Hybrid MPI and OpenMP Parallel Programming

MPI + OpenMP and other models on clusters of SMP nodes

Rolf Rabenseifner1) Georg Hager2) Gabriele Jost3) Rainer Keller1)
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 SUPERCOMPETING 2008 (SC08)
Austin, Texas, USA, Nov. 17

Outline

- Introduction / Motivation 2
- Programming models on clusters of SMP nodes 8
- Case Studies / pure MPI vs hybrid MPI+OpenMP 15
- Practical “How-To” on hybrid programming 45
- Mismatch Problems 80
- Opportunities: Application categories that can benefit from hybrid parallelization 107
- Thread-safety quality of MPI libraries 116
- Tools for debugging and profiling MPI+OpenMP 133
- Summary 144
  - Appendix 152
  - Content (detailed) 177
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.
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

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

Hybrid Parallel Programming Models on Hybrid Platforms

- Pure MPI
  - one MPI process on each CPU
- Hybrid MPI+OpenMP
  - MPI: inter-node communication
  - OpenMP: inside each SMP node
- OpenMP only
  - distributed virtual shared memory

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

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

Masteronly
- MPI only outside of parallel regions
Pure MPI

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

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

Hybrid Masteronly

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

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

→ Section Thread-safety quality of MPI libraries
Overlapping Communication and Computation
MPI communication by one or a few threads while other threads are computing

```c
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)

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

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

- Introduction / Motivation
- Programming models on clusters of SMP nodes

- 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 parallelism
- Thread-safety quality of MPI libraries
- Tools for debugging and profiling MPI+OpenMP
- Summary

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

- 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

```fortran
call omp_set_numthreads (weight)
do step = 1, itmax
  call exch_qbc(u, qbc, nx, 1, ny, nz)
do zone = 1, num_zones
  if (iam .eq. pzone_id(zone))
    call ssor(u, rsd, 1, 1, 1)
end if
end do
end do...
call mpi_send/recv
```

- 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
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
- Limited MPI Parallelism: \( \rightarrow \) 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`

---

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:
  `sxmplf90  -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

```plaintext
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

```plaintext
do zone = myzone_first, myzone_last
  (MPI communication)
$OMP PARALLEL DO
  do k
    do j
      do i
        ...
        rhs_3d(i, j, k) = c * rhs_3d(i-1, j, k) + ....
      
Loop interchange yields vectorizable inner loop
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
  - 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
  - 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

<table>
<thead>
<tr>
<th>BT-MZ Class B 8x8x1</th>
<th>Comm Time in secs.</th>
<th>Size in 10MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI Proc ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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
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
\[\begin{align*}
\text{cpu1} \\
\text{cpu2} \\
\ldots \\
\text{cpun}
\end{align*}\]

2 MPI per node: machines
\[\begin{align*}
\text{cpu1} \\
\text{cpu1} \\
\text{cpu2} \\
\text{cpu2} \\
\ldots \\
\text{cpun} \\
\text{cpun}
\end{align*}\]

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
Hybrid MPI and OpenMP Parallel Programming
Tutorial M09 at SC'08, Austin, Texas, USA, Nov. 17, 2008

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

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

BT-MZ Class B on 16 MPI x 2 OpenMP

• 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

BT-MZ Pure MPI vs MPI/OpenMP on NEC EMT64
Hybrid MPI and OpenMP Parallel Programming

**Slide 35 / 151**

Rabenseifner, Hager, Jost, Keller

Hybrid Parallel Programming

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.

**Slide 36 / 151**

Rabenseifner, Hager, Jost, Keller

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
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 ftln compiler based on PGI pgf90 7.1
  - Ftln -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)

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 limited

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 aggregate 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
    - lbrun numacl bt-mz.exe
- **numactl** controls
  - Socket affinity: select sockets to run
  - Core affinity: select cores within socket
  - Memory policy: where to allocate memory
Hybrid Parallel Programming

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

NPB-MZ Class E Scalability on Sun Constellation

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 built for 8192 processes!

Hybrid:
SP: still scales
BT: does not scale

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): 3.4 sec max
      - Count: 4531 isend per MPI Process
      - Size: 802 MB per MPI Process
      - Total size: ~51328 MB
  - 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, 236 MB Min
      - Total Size: ~110000 MB.

- **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 = \#zones yields poor performance
  - \#nprocs < \#zones => room for better workload balance, but decreases 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
      - 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

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>
    $ MPIEXEC=“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
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
    ```sh
    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
      ```sh
      test -s ~/myexports && . ~/myexports
      ```
      in your ~/.bashrc
    - Add
      ```sh
      echo '
      ```
      in your ~/.bashrc
    - Caution: Several invocations of mpirun cannot be executed at the same time with this trick!

Handling of OMP_NUM_THREADS (continued)

- with support by OpenMPI -x option:
  ```sh
  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 “-nperrnode #” 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 -npnode 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 \( A(:,)=B(:,)+C(:,)*D(:,) \)
Contestation vs. parallel access on HP DL585 G5:
OpenMP vector triad \(A(:)=B(:)+C(:)*D(:)\)

