Hybrid Parallel Programming with MPI & OpenMP

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@supersmith.com

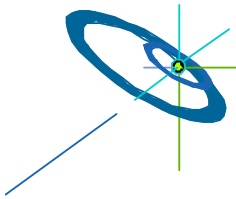
[1)] High Performance Computing Center (HLRS), University of Stuttgart, Germany

[2)] Regional Computing Center (RRZE), University of Erlangen, Germany

[3)] Supersmith, Maximum Performance Software, USA

Tutorial 09 at ISC13,
June 16, 2013, Leipzig, Germany

INTERNATIONAL SUPERCOMPUTING CONFERENCE

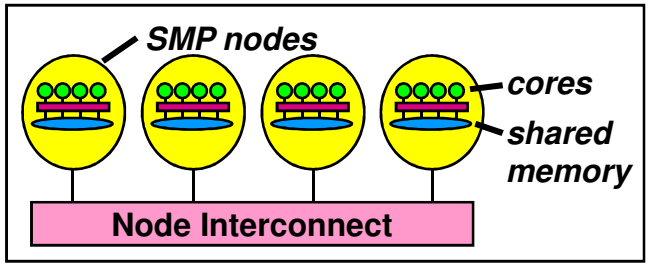


H L R S

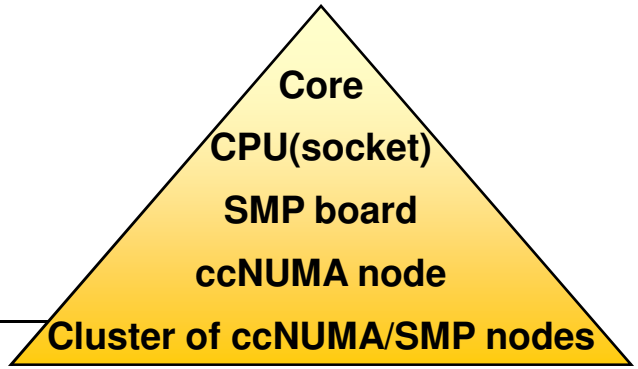
SUPERsmith

Motivation

- Efficient programming of clusters of shared memory (SMP) nodes

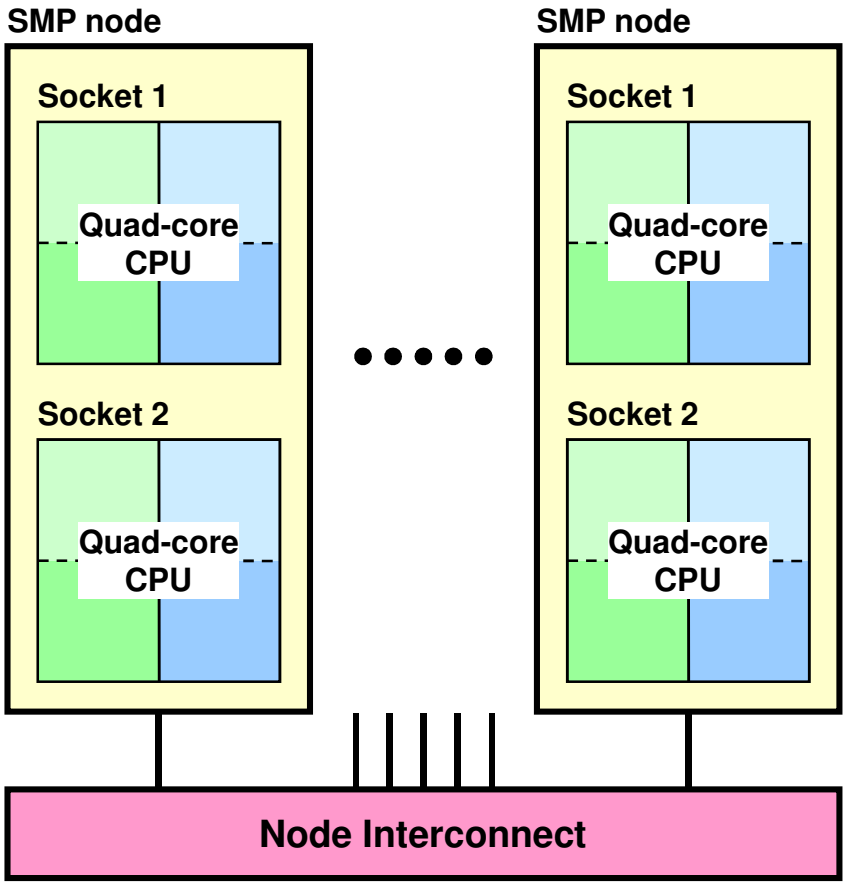


- Hierarchical system layout

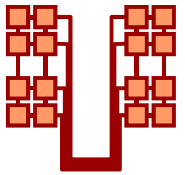


- Hybrid programming seems natural
  - **MPI between the nodes**
  - **Shared memory programming inside of each SMP node**
    - OpenMP
    - new – MPI-3 shared memory programming
    - Accelerator support in new OpenMP 4.0 and OpenACC

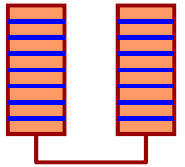
Motivation



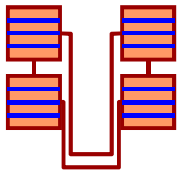
- Which programming model is fastest?
- MPI everywhere?



- 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

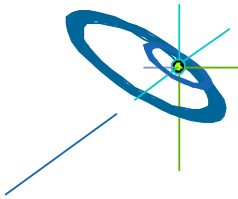
Outline

Includes additional slides, marked as — skipped —

SUPERsmith

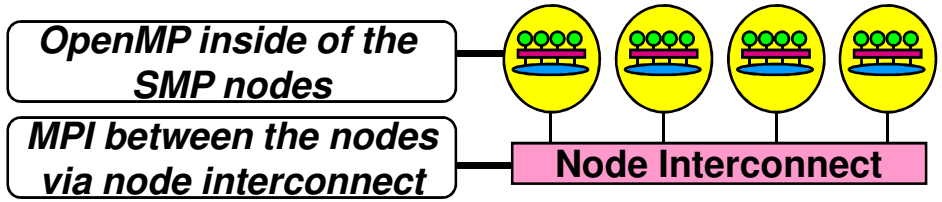
Outline

- Introduction / Motivation
- **Programming models on clusters of SMP nodes**
- Case Studies / pure MPI vs hybrid MPI+OpenMP
- Hybrid programming & accelerators
- 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



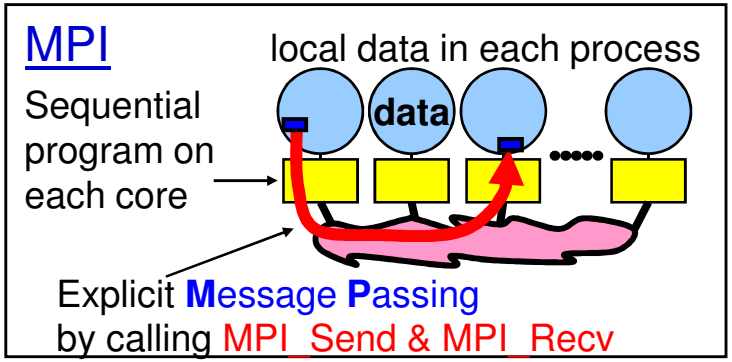
Major Programming models on hybrid systems

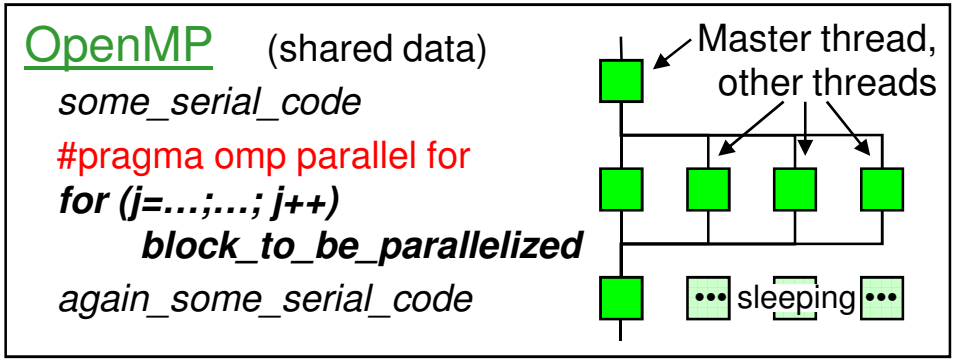
- Pure MPI (one MPI process on each core)
- Hybrid: **MPI + OpenMP**
  - shared memory OpenMP
  - distributed memory MPI



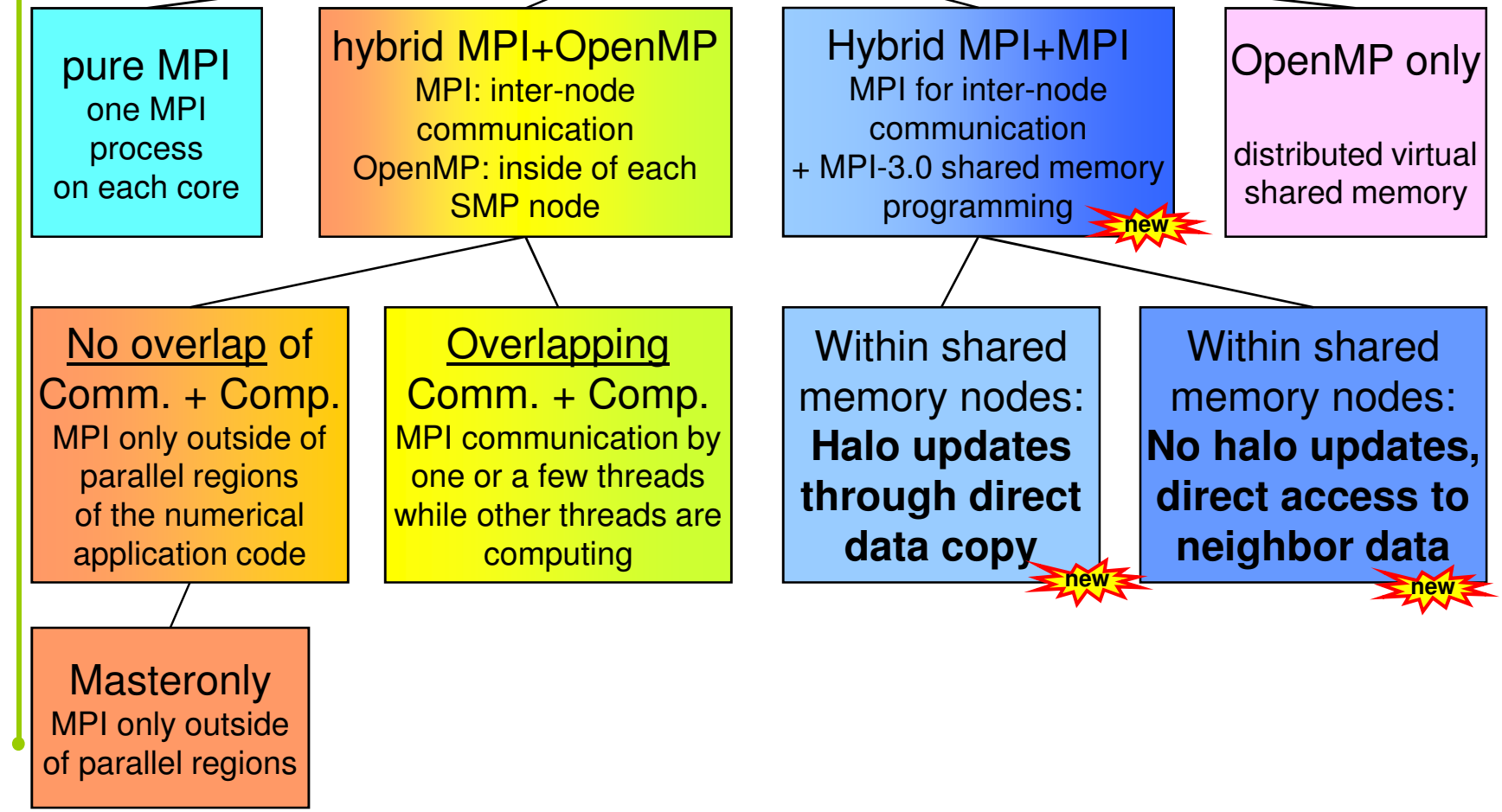
new • Hybrid: MPI message passing + **MPI-3.0 shared memory programming**

- Other: PGAS (UPC, Coarray Fortran, ….) / together with MPI
- Often **hybrid programming (MPI+OpenMP)** slower than **pure MPI**
  - why?





Parallel Programming Models on Hybrid Platforms



Pure MPI

pure MPI
one MPI process
on each core

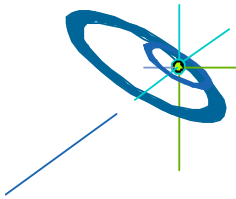
Advantages

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

Major problems

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

Discussed
in detail later on
in the section
Mismatch Problems



Hybrid MPI+OpenMP Masteronly Style

Masteronly
MPI only outside of parallel regions

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

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

Advantages

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

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)
```

Hybrid MPI + MPI-3 shared memory

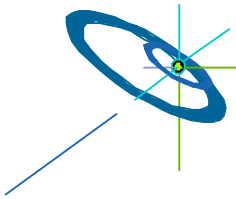
Hybrid MPI+MPI
MPI for inter-node communication
+ MPI-3.0 shared memory programming

Advantages

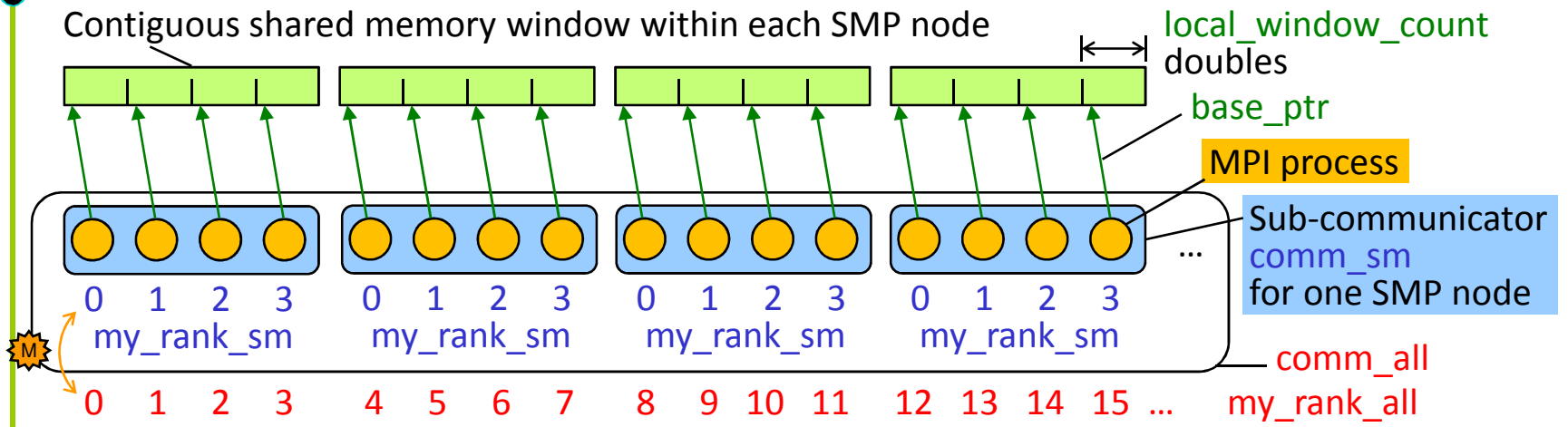
- No message passing inside of the SMP nodes
- Using only one parallel programming standard
- No OpenMP problems (e.g., thread-safety isn't an issue)

Major Problems

- Communicator must be split into shared memory islands
- To minimize shared memory communication overhead:
  Halos (or the data accessed by the neighbors) must be stored in
  MPI shared memory windows
- Same work-sharing as with pure MPI



Splitting the communicator & contiguous shared memory allocation




```
MPI_Aint /*IN*/ local_window_count;   double /*OUT*/  *base_ptr;
MPI_Comm  comm_all,  comm_sm;         int  my_rank_all,  my_rank_sm,  size_sm,  disp_unit;
MPI_Comm_rank (comm_all, &my_rank_all);
MPI_Comm_split_type (comm_all, MPI_COMM_TYPE_SHARED, 0,
                     MPI_INFO_NULL,  &comm_sm);
MPI_Comm_rank (comm_sm, &my_rank_sm);  MPI_Comm_size (comm_sm, &size_sm);
disp_unit = sizeof(double);  /* shared memory should contain doubles */
MPI_Win_allocate_shared (local_window_count*disp_unit,  disp_unit,  MPI_INFO_NULL,
                         comm_sm,  &base_ptr,  &win_sm);
```

Sequence in comm_sm as in comm_all

F

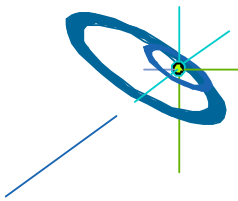
F In Fortran, MPI-3.0, page 341, Examples 8.1 (and 8.2) show how to convert buf_ptr into a usable array a.

M This mapping is based on a sequential ranking of the SMP nodes in comm_all.

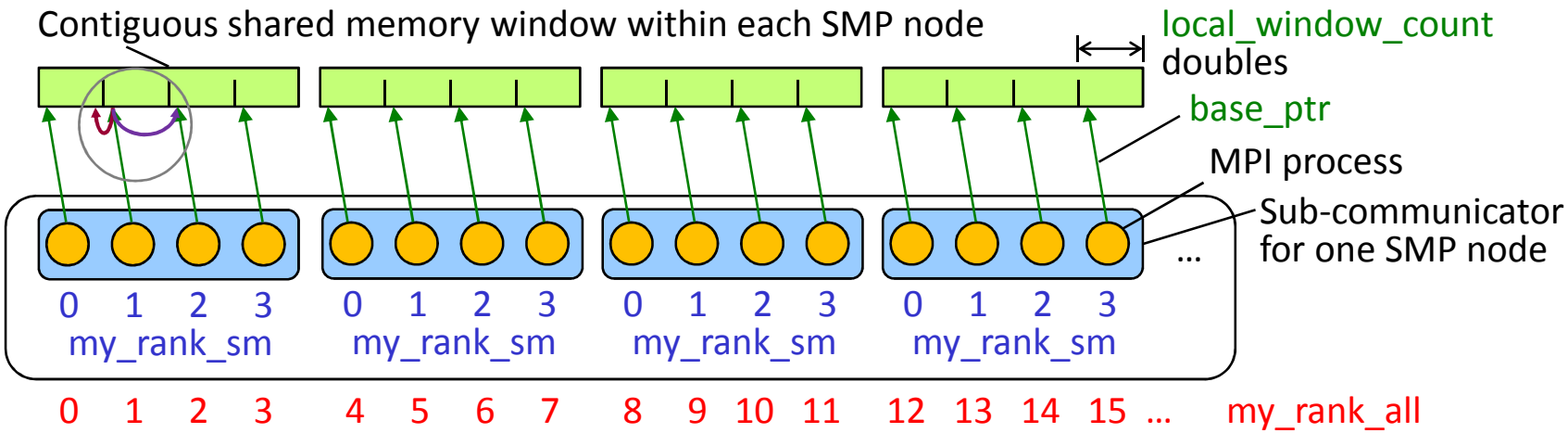
Within each SMP node – Essentials

- The allocated shared memory is contiguous across process ranks,
- i.e., the first byte of rank i starts right after the last byte of rank i-1.
- Processes can calculate remote addresses' offsets with local information only.
- Remote accesses through load/store operations,
- i.e., without MPI RMA operations (MPI_GET/PUT, …)
- Although each process in comm_sm accesses the same physical memory, the virtual start address of the whole array may be different in all processes!
  → **linked lists** only with offsets in a shared array, but **not with binary pointer addresses!**

- Following slides show only the shared memory accesses, i.e., communication between the SMP nodes is not presented.



Shared memory access example




```
MPI_Aint /*IN*/ local_window_count;      double /*OUT*/  *base_ptr;
MPI_Win_allocate_shared (local_window_count*disp_unit, disp_unit, MPI_INFO_NULL,
                         comm_sm, &base_ptr, &win_sm);
```

Synchroni-zation

Synchroni-zation

```
MPI_Win_fence (0, win_sm);  /*local store epoch can start*/
for (i=0; i<local_window_count; i++)  base_ptr[i] = ... /* fill values into local portion */
MPI_Win_fence (0, win_sm);  /* local stores are finished, remote load epoch can start */
if (my_rank_sm > 0)          printf("left neighbor's rightmost value = %lf \n", base_ptr[-1] );
if (my_rank_sm < size_sm-1)  printf("right neighbor's leftmost value = %lf \n",
                                    base_ptr[local_window_count] );
```

Local stores

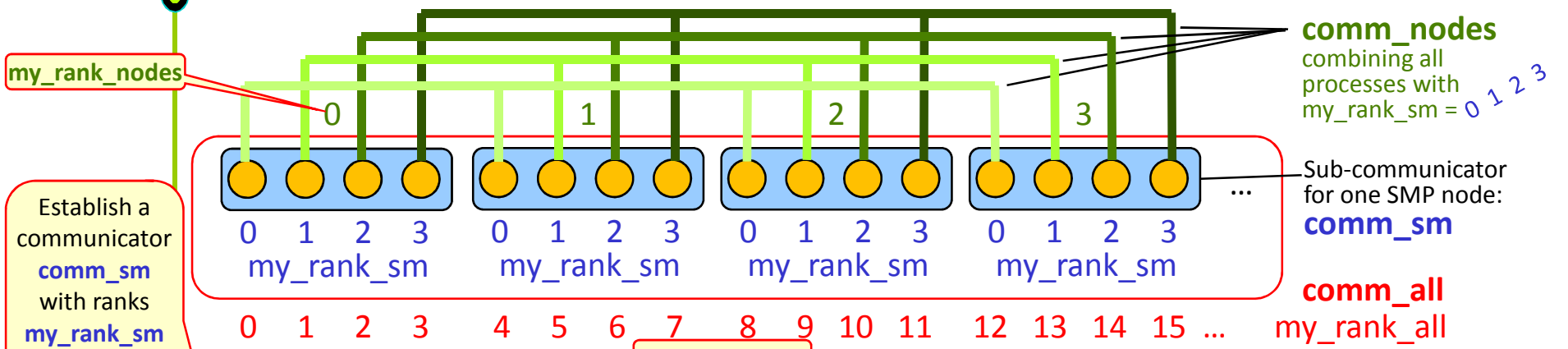
Direct load access to remote window portion

F In Fortran, before and after the synchronization, on must add: CALL MPI_F_SYNC_REG (buffer)
to guarantee that register copies of buffer are written back to memory, respectively read again from memory.

skipped

Establish comm_sm, comm_nodes, comm_all, if SMPs are not contiguous within comm_orig

Hybrid MPI+MPI
MPI for inter-node communication
+ MPI-3.0 shared memory programming



Establish a communicator **comm_sm** with ranks **my_rank_sm** on each SMP node

Input

```
MPI_Comm_split_type (comm_orig, MPI_COMM_TYPE_SHARED, 0, MPI_INFO_NULL, &comm_sm);
MPI_Comm_size (comm_sm, &size_sm); MPI_Comm_rank (comm_sm, &my_rank_sm);
MPI_Comm_split (comm_orig, my_rank_sm, 0, &comm_nodes);
MPI_Comm_size (comm_nodes, &size_nodes);
if (my_rank_sm==0) {
  MPI_Comm_rank (comm_nodes, &my_rank_nodes);
  MPI_Exscan (&size_sm, &my_rank_all, 1, MPI_INT, MPI_SUM, comm_nodes);
  if (my_rank_nodes == 0)  my_rank_all = 0;
}
MPI_Comm_free (&comm_nodes);
MPI_Bcast (&my_rank_nodes, 1, MPI_INT, 0, comm_sm);
MPI_Comm_split (comm_orig, my_rank_sm, my_rank_nodes, &comm_nodes);
MPI_Bcast (&my_rank_all, 1, MPI_INT, 0, comm_sm); my_rank_all = my_rank_all + my_rank_sm;
MPI_Comm_split (comm_orig, /*color*/ 0, my_rank_all, &comm_all);
```

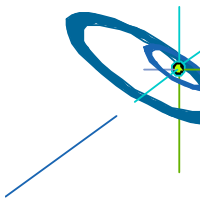
Result: comm_nodes combines all processes with a given my_rank_sm into a separate communicator.

On processes with my_rank_sm > 0, this comm_nodes is unused because node-numbering within these comm_nodes may be different.

Exscan does not return value on the first rank, therefore

Expanding the numbering from **comm_nodes** with my_rank_sm == 0 to all new node-to-node communicators **comm_nodes**.

Calculating **my_rank_all** and establishing global communicator **comm_all** with sequential SMP subsets.



Alternative: Non-contiguous shared memory

- Using info key "alloc_shared_noncontig“
- MPI library can put processes’ window portions
  - on page boundaries,
    - **(internally, e.g., only one OS shared memory segment with some unused padding zones)**
  - into the local ccNUMA memory domain + page boundaries
    - **(internally, e.g., each window portion is one OS shared memory segment)**

Pros:

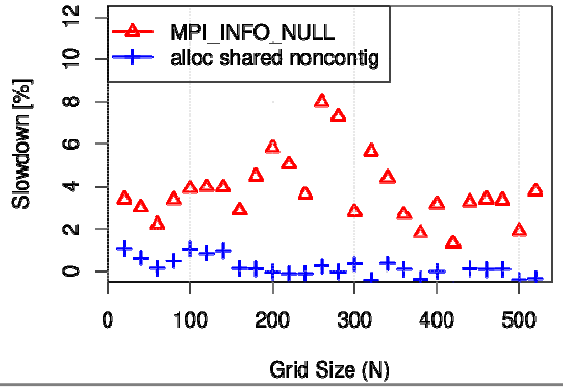
- Faster local data accesses especially on ccNUMA nodes

Cons:

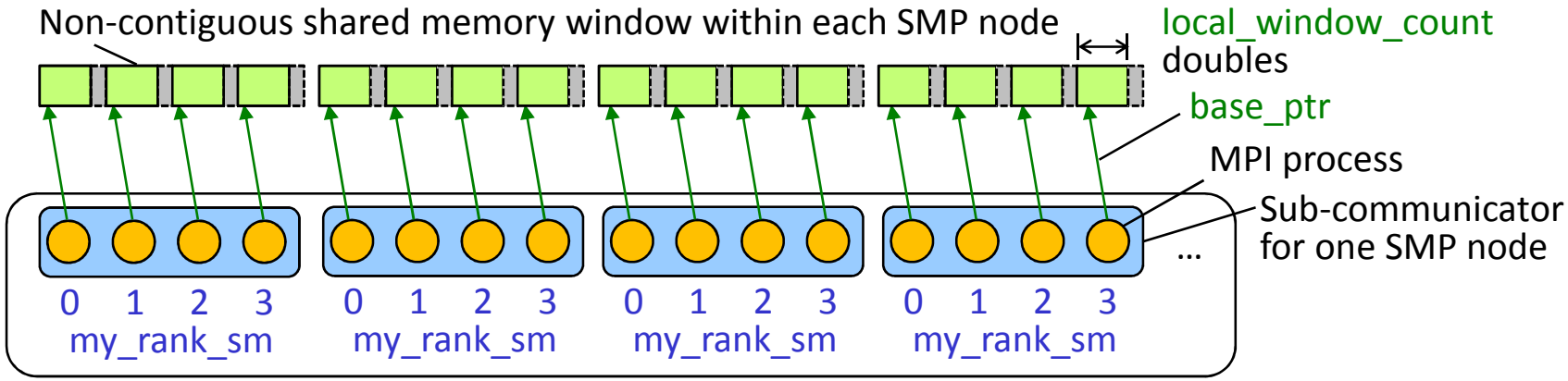
- Higher programming effort for neighbor accesses: MPI_WIN_SHARED_QUERY

Further reading:
Torsten Hoefler, James Dinan, Darius Buntinas, Pavan Balaji, Brian Barrett, Ron Brightwell, William Gropp, Vivek Kale, Rajeev Thakur:
MPI + MPI: a new hybrid approach to parallel programming with MPI plus shared memory.
http://link.springer.com/content/pdf/10.1007%2Fs00607-013-0324-2.pdf

NUMA effects?
Significant impact of alloc_shared_noncontig



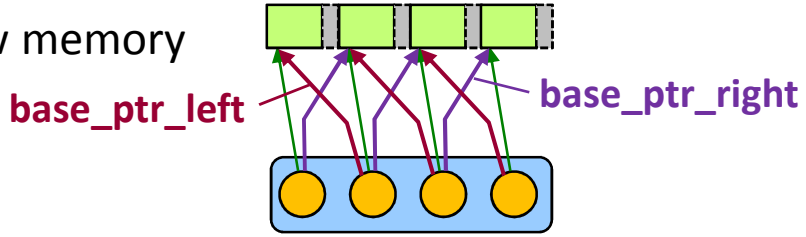
Non-contiguous shared memory allocation




