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 Keller@hlrs.de

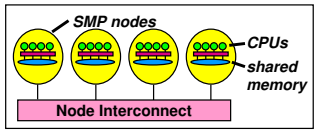
[1)] High Performance Computing Center (HLRS), University of Stuttgart, Germany
[2)] Regional Computing Center (RRZE), University of Erlangen, Germany
[3)] Sun Microsystems, USA

Tutorial at EuroPVM/MPI 2006
Bonn, Germany, Aug. 17–20

Outline

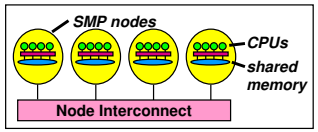
Motivation

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



- Hardware range
  - **mini-cluster with dual-core CPUs**
  - **…**
  - **large constellations with large SMP nodes**
- Hybrid MPI/OpenMP programming seems natural
  - **MPI between the nodes**
  - **OpenMP inside of each SMP node**
- Often hybrid programming **slower** than pure MPI
  - **Examples, Reasons, …**

Motivation



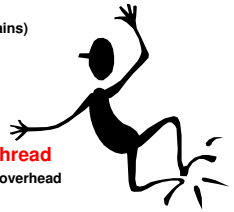
- Using the communication bandwidth of the hardware
- Minimizing synchronization = idle time

} **optimal usage of the hardware**

- Appropriate parallel programming models / Pros & Cons

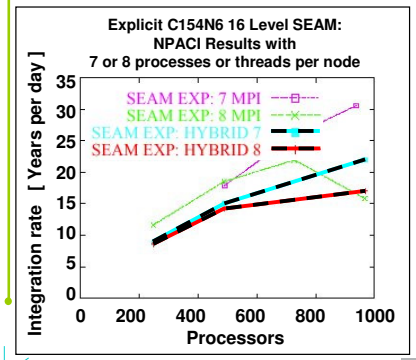
But results may surprise!

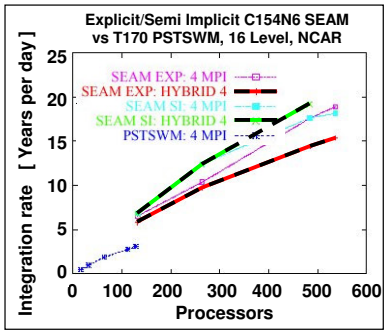
- Example code - HYDRA
- Domain-decomposed hydrodynamics
  - (almost) independent mesh domains with ghost cells on boundaries
  - ghost cells communicate boundary information ~40-50 times per cycle
- Parallelism model: single level
  - MPI divides domains among compute nodes
  - OpenMP further subdivides domains among processors
  - domain size set for cache efficiency
    - **minimizes memory usage, maximizes efficiency**
    - **scales to very large problem sizes (>10^7 zones, >10^3 domains)**
- Results:
  - **MPI** (256 proc.) **~20% faster than MPI / OpenMP** (64 nodes x 4 proc./node)
  - domain-domain communication not threaded, i.e., **MPI communication is done only by main thread**
    - **accounts for ~10% speed difference, remainder in thread overhead**



Example from SC

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





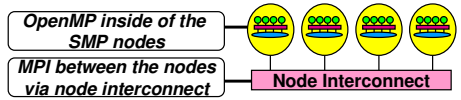
Figures: Richard D. Loft, Stephen J. Thomas, John M. Dennis:
Terascale Spectral Element Dynamical Core for Atmospheric General Circulation Models.
Proceedings of SC2001, Denver, USA, Nov. 2001.
http://www.sc2001.org/papers/pap.pap189.pdf
Fig. 9 and 10.

Outline

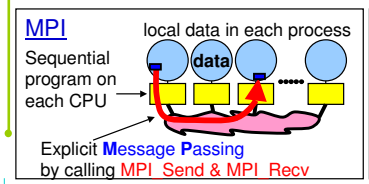
- Introduction / Motivation
- **Programming models on clusters of SMP nodes**
- Case Studies / pure MPI vs. hybrid MPI+OpenMP
- Mismatch Problems
- Thread-safety quality of MPI libraries
- Case Studies / pure OpenMP
- Summary

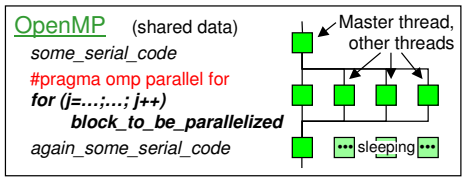
Major Programming models on hybrid systems

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

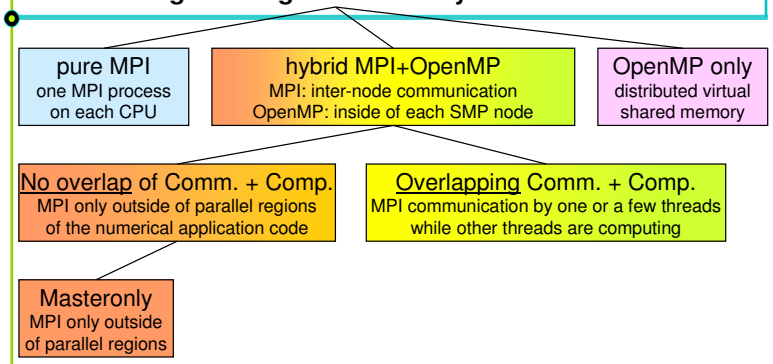


- Other: Virtual shared memory systems, HPF, …
- Often **hybrid programming (MPI+OpenMP)** slower than **pure MPI**
  - why?





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!

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

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

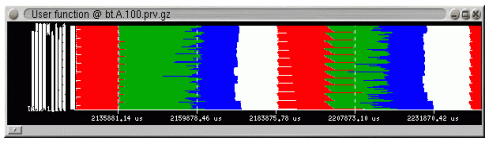
- Distributed shared virtual memory system needed
- Must support clusters of SMP nodes
- e.g., Intel® Cluster OpenMP
  - Shared memory parallel inside of SMP nodes
  - Communication of modified parts of pages at OpenMP flush (part of each OpenMP barrier)

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

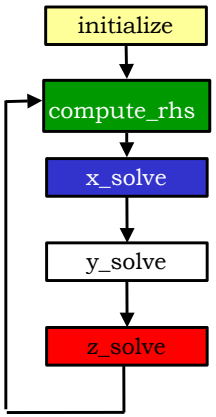
Outline

A Computational Fluid Dynamics (CFD) Benchmark

- The NAS Parallel Benchmark BT:
  - Simulated CFD application
  - Uses ADI method to solve Navier-Stokes equations in 3D
  - Decouples the three spatial dimensions
  - Solves a tri-diagonal system of equation in each dimension
- Compare 5 different implementations:
  - Message Passing based on MPI
  - OpenMP
  - Hybrid MPI/OpenMP (2 Versions)

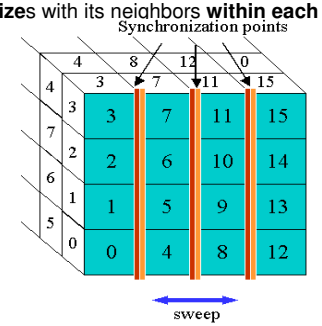


Paraver Timeline View for some iterations of BT



MPI Based Parallelization Strategy

- As in the benchmark distribution NPB3.2:
  - 3D Multi-partition Scheme
  - Each process receives multiple blocks
  - For each sweep directions all processes can start their work in parallel
  - **Exchange** of boundary data **before each iteration**
  - Each process **synchronizes** with its neighbors **within each sweep**



(Nested) OpenMP Parallelization

- Add OpenMP directives to 2 outermost loops within the time consuming routines
  - Outer level as in NPB3.2

Nested OpenMP

```
!$omp parallel do
  do k= 1, nz
!$omp parallel do
   do j= 1, ny
    do i=1,nx
      .. = u(i,j,k-1)
           + u(i,j,k+1)
    enddo
   enddo
  enddo
```


Hybrid MPI/OpenMP Parallelization (V1)

- MPI: 1D data distribution in z-dimension (k-loop).
- OpenMP: directives in y-dimension (j-loop).

```
!$omp parallel
 do k=k_low,k_high
   synchronize neighbor threads
!$omp do
   do j=1,ny
    do i=1,nx
      rhs(i,j,k) = rhs(i,j-1,k)
                 + ...
    enddo
   enddo
   synchronize neighbor threads
 enddo
!$omp end parallel
```

```
!$omp parallel do
  do j=1,ny
    call receive
    do k=k_low,k_high
      do i=1,nx
        rhs(i,j,k) = rhs(i,j,k-1)
                   + ...
      enddo
    enddo
    call send
   enddo
```


Hybrid MPI/OpenMP (V2)

- 3D Multi-partition scheme as in NPB3.2
- Add OpenMP directives to outermost loop in time consuming routines.
- MPI/OpenMP (2) without OpenMP <=> BT MPI
  - Differences to MPI/OpenMP (1):
    - 3D Data Decomposition.
    - MPI and OpenMP employed in same dimension.
    - All communication occurs outside of parallel regions.

```
 do ib = 1, nblock
 call receive
!$omp parallel do
  do j=j_low,j_high
    do i=i_low,i_high
      do k=k_low,k_high
      rhs(i,j,k,ib)=
           rhs(i,j,k-1,ib)+ . . .
      enddo
    enddo
   enddo
  call send
 end do
```


Testbed Configurations

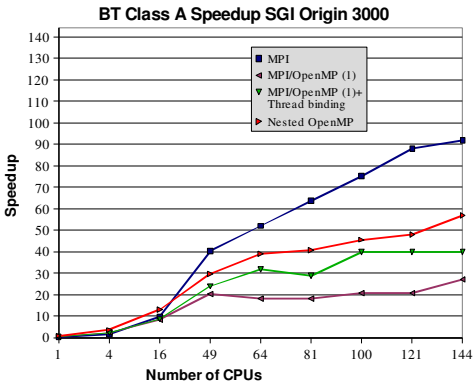
- Gigabit Ethernet (GE) MPI:
  - 100 us latency
  - 100 MB/s bandwidth
- Sun Fire Link (SFL) MPI:
  - 4 us latency
  - 2GB/s bandwidth
- 4 Sun Fire 6800 connected by GE
  - 96 (4x24) CPU total
- 4 Sun Fire 6800 connected by SFL
  - 96 (4x24) CPUs total
- 1 Sun Fire 15K node
  - 72 (1x72) CPUs total
- SGI Origin 3000
  - 512 CPUs
  - Type R12000
  - 400 MHz
  - 4 CPUs per node
  - 256GB of main memory (2GB per node)
  - 8MB L2 cache
  - 0.8 Gflops peak performance per CPU
  - Compiler:
    - MIPSpro 7.4 Fortran for hybrid codes
    - MIPSpro 7.4 Fortran + Nanos Compiler for nested OpenMP
    - Always use *-mp –O3 –64*

Hardware Details Sun Fire Cluster

- **UltraSPARC-III Cu processors**
  - Superscalar 64-bit processor
  - 900 MHz
  - L1 cache (on chip) 64KB data and 32KB instructions
  - L2 cache (off chip) 8 MB for data and instructions
- Sun Fire 6800 node:
  - 24 UltraSPARC-III 900 MHz CPU
  - 24 GB of shared main memory
  - Flat memory system: approx. 270 ns latency, 9.6 GB/s bandwidth
- Sun Fire 15K node:
  - 72 UltraSPARC-III 900 MHz CPU
  - 144 GB of shared main memory
  - NUMA memory system: Latency 270 ns onboard to 600 ns off board Bandwidth 173GB/s on board to 43 GB (worst case)
- Located at RWTH Aachen

Results SGI Origin 3000

- Problem size:
  - 64x64x64 Points
- Speed-up:
  - Measured against time of fastest implementation on 1 CPU (OpenMP)
- For multilevel versions the best time of the nesting combination is reported.



