Hybrid MPI and OpenMP Parallel Programming

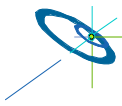
MPI + OpenMP and other models on clusters of SMP nodes

Rolf Rabenseifner[1)] Georg Hager[2)] Gabriele Jost[3)] Rainer Keller[1)]

Rabenseifner@hlrs.de Georg.Hager@rrze.uni-erlangen.de gjost@tacc.utexas.edu Keller@hlrs.de

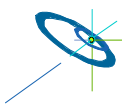
[1)] High Performance Computing Center (HLRS), University of Stuttgart, Germany
[2)] Regional Computing Center (RRZE), University of Erlangen, Germany
[3)] Texas Advanced Computing Center / Naval Postgraduate School, USA

Tutorial M09 at SUPERCOMPUTING 2008 (SC08)
Austin, Texas, USA, Nov. 17



H L R S

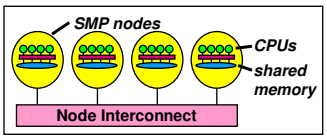
Outline



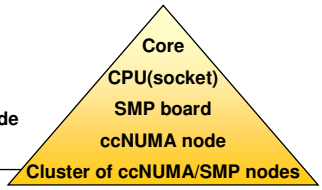
H L R S

Motivation

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

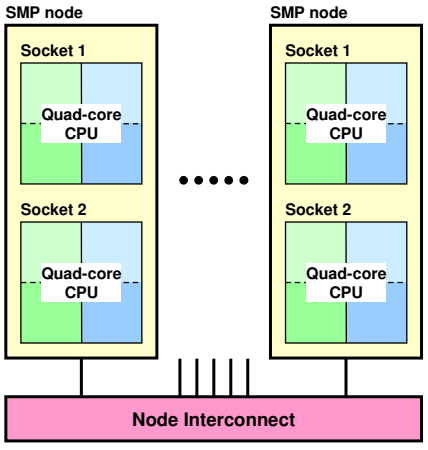


- Hardware range
  - **mini-cluster with dual-core CPUs**
  - **…**
  - **large constellations with large SMP nodes**
    … with several sockets (CPUs) per SMP node
    … with several cores per socket
  - **→ Hierarchical system layout**

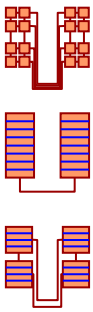


- Hybrid MPI/OpenMP programming seems natural
  - **MPI between the nodes**
  - **OpenMP inside of each SMP node**

Motivation



- Which programming model is fastest?
- MPI everywhere?
- Fully hybrid MPI & OpenMP?
- Something between? (Mixed model)

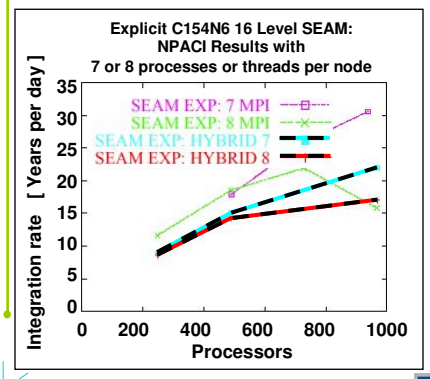


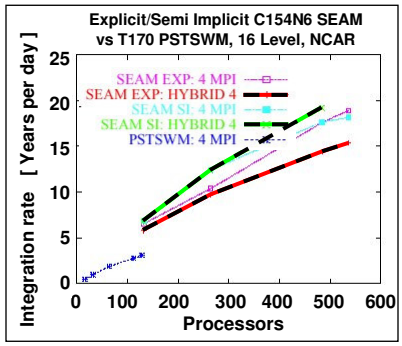
- Often hybrid programming **slower** than pure MPI
  - Examples, Reasons, …

?

Example from SC

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





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

Motivation

Minimizing

- Communication overhead,
  - **e.g., messages inside of one SMP node**
- Synchronization overhead
  - **e.g., OpenMP fork/join**
- Load imbalance
  - **e.g., using OpenMP *guided* worksharing schedule**
- Memory consumption
  - **e.g., replicated data in MPI parallelization**
- Computation overhead
  - **e.g., duplicated calculations in MPI parallelization**

Optimal parallel scaling

Goals of this tutorial

- Sensitize to problems on clusters of SMP nodes
  - see sections → Case studies
    → Mismatch problems
- Technical aspects of hybrid programming
  - see sections → Programming models on clusters
    → Examples on hybrid programming
- Opportunities with hybrid programming
  - see section → Opportunities: Application categories that can benefit from hybrid paralleliz.
- Issues and their Solutions
  - with sections → Thread-safety quality of MPI libraries
    → Tools for debugging and profiling for MPI+OpenMP

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

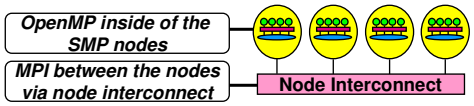
with your parallel program on clusters of SMP nodes

Outline

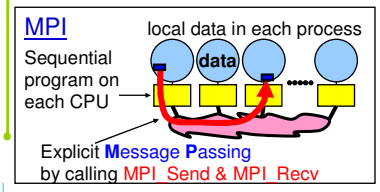
- Introduction / Motivation
- **Programming models on clusters of SMP nodes**
- Case Studies / pure MPI vs hybrid MPI+OpenMP
- Practical "How-To" on hybrid programming
- Mismatch Problems
- Opportunities:
  Application categories that can benefit from hybrid parallelization
- Thread-safety quality of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

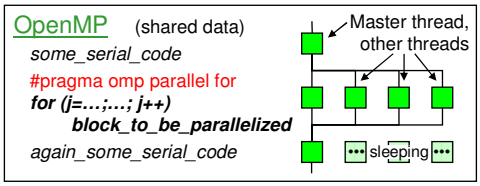
Major Programming models on hybrid systems

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

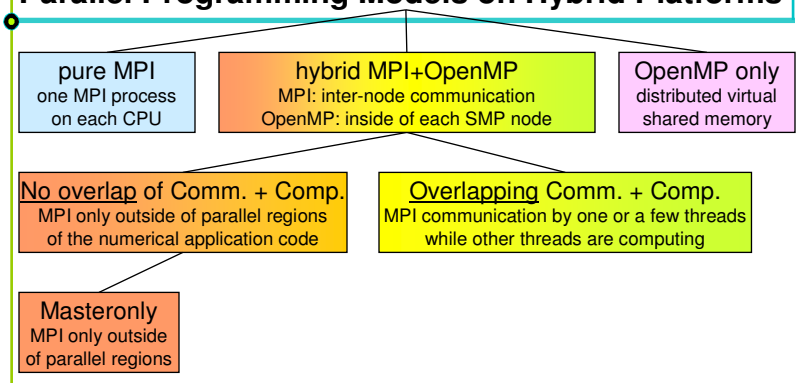


- Other: Virtual shared memory systems, PGAS, HPF, …
- Often **hybrid programming (MPI+OpenMP)** slower than **pure MPI**
  - why?





Parallel Programming Models on Hybrid Platforms



Pure MPI

pure MPI
one MPI process
on each CPU

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

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

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

Hybrid Masteronly

Masteronly
MPI only outside
of parallel regions

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

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

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

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

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

Overlapping Communication and Computation

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

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

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


Pure OpenMP (on the cluster)

OpenMP only
distributed virtual
shared memory

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

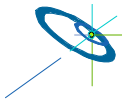
Experience:
→ **Mismatch** section

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

Outline

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

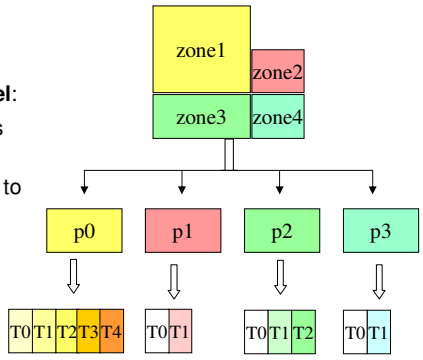
  Gabriele Jost (University of Texas,TACC/Naval Postgraduate School, Monterey CA)
- Practical "How-To" on hybrid programming
- Mismatch Problems
- Opportunities: Application categories that can benefit from hybrid paralleli.
- Thread-safety quality of MPI libraries
- Tools for debugging and profiling MPI+OpenMP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

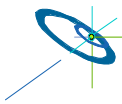


skipped

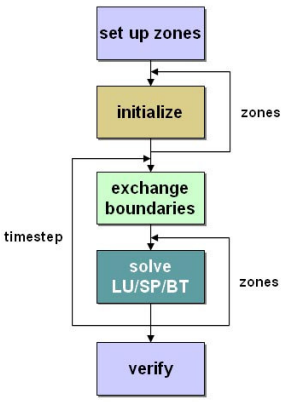
Why Multiple Levels of Parallelism?

- **Match hardware hierarchy**:
  - e.g. Clusters of SMP nodes
- **Limited parallelism on MPI level**
- **Unbalanced workload on MPI level**:
  - Assign more threads to process with high workload
  - Limit number of MPI processes to achieve better load-balance
- **Example: CFD Multi-zone codes**
  - Coarse grain parallelism between different zones
  - Fine grain loop-level parallelism in solver routines





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

Using MPI/OpenMP

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

call mpi_send/recv

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

end do
  ...
```

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

Benchmark Characteristics

- Aggregate sizes:
  - Class B: 304 x 208 x 17 grid points
  - Class C: 480 x 320 x 28 grid points
  - 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)**
  - #Zones: 64 (Class B), 256 (C), 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)**
  - #Zones: 16 (Class B, C, and D)
  - Size of the zones identical:
    - **no load-balancing required**
    - **limited parallelism on outer level**
- **SP-MZ: (Scalar Pentadiagonal simulated CFD application)**
  - #Zones: 64 (Class B), 256 (C), 1024 (D), 4096 (E)
  - Size of zones identical
    - **no load-balancing required**

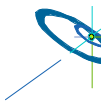
Expectations:

Pure MPI: Load-balancing problems!

Good candidate for MPI+OpenMP

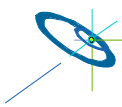
Limitted MPI Parallelism: → MPI+OpenMP increases Parallelism

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



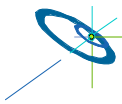
Benchmark Architectures

- Cluster of SMP Vector Nodes:
  - NEC SX8
- Linux Clusters:
  - NEC EMT64 (only in the handouts)
  - Cray XT4 (only in the handouts)
  - Sun Constellation



Hybrid code on cc-NUMA architectures

- **OpenMP**:
  - Support only per MPI process
  - Version 2.5 does not provide support to control to map threads onto CPUs. Support to specify thread affinities was under discussion for 3.0 but has not been included
- **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*
    → *http://www.halobates.de/numaapi3.pdf*



NEC SX8:MPI/OpenMP/Vectorization

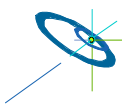
- Located at HLRS, Stuttgart, Germany
- 72 SX8 vector nodes with 8 CPUs each
- 12 TFlops peak performance
- Node-node interconnect IXS 16 GB/s per node
- Compilation:

  sxmpif90 –C hopt –P openmp
- Execute:

  export MPIMULTITASK=ON

  export OMP_NUM_THREADS=<#num threads pr MPI proc>

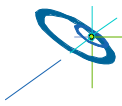
  mpirun –nn <#nodes> –nnp <#MPI procs per node> a.out
- Vectorization is required to achieve good performance
- A maximum of 64 nodes (512 CPUs) were used for the study



BT-MZ Cache Optimized Version

- **NPB 3.2 optimized for cache** based architectures with limited memory bandwidth
  - Use 1D temporary arrays to store intermediate values of 3d arrays
  - Decreases memory use but introduces data dependences

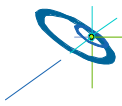
```
do zone = myzone_first, myzone_last
  ( MPI communication )
$OMP PARALLEL DO
do k
  do j
    do i        <-- non-vectorizable inner loop
      …
    rhs_1d(i) = c * rhs_1d(i-1) + …..
```



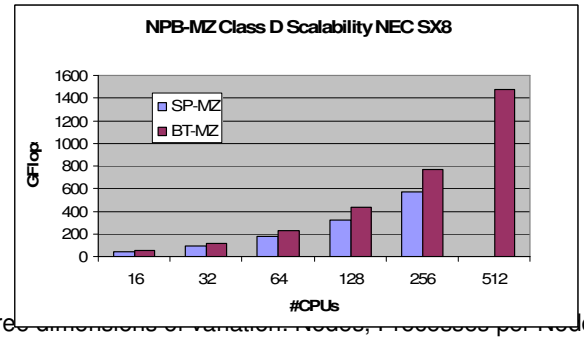
BT-MZ Vectorizable

- **SX8 requires vectorization:**
  - Re-introduce 3D arrays
  - Loop interchange to remove data dependence from inner loop
  - manual procedure in-lining to allow vectorization
  - Note: OpenMP directives within routines prevented automatic inlining

```
do zone = myzone_first, myzone_last
 ( MPI communication )