```
MPI_Aint /*IN*/ local_window_count;        double /*OUT*/  *base_ptr;
disp_unit = sizeof(double);  /* shared memory should contain doubles */
MPI_Info  info_noncontig;
MPI_Info_create (&info_noncontig);
MPI_Info_set (info_noncontig, "alloc_shared_noncontig", "true");
MPI_Win_allocate_shared (local_window_count*disp_unit,  disp_unit,  info_noncontig,
                         comm_sm,  &base_ptr,  &win_sm );
```

Non-contiguous shared memory: Neighbor access through MPI_WIN_SHARED_QUERY

- Each process can retrieve each neighbor's base_ptr with calls to MPI_WIN_SHARED_QUERY
- Example: only pointers to the window memory of the left & right neighbor




```
if (my_rank_sm > 0)          MPI_Win_shared_query (win_sm, my_rank_sm – 1,
                                  &win_size_left,  &disp_unit_left,  &base_ptr_left);
if (my_rank_sm < size_sm-1)  MPI_Win_shared_query (win_sm, my_rank_sm + 1,
                                  &win_size_right, &disp_unit_right, &base_ptr_right);
…
MPI_Win_fence (0, win_sm);  /* local stores are finished, remote load epoch can start */
if (my_rank_sm > 0)          printf("left neighbor's rightmost value = %lf \n",
                                  base_ptr_left[ win_size_left/disp_unit_left – 1 ] );
if (my_rank_sm < size_sm-1)  printf("right neighbor's leftmost value = %lf \n",
                                  base_ptr_right[ 0 ] );
```


Thanks to Steffen Weise (TU Freiberg) for testing and correcting the example codes.

Other technical aspects with MPI_WIN_ALLOCATE_SHARED

Caution: On some systems
- the number of shared memory windows, and
- the total size of shared memory windows

may be limited.

Some OS systems may provide options, e.g.,
- at job launch, or
- MPI process start,

to enlarge restricting defaults.

Another restriction in a low-quality MPI: MPI_COMM_SPLIT_TYPE may return always MPI_COMM_SELF

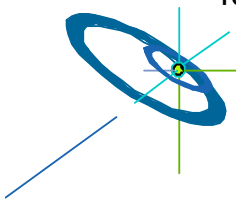
If MPI shared memory support is based on POSIX shared memory:
- Shared memory windows are located in memory-mapped /dev/shm
- Default: 25% or 50% of the physical memory, but a maximum of ~2043 windows!
- Root may change size with: mount –o remount,size=6G /dev/shm .

Due to default limit of context IDs in mpich

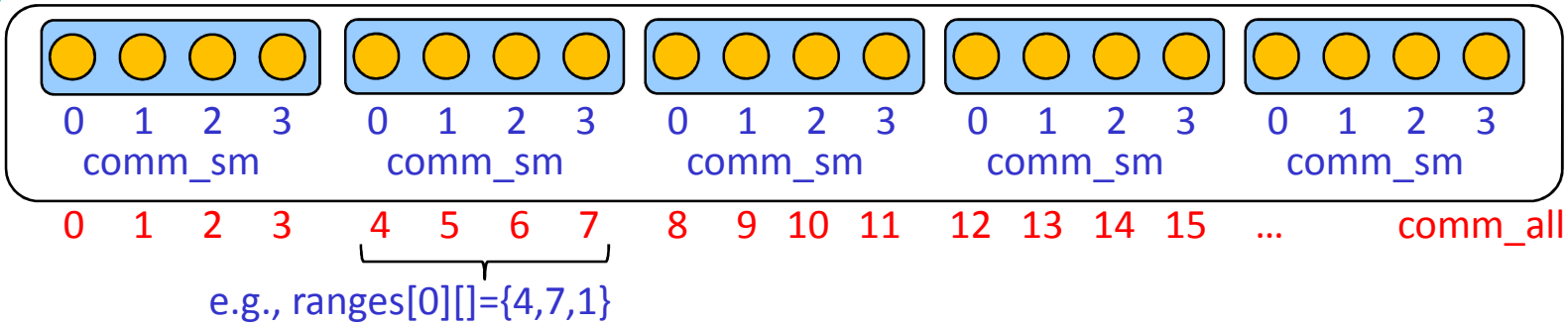
Cray XT/XE/XC (XPMEM): No limits.

On a system without virtual memory (like CNK on BG/Q), you have to reserve a chunk of address space when the node is booted (default is 64 MB).

Thanks to Jeff Hammond and Jed Brown (ANL), Brian W Barrett (SANDIA), and Steffen Weise (TU Freiberg), for input and discussion.



Splitting the communicator without MPI_COMM_SPLIT_TYPE



Alternative, if you want to group based on a fixed amount size_sm of shared memory cores in comm_all:

- Based on sequential ranks in comm_all
- Pro: comm_sm can be restricted to ccNUMA locality domains
- Con: MPI does not guarantee MPI_WIN_ALLOCATE_SHARED() on whole SMP node (MPI_COMM_SPLIT_TYPE() may return MPI_COMM_SELF or partial SMP node)

```
MPI_Comm_rank (comm_all, &my_rank);   MPI_Comm_group (comm_all, &group_all);
ranges[0][0] = (my_rank / size_sm) * size_sm;   ranges[0][1] = ranges[0][0]+size_sm–1;   ranges[0][2] = 1;
MPI_Group_range_incl (group_all, 1, ranges, &group_sm);
MPI_Comm_create (comm_all, group_sm,  &comm_sm);
MPI_Win_allocate_shared (…);
```

To guarantee shared memory, one may add an additional **MPI_Comm_split_type** (comm_sm, MPI_COMM_TYPE_SHARED, 0, MPI_INFO_NULL, &comm_sm_really);

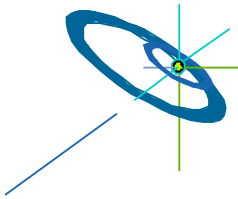
Pure OpenMP (on the cluster)

OpenMP only
distributed virtual shared memory

- Distributed shared virtual memory system needed
- Must support clusters of SMP nodes, e.g.,
  - 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!

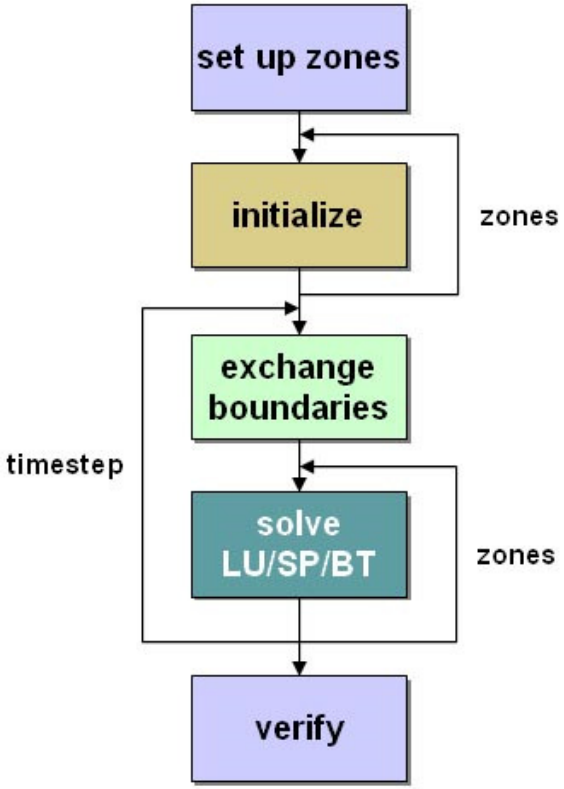


Outline

- 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 (Supersmith, Maximum Performance Software)
- Hybrid programming & accelerators
- 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

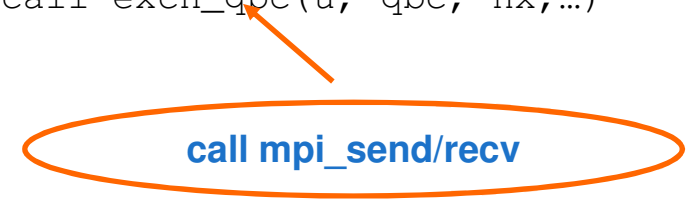


	MPI/OpenMP	MLP	Nested OpenMP
Time step	sequential	sequential	sequential
inter-zones	MPI Processes	MLP Processes	OpenMP
exchange boundaries	Call MPI	data copy+ sync.	OpenMP
intra-zones	OpenMP	OpenMP	OpenMP

- 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,…)
```

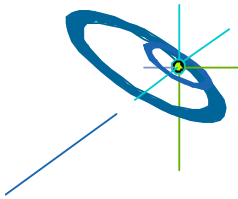



```
do zone = 1, num_zones
    if (iam .eq. pzone_id(zone)) then
        call zsolve(u,rsd,…)
      end if
    end do

end do
  ...
```

```
 subroutine zsolve(u, rsd,…)
  ...
!$OMP PARALLEL DEFAULT(SHARED)
!$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!
- "**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

Hybrid code on cc-NUMA architectures

- **OpenMP**:
  - Support only per MPI process
  - Version 3.1 has support for binding of threads via OMP_PROC_BIND environment variable.
  - Under consideration for Version 4.0: OMP_PROC_SET or OMP_LIST to restrict the execution to a subset of the machine; OMP_AFFINITY to influence how the threads are distributed and bound on the machine
  - **Version 4.0 announced at SC12**
- **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

NUMA Control (numactl) – 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

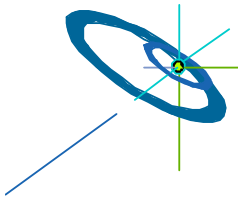
Command to run mpi job

NUMA Operations

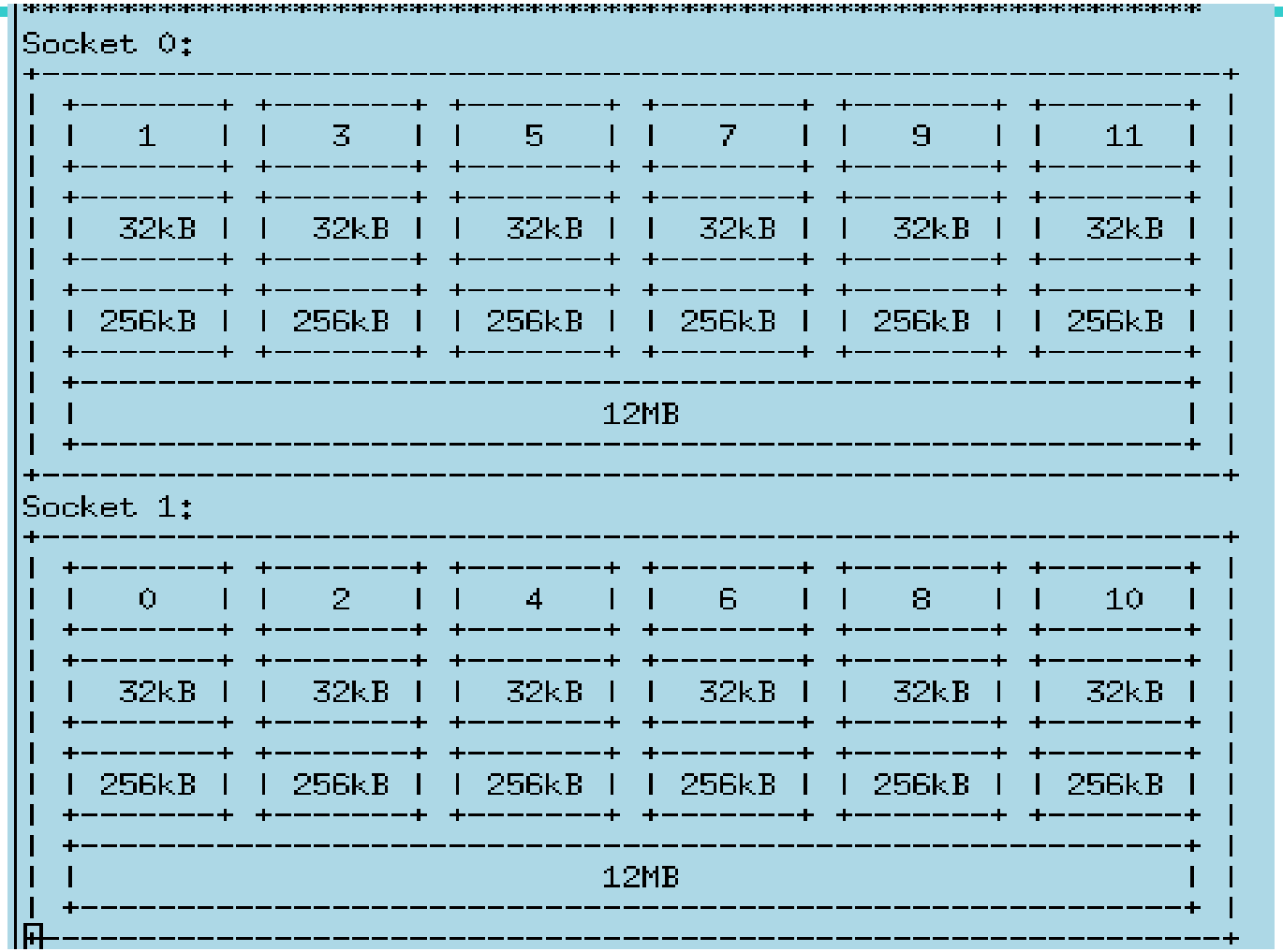
	cmd	option	arguments	description
Socket Affinity	numactl	-c	{0,1,2,3}	Only execute process on cores of this (these) socket(s).
Memory Policy	numactl	-l	{no argument}	Allocate on current socket.
Memory Policy	numactl	-i	{0,1,2,3}	Allocate round robin (interleave) on these sockets.
Memory Policy	numactl	--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).

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 $*
```



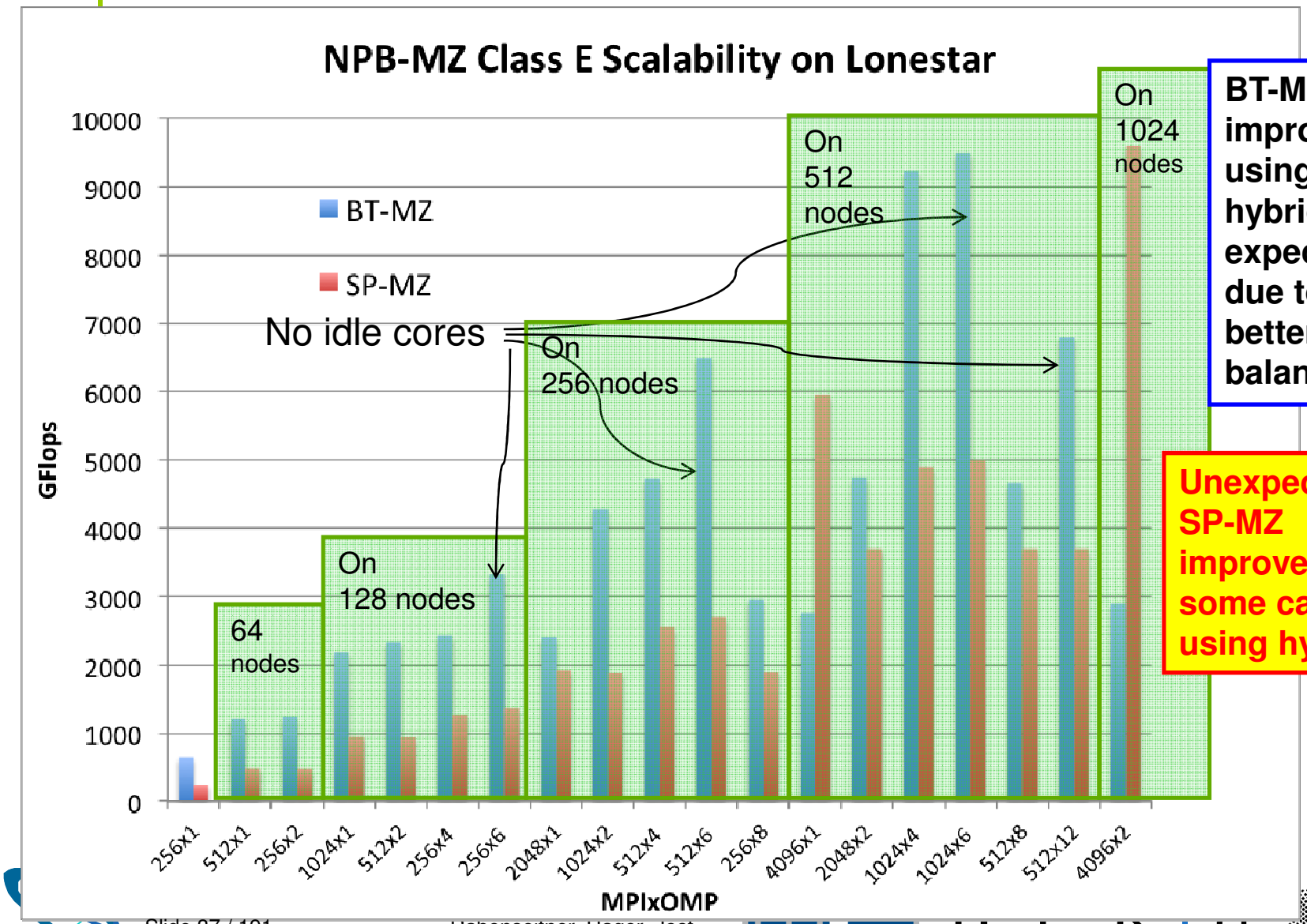
Dell Linux Cluster Lonestar Topology



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



BT-MZ improves using hybrid as expected due to better load balance

Unexpected: SP-MZ improves in some cases using hybrid

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 $*
```

Use cores and memory on 2 sockets

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

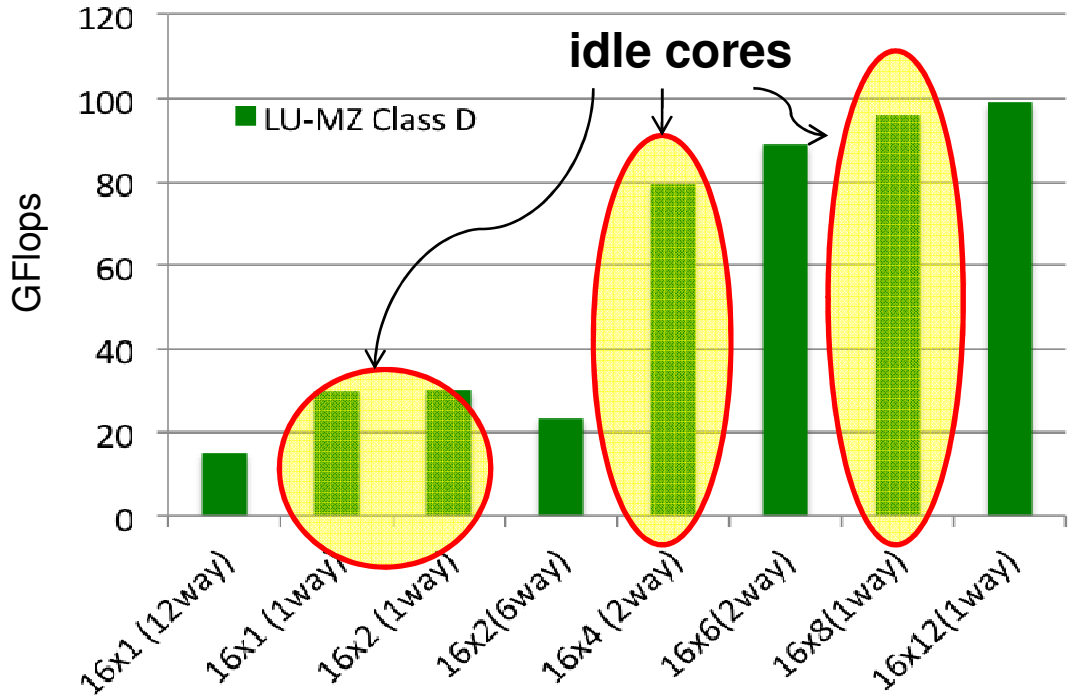
Pinning B:

```
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

LU-MZ Class D Scalability on Lonestar



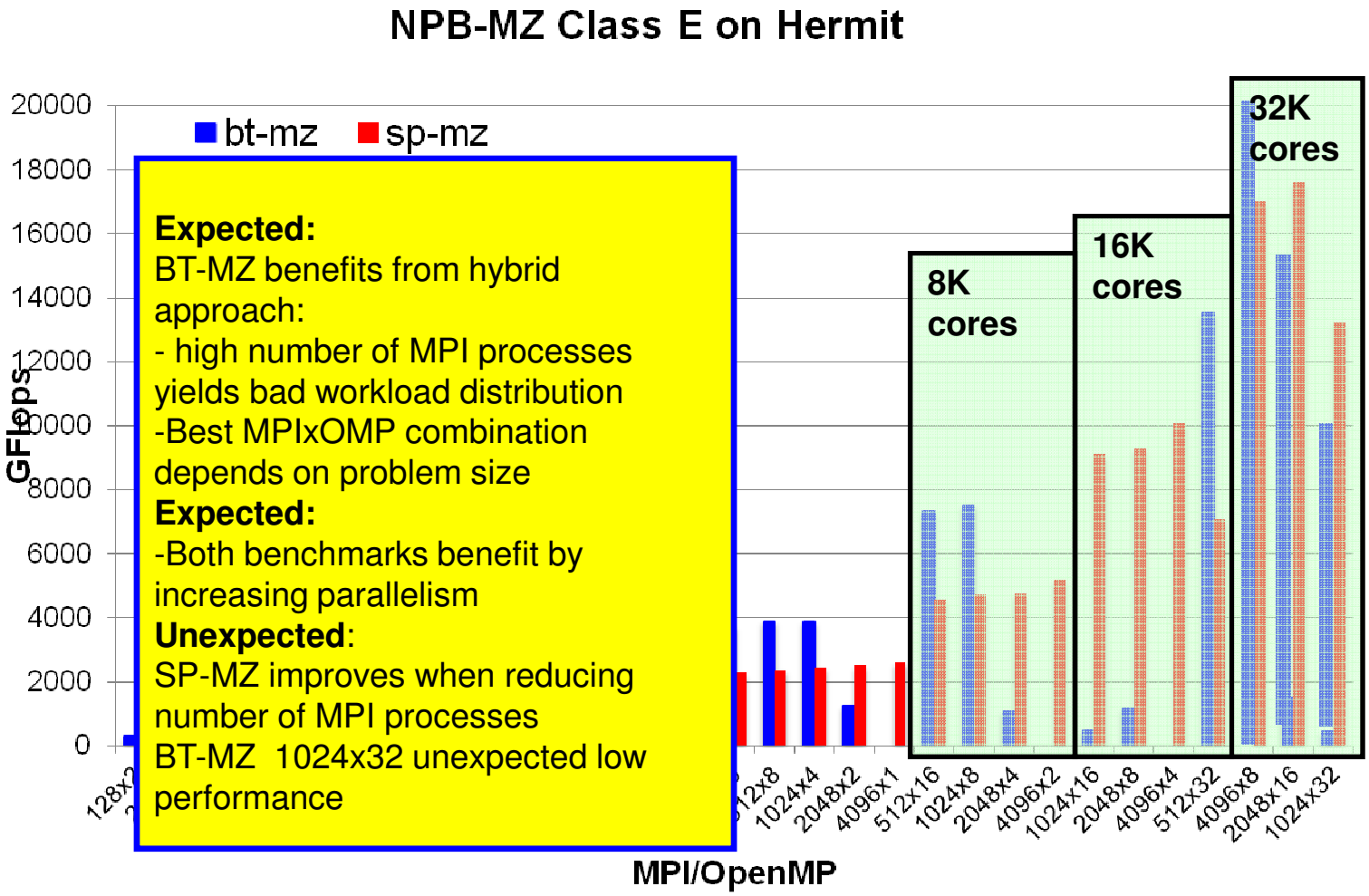
- LU-MZ significantly benefits from hybrid mode:
  - Pure MPI limited to 16 cores, due to #zones = 16
- Decrease of resource contention large contribution to improvement