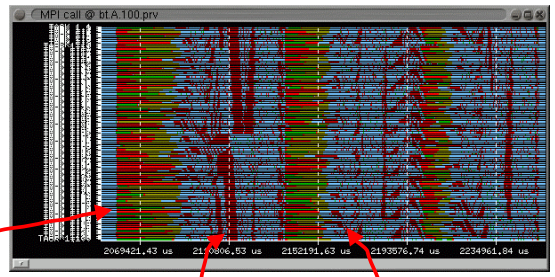
Paraver Timeline View of BT-MPI (100 CPUs)

100 MPI Processes

Vertical axis displays process ID

Horizontal axis displays time

Colours indicate time in communication



Very long MPI_Isends (red) and MPI_Irecvs (green)

Sometime very long MPI_Wait (dark red) ...

...sometimes not so long

Flow of Useful Computations of BT-Hybrid (V1) (100 CPUs)

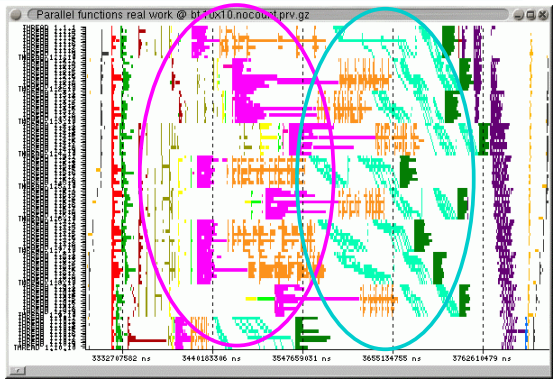
10 Processes running on 10 threads each

Vertical axis displays thread ID

Horizontal axis displays time

Colors indicate different parallel regions

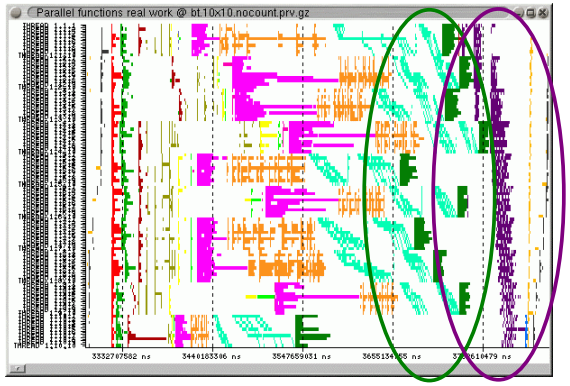
White indicate non-useful time



y_solve:

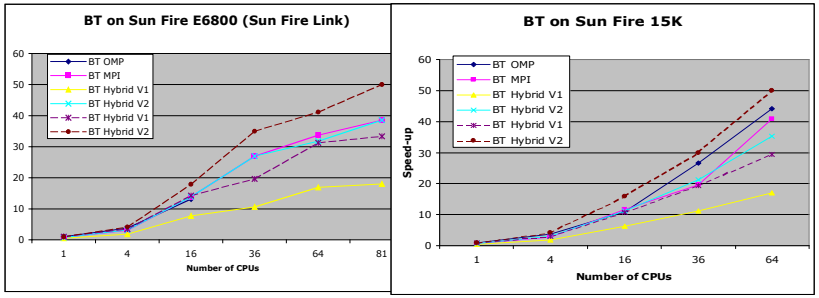
- Contains 2 parallel regions
- Pipelined thread execution within each process in second parallel region

Paraver Timeline View of Hybrid (1) (100 CPUs)



z_solve:
- Contains communication within parallel regions
- 1 dimensional pipeline across all processes

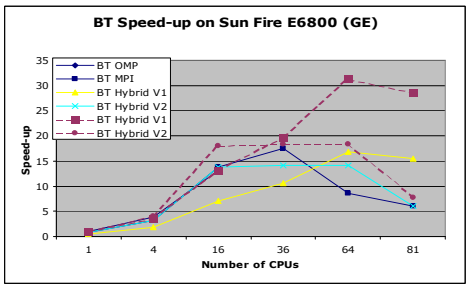
Sun Fire using Fast Networks



- BT MPI shows best scalability
- BT Hybrid V1:
  - Best performance for 16 MPI processes and 4 or 5 thread
- Solid lines indicate speed-up compared to best implementation on 1 CPU (BT-OMP), dashed lines indicate the speed-up compared to 1 CPU of the same implementation
- BT Hybrid V2:
  - Best performance with only 1 thread per MPI process:
  - No benefit employing hybrid programming paradigm

BT Class A (64x64x64 Grid Points) using GE

- Performance using Gigabit Ethernet (GE):
  - Hybrid implementations outperform pure MPI implementation
  - BT Hybrid V1: shows best scalability
  - BT Hybrid V1: best performance employing 16 MPI processes and 4 or 5 threads respectively:
    - Tight interaction between MPI and OpenMP limits number of threads that can be used efficiently
  - BT Hybrid V2 achieves best performance using 4 MPI processes employing 16 threads each:
    - Large messages saturate slow network and limit number of MPI processes that can be used efficiently



Characteristics of Hybrid Codes

- **BT Hybrid V1:**
  - Message exchange with 2 neighbour processes
  - Many short messages
  - Length of messages remain the same
  - Increase number of threads:
    - Increase of OpenMP barrier time (threads from different MPI processes have to synchronize)
    - Increase of MPI time (MPI calls within parallel regions are serialized)
- **BT Hybrid V2**:
  - Message exchange with 6 neighbour processes
  - Few long messages
  - Length of messages decreases with increasing number of processes
  - Increase number of threads:
    - Increase of OpenMP barrier time

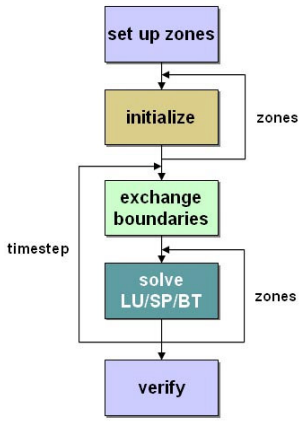
Observation on fast networks:

- Single Level MPI:
  - Best performance, best scalability
  - Coarse-grained well balanced distribution and scheduling of work
- Hybrid MPI/OpenMP V2 did not yield performance advantage
- Hybrid MPI/OpenMP V1:
  - Implementation non-typical: pipelined thread execution, communication within parallel regions.
  - Low percentage of useful thread work time:
    - 1D data distribution limits parallelism on coarse grain
    - OpenMP introduces extra synchronization overhead at the end of parallel regions
    - Interaction of OpenMP and MPI yields thread pre-emption and thread migration
  - Performance improves through explicit binding.

Observation on slow networks:

- Hybrid MPI/OpenMP V1 showed better performance than V2 or pure MPI:
  - Message exchange with only 2 neighbours vs 6 neighbours
  - Many short messages vs few longer messages:
    - BT V1 4x16: 14880 send, avg. length 10600 bytes
    - BT V2 4x16: 960 send, avg. length 116360 bytes
  - Long messages sent by many MPI processes potentially saturate a slow network quickly.

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
- Nested OpenMP based on NanosCompiler extensions was developed for this study

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
```

Using MLP

```
call omp_set_numthreads (weight)
do step = 1, itmax
  call exch_qbc(u, qbc, nx,…)
     do i = 1, n
        sh_buf(i) = u(i)
     end do
     call mlp_barrier
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
```


Using Nested OpenMP

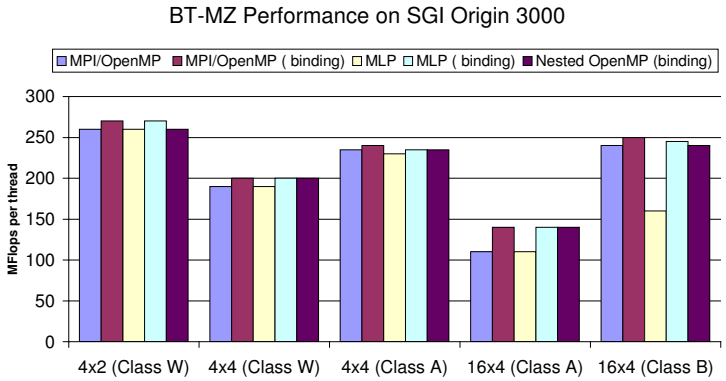
```
call omp_set_numthreads (weight)
do step = 1, itmax
  call exch_qbc(u, qbc, nx,…)
!$OMP PARALLEL
!$OMP& PRIVATE(iam, zone,...)
!$OMP& NUM_THREADS(num)
 iam = omp_get_thread_num()
do zone = 1, num_zones
    if (iam .eq. pzone_id(zone)) then
        call ssor(u,rsd,…)
      end if
    end do
!$OMP END PARALLEL