In-cache performance unharmed by ccNUMA

Affinity matters!

Perfect scaling across LDs

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:

  ```c
  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!)

---

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

- Between two cores of one socket
- Between two sockets of one node
- Between two nodes via InfiniBand
IMB Ping-Pong: Latency

Latency [µs]

IB internode | IB intranode 2S | IB intranode 1S

0.32 | 0.62 | 0.24

Chipset | Memory

Affinity matters!

IMB Ping-Pong: Bandwidth Characteristics

Intra-node vs. Inter-node

Shared cache advantage

PCle 8x and DDR unid. IB limit

Between two cores of one socket

Between two sockets of one node

Between two nodes via InfiniBand

Affinity matters!

Hybrid MPI and OpenMP Parallel Programming
Tutorial M09 at SC’08, Austin, Texas, USA, Nov. 17, 2008
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

```c
#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

Chipset
Memory

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!


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

```c
#include <plpa.h>
...
#pragma omp parallel
{
#pragma omp critical
{
  if(PLPA_NAME(api_probe)() != PLPA_PROBE_OK) {
    cerr << "PLPA failed!" << endl; exit(1);
  }

  plpa_cpu_set_t msk;
  PLPA_CPU_ZERO(&msk);
  int cpu = omp_get_thread_num();
  PLPA_CPU_SET(cpu, &msk);
  PLPA_NAME(sched_setaffinity)(pid_t)0, sizeof(cpu_set_t), &msk);
}
```

Pinning available?

Which CPU to run on?

Pin "me"
**Process/Thread Binding With PLPA**

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

```c
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:

```c
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 nonzeros (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 \( c(\cdot) = m(:, :) \ast b(\cdot) \)
- Operation count = \( 2N_{nz} \)

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

- Long inner loop (max. \( N_z \)): 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

Row-wise distribution

Each processor: local JDS (shift&order)

Avoid mixing of local and non-local diagonals:
1. Shift within local subblock
2. Fill local subblock with non-local elements from the right
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 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 your application

Examples:

<table>
<thead>
<tr>
<th>Benefit through hybrid (see next section)</th>
<th>No.1</th>
<th>No.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loss by mismatch problems</td>
<td>-10%</td>
<td>-25%</td>
</tr>
<tr>
<td>Total</td>
<td>+20%</td>
<td>-15%</td>
</tr>
</tbody>
</table>

In most cases: Both categories!