Cray XE6 Hermit

- Located at HLRS Stuttgart, Germany (https://wickie.hlrs.de/platforms/index.php/Cray_XE6)
- 3552 compute nodes 113.664 cores
- Two AMD 6276 Interlagos processors with 16 cores each, running at 2.3 GHz (TurboCore 3.3GHz) per node
- Around 1 Pflop theoretical peak performance
- 32 GB of main memory available per node
- 32-way shared memory system
- High-bandwidth interconnect using Cray Gemini communication chips

```
--------------------------------------------------------------
CPU type:           AMD Interlagos processor
**************************************************************
Hardware Thread Topology
**************************************************************
Sockets:                2
Cores per socket:       16
Threads per core:       1
--------------------------------------------------------------
Socket 0:
+-----------------------------------------------------------------------------------------------------------------------------------------------+
| +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ |
| |  0   | |  1   | |  2   | |  3   | |  4   | |  5   | |  6   | |  7   | |  8   | |  9   | |  10  | |  11  | |  12  | |  13  | |  14  | |  15  | |
| +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ |
| +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ |
| | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | |
| +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ |
| +----------------+ +----------------+ +----------------+ +----------------+ +----------------+ +----------------+ +----------------+ +----------------+ |
| |      2MB       | |      2MB       | |      2MB       | |      2MB       | |      2MB       | |      2MB       | |      2MB       | |      2MB       | |
| +----------------+ +----------------+ +----------------+ +----------------+ +----------------+ +----------------+ +----------------+ +----------------+ |
| +-------------------------------------------------------------------------+ +-------------------------------------------------------------------------+ |
| |                                  6MB                                    | |                                  6MB                                    | |
| +-------------------------------------------------------------------------+ +-------------------------------------------------------------------------+ |
```

Cray XE6 Hermit Scalability, continued



Expected:
BT-MZ benefits from hybrid approach:
- high number of MPI processes yields bad workload distribution
-Best MPIxOMP combination depends on problem size

Expected:
-Both benchmarks benefit by increasing parallelism

Unexpected:
SP-MZ improves when reducing number of MPI processes
BT-MZ 1024x32 unexpected low performance

H L R S

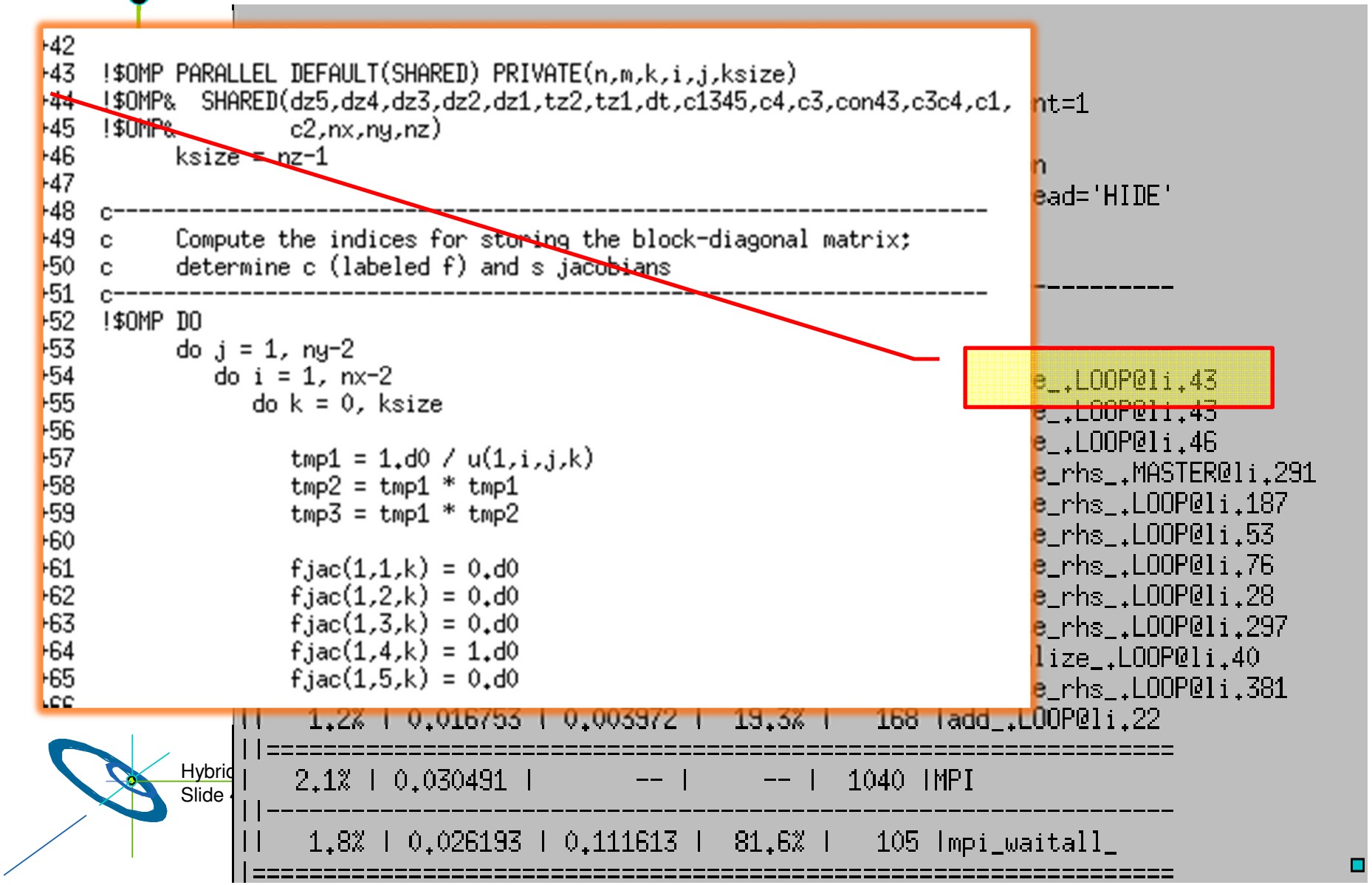
SUPERsmith

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



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

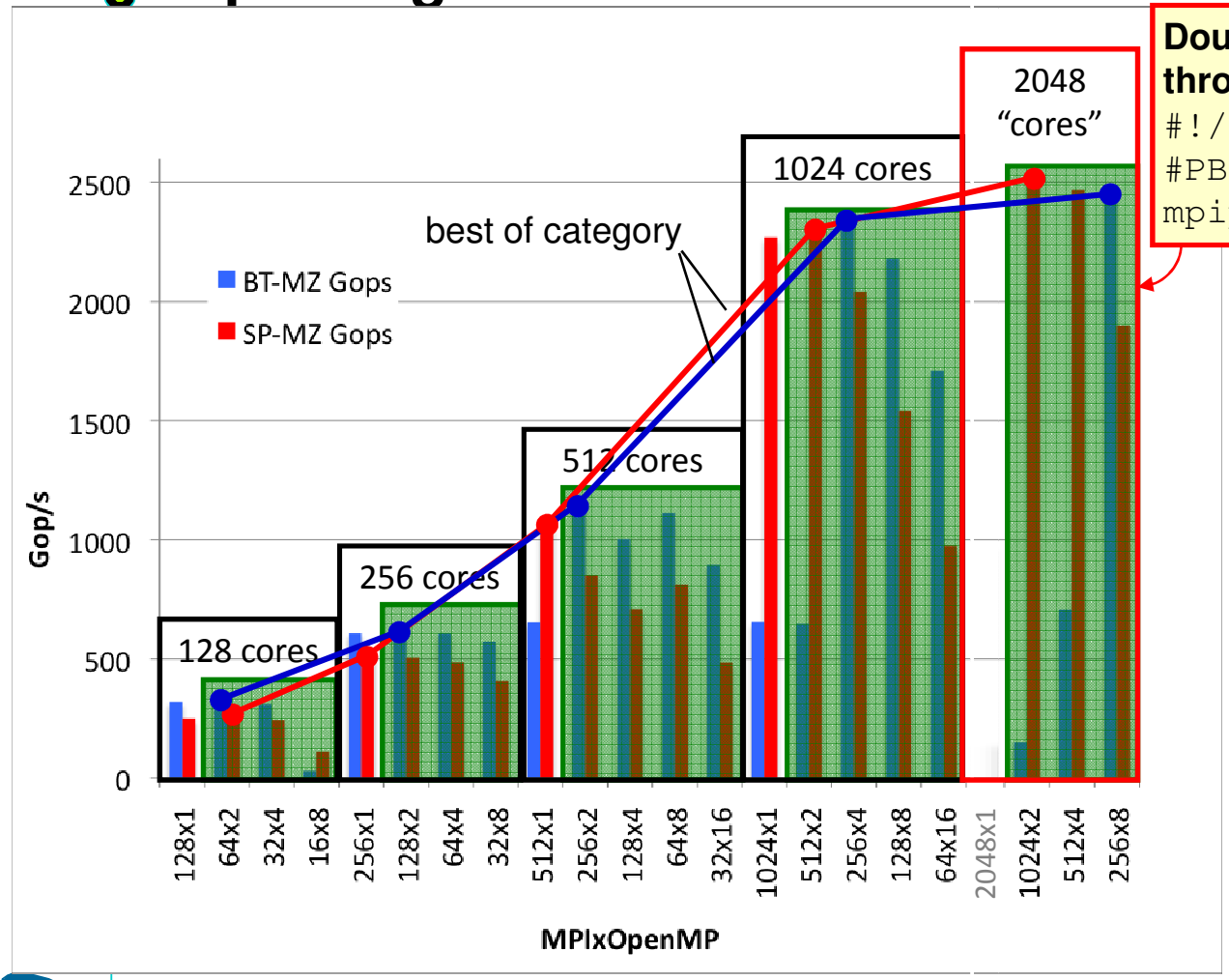
IBM Power 6

- 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

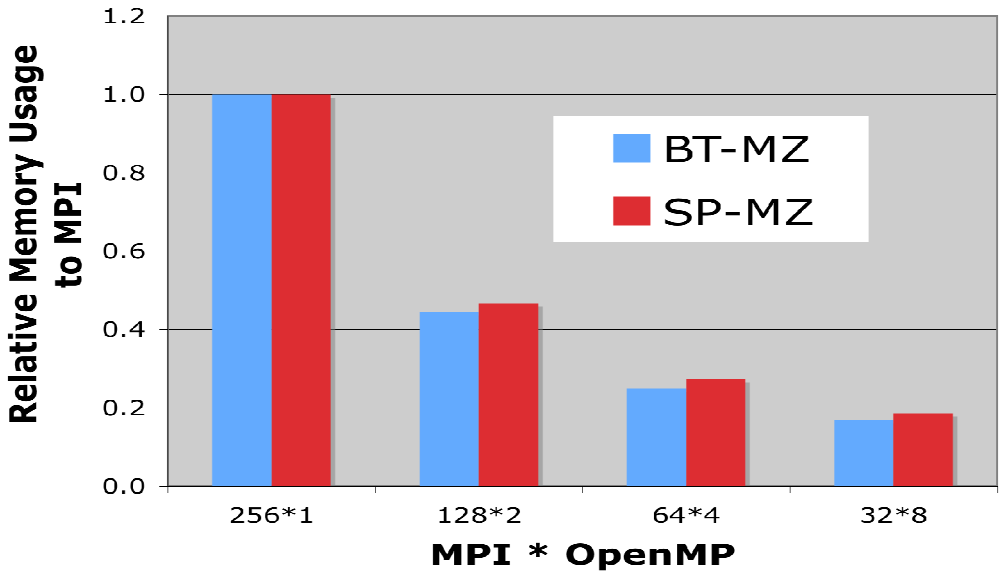


Doubling the number of threads through hyperthreading (SMT):

```
#!/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

MPI+OpenMP memory usage of NPB-MZ



Always same number of cores

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

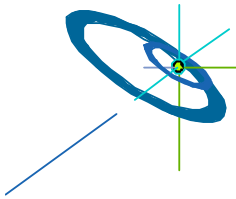
Hongzhang Shan, Haoqiang Jin, Karl Fuerlinger, Alice Koniges, Nicholas J. Wright:
Analyzing the Effect of Different Programming Models Upon Performance and Memory Usage on Cray XT5 Platforms.
Proceedings, CUG 2010, Edinburgh, GB, May 24-27, 2010.

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
- **Hybrid programming & accelerators**
  Gabriele Jost (Supersmith, Maximum Performance Software)
- 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



H L R S

SUPERsmith

OpenMP 4.0 Support for Co-Processors

- **New concepts:**
  - **Device:** An implementation defined logical execution engine; local storage which could be shared with other devices; device could have one or more processors
- **Extension to the previous Memory Model:**
  - **Previous:** Relaxed-Consistency Shared-Memory
  - **Added in 4.0 :**
    - **Device** with local storage
    - Data movement can be explicitly indicated by compiler directives
    - **League:** Set of thread teams created by a “teams” construct
    - **Contention group**: threads within a team; OpenMP synchronization restricted to contention groups.
- **Extension to the previous Execution Model**
  - **Previous:** Fork-join of OpenMP threads
  - **Added in 4.0**:
    - Host device offloads a region for execution on a **target device**
    - Host device waits for completion of execution on the target device

OpenMP Accelerator Additions

Target data
Place objects on the device
Target
Move execution to a device
Target update
Update objects on the device or host
Declare target
Place objects on the device
Place subroutines/functions on the device
Teams
Start multiple **contention groups**
Distribute
Similar to the OpenACC loop construct, binds to teams construct
Array sections
Current Status:
Accelerator support version 1 accepted
Currently open for public review:
http://www.openmp.org/mp-documents/OpenMP_4.0_RC2.pdf

- The "**target data**" construct**:**
  - When a target data construct is encountered, a new device data environment is created, and the encountering task executes the target data region

pragma omp target data [device, map, if]

- The "**target**" construct:
  - Creates device data environment and specifies that the region is executed by a device. The encountering task waits for the device to complete the target region at the end of the construct

pragma omp target [device, map, if]

- The "**teams**" construct:
  - Creates a league of thread teams. The master thread of each team executes the teams region

pragma omp teams [num_teams, num_threads, …]

- The "**distribute**" construct:
  - Specifies that the iterations of one or more loops will be executed by the thread teams. The iterations of the loop are distributed across the master threads of all teams

pragma omp distribute [collapse, dist_schedule, ….]

OpenMP 4.0 Example

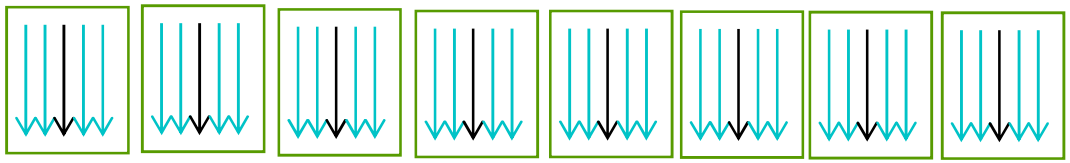
```
      void smooth( float* restrict a, float* restrict b,
        float w0, float w1, float w2, int n, int m, int niters )
{
   int i, j, iter;
   float* tmp;

   for( iter = 1; iter < niters; ++iter ){

      for( i = 1; i < n-1; ++i )

         for( j = 1; j < m-1; ++j )
            a[i*m+j] = w0 * b[i*m+j] +
               w1*(b[(i-1)*m+j] + b[(i+1)*m+j] + b[i*m+j-1] +
                                  b[i*m+j+1]) +
               w2*(b[(i-1)*m+j-1] + b[(i-1)*m+j+1] +b[(i+1)*m+j-1] +
                                    b[(i+1)*m+j+1]);
      tmp = a;  a = b;  b = tmp;
   }
}
In main:

{
smooth( a, b, w0, w1, w2, n, m, iters );
}
```

OpenMP 4.0 Example

```
      void smooth( float* restrict a, float* restrict b,
        float w0, float w1, float w2, int n, int m, int niters )
{
   int i, j, iter;
   float* tmp;
#pragma omp target mapto(b[0:n*m]) map(a[0:n*m])
#pragma omp team num_teams(8) num_maxthreads(5)
   for( iter = 1; iter < niters; ++iter ){
#pragma omp distribute dist_schedule(static) // chunk across teams
      for( i = 1; i < n-1; ++i )
#pragma omp parallel for // chunk across threads
         for( j = 1; j < m-1; ++j )
            a[i*m+j] = w0 * b[i*m+j] +
               w1*(b[(i-1)*m+j] + b[(i+1)*m+j] + b[i*m+j-1] +
                                  b[i*m+j+1]) +
               w2*(b[(i-1)*m+j-1] + b[(i-1)*m+j+1] +b[(i+1)*m+j-1] +
                                    b[(i+1)*m+j+1]);
      tmp = a;  a = b;  b = tmp;
   } }
In main:
#pragma omp target data map(b[0:n*m],a[0:n*m])
{
smooth( a, b, w0, w1, w2, n, m, iters );
}
```

OpenMP 4.0 *Team* and *Distribute* Construct

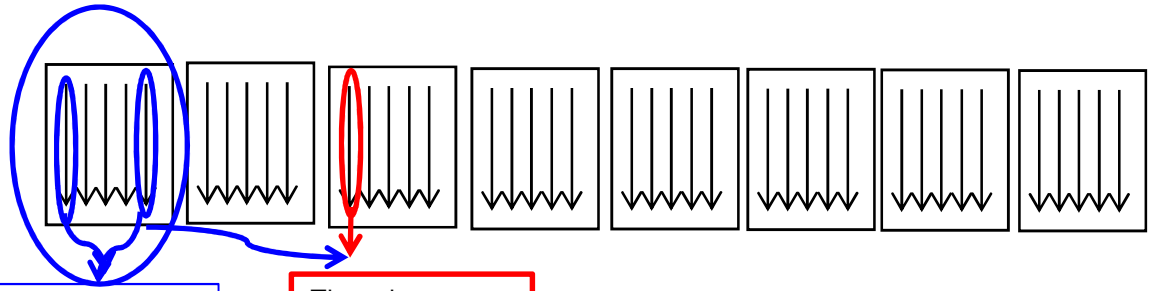
```
#pragma omp target device(acc)
#pragma omp team num_teams(8) num_maxthreads(5)
{
```



```
  Stmt1;
```

only executed by master thread of each team

```
  #pragma omp distribute   // chunk across thread blocks
  for (i=0; i<N; i++)
  #pragma omp parallel for  // chunk across threads
  for (j=0; j<M; j++)
  {
```



What is OpenACC?

- API that supports off-loading of loops and regions of code (e.g. loops) from a host CPU to an attached accelerator in C, C++, and Fortran
- Managed by a nonprofit corporation formed by a group of companies:
  - CAPS Enterprise, Cray Inc., PGI and NVIDIA
- Set of compiler directives, runtime routines and environment variables
- Simple programming model for using accelerators (focus on GPGPUs)
- Memory model:
  - Host CPU + Device may have completely separate memory; Data movement between host and device performed by host via runtime calls; Memory on device may not support memory coherence between execution units or need to be supported by explicit barrier
- Execution model:
  - Compute intensive code regions offloaded to the device, executed as kernels ; Host orchestrates data movement, initiates computation, waits for completion; Support for multiple levels of parallelism, including SIMD (gangs, workers, vector)
- Example constructs: ***acc parallel loop, acc data***

OpenACC Example

```
      void smooth( float* restrict a, float* restrict b,
        float w0, float w1, float w2, int n, int m, int niters )
{
   int i, j, iter;
   float* tmp;
   for( iter = 1; iter < niters; ++iter ){
       #pragma acc parallel loop gang(16) worker(8) //chunk across gangs and workers
      for( i = 1; i < n-1; ++i )
         #pragma acc vector (32) // execute in SIMD mode
         for( j = 1; j < m-1; ++j )
             a[i*m+j] = w0 * b[i*m+j] +
                 w1*(b[(i-1)*m+j] + b[(i+1)*m+j] + b[i*m+j-1] +
                                    b[i*m+j+1]) +
                 w2*(b[(i-1)*m+j-1] + b[(i-1)*m+j+1] +b[(i+1)*m+j-1] +
                                      b[(i+1)*m+j+1]);
      tmp = a;   a = b;   b = tmp;
   } }
In main:
#pragma acc data copy (b[0:n*m],a[0:n*m])
{
smooth( a, b, w0, w1, w2, n, m, iters );
}
```

CAPS HMPPWorkbench compiler:

acc_test.c:11: Loop 'j' was vectorized(32)
acc_test.c:9: Loop 'i' was shared among gangs(16) and workers(8)

Mantevo miniGhost on Cray XK7

- Mantevo 1.0.1 miniGhost 1.0
  - -Finite-Difference Proxy Application
  - -27 PT Stencil + Boundary Exchange of Ghost Cells
  - -Implemented in Fortran;
  - -MPI+OpenMP and MPI+OpenACC
  - -http://www.mantevo.org
- Test System:
  - -Located at HLRS Stuttgart,
- Test Case:Problem size 384x796x384, 10 variables, 20 time steps
- Compilation:
  - •pgf90 13.4-0
  - •-O3 -fast –fastsse –m -acc

```
!$acc data present ( GRID)

! Back boundary

IF ( NEIGHBORS(BACK) /= -1 ) THEN
        TIME_START_DIR = MG_TIMER ()
!$acc data present ( SEND_BUFFER_BACK )
!$acc parallel loop

 DO J = 0, NY+1
   DO I = 0, NX+1
    SEND_BUFFER_BACK(COUNT_SEND_BACK + J*(NX+2) + I + 1) = &
                GRID ( I, J, 1 )
           END DO
         END DO
!$acc end data
#endif

. . .
```

Packing of boundary data

```
CALL MPI_WAITANY ( MAX_NUM_SENDS + MAX_NUM_RECVS, MSG_REQS, ...  )
....
!$acc            data present ( RECV_BUFFER_BACK )
!$acc            update device ( RECV_BUFFER_BACK )
!$acc            end data$acc data present ( GRID)
```

Unpacking of boundary data

Cray XK7 Hermit

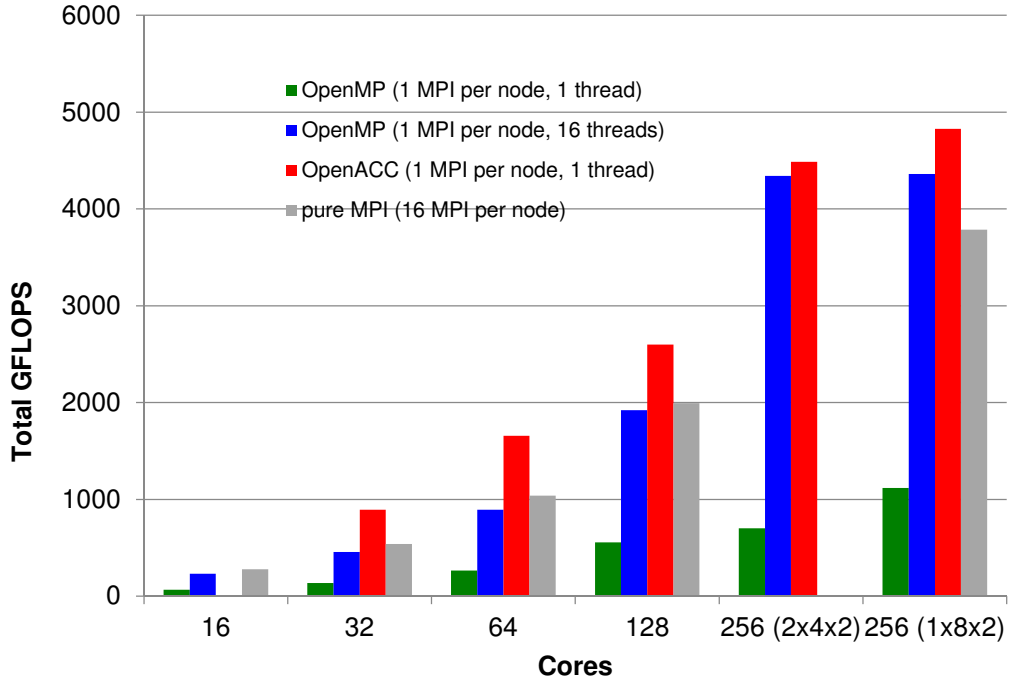
- Located at HLRS Stuttgart, Germany (https://wickie.hlrs.de/platforms/index.php/Cray_XE6)
- 16 Cray XK7 compute nodes; AMD's 16-core Opteron™ 6200 Series processor with NVIDIA® Tesla® K20 GPU Accelerator Cards

```
--------------------------------------------------------------
CPU type:       AMD Interlagos processor
*************************************************************
Hardware Thread Topology
*************************************************************
Sockets:        1
Cores per socket:       16
Threads per core:       1
--------------------------------------------------------------

Socket 0:
+-------------------------------------------------------------------------------------------------------------------------------+
| +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ |
| |  0 | |  1 | |  2 | |  3 | |  4 | |  5 | |  6 | |  7 | |  8 | |  9 | | 10 | | 11 | | 12 | | 13 | | 14 | | 15 | |
| +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ |
| +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ |
| | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | | 16kB | |
| +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+ |
| +-------------+ +-------------+ +-------------+ +-------------+ +-------------+ +-------------+ +-------------+ +-------------+ |
| |    2MB    | |    2MB    | |    2MB    | |    2MB    | |    2MB    | |    2MB    | |    2MB    | |    2MB    | |
| +-------------+ +-------------+ +-------------+ +-------------+ +-------------+ +-------------+ +-------------+ +-------------+ |
| +---------------------------------------------------------+ +---------------------------------------------------------+ |
| |                          6MB                            | |                          6MB                            | |
| +---------------------------------------------------------+ +---------------------------------------------------------+ |
+-------------------------------------------------------------------------------------------------------------------------------+
```

Mantevo miniGhost: 27-PT Stencil