end do
  ...
```

```
 subroutine ssor(u, rsd,…)
  ...
!$OMP PARALLEL DEFAUL(SHARED)
!$OMP& PRIVATE(m,i,j,k...)
!$OMP&
  NUM_THREADS(weight(iam))
  do k = 2, nz-1
!$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
```

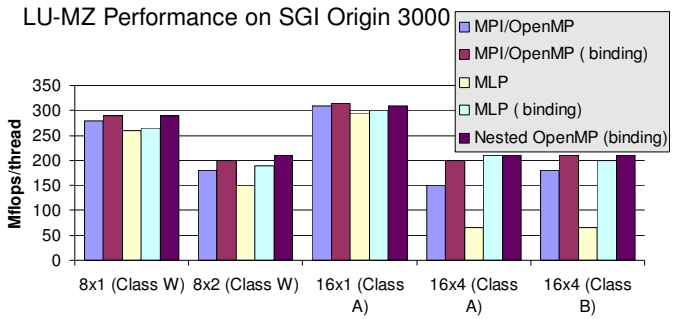

Benchmark Characteristics

- Aggregate sizes:
  - Class W: aggregate 64x64x8 grid points
  - Class A: aggregate 128x128x16 grid points
  - Class B: aggregate 304x208x17 grid points
- BT-MZ:
  - #Zones: 16 (Class W), 16 (Class A), 64 (Class B)
  - Size of the zones varies widely:
    - **large/small ≈ 20**
    - **requires multi-level parallelism to achieve a good load-balance**
- LU-MZ:
  - #Zones: 16 (Class W), 16 (Class A), 16 (Class B)
  - Size of the zones identical:
    - **no load-balancing required**
    - **limited parallelism on outer level**
- SP-MZ:
  - #Zones: 16 (Class W), 16 (Class A), 64 (Class B)
  - Size of zones identical

Performance of BT-MZ on SGI Origin 3000



Performance of LU-MZ on SGI Origin 3000

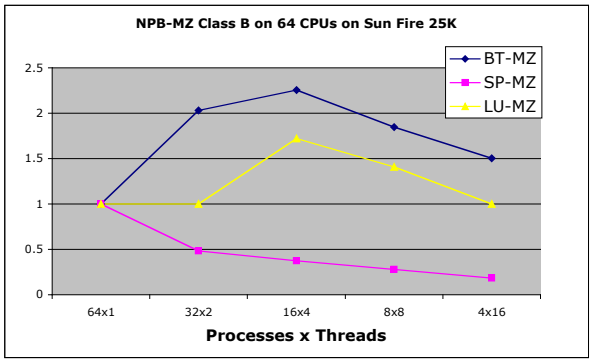


Comparison of the different implementations

- Thread binding improves performance for all implementations
- Little performance difference between the different implementations
- Which paradigm is best for NPB-MZ?

	MPI +OpenMP	MLP	Nested OpenMP
Ease of use	**difficult**	**medium**	**easy**
Performance	**good**	**good**	**good**
Portability	**shared and SMP Clusters**	**shared only**	**shared only**

NAS NPB-MZ on Sun Fire 72 US-IV+ 25K



Combining Processes and Threads

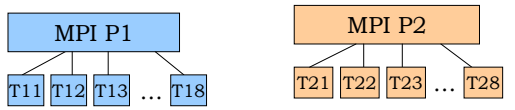
- SP-MZ runs fastest when using as many processes as possible on the outer level.
- LU-MZ the number of MPI processes that can be used is very small. Threads are necessary to exploit extra parallelism
- BT-MZ can not achieve a good load balance on the MPI level. Threads are necessary to counter balance the uneven workload distribution.
- Thread binding is essential when running hybrid codes on cc-NUMA architectures.

Hybrid code on cc-NUMA architectures

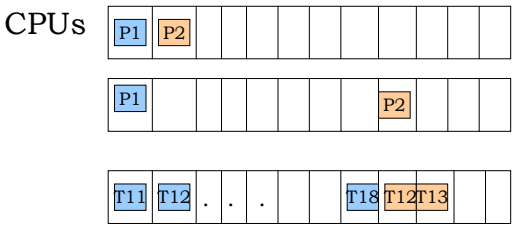
MPI:

- Initially not designed for NUMA architectures or mixing of threads and processes
- API does not provide support for memory/thread placement
- Vendor specific APIs to control thread and memory placement

Thread binding and memory placement



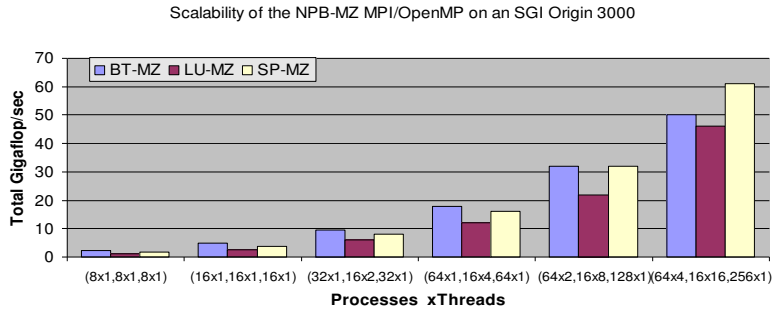
Space MPI Processes such that threads from the same process can run on adjacent CPUs!



Not good

Good

Scalability of the Multi-Zone Benchmarks



- Thread binding and efficient placement was used for all runs
- Benchmarks show good scalability
- Low communication overhead:
  - 1-10% depending on benchmark and number of MPI processes

Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / pure MPI vs. hybrid MPI+OpenMP
- **Mismatch Problems**
- Thread-safety quality of MPI libraries
- Case Studies / pure OpenMP
- Summary

Mismatch Problems

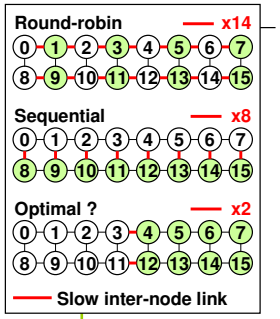
- **Topology problem** [with pure MPI]
- **Unnecessary intra-node communication** [with pure MPI]
- **Inter-node bandwidth problem** [with hybrid MPI+OpenMP]
- **Sleeping threads and saturation problem** [with masteronly] [with pure MPI]
- **Additional OpenMP overhead** [with hybrid MPI+OpenMP]
  - Thread startup / join
  - Cache flush (data source thread – communicating thread – sync. → flush)
- **Overlapping communication and computation** [with hybrid MPI+OpenMP]
  - an application problem → separation of local or halo-based code
  - a programming problem → thread-ranks-based vs. OpenMP work-sharing
  - a load balancing problem, if only some threads communicate / compute
- **Communication overhead with DSM** [with pure (Cluster) OpenMP]

➔ **no silver bullet, i.e., each parallelization scheme has its problems**

The Topology Problem with pure MPI

one MPI process on each CPU

Exa.: 2 SMP nodes, 8 CPUs/node



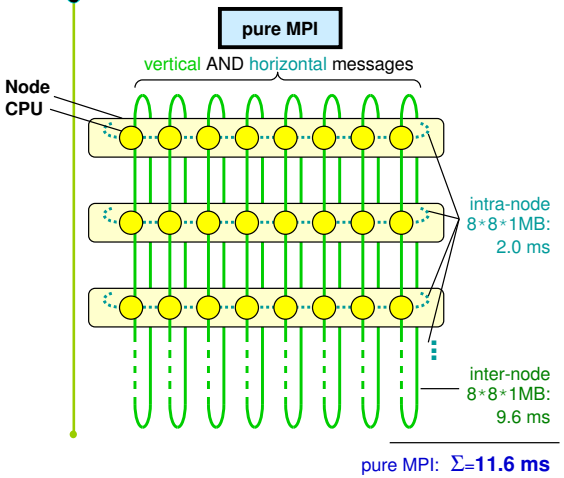
Problems
- To fit application topology on hardware topology

Solutions for Cartesian grids:
- E.g. choosing ranks in MPI_COMM_WORLD ???
  - **round robin (rank 0 on node 0, rank 1 on node 1, ...)**
  - **Sequential (ranks 0-7 on 1st node, ranks 8-15 on 2nd ...)**

… in general
- load balancing in two steps:
  - **all cells among the SMP nodes (e.g. with ParMetis)**
  - **inside of each node: distributing the cells among the CPUs**
- or …

➔ **using hybrid programming models**

Unnecessary intra-node communication



Alternative:
- Hybrid MPI+OpenMP
- No intra-node messages
- Longer inter-node messages
- **Really faster ???????**
  (… wait 2 slides)

Timing:
Hitachi SR8000, MPI_Sendrecv
8 nodes, each node with 8 CPUs

Programming Models on Hybrid Platforms: Hybrid Masteronly

Masteronly
MPI only outside of parallel regions

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

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

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

Problems
- MPI-lib must support MPI_THREAD_FUNNELED

Disadvantages
- do we get full inter-node bandwidth? **... next slide**
- all other threads are sleeping while master thread communicates

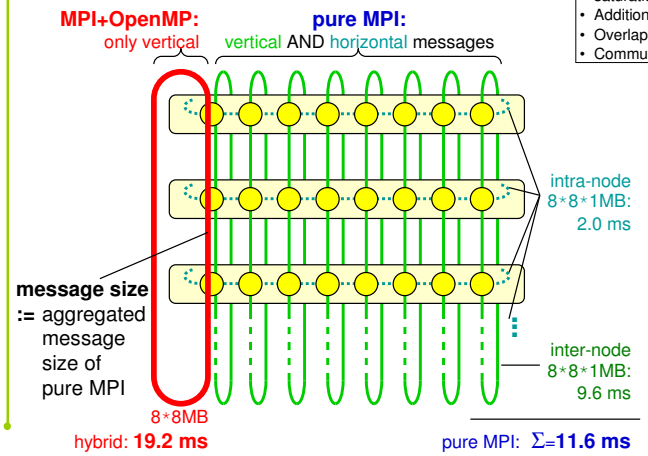
➔ **Reason for implementing overlapping of communication & computation**