$OMP PARALLEL DO
do k
  do j          <-- Loop interchange yields vectorizable inner loop
    do i
     ….
    rhs_3d(i,j,k) = c * rhs_3d(i-1,j,k) + …..
```



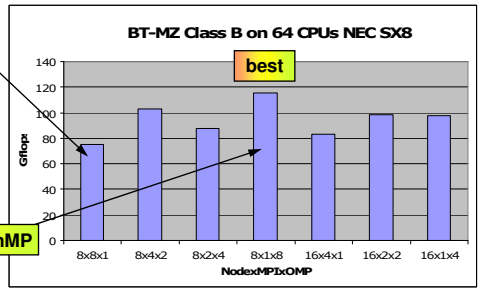
NPB-MZ Scalability on SX8



- Three dimensions of variation: Nodes, Processes per Node, Threads per Process
- Reported is the best performance for a given number of CPUs on a combination of Nodes x MPI x OMP
- SP-MZ performs best for pure MPI => Meets expectations

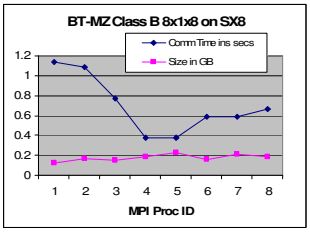
BT-MZ on SX-8: Combining MPI and OpenMP

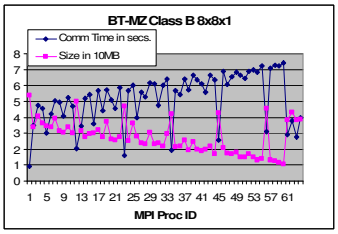
- **Metrics for MPI Procs Max/Min**
- **8x8x1:** 75 GFlops **pure MPI**
  - Total time: 8 sec
  - Workload size:59976 /2992
  - Vector length 75/12
  - Communication:
    - Time (sec): 6.4 /0.6
    - Count: 1608/ 1608
    - Size: 53 MB /38.6 MB
- **8x1x8:** 117 GFlops **hybrid MPI+OpenMP**
  - Total time: 5.2 sec
  - Workload size: 17035/16704
  - Vector length: 53/35
  - Communication:
    - Time (sec): 1.1 /0.4
    - Count: 13668 /8040
    - Size: 230 MB/120 MB



Impact of Combining MPI and OpenMP (2)

- The charts show communication time and size of communicated data per MPI process
- The time spent in communication is reciprocal to the size of data that is communicated
- The communication time is caused by load-imbalance





x86/x86-64 SSE vs SX8 Vectorization

- **SSE**
  - Vector length:
    - 2 (double prec)
    - 4 (single prec)
  - Vector memory load alignment must be 128 bit
  - Difficult for compiler to vectorize non-unit stride, SSE registers must be filled in piece-meal fashion
  - Increasingly important for new AMD and Intel chips with 128-bit-wide floating point pipeline
- **SX8 Vector Processor**
  - Vector length is 256
  - No special alignment requirement
  - Compiler to will vectorize non-unit stride, HW allows any stride on memory ops
  - Full vectorization is necessary to achieve good performance

skipped

NEC Xeon EM64 T Cluster

- **Located at HLRS, Stuttgart, Germany**
- **Peak Performance 2.5 Tflops, 400 Intel Xeon EM64T CPUs(3.2GHZ), 1GB Memory, Infiniband 1000 MB/s, Linux 2.6.9 ELsmp**
- **Compilation:**
  - Intel Compiler Version 9.0, flags used for the study: -openMP -O2
  - No SSE instructions were generated for vector version of the benchmarks, even after applying the "VECTOR ALWAYS" directive: Not considered efficient by compiler
  - Cache optimized benchmarks were used for the study
- **Execution:**

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

1 MPI per node:

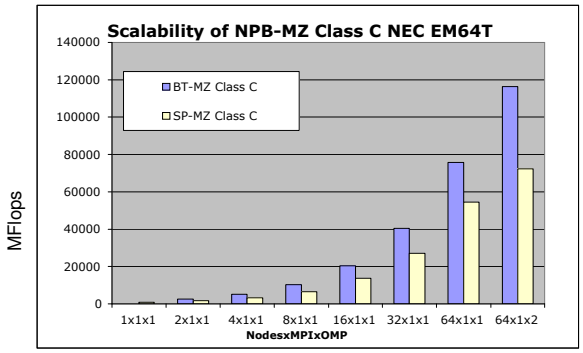
```
machines
cpu1
cpu2
…
cpun
```

2 MPI per node:

```
machines
cpu1
cpu1
cpu2
cpu2
…
cpun
cpun
```


skipped

NPB-MZ Class C on NEC EM64T Cluster

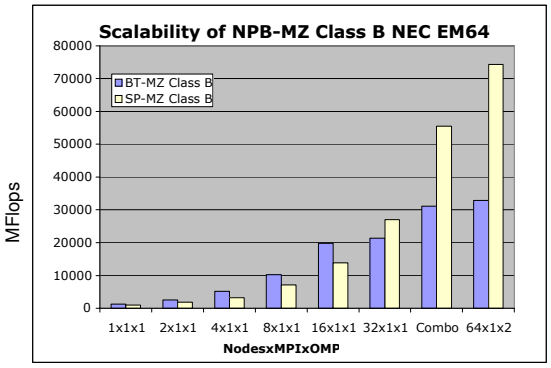


- Scalability in Mflops
- Reported is the best combination of Nodes x MPI x OMP
- Class C:
  - Up to 64 CPUs: Best performance with all MPI, 1 MPI process per node

skipped

NPB-MZ Class B on NEC EM64T Cluster

- Scalability in Mflops
- Reported is the best combination of Nodes x MPI x OMP
- Class B:
  - Performance of BT-MZ better than SP-MZ on small number of CPUs, but scalability drops early
  - BT-MZ Combo=32x1x2
  - SP-MZ Combo=64x1x1

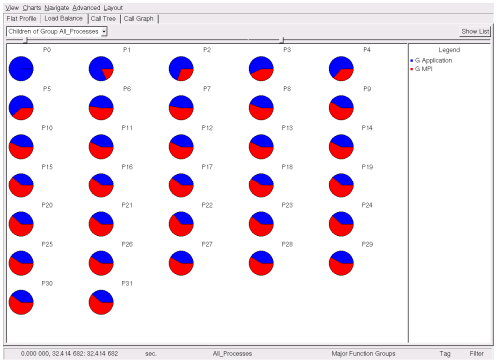


skipped

BT-MZ Class B on 32 MPI x 1 OpenMP

- Intel Trace Analyzer:
  - **Application**
  - **Communication**
- Load unbalanced on MPI level
- Large communication overhead

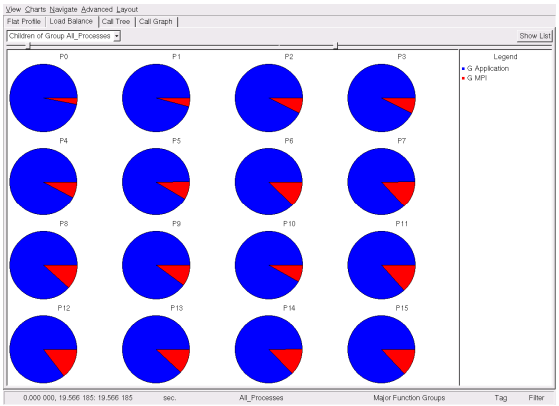
Most is MPI wait time due to unbalanced computation



skipped

BT-MZ Class B on 16 MPI x 2 OpenMP

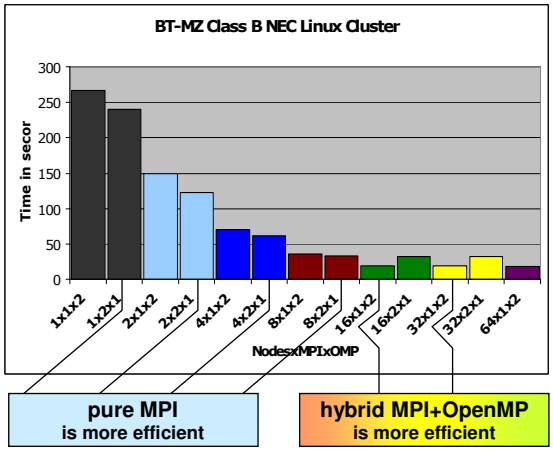
- Intel Trace Analyzer:
  - Application
  - Communication
- Load relatively well balanced on MPI level
- 10% of overall time in communication



skipped

BT-MZ Pure MPI vs MPI/OpenMP on NEC EMT64

- 2 MPI per node vs 1 MPI x 2 OpenMP threads
- Cross-over points when using OpenMP is advantageous
- BT-MZ Class B cross-over is at 32 CPUs

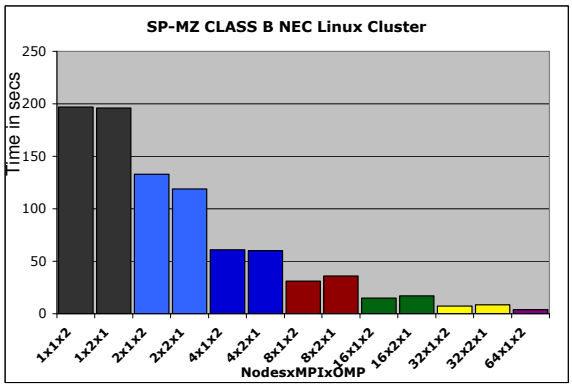


skipped

SP-MZ Pure MPI vs MPI/OpenMP on NEC EMT64

- 2 MPI per node vs 1 MPI x 2 OpenMP threads
- Cross-over points when using OpenMP is advantageous
- SP-MZ Class B cross over is at 16 CPUs, but difference is **marginal**

As expected! Pure MPI version is already load-balanced.



skipped

Performance aspects on NEC EM64T Cluster

- **BT-MZ load imbalance**:
  - 16 MPI x 2 OMP:
    - Max/Min size per thread: 34365/32793
  - 32 MPI x 1 OMP:
    - Max/Min size per thread: 59976/27319
- **SP-MZ no workload load imbalance**
  - Communication overhead for MPI

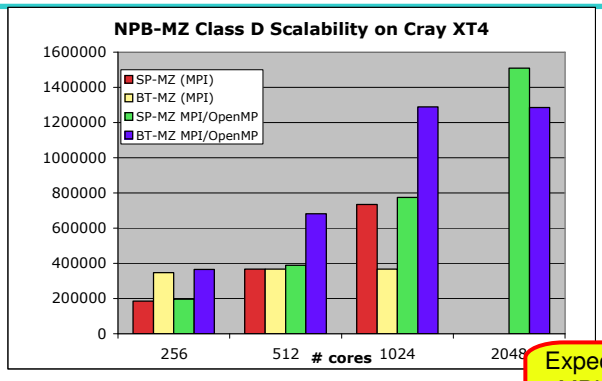
skipped

Cray XT4

- **2,152 AMD Opteron 64bit 2.1GHz quad-core processors (8608 core), 8 GBytes of dedicated memory**
- **Peak Performance 72.3 Tflops**
- **Nodes are connected via a HypeTransport link to a Cray SeaStar2 communication engine.**
- **Compilation:**
  - Cray ftn compiler based on PGI pgf90 7.1
  - Ftn -fastsse -tp barcelona-64 -r8 -mp=nonuma
  - Cache optimized benchmarks were used for the study
- **Execution:**
  - setenv OMP_NUM_THREAD NTHREAD
  - aprun -n NPROCS -N (1,2,4) bt-mz.exe
  - -N specifies to run 1, 2, or 4 MPI processes
- **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)**

skipped

NPB-MZ Class D Scalability on Cray XT4



SP-MZ pure MPI scales up to 1024 cores
SP-MZ MPI/OpenMP scales to 2048 cores
SP-MZ MPI/OpenMP outperforms pure MPI
BT-MZ MPI does not scale
BT-MZ MPI/OpenMP scales to 1024 cores, outperforms pure MPI

Expected: #MPI Processes limitted

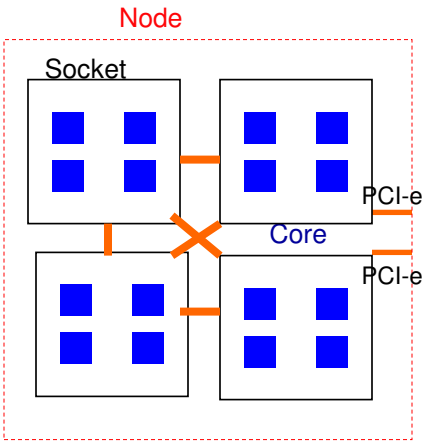
Unexpected!

Sun Constellation Cluster Ranger (1)

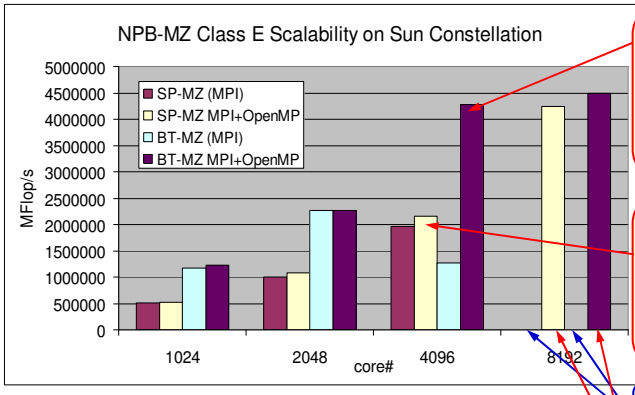
- Located at the Texas Advanced Computing Center (TACC), University of Texas at Austin (http://www.tacc.utexas.edu)
- 3936 Sun Blades, 4 AMD Quad-core 64bit 2.3GHz processors per node (blade), 62976 cores total
- 123TB aggregrate memory
- Peak Performance 579 Tflops
- InfiniBand Switch interconnect
-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

Sun Constellation Cluster Ranger (2)

- **Compilation:**
  - PGI pgf90 7.1
  - mpif90 -tp barcelona-64 -r8
- **Cache optimized benchmarks Execution:**
  - MPI MVAPICH
  - setenv OMP_NUM_THREAD NTHREAD
  - Ibrun numactl bt-mz.exe
- **numactl** controls
  - Socket affinity: select sockets to run
  - Core affinity: select cores within socket
  - Memory policy:where to allocate memory
  - http://www.halobates.de/numaapi3.pdf



NPB-MZ Class E Scalability on Ranger



BT
Significant improvement (235%):
Load-balancing issues solved with MPI+OpenMP

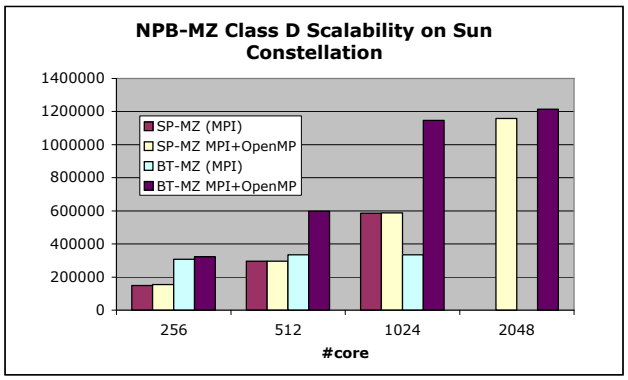
SP
Pure MPI is already load-balanced.
But hybrid programming 9.6% faster

Cannot be build for 8192 processes!

Hybrid:
SP: still scales
BT: does not scale

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

NPB-MZ Class D Scalability on Ranger



- SP-MZ hybrid outperforms SP-MZ pure MPI for Class D, Class E =>
- Does not meet expectations!

SP-MZ:Hybrid vs Pure MPI

•**Performance metrics for Class D:**

•**64x4:**

–Workload: HW FP OPS:91G x 4 per MPI Process

–Communication:

•Time (sec): 3.4sec max

•Count: 4531 isend per MPI Process

•Size: 802MB per MPI Process

•Total size: ~51328MB

•**256x1:**

–Workload: HW FP OPS: 91G per MPI Process

–Communication:

•Time (sec):17 sec Max

•Count: 2004 isend per MPI Process

•Size: 436 MB Max, 236MB Min

•Total Size: ~110000MB.

mpi_waitall

Imbalance

•**Performance issues for pure MPI:**

•Large amount of data communicated (2 x hybrid)

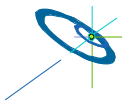
•Imbalance in message size across processes

Conclusions:

- **BT-MZ:**
  - Inherent workload imbalance on MPI level
  - #nprocs = #nzones yields poor performance
  - #nprocs < #zones => room for better workload balance, but decrases parallelism
  - Hybrid MPI/OpenMP offers possibility for load-balancing while maintaining amount of parallelism
  - Best performance in hybrid mode across all platforms
- SP-MZ:
  - No workload imbalance on MPI level
  - Pure MPI should perform best
  - Surprising results on some platforms due to unexpected zone-assignment inherent in benchmark

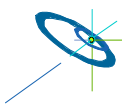
Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / pure MPI vs hybrid MPI+OpenMP
- **Practical "How-To" on hybrid programming**
  Georg Hager, Regionales Rechenzentrum Erlangen (RRZE)
- Mismatch Problems
- Application categories that can benefit from hybrid parallelization
- Thread-safety quality of MPI libraries
- Tools for debugging and profiling MPI+OpenMP
- Summary



Hybrid Programming How-To: Overview

- A practical introduction to hybrid programming
  - How to compile and link
  - Getting a hybrid program to run on a cluster
- Running hybrid programs efficiently on multi-core clusters
  - Affinity issues
    - **ccNUMA**
    - **Bandwidth bottlenecks**
  - Intra-node MPI/OpenMP anisotropy
    - **MPI communication characteristics**
    - **OpenMP loop startup overhead**
  - Thread/process binding



How to compile, link and run

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

Some examples for compilation and execution (1)

- **NEC SX8**
  - NEC SX8 compiler
  - `mpif90 -C hopt -P openmp … # -ftrace for profiling info`
  - Execution:

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

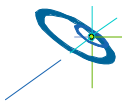
- **Standard Intel Xeon cluster (e.g. @HLRS):**
  - Intel Compiler
  - `mpif90 -openmp …`
  - Execution (**handling of OMP_NUM_THREADS, see next slide**):

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


Some examples for compilation and execution (2)

Handling of OMP_NUM_THREADS

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

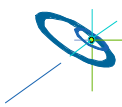


Some examples for compilation and execution (3)

Handling of OMP_NUM_THREADS (continued)

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

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



Some examples for compilation and execution (4)

- **Sun Constellation Cluster:**
  - `mpif90 -fastsse -tp barcelona-64 -mp …`
  - SGE Batch System
  - `setenv OMP_NUM_THREADS`
  - `ibrun numactl.sh a.out`
  - Details see TACC Ranger User Guide (www.tacc.utexas.edu/services/userguides/ranger/#numactl)
- **Cray XT4:**
  - `ftn -fastsse -tp barcelona-64 -mp=nonuma …`
  - `aprun -n nprocs -N nprocs_per_node a.out`

Interlude: Advantages of mpiexec

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

Running the code

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

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

- Same but 2 MPI processes per node:

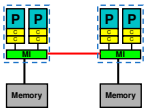
```
$ export OMP_NUM_THREADS=2
$ mpiexec -npernode 2 ./a.out
```

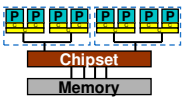
- Pure MPI:

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


Running the code *efficiently?*

- Symmetric, UMA-type compute nodes have become rare animals
  - NEC SX
  - Intel 1-socket (“Port Townsend/Melstone”) – see case studies
  - Hitachi SR8000, IBM SP2, single-core multi-socket Intel Xeon… (all dead)
- Instead, systems have become “non-isotropic” on the node level
  - ccNUMA (AMD Opteron, SGI Altix, IBM Power6 (p575), larger Sun Enterprise systems, Intel Nehalem)
  - Multi-core, multi-socket
    - **Shared vs. separate caches**
    - **Multi-chip vs. single-chip**
    - **Separate/shared buses**



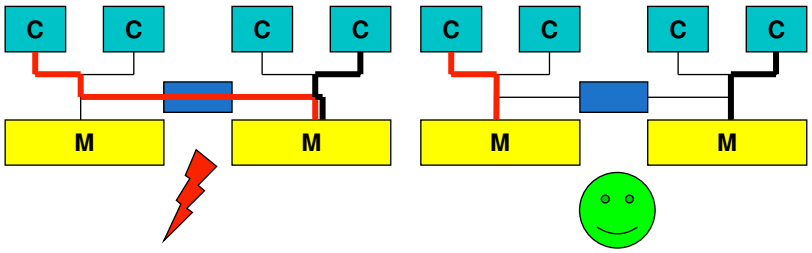


Issues for running code efficiently on "non-isotropic" nodes

- ccNUMA locality effects
  - Penalties for inter-LD access
  - Impact of contention
  - Consequences of file I/O for page placement
  - Placement of MPI buffers
- Multi-core / multi-socket anisotropy effects
  - Bandwidth bottlenecks, shared caches
  - Intra-node MPI performance
    - **Core ↔ core vs. socket ↔ socket**
  - OpenMP loop overhead depends on mutual position of threads in team

A short introduction to ccNUMA

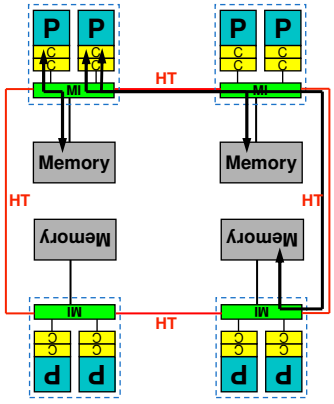
- ccNUMA:
  - **whole memory is transparently accessible by all processors**
  - **but physically distributed**
  - **with varying bandwidth and latency**
  - **and potential contention (shared memory paths)**



Example: HP DL585 G5

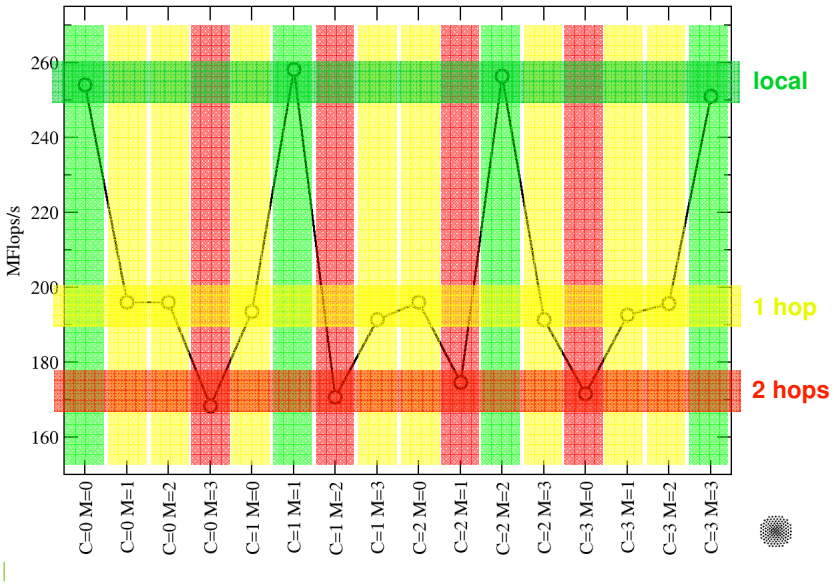
4-socket ccNUMA Opteron 8220 Server

- CPU
  - **64 kB L1 per core**
  - **1 MB L2 per core**
  - **No shared caches**
  - **On-chip memory controller (MI)**
  - **10.6 GB/s local memory bandwidth**
- HyperTransport 1000 network
  - **4 GB/s per link per direction**
- 3 distance categories for core-to-memory connections:
  - **same LD**
  - **1 hop**
  - **2 hops**
- Q1: What are the real penalties for non-local accesses?
- Q2: What is the impact of contention?



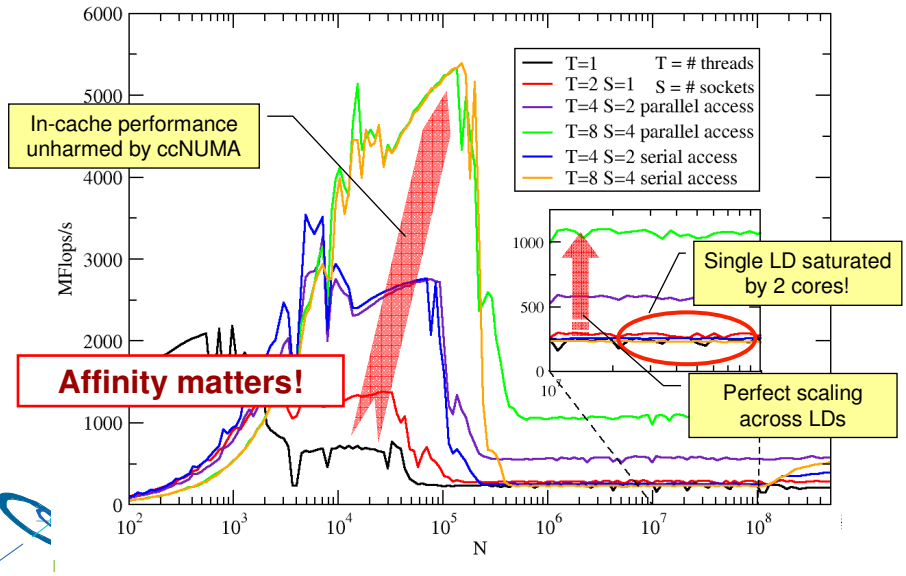
Effect of non-local access on HP DL585 G5:

Serial vector triad `A(:)=B(:)+C(:)*D(:)`



Contention vs. parallel access on HP DL585 G5:

OpenMP vector triad `A(:)=B(:)+C(:)*D(:)`

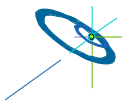


ccNUMA Memory Locality Problems

- 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

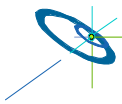


Solving Memory Locality Problems: First Touch

- "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:

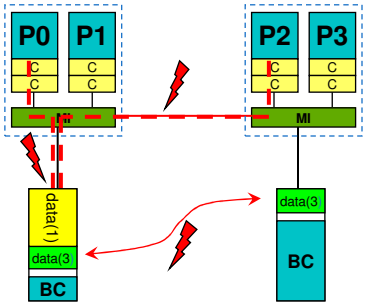
```
double *huge = (double*)malloc(N*sizeof(double));
// memory not mapped yet
for(i=0; i<N; i++) // or i+=PAGE_SIZE
   huge[i] = 0.0;  // mapping takes place here!
```

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



ccNUMA problems beyond first touch

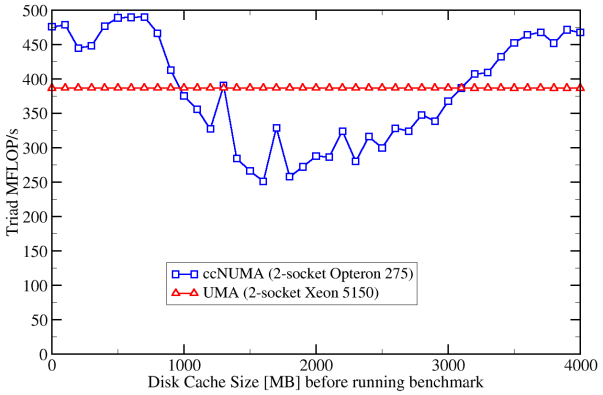
- OS uses part of main memory for disk buffer (FS) cache
  - If FS cache fills part of memory, apps will probably allocate from foreign domains
  - → non-local access!
  - Locality problem even on hybrid and pure MPI with "asymmetric" file I/O, i.e. if not all MPI processes perform I/O



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

ccNUMA problems beyond first touch

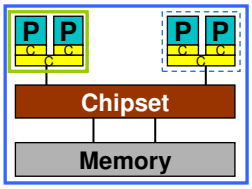
- Real-world example: ccNUMA vs. UMA and the Linux buffer cache
- Compare two 4-way systems: AMD Opteron ccNUMA vs. Intel UMA, 4 GB main memory
- Run 4 concurrent triads (512 MB each) after writing a large file
- Report performance vs. file size
- Drop FS cache after each data point



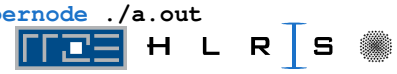
Intra-node MPI characteristics: IMB Ping-Pong benchmark

- Code (to be run on 2 processors):

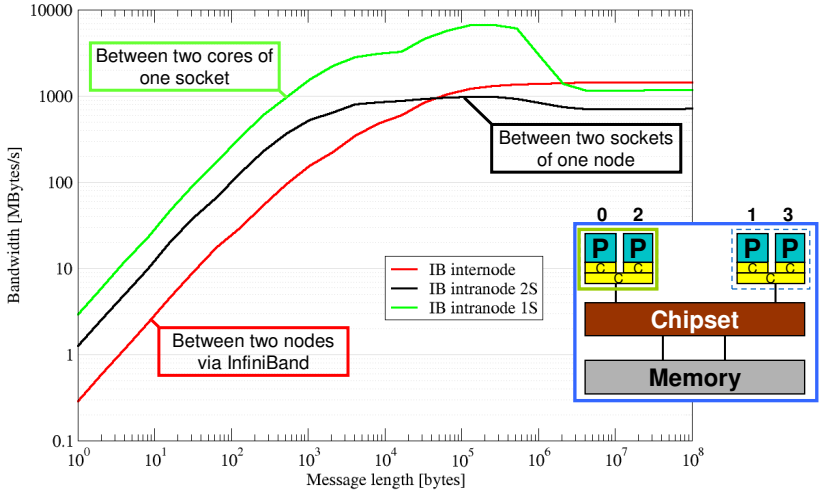
```
wc = MPI_WTIME()
do i=1,NREPEAT
  if(rank.eq.0) then
   MPI_SEND(buffer,N,MPI_BYTE,1,0,MPI_COMM_WORLD,ierr)
   MPI_RECV(buffer,N,MPI_BYTE,1,0,MPI_COMM_WORLD, &
                status,ierr)
  else
   MPI_RECV(…)
   MPI_SEND(…)
  endif
enddo
wc = MPI_WTIME() - wc
```



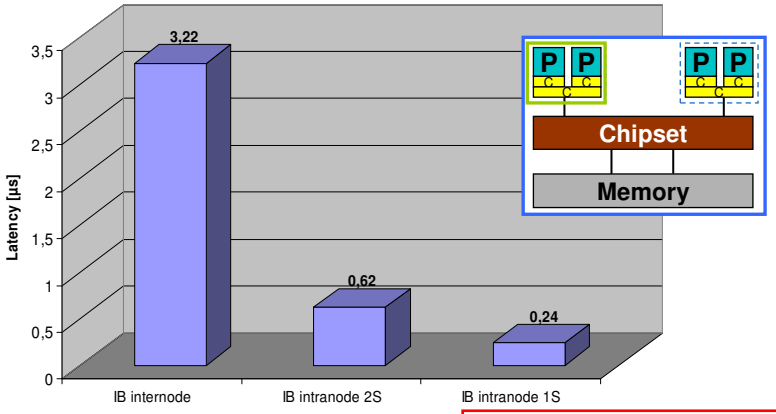
- Intranode (1S): `mpirun -np 2 -pin "1 3" ./a.out`
- Intranode (2S): `mpirun -np 2 -pin "2 3" ./a.out`
- Internode: `mpirun -np 2 -pernode ./a.out`



IMB Ping-Pong on DDR-IB Woodcrest cluster: log-log plot



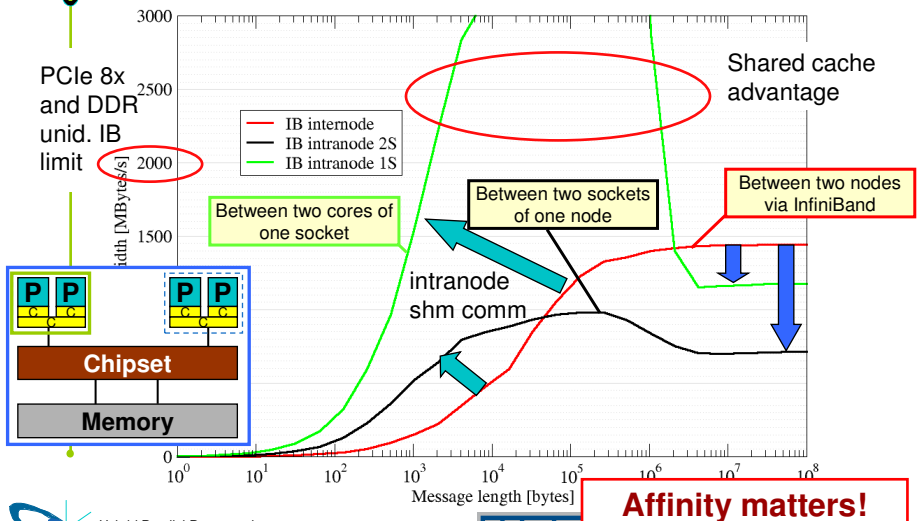
IMB Ping-Pong: Latency



Affinity matters!

IMB Ping-Pong: Bandwidth Characteristics

Intra-node vs. Inter-node



Affinity matters!

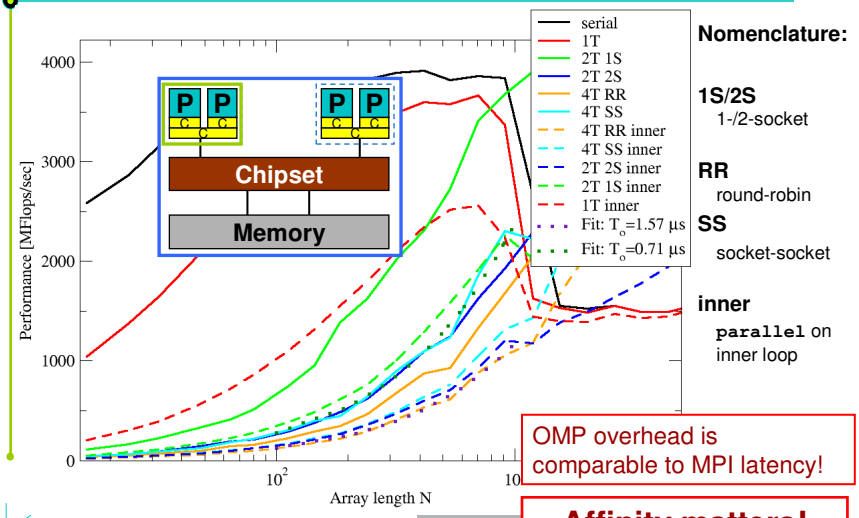
OpenMP Overhead

- As with intra-node MPI, OpenMP loop start overhead varies with the mutual position of threads in a team
- Possible variations
  - Intra-socket vs. inter-socket
  - Different overhead for "parallel for" vs. plain "for"
  - If one multi-threaded MPI process spans multiple sockets,
    - **… are neighboring threads on neighboring cores?**
    - **… or are threads distributed "round-robin" across cores?**
- Test benchmark: **Vector triad**

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

Look at performance for small array sizes!

OpenMP overhead



Nomenclature:

1S/2S
1-/2-socket

RR
round-robin

SS
socket-socket

inner
`parallel` on inner loop

OMP overhead is comparable to MPI latency!

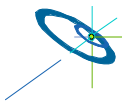
Affinity matters!

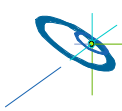
Thread/Process Affinity ("Pinning")

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

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

Widely usable example: Using PLPA under Linux!





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

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

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

Pinning available?

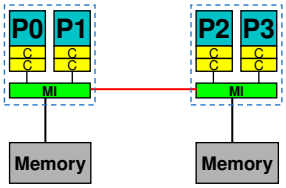
Care about correct core numbering! 0…N-1 is not always contiguous! If required, reorder by a map:
`cpu = map[cpu];`

Which CPU to run on?

Pin "me"

Process/Thread Binding With PLPA

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



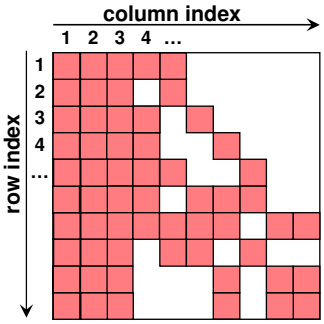