```
#if defined _MOG_OMP
!$OMP PARALLEL DO PRIVATE(SLICE_BACK, SLICE_MINE, SLICE_FRONT)
#else
!$acc data present ( WORK )
!$acc parallel
!$acc loop
#endif
      DO K = 1, NZ
         DO J = 1, NY
            DO I = 1, NX

               SLICE_BACK =  GRID(I-1,J-1,K-1) + GRID(I-1,J,K-1) + GRID(I-1,J+1,K-1) + &
                             GRID(I  ,J-1,K-1) + GRID(I  ,J,K-1) + GRID(I  ,J+1,K-1) + &
                             GRID(I+1,J-1,K-1) + GRID(I+1,J,K-1) + GRID(I+1,J+1,K-1)

               SLICE_MINE =  GRID(I-1,J-1,K)   + GRID(I-1,J,K)   + GRID(I-1,J+1,K) + &
                             GRID(I  ,J-1,K)   + GRID(I  ,J,K)   + GRID(I  ,J+1,K) + &
                             GRID(I+1,J-1,K)   + GRID(I+1,J,K)   + GRID(I+1,J+1,K)

               SLICE_FRONT = GRID(I-1,J-1,K+1) + GRID(I-1,J,K+1) + GRID(I-1,J+1,K+1) + &
                             GRID(I  ,J-1,K+1) + GRID(I  ,J,K+1) + GRID(I  ,J+1,K+1) + &
                             GRID(I+1,J-1,K+1) + GRID(I+1,J,K+1) + GRID(I+1,J+1,K+1)

               WORK(I,J,K) = ( SLICE_BACK + SLICE_MINE + SLICE_FRONT ) / 27.0

            END DO
         END DO
      END DO
```

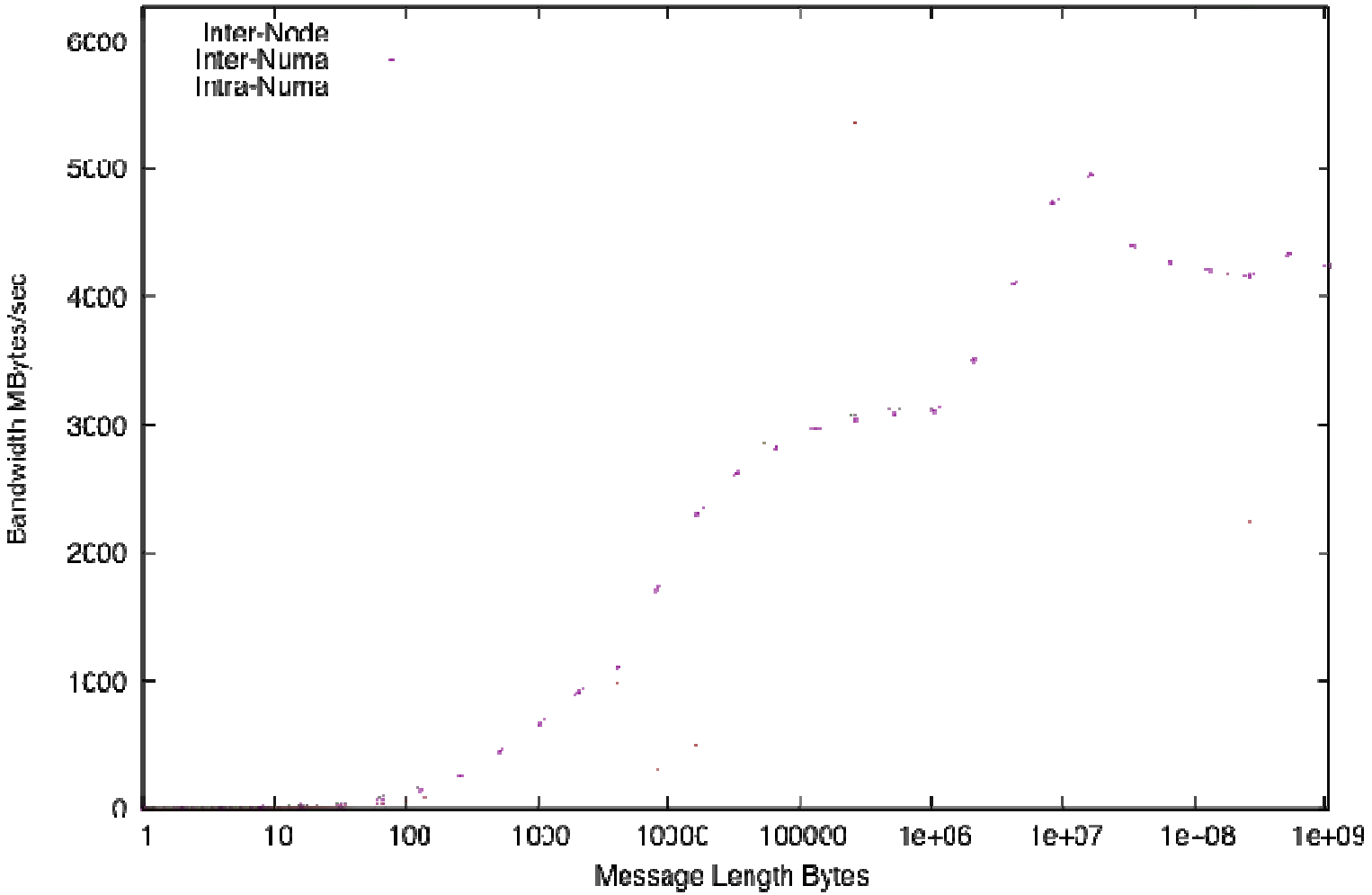
Scalability of miniGhost on Cray XK7



	Total Time(sec)	Comm. Time (sec)
OpenMP (16x1t)	12.1	0.4
OpenMP (16x16t)	1.9	0.16
OpenACC (16x16t)	1.17	0.34
Pure MPI (256 Ranks)	1.5	0.28

Elapsed time as reported by the application
Communication includes packing/unpacking

IMB Bandwidth Ping-Pong XK7



Profiling Information: export PGI_ACC_TIME=1

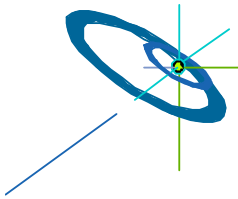
```
/univ_1/ws1/ws/hpcjost-ISC13_GJOST-0/miniGhost_OpenACC_1.0/MG_UNPACK_BSPMA.F
  mg_unpack_bspma  NVIDIA  devicenum=0
        time(us): 36,951
        124: data copyin reached 20 times
             device time(us): total=8,603 max=431 min=429 avg=430
          ….

/univ_1/ws1/ws/hpcjost-ISC13_GJOST-0/miniGhost_OpenACC_1.0/MG_STENCIL_COMPS.F
  mg_stencil_3d27pt  NVIDIA  devicenum=0
        time(us): 1,063,875
        330: kernel launched 200 times
            grid: [160]  block: [256]
             device time(us): total=1,063,875 max=5,337 min=5,302 avg=5,319
            elapsed time(us): total=1,073,817 max=5,444 min=5,349 avg=5,369
        …

/univ_1/ws1/ws/hpcjost-ISC13_GJOST-0/miniGhost_OpenACC_1.0/MG_SEND_BSPMA.F
  mg_send_bspma  NVIDIA  devicenum=0
        time(us): 33,150
        94: data copyout reached 20 times
             device time(us): total=7,800 max=392 min=389 avg=390
         …

device time(us): total=12,618 max=633 min=630 avg=630
/univ_1/ws1/ws/hpcjost-ISC13_GJOST-0/miniGhost_OpenACC_1.0/MG_PACK.F
  mg_pack  NVIDIA  devicenum=0
        time(us): 9,615
        91: kernel launched 200 times
            grid: [98]  block: [256]
             device time(us): total=2,957 max=68 min=13 avg=14
            elapsed time(us): total=11,634 max=107 min=51 avg=58
```

Profiling Information: export PGI_ACC_TIME=1

```
Accelerator Kernel Timing data
/univ_1/ws1/ws/hpcjost-ISC13_GJOST-0/miniGhost_OpenACC_1.0/MG_STENCIL_COMPS.F
  mg_stencil_3d27pt  NVIDIA  devicenum=0
        time(us): 1,064,197
        330: kernel launched 200 times
            grid: [160]  block: [256]
             device time(us): total=1,064,197 max=5,351 min=5,299 avg=5,320
            elapsed time(us): total=1,074,081 max=5,442 min=5,348 avg=5,370

/univ_1/ws1/ws/hpcjost-ISC13_GJOST-0/miniGhost_OpenACC_1.0/MG_PACK.F
  mg_pack  NVIDIA  devicenum=0
        time(us): 9,568
        91: kernel launched 200 times
            grid: [98]  block: [256]
             device time(us): total=2,924 max=70 min=12 avg=14
            elapsed time(us): total=11,624 max=110 min=51 avg=58
        195: kernel launched 200 times
            grid: [162]  block: [256]
             device time(us): total=3,432 max=120 min=15 avg=17
            elapsed time(us): total=11,385 max=160 min=53 avg=56
        221: kernel launched 200 times
            grid: [162]  block: [256]
             device time(us): total=3,212 max=19 min=15 avg=16
            elapsed time(us): total
```

Conclusions for miniGhost Experiment:

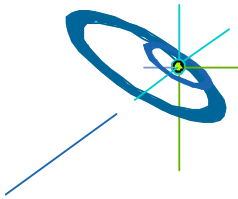
- Hybrid MPI/OpenMP and MPI/OpenACC yield performance increase over pure MPI
- Compiler pragma based API provides relatively easy way to exploit coprocessors
- OpenACC targeted toward GPU type coprocessors
- OpenMP 4.0 extensions will provide flexibility to exploit a wide range of heterogeneous coprocessors (GPU, APU, heterogeneous many-core types)



Outline

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

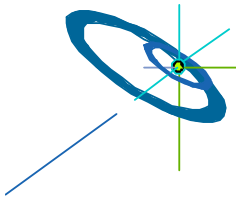
  Georg Hager, Regionales Rechenzentrum Erlangen (RRZE)
- Mismatch Problems
- Application categories that can benefit from hybrid parallelization
- Thread-safety quality of MPI libraries
- Tools for debugging and profiling MPI+OpenMP
- Other options on clusters of SMP nodes
- Summary



Hybrid Programming How-To: Overview

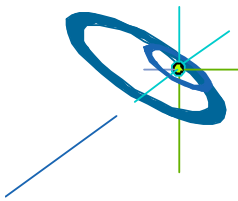
- A practical introduction to hybrid programming
  - How to compile and link
  - Getting a hybrid program to run on a cluster

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



How to compile, link and run

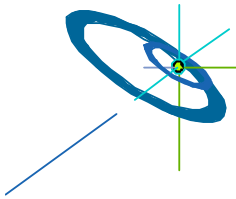
- Use appropriate OpenMP compiler switch (-openmp, -fopenmp, -mp, -qsmp=openmp, …) and MPI compiler script (if available)
- Link with MPI library
  - Usually wrapped in MPI compiler script
  - If required, specify to link against thread-safe MPI library
    - **Often automatic when OpenMP or auto-parallelization 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 fewer MPI processes than cores on your nodes



Examples for compilation and execution

- **Cray XE6 (4 NUMA domains w/ 8 cores each):**
  - `ftn -h omp ...`
  - `export OMP_NUM_THREADS=8`
  - `aprun -n nprocs -N nprocs_per_node \`
    `-d $OMP_NUM_THREADS a.out`

- **Intel Sandy Bridge (8-core 2-socket) cluster, Intel MPI/OpenMP**
  - `mpiifort -openmp ...`
  - `OMP_NUM_THREADS=8 mpirun -ppn 2 -np 4 \`
    `-env I_MPI_PIN_DOMAIN socket \`
    `-env KMP_AFFINITY scatter ./a.out`



Interlude: Advantages of mpiexec or similar mechanisms

- Startup mechanism should use a resource manager 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**
    - **Allowing password-less user login not required between nodes**
  - Interfaces directly with batch system to determine number of procs

- Provisions for starting fewer processes per node than available cores
  - Required for hybrid programming
  - E.g., "**`-pernode`**" and "**`-npernode #`**" options – does not require messing around with nodefiles

Running the code

More 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
```

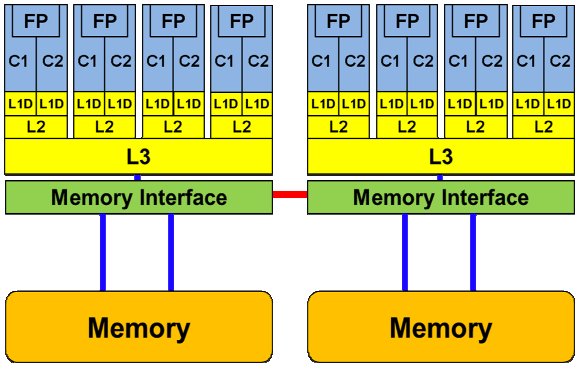
Where do the threads run?
→ see later!

- Pure MPI:

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

Running the code *efficiently?*

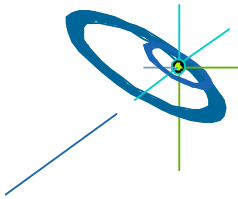
- Symmetric, UMA-type compute nodes have become rare animals
  - NEC SX
  - Intel 1-socket (Xeon 12XX) – rare in cluster environments
  - Hitachi SR8000, IBM SP2, single-core multi-socket Intel Xeon… (all dead)
- Instead, systems have become “non-isotropic” on the node level
  - ccNUMA (AMD Opteron, SGI Altix, IBM Power6 (p575), Intel Sandy Bridge)
  - 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

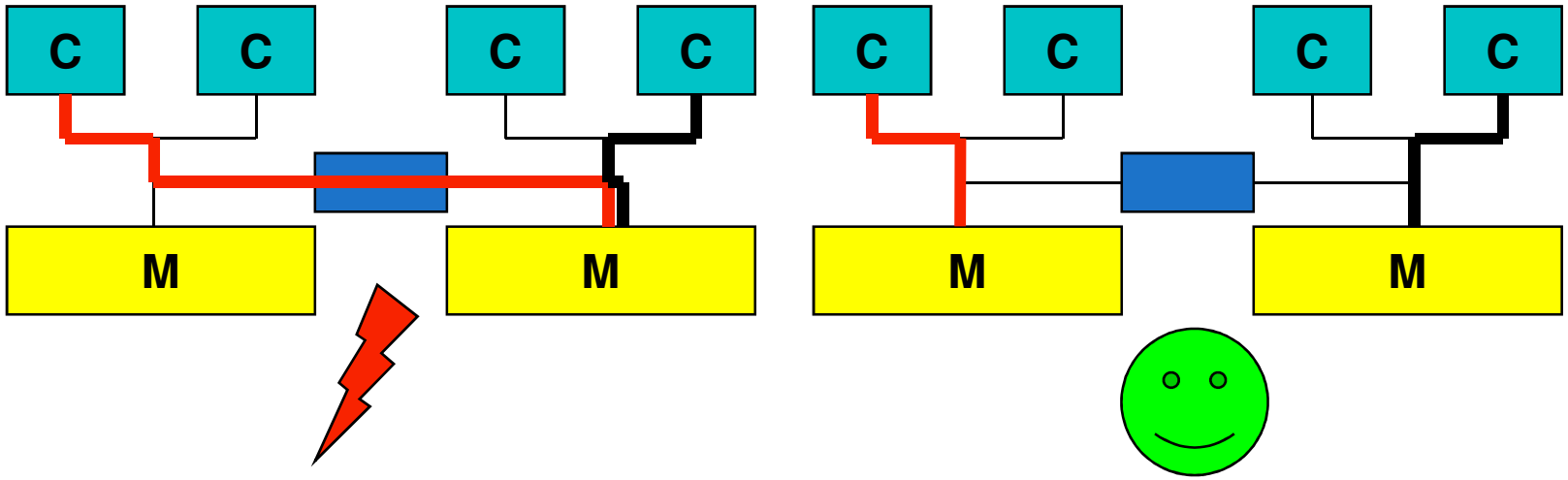
- ccNUMA locality effects
  - Penalties for access across locality domains
  - Impact of contention
  - Consequences of file I/O for page placement
  - Placement of MPI buffers

- 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



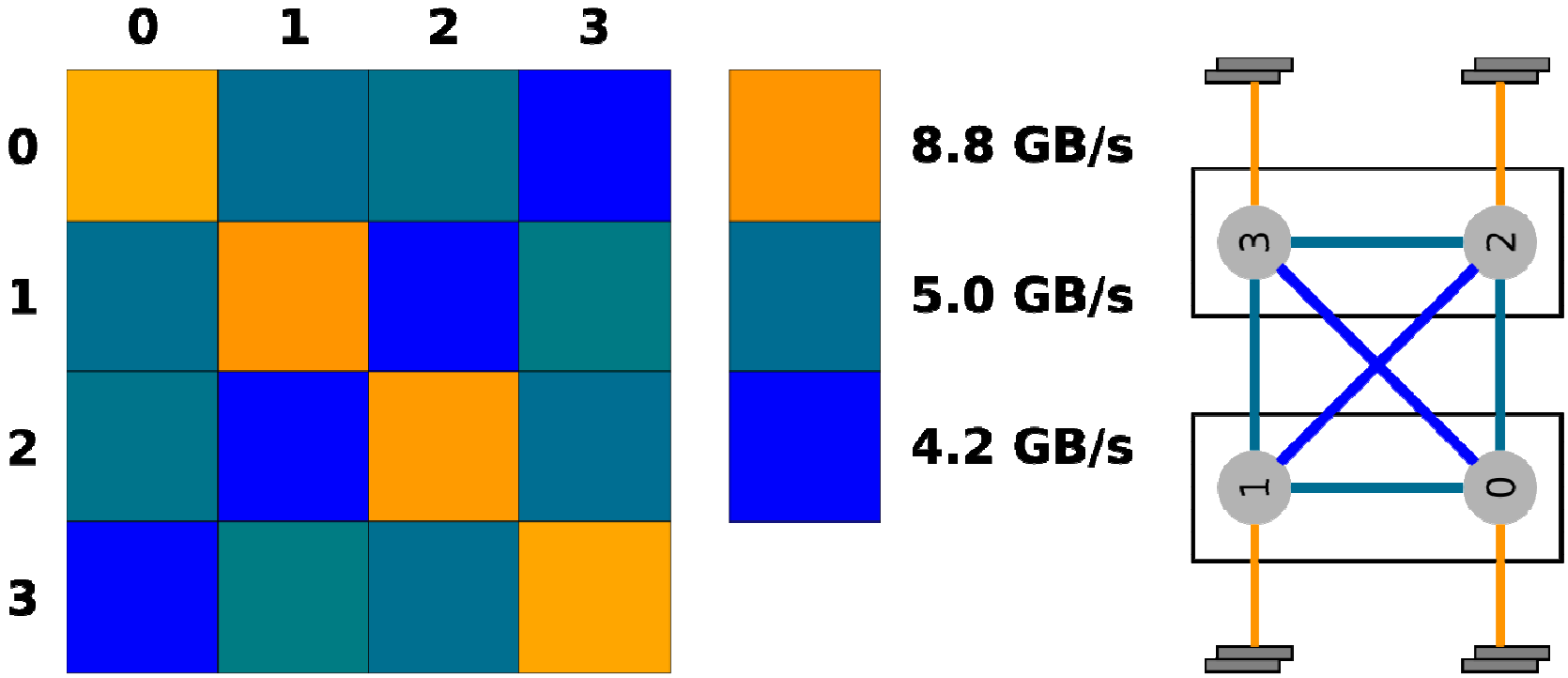
A short introduction to ccNUMA

- 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

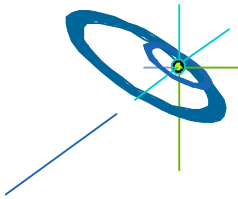


ccNUMA Memory Locality Problems

- 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

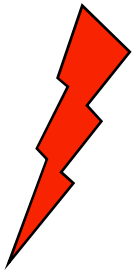
- "Golden Rule" of ccNUMA:
  A memory page gets mapped into the local memory of the processor that first touches it!
  - Except if there is not enough local memory available
  - this might be a problem, see later
  - Some OSs allow to influence placement in more direct ways
    - **cf. libnuma (Linux),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

Most simple case: explicit initialization

```
integer,parameter :: N=10000000
double precision A(N), B(N)



A=0.d0



!$OMP parallel do
do i = 1, N
  B(i) = function ( A(i) )
end do
!$OMP end parallel do
```

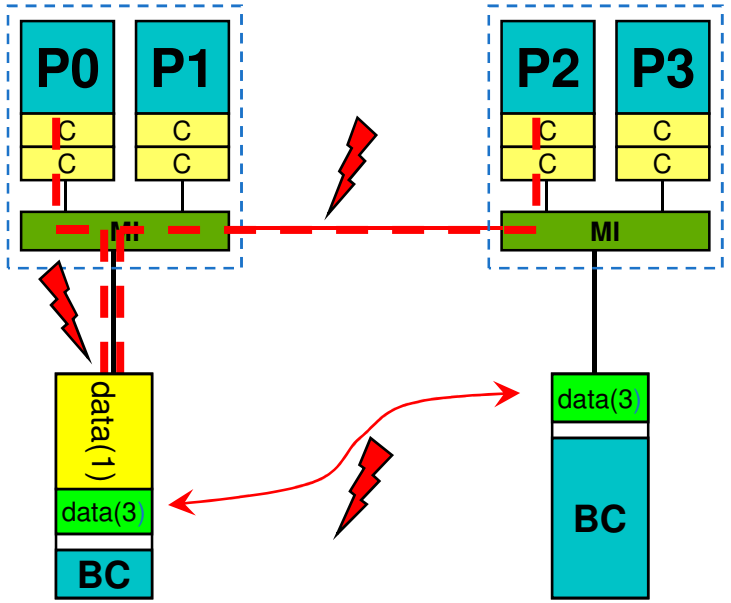


```
integer,parameter :: N=10000000
double precision A(N),B(N)
!$OMP parallel
!$OMP do schedule(static)
do i = 1, N
  A(i)=0.d0
end do
!$OMP end do
...
!$OMP do schedule(static)
do i = 1, N
  B(i) = function ( A(i) )
end do
!$OMP end do
!$OMP end parallel
```

skipped

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!)

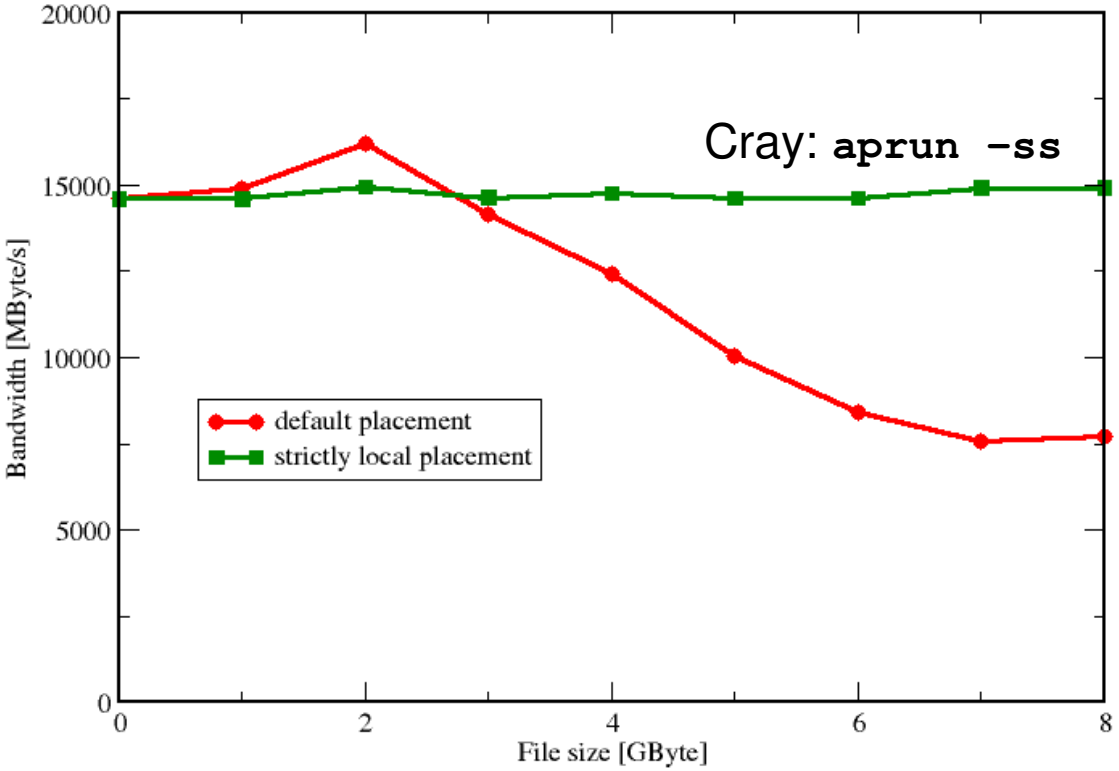
skipped

ccNUMA problems beyond first touch:
Buffer cache

Real-world example: ccNUMA and the Linux buffer cache

Benchmark:

1. Write a file of some size from LD0 to disk
2. Perform bandwidth benchmark using all cores in LD0 and maximum memory installed in LD0



Result: By default, Buffer cache is given priority over local page placement

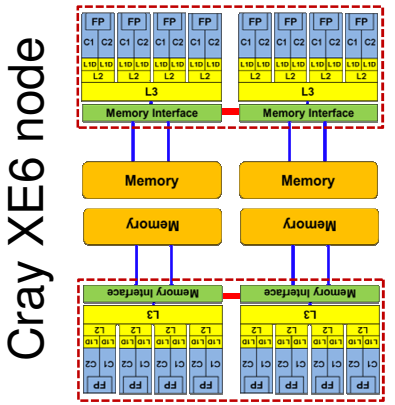
→ restrict to local domain if possible!