pure MPI
Masteronly

Experiment: Orthogonal parallel communication

Mismatch Problems
- Topology problem
- Unnecessary intra-node comm.
- ➢ **Inter-node bandwidth problem**
- Sleeping threads and saturation problem
- Additional OpenMP overhead
- Overlapping comm. and comp.
- Communication overhead w. DSM

Hitachi SR8000
- 8 nodes
- each node with 8 CPUs
- MPI_Sendrecv

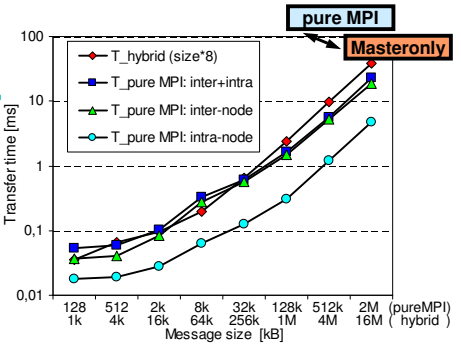


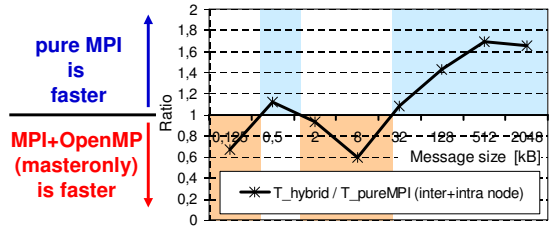
→ **1.6x slower** than with pure MPI, **although**
- only half of the transferred bytes
- and less latencies due to 8x longer messages

pure MPI
Masteronly

Results of the experiment

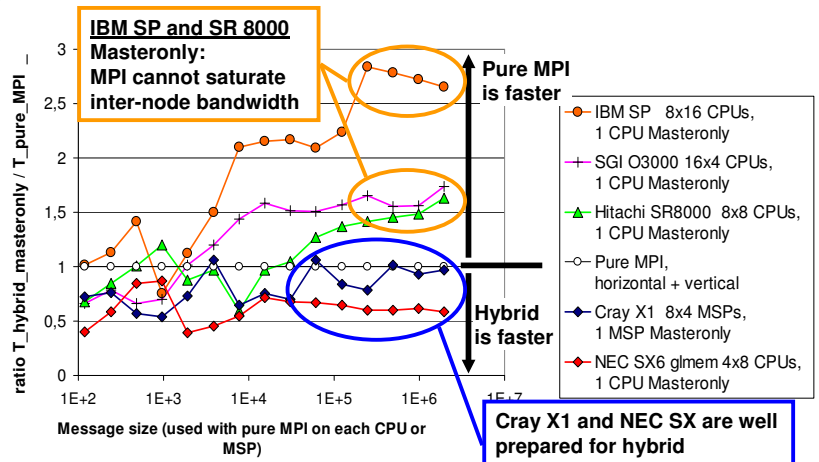
- pure MPI is better for message size > 32 kB
- long messages: $T_{hybrid} / T_{pureMPI} > 1.6$
- OpenMP master thread cannot saturate the inter-node network bandwidth





pure MPI
Masteronly

Ratio on several platforms



Cray X1 and SGI results are preliminary

pure MPI
Masteronly

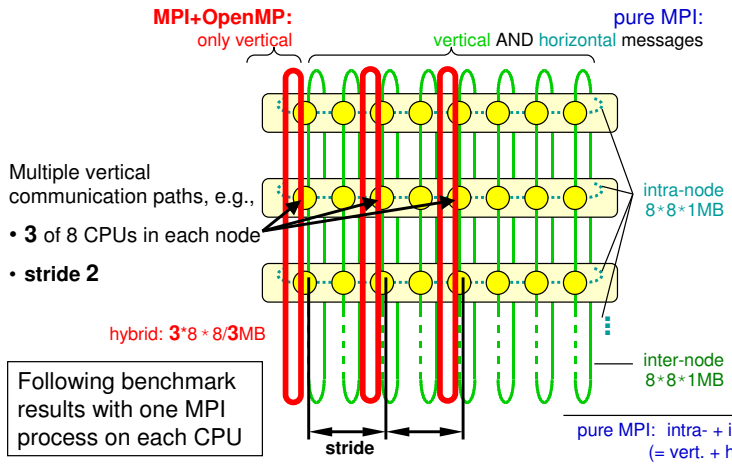
Possible Reasons

- Hardware:
  - is one CPU able to saturate the inter-node network?
- Software:
  - internal MPI buffering may cause additional memory traffic
    → memory bandwidth may be the real restricting factor?

➔ **Let's look at parallel bandwidth results**

pure MPI
Masteronly

Multiple inter-node communication paths

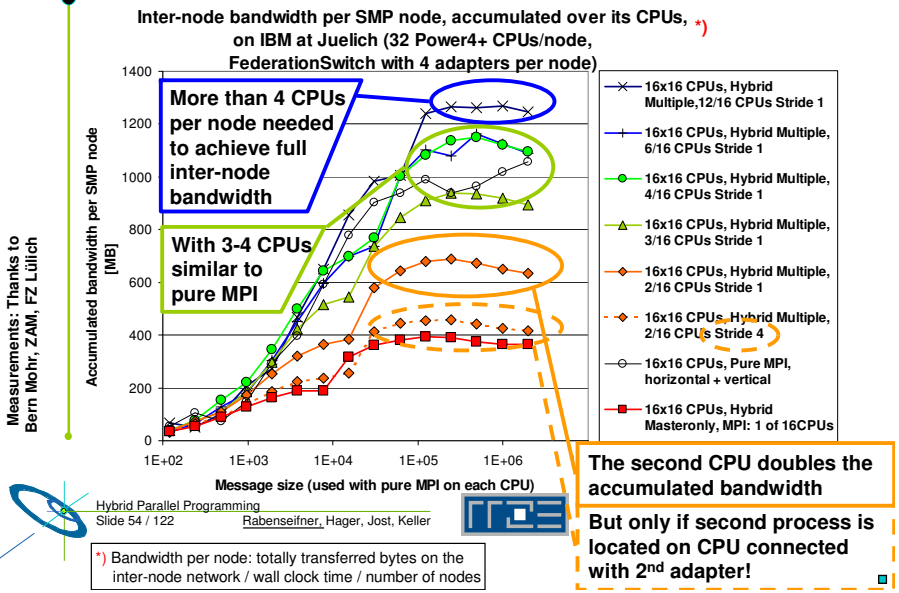


Following benchmark results with one MPI process on each CPU

pure MPI: intra- + inter-node (= vert. + horizontal)

pure MPI
Masteronly

Multiple inter-node communication paths: IBM SP



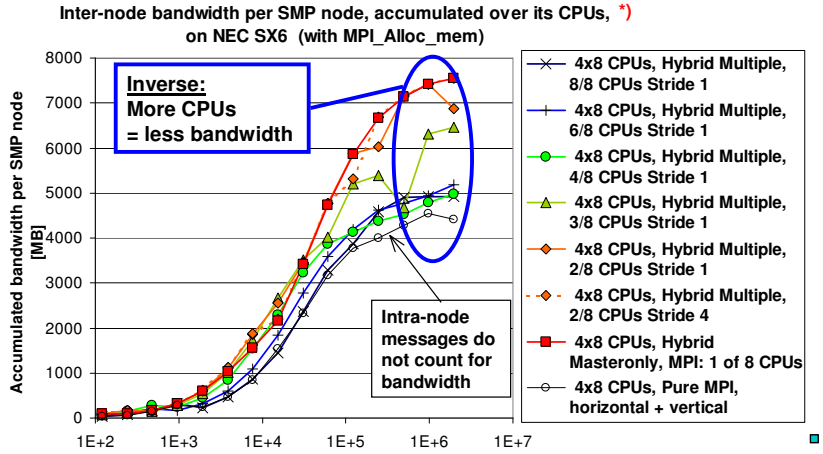
The second CPU doubles the accumulated bandwidth

But only if second process is located on CPU connected with 2nd adapter!

*) Bandwidth per node: totally transferred bytes on the inter-node network / wall clock time / number of nodes

pure MPI
Masteronly

Multiple inter-node communication paths: NEC SX-6 (using global memory)

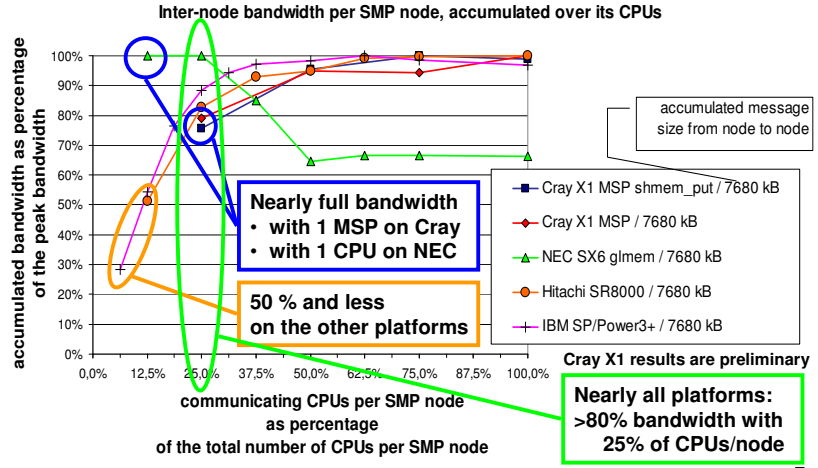


*) Bandwidth per node: totally transferred bytes on the inter-node network / wall clock time / number of nodes

Measurements: Thanks to Holger Berger, NEC.

pure MPI
Masteronly

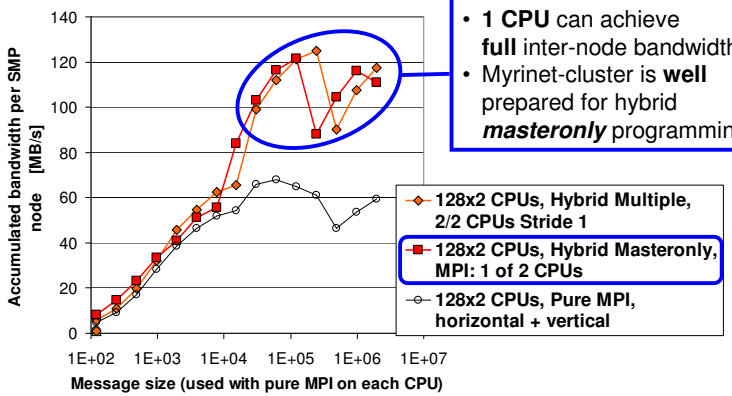
Comparison (as percentage of maximal bandwidth and #CPUs)



Myrinet Cluster

pure MPI / Masteronly

Inter-node bandwidth per SMP node, accumulated over its CPUs, on HELICS, 2 CPUs / node, Myrinet



- **1 CPU** can achieve **full** inter-node bandwidth
- Myrinet-cluster is **well** prepared for hybrid ***masteronly*** programming

Mismatch Problems
- Topology problem
- Unnecessary intra-node comm.
- ➢ **Inter-node bandwidth problem**
- Sleeping threads and saturation problem
- Additional OpenMP overhead
- Overlapping comm. and comp.
- Communication overhead w. DSM

Inter-node bandwidth problem – Summary and Work-around

With (typically) more than 4 threads / MPI process inter-node communication network cannot be saturated

→ On constellation type systems (more than 4 CPUs per SMP node)
- With (typically) **more** than 4 threads / MPI process inter-node communication network cannot be saturated
- Work-around: Several multi-threaded MPI process on each SMP node
- Finished problems come back:
  - **Topology problem:**
    - those processes should work on neighboring domains
    - to minimize inter-node traffic
  - **Unnecessary intra-node communication between these processes**
    - instead of operating on common shared memory
    - but **less** intra-node communication than with pure MPI

Mismatch Problems
- Topology problem
- Unnecessary intra-node comm.
- Inter-node bandwidth problem
- ➢ **Sleeping threads and saturation problem**
- Additional OpenMP overhead
- Overlapping comm. and comp.
- Communication overhead w. DSM

The sleeping-threads and the saturation problem

- Masteronly:
  - all other threads are sleeping while master thread calls MPI
    → wasting CPU time
    →→→ wasting plenty of CPU time if master thread cannot saturate the inter-node network
- Pure MPI:
  - all threads communicate, but already 1-3 threads could saturate the network
    → wasting CPU time

→ **Overlapping communication and computation**

Mismatch Problems
- Topology problem
- Unnecessary intra-node comm.
- Inter-node bandwidth problem
- Sleeping threads and saturation problem
- ➢ **Additional OpenMP overhead**
- Overlapping comm. and comp.
- Communication overhead w. DSM

Additional OpenMP Overhead

- Thread fork / join
- Cache flush
  - synchronization between *data source thread* and *communicating thread* implies → a cache flush
- Amdahl's law for each level of parallelism

Mismatch Problems

- Topology problem [with pure MPI]
- Unnecessary intra-node communication [with pure MPI]
- Inter-node bandwidth problem [with hybrid MPI+OpenMP]
- Sleeping threads and saturation problem [with masteronly] [with pure MPI]
- Additional OpenMP overhead [with hybrid MPI+OpenMP]
  - Thread fork / join
  - Cache flush (data source thread – communicating thread – sync. → flush)
- **Overlapping communication and computation** [with hybrid MPI+OpenMP]
  - an application problem → separation of local or halo-based code
