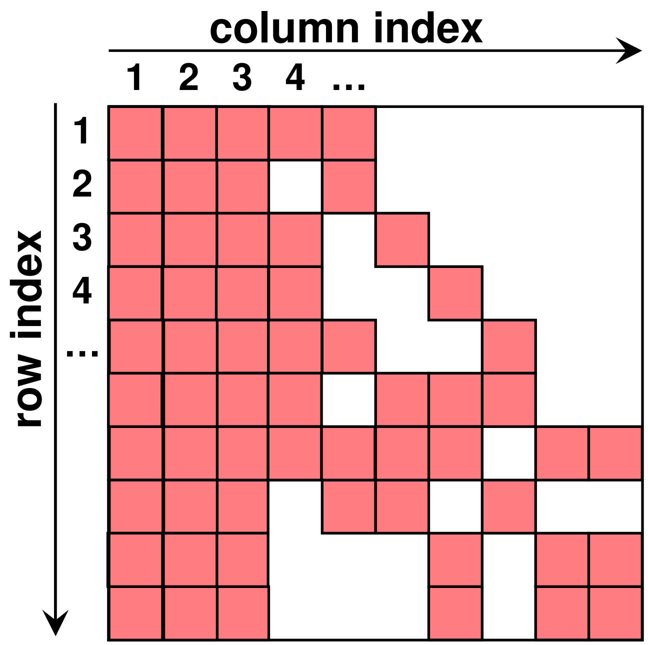
Performance estimate (GE) for *n* nodes:

$$P(n) = N^3 / ((T_{COMP}/n) + T_{COMM}(n))$$

skipped

Example: Sparse MVM

JDS parallel sparse matrix-vector multiply – storage scheme



- `val[]` stores all the nonzeroes (length N_{nz})
- `col_idx[]` stores the column index of each nonzero (length N_{nz})
- `jd_ptr[]` stores the starting index of each new jagged diagonal in `val[]`
- `perm[]` holds the permutation map (length N_r)

`val[]`: …

`col_idx[]`: 4 | 1 | 1 | 1 | 3 | 1 | 3 | 5 | 7 | 7 | 6 | 2 | 3 | 2 | 6 | 2 | 4 | 6 | 9 | 9 | …

`jd_ptr[]`: 1 | 11 | 21 | …

`perm[]`: 2 | 6 | 3 | 7 | 4 | 5 | 1 | 8 | 9 | 10 | …

(JDS = Jagged Diagonal Storage)

skipped

JDS Sparse MVM – Kernel Code

OpenMP parallelization

- Implement `c(:) = m(:,:) * b(:)`
- Operation count = $2N_{nz}$

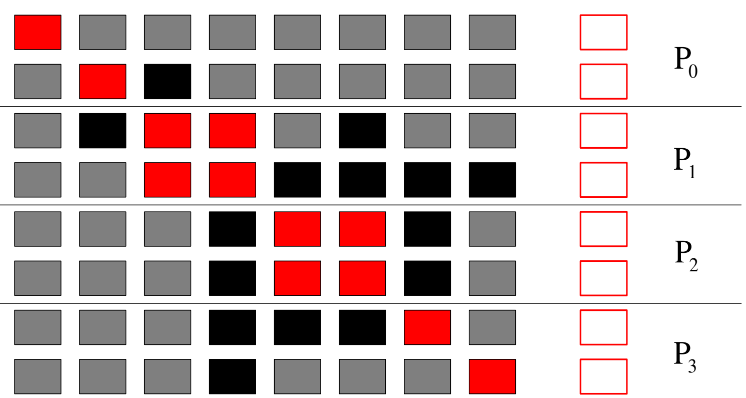
```
do diag=1, zmax
  diagLen = jd_ptr(diag+1) - jd_ptr(diag)
  offset  = jd_ptr(diag) - 1
!$OMP PARALLEL DO
  do i=1, diagLen
    c(i) = c(i) + val(offset+i) * b(col_idx(offset+i))
  enddo
!$OMP END PARALLEL DO
enddo
```

- Long inner loop (max. N_r): OpenMP parallelization / vectorization
- Short outer loop (number of jagged diagonals)
- Multiple accesses to each element of result vector `c[]`
  - optimization potential!
- Stride-1 access to matrix data in `val[]`
- Indexed (indirect) access to RHS vector `b[]`

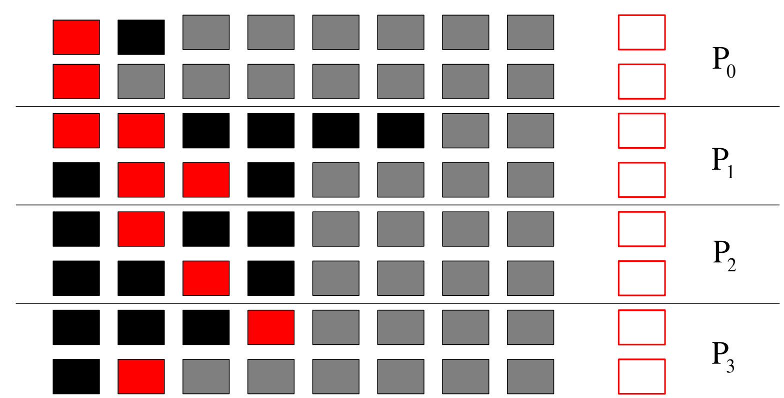
skipped

JDS Sparse MVM
MPI parallelization

Row-wise distribution

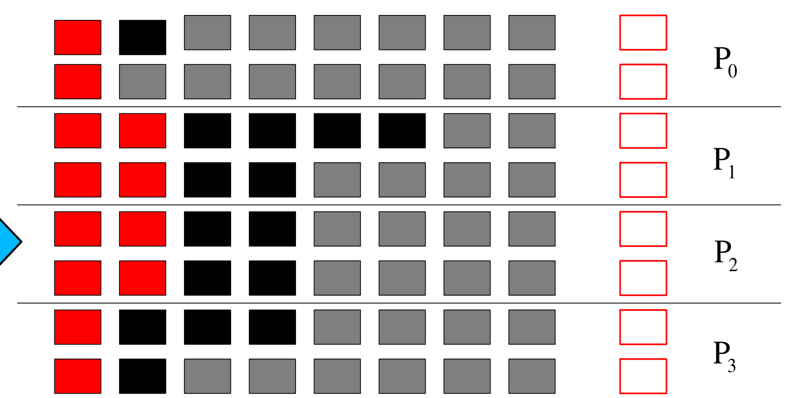


Each processor: local JDS (shift&order)



Avoid mixing of local and non-local diagonals:

1. Shift within local subblock
2. Fill local subblock with non-local elements from the right

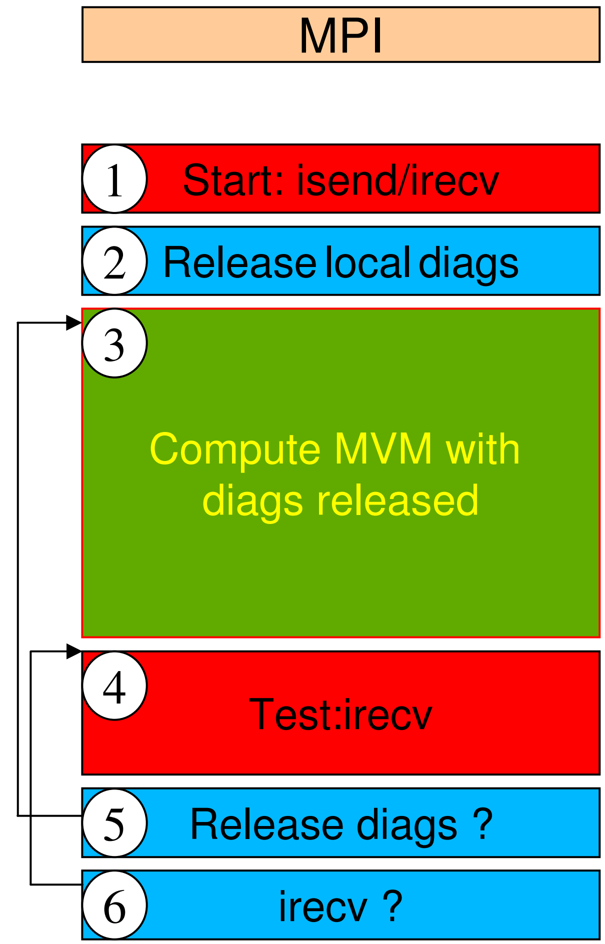


skipped

JDS Sparse MVM

Parallel MVM implementations: MPP

- One MPI process per processor
- Non-blocking MPI communication
- *Potential* overlap of communication and computation
  - However, MPI progress is only possible inside MPI calls on many implementations
- SMP Clusters: Intra-node and inter-node MPI



skipped

JDS Sparse MVM

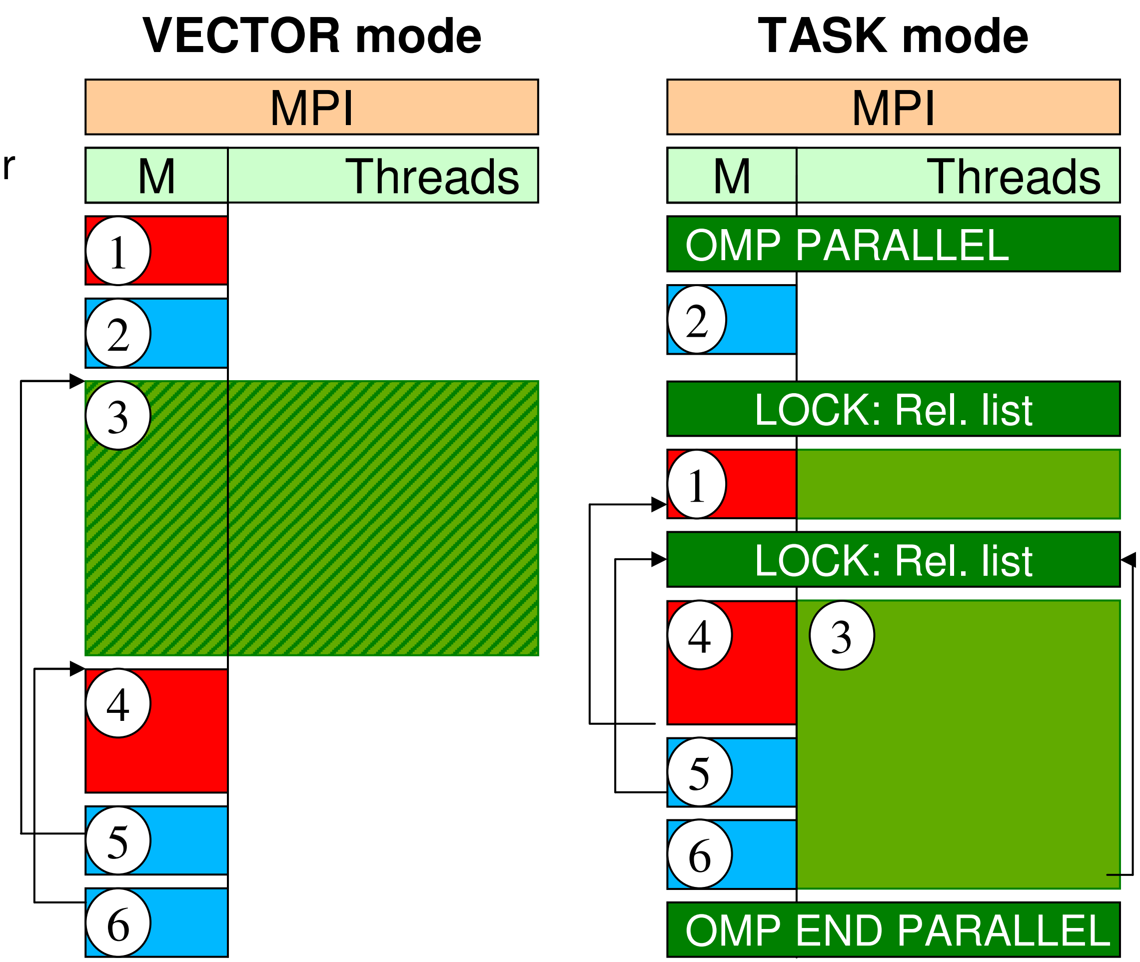
Parallel MVM implementations: Hybrid

VECTOR mode:

- Automatic parallel. of inner i loop (data parallel)
- Single threaded MPI calls

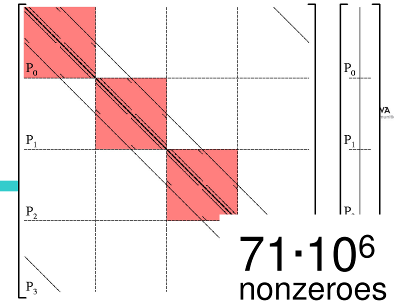
TASK mode:

- Functional parallelism: Simulate asynchronous data transfer! (OpenMP)
- Release list - LOCK
- Single threaded MPI calls
- Optional: Comm. Thread executes configurable fraction of work (load = 0...1)