The Topology Problem with Application example on 80 cores:

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

17 x inter-node connections per node
1 x inter-socket connection per node

Does it matter?
The Topology Problem with

Application example on 80 cores:
- Cartesian application with 5 x 16 = 80 sub-domains
- On system with 10 x dual socket x quad-core

Two levels of domain decomposition

Never trust the default !!!
The Topology Problem with **pure MPI**

Application example on 80 cores:
- Cartesian application with $5 \times 16 = 80$ sub-domains
- On system with $10 \times$ dual socket $\times$ quad-core

- One MPI process on each CPU

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

---

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

Exa.: 2 SMP nodes, 8 cores/node

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

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

Successful examples:
- Multi-Zone NAS Parallel Benchmarks (MZ-NPB)
The Topology Problem with hybrid MPI+OpenMP

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

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

+ 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

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

Hybrid MPI and OpenMP Parallel Programming
Tutorial M09 at SC’08, Austin, Texas, USA, Nov. 17, 2008
Unnecessary intra-node communication

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

Sleeping threads and network saturation

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
### 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 controllers – increases memory capacity and bandwidth

---

### Overlapping Communication and Computation

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**

```cpp
if (my_thread_rank < 1) {
    MPI_SendRecv....
} 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)*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,

#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 */
  }  /* work based on halo information */
}  /*end omp parallel */

Parallel Programming Models on Hybrid Platforms

- pure MPI
  one MPI process on each CPU
- hybrid MPI+OpenMP
  MPI: inter-node communication
  OpenMP: inside of each SMP node
- OpenMP only
  distributed virtual shared memory

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

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

- Masteronly
  MPI only outside of parallel regions
- Multiple/only
  - appl. threads
  - inside of MPI

- Funneled & Reserved
  reserved thread for communication
- Funneled with Full Load Balancing
- Multiple & Reserved
  reserved threads for communication
- Multiple with Full Load Balancing

Different strategies to simplify the load balancing
Hybrid Parallel Programming

Performance ratio (theory)

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

- \( r > 1 \) funneled & reserved is faster
- \( r < 1 \) masteronly is faster

Good opportunity of funneled & reserved:
\[ \epsilon_{\text{max}} = 1 + m(1 - 1/n) \]

Small risk of funneled & reserved:
\[ \epsilon_{\text{min}} = 1 - m/n \]

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

Experiment: Matrix-vector-multiply (MVM)

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

Experiment: Matrix-vector-multiply (MVM)

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


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
**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.

**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 too small
  - Significant latency
  - Communication not oriented on boundaries of a domain decomposition
  - probably more data must be transferred than necessary

*by rule of thumb: Communication may be 10 times slower than with MPI*
Comparing results with heat example

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

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

![Graph showing speedup vs. threads for different grid sizes.]

- Cluster OpenMP on a Dual-Xeon cluster

heats2_x.c with Cluster OpenMP on NEC dual Xeon EM64T cluster

![Graph showing speedup vs. nodes for different grid sizes.]

Efficiency only with small communication foot-print

Up to 3 CPUs with 3000x3000

No speedup with 1000x1000

Second CPU only usable in small cases

Terrible with non-default schedule
Back to the mixed model – an Example

• Topology-problem solved:
  - Only horizontal inter-node comm.
  - Still intra-node communication

• 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)

- 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 < Max workload of a zone
  - Inner loop:
    - OpenMP parallelized (static schedule)
    - Not suitable for distributed memory parallelization

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

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

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

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

- MPI & OpenMP
  - Simple static load-balancing on MPI level, dynamic or guided on OpenMP level
  } medium quality cheap implementation
Memory consumption

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

- MPI & OpenMP
  With n threads per MPI process:
  - Duplicated data 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 domains inside of SMP node

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
      - CC numa 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:

  ```c
  int MPI_Init_thread( int * argc, char ** argv[],
                      int thread_level_required,
                      int * thread_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, but only master thread will make MPI-calls
  - `MPI_THREAD_FUNNELED`: Only master thread will make MPI-calls AND only while other threads are sleeping
  - `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!