  - a programming problem → thread-ranks-based vs. OpenMP work-sharing
  - a load balancing problem, if only some threads communicate / compute
- Communication overhead with DSM [with pure (Cluster) OpenMP]

→ no silver bullet, **i.e., each parallelization scheme has its problems**

Overlapping Communication and Computation

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

- 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
- 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

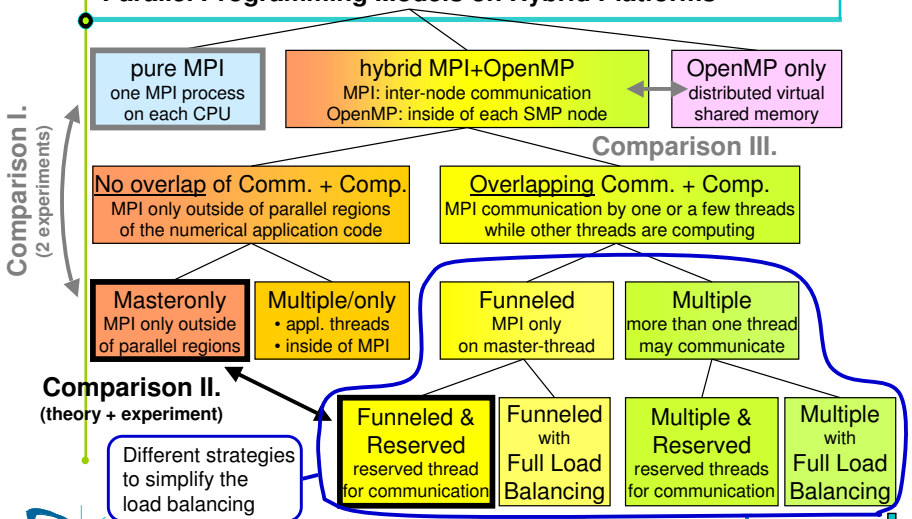
Subteams

- 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



Overlapping communication and computation (cont'd)

- the load balancing problem:
  - some threads communicate, others not
  - balance work on both types of threads
  - strategies:

Funneled & Reserved	Multiple & Reserved
reserved thread for communi.	reserved threads for communic.

  - reservation of one a fixed amount of threads (or portion of a thread) for communication
  - see example last slide: **1** thread was reserved for communication

➔ **a good chance !!! … see next slide**

Funneled with Full Load Balancing	Multiple with Full Load Balancing

➔ **very hard to do !!!**

Overlapping computation & communication (cont'd)

funneled & reserved

Funneled & reserved or Multiple & reserved:

- reserved tasks on threads:
  - master thread or some threads: communication
  - all other threads …….. : computation
- cons:
  - bad load balance, if

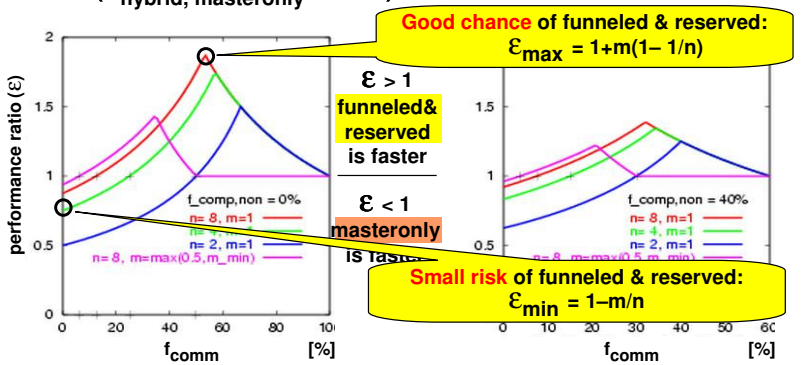
$$\frac{T_{communication}}{T_{computation}} \neq \frac{n_{communication\_threads}}{n_{computation\_threads}}$$

- pros:
  - more easy programming scheme than with full load balancing
  - chance for good performance!

Performance ratio (theory)

Masteronly / funneled & reserved

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

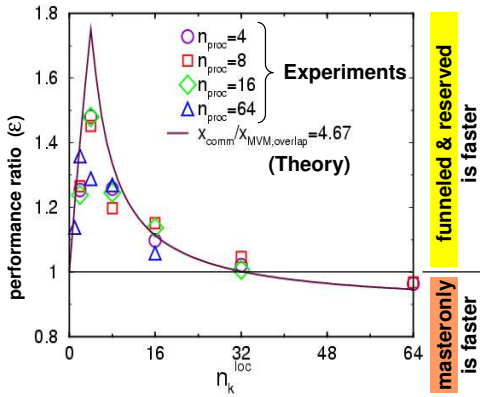


$T_{hybrid,\ masteronly} = (f_{comm} + f_{comp,\ non\text{-}overlap} + f_{comp,\ overlap})\ T_{hybrid,\ masteronly}$

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

Experiment: Matrix-vector-multiply (MVM)

Masteronly / funneled & reserved

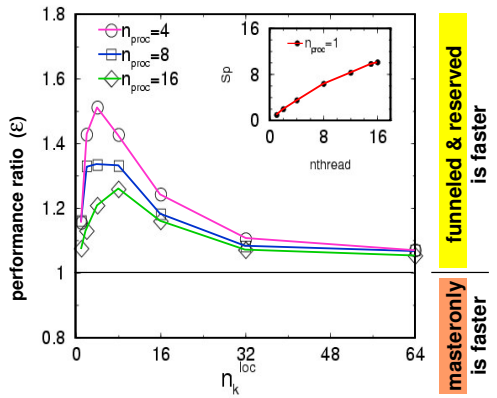


- Jacobi-Davidson-Solver
- Hitachi SR8000
- 8 CPUs / SMP node
- JDS (Jagged Diagonal Storage)
- vectorizing
- n_{proc} = # SMP nodes
- $D_{Mat} = 512 * 512 * (n_k^{loc} * n_{proc})$
- Varying n_k^{loc} ⇒ Varying $1/f_{comm}$
- $\frac{f_{comp,non\text{-}overlap}}{f_{comp,overlap}} = \frac{1}{6}$

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

Masteronly
funneled & reserved

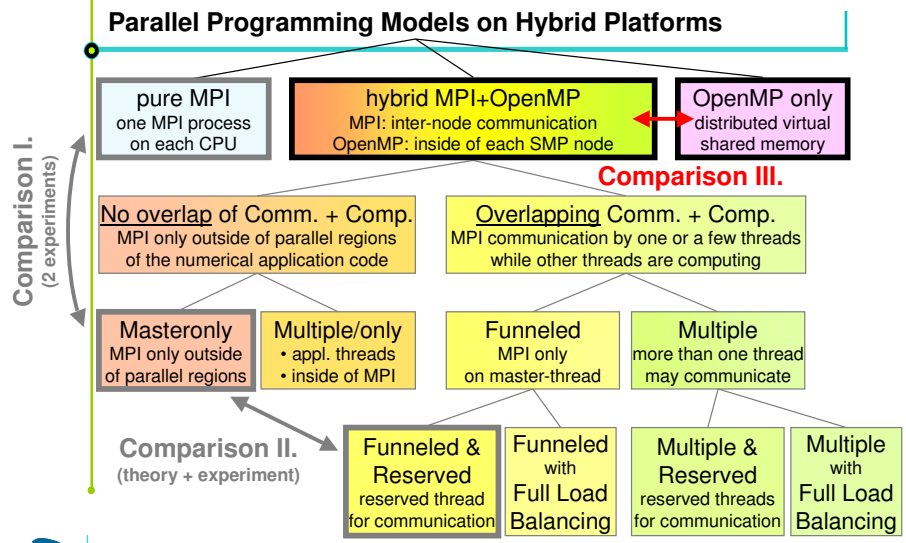
Experiment: Matrix-vector-multiply (MVM)



- 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 .

Parallel Programming Models on Hybrid Platforms



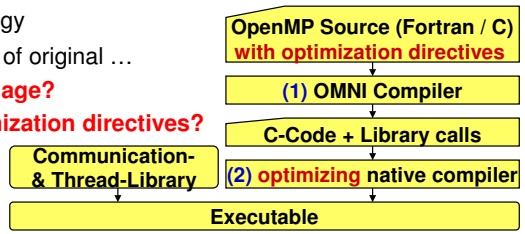
hybrid MPI+OpenMP ↔ OpenMP only

Compilation and Optimization

- Library based communication (e.g., MPI)
  - clearly separated optimization of
    - **(1)** communication → MPI library
    - **(2)** computation → Compiler

  } **essential for success of MPI**

- Compiler based parallelization (including the communication):
  - similar strategy
  - preservation of original …
    - **… language?**
    - **… optimization directives?**



- **Optimization of the computation more important than optimization of the communication**

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

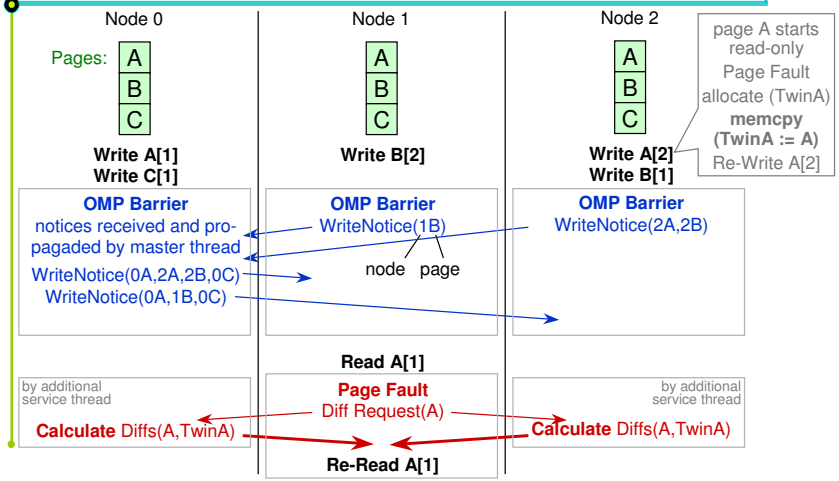
Intel® Compilers with Cluster OpenMP – Consistency Protocol

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

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
- Examples in the Appendix

Courtesy of J. Cownie, Intel

hybrid MPI+OpenMP ↔ OpenMP only

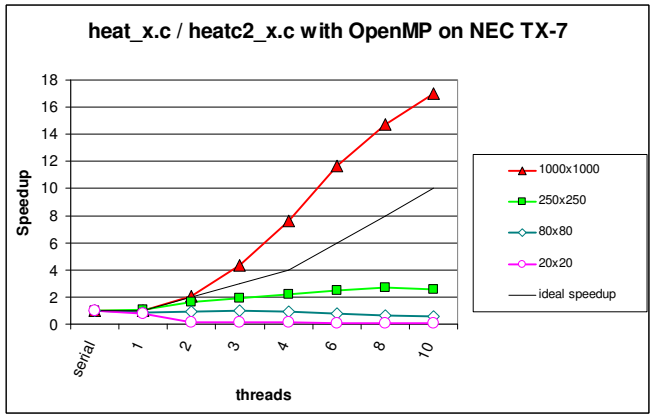
Comparison: MPI based parallelization ↔ DSM

- 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

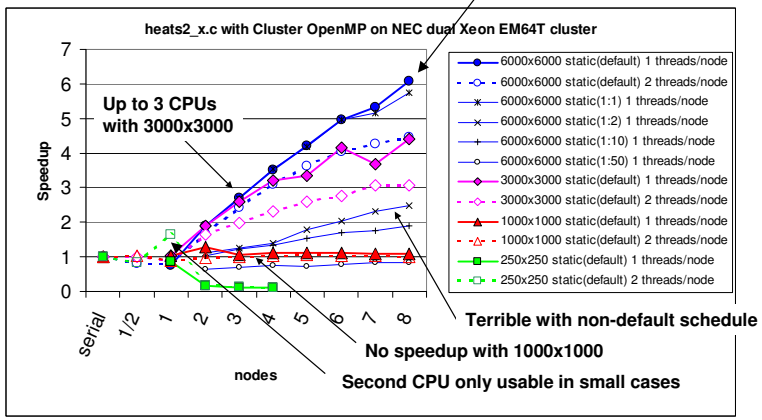
Comparing results with heat example

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



Heat example: Cluster OpenMP Efficiency

- Cluster OpenMP on a Dual-Xeon cluster



Mismatch Problems

- **Topology problem** [with pure MPI]
- **Unnecessary intra-node communication** [with pure MPI]
- **Inter-node bandwidth problem** [with hybrid MPI+OpenMP]
- **Sleeping threads and saturation problem** [with masteronly] [with pure MPI]
- **Additional OpenMP overhead** [with hybrid MPI+OpenMP]
  - Thread startup / join
  - Cache flush (data source thread – communicating thread – sync. → flush)
- **Overlapping communication and computation** [with hybrid MPI+OpenMP]
  - an application problem → separation of local or halo-based code
  - a programming problem → thread-ranks-based vs. OpenMP work-sharing
  - a load balancing problem, if only some threads communicate / compute
- **Communication overhead with DSM** [with pure (Cluster) OpenMP]

➔ **no silver bullet, i.e., each parallelization scheme has its problems**

No silver bullet

- The analyzed programming models do **not** fit on hybrid architectures
  - whether drawbacks are minor or major
    - **depends on applications' needs**
  - problems …
    - **to utilize the CPUs the whole time**
    - **to achieve the full inter-node network bandwidth**