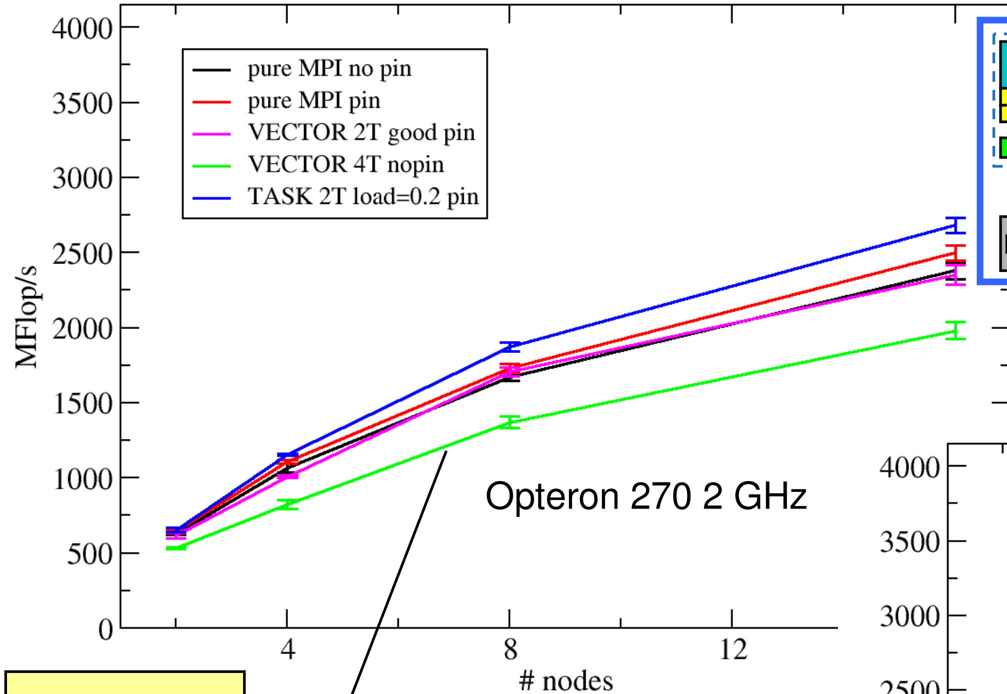


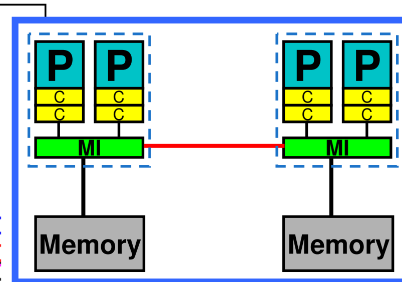
skipped

JDS Sparse MVM:
Performance and scalability on two different platforms



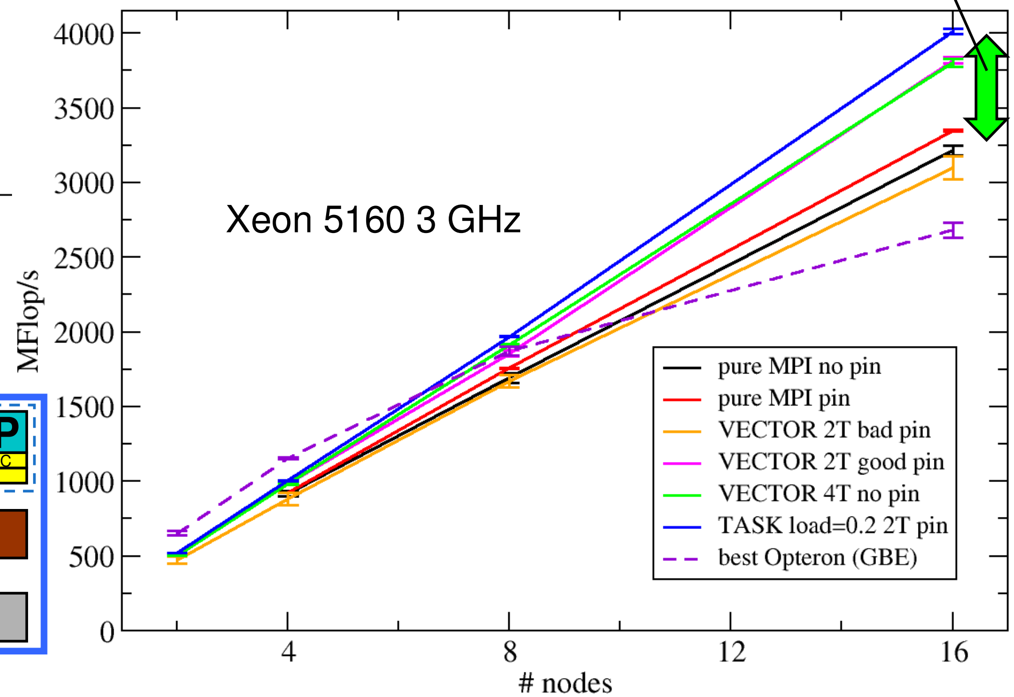
$71 \cdot 10^6$ nonzeroes





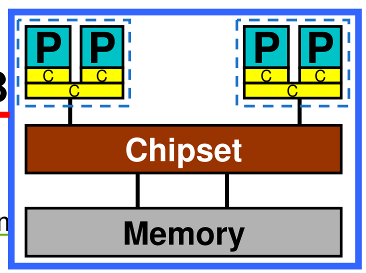
GBE

hybrid advantage



no NUMA placement!

SDR IB



MPI/OpenMP hybrid "how-to": Take-home messages

- Do not use hybrid if the pure MPI code scales ok

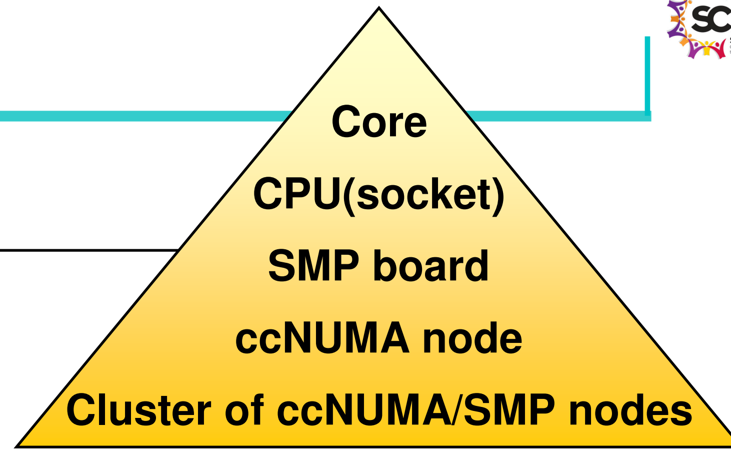
- Be aware of intranode MPI behavior
- Always observe the topology dependence of
  - Intranode MPI
  - OpenMP overheads
- Enforce proper thread/process to core binding, using appropriate tools (whatever you use, but use SOMETHING)
- Multi-LD OpenMP processes on ccNUMA nodes require correct page placement

- Finally: Always compare the best pure MPI code with the best OpenMP code!

Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / pure MPI vs hybrid MPI+OpenMP
- Practical “How-To” on hybrid programming
- **Mismatch Problems**
- Opportunities:
  Application categories that can benefit from hybrid parallelization
- Thread-safety quality of MPI libraries
- Tools for debugging and profiling MPI+OpenMP
- 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%

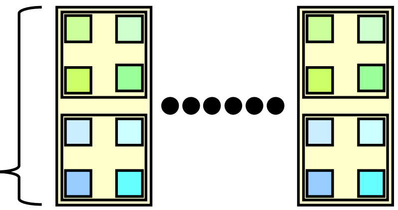
In most cases: **Both categories!**

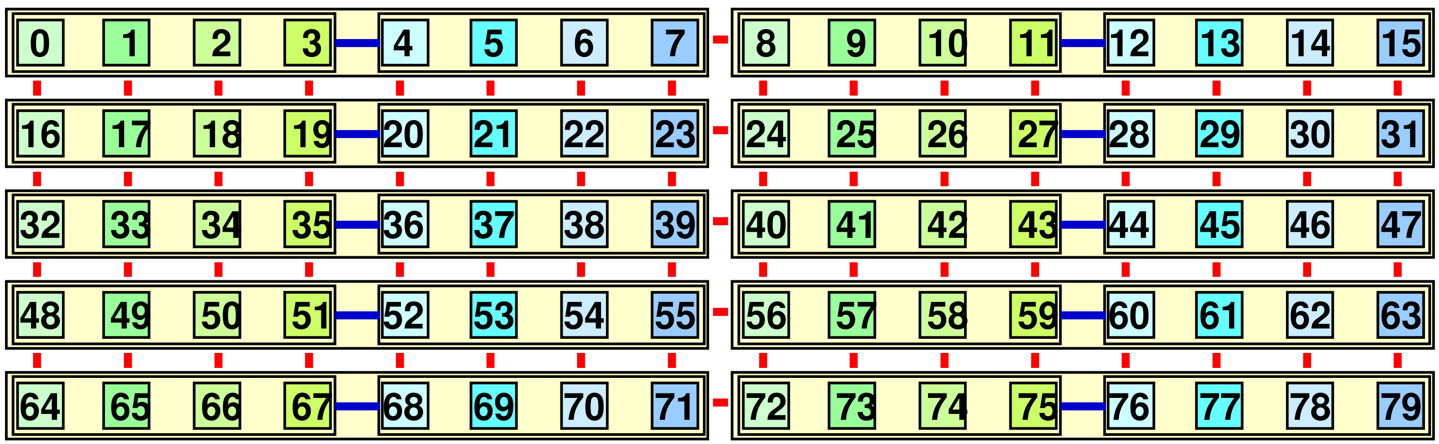
The Topology Problem with **pure MPI**

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

Sequential ranking of MPI_COMM_WORLD

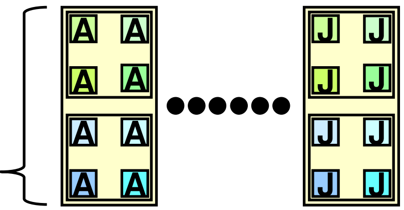
Does it matter?