```
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
int mask = (rank % 4);
PLPA_CPU_SET(mask,&msk);
PLPA_NAME(sched_setaffinity)((pid_t)0,
                             sizeof(cpu_set_t), &msk);
```

- Hybrid case:

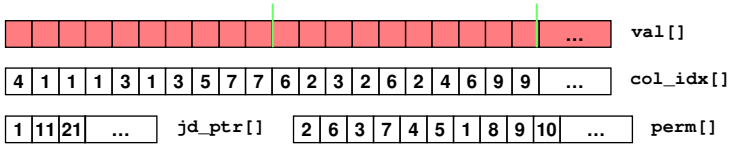
```
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
#pragma omp parallel
  {
    plpa_cpu_set_t msk;
    PLPA_CPU_ZERO(&msk);
    int cpu = (rank % MPI_PROCESSES_PER_NODE)*omp_num_threads
              + omp_get_thread_num();
    PLPA_CPU_SET(cpu,&msk);
    PLPA_NAME(sched_setaffinity)((pid_t)0, sizeof(cpu_set_t),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


Hybrid Programming How-To: Example sMVM

JDS parallel sparse matrix-vector multiply – storage scheme



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



(JDS = Jagged Diagonal Storage)

JDS Sparse MVM – Kernel Code

OpenMP parallelization

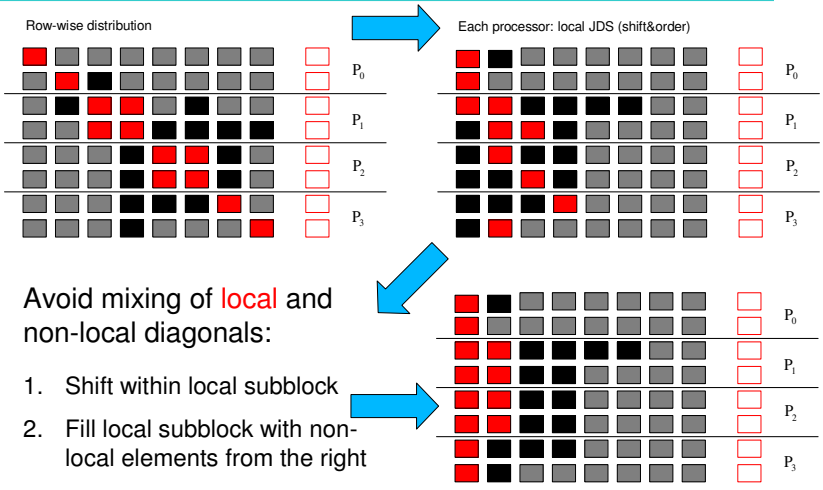
- Implement `c(:) = m(:,:) * b(:)`
- Operation count = $2N_{nz}$

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

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

JDS Sparse MVM

MPI parallelization



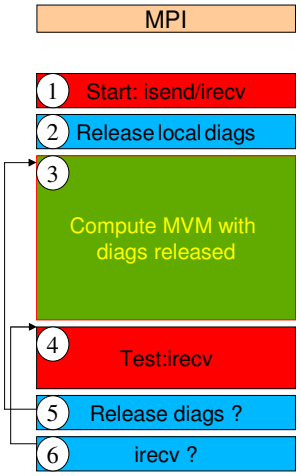
Avoid mixing of local and non-local diagonals:

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

JDS Sparse MVM

Parallel MVM implementations: MPP

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



JDS Sparse MVM

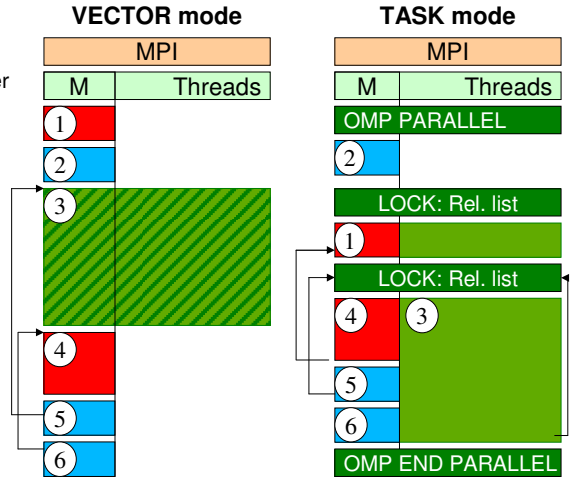
Parallel MVM implementations: Hybrid

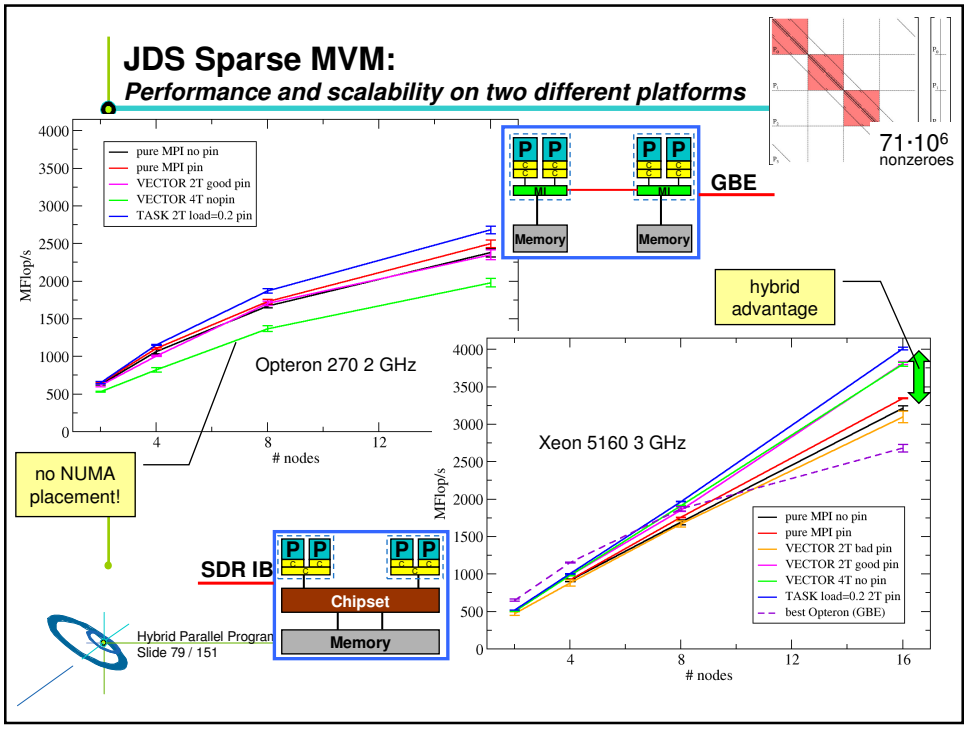
VECTOR mode:

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

TASK mode:

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



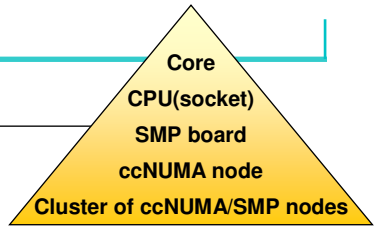


Outline

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

Mismatch Problems

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



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

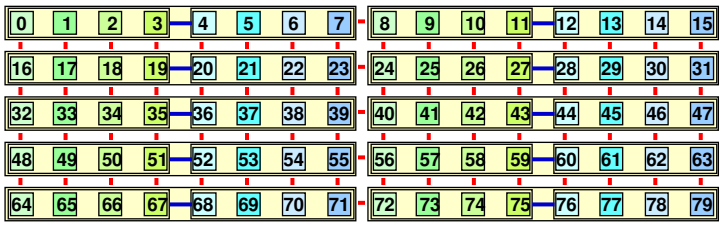
In most cases: **Both categories!**

The Topology Problem with **pure MPI**

one MPI process on each CPU

Application example on 80 cores:

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



\+ 17 x inter-node connections per node

– 1 x inter-socket connection per node

Sequential ranking of MPI_COMM_WORLD

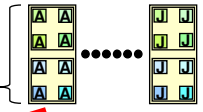
Does it matter?