    - **to minimize inter-node messages**
    - **to prohibit intra-node**
      - **message transfer,**
      - **synchronization and**
      - **balancing (idle-time) overhead**
    - **with the programming effort**

Chances for optimization

- with hybrid masteronly (**MPI only outside of parallel OpenMP regions**), e.g.,
  - **Minimize work of MPI routines, e.g.,**
    - application can copy non-contiguous data into contiguous scratch arrays (instead of using derived datatypes)
  - **MPI communication parallelized with multiple threads to saturate the inter-node network**
    - by internal parallel regions inside of the MPI library
    - by the user application
  - **Use only hardware that can saturate inter-node network with 1 thread**
  - **Optimal throughput:**
    - reuse of idling CPUs by other applications
- On constellations:
  - **Hybrid Masteronly with several MPI multi-threaded processes on each SMP node**

Summary of mismatch problems

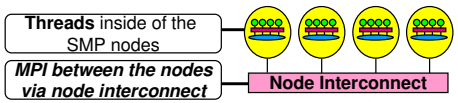
Performance and Programming Problems with ...	Pure MPI	Master-only 1 process per node	Master-only several processes per node	Over-lapping 1 process per node	Over-lapping several processes per node	Pure OpenMP: e.g., Intel Cluster OpenMP
Application topology problem (neighbor domains inside of SMP node)	↯		↯		↯	↯
Additional MPI communication inside of SMP nodes	↯		↯		↯	
Do we achieve full inter-node bandwidth on constellations?		↯↯↯		↯		↯↯↯
Sleeping CPUs while MPI communication	(↯)	↯↯	↯			↯
Additional OpenMP overhead		↯	↯	↯	↯	
Separation of (a) halo data and (b) inner data based calculations				↯↯	↯↯	
OpenMP work sharing only partially usable				↯↯	↯↯	
Load balancing problem due to hybrid programming model				↯	↯	

Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / pure MPI vs. hybrid MPI+OpenMP
- Mismatch Problems
- **Thread-safety quality of MPI libraries**
  Rainer Keller, High Performance Computing Center Stuttgart (HLRS)
- Case Studies / pure 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!

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

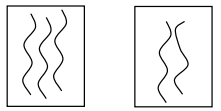
- But this implies that all other threads are sleeping!
- The additional barrier implies also the necessary cache flush!

… the barrier is necessary – example with MPI_Recv

!$OMP PARALLEL	#pragma omp parallel
!$OMP DO	{
do i=1,1000	#pragma omp for nowait
a(i) = buf(i)	for (i=0; i<1000; i++)
end do	a[i] = buf[i];
!$OMP END DO NOWAIT	
!$OMP BARRIER	#pragma omp barrier
!$OMP MASTER	#pragma omp master
call MPI_RECV(buf,...)	MPI_Recv(buf,...);
!$OMP END MASTER	#pragma omp barrier
!$OMP BARRIER	
!$OMP DO	#pragma omp for nowait
do i=1,1000	for (i=0; i<1000; i++)
c(i) = buf(i)	c[i] = buf[i];
end do	
!$OMP END DO NOWAIT	}
!$OMP END PARALLEL	/* omp end parallel */

Threads – Overview 1/2

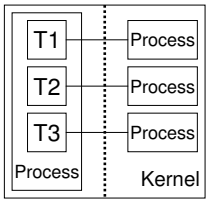
- Abstraction of the concept of a UNIX process.
- Change between processes is expensive (Context-Switch):
  - Switch into + out of privileged kernel mode.
  - Save the complete register set + status of processor.
  - Change the memory mapping of processes of MMU.



Data per process	Data per thread
Address space	Program counter
Open Files	Processor register
Child processes	Processor status
Signal handler	Signal masks
Timer	Stack
Accounting	

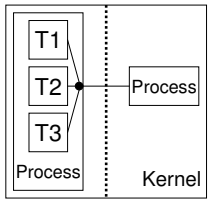
- POSIX:
  - Set of standards produced by IEEE Computer Society.
  - POSIX Threads published under POSIX 1003.1c.
  - Standardized by ISO as ISO/IEC 9945-1:1996.

Threads – Overview 2/2



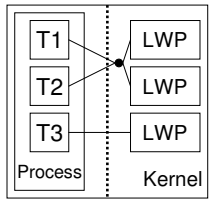
Kernel-Level Threads:
+ Simple model.
+ SMP systems used efficiently.

- Signals are not per process
- On plattforms with expensive switch to kernelmode inefficient.



User-Level Threads:
+ Faster thread-switch possible
+ Scheduler independent of system

- Blocking thread blocks all threads
- Programming bugs not always noticeable
- SMP systems not efficiently used.



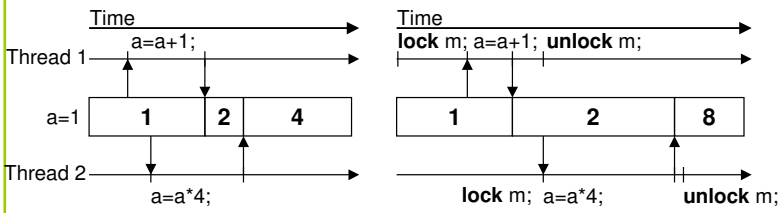
Hybrid Implement.:
+ Flexible
+ Application may interact (request more LWPs)

- Very complex
- Error-prone to implement (Signals).

Threads – Drawbacks 1/2

- Threads have a major drawback:

Thread-Safety



- Ensure, that **all** accesses to shared resources are protected
- Ensure, that **all** possible execution orders are sensible
- When using multiple locks, always lock/unlock in same order!

Threads – Drawbacks 2/2

- Many functions of the C-library are not thread-safe.
- These are:

`asctime`	`ctime`	`getgrgid`
`getgrnam`	`getpwnam`	`getpwuid`
`gmtime`	`localtime`	`rand`
`readdir`	`strtok`	

- For these functions, new thread-safe implementations are defined (suffix `_r`).
- To use these definitions, compile with `-D_REENTRANT`. Also will make the use of global error variable `errno` thread-safe.
  (With OpenMP compilation, this is on per default.)

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.
  - Easy integration of new tests
  - Tests must be runable with any number of processes.
  - Tests must run with as many:
    - **Communicators**
    - **Datatypes**
    - **Reduction-Operations**
    - **Lengths**

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
```