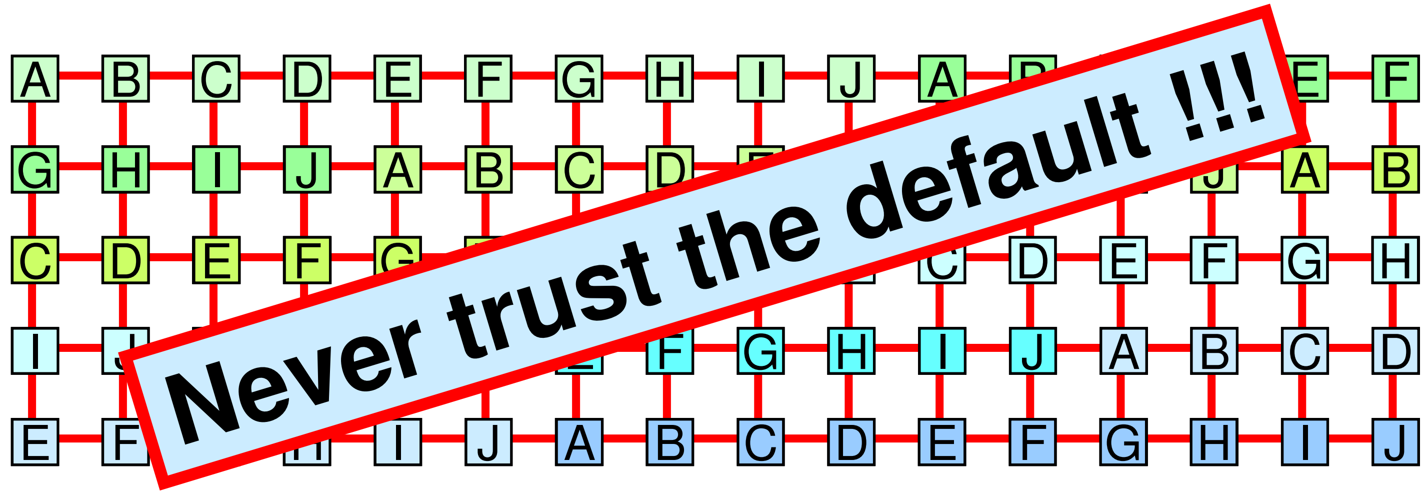
The Topology Problem with **pure MPI**

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





+ 32 x inter-node connections per node

– 0 x inter-socket connection per node

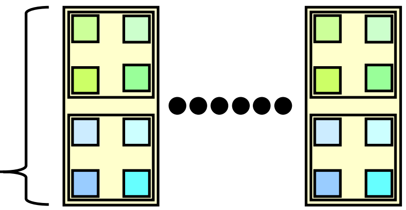
Round robin ranking of MPI_COMM_WORLD

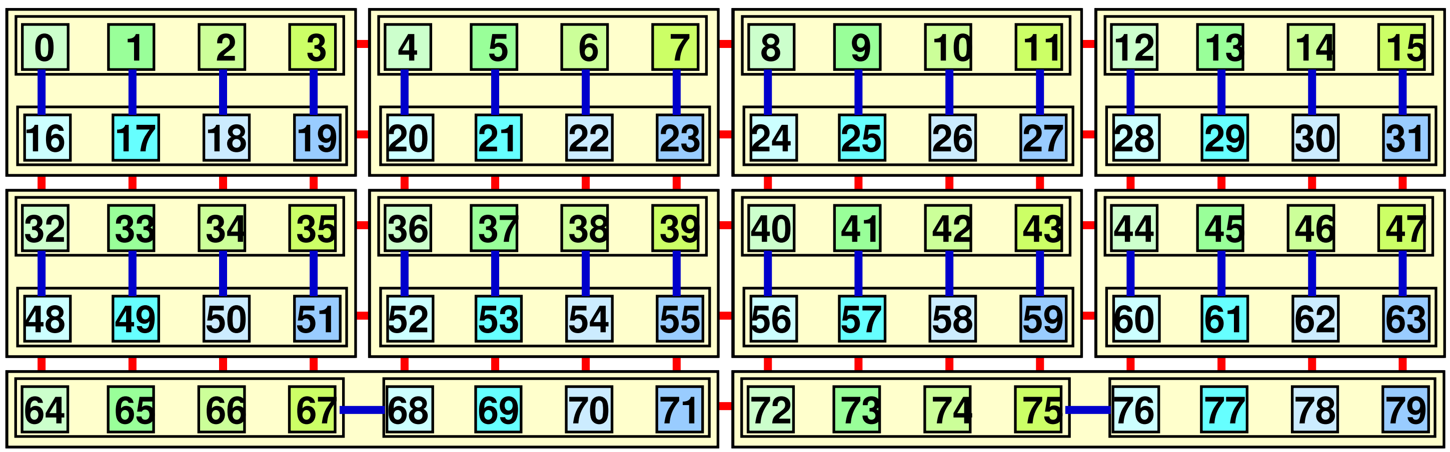
The Topology Problem with **pure MPI**

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

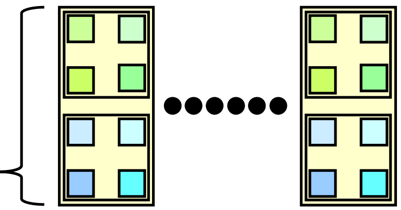
Bad affinity of cores to thread ranks

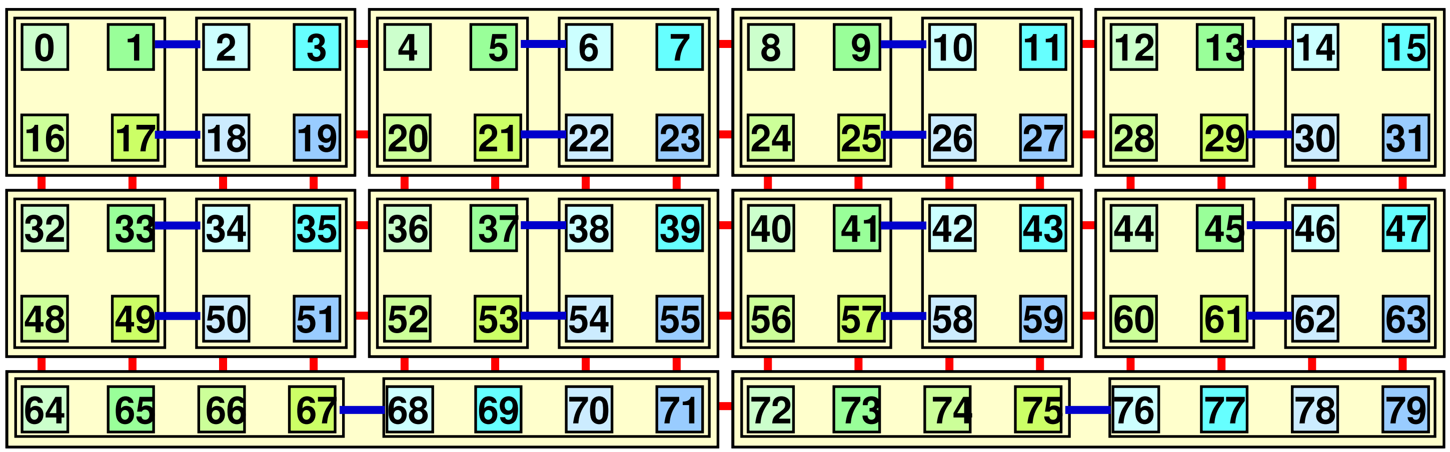
The Topology Problem with **pure MPI**

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

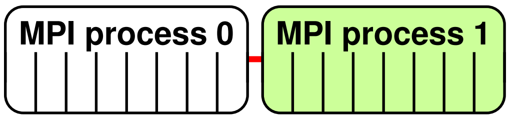
skipped

The Topology Problem with hybrid MPI+OpenMP

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

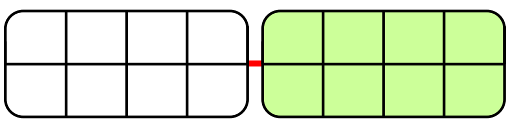
Exa.: 2 SMP nodes, 8 cores/node

Optimal ?



Loop-worksharing on 8 threads

Optimal ?



Minimizing ccNUMA data traffic through domain decomposition inside of each MPI process

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)

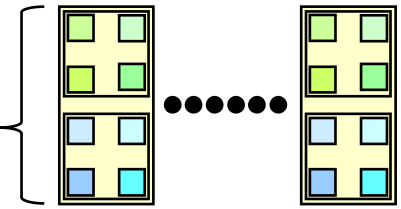
skipped

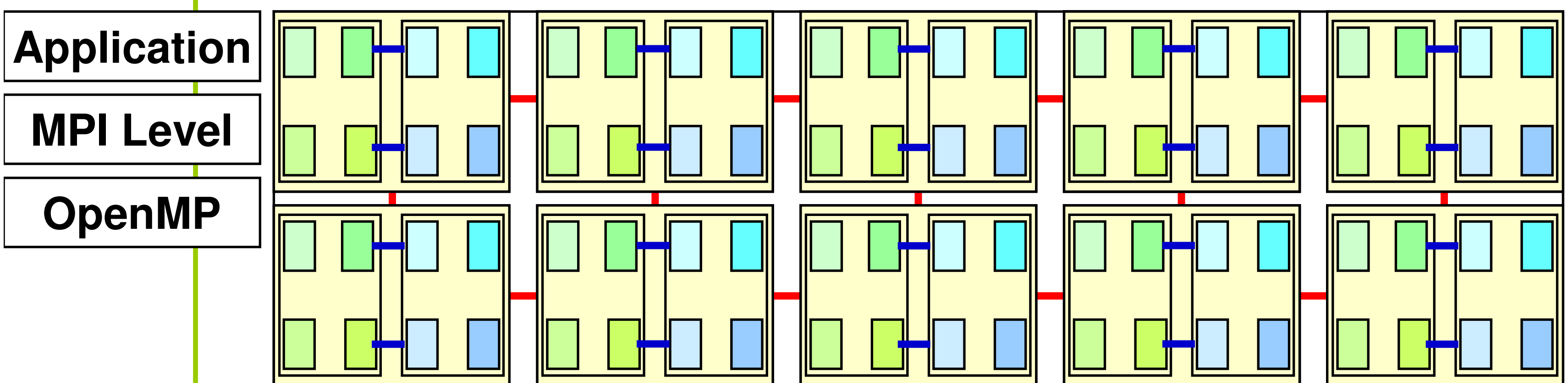
The Topology Problem with hybrid MPI+OpenMP

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



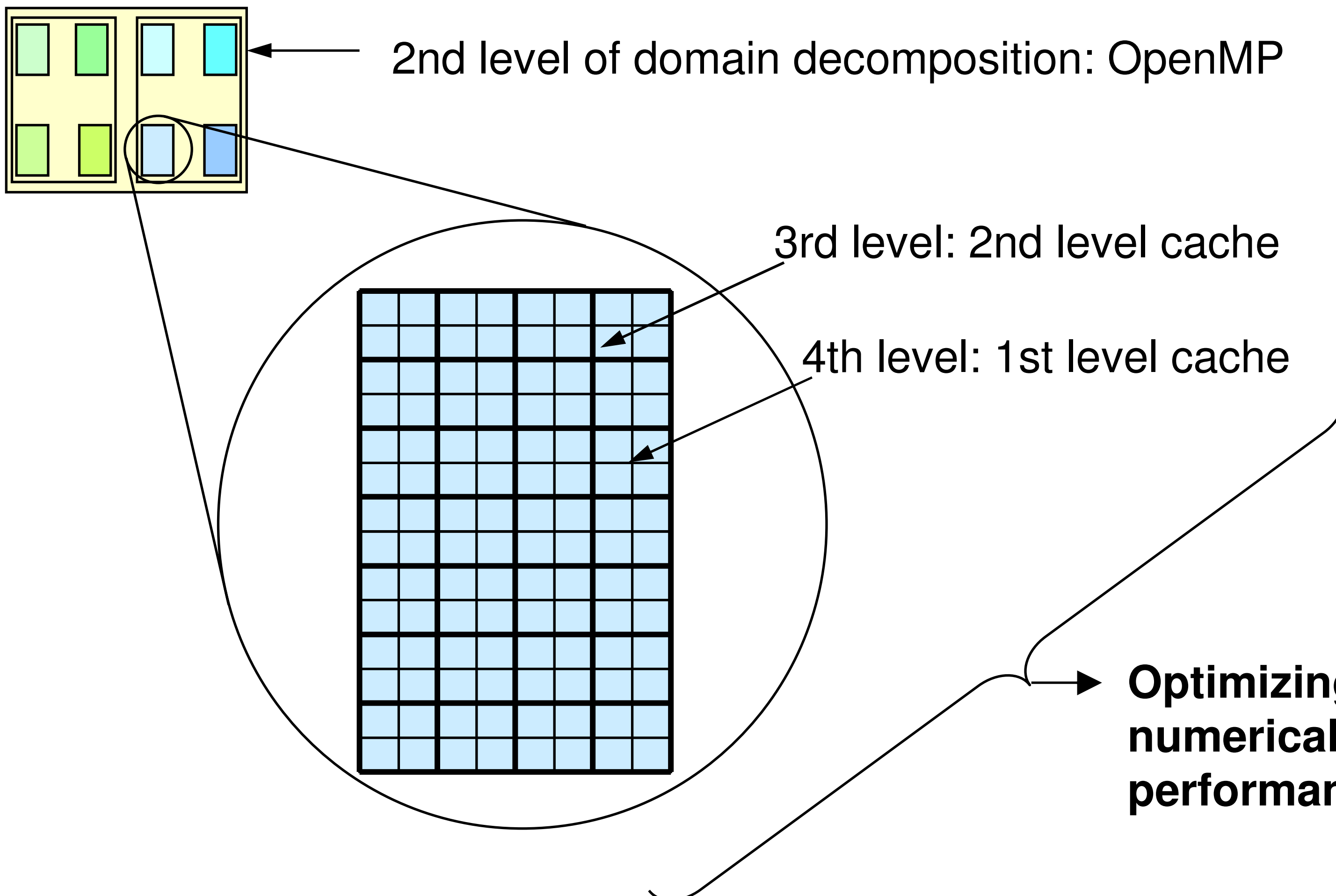


+ 3 x inter-node connections per node, but ~ 4 x more traffic
+ 2 x inter-socket connection per node

Affinity of cores to thread ranks !!!

skipped

Numerical Optimization inside of an SMP node

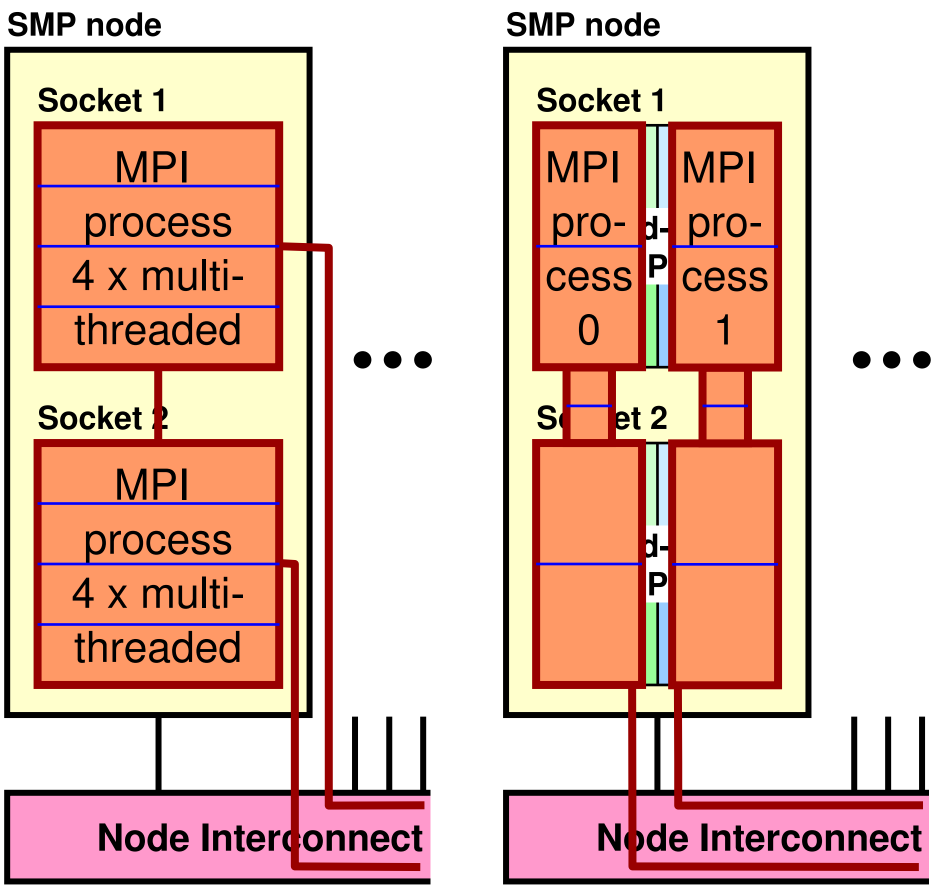


Optimizing the numerical performance

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?

[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:

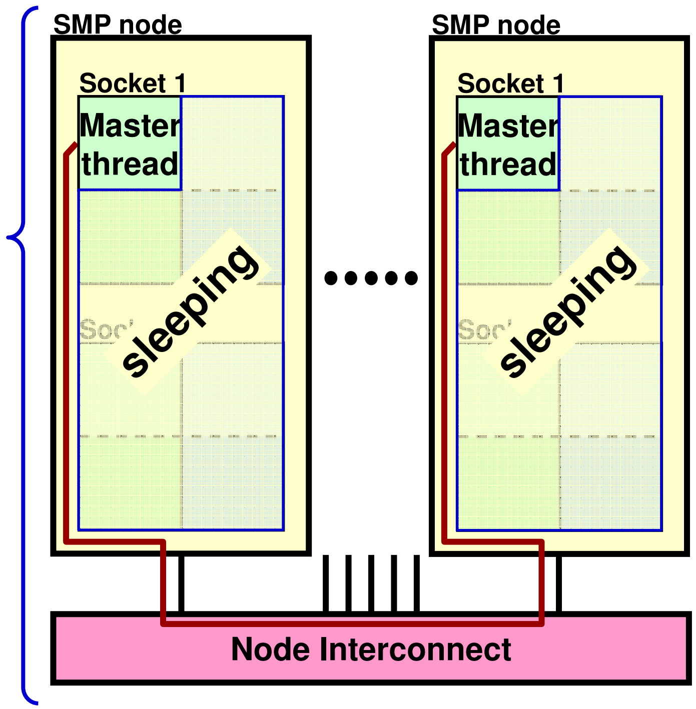
- MPI library must use appropriate fabrics / protocol for intra-node communication
- Intra-node bandwidth higher than inter-node bandwidth
  → problem may be small
- MPI implementation may cause unnecessary data copying
  → waste of memory bandwidth

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**

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

- 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?