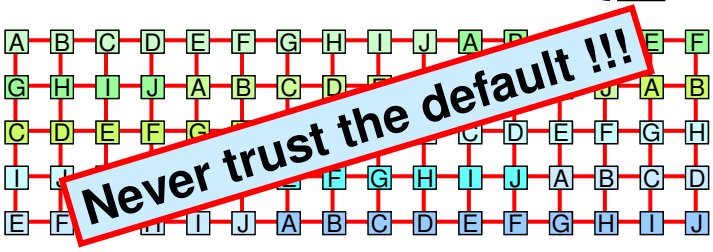
The Topology Problem with **pure MPI**

one MPI process on each CPU

Application example on 80 cores:

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





Never trust the default !!!

\+ 32 x inter-node connections per node

− 0 x inter-socket connection per node

Round robin ranking of MPI_COMM_WORLD

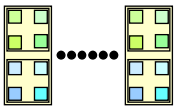
H L R S

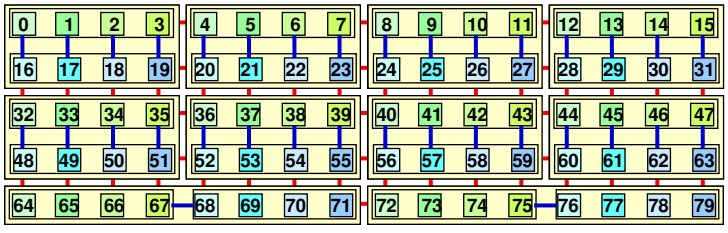
The Topology Problem with **pure MPI**

one MPI process on each CPU

Application example on 80 cores:

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





\+ 10 x inter-node connections per node

\+ 4 x inter-socket connection per node

Two levels of domain decomposition

Bad affinity of cores to thread ranks

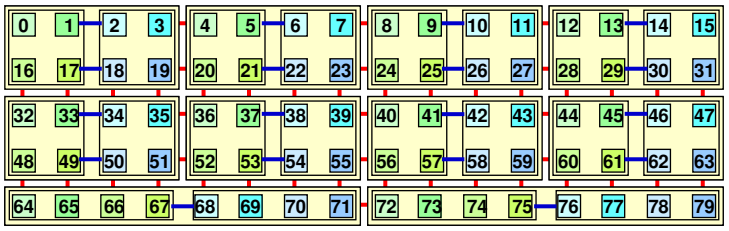
H L R S

The Topology Problem with **pure MPI**

one MPI process on each CPU

Application example on 80 cores:

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



\+ 10 x inter-node connections per node

\+ 2 x inter-socket connection per node

Two levels of domain decomposition

Good affinity of cores to thread ranks

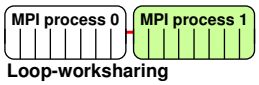
H L R S

The Topology Problem with **hybrid MPI+OpenMP**

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

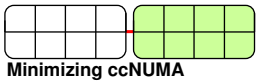
Exa.: 2 SMP nodes, 8 cores/node

Optimal ?



Loop-worksharing on 8 threads

Optimal ?



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

Problem

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

Solutions:

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

Successful examples:

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

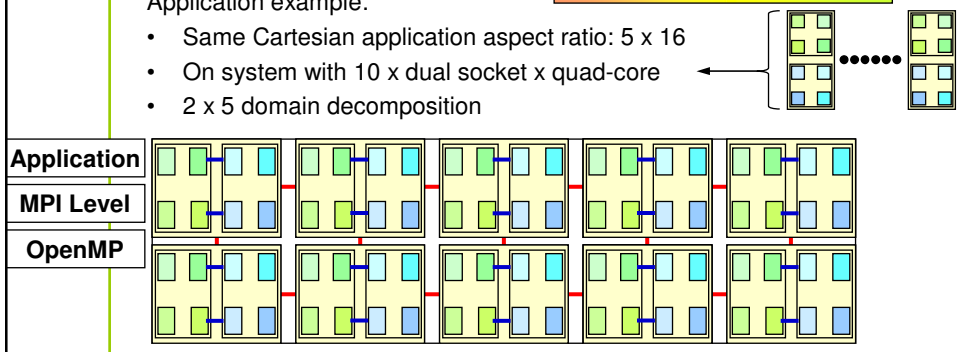
H L R S

The Topology Problem with hybrid MPI+OpenMP

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

Application example:

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



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

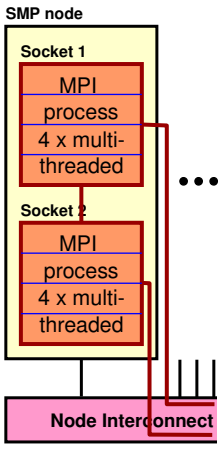
\+ 2 x inter-socket connection per node

Affinity of cores to thread ranks !!!

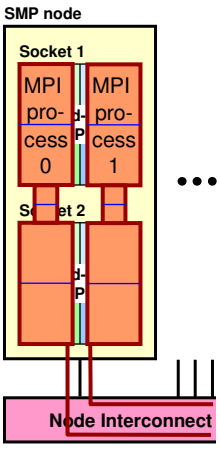
The Mapping Problem with mixed model

pure MPI
&
hybrid MPI+OpenMP

Do we have this?



… or that?



Several multi-threaded MPI process per SMP node:

Problem

- Where are your processes and threads really located?

Solutions:

- Depends on your platform,
- e.g., lbrun **numactl** option on Sun

As seen in case-study on Sun Constellation Cluster Ranger with BT-MZ and SP-MZ

Unnecessary intra-node communication

pure MPI

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

Problem:

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

Solution:

- Only one MPI process per SMP node

Remarks:

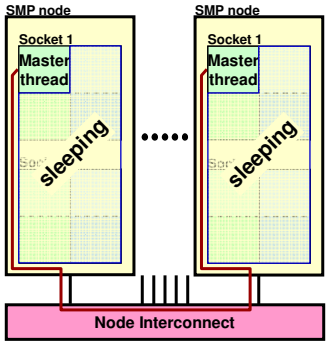
- MPI library must use appropriate fabrics / protocol for intra-node communication
- Intra-node bandwidth higher than inter-node bandwidth → problem may be small
- MPI implementation may cause unnecessary data copying → waste of memory bandwidth

Quality aspects of the MPI library

Sleeping threads and network saturation with Masteronly

MPI only outside of parallel regions

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

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



Problem 1:

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

Solution:

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

Problem 2:

- **Sleeping threads are wasting CPU time**

Solution:

- Overlapping of computation and communication

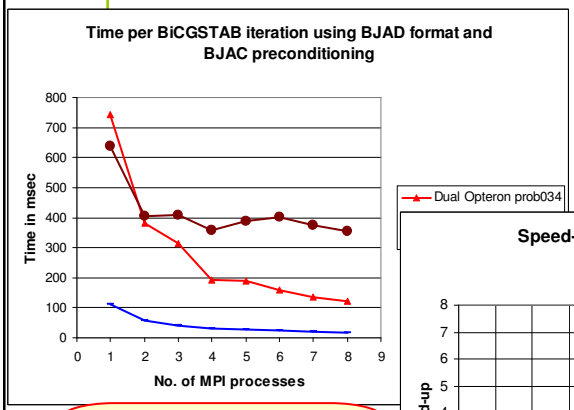
Problem 1&2 together:

- Producing more idle time through lousy bandwidth of master thread

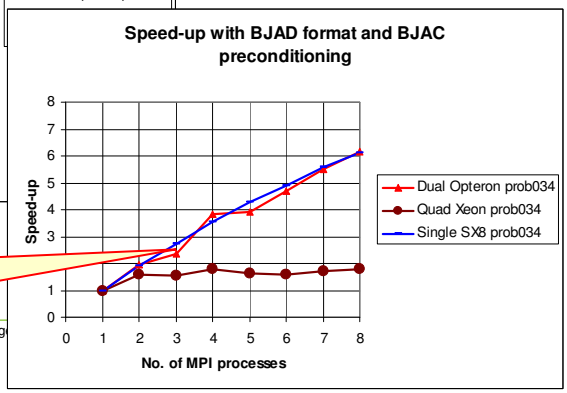
OpenMP: Additional Overhead & Pitfalls

- Using OpenMP
  - → may prohibit compiler optimization
  - → may cause significant loss of computational performance
- Thread fork / join
- On ccNUMA SMP nodes:
  - E.g. in 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?

Memory bandwidth on multi-core clusters with sparse linear solver



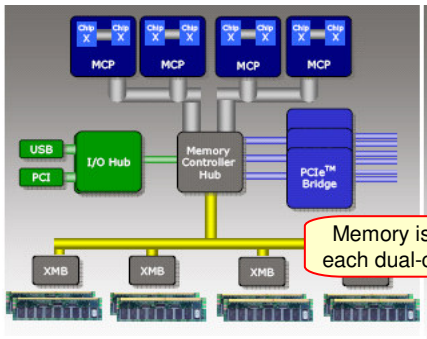
- Problem details: Num of rows & non-zeros = 150K & 15M
- On Dual-core AMD Opteron @ 2.61GHz, Cache: 1024 KB
- Quad-core Intel Xeon @ 2.13GHz, Cache: 4096 KB
- Single-core NEC SX-8 @ 2GHz, Pipes: 4 add, 4 multi

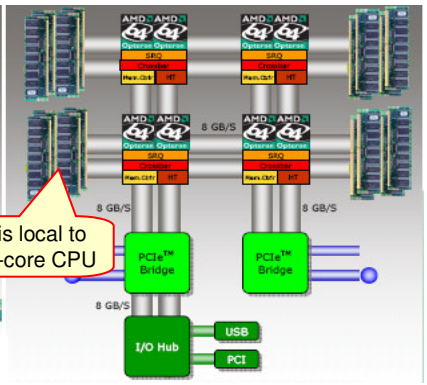


To achieve high speedup, memory locality and first touch policy is essential

Courtesy of Sunil Tiyyagura, HLRS

Multi-core performance with sparse linear solver





Legacy x86 Architecture
- 20-year old traditional front-side bus (FSB) architecture
- CPUs, Memory, I/O all share a bus
- Major bottleneck to performance
- Faster CPUs or more cores ≠ performance

AMD64's Direct Connect Architecture
- Industry-standard technology
- Direct Connected Architecture reduces FSB bottlenecks
- HyperTransport™ interconnect offers scalable high bandwidth and low latency
- 4 memory controlers – increases memory capacity and bandwidth

Courtesy of Sunil Tiyyagura, HLRS and slide: Richard Oehler, AMD

Overlapping Communication and Computation

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

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

  ➔ very hard to do !!!
- the thread-rank problem:
  - comm. / comp. via thread-rank
  - cannot use work-sharing directives