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

```c
!$OMP PARALLEL #pragma omp parallel
!$OMP DO #pragma omp for nowait
do i=1,1000
  a[i] = buf[i];
end do
!$OMP END DO NOWAIT #pragma omp barrier
!$OMP MASTER #pragma omp master
call MPI_RECV(buf,...) MPI_Recv(buf,...);
!$OMP END MASTER #pragma omp barrier
!$OMP DO #pragma omp for nowait
do i=1,1000
  c[i] = buf[i];
end do
!$OMP END DO NOWAIT #pragma omp barrier
!$OMP END PARALLEL /* 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 runnable 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,Halfed 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:

<table>
<thead>
<tr>
<th>Communicator Type</th>
<th>List Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_COMM_WORLD</td>
<td></td>
</tr>
<tr>
<td>Duplicated MPI_COMM_WORLD</td>
<td>Reversed MPI_COMM_WORLD</td>
</tr>
<tr>
<td>Odd-/Even Split MPI_COMM_WORLD</td>
<td>Halved MPI_COMM_WORLD</td>
</tr>
<tr>
<td>Zero-and-Rest Intercommunicator</td>
<td>Halved Intercommunicators</td>
</tr>
<tr>
<td>Two-dimensional Cartesian</td>
<td>Three-dimensional Cartesian</td>
</tr>
<tr>
<td></td>
<td>Fully-connected Topology</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Testsuite – Implemented Datatypes

- List of implemented Datatypes:

<table>
<thead>
<tr>
<th>Datatype Type</th>
<th>List Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Predefined MPI1 C-types… (15)</td>
<td>All Predefined MPI2 C-types… (+5)</td>
</tr>
<tr>
<td>Reduce types for std. communication (5)</td>
<td></td>
</tr>
<tr>
<td>Derived Types Contiguous Memory</td>
<td>Derived Types w/ Holes</td>
</tr>
<tr>
<td>Duplicated MPI_Char (MPI2)</td>
<td>Duplicated Derived w/ Holes &amp; LB/UB</td>
</tr>
<tr>
<td>Duplicated Intercommunicators</td>
<td></td>
</tr>
<tr>
<td>Intracomm merged of Halved Intercomms</td>
<td></td>
</tr>
<tr>
<td>Three-dimensional Cartesian</td>
<td></td>
</tr>
<tr>
<td>Two-dimensional Cartesian</td>
<td></td>
</tr>
</tbody>
</table>
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):
  
<table>
<thead>
<tr>
<th>MIN of Type</th>
<th>MAX of Type</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00 0x00 0x00 0x80</td>
<td>0xFF 0xFF 0xFF 0x7F</td>
<td>0xA5</td>
</tr>
</tbody>
</table>

  4B Min Integer 4B Max Integer 1 Byte Hole

- E.g. the following derived datatype `MPI_TYPE_MIX_LB_UB`:
  
<table>
<thead>
<tr>
<th>1B Char</th>
<th>4B Int</th>
<th>4B Float</th>
</tr>
</thead>
<tbody>
<tr>
<td>8B Double</td>
<td>4B Long</td>
<td>2B Short</td>
</tr>
<tr>
<td>MIN</td>
<td>MPI_LB</td>
<td>MPI_UB</td>
</tr>
<tr>
<td>MAX</td>
<td>Zero Position</td>
<td>8B Double</td>
</tr>
</tbody>
</table>

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)

  ![Simple Ring](image)

  e.g. Simple Ring (w/ different tag)

  Process 0

  Process 1

  - Scheduling different tests to different threads:

    ![Bcast, Scatter, Gather](image)

    e.g. Bcast, Scatter, Gather
    (on the same comm?)

    Process 0

    Process 1
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?

  ![Diagram](image)

  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:

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Thread support level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPICH-1.2.7p1</td>
<td>Always announces MPI_THREAD_FUNNELED</td>
</tr>
<tr>
<td>MPICH2-1.0.6</td>
<td>ch3:sock supports MPI_THREAD_MULTIPLE</td>
</tr>
<tr>
<td></td>
<td>ch:nemesis has “Initial Thread-support”</td>
</tr>
<tr>
<td>MPICH2-1.1.0a1</td>
<td>ch3:sock (default) supports MPI_THREAD_MULTIPLE</td>
</tr>
<tr>
<td>Intel MPI 3.1</td>
<td>Full MPI_THREAD_MULTIPLE</td>
</tr>
<tr>
<td>SciCortex MPI</td>
<td>MPI_THREAD_FUNNELED</td>
</tr>
<tr>
<td>HP MPI-2.2.7</td>
<td>Full MPI_THREAD_MULTIPLE (with libmtmpi)</td>
</tr>
<tr>
<td>SGI MPT-1.14</td>
<td>Not thread-safe?</td>
</tr>
<tr>
<td>IBM MPI</td>
<td>Full MPI_THREAD_MULTIPLE</td>
</tr>
<tr>
<td>Nec MPI/SX</td>
<td>MPI_THREAD_SERIALIZE</td>
</tr>
</tbody>
</table>

- 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:
  ```bash
  configure --enable-mpi-threads
  ```

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