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++) {
            ….
   }
}
```

skipped

Overlapping Communication and Computation

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

Subteams

Not yet part of the OpenMP standard

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

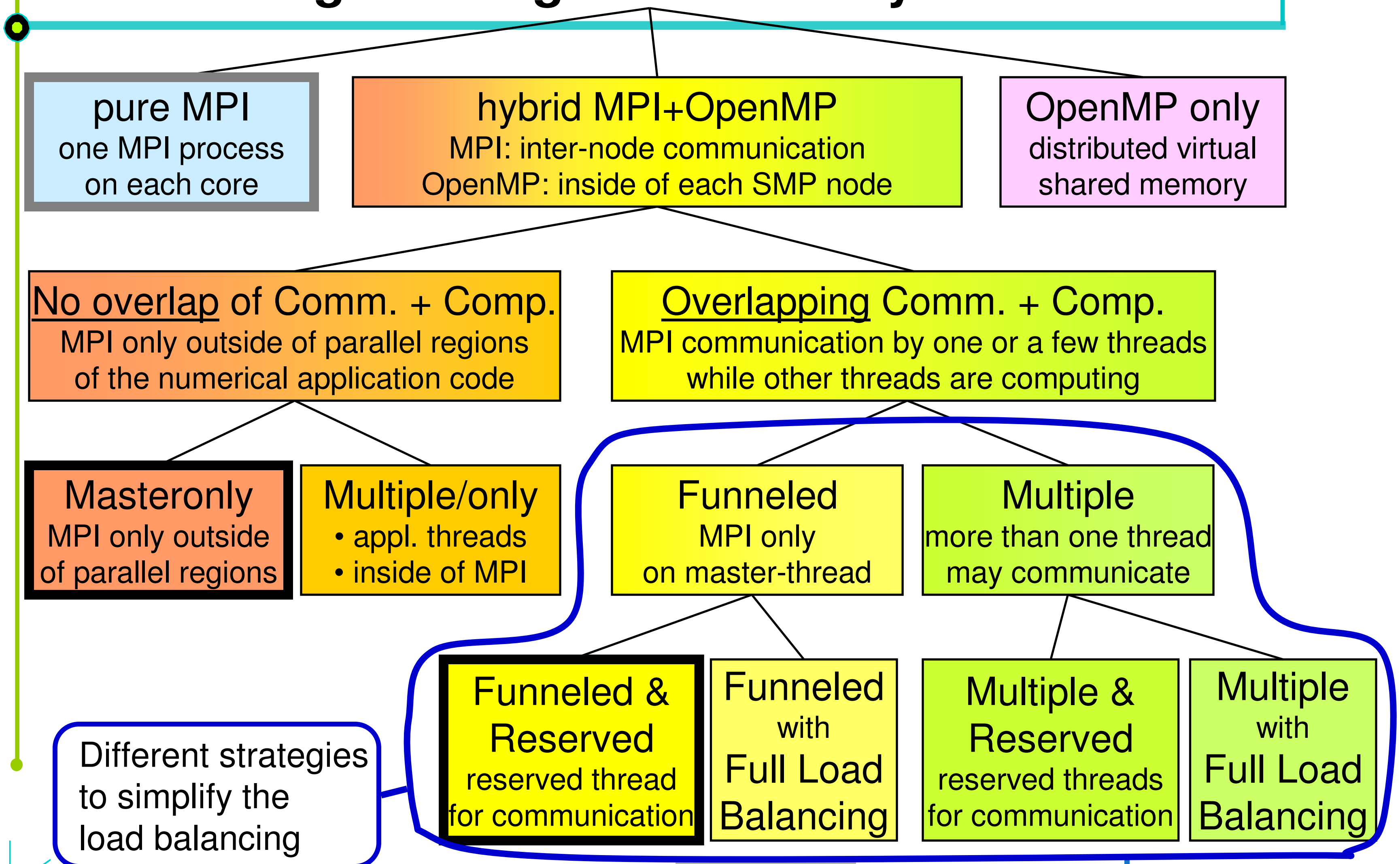
Barbara Chapman et al.:

Toward Enhancing OpenMP's Work-Sharing Directives.

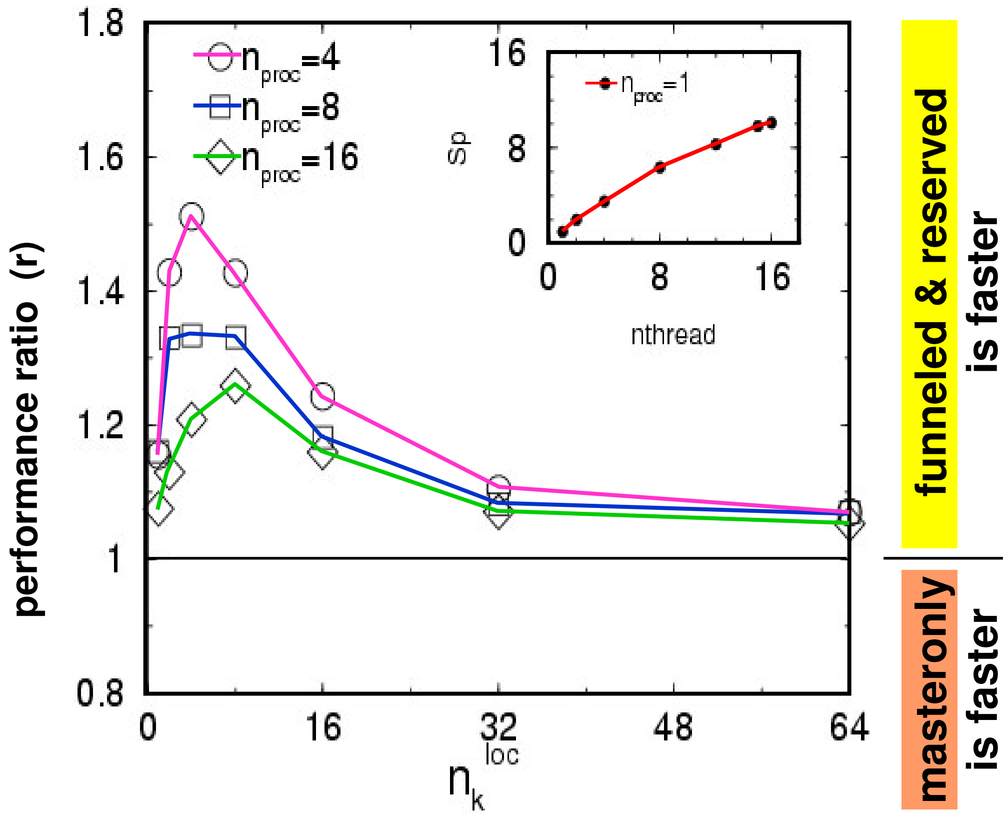
In proceedings, W.E. Nagel et al. (Eds.): Euro-Par 2006, LNCS 4128, pp. 645-654, 2006.

```
#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



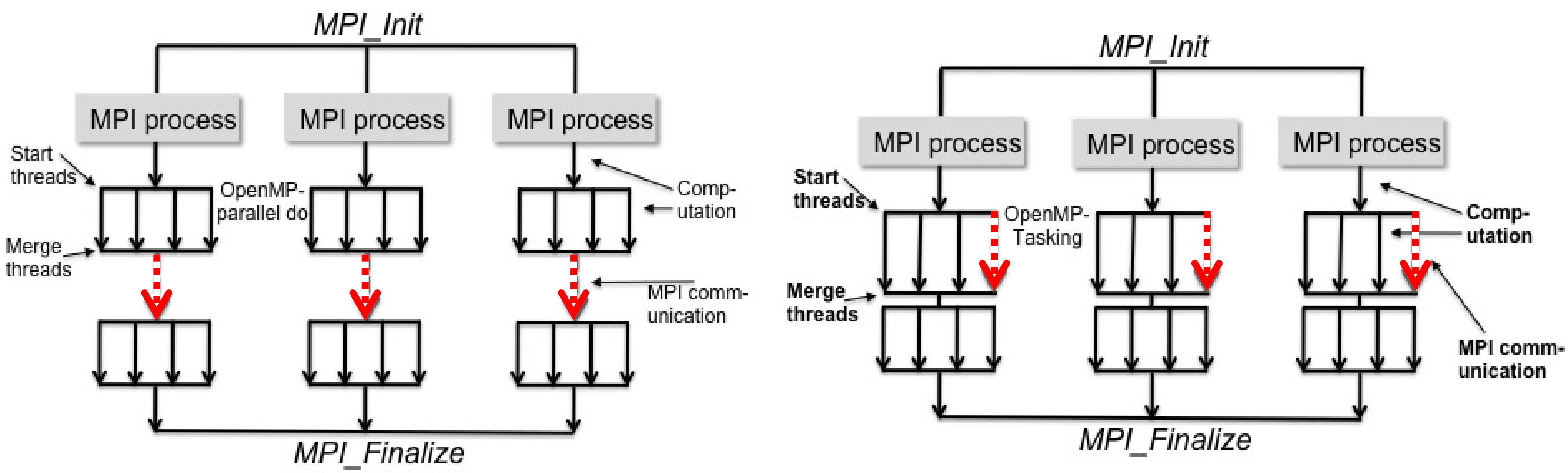
Experiment: Matrix-vector-multiply (MVM)



- 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

skipped

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
```

```
!reorder remaining particles: fill holes
  fill_hole(p_array);

!send number of particles to move right
  MPI_SENDRECV(x,length=2,..);
!send to right and receive from left
  MPI_SENDRECV(sendright,length=g(x),..);
!send number of particles to move left
  MPI_SENDRECV(y,length=2,..);
!send to left and receive from right
  MPI_SENDRECV(sendleft,length=g(y),..);

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

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

skipped

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

skipped

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

SC11 Seattle, WA

```
!$omp parallel
!$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
```

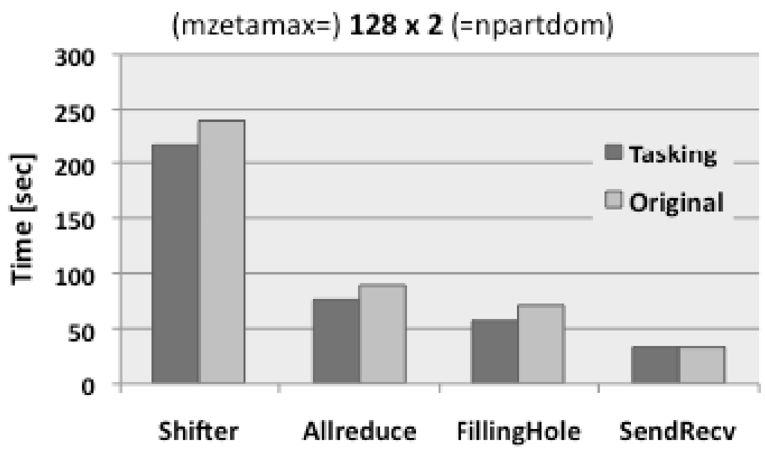
Overlapping remaining MPI_Sendrecv

Slides, courtesy of Alice Koniges, NERSC, LBNL

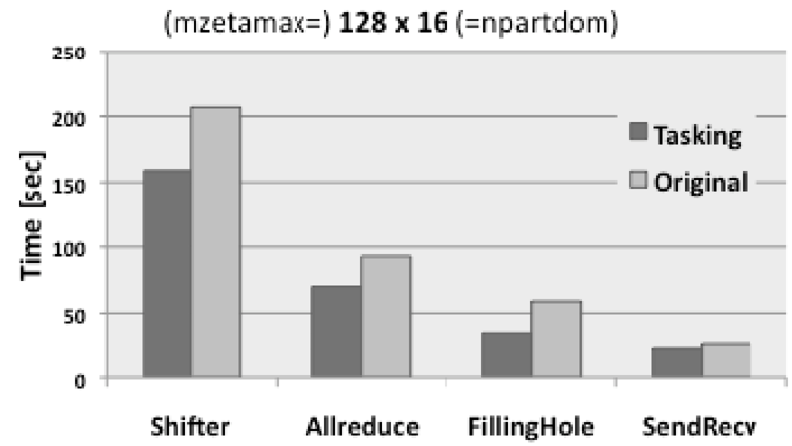
skipped

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.

Slides, courtesy of Alice Koniges, NERSC, LBNL

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

skipped

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

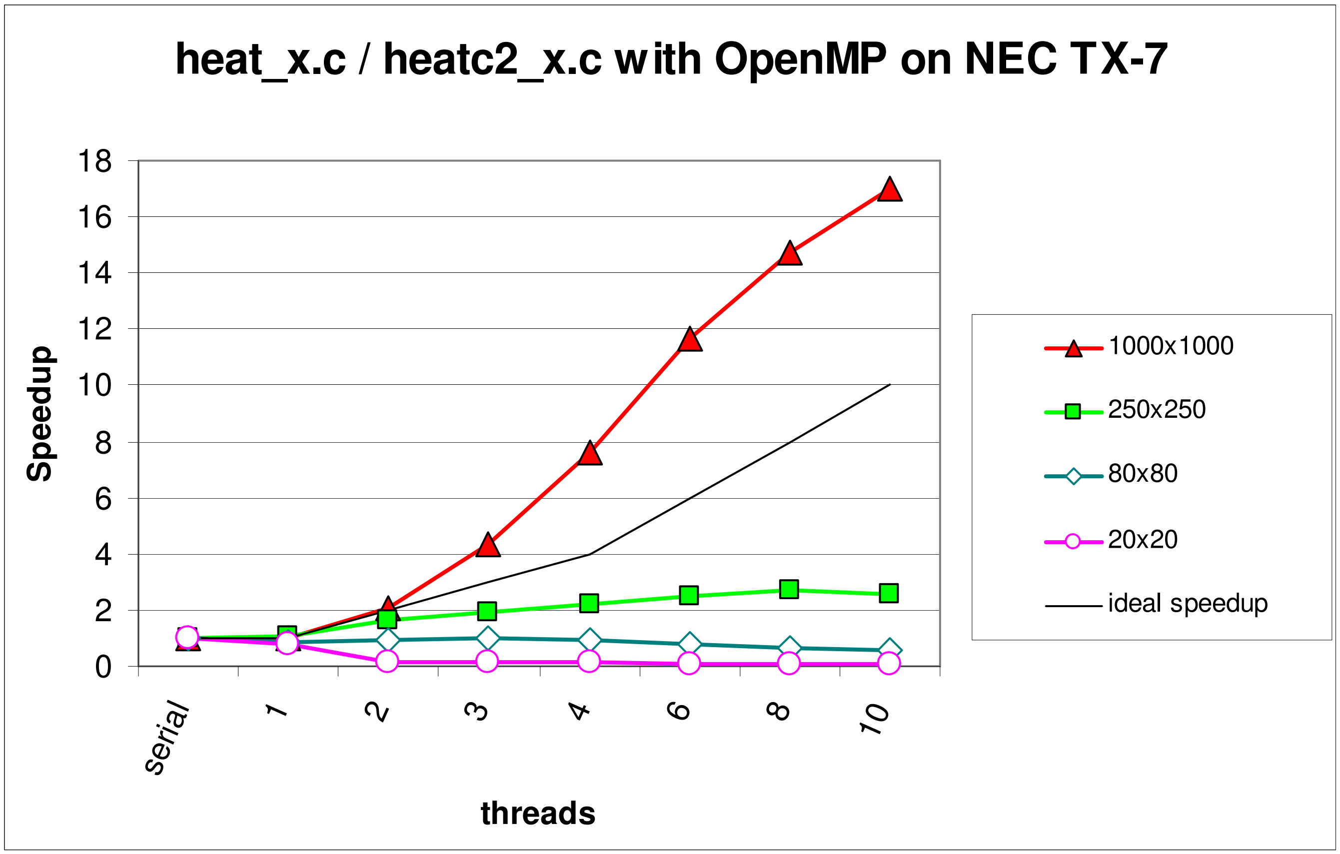
by rule of thumb:

Communication may be 10 times slower than with MPI

skipped

Comparing results with heat example

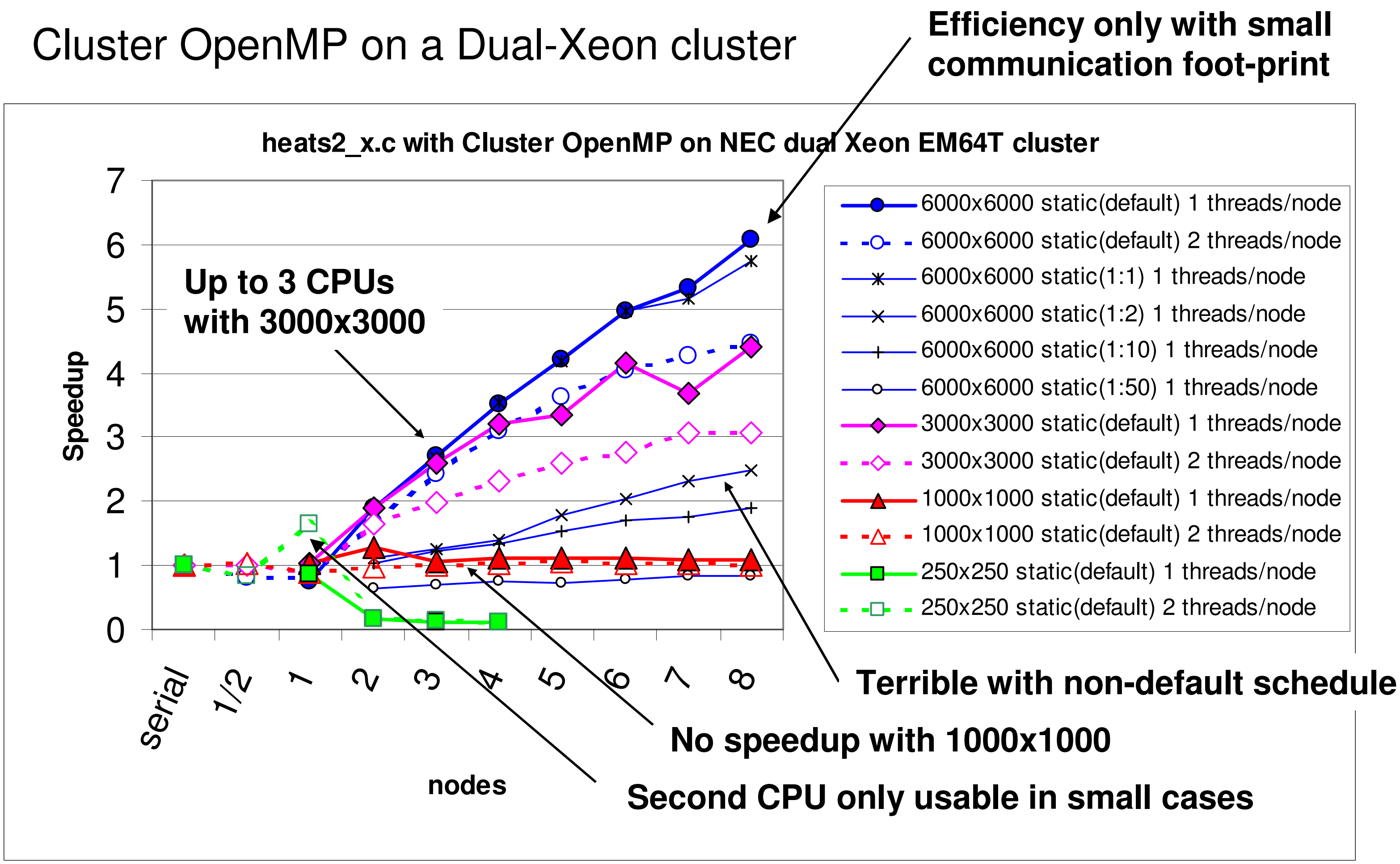
- Normal OpenMP on shared memory (ccNUMA) NEC TX-7



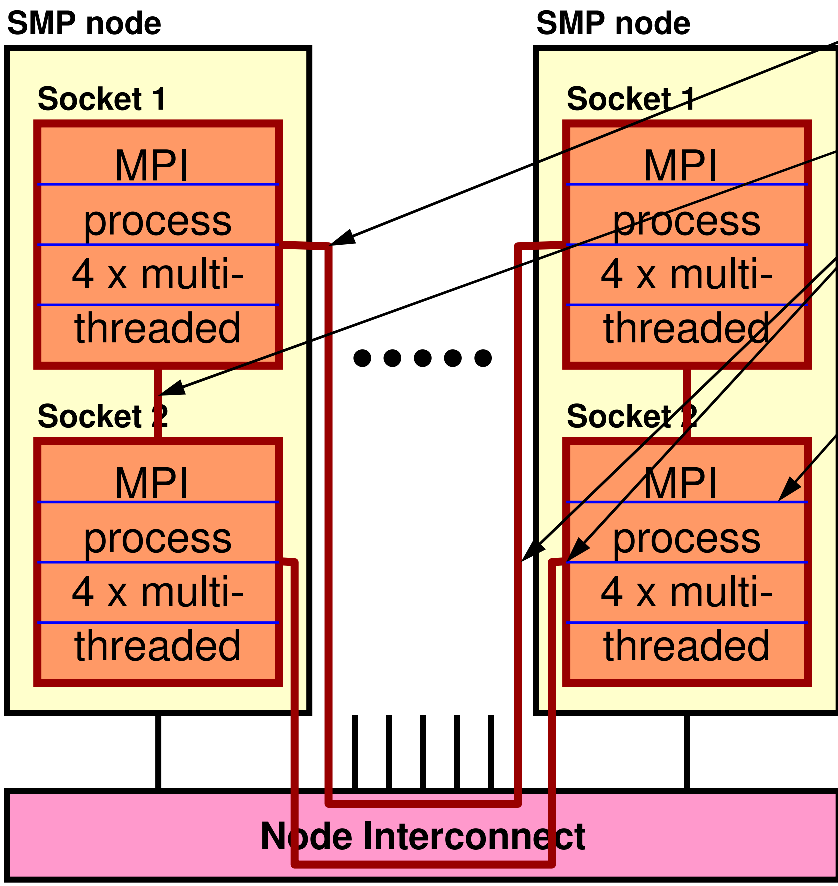
skipped

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 / pure MPI vs hybrid MPI+OpenMP
- Practical "How-To" on hybrid programming
- Mismatch Problems
- **Opportunities:**
  Application categories that can benefit from hybrid parallelization
- Thread-safety quality of MPI libraries
- Tools for debugging and profiling MPI+OpenMP
- 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 <** $\frac{\textbf{Sum of workload of all zones}}{\textbf{Max workload of a zone}}$
  - 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

```
#pragma omp parallel for schedule(dynamic)
for (i=0; i<n; i++) {
  /* poorly balanced iterations */ …
}
```

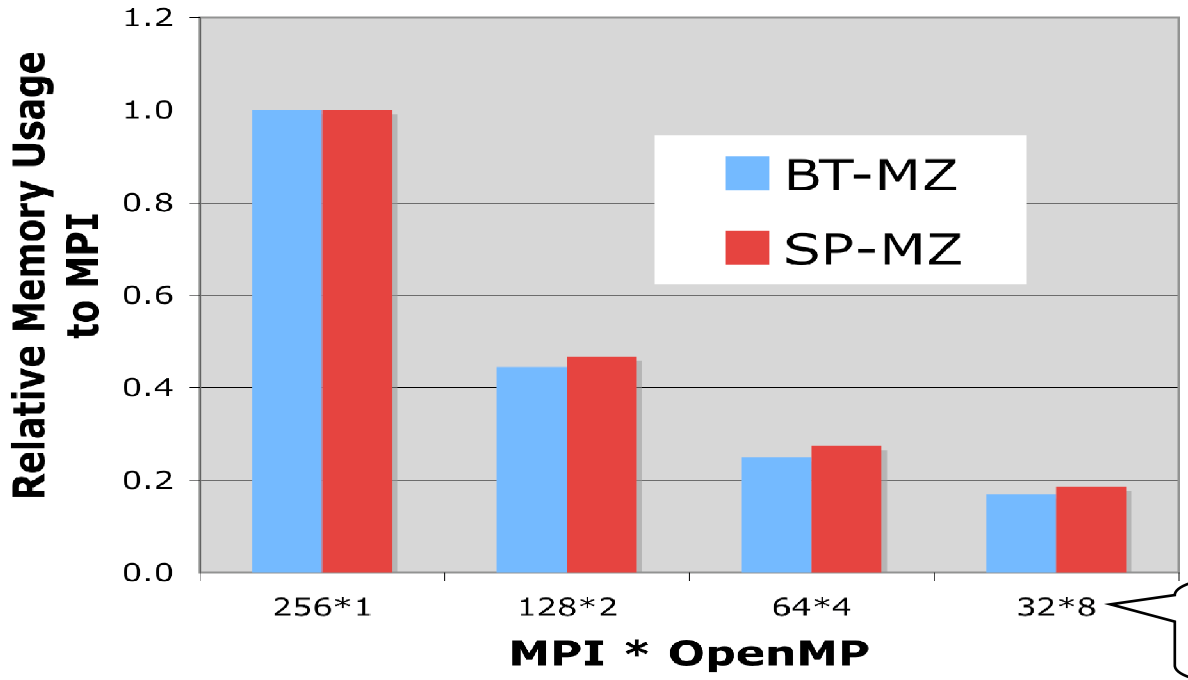
- On MPI level
  - **Dynamic load balancing** requires moving of parts of the data structure through the network
  - Significant runtime overhead
  - Complicated software / therefore not implemented

- **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 Platorms. Proceedings, CUG 2010, Edinburgh, GB, May 24-27, 2010.

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 / pure MPI vs hybrid MPI+OpenMP
- Practical "How-To" on hybrid programming
- Mismatch Problems
- Opportunities:
  Application categories that can benefit from hybrid parallelization
- **Thread-safety quality of MPI libraries**
- Tools for debugging and profiling MPI+OpenMP
- 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: (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

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!

skipped

… the barrier is necessary – example with MPI_Recv

```fortran
!$OMP PARALLEL
!$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
```

```c
#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 */
```

skipped

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"
MPIch2-1.1.0a2	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

Courtesy of Rainer Keller, HLRS and ORNL

skipped

Thread support within Open MPI

- In order to enable thread support in Open MPI, configure with:

```
configure --enable-mpi-threads
```

- This turns on:
  - Support for full `MPI_THREAD_MULTIPLE`
  - internal checks when run with threads (`--enable-debug`)

```
configure --enable-mpi-threads --enable-progress-threads
```

- This (additionally) turns on:
  - Progress threads to asynchronously transfer/receive data per network BTL.
- Additional Feature:
  - Compiling **with** debugging support, but **without** threads will check for recursive locking

Courtesy of Rainer Keller, HLRS and ORNL

Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / pure MPI vs hybrid MPI+OpenMP
- Practical “How-To” on hybrid programming
- Mismatch Problems
- Opportunities:
  Application categories that can benefit from hybrid parallelization
- Thread-safety quality of MPI libraries
- **Tools for debugging and profiling MPI+OpenMP**
- Other options on clusters of SMP nodes
- Summary

Thread Correctness – Intel ThreadChecker 1/3

- Intel ThreadChecker operates in a similar fashion to `helgrind`,
- Compile with `-tcheck`, then run program using `tcheck_cl`:

With new Intel Inspector XE 2011: Command line interface for mpirun is undocumented

```
Application finished

_______________________________________________________________________________
|ID|Short De|Sever|C|Contex|Description                       |1st Acc|2nd Acc|
|  |scriptio|ity  |o|t[Best|                                  |ess[Bes|ess[Bes|
|  |n       |Name |u|]     |                                  |t]     |t]     |
|  |        |     |n|      |                                  |       |       |
|  |        |     |t|      |                                  |       |       |
_______________________________________________________________________________
|1 |Write ->|Error|1|"pthre|Memory write of global_variable at|"pthrea|"pthrea|
|  |Write da|     | |ad_rac|"pthread_race.c":31 conflicts with|d_race.|d_race.|
|  |ta-race |     | |e.c":2|a prior memory write of           |c":31  |c":31  |
|  |        |     | |5     |global_variable at                |       |       |
|  |        |     | |      |"pthread_race.c":31 (output       |       |       |
|  |        |     | |      |dependence)                       |       |       |
_______________________________________________________________________________
```