  ➔ loss of major OpenMP support
  (see next slide)
- the load balancing problem

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

Overlapping Communication and Computation

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

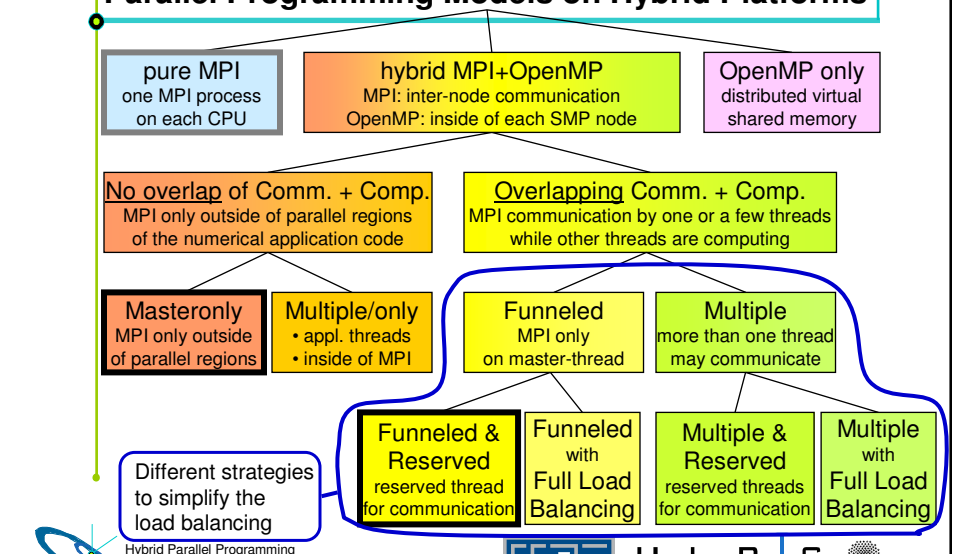
Subteams

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

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

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

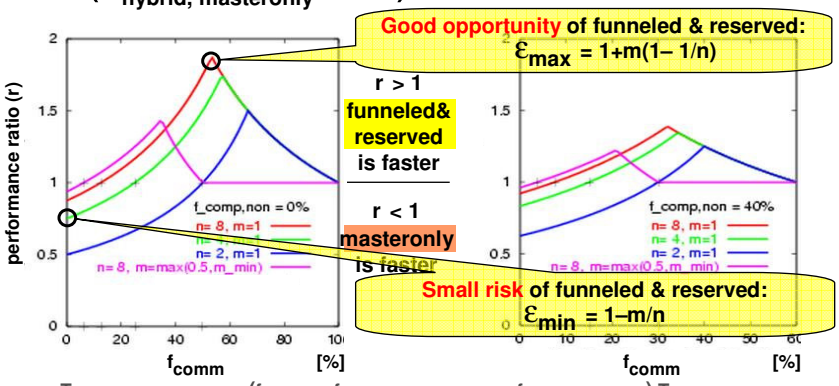

Parallel Programming Models on Hybrid Platforms



Performance ratio (theory)

Masteronly / funneled & reserved

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



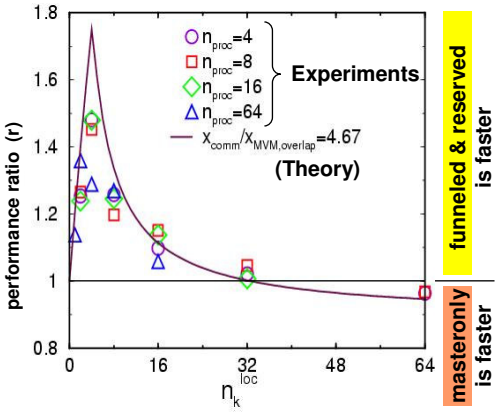
$T_{hybrid,\ masteronly} = (f_{comm} + f_{comp,\ non\text{-}overlap} + f_{comp,\ overlap})\ T_{hybrid,\ masteronly}$

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

skipped

Experiment: Matrix-vector-multiply (MVM)

Masteronly / funneled & reserved



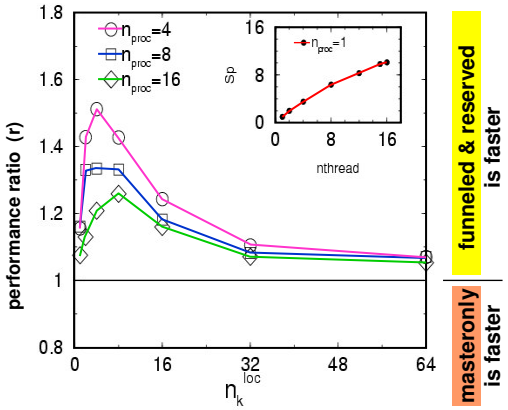
- Jacobi-Davidson-Solver
- Hitachi SR8000
- 8 CPUs / SMP node
- JDS (Jagged Diagonal Storage)
- vectorizing
- n_{proc} = # SMP nodes
- $D_{Mat} = 512*512*(n_k^{loc}*n_{proc})$
- Varying n_k^{loc} ⇒ Varying $1/f_{comm}$
- $\frac{f_{comp,non\text{-}overlap}}{f_{comp,overlap}} = \frac{1}{6}$

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

Masteronly

funneled & reserved

Experiment: Matrix-vector-multiply (MVM)



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

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

OpenMP only

OpenMP/DSM

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

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

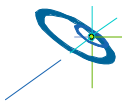
- Implementations
  - e.g., Intel® Cluster OpenMP

OpenMP only

Intel® Compilers with Cluster OpenMP – Consistency Protocol

Basic idea:

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



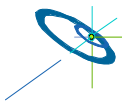
Courtesy of J. Cownie, Intel

hybrid MPI+OpenMP ⟷ OpenMP only

Comparison: MPI based parallelization ←→ DSM

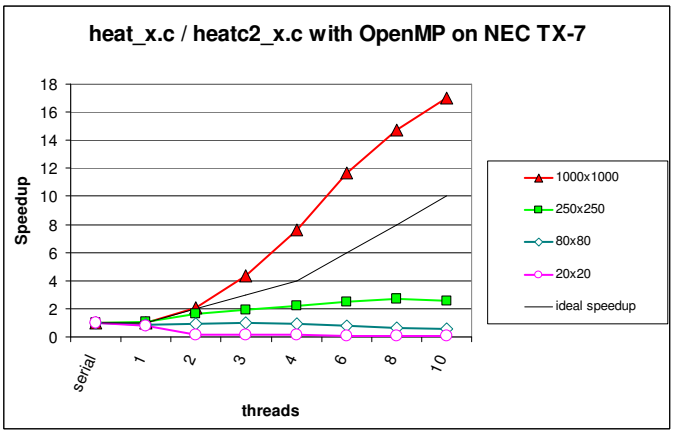
- MPI based:
  - Potential of boundary exchange between two domains in one large message
    - → Dominated by **bandwidth** of the network
- DSM based (e.g. Intel® Cluster OpenMP):
  - Additional latency based overhead in each barrier
    - → May be marginal
  - Communication of **updated data of pages**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

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



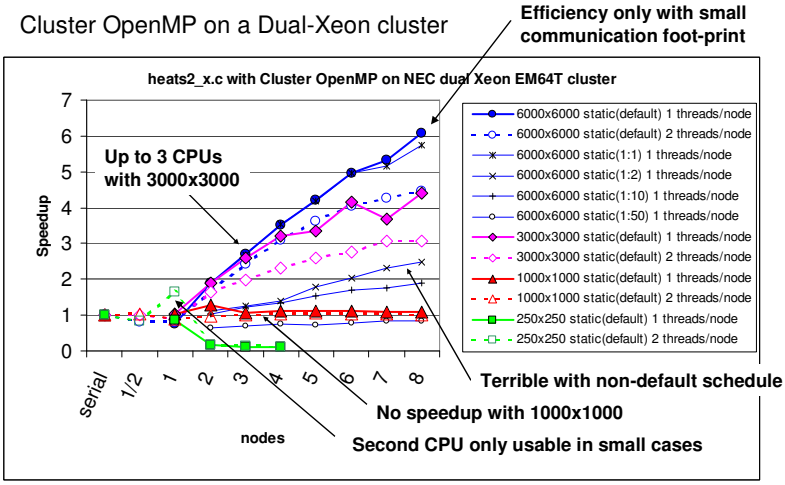
Comparing results with heat example

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

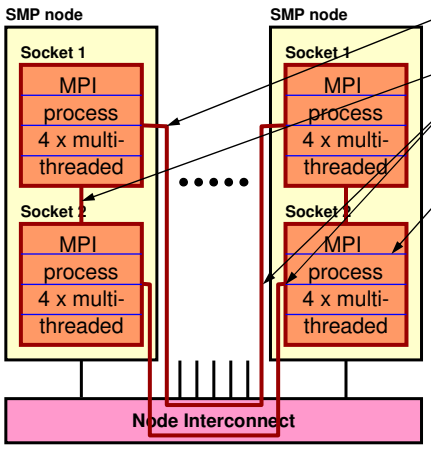


Heat example: Cluster OpenMP Efficiency

- Cluster OpenMP on a Dual-Xeon cluster



Back to the mixed model – an Example



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

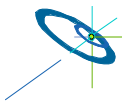
No silver bullet

- The analyzed programming models do **not** fit on hybrid architectures
  - whether drawbacks are minor or major
    - **depends on applications' needs**
  - But there are major opportunities → next section
- In the NPB-MZ case-studies
  - We tried to use optimal parallel environment
    - **for pure MPI**
    - **for hybrid MPI+OpenMP**
  - i.e., the developers of the MZ codes and we tried to minimize the mismatch problems

→ the opportunities in next section dominated the comparisons

Outline

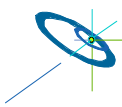
- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / pure MPI vs hybrid MPI+OpenMP
- Practical "How-To" on hybrid programming
- Mismatch Problems
- **Opportunities:**
  Application categories that can benefit from hybrid parallelization
- Thread-safety quality of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



Opportunities of hybrid parallelization (MPI & OpenMP)

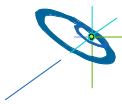
Overview

- 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
- … **(→ slide on "Further Opportunities")**



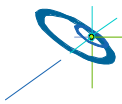
Nested Parallelism

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



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

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



Memory consumption

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

Memory consumption (continued)

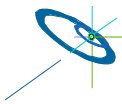
- Future:
  With 100+ cores per chip the memory per core is limited.
  - Data reduction through usage of shared memory may be a key issue
  - Domain decomposition on each hardware level
    - **Maximizes**
      - Data locality
      - Cache reuse
    - **Minimizes**
      - CCnuma accesses
      - Message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

How many multi-threaded MPI processes per SMP node

- SMP node = 1 Chip
  - 1 MPI process per SMP node
- SMP node is n-Chip ccNUMA node
  - With x NICs (network interface cards) per node
- How many MPI processes per SMP node are optimal?
  - somewhere between 1 and n

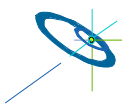
In other words:

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



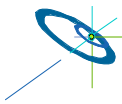
Opportunities, if MPI speedup is limited due to algorithmic problems

- 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



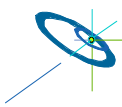
Further Opportunities

- Reduced number of MPI messages, reduced aggregated message size } compared to pure MPI
- Functional parallelism
  → e.g., I/O in an other thread
- MPI shared memory fabrics not loaded
  if whole SMP node is parallelized with OpenMP



Outline

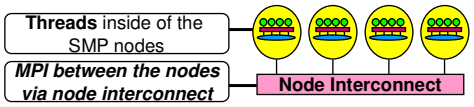
- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / pure MPI vs hybrid MPI+OpenMP
- Practical "How-To" on hybrid programming
- Mismatch Problems
- Opportunities:
  Application categories that can benefit from hybrid parallelization
- **Thread-safety quality of MPI libraries**
  Rainer Keller, High Performance Computing Center Stuttgart (HLRS)
- Tools for debugging and profiling MPI+OpenMP
- Summary



Thread-safety of MPI Libraries

- Make most powerful usage of hierarchical structure of hardware:
- Efficient programming of clusters of SMP nodes

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



- No restriction to the usage of OpenMP for intranode-parallelism:
  - OpenMP does not (yet) offer binding threads to processors
  - OpenMP does not guarantee thread-ids to stay fixed.
- OpenMP is based on the implementation dependant thread-library: LinuxThreads, NPTL, SolarisThreads.

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, not part of the standard) **but only master thread will make MPI-calls AND only while other threads are sleeping**
  - **MPI_THREAD_FUNNELED:** **Only master thread will make MPI-calls**
  - **MPI_THREAD_SERIALIZED:** **Multiple threads may make MPI-calls, but only one at a time**
  - **MPI_THREAD_MULTIPLE:** **Multiple threads may call MPI, with no restrictions**
- returned `provided` may be less than REQUIRED by the application

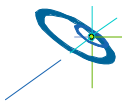
Calling MPI inside of OMP MASTER

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

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

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

#pragma omp barrier
```

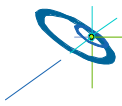
- But this implies that all other threads are sleeping!
- The additional barrier implies also the necessary cache flush!



… the barrier is necessary – example with MPI_Recv

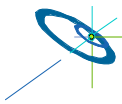
```
!$OMP PARALLEL
!$OMP DO
        do i=1,1000
           a(i) = buf(i)
        end do
!$OMP END DO NOWAIT
!$OMP BARRIER
!$OMP MASTER
        call MPI_RECV(buf,...)
!$OMP END MASTER
!$OMP BARRIER
!$OMP DO
        do i=1,1000
           c(i) = buf(i)
        end do
!$OMP END DO NOWAIT
!$OMP END PARALLEL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

```
#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 */
```



Testsuite – Goals

- There exist many different test-suites:
  - MPIch: Collection regression tests for specific functions.
  - Intel: Single program for every MPI-1.2 function.
  - IBM: Single program MPI-1 and MPI-2 tests; but incomplete.
- Aims of the testsuite:
  - Single program (PACX-MPI, Queue-System limits, late Errors) Expected Passes: checking boundaries of the MPI standard.
  - Easy to configure, compile and install.
  - Tests must be runable with any number of processes.
  - Tests must run with as many:
    - **Communicators**
    - **Datatypes**
    - **Reduction-Operations**
    - **Lengths**
- Currently ~7200 combinations w/ ~70% of src-coverage: ompi/gcov

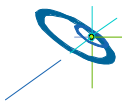


Testsuite – Startup

- Easy startup – or complete control:

```
mpirun -np 16 ./mpi_test_suite
     -t 'Many-to-one,Collective,!Bcast'
     -d MPI_INT,TST_MPI_STRUCT_C_TYPES
     -c 'MPI_COMM_WORLD,Halved Intercommunicator'
     -r FULL  -x STRICT
```

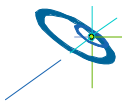
- Each test has to implement three functions:
  - `Init` One time test-initialization (buffer allocation
  - `Run` Main test-function, may be run multiple times.
  - `Cleanup` After the particular test was run.
- Make usage of convenience functions:
  - `tst_test_setstandardarray` Set buffer to known value.
  - `tst_test_checkstandardarray` Corresponding check