  ```bash
  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 cost, by adding mutex-locks/unlocks
- Measurement for Shared-Memory communication using Netpipe

### Measurements

- **NetPipe-3.7.1**
  - MPIch2 1.1.0a1
  - Open MPI R19511
  - HP-MPI 2.2.7

<table>
<thead>
<tr>
<th></th>
<th>Plain</th>
<th>With Threads &amp; MPI_THREAD_MULTIPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No progress-threads</td>
<td>No progress-threads</td>
</tr>
<tr>
<td>MPIch2 1.1.0a1</td>
<td>0.24μs</td>
<td>0.29μs</td>
</tr>
<tr>
<td>Open MPI R19511</td>
<td>0.09μs</td>
<td>0.85μs</td>
</tr>
<tr>
<td>HP-MPI 2.2.7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Latency (NetPipe-3.7.1)</th>
<th>1027</th>
<th>2025</th>
<th>830</th>
<th>1487</th>
<th>3170</th>
<th>2829</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ø Instructions per Send+Recv</td>
<td>1027</td>
<td>2025</td>
<td>830</td>
<td>1487</td>
<td>3170</td>
<td>2829</td>
</tr>
<tr>
<td>Ø Cycles per Send+Recv</td>
<td>785</td>
<td>1508</td>
<td>462</td>
<td>1312</td>
<td>4379</td>
<td>1749</td>
</tr>
<tr>
<td>Ø Branches per Send+Recv</td>
<td>203</td>
<td>367</td>
<td>264</td>
<td>340</td>
<td>611</td>
<td>677</td>
</tr>
</tbody>
</table>

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

- 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

<table>
<thead>
<tr>
<th>ID</th>
<th>Short Dev.</th>
<th>Severity</th>
<th>Context</th>
<th>Description</th>
<th>1st Acc.</th>
<th>2nd Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Write</td>
<td>Error</td>
<td>pthread</td>
<td>Memory write of global_variable at: &quot;pthread: &quot;</td>
<td>pthread: &quot;</td>
<td>pthread:</td>
</tr>
<tr>
<td></td>
<td>da</td>
<td></td>
<td>ad_race</td>
<td>&quot;pthread_race.c&quot;:31 conflicts with d_race.d_race:</td>
<td>d_race.d_race:</td>
<td>d_race.d_race:</td>
</tr>
<tr>
<td></td>
<td>ra-race</td>
<td></td>
<td>ee.c*:12</td>
<td>a prior memory write of</td>
<td>ee.c*:31</td>
<td>ee.c*:31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>global_variable at</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>pthread_race.c&quot;:31 (output</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(dependence)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Thread Correctness – Intel ThreadChecker 2/3

- One may output to HTML:

  \texttt{tcheck\_cl --format HTML --report pthread\_race.html pthread\_race}

- Options:

  \texttt{--options 'file=tc\_mpi\_test\_suite\_%H\_%I, IB}

  --enable-mca-no-build=memory-ptmalloc2

  CC=icc F77=ifort FC=ifort

  --cache\_dir '/tmp/hpcraink\_$$\_tc\_cl\_cache' \(./\)mpi\_test\_suite-j 2 -r FULL -t 'Ring Ibsend' -d MPI\_INT

  --reinstrument-u full --format html

Thread Correctness – Intel ThreadChecker 3/3

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

  \begin{verbatim}
  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’
  LDFLAGS=’-debug all -tcheck’
  \end{verbatim}

- Then run with:

  \begin{verbatim}
  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
  \end{verbatim}
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”

Indication of non-optimal load balance.
Kojak – Example “Wait at Barrier”, Solution

Better load balancing with dynamic loop schedule

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
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 → f×x% loss
  - If loss is small x=1%
    and factor f=3 is medium
    → don’t worry ?!
- Optimization
  - 1 MPI process per core .................................................. per SMP node
    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

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
**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.

---

**Exa. 1**

<table>
<thead>
<tr>
<th>Node 0</th>
<th>Node 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barrier 0</td>
<td>Barrier 0</td>
</tr>
<tr>
<td>Twin(A)</td>
<td>Twin(A)</td>
</tr>
<tr>
<td>Barrier 1</td>
<td>Barrier 1</td>
</tr>
<tr>
<td>WriteNotice(A)</td>
<td>Writtenotice(A)</td>
</tr>
<tr>
<td>Diffreq_A_1(0:1)-&gt;&lt;Diff_A_1(0:1)</td>
<td></td>
</tr>
<tr>
<td>Apply diffs</td>
<td></td>
</tr>
<tr>
<td>Barrier 2</td>
<td>Barrier 2</td>
</tr>
<tr>
<td>WriteNotice(A)</td>
<td>WriteNotice(A)</td>
</tr>
</tbody>
</table>
Exa.

Node 0          Node 1          Node 2
Barrier 0       Barrier 0       Barrier 0
Twin(A)         Twin(A)
Barrier 1       Barrier 1       Barrier 1
WriteNotice(A)   WriteNotice(A)
Barrier 2       Barrier 2       Barrier 2
WriteNotice(A)   WriteNotice(A)
Barrier 3       Barrier 3       Barrier 3
WriteNotice(A)   WriteNotice(A)
Barrier 4       Barrier 4       Barrier 4
WriteNotice(A)   WriteNotice(A)

Exa. 3 (start)

Node 0          Node 1          Node 2          Node 3
Barrier 0       Barrier 0       Barrier 0       Barrier 0
Twin(A)         Twin(A)
Barrier 1       Barrier 1       Barrier 1       Barrier 1
WriteNotice(A)   WriteNotice(A)
Barrier 2       Barrier 2       Barrier 2       Barrier 2
WriteNotice(A)   WriteNotice(A)
Barrier 3       Barrier 3       Barrier 3       Barrier 3
WriteNotice(A)   WriteNotice(A)
Barrier 4       Barrier 4       Barrier 4       Barrier 4
WriteNotice(A)   WriteNotice(A)

Courtesy of J. Cownie, Intel
These examples may give an impression of the overhead induced by the Cluster OpenMP consistency protocol.

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


References

- Rolf Rabenseifner,  
  Hybrid Parallel Programming on HPC Platforms.  
  In proceedings of the Fifth European Workshop on OpenMP, EWOMP '03,  

- 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,  

- 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,  

- 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,  

- 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


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


- 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*
  http://www.fz-juelich.de/zam/kojak/documentation/publications/

• Felix Wolf and Bernd Mohr,
  *Automatic performance analysis of hybrid MPI/OpenMP applications*
  http://www.fz-juelich.de/zam/kojak/documentation/publications/

• Felix Wolf and Bernd Mohr,
  *Automatic Performance Analysis of Hybrid MPI/OpenMP Applications*
  http://www.fz-juelich.de/zam/kojak/documentation/publications/

• Frank Cappello and Daniel Etemble,
  *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

• 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
Further references


Further references