Courtesy of Rainer Keller, HLRS and ORNL

skipped

Thread Correctness – Intel ThreadChecker 2/3

- One may output to HTML:

```
tcheck_cl --format HTML --report pthread_race.html pthread_race
```

Thread Checker Output - Konqueror

Dokument Bearbeiten Ansicht Gehe zu Lesezeichen Extras Einstellungen Fenster Hilfe

Adresse: /home/hpcraink/C/PTHREAD/DEBUGGING/pthread_race_pcglap12.html

Google-Suche

ID	Short Description	Severity Name	Count	Context[Best]	Description	1st Access[Best]	2nd Access[Best]
1	Write -> Write data-race	Error	1	"pthread_race.c":25	Memory write of global_variable at "pthread_race.c":31 conflicts with a prior memory write of global_variable at "pthread_race.c":31 (output dependence)	"pthread_race.c":31	"pthread_race.c":31
2	Thread termination	Information	1	Whole Program 1	Thread termination at "pthread_race.c":43 - includes stack allocation of 8,004 MB and use of 4,672 KB	"pthread_race.c":43	"pthread_race.c":43
3	Thread termination	Information	1	Whole Program 2	Thread termination at "pthread_race.c":43 - includes stack allocation of 8,004 MB and use of 4,672 KB	"pthread_race.c":43	"pthread_race.c":43
4	Thread termination	Information	1	Whole Program 3	Thread termination at "pthread_race.c":37 - includes stack allocation of 8 MB and use of 4,25 KB	"pthread_race.c":37	"pthread_race.c":37

Courtesy of Rainer Keller, HLRS and ORNL

skipped

Thread Correctness – Intel ThreadChecker 3/3

- If one wants to compile with threaded Open MPI (option for IB):

```
configure --enable-mpi-threads
          --enable-debug
          --enable-mca-no-build=memory-ptmalloc2
      CC=icc F77=ifort FC=ifort
      CFLAGS='-debug all -inline-debug-info tcheck'
      CXXFLAGS='-debug all -inline-debug-info tcheck'
      FFLAGS='-debug all -tcheck'    LDFLAGS='tcheck'
```

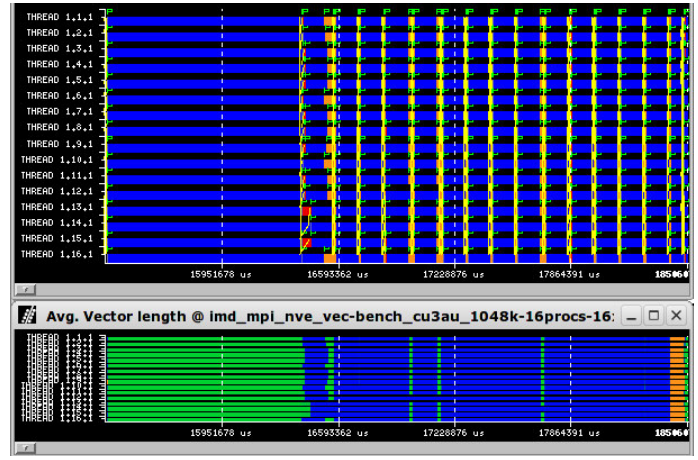
- Then run with:

```
mpirun --mca tcp,sm,self -np 2 tcheck_cl              \
    --reinstrument -u full --format html              \
    --cache_dir '/tmp/my_username_$$__tc_cl_cache'    \
    --report 'tc_mpi_test_suite_$$'                   \
    --options 'file=tc_my_executable_%H_%I,           \
               pad=128, delay=2, stall=2'         --  \
./my_executable my_arg1 my_arg2 …
```

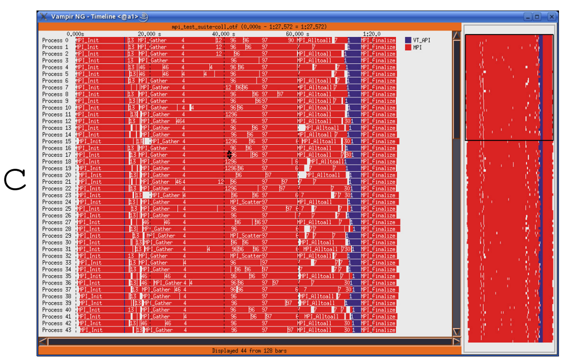

Courtesy of Rainer Keller, HLRS and ORNL

Performance Tools Support for Hybrid Code

- Paraver examples have already been shown, tracing is done with linking against (closed-source) `omptrace` or `ompitrace`

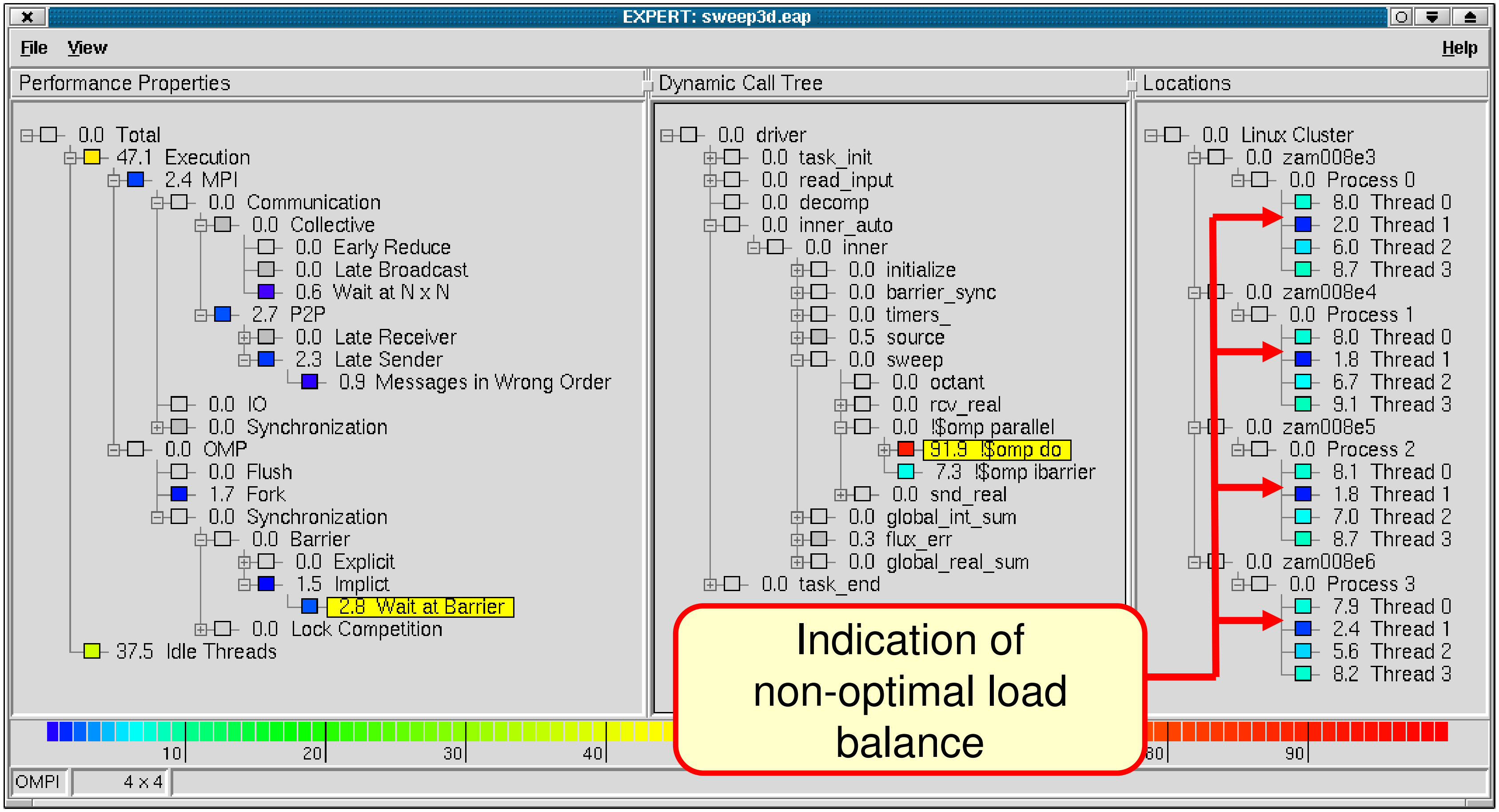


- For Vampir/Vampirtrace performance analysis:
```
./configure –enable-omp
  –enable-hyb
  –with-mpi-dir=/opt/OpenMPI/1.3-icc
CC=icc F77=ifort FC=ifort
```
(Attention: does not wrap `MPI_Init_thread`!)



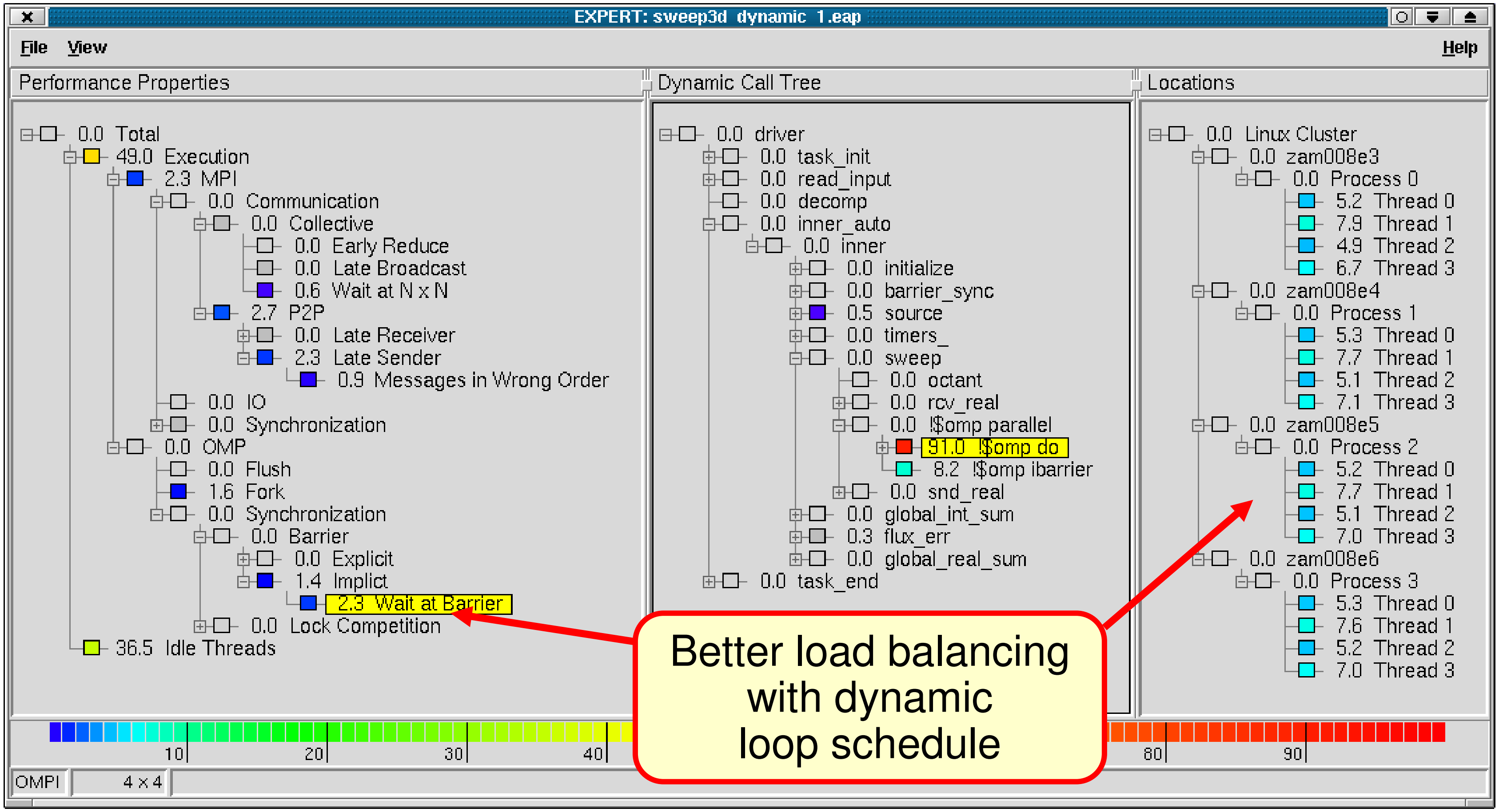
Courtesy of Rainer Keller, HLRS and ORNL

Scalasca – Example “Wait at Barrier”