SUPERsmith

Intra-node MPI characteristics: IMB Ping-Pong benchmark

- Code (to be run on 2 cores):

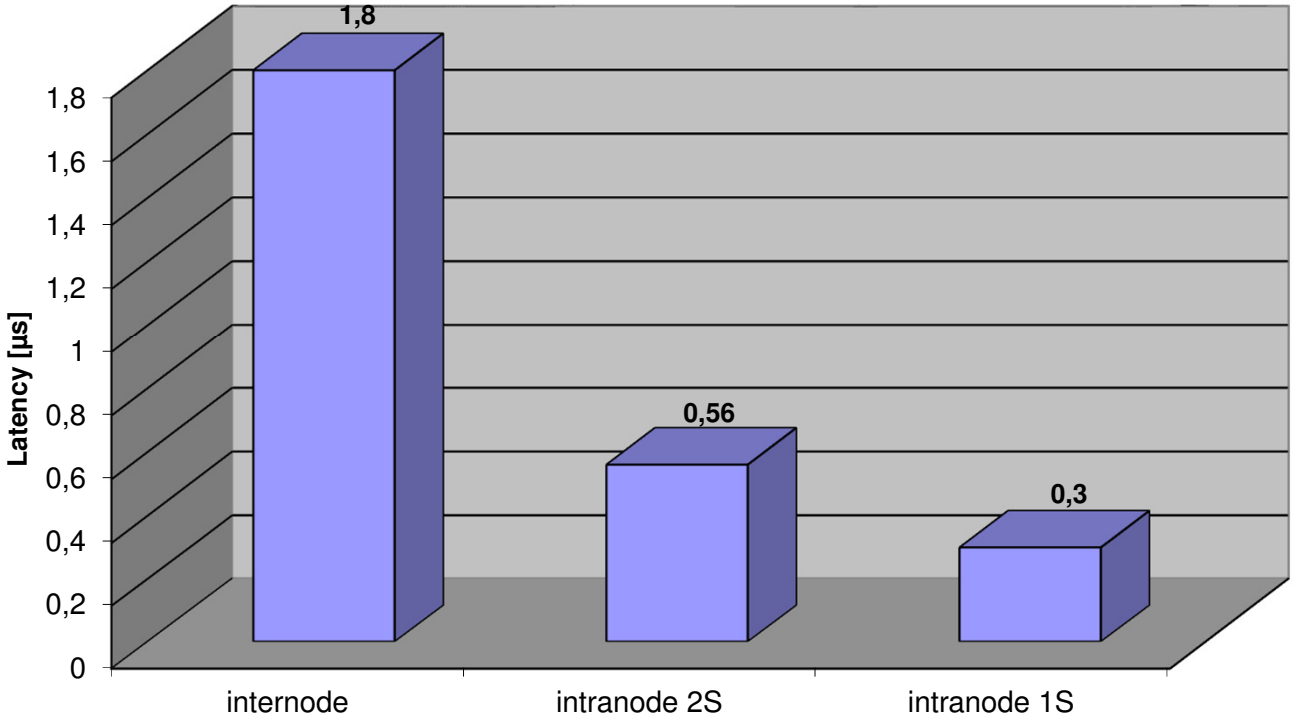
```
wc = MPI_WTIME()

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
```

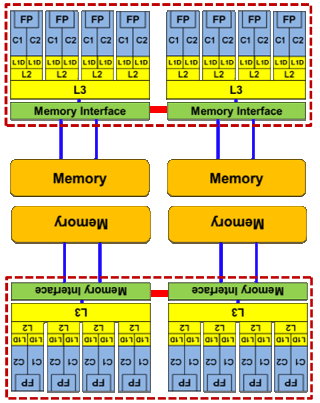


- Intranode (1S): `aprun -n 2 -cc 0,1 ./a.out`
- Intranode (2S): `aprun -n 2 -cc 0,16 ./a.out`
- Internode: `aprun -n 2 -N 1 ./a.out`

IMB Ping-Pong: Latency

Intra-node vs. Inter-node on Cray XE6



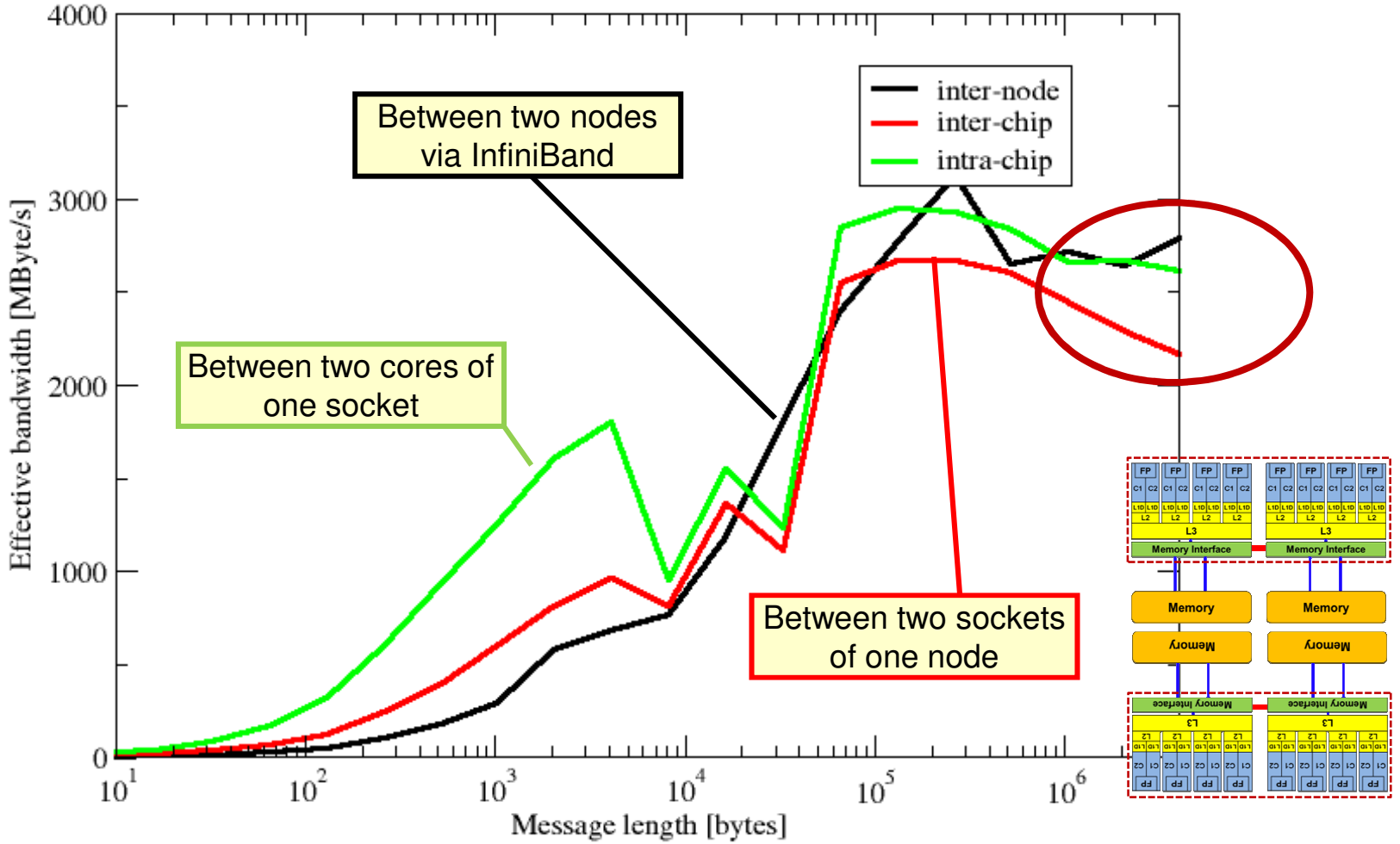


Affinity matters!

SUPERsmith

IMB Ping-Pong: Bandwidth Characteristics

Intra-node vs. Inter-node on Cray XE6



The throughput-parallel vector triad benchmark

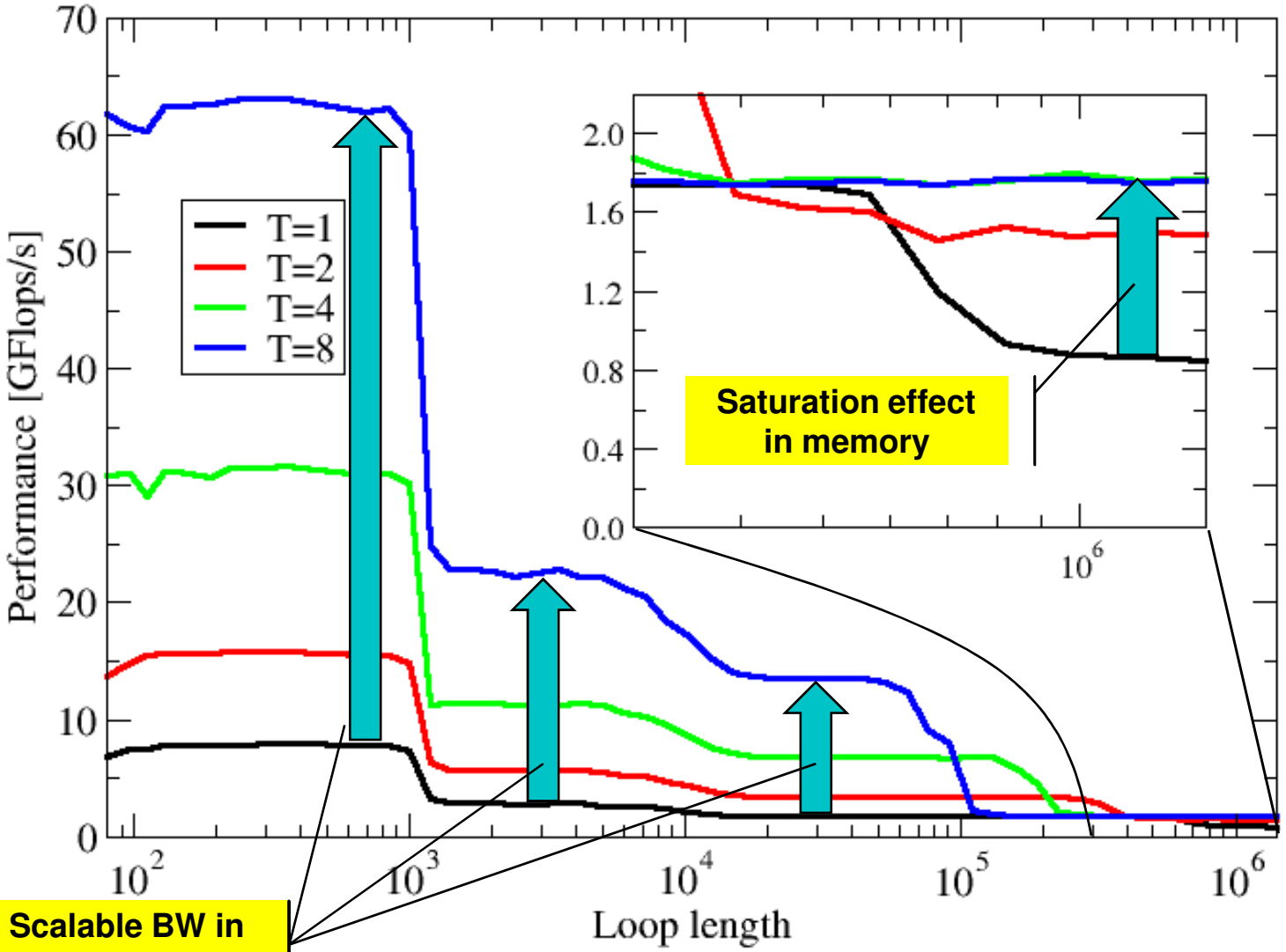
Microbenchmarking for architectural exploration

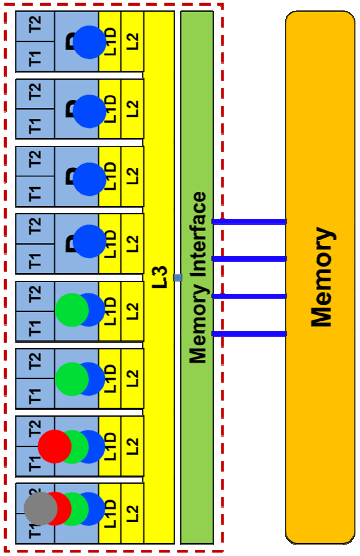
- Every core runs its own, independent triad benchmark

```
double precision, dimension(:), allocatable :: A,B,C,D

!$OMP PARALLEL private(i,j,A,B,C,D)
allocate(A(1:N),B(1:N),C(1:N),D(1:N))
A=1.d0; B=A; C=A; D=A
do j=1,NITER
  do i=1,N
    A(i) = B(i) + C(i) * D(i)
  enddo
  if(.something.that.is.never.true.) then
    call dummy(A,B,C,D)
  endif
enddo
!$OMP END PARALLEL
```

- → pure hardware probing, no impact from OpenMP overhead

Throughput vector triad on Sandy Bridge socket (3 GHz)





Scalable BW in L1, L2, L3 cache

SUPERsmith

The OpenMP-parallel vector triad benchmark

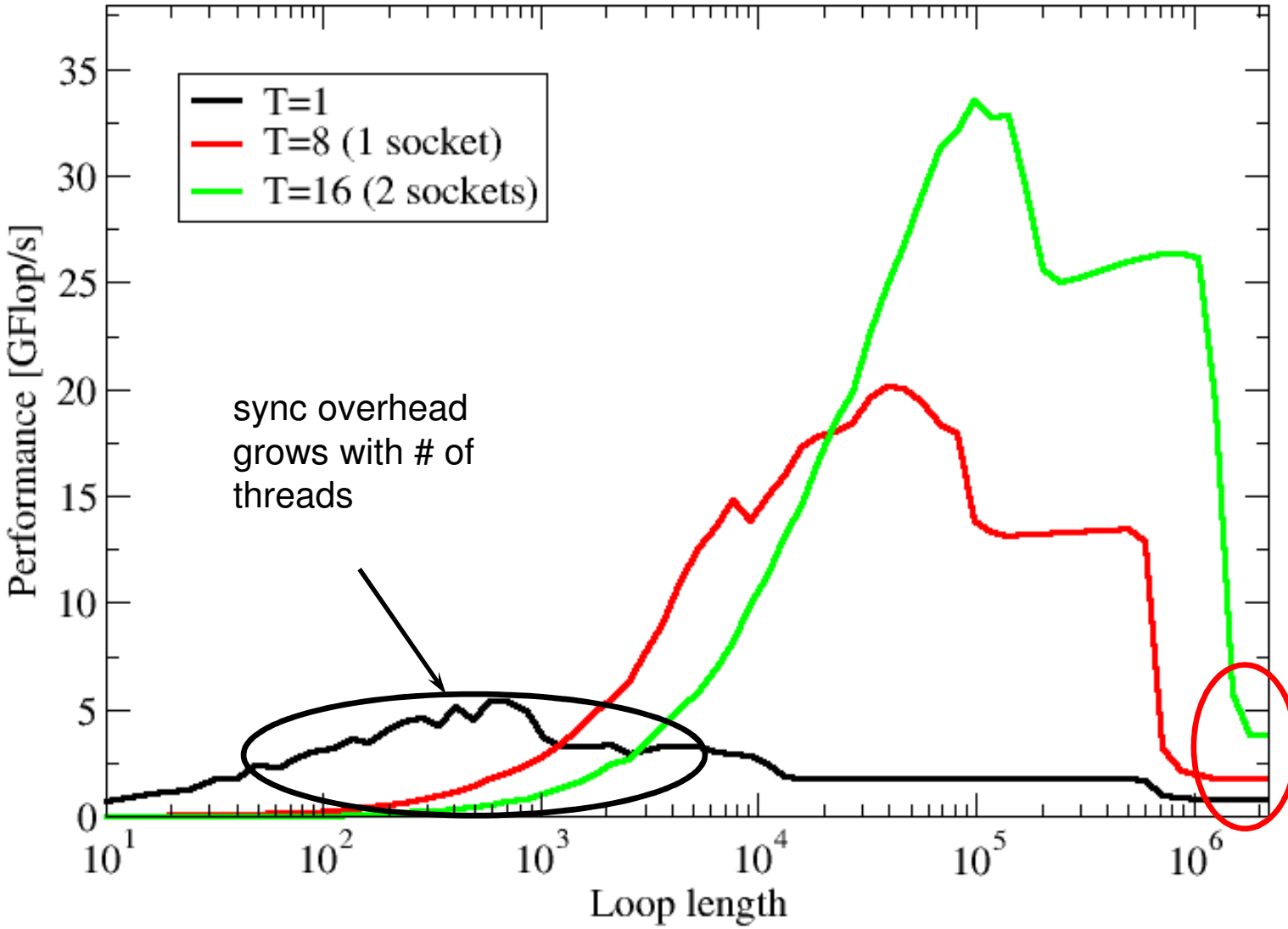
Visualizing OpenMP overhead

- OpenMP work sharing in the benchmark loop

```
double precision, dimension(:), allocatable :: A,B,C,D

allocate(A(1:N),B(1:N),C(1:N),D(1:N))
A=1.d0; B=A; C=A; D=A
!$OMP PARALLEL private(i,j)
do j=1,NITER
!$OMP DO
  do i=1,N
    A(i) = B(i) + C(i) * D(i)
  enddo
!$OMP END DO
  if(.something.that.is.never.true.) then
    call dummy(A,B,C,D)
  endif
enddo
!$OMP END PARALLEL
```

Implicit barrier

OpenMP vector triad on Sandy Bridge socket (3 GHz)



sync overhead grows with # of threads

bandwidth scalability across memory interfaces

SUPERsmith

Thread synchronization overhead on SandyBridge-EP

Direct measurement of barrier overhead in CPU cycles

2 Threads	Intel 13.1.0	GCC 4.7.0	GCC 4.6.1
Shared L3	**384**	5242	4616
SMT threads	**2509**	3726	3399
Other socket	1375	5959	4909

Gcc still not very competitive

Intel compiler

Full domain	Intel 13.1.0	GCC 4.7.0	GCC 4.6.1
Socket	1497	14546	14418
Node	3401	34667	29788
Node +SMT	6881	59038	58898

Thread/Process Affinity ("Pinning")

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

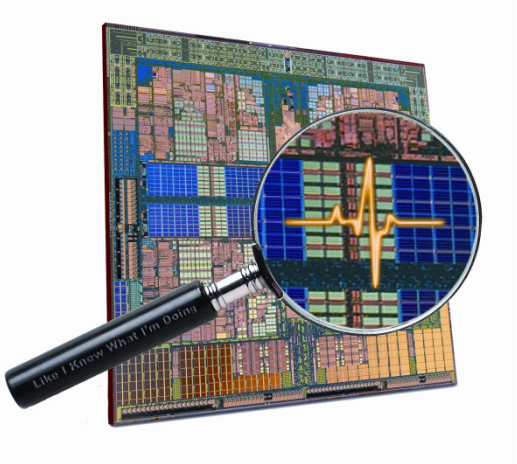
    Linux: `sched_setaffinity()`, PLPA → hwloc
    Solaris: `processor_bind()`
    Windows: `SetThreadAffinityMask()`
    …
- Support for "semi-automatic" pinning in some compilers/environments
  - Intel compilers > V9.1 (**`KMP_AFFINITY`** environment variable)
  - Pathscale
  - Generic Linux: **`taskset`**,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

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

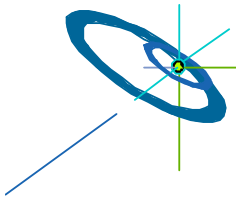
Likwid Tool Suite

- 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
  - AMD Magny Cours, Interlagos
  - Intel Xeon Phi in beta stage



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!

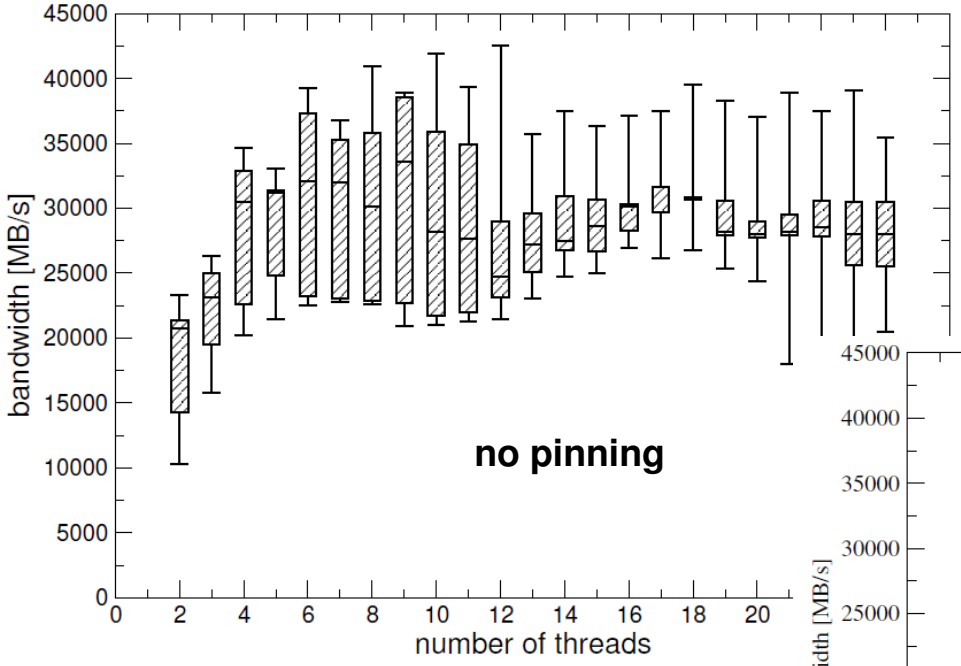
likwid-pin

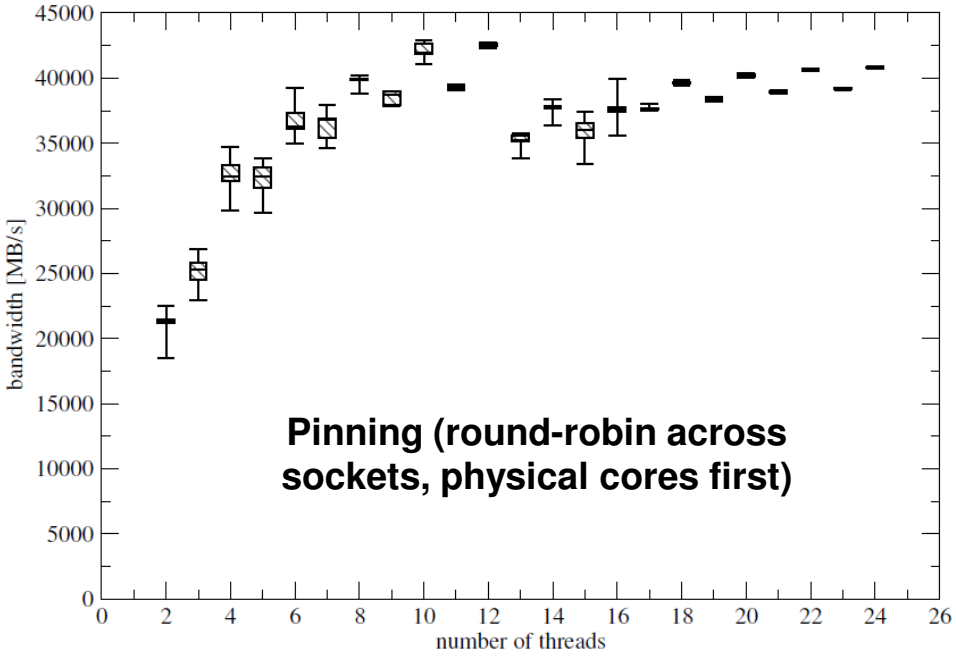
- Inspired and based on **`ptoverride`** (Michael Meier, RRZE) and **`taskset`**
- Pins process and threads to specific cores without touching code
- Directly supports pthreads, gcc OpenMP, Intel OpenMP
- Allows user to specify skip mask (i.e., supports many different compiler/MPI combinations)
- Can 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 -c 0,2,4-6  ./myApp parameters`
  - `env OMP_NUM_THREADS=6 likwid-pin -c S0:0-2@S1:0-2 ./myApp`

Example: STREAM benchmark on 12-core Intel Westmere:

Anarchy vs. thread pinning





SUPERsmith

Likwid-pin

Example: Intel OpenMP

- Running the STREAM benchmark with likwid-pin:

```
$ export OMP_NUM_THREADS=4
$ likwid-pin -c 0,1,4,5 ./stream
[likwid-pin] Main PID -> core 0 - OK
-------------------------------------------------
 Double precision appears to have 16 digits of accuracy
 Assuming 8 bytes per DOUBLE PRECISION word
-------------------------------------------------
[... some STREAM output omitted ...]
 The *best* time for each test is used
 *EXCLUDING* the first and last iterations
[pthread wrapper] PIN_MASK: 0->1  1->4  2->5
[pthread wrapper] SKIP MASK: 0x1
[pthread wrapper 0] Notice: Using libpthread.so.0
        threadid 1073809728 -> SKIP
[pthread wrapper 1] Notice: Using libpthread.so.0
        threadid 1078008128 -> core 1 - OK
[pthread wrapper 2] Notice: Using libpthread.so.0
        threadid 1082206528 -> core 4 - OK
[pthread wrapper 3] Notice: Using libpthread.so.0
        threadid 1086404928 -> core 5 - OK
[... rest of STREAM output omitted ...]
```

Main PID always pinned

Skip shepherd thread

Pin all spawned threads in turn

skipped

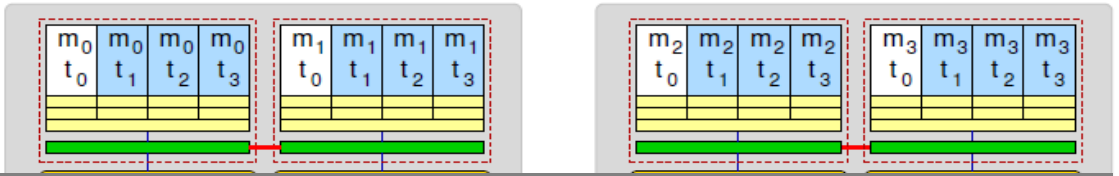
Topology ("mapping") choices with MPI+OpenMP:

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

One MPI process per node

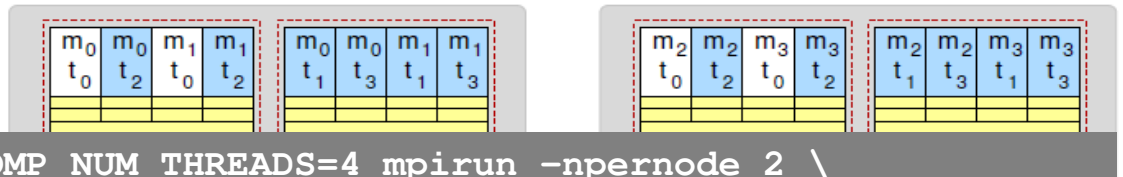

```
env OMP_NUM_THREADS=8 mpirun -pernode \
          likwid-pin -c 0-7 ./a.out
```

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 -c L:0,2,1,3 ./a.out
```

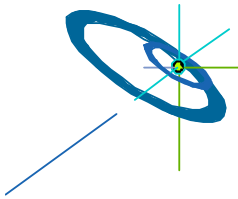
Two MPI processes per socket


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

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

- Learn how to take control of hybrid execution!

- Be aware of intranode MPI behavior
- Always observe the topology dependence of
  - Intranode MPI
  - OpenMP overheads
  - Saturation effects / scalability behavior with bandwidth-bound code
- 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

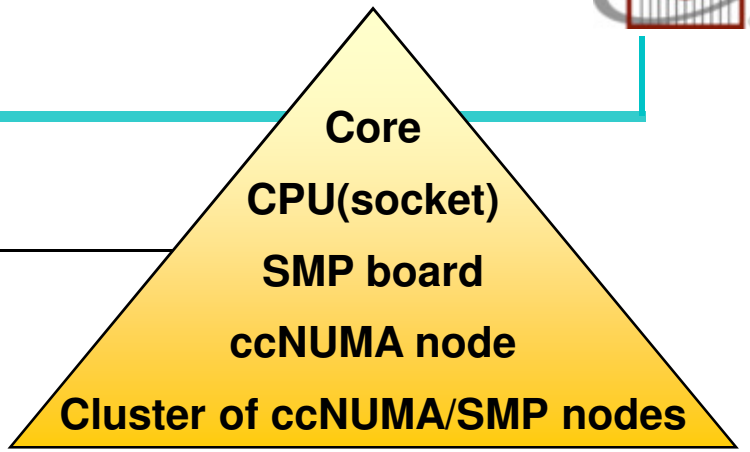


Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / pure MPI vs hybrid MPI+OpenMP
- Hybrid programming & accelerators
- 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

Problem

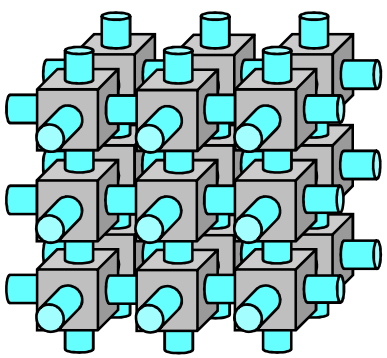
- Application topology is mapped to the hardware topology
  → communication topology and message sizes
  → communication overhead

Partially independent of the programming model:

pure MPI | hybrid MPI+OpenMP | Hybrid MPI+MPI

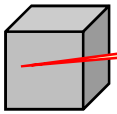
Simplifications:

- Ignore the hardware topology of the cluster network
- First optimize the cluster communication:
  N x N x N data on each SMP node
  (surface ~ neighbor communication,
  cube has minimal surface)
- Contiguous data on each SMP node

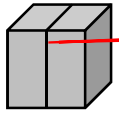


The Topology Problem, without inner halo communication

hybrid MPI+OpenMP
Hybrid MPI+MPI

Let's look into one SMP node 

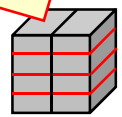
Shared data on one SMP node …



and how it may be split into ccNOMA domains

- With *d* ccNUMA locality domains (e.g., *d*=2)
- And *c* cores (i.e., *c*/*d* cores per physical shared memory, e.g. *c*/*d* = 4)

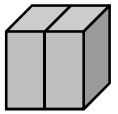
… and used by the cores



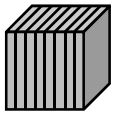
Without halo cells between the cores:
Communication only through neighbor accesses between ccNUMA domains

- Compare the ccNUMA communication (s = communication size per domain)
  - with *d*=2, and
  - *d*=8 and 1-dimensional data decomposition
  - *d*=8 and 3-dimensional data decomposition between the ccNUMA domains

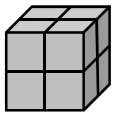
Example with N x N x N = 8 x 8 x 8 units, and neighbor access width = 1



s ~ 2 * 8*8 = 128

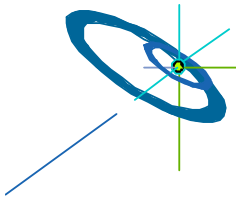


s ~ 2 * 8*8 = 128



s ~ 6 * 4*4 = 96

No real win!
Don't care about dimensions within the SMP nodes!
Make your software simple!



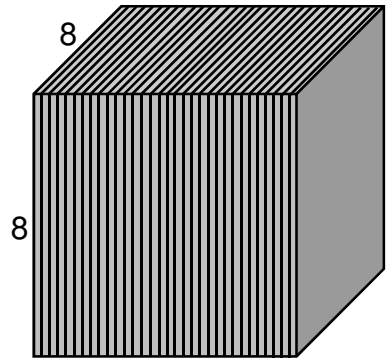
pure MPI

Hybrid MPI+MPI

The Topology Problem, with inner halo communication

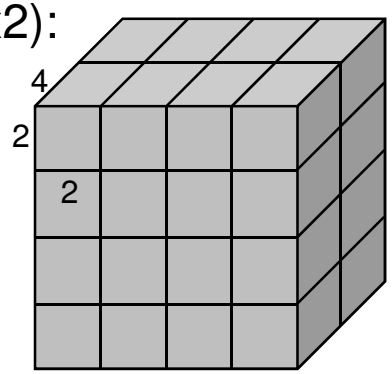
With halo cells and halo communication between the cores:

We ignore differences in core-to-core communication speed
- within ccNUMA domain, and
- between ccNUMA domains of one SMP node



Example with *c*=32 cores per SMP node

- *c*=32 and 1-dimensional data decomposition:
  s ~ 2 * 8*8 = 128
- *c*=32 and 3-dimensional data decomposition (4x4x2):
  s ~ 2 * (2*2 + 2*4 + 4*2) = 40



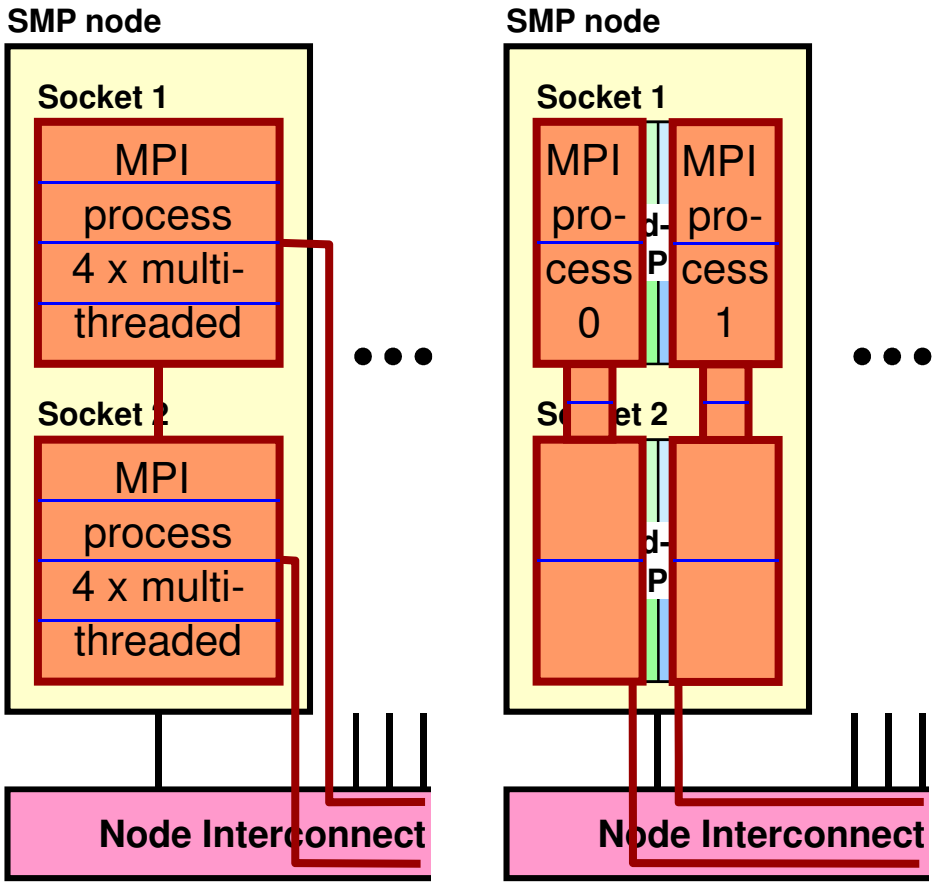
- In general: $\text{win} = \frac{s_{1\text{–dim}}}{s_{3\text{–dim}}} = \frac{\sqrt[3]{c^2}}{3}$

Example with N x N x N = 8 x 8 x 8units,
and neighbor access width = 1
(s = communication size per core)

c=16, 32, 64, … → win= factor 2, 3, 5, … ! **Real win?**
You may **not** care as long as your inner-node communication is below xx% !
Make your software simple !?

The Mapping Problem with mixed mode

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)

Hybrid MPI+MPI
(with halo communication)

Problem:

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

Solution:

- MPI+OpenMP: Only one MPI process per SMP node
- MPI+MPI: No halo-communication within an SMP node

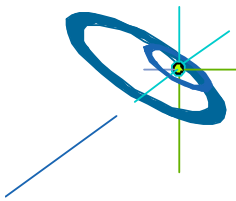
Remarks:

- MPI communication within an SMP node: 2 copies
  (user send buffer → shared memory → user recv buffer)
- MPI-3 shared memory halo commincation: 1 copy
  (user send buffer → user recv buffer)
- MPI-3 with direct access to neighbor data: 0 copy

pure MPI & Mixed model

Hybrid MPI+MPI
(with halo communication)

Hybrid MPI+MPI
(with direct neighbor access)

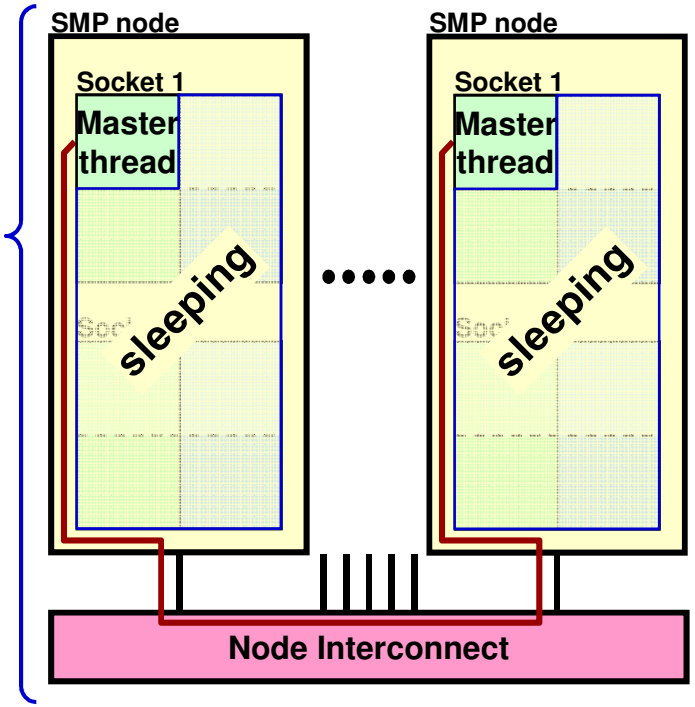


Sleeping threads and network saturation with Masteronly

hybrid MPI+OpenMP

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

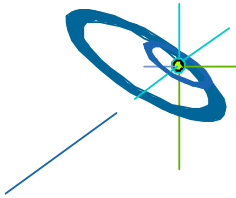
H L R S

SUPERsmith

OpenMP: Additional Overhead & Pitfalls

- Using OpenMP
  - → may prohibit compiler optimization
  - → **may cause significant loss of computational performance**
- Thread fork / join overhead
- On ccNUMA SMP nodes:
  - **Loss of performance due to missing memory page locality or missing first touch strategy**
  - E.g. with the masteronly scheme:
    - One thread produces data
    - Master thread sends the data with MPI
    - → data may be internally communicated from one memory to the other one
- Amdahl's law for each level of parallelism
- Using MPI-parallel application libraries? → Are they prepared for hybrid?
- Using thread-local application libraries? → Are they thread-safe?

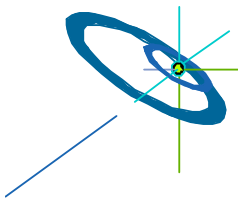
See, e.g., the necessary **–O4** flag with mpxlf_r on IBM Power6 systems



MPI-3 shared memory programming

- Pros
  - ISV and application libraries need not to be thread-safe
  - No additional OpenMP overhead
  - No OpenMP problems

- Cons
  - Library calls (MPI_WIN_ALLOCATE_SHARED) instead of SHARED / PRIVATE compiler directives
  - No work-sharing directives
    - **Loop scheduling must be programmed by hand**
  - No support for fine-grained or auto-balanced work-sharing
    - **As with OpenMP tasks, and dynamic or guided loop schedule**
  - Virtual addresses of a shared memory window may be different in each MPI process
    → no binary pointers
    → i.e., linked lists must be stored with offsets rather than pointers



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

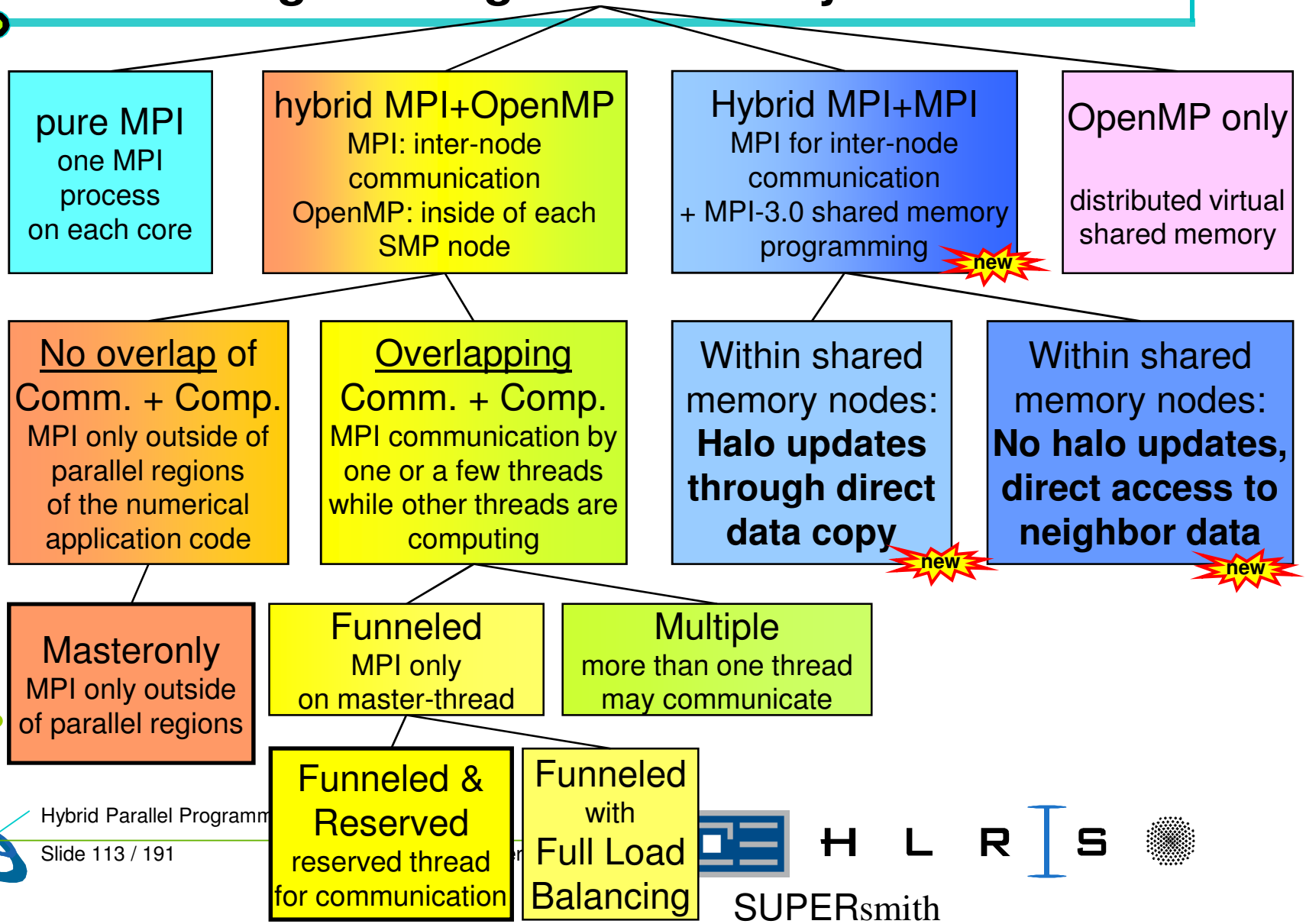
Not yet part of the OpenMP standard

- 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 */
```

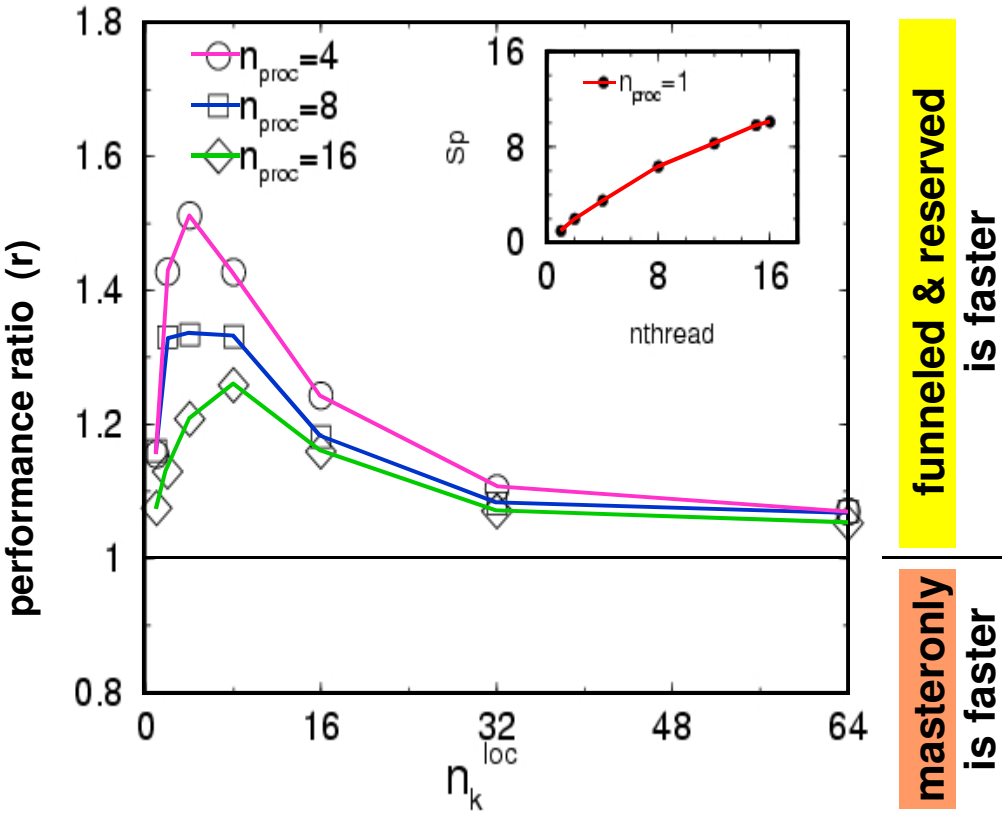
Parallel Programming Models on Hybrid Platforms



SUPERsmith

Experiment: Matrix-vector-multiply (MVM)

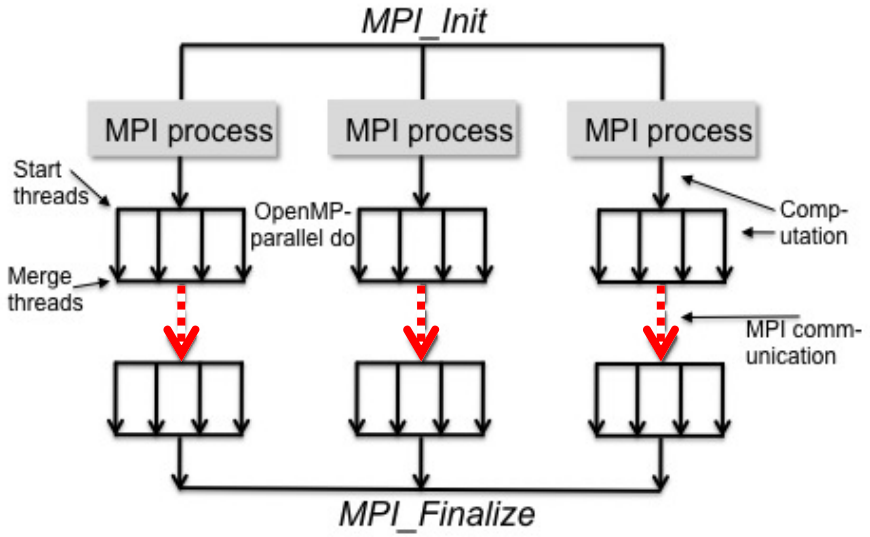
Masteronly
funneled & reserved

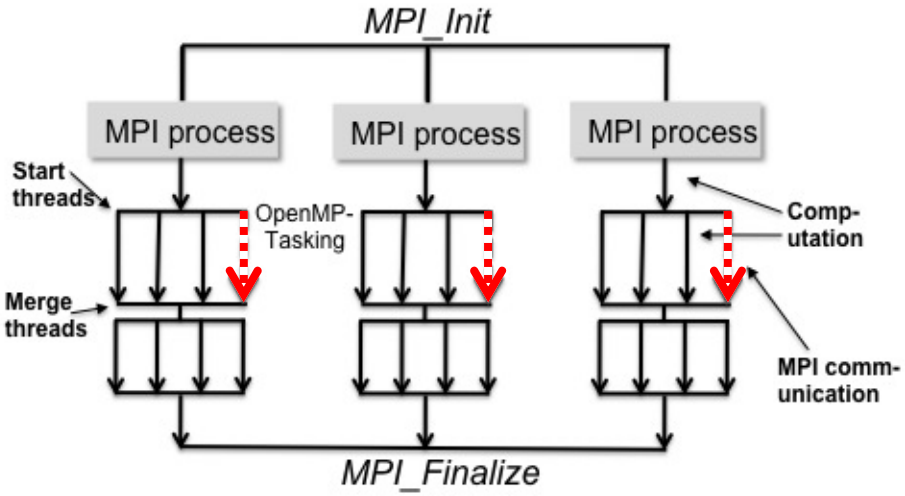


- Jacobi-Davidson-Solver on **IBM SP Power3** nodes with **16 CPUs per node**
- funneled&reserved is **always faster** in this experiments
- Reason: Memory bandwidth is already saturated by 15 CPUs, see inset
- Inset: Speedup on 1 SMP node using different number of threads

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

Overlapping: Using OpenMP tasks





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

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

Slides, courtesy of Alice Koniges, NERSC, LBNL

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
```

```
1  !reorder remaining particles: fill holes
     fill_hole(p_array);
23
3  !send number of particles to move right
25   MPI_SENDRECV(x,length=2,..);
5  !send to right and receive from left
27   MPI_SENDRECV(sendright,length=g(x),..);
7  !send number of particles to move left
29   MPI_SENDRECV(y,length=2,..);
9  !send to left and receive from right
31   MPI_SENDRECV(sendleft,length=g(y),..);
33
11 !adding shifted particles from right
35   !$omp parallel do
13   do m=1,x
37     p_array(h(m))=sendright(m);
15   enddo
   !adding shifted particles from left
39   !$omp parallel do
17   do n=1,y
41     p_array(h(n))=sendleft(n);
19   enddo
43
21 }
```

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

HLRS

SUPERsmith

skipped

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

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

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

Overlapping particle reordering

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

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

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

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

Overlapping remaining MPI_Sendrecv

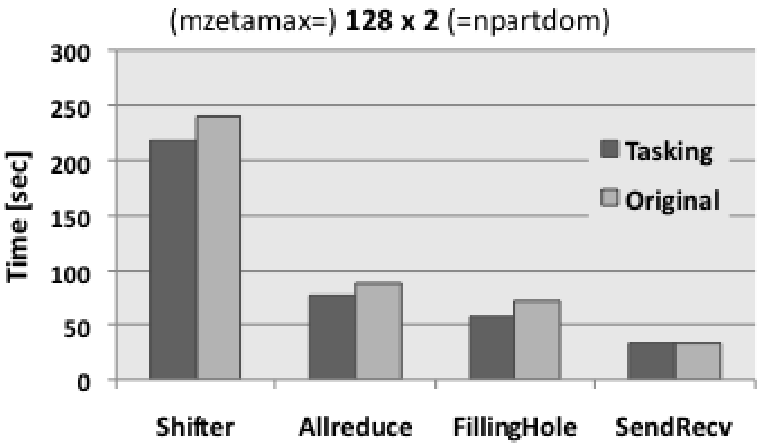
Slides, courtesy of Alice Koniges, NERSC, LBNL

HLRS SUPERsmith

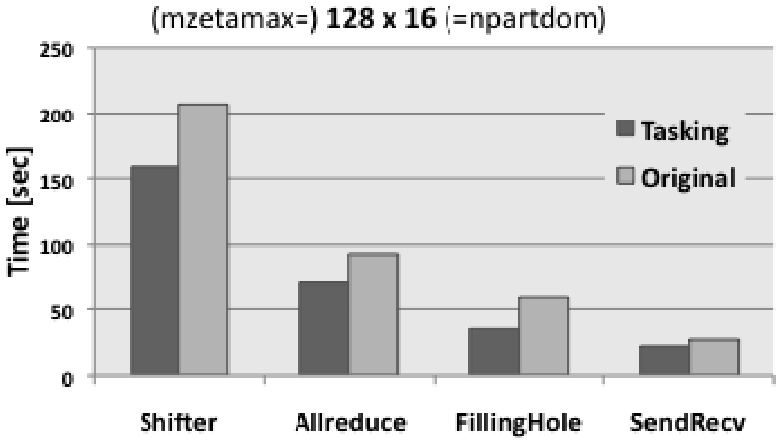
skipped

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.

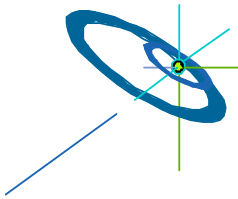
HLRS SUPERsmith

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



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:
  - 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

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
- Hybrid programming & accelerators
- 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 < $\frac{\text{Sum of workload of all zones}}{\text{Max workload of a zone}}$**
  - Inner loop:
    - **OpenMP parallelized (static schedule)**
    - **Not suitable for distributed memory parallelization**

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

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

- OpenMP enables
  - Cheap ***dynamic*** and ***guided*** load-balancing
  - Just a parallelization option (clause on omp for / do directive)
  - Without additional software effort
  - Without explicit data movement