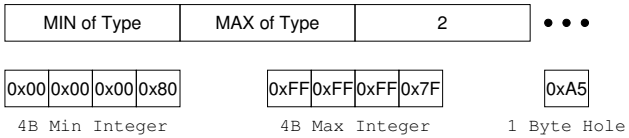
- Each test has to implement three functions:
  - `Init` One time test-initialization (buffer allocation
  - `Run` Main test-function, may be run multiple times.
  - `Cleanup` After the particular test was run.

- Make usage of convenience functions:
  - `tst_test_setstandardarray` Set buffer to known value.
  - `tst_test_checkstandardarray` Corresponding check

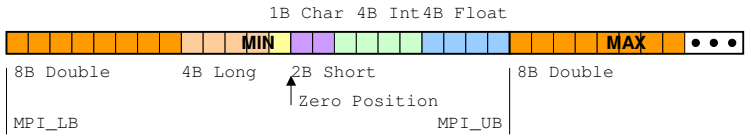
Testsuite – Derived Datatypes

- 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:



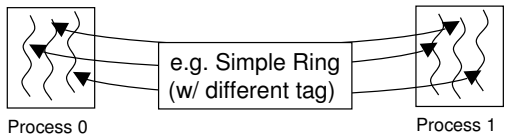
Testsuite – Implemented Communicators

- List of implemented communicators:

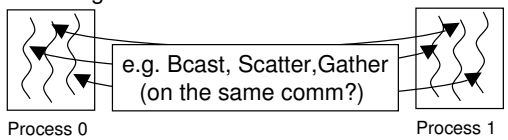
MPI_COMM_WORLD	MPI_COMM_NULL	MPI_COMM_SELF
Duplicated MPI_COMM_WORLD	Reversed MPI_COMM_WORLD	Halved MPI_COMM_WORLD
Odd-/Even Split MPI_COMM_WORLD		
Zero-and-Rest Intercommunicator	Intracomm merged of Halved Intercomms	Halved Intercommunicators
Two-dimensional Cartesian	Three-dimensional Cartesian	Fully-connected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

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:

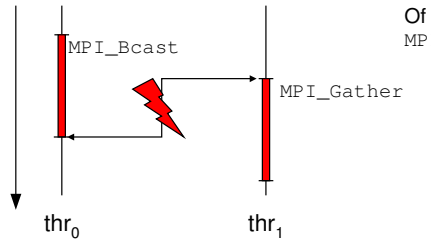


Testsuite – Implemented threaded tests

- 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:

Implemenation	Thread support level
MPIch-1.2.7p1	Always announces `MPI_THREAD_FUNNELED`.
MPIch2-1.0.4	ch:sock3 (default) supports `MPI_THREAD_MULTIPLE`
Intel MPI 2.0	`MPI_THREAD_FUNNELED`
Intel MPI 3.0	`MPI_THREAD_SERIALIZED`
SGI MPT-1.14	Not thread-safe?
IBM MPI	Full `MPI_THREAD_MULTIPLE`
Nec MPI/SX	Full `MPI_THREAD_MULTIPLE`

- 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 --enable-progress-threads
```

- This turns on:
  - Support for threaded initialization functions and internal checks to enable locking when run with threads
  - Progress threads to asynchronously transfer/receive data per network BTL.
  - However, some BTLs (mvapi, openib, mx) are still marked non-thread-safe.

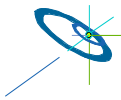
Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / pure MPI vs. hybrid MPI+OpenMP
- Mismatch Problems
- Thread-safety quality of MPI libraries

- **Case Studies / pure OpenMP**
  - First Experiences with Intel® Cluster OpenMP (**CLOMP**)

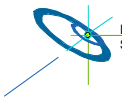
  Georg Hager, Regionales Rechenzentrum Erlangen (RRZE)

- Summary



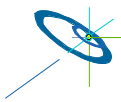
Overview

- Cluster OpenMP is part of every 9.1 Intel compiler
  - separate license must be purchased
- Systems used
  - EM64T (dual Nocona) with Gbit Ethernet and Infiniband, Debian 3.1 (Sarge)
  - Itanium2 (HP zx6000) with Gbit Ethernet, SLES9pl3
  - AMD Opteron is supported with latest CLOMP compiler versions
- Basic numbers: Triad tests on Nocona nodes
- Application: Lattice-Boltzmann code
  - influence of algorithmic details (locality of access, page sharing)
  - data layout considerations
- Odds and ends



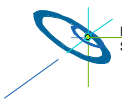
General Remarks on Intel® Cluster OpenMP (CLOMP)

- CLOMP == "extreme" ccNUMA
  - very long latencies, expensive non-local access
  - page replications can lead to memory problems
  - but: placement is handled “automatically”
- Consequence: A well-optimized, ccNUMA-aware OMP code that scales well on Altix does not necessarily scale well with CLOMP
  - example: boundary code must be optimized for local access
- Good stability on all systems with latest CLOMP release
- No problems and good performance with IP over IB
  - native IB not working yet (but check latest CLOMP versions!)



General Remarks

- Problems
  - memory footprint is about 2.5 times larger than expected from serial code (270MB instead of 61MB for vector triad)
    - **Partially resolved by Intel (Jim C.)**
    - **Problem is specific to RRZE kernel and system libs**
  - huge core dumps even with small sharable heap and resident memory (2.4GB core with 200MB code)
    - **Problem is specific to RRZE kernel and system libs**



Parallel Triad A(:)=B(:)+C(:)*D(:)

Three flavors

1. Standard triad, OMP parallel

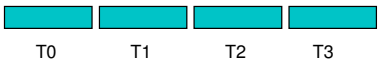

```
#pragma omp parallel for
   for(i=0; i<N; i++)
      a[i]=b[i]+c[i]*d[i];
```

2. Throughput triad (separate local arrays on each thread)

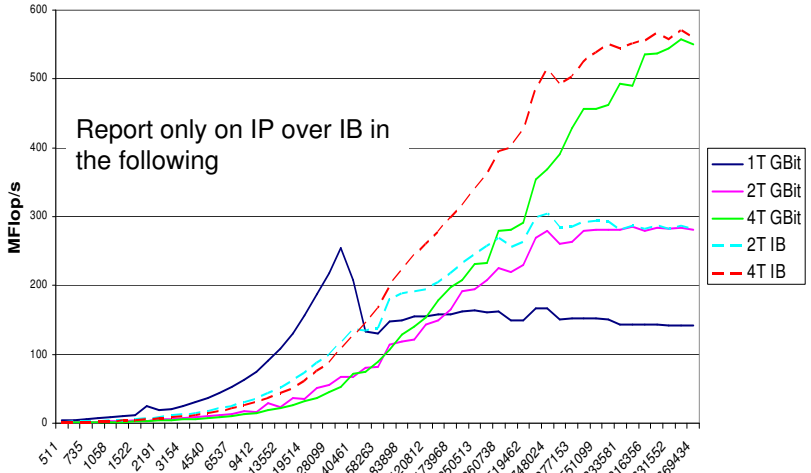
```
#pragma omp parallel
   sub_triad(N);
```

3. Padded triad