Screenshots, courtesy of KOJAK JSC, FZ Jülich

Scalasca – Example “Wait at Barrier”, Solution



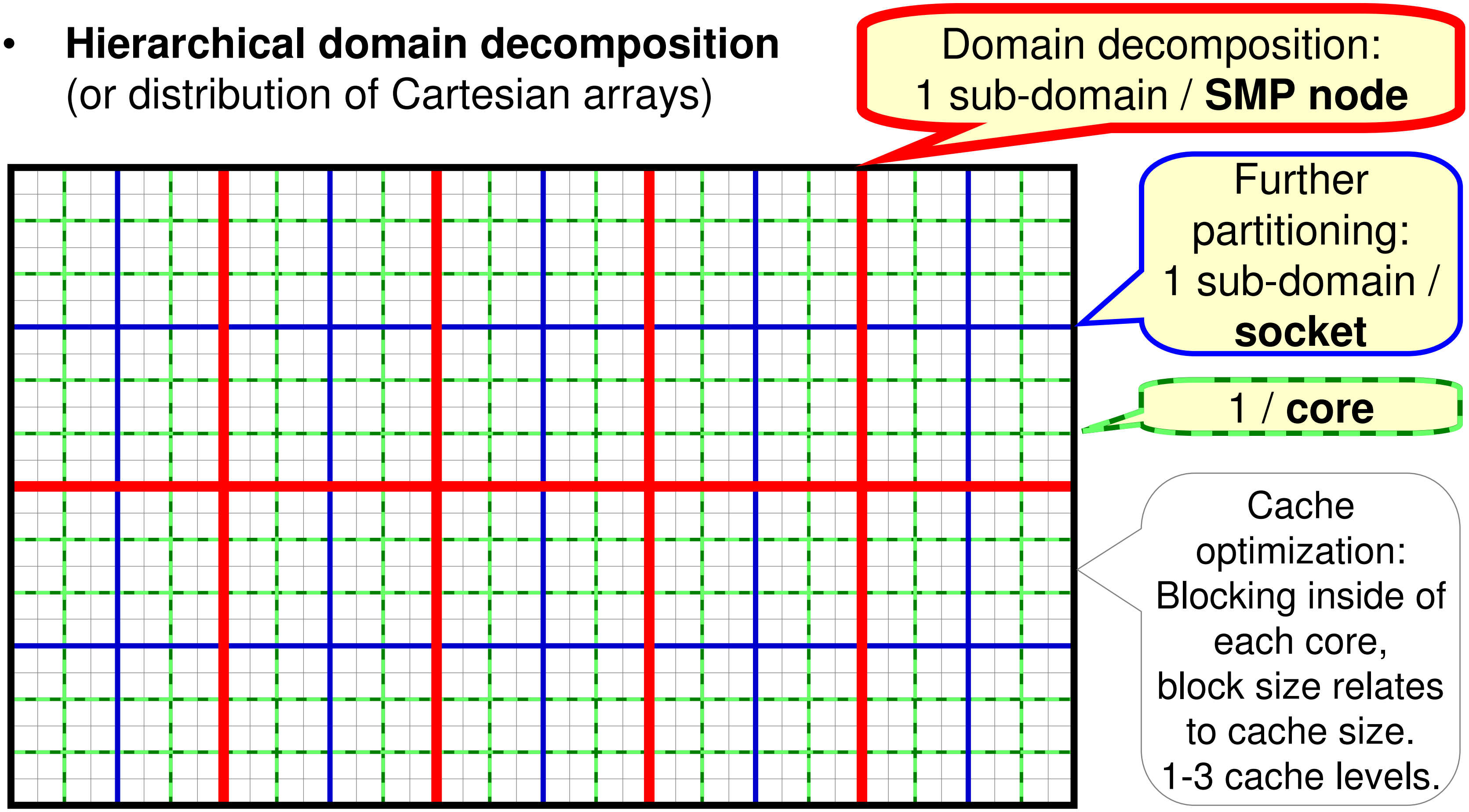
Screenshots, courtesy of KOJAK JSC, FZ Jülich

Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / pure MPI vs hybrid MPI+OpenMP
- Practical “How-To” on hybrid programming
- Mismatch Problems
- Opportunities:
  Application categories that can benefit from hybrid parallelization
- Thread-safety quality of MPI libraries
- Tools for debugging and profiling MPI+OpenMP
- **Other options on clusters of SMP nodes**
  - **Pure MPI – multi-core aware** **(Rolf Rabenseifner)**
  - **Remarks on MPI scalability / Cache Optimization / Cost-benefit /PGAS** **(R.R.)**
  - **Hybrid programming and accelerators** **(Gabriele Jost)**
- 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.

skipped

How to achieve a hierarchical domain decomposition (DD)?

Implementation hints to previous slide

- **Cartesian grids:**
  - Several levels of subdivide
  - Ranking of MPI_COMM_WORLD – three 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.

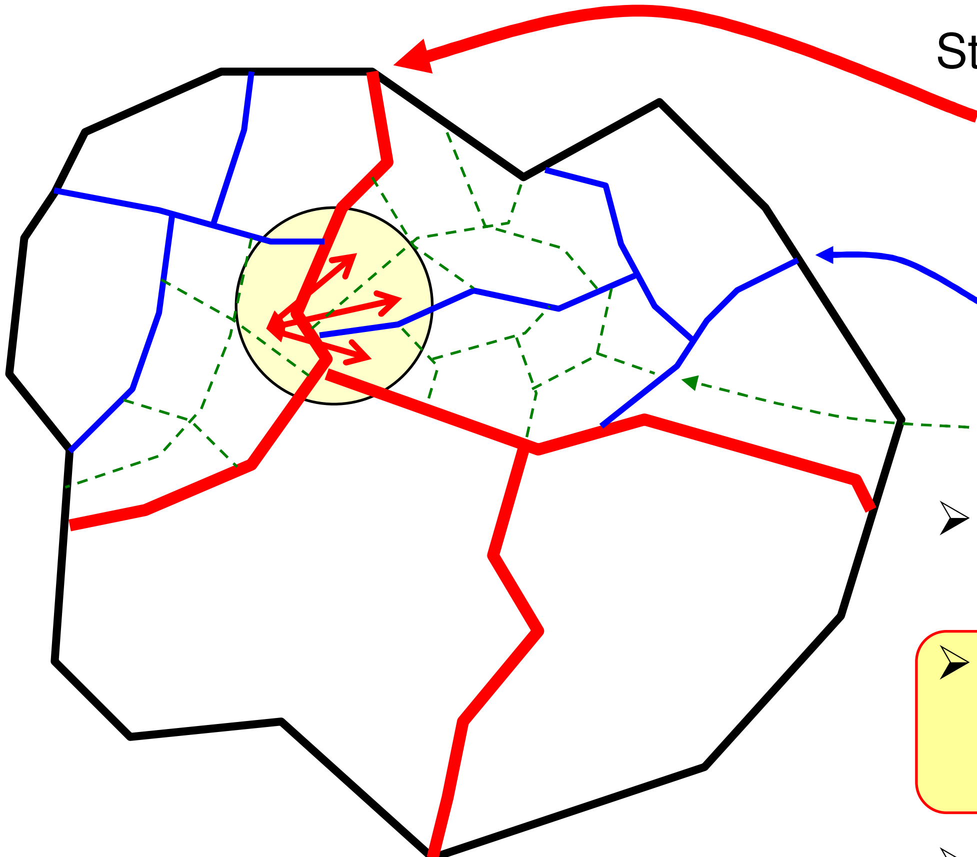
- **Unstructured grids:**
  → next slide

How to achieve a hierarchical domain decomposition (DD)?

- **Unstructured grids:**
  - Multi-level DD:
    - Top-down: Several levels of (Par)Metis
      → unbalanced communication
      → demonstrated on next (skipped) slide
    - Bottom-up: Low level DD
      + higher level recombination
      → based on DD of the grid of subdomains
  - 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

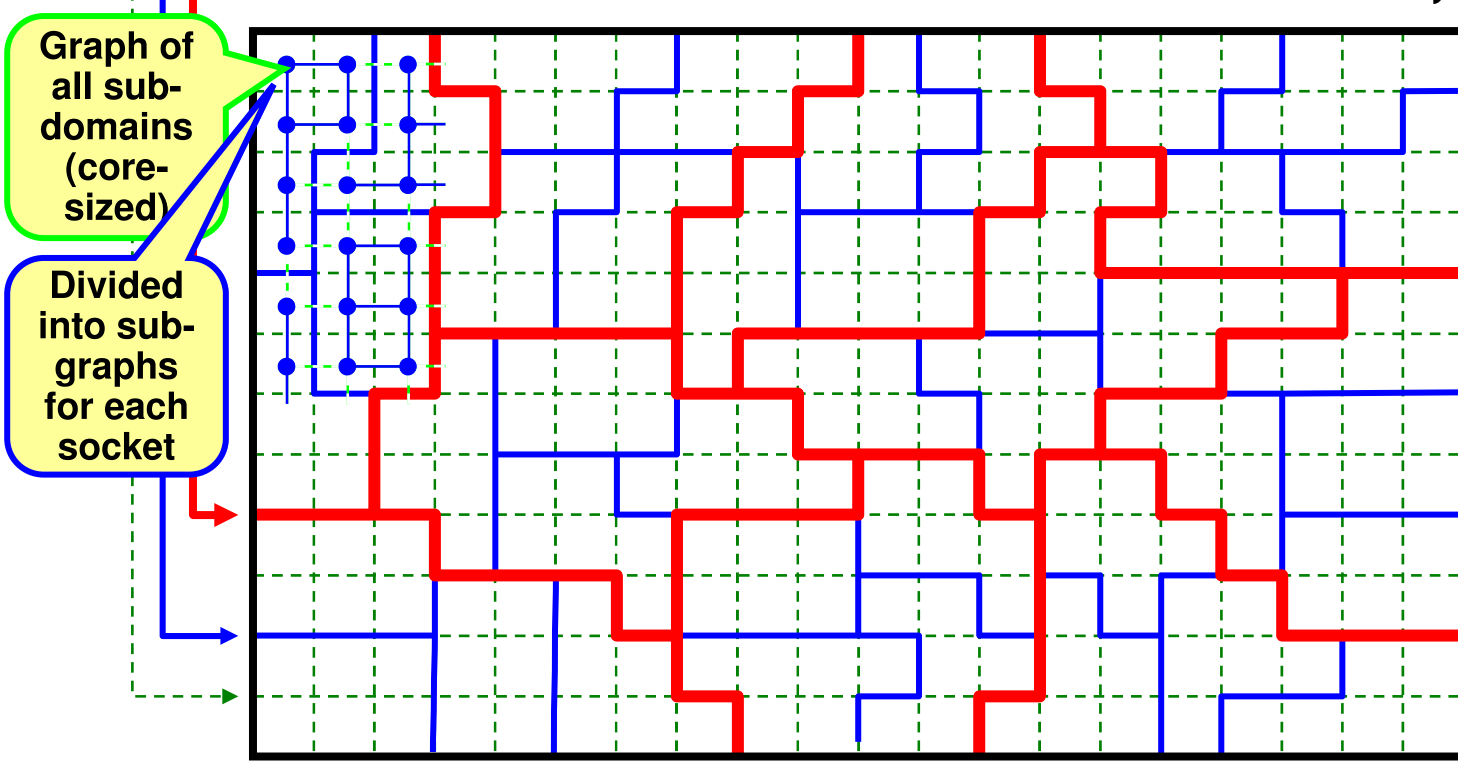
➢ Subdivide does not care on balancing of the outer boundary

➢ 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!

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

Hybrid Programming and Accelerators

- **Under Discussion: OpenMP for Accelerators!**
  – Extensions to support usage of accelerators under discussion for OpenMP 4.0
- **Current Memory Model:**
  – Relaxed-Consisteny Shared-Memory
  – All threads have access to the memory
  – Data-sharing attributes: shared, private
- **Proposed Additions to Memory Model**
  – Separate Host and Accelerator Memory
  – Data Movement Host<->Accelerator indicated by compiler directives
  – Updates to different memoris indicated by compiler directives

Hybrid Programming and Accelerators (cont)

- **Current OpenMP Execution Model:**
  - Execution starts single threaded
  - Fork-Join Threads at OpenMP parallel regions
  - Work-sharing indicated via compiler directives
- **Proposed additions to the Execution Model:**
  - Explicit accelerator tasks are generated at beginning of accelerator regions

Examples of proposed new directives

- **`#pragma acc_region [clause]`**
  - Purpose: Define code that is to be run on accelerator
  - `acc_copyin (list)`
  - `acc_copyout (list)`
  - …
- **`#pragma omp acc_data [clause]`**
  - Purpose: Define data scope across multiple accelerator constructs.
  - `acc_shared`
  - `acc_copyout, acc_copyin`
- **`#pragma omp acc_loop [clause]`**
  - Purpose: specify that the iterations of one or more associated loops will be executed in an accelerated manner
  - `cache`
  - `collapse`
  - `reduction`
  - `schedule`

Example: Jacobi Iteration OpenMP directives

```
!$OMP PARALLEL DO PRIVATE(i,j,k)
     DO k = 1, Z, 1
       DO j = 1, Y, 1
         DO i = 1, X, 1
             data(i,j,k,new) = &
                 ( data(i,j,k,old) +  + &
                  data(i,j-1,k,old) + data(i,j+1,k,old) + &
                  data(i,j,k-1,old) + data2(i,j,k+1,old) - &
                  edge(i,j,k) ) / 6.0
          END DO
      END DO
    END DO
```

Example: Jacobi Iteration OpenMP Accelerator

```
!$omp acc_data copyin( edge ) copy( data )
!$omp acc_region_loop PRIVATE(i,j,k)
DO k = 1, Z, 1
   DO j = 1, Y, 1
      DO i = 1, X, 1
         data(i,j,k,new) = &
        ( data(i,j,k,old) +  + &
                   data(i,j-1,k,old) + data(i,j+1,k,old) + &
                   data(i,j,k-1,old) + data2(i,j,k+1,old) - &
                   edge(i,j,k) ) / 6.0
     END DO
   END DO
 END DO
!$omp end acc_region_loop
!$omp end acc_data
```

Test Case: Hybrid Jacobi using PGI directives

- PGI (http://www.pgroup.com) provides compiler directives for accelerators
  - Website for some documentation
- PGI active member of OpenMP Language committee
  - Use PGI Directives
- OpenMP Language committee at this time closely follows path set by PGI
- Original Hybrid MPI/OpenMP implementation provided by courtesy of EPCC (Edingburgh Parallel Computing Center) (http://www.epcc.ed.ac.uk)

Test System

- TACC's Dell XD Visualization Cluster Longhorn (http://www.tacc.utexas.edu/user-services/user-guides/longhorn-user-guide)
- 240 nodes containing 48GB of RAM,
- 8 Intel Nehalem cores (@ 2.5 GHz), and 2 NVIDIA Quadro FX 5800 GPUs per node
- Test System: Longhorn at TACC
- pgf90 11.5
- -fastsse -ta=nvidia,time -Minfo=vect,accel -Mcuda=cuda3.2

Version 0: Unoptimized

```
!$omp acc_region
DO k = 1, Z, 1
   DO j = 1, Y, 1
      DO i = 1, X, 1
         data(i,j,k,new) = &
        ( data(i,j,k,old) +  + &
         data(i,j-1,k,old) +
         data(i,j+1,k,old) + &
         data(i,j,k-1,old) +
         data2(i,j,k+1,old) - &
         edge(i,j,k) ) / 6.0
     END DO
   END DO
 END DO
!$omp end acc_region
```

```
jacobistep:
    59, Loop carried dependence of 'data' prevents parallelization
        Loop carried backward dependence of 'data' prevents vectorization
    60, Loop carried dependence of 'data' prevents parallelization
        Loop carried backward dependence of 'data' prevents vectorization
    61, Loop carried dependence of 'data' prevents parallelization
        Loop carried backward dependence of 'data' prevents vectorization
        Accelerator kernel generated
        59, !$acc do seq
        60, !$acc do seq
        61, !$acc do seq
            Non-stride-1 accesses for array 'data'
            Non-stride-1 accesses for array 'edge'
```

No performance increase when using accelerator

Version 1: Optimized for parallelization….