```
#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

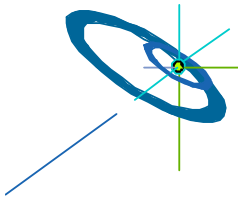
- **MPI & OpenMP**
  - Simple static load-balancing on MPI level, dynamic or guided on OpenMP level

} **medium quality**
cheap implementation

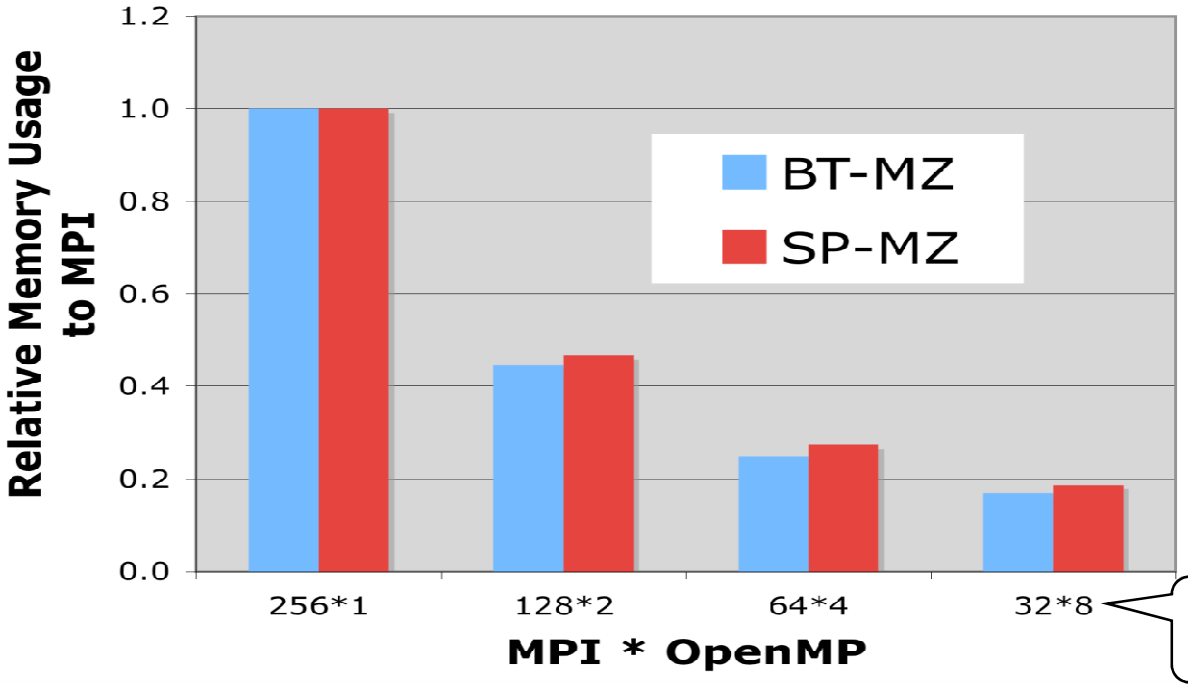
Memory consumption

- 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



Always same number of cores

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

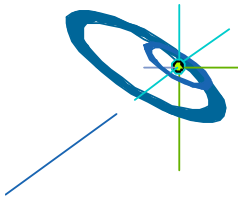
Hongzhang Shan, Haoqiang Jin, Karl Fuerlinger, Alice Koniges, Nicholas J. Wright: Analyzing the Effect of Different Programming Models Upon Performance and Memory Usage on Cray XT5 Platorms. Proceedings, CUG 2010, Edinburgh, GB, May 24-27, 2010.

SUPERsmith

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

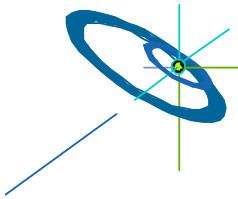


How many threads per MPI process?

- 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

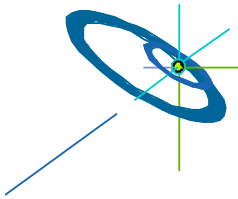


To overcome MPI scaling problems

- 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)

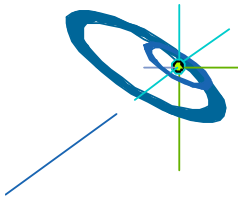
- Nested Parallelism
  → Outer loop with MPI / inner loop with OpenMP
- Load-Balancing
  → Using OpenMP ***dynamic*** and ***guided*** worksharing
- Memory consumption
  → Significantly reduction of replicated data on MPI level
- Opportunities, if MPI speedup is limited due to algorithmic problem
  →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



SUPERsmith

Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / pure MPI vs hybrid MPI+OpenMP
- Hybrid programming & accelerators
- 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:** **MPI processes may be multi-threaded,**
    (virtual value, **but only master thread will make MPI-calls**
    not part of the standard) **AND only while other threads are sleeping**
  - **MPI_THREAD_FUNNELED:** **Only master thread will make MPI-calls**
  - **MPI_THREAD_SERIALIZED:** **Multiple threads may make MPI-calls, but only one at a time**
  - **MPI_THREAD_MULTIPLE:** **Multiple threads may call MPI, with no restrictions**
- returned `provided` may be less than REQUIRED by the application

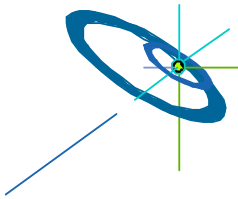
Calling MPI inside of OMP MASTER

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

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

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

#pragma omp barrier
```

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

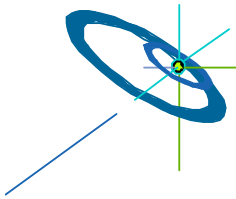


skipped

… the barrier is necessary – example with MPI_Recv

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

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

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

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

}
/* omp end parallel */
```



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”
MPIch2-1.1.0a**2**	ch3:nemesis (default) has `MPI_THREAD_MULTIPLE`
Intel MPI 3.1	Full `MPI_THREAD_MULTIPLE`
SciCortex MPI	`MPI_THREAD_FUNNELED`
HP MPI-2.2.7	Full `MPI_THREAD_MULTIPLE (with libmtmpi)`
SGI MPT-1.14	Not thread-safe?
IBM MPI	Full `MPI_THREAD_MULTIPLE`
Nec MPI/SX	`MPI_THREAD_SERIALIZED`

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

SUPERsmith

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`)

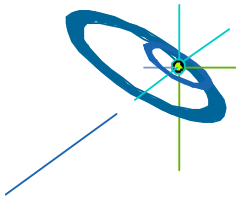
```
configure --enable-mpi-threads --enable-progress-threads
```

- 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

HLRS
SUPERsmith
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
- Hybrid programming & accelerators
- 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 must be used within mpirun / mpiexec

```
Application finished
```

ID	Short Description	Severity	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

SUPERsmith

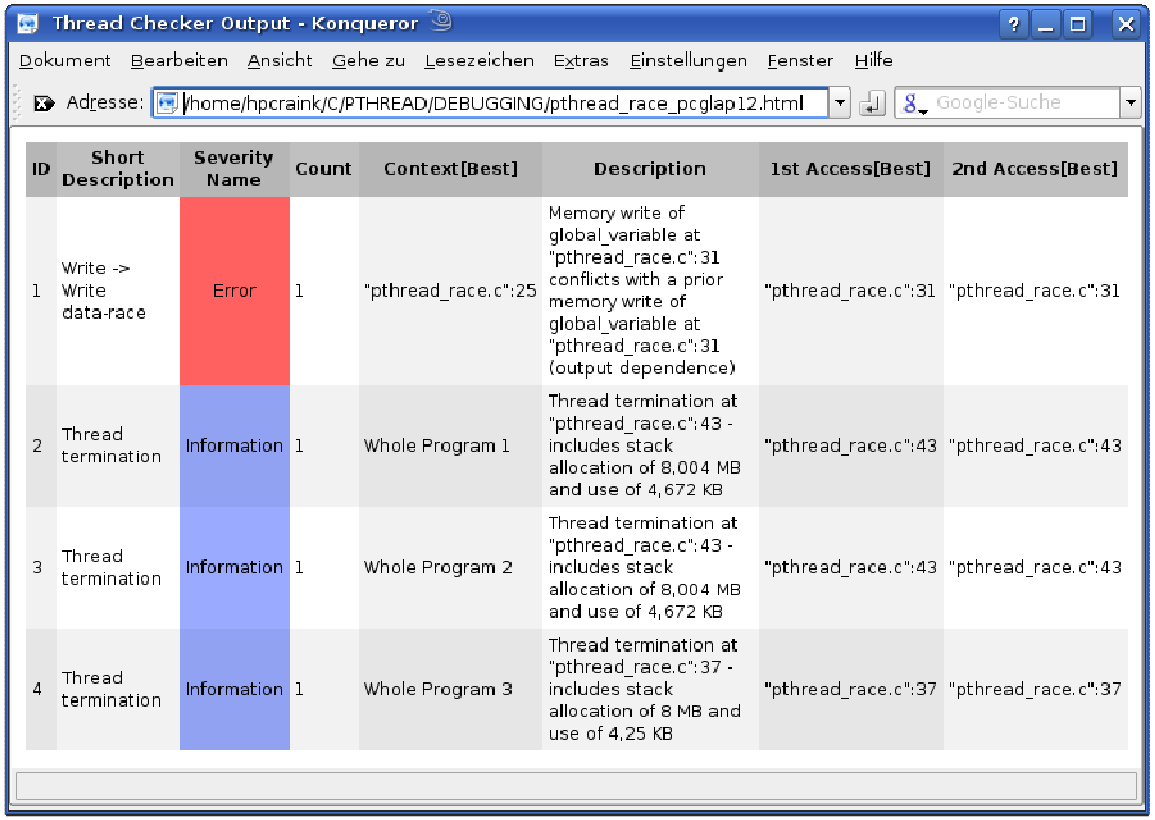
Courtesy of Rainer Keller, HLRS and ORNL

skipped

Thread Correctness – Intel ThreadChecker 2/3

- One may output to HTML:

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



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

HLRS

SUPERsmith

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'
```

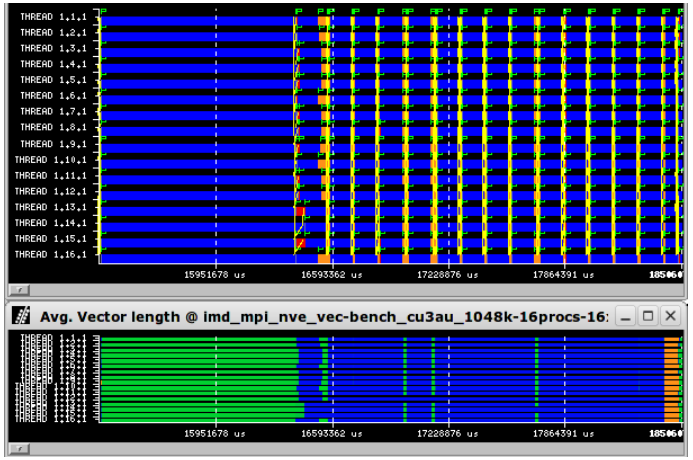
- Then run with:

```
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

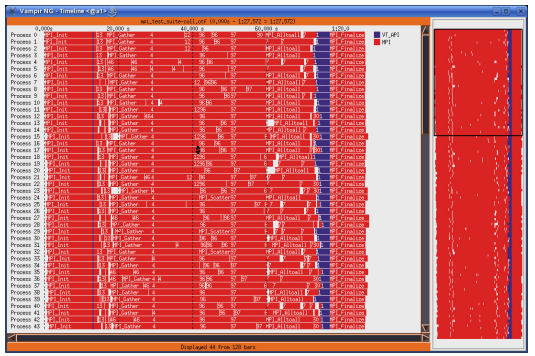
- Paraver examples have already been shown, tracing is done with linking against (closed-source) `omptrace` or `ompitrace`



- For Vampir/Vampirtrace performance analysis:

```
./configure -enable-omp
  -enable-hyb
  -with-mpi-dir=/opt/OpenMPI/1.3-icc
CC=icc F77=ifort FC=ifort
```

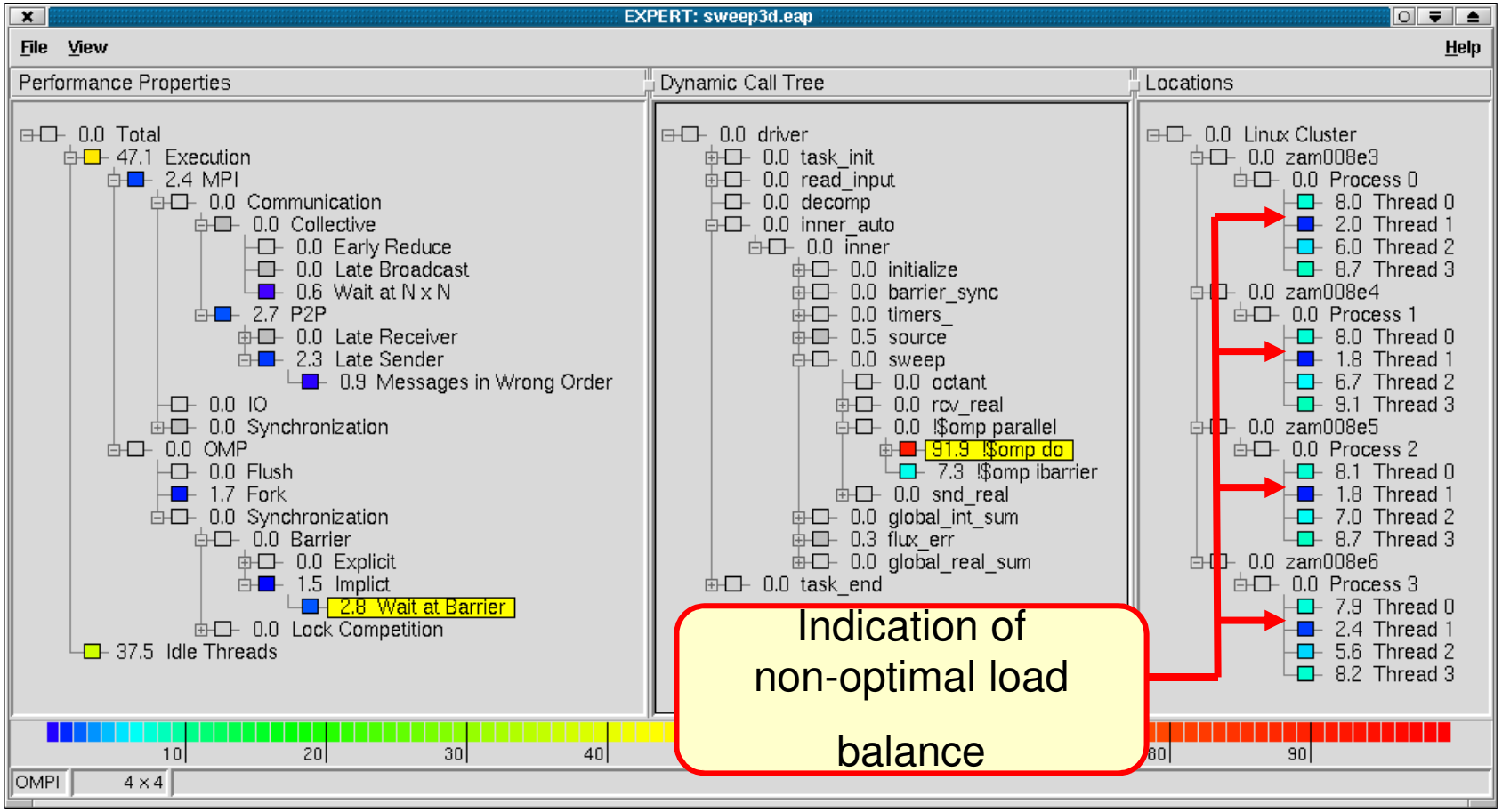
(Attention: does not wrap `MPI_Init_thread`!)



SUPERsmith

Courtesy of Rainer Keller, HLRS and ORNL

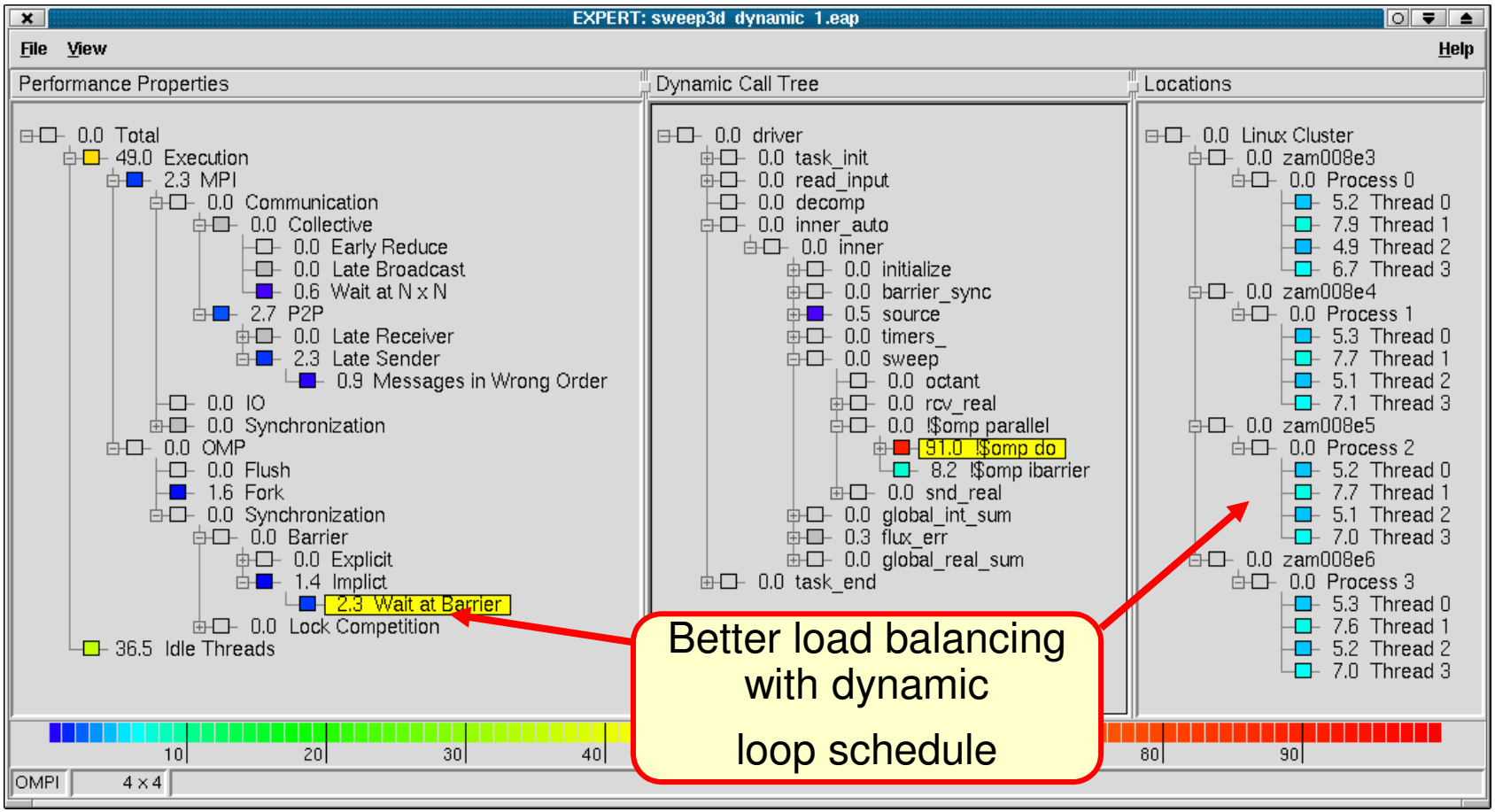
Scalasca – Example "Wait at Barrier"



SUPERsmith

Screenshots, courtesy of KOJAK JSC, FZ Jülich

Scalasca – Example "Wait at Barrier", Solution



SUPERsmith

Screenshots, courtesy of KOJAK JSC, FZ Jülich

Outline

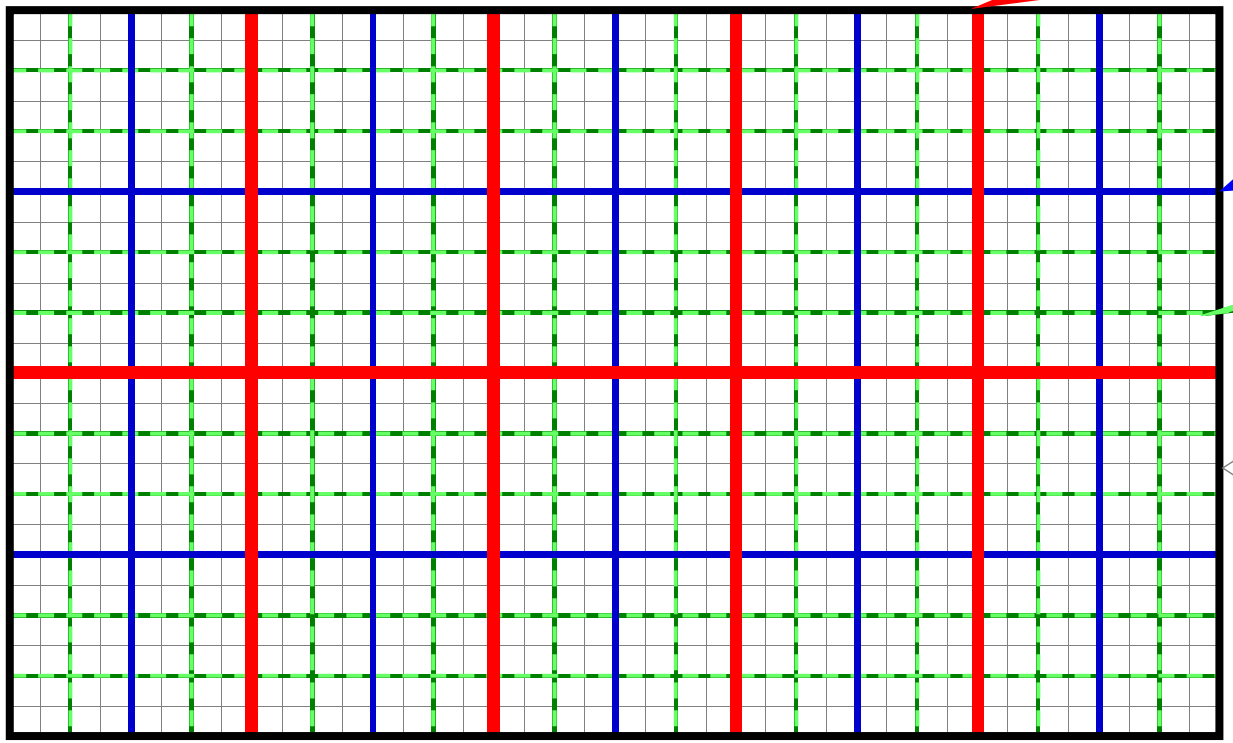
- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / pure MPI vs hybrid MPI+OpenMP
- Hybrid programming & accelerators
- Practical “How-To” on hybrid programming
- Mismatch Problems
- Opportunities:
  Application categories that can benefit from hybrid parallelization
- 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

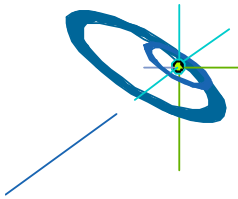
pure MPI
Hybrid MPI+MPI

How to achieve a hierarchical domain decomposition (DD)?