```
#pragma omp parallel
   do_triad(N[myID],
     start[myID],a,b,c,d)
```



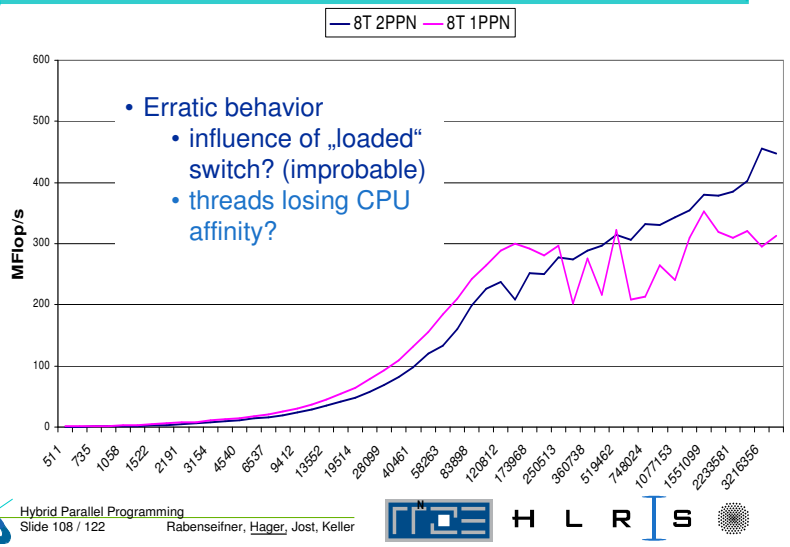
Standard Triad on GBit Ethernet vs. IP over IB (1T/node)



Filled vs. Half-filled nodes

- 2 ways to „fill the node“
  1. Keep unique names in hostfile and use 2 „real“ OpenMP threads per node with `--process_threads=2`
  2. Duplicate names in hostfile and use `--process_threads=1`
- Observations
  - breakdown of performance compared to the half-filled case for large N
  - Improvement with OpenMP for medium-sized arrays
  - `--process_threads=2`: quite erratic performance data
- Breakdown was actually expected (the same happens on single node with pure OpenMP)
- Erratic behaviour
  - influence of „loaded“ switch? (improbable)
  - Threads losing CPU affinity?

Threads vs. processes on node

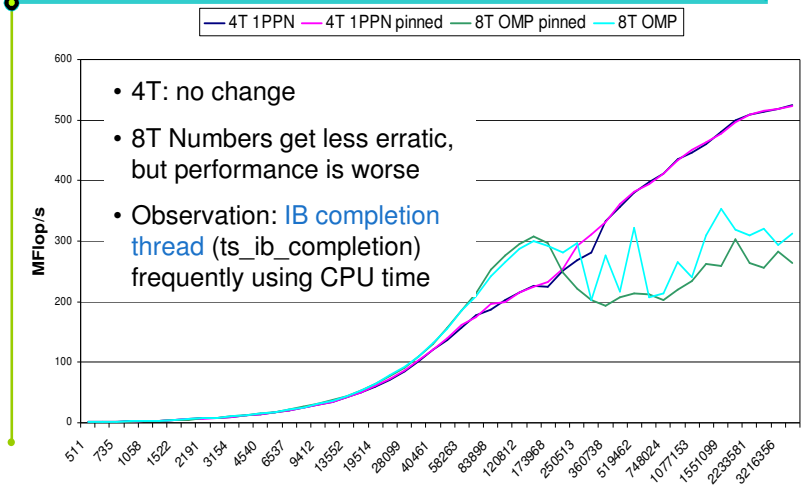


Pinning of threads

- Performance results seem quite erratic when using all available CPUs on a node
- Possible remedy? → pin threads to CPUs
  - using PLPA (http://www.open-mpi.org/software/plpa/) for portability

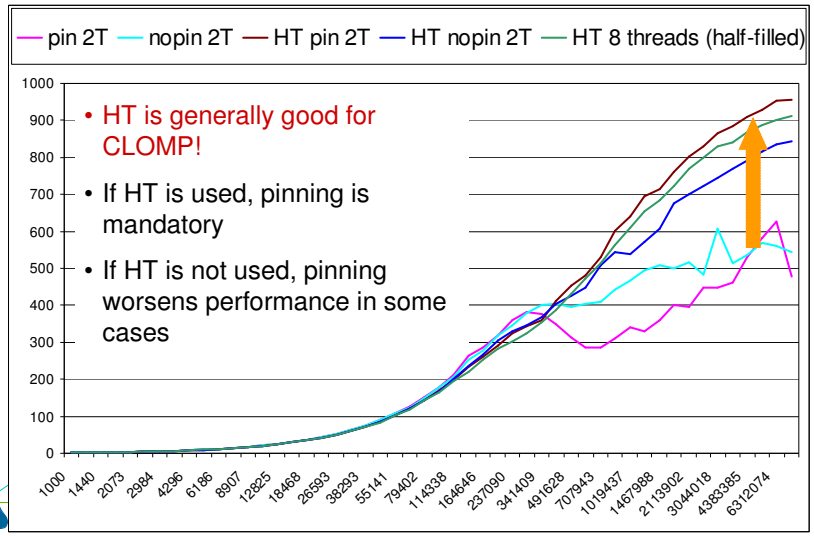
```
#pragma omp parallel
{
#pragma omp critical
{
    if(PLPA_NAME(api_probe)()!=PLPA_PROBE_OK) {
      cerr << "PLPA failed!" << endl;
  } else {
    plpa_cpu_set_t msk;
    PLPA_CPU_ZERO(&msk);
    PLPA_CPU_SET((omp_get_thread_num() & 1),&msk);
    PLPA_NAME(sched_setaffinity)((pid_t)0, (size_t)32, &msk);
  }
}
}
```


Results for pinned triad (4 and 8 threads)



- 4T: no change
- 8T Numbers get less erratic, but performance is worse
- Observation: IB completion thread (ts_ib_completion) frequently using CPU time

Remedy for the IB completion problem: Hyperthreading!



- HT is generally good for CLOMP!
- If HT is used, pinning is mandatory
- If HT is not used, pinning worsens performance in some cases

Application: Lattice Boltzmann Method

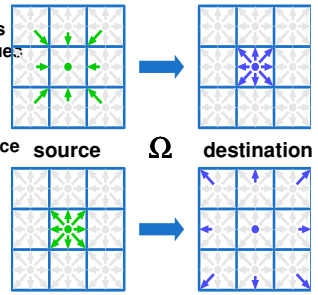
- Numerical Method for Simulation of Fluids

Stream-Collide (Pull-Method)

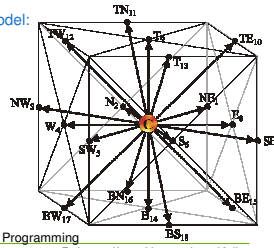
Get the distributions from the neighboring cells in the source array and store the relaxated values to one cell in the destination array

Collide-Stream (Push-Method)

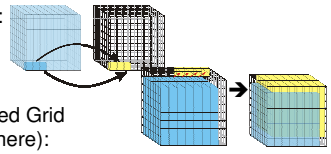
Take the distributions from one cell in the source array and store the relaxated values to the neighboring cells in the destination array



D3Q19 model:



Two Grids:



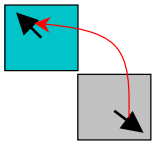
Compressed Grid (not used here):

LBMKernel – Code Structure for Collide-Stream Step

```
double precision f(0:xMax+1,0:yMax+1,0:zMax+1,0:18,0:1)
!$OMP PARALLEL DO
do z=1,zMax
   do y=1,yMax
      do x=1,xMax
           if( fluidcell(x,y,z) ) then
              LOAD f(x,y,z, 0:18,t)
              ...Relaxation (complex computations)...
              SAVE f(x  ,y  ,z  , 0,t+1)
              SAVE f(x+1,y+1,z  , 1,t+1)
              SAVE f(x  ,y+1,z  , 2,t+1)
              SAVE f(x-1,y+1,z  , 3,t+1)
              …
              SAVE f(x  ,y-1,z-1,18,t+1)
           endif
      enddo
   enddo
enddo
```

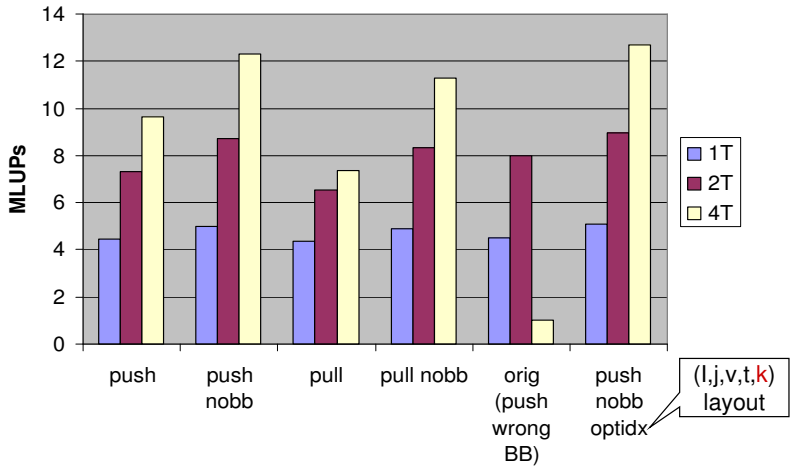

LBMKernel

- Scalability beyond 2 nodes was very bad with standard code
- proper choice of geometry (long thin channel) can restore scalability
  - not a general solution
- Solution: bounceback (boundary) routine was not properly optimized for local access
  - on ccNUMA, this is a negligible effect for small obstacle density (n^2)
  - on CLOMP, it is devastating



- Still: indexing has significant impact on performance
  - "push" vs. "pull" algorithm
  - parallelized dimension should be the outermost one to minimize false sharing: (i,j,v,t,k) better than (I,j,k,v,t)
- Might profit from ghost layers, but is this still OpenMP???

Influence of Bounceback and Push vs. Pull for 128x64x128 and (i,j,k,v,t) layout



DMRG (work in progress)

- Large C++ code, OpenMP parallelized
  - good scalability not really expected, but a good example for porting
  - cache-bound, so not optimized for ccNUMA
- Important issues:
  - use **`new (kmp_sharable)`** for dynamic objects used in parallel regions
  - derive classes from **`kmp_sharable_base`** if dynamic objects are used in parallel regions
- Possible problem with global objects (still under investigation)

Conclusions on CLOMP

- Cluster OpenMP is an interesting programming experience
- Imagine a ccNUMA machine with automatic page migration (wow!) and an awfully slow network
- If something strange happens (performancewise), use profiler by all means
  - Otherwise (with OMP) negligible boundary effects may become dominant with CLOMP
- With CLOMP, performance results tend to be more scattered than usual
- There is a lot more to say
  - role of pinning on ccNUMA nodes (Opteron)
  - automatic padding
  - C++ issues
  - Intel tools for profiling

Outline

Acknowledgements

- I want to thank
  - Gerhard Wellein, RRZE
  - Monika Wierse, Wilfried Oed, and Tom Goozen, CRAY
  - Holger Berger, NEC
  - Reiner Vogelsang, SGI
  - Gabriele Jost, NASA
  - Dieter an Mey, RZ Aachen
  - Horst Simon, NERSC
  - Matthias Müller, HLRS
  - my colleges at HLRS

On clusters with small nodes (≤ 4 CPUs)

Performance and Programming Problems with ...	Pure MPI	Master-only 1 process per node	Master-only several processes per node	Over-lapping 1 process per node	Over-lapping several processes per node	Pure OpenMP: e.g., Intel Cluster OpenMP
Application topology problem (neighbor domains inside of SMP node)	ϟ		ϟ		ϟ	ϟ
Additional MPI communication inside of SMP nodes	ϟ		ϟ		ϟ	
~~Do we achieve full inter-node bandwidth on constellations?~~		~~ϟϟϟ~~		~~ϟ~~		ϟϟϟ
Sleeping CPUs while MPI communication	(ϟ)	(~~ϟϟϟ~~)	ϟ			ϟ
Additional OpenMP overhead		ϟ	ϟ	ϟ	ϟ	
Separation of (a) halo data and (b) inner data based calculations				ϟϟ	ϟϟ	
OpenMP work sharing only partially usable				ϟϟ	ϟϟ	
Load balancing problem due to hybrid programming model				ϟ	ϟ	

Good candidates with limited programming expense

On constellations (> 4 CPUs per node)

Performance and Programming Problems with ...	Pure MPI	Master-only 1 process per node	Master-only several processes per node	Over-lapping 1 process per node	Over-lapping several processes per node	Pure OpenMP: e.g., Intel Cluster OpenMP
Application topology problem (neighbor domains inside of SMP node)	↯		↯		↯	↯
Additional MPI communication inside of SMP nodes	↯		↯		↯	
Do we achieve full inter-node bandwidth on constellations?		↯↯↯		↯		↯↯↯
Sleeping CPUs while MPI communication	(↯)	↯↯	↯			↯
Additional OpenMP overhead		↯	↯	↯	↯	
Separation of (a) halo data and (b) inner data based calculations				↯↯	↯↯	
OpenMP work sharing only partially usable				↯↯	↯↯	
Load balancing problem due to hybrid programming model				↯	↯	

Good candidates with limited programming expense

For extreme HPC, probably best chance

Non-MPI applications with extremely small communication foot-print

Performance and Programming Problems with ...	Pure MPI	Master-only 1 process per node	Master-only several processes per node	Over-lapping 1 process per node	Over-lapping several processes per node	Pure OpenMP: e.g., Intel Cluster OpenMP
Application topology problem (neighbor domains inside of SMP node)	↯		↯		↯	↯
Additional MPI communication inside of SMP nodes	↯		↯		↯	
Do we achieve full inter-node bandwidth on constellations?		↯↯↯		↯		↯↯↯
Sleeping CPUs while MPI communication	(↯)	↯↯	↯			↯
Additional OpenMP overhead		