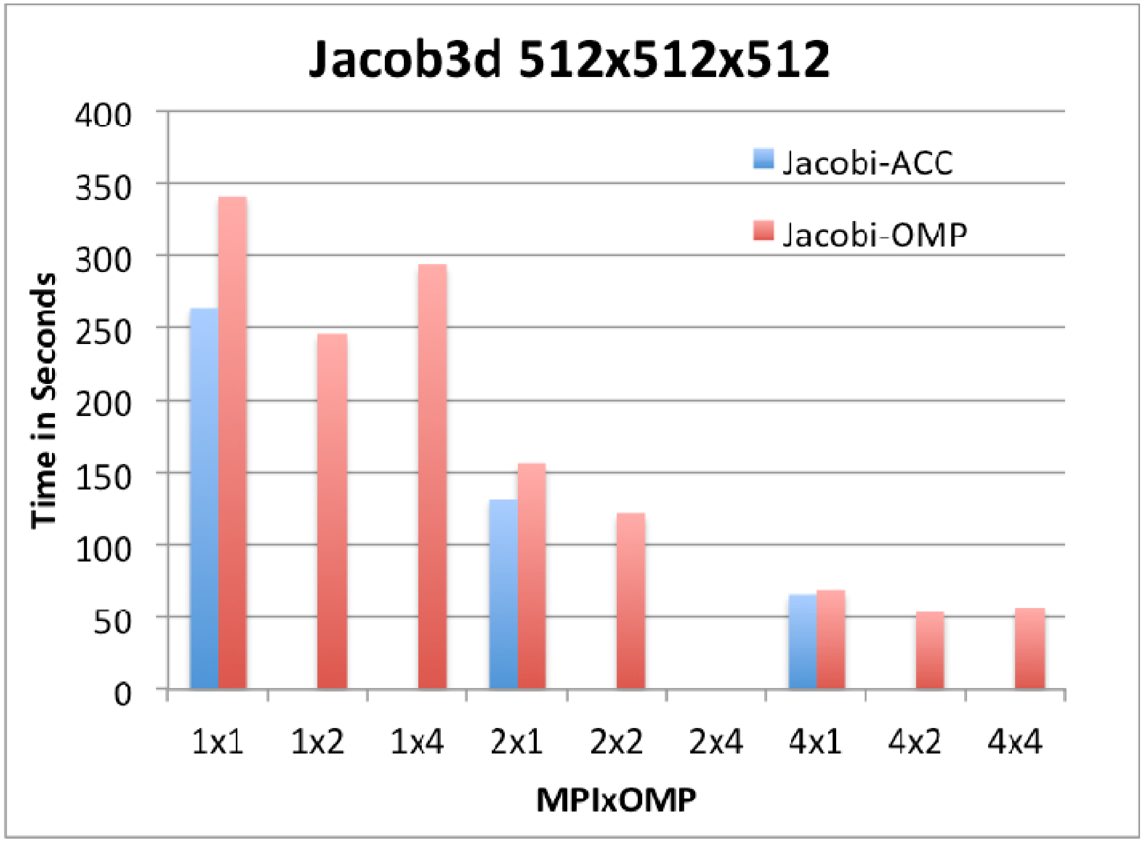
```
!$acc data region local(temp2)
updatein(data(0:X+1,0:Y+1,0:Z+1,old))
updateout(data(0:X+1,0:Y+1,0:Z+1,new)) updatein(edge)
!$acc region
 temp2 = data (:,:,:,ol
 DO k = 1, Z, 1
  DO j = 1, Y, 1
   DO i = 1, X, 1
    data(i,j,k,new) =
 &( temp2(i-1,j,k)
 + temp2(i+1,j,k) + &
 & ……
   edge(i,j,k) ) / 6.0
  END DO
 END DO
END DO
!$acc end region
!$acc end data region
```

244, Loop is parallelizable
245, Loop is parallelizable
246, Loop is parallelizable
Accelerator kernel generated
244, !$acc do parallel, vector(4) ! blockidx%y threadidx%z
245, !$acc do parallel, vector(4) ! blockidx%x threadidx%y
246, !$acc do vector(16) ! threadidx%x
Cached references to size [18x6x6] block of 'temp2'

Version 1 (cont): ….and data movement

```
module glob
    real (kind(1.0e0)), dimension(:,:,:,:), allocatable,pinned :: data
    real (kind(1.0e0)), dimension(:,:,:), allocatable,pinned :: edge
    logical first
!$acc mirror(data,edge)
 end module glob

!$acc data region local(temp2)
updatein(data(0:X+1,0:Y+1,0:Z+1,old))
updateout(data(0:X+1,0:Y+1,0:Z+1,new)) updatein(edge)
!$acc region
 temp2 = data (:,:,:,old)
 DO k = 1, Z, 1
  DO j = 1, Y, 1
   DO i = 1, X, 1
   data(i,j,k,new) = ( temp2(i-1,j,k) + temp2(i+1,j,k) + … edge (I,j,k))/6.
  END DO
 END DO
END DO
!$acc end region
!$acc end data region
```

Some Timing Results



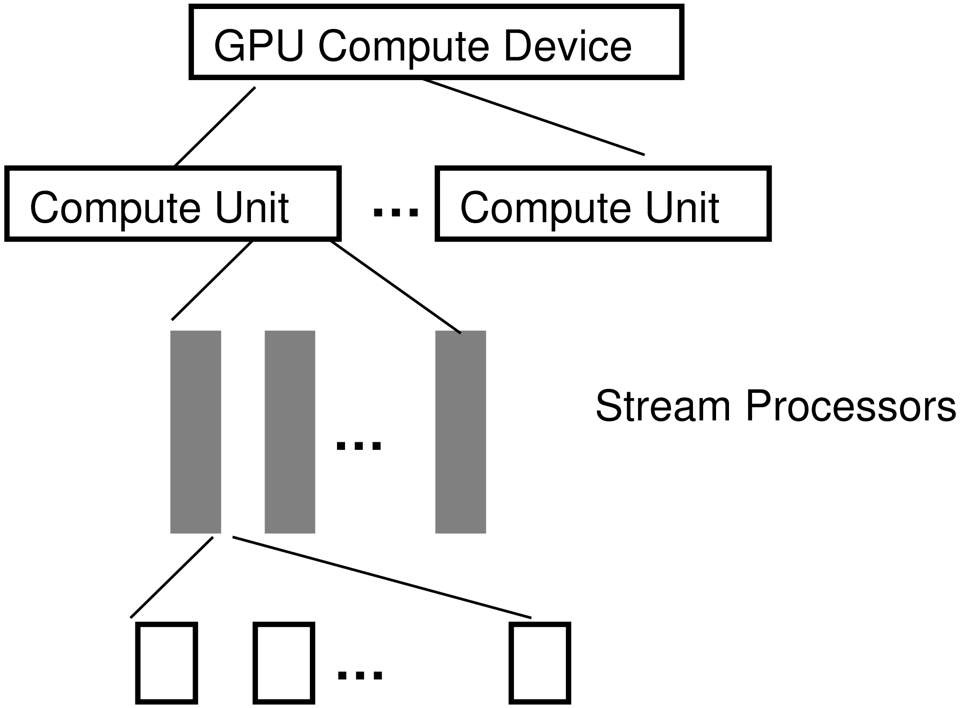
- Only one accelerator per node is being used
- Using accelerator yields improvement for single threaded execution
- CPU performance better when using multiple OpenMP threads

Concluding Remarks

- Still many other issues that need to be considered:
- Multi-core vs accelerator: General purpose vs specialized, e.g.: ▷
  - GPU runs kernels independently of the
  - GPU accelerator has large team of threads
  - GPU thread counts exceed number of cores
  - GPU uses scheduling algorithm to hide memory latency, synchronize threads into groups.
  - Stream processing
- How do we address parallelism within accelerator?
- Other types of co-processors?
- Which of the differences should be exposed via OpenMP?

- **Others?**

Structure of GPU based Compute Device



- Characteristics vary between different devices
- Memory of host CPU and GPU device might be shared

Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / pure MPI vs hybrid MPI+OpenMP
- Practical “How-To” on hybrid programming
- Mismatch Problems
- Opportunities:
  Application categories that can benefit from hybrid parallelization
- Thread-safety quality of MPI libraries
- Tools for debugging and profiling MPI+OpenMP
- Other options on clusters of SMP nodes
- **Summary**

Acknowledgements

- We want to thank
  - Gerhard Wellein, RRZE
  - Alice Koniges, NERSC, LBNL
  - Rainer Keller, HLRS and ORNL
  - Jim Cownie, Intel
  - SCALASCA/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 – hybrid MPI+OpenMP

MPI + OpenMP

- Seen with NPB-MZ examples
  - BT-MZ → strong improvement (as expected)
  - SP-MZ → small improvement
  - Usability on higher number of cores
- Advantages
  - Memory consumption
  - Load balancing
  - 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"
- **Huge amount of pitfalls**
- Optimum: Somewhere in the area of 1 MPI process per socket

Maybe the most important advantage!

skipped

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

skipped

Summary – good and bad

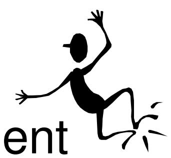
- Optimization
  - 1 MPI process per core .... mismatch problem .... 1 MPI process per SMP node

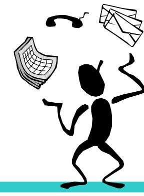
    ^– somewhere between may be the optimum

- ☺ Efficiency of MPI+OpenMP is not for free:

  The efficiency strongly depends on

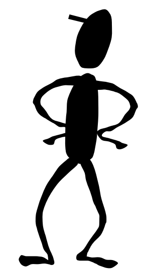
  ☹ the amount of work in the source code development



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 <u>appropriate</u> application a suitable tool

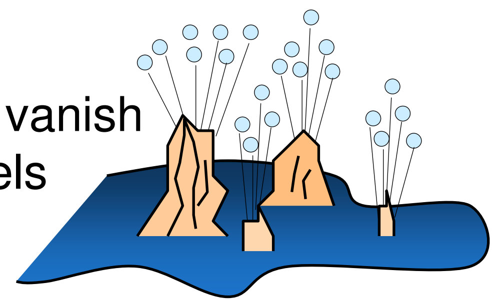
Summary

- 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

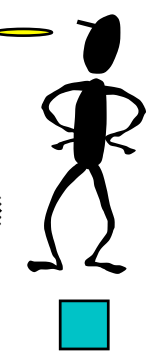
  → Area of interesting research in next years



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

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

Abstract

Half-Day Tutorial (Level: 20% Introductory, 50% Intermediate, 30% Advanced)

Authors. Rolf Rabenseifner, HLRS, University of Stuttgart, Germany
Georg Hager, University of Erlangen-Nuremberg, Germany
Gabriele Jost, Texas Advanced Computing Center, The University of Texas at Austin, USA

Abstract. Most HPC systems are clusters of shared memory nodes. Such systems can be PC clusters with single/multi-socket and multi-core SMP nodes, but also "constellation" type systems with large SMP nodes. Parallel programming may combine the distributed memory parallelization on the node interconnect with the shared memory parallelization inside of each node.

This tutorial analyzes the strengths and weaknesses of several parallel programming models on clusters of SMP nodes. Multi-socket-multi-core systems in highly parallel environments are given special consideration. This includes a discussion on planned future OpenMP support for accelerators. Various hybrid MPI+OpenMP approaches are compared with pure MPI, and benchmark results on different platforms are presented. Numerous case studies demonstrate the performance-related aspects of hybrid MPI/OpenMP programming, and application categories that can take advantage of hybrid programming are identified. Tools for hybrid programming such as thread/process placement support and performance analysis are presented in a "how-to" section.

Details. https://fs.hlrs.de/projects/rabenseifner/publ/SC2011-hybrid.html

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.

Georg Hager

Georg Hager holds a PhD in computational physics from the University of Greifswald. He has been working with high performance systems since 1995, and is now a senior research scientist in the HPC group at Erlangen Regional Computing Center (RRZE). His daily work encompasses all aspects of HPC user support and training, assessment of novel system and processor architectures, and supervision of student projects and theses. Recent research includes architecture-specific optimization for current microprocessors, performance modeling on processor and system levels, and the efficient use of hybrid parallel systems. A full list of publications, talks, and other HPC-related stuff he is interested in can be found in his blog: http://blogs.fau.de/hager.

Gabriele Jost

Gabriele Jost obtained her doctorate in Applied Mathematics from the University of Göttingen, Germany. For more than a decade she worked for various vendors (Suprenum GmbH, Thinking Machines Corporation, and NEC) of high performance parallel computers in the areas of vectorization, parallelization, performance analysis and optimization of scientific and engineering applications.

In 2005 she moved from California to the Pacific Northwest and joined Sun Microsystems as a staff engineer in the Compiler Performance Engineering team, analyzing compiler generated code and providing feedback and suggestions for improvement to the compiler group. She then decided to explore the world beyond scientific computing and joined Oracle as a Principal Engineer working on performance analysis for application server software. That was fun, but she realized that her real passions remains in area of performance analysis and evaluation of programming paradigms for high performance computing and that she really liked California. She is now a Research Scientist at the Texas Advanced Computing Center (TACC), working remotely from Monterey, CA on all sorts of exciting projects related to large scale parallel processing for scientific computing.

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, 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.

References

- J. Treibig, G. Hager and G. Wellein:
  LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments.
  Proc. of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures, San Diego CA, September 13, 2010.
  Preprint: http://arxiv.org/abs/1004.4431
- H. Stengel:
  Parallel programming on hybrid hardware: Models and applications.
  Master's thesis, Ohm University of Applied Sciences/RRZE, Nuremberg, 2010.
  http://www.hpc.rrze.uni-erlangen.de/Projekte/hybrid.shtml

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

Content

Content