Implementation hints to previous slide

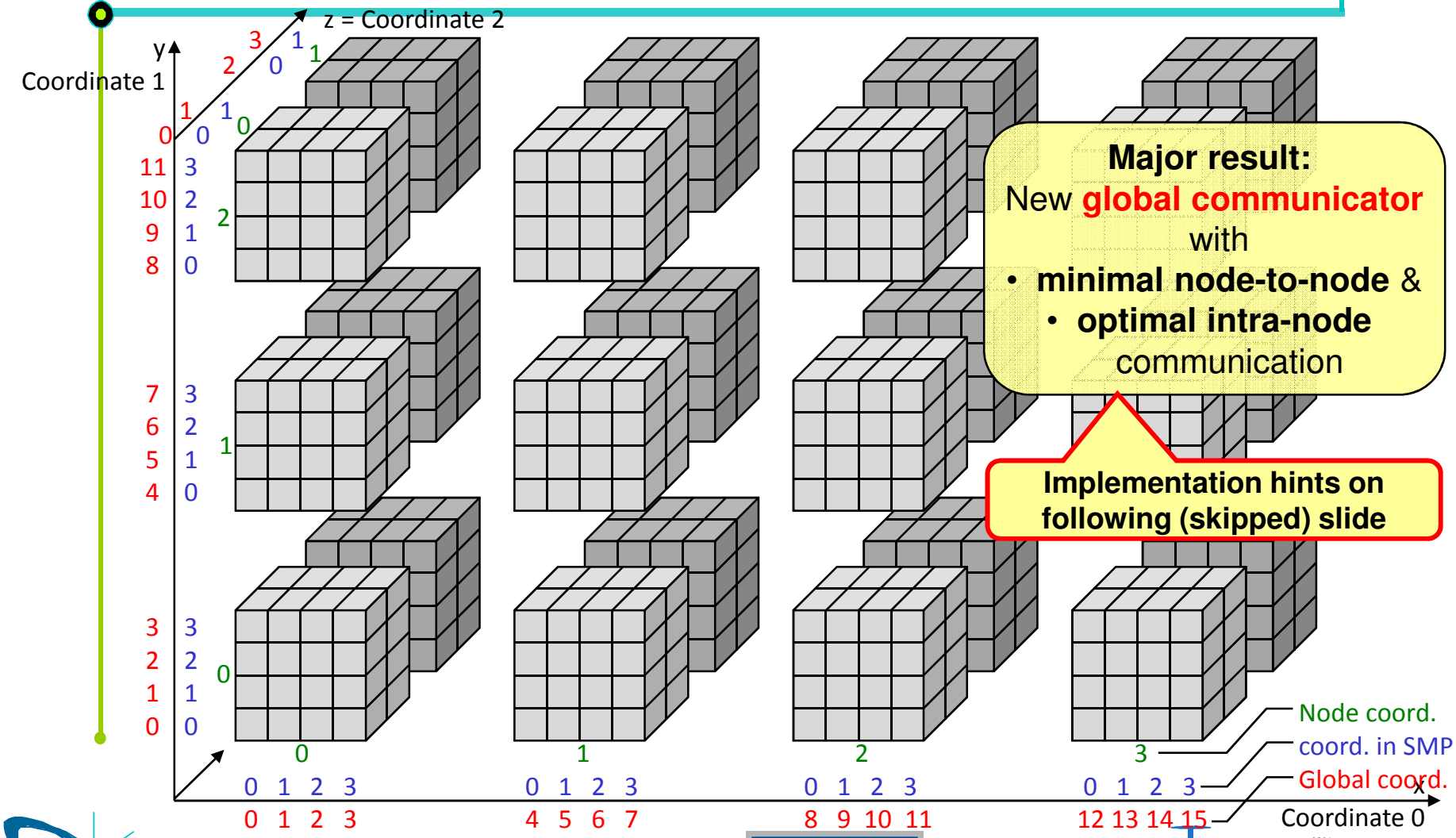
- **Cartesian grids:**
  - Several levels of subdivide
  - Ranking of MPI_COMM_WORLD – three choices:
    - **a) Sequential ranks through original data structure**
      + locating these ranks correctly on the hardware
      - ➢ can be achieved with one-level DD on finest grid
        + special startup (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



pure MPI
Hybrid MPI+MPI

Hierarchical Cartesian DD



Major result:
New **global communicator** with
- **minimal node-to-node** &
- **optimal intra-node** communication

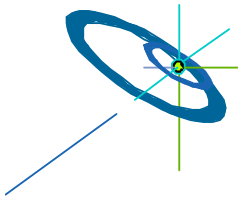
Implementation hints on following (skipped) slide

skipped

pure MPI
Hybrid MPI+MPI

Hierarchical Cartesian DD (Step 1)

```
// Input:  Original communicator:  MPI_Comm comm_orig;   (e.g. MPI_COMM_WORLD)
//         Number of dimensions:   int      ndims = 3;
//         Global periods:         int      periods_global[] = /*e.g.*/ {1,0,1};
MPI_Comm_size (comm_orig,  &size_global);
MPI_Comm_rank (comm_orig,  &myrank_orig);

// Establish a communicator on each SMP node:
MPI_Comm_split_type (comm_orig,  MPI_COMM_TYPE_SHARED,  0,  MPI_INFO_NULL,  &comm_smp_flat);
MPI_Comm_size (comm_smp_flat,  &size_smp);
int  dims_smp[] = {0,0,0};   int  periods_smp[] = {0,0,0} /*always non-period*/;
MPI_Dims_create (size_smp,  ndims,  dims_smp);
MPI_Cart_create (comm_smp_flat,  ndims,  dims_smp,  periods_smp, /*reorder=*/ 1,  &comm_smp_cart);
MPI_Comm_free  (&comm_smp_flat);
MPI_Comm_rank (comm_smp_cart,  &myrank_smp);
MPI_Cart_coords (comm_smp_cart,  myrank_smp,  ndims,  mycoords_smp);

// This source code requires that all SMP nodes have the same size. It is tested:
MPI_Allreduce (&size_smp,  &size_smp_min,   1,  MPI_INT,  MPI_MIN,  comm_orig);
MPI_Allreduce (&size_smp,  &size_smp_max,  1,  MPI_INT,  MPI_MAX,  comm_orig);
if (size_smp_min < size_smp_max)  { printf("non-equal SMP sizes\n");  MPI_Abort (comm_orig, 1); }
```



skipped

pure MPI
Hybrid MPI+MPI

Hierarchical Cartesian DD (Step 2)

```
// Establish the node rank. It is calculated based on the sequence of ranks in comm_orig
// in the processes with myrank_smp == 0:
MPI_Comm_split (comm_orig, myrank_smp, 0, &comm_nodes_flat);
// Result: comm_nodes_flat combines all processes with a given myrank_smp into a separate communicator.
// Caution: The node numbering within these comm_nodes-flat may be different.
// The following source code expands the numbering from comm_nodes_flat with myrank_smp == 0
// to all node-to-node communicators:
MPI_Comm_size (comm_nodes_flat,  &size_nodes);
int dims_nodes[] =  {0,0,0};   for (i=0; i<ndims; i++) periods_nodes[i] = periods_global[i];
MPI_Dims_create (size_nodes,  ndims,  dims_nodes);
if (myrank_smp==0) {
    MPI_Cart_create (comm_nodes_flat,  ndims,  dims_nodes,  periods_nodes, 1,  &comm_nodes_cart);
    MPI_Comm_rank (comm_nodes_cart,  &myrank_nodes);
    MPI_Comm_free  (&comm_nodes_cart); /*was needed only to calculate myrank_nodes*/
}
MPI_Comm_free (&comm_nodes_flat);
MPI_Bcast (&myrank_nodes, 1, MPI_INT, 0, comm_smp_cart);
MPI_Comm_split (comm_orig, myrank_smp, myrank_nodes, &comm_nodes_flat);
MPI_Cart_create (comm_nodes_flat,  ndims,  dims_nodes,  periods_nodes, 0,  &comm_nodes_cart);
MPI_Cart_coords (comm_nodes_cart,  myrank_nodes,  ndims,  mycoords_nodes);
MPI_Comm_free (&comm_nodes_flat);
```

Optimization according to inter-node network of the first processes in each SMP node

Copying it for the other processes in each SMP node

skipped

Hierarchica

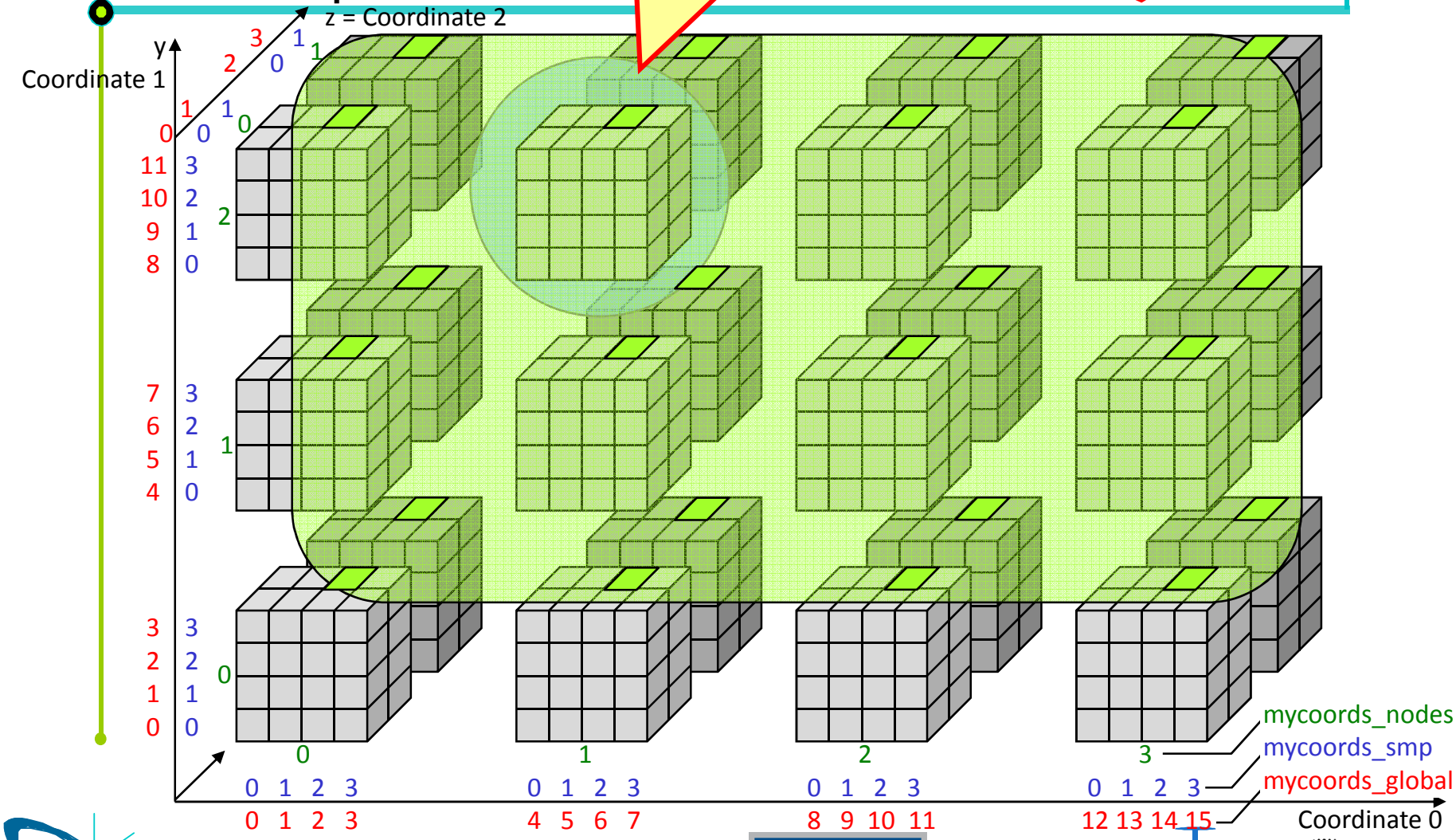
Result of Step2

e MPI

MPI+MPI

comm_smp_cart
for all processes with
coord_nodes== {1,2,0}

comm_nodes_cart
for all processes with
mycoord_smp== {2,3,1}



HLRS

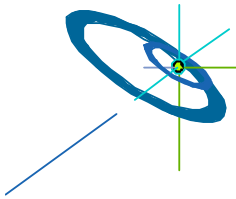
SUPERsmith

skipped

pure MPI
Hybrid MPI+MPI

Hierarchical Cartesian DD (Step 3)

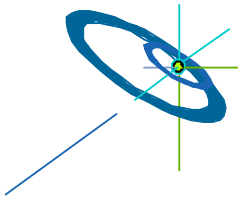
```
// Establish the global Cartesian communicator:
for (i=0; i<ndims; i++) { dims_global[i] = dims_smp[i] * dims_nodes[i];
   mycoords_global[i] = mycoords_nodes[i] * dims_smp[i] + mycoords_smp[i];
}
myrank_global = mycoords_global[0];
for (i=1; i<ndims; i++)  { myrank_global = myrank_global * dims_global[i] + mycoords_global[i]; }
MPI_Comm_split (comm_orig,  /*color*/ 0,  myrank_global,  &comm_global_flat);
MPI_Cart_create (comm_global_flat,  ndims,  dims_global,  periods_global,  0,  &comm_global_cart);
MPI_Comm_free (&comm_global_flat);

// Result:
//   Input was:
//      comm_orig,  ndims,  periods_global
//   Result is:
//      comm_smp_cart,     size_smp,     myrank_smp,     dims_smp,     periods_smp,     my_coords_smp,
//      comm_nodes_cart,   size_nodes,   myrank_nodes,   dims_nodes,   periods_nodes,   my_coords_nodes,
//      comm_global_cart,  size_global,  myrank_global,  dims_global,                   my_coords_global
```



How to achieve a hierarchical domain decomposition (DD)?

- **Unstructured grids:**
  - 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
  - Multi-level DD:
    - Top-down: Several levels of (Par)Metis
      → unbalanced communication
      → demonstrated on next (skipped) slide

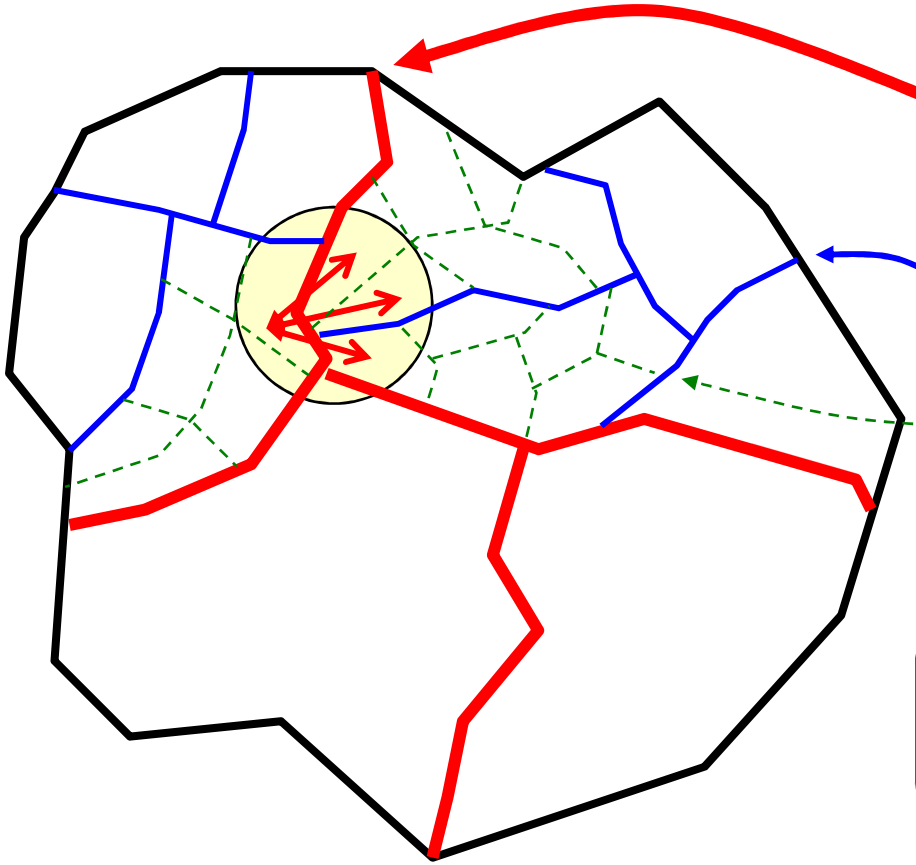
    - Bottom-up: Low level DD
      + higher level recombination
      → based on DD of the grid of subdomains



skipped

pure MPI
Hybrid MPI+MPI

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

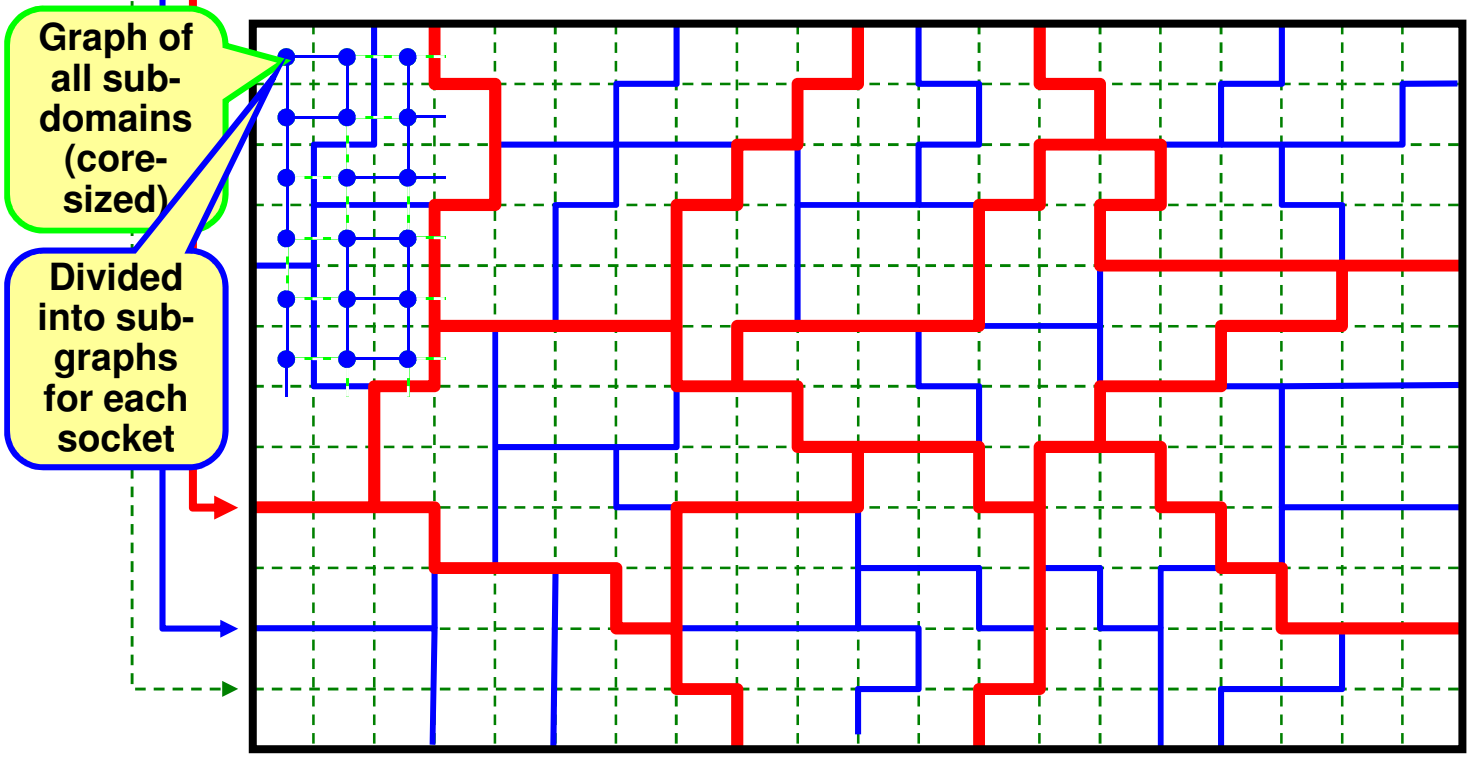
➢ Subdivide does not care on balancing of the outer boundary

➢ 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. **Numa-domain-level DD:** recombining of core-domains
3. **SMP node level DD:** recombining of socket-domains

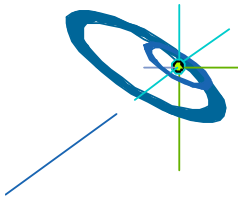


- **Problem:** Recombination must **not** calculate patches that are smaller or larger than the average
- In this example the load-balancer **must** combine always
  - 6 cores, and
  - 4 numa-domains (i.e., sockets or dies)
- **Advantage:** Communication is balanced!

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 …

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 with O(log N)**
      - From 1000 to 1000,000 process O(Log N) only doubles!

The vendors should deliver scalable MPI libraries for their largest systems!

Remarks on Cache Optimization

- **After** all parallelization domain decompositions (DD, up to 3 levels) are done:
- Cache-blocking is an 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 run from block to block
  - Inner loops inside of each block
  - Cartesian example: 3-dim loop is split into

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

Access to 13-point stencil

Remarks on Cost-Benefit Calculation

Costs

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

Benefit

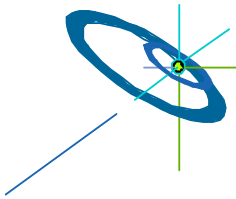
- from reduced CPU utilization
  - e.g., Example 1:
    100,000 € hardware costs of the cluster
    x 20% used by this application over whole lifetime of the cluster
    x 7% performance win through the optimization
    = 1,400 € → **total loss = 13,600 €**
  - e.g., Example 2:
    10 Mio € system x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

skipped

Remarks on MPI and PGAS (UPC & CAF)

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

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



skipped

Remarks on MPI and PGAS (UPC & CAF)

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

skipped

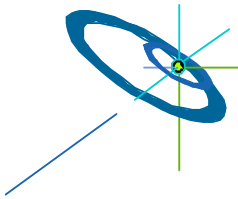
Remarks on MPI and PGAS (UPC & CAF)

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

skipped

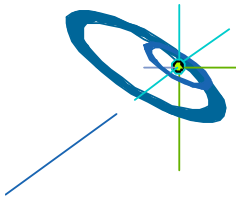
Remarks on MPI and PGAS (UPC & CAF)

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



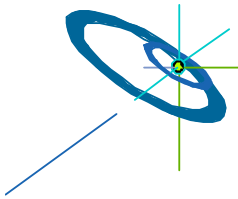
Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / pure MPI vs hybrid MPI+OpenMP
- Hybrid programming & accelerators
- 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

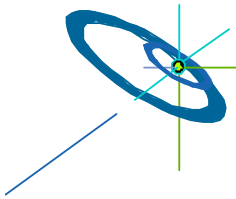
- We want to thank
  - Gerhard Wellein, RRZE
  - Alice Koniges, NERSC, LBNL
  - Rainer Keller, HLRS and ORNL
  - Jim Cownie, Intel
  - SCALASCA/KOJAK project at JSC, Research Center Jülich
  - HPCMO Program and the Engineer Research and Development Center Major Shared Resource Center, Vicksburg, MS (http://www.erdc.hpc.mil/index)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



Summary – Hybrid MPI+MPI

MPI + MPI-3 shared memory

- Two levels of parallelism
  - **Outer → distributed memory → halo data transfer → MPI**
  - **Inner → shared memory → halo transfer or direct access → MPI-3**
- New promising hybrid parallelization model
- No real experience up to now
- No OpenMP and thread-safety problems



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

Maybe the most important advantage!

  - 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 NUMA domain

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

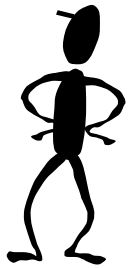
Summary – good and bad

- 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

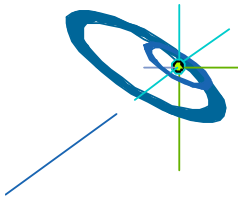
pure MPI

- \+ 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



OpenMP only

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



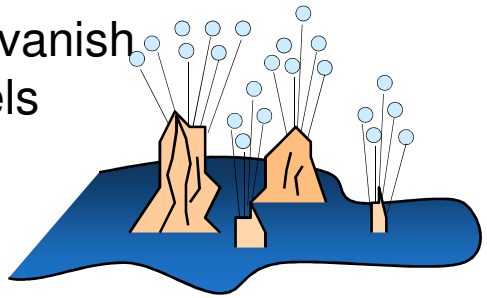
Summary

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

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

  → 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



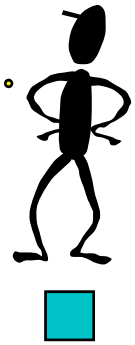
Conclusions

- 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**
- **MPI + MPI-3 → new promising alternative to MPI + OpenMP**
- Pure MPI → still on smaller cluster
- OpenMP → on large ccNUMA nodes
  (**not ClusterOpenMP**)

Thank you for your interest

Q & A

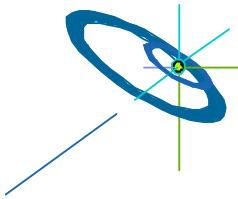
Please fill in the feedback sheet – Thank you



SUPERsmith

Appendix

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



Abstract

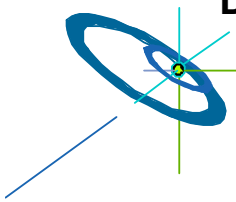
Half-Day Tutorial (Level: 25% Introductory, 50% Intermediate, 25% Advanced)

Authors. Rolf Rabenseifner, HLRS, University of Stuttgart, Germany
Georg Hager, University of Erlangen-Nuremberg, Germany
Gabriele Jost, Supersmith, Maximum Performance Software, 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. MPI-3.0 introduced a new shared memory programming interface, which can be combined with MPI message passing and remote memory access on the cluster interconnect. It can be used for direct neighbor accesses similar to OpenMP or for direct halo copies, and enables new hybrid programming models. These models are compared with various hybrid MPI+OpenMP approaches and pure MPI. This tutorial also includes a discussion on planned future OpenMP support for accelerators. Benchmark results on different platforms are presented. Numerous case studies demonstrate the performance-related aspects of hybrid programming, and application categories that can take advantage of this model 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/ISC2013-hybrid.html



H L R S

SUPERsmith

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, and in Jan. 2012, he was appointed as GCS' PATC director.

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. His textbook “Introduction to High Performance Computing for Scientists and Engineers” is recommended reading for many HPC-related courses and lectures worldwide. A full list of publications, talks, and other things 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 joined the Texas Advanced Computing Center (TACC), working on all sorts of exciting projects related to large scale parallel processing for scientific computing. In 2011, she joined Advanced Micro Devices (AMD) as a design engineer in the Systems Performance Optimization group.

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
- Torsten Hoefler, James Dinan, Darius Buntinas, Pavan Balaji, Brian Barrett, Ron Brightwell, William Gropp, Vivek Kale, Rajeev Thakur:
  MPI + MPI: a new hybrid approach to parallel programming with MPI plus shared memory.
  http://link.springer.com/content/pdf/10.1007%2Fs00607-013-0324-2.pdf

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