Testsuite – Implemented Communicators

- List of implemented communicators:

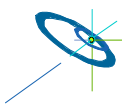
MPI_COMM_WORLD	MPI_COMM_NULL	MPI_COMM_SELF
Duplicated MPI_COMM_WORLD	Reversed MPI_COMM_WORLD	Halved MPI_COMM_WORLD
Odd-/Even Split MPI_COMM_WORLD		
Zero-and-Rest Intercommunicator	Halved Intercommunicators	Intracomm merged of Halved Intercomms
Two-dimensional Cartesian	Three-dimensional Cartesian	Fully-connected Topology



Testsuite – Implemented Datatypes

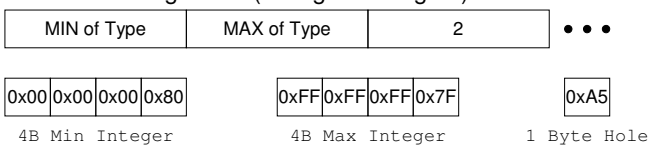
- List of implemented Datatypes:

All Predefined MPI1 C-types… (15)	All Predefined MPI2 C-types… (+5)	
Reduce types for std. communication (5)		
Derived Types Contiguous Memory	Derived Types w/ Holes	Derived Types w/ Holes and LB/UB
Duplicated MPI_Char (MPI2)	Duplicated Derived w/ Holes & LB/UB	

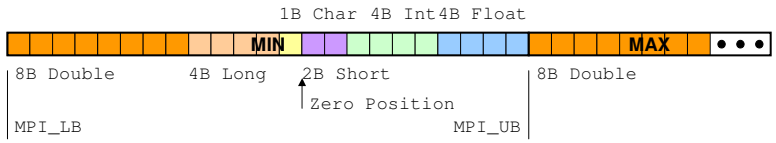


Testsuite – Derived Datatypes

- Make usage of convenience functions:
  - `tst_test_setstandardarray` Set buffer to known value.
- Sets the following buffer (so e.g. for Integers):

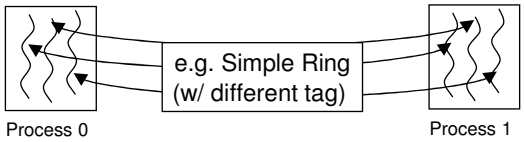


- E.g. the following derived datatype MPI_TYPE_MIX_LB_UB:

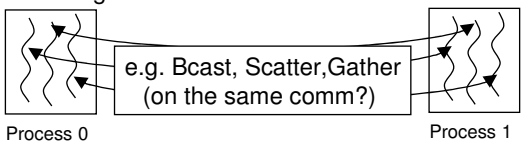


Testsuite – Implemented threaded tests

- Additional tests added:
  - Local send from one thread to self on MPI_COMM_SELF
  - Calling MPI_Init_thread from thread.
- Threaded running of already implemented tests:
  - Scheduling the same test to many threads (pt2pt)



  - Scheduling different tests to different threads:

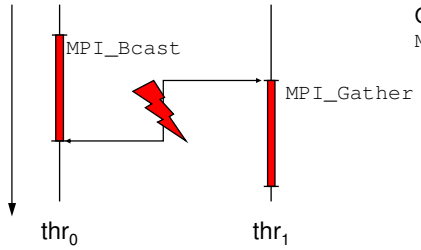


Testsuite – Implemented threaded tests

- Scheduling different Collective Operations to different threads but on the same communicator? Allowed?

  `(MPI-2, p195): Matching of collective calls on a communicator, window, or file handle is done according to the order in which they are issued at each process.`

  User has to order **calling** sequence, or the **execution** sequence?



Of course, one may use MPI-2's
`MPI_Comm_dup(MPI_COMM_WORLD, &new_comm);`

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`.
MPIch2-1.0.6	ch3:sock supports `MPI_THREAD_MULTIPLE`
	ch:nemesis has "Initial Thread-support"
MPIch2-1.1.0a1	ch3:sock (default) supports `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`

- Examples of failures in MPI libraries uncovered are shown.
- Failure logs are shown **only** for Open MPI.

Examples of failed multi-threaded tests

- Standard send in comm. "`Reversed MPI_COMM_WORLD`":

```
P2P tests Ring, comm Reversed MPI_COMM_WORLD, type MPI_INT
mpi_test_suite:
../../../../../ompi/mca/pml/ob1/pml_ob1_sendreq.c:196:
mca_pml_ob1_match_completion_free: Assertion `0 == sendreq-
>req_send.req_base.req_pml_complete' failed.
```

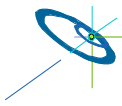
- 2-threads Collective (Bcast, Bcast) on different comms wrong data:

```
mpirun -np 4 ./mpi_test_suite -r FULL -j 2 -t "Bcast" -c
"MPI_COMM_WORLD,Duplicated MPI_COMM_WORLD"
```

- 2-threads Collective (Bcast, Gather) on different comms hangs:

```
mpirun -np 4 ./mpi_test_suite -r FULL -j 2 -t "Bcast,Gather"
-c "MPI_COMM_WORLD,Duplicated MPI_COMM_WORLD"
```

- Of course, a **test-suite** may contain errors as well ,-]



Thread support within Open MPI

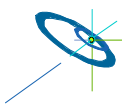
- In order to enable thread support in Open MPI, configure with:

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

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

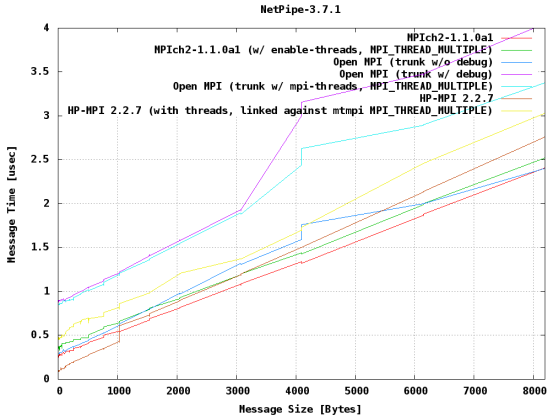
```
configure --enable-mpi-threads --enable-progress-threads
```

- 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



Thread Overhead in MPI implementations

- Multi-thread safety has a a cost, by adding mutex-locks/unlocks
- Measurement for Shared-Memory communication using Netpipe



Measured on IBM T61p, w/ Intel T9500, 2.6GHz, 6MB L2 Cache, using own Benchmark + PAPI

Thread Overhead in MPI implementations

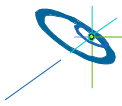
- Multi-thread safety has a a cost, by adding mutex-locks/unlocks
- Measurement for Shared-Memory communication:

	Plain No progress-threads			**With Threads & MPI_THREAD_MULTIPLE** No progress-threads		
	MPIch2 1.1.0a1	Open MPI R19511	HP-MPI 2.2.7	MPIch2 1.1.0a1	Open MPI r19511	HP-MPI 2.2.7 -lmtmpi
Latency (NetPipe-3.7.1)	0.24µs	0.29µs	0.09µs	0.34µs	0.85µs	0.46µs
Ø Instructions per Send+Recv	1027	2025	830	1487	3170	2829
Ø Cycles per Send+Recv	785	1508	462	1312	4379	1749
Ø Branches per Send+Recv	203	367	264	340	611	677

Measured on IBM T61p, w/ Intel T9500, 2.6GHz, 6MB L2 Cache, using own Benchmark + PAPI

Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / pure MPI vs hybrid MPI+OpenMP
- Practical "How-To" on hybrid programming
- Mismatch Problems
- Opportunities:
  Application categories that can benefit from hybrid parallelization
- Thread-safety quality of MPI libraries
- **Tools for debugging and profiling MPI+OpenMP**
  Rainer Keller, High Performance Computing Center Stuttgart (HLRS)
- Summary

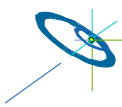


Valgrind Analysis 1/4

- An Open-Source Debugging & Profiling tool
- Works with any dynamically & statically linked application
- Emulates CPU, i.e. executes instructions on a synthetic CPU
- Currently it's only available for Linux/x86/x86_64, Linux/Power

- Has been used on many **large** Projects:
  KDE, Emacs, Gnome, Mozilla, OpenOffice.

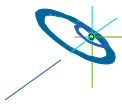
- It's easily configurable to ease debugging & profiling as *tools*:
  - Memcheck: Complete Checking (every memory access)
  - Helgrind: Find Races in multithreaded programs
  - Massif: Memory Allocation Profiler
  - Callgrind: A Cache & Call-tree profiler.



Valgrind Analysis – Thread Correctness 2/4

- Checking a threaded program with race-conditions:
  `valgrind --tool=helgrind ./pthread_race-gcc`

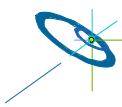
```
==8187== Thread #2 was created
==8187==    at 0x511A08E: clone (in /lib64/libc-2.8.so)
...
==8187==    by 0x4007E5: main (pthread_race.c:43)
...
==8187== Possible data race during write of size 4 at 0x601068
==8187==    at 0x4007B9: thread (pthread_race.c:31)
==8187==    by 0x4C2875F: mythread_wrapper (hg_intercepts.c:193)
==8187==    by 0x4E3203F: start_thread (in /lib64/libpthread-2.8.so)
==8187==    by 0x511A0CC: clone (in /lib64/libc-2.8.so)
==8187==   Old state: owned exclusively by thread #2
==8187==   New state: shared-modified by threads #2, #3
==8187==   Reason:    this thread, #3, holds no locks at all
==8187==   Location 0x601068 is 0 bytes inside local var "global_variable"
==8187==   declared at pthread_race.c:22, in frame #0 of thread 2
```



Valgrind Analysis – Thread Correctness 3/4

- Checking a threaded program with race-conditions:
  `valgrind --tool=drd ./pthread_race-gcc`

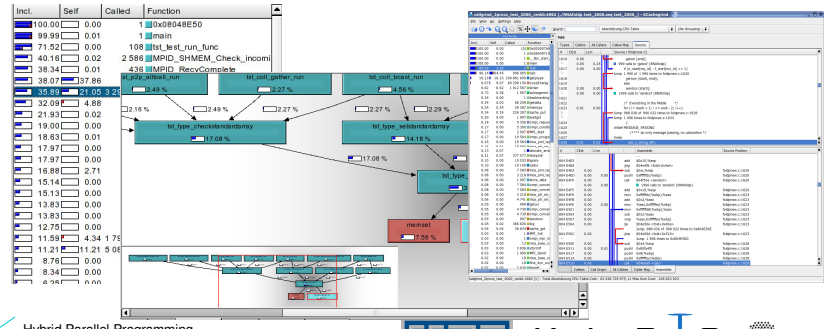
```
==8562== Thread 3:
==8562== Conflicting store by thread 3/3 at 0x00601068 size 4
==8562==    at 0x4007B9: thread (pthread_race.c:31)
==8562==    by 0x4C29B97: vg_thread_wrapper (drd_pthread_intercepts.c:186)
==8562==    by 0x4E3503F: start_thread (in /lib64/libpthread-2.8.so)
==8562==    by 0x511D0CC: clone (in /lib64/libc-2.8.so)
==8562== Allocation context: BSS section of …/pthread_race-gcc
==8562== Other segment start (thread 0/2)
==8562==    (thread finished, call stack no longer available)
==8562== Other segment end (thread 0/2)
==8562==    (thread finished, call stack no longer available)
Main: global_variable:108677456
==8562==
==8562== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 26 from 1)
```



Valgrind Analysis – Thread Performance 4/4

- The well-known kcachegrind may be used with Threads,
- However, the callgrind-tool has to be specially called:

```
valgrind --tool=callgrind
      --simulate-cache=yes --simulate-wb=yes
      --collect-jumps=yes --dump-instr=yes
      --separate-threads=yes ./matrix-multiply
```



Thread Correctness – Intel ThreadChecker 1/3

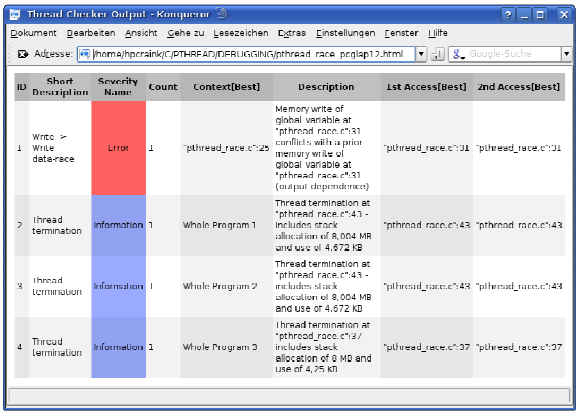
- Intel ThreadChecker operates in a similar fashion to `helgrind`,
- Compile with `-tcheck`, then run program using `tcheck_cl`:

```
Application finished

|ID|Short De|Sever|C|Contex|Description                          |1st Acc|2nd Acc|
|  |scriptio|ity  |o|t[Best|                                       |ess[Bes|ess[Bes|
|  |n       |Name |u|]     |                                       |t]     |t]     |
|  |        |     |n|      |                                       |       |       |
|  |        |     |t|      |                                       |       |       |

|1 |Write ->|Error|1|"pthre|Memory write of global_variable at|"pthrea|"pthrea|
|  |Write da|     | |ad_rac|"pthread_race.c":31 conflicts with|d_race.|d_race.|
|  |ta-race |     | |e.c":2|a prior memory write of           |c":31  |c":31  |
|  |        |     | |5     |global_variable at                |       |       |
|  |        |     | |      |"pthread_race.c":31 (output       |       |       |
|  |        |     | |      |dependence)                       |       |       |
```

Thread Correctness – Intel ThreadChecker 2/3

- One may output to HTML:

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



Thread Correctness – Intel ThreadChecker 3/3

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

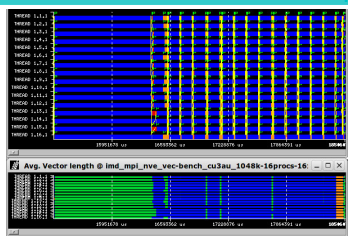
```
configure --enable-mpi-threads
          --enable-debug
          --enable-mca-no-build=memory-ptmalloc2
      CC=icc F77=ifort FC=ifort
      CFLAGS='-debug all -inline-debug-info tcheck'
      CXXFLAGS='-debug all -inline-debug-info tcheck'
      FFLAGS='-debug all -tcheck'    LDFLAGS='tcheck'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

- Then run with:

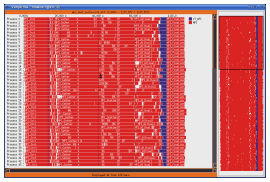
```
mpirun --mca tcp,sm,self -np 2 tcheck_cl              \
    --reinstrument -u full --format html              \
    --cache_dir '/tmp/hpcraink_$$__tc_cl_cache'       \
    --report 'tc_mpi_test_suite_$$'                   \
    --options 'file=tc_mpi_test_suite_%H_%I,          \
               pad=128, delay=2, stall=2'        --   \
./mpi_test_suite -j 2 -r FULL -t 'Ring Ibsend' -d MPI_INT
```

Performance Tools Support for Hybrid Code

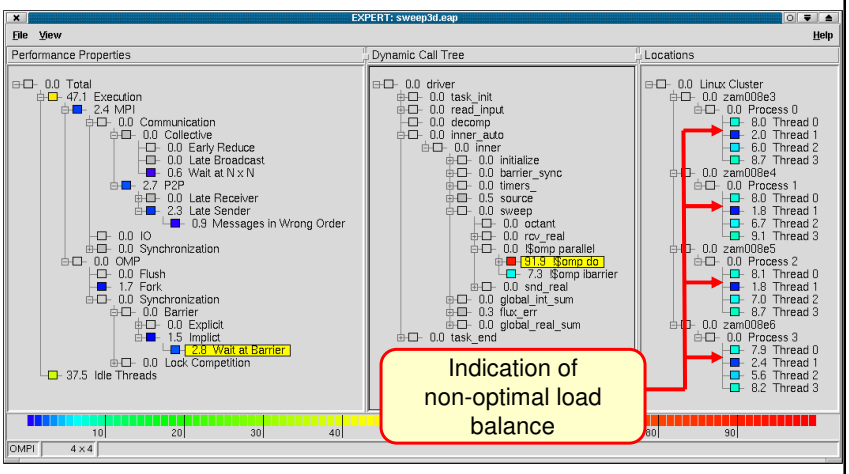
- Paraver examples have already been shown, tracing is done with linking against (closed-source) `omptrace` or `ompitrace`



- 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`!)

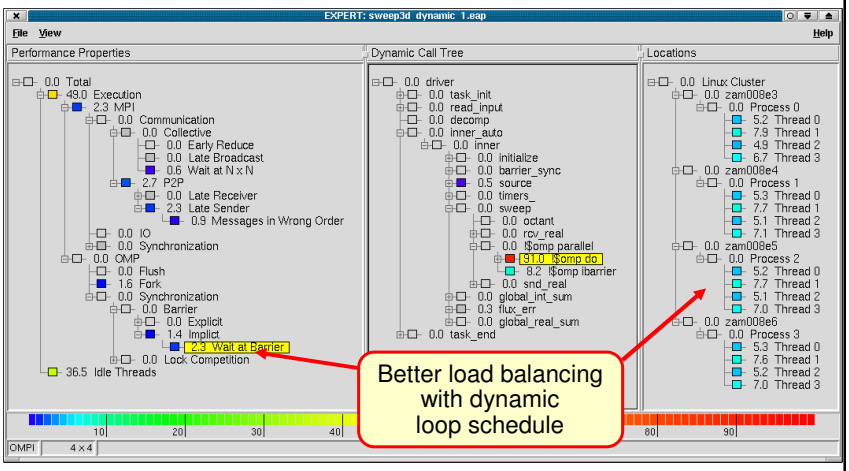


Kojak – Example "Wait at Barrier"



Screenshots, courtesy of KOJAK JSC, FZ Jülich

Kojak – Example "Wait at Barrier", Solution



H L R S

Screenshots, courtesy of KOJAK JSC, FZ Jülich

Outline

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

H L R S

Acknowledgements

- We want to thank
  - Gerhard Wellein, RRZE
  - Sunil Tiyyagura, HLRS
  - Richard Oehler, AMD
  - Jim Cownie, Intel
  - KOJAK project at JSC, Research Center Jülich
  - HPCMO Program and the Engineer Research and Development Center Major Shared Resource Center, Vicksburg, MS (http://www.erdc.hpc.mil/index)

Summary – the good news

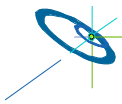
MPI + OpenMP

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

Summary – the bad news

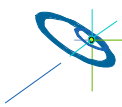
MPI+OpenMP: There is a huge amount of pitfalls

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



Summary – good and bad

- Problems may be small
  - x% loss efficiency ----mismatch problem----> f•x% loss
  - If loss is small x=1% and factor f=3 is medium → don't worry ?!
- Optimization
  - 1 MPI process per core ..... 1 MPI process per SMP node
    ^– somewhere between may be the optimum
- ☺Efficiency of MPI+OpenMP is not for free:
  The efficiency strongly depends on
  ☹the amount of work in the source code development



Summary – Alternatives

Pure MPI

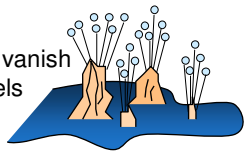
- \+ Ease of use
- – Topology and mapping problems may need to be solved
  (depends on loss of efficiency with these problems)
- – Number of cores may be more limited than with MPI+OpenMP
- \+ Good candidate for perfectly load-balanced applications

Pure OpenMP

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

Summary

- This tutorial tried to
  - – help to negotiate obstacles with hybrid parallelization,
  - – give hints for the design of a hybrid parallelization,
  - – and technical hints for the implementation → "How To",
  - – show tools if the application does not work as designed.
- This tutorial was not an introduction into other parallelization models:
  - – Partitioned Global Address Space (PGAS) languages
    (Unified Parallel C (UPC), Co-array Fortran (CAF), Chapel, Fortress, Titanium, and X10).
  - – High Performance Fortran (HPF)
  - → Many rocks in the cluster-of-SMP-sea do not vanish into thin air by using new parallelization models
  - → Area of interesting research in next years



Conclusions

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

Thank you for your interest

Q & A

Please fill in the feedback sheet – Thank you

Appendix

- Abstract
- Intel® Compilers with Cluster OpenMP – Consistency Protocol – Examples
- 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, Texas Advanced Computing Center / Naval Postgraduate School, USA
Rainer Keller, HLRS, University of Stuttgart, Germany

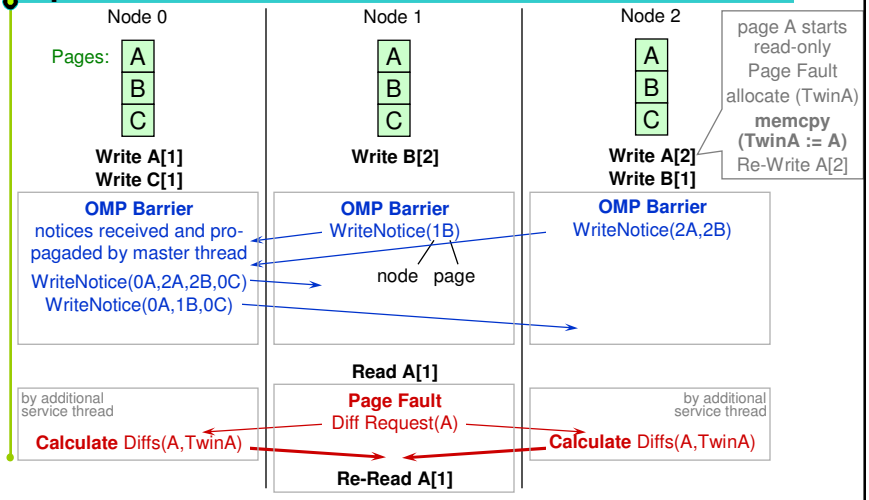
Abstract. Most HPC systems are clusters of shared memory nodes. Such systems can be PC clusters with dual or quad boards and single or multi-core CPUs, but also "constellation" type systems with large SMP nodes. Parallel programming may combine the distributed memory parallelization on the node interconnect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

This tutorial analyzes the strength and weakness of several parallel programming models on clusters of SMP/multi-core nodes. Various hybrid MPI+OpenMP programming models are compared with pure MPI. Benchmark results of several platforms are presented. The thread-safety quality of several existing MPI libraries is also discussed. Case studies are provided to demonstrate various aspect of hybrid MPI/OpenMP programming. Another option is the use of distributed virtual shared-memory technologies. Application categories that can take advantage of hybrid programming are identified. Multi-socket-multi-core systems in highly parallel environments are given special consideration. This tutorial analyzes strategies to overcome typical drawbacks of easily usable programming schemes on clusters of SMP nodes.

Intel® Cluster OpenMP

- The following slides show the complexity of the communication protocol of Intel® Cluster OpenMP

Consistency Protocol Detail of Intel® Cluster OpenMP



Courtesy of J. Cownie, Intel

Real consistency protocol is more complicated

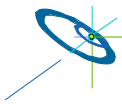
- Diffs are done only when requested
- Several diffs are locally stored and transferred later if a thread first reads a page after several barriers.
- Each write is internally handled as a read followed by a write.
- If too many diffs are stored, a node can force a "reposession" operation, i.e., the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

Courtesy of J. Cownie, Intel

Intel® Compilers with Cluster OpenMP – Consistency Protocol – Examples

Notation

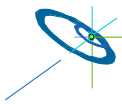
- ..=A[i] Start/End — Start/end a read on element i on page A
- A[i]=.. Start/End — Start/end a write on element i on page A, trap to library
- Twin(A) — Create a twin copy of page A
- WriteNotice(A) — Send write notice for page A to other processors
- DiffReq_A_n(s:f) — Request diffs for page A from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



Courtesy of J. Cownie, Intel

Exa. 1

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



Courtesy of J. Cownie, Intel

Exa. 2

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

Courtesy of J. Cownie, Intel

Exa. 3 (start)

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

Courtesy of J. Cownie, Intel

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

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

Courtesy of J. Cownie, Intel

Rolf Rabenseifner

Dr. Rolf Rabenseifner studied mathematics and physics at the University of Stuttgart. Since 1984, he has worked at the High-Performance Computing-Center Stuttgart (HLRS). He led the projects DFN-RPC, a remote procedure call tool, and MPI-GLUE, the first metacomputing MPI combining different vendor's MPIs without loosing the full MPI interface. In his dissertation, he developed a controlled logical clock as global time for trace-based profiling of parallel and distributed applications. Since 1996, he has been a member of the MPI-2 Forum and since Dec. 2007, he is in the steering committee of the MPI-3 Forum. From January to April 1999, he was an invited researcher at the Center for High-Performance Computing at Dresden University of Technology. Currently, he is head of Parallel Computing - Training and Application Services at HLRS. He is involved in MPI profiling and benchmarking, e.g., in the HPC Challenge Benchmark Suite. In recent projects, he studied parallel I/O, parallel programming models for clusters of SMP nodes, and optimization of MPI collective routines. In workshops and summer schools, he teaches parallel programming models in many universities and labs in Germany.

Georg Hager

Dr. Georg Hager studied theoretical physics at the University of Bayreuth, specializing in nonlinear dynamics and holds a PhD in Computational Physics from the University of Greifswald. Since 2000 he is a member of the HPC Services group at the Regional Computing Center Erlangen (RRZE), which is part of the University of Erlangen-Nuremberg. His daily work encompasses all aspects of user support in High Performance Computing like tutorials and training, code parallelization, profiling and optimization and the assessment of novel computer architectures and tools. Recent research includes architecture-specific optimization strategies for current microprocessors and special topics in shared memory programming.

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 1998 she joined the NASA Ames Research Center in Moffett Field, California, USA as a Research Scientist. Here her work focused on evaluating and enhancing tools for parallel program development and investigating the usefulness of different parallel programming paradigms.

In 2005 she moved from California to the Pacific Northwest and joined Sun Microsystems as a staff engineer in the Compiler Performance Engineering team. Her task is the analysis of compiler generated code and providing feedback and suggestions for improvement to the compiler group. Her research interest remains in area of performance analysis and evaluation of programming paradigms for high performance computing. Currently, she is working at the Texas Advanced Computing Center / Naval Postgraduate School.

Rainer Keller

Rainer Keller is a scientific employee at the High Performance Computing Center Stuttgart (HLRS) since 2001. He earned his diploma in Computer Science at the University of Stuttgart. Currently, he is the head of the group Applications, Models and Tools at the HLRS.

His professional interest are Parallel Computation using and working on MPI with Open MPI and shared memory parallelization with OpenMP, as well as distributed computing using the Meta-Computing Library PACX-MPI.

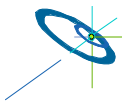
His work includes performance analysis and optimization of parallel applications, as well as the assessment of and porting to new hardware technologies, including the training of HLRS users in parallel application development. He is involved in several European projects, such as HPC-Europa.

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

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

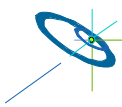
References

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



References

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



References

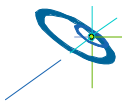
- Barbara Chapman et al.:
  Toward Enhancing OpenMP's Work-Sharing Directives.
  In proceedings, W.E. Nagel et al. (Eds.): Euro-Par 2006, LNCS 4128, pp. 645-654, 2006.
- Barbara Chapman, Gabriele Jost, and Ruud van der Pas:
  Using OpenMP.
  The MIT Press, 2008.

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

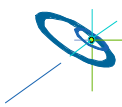
Further references

- 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

Intentionally empty.

Content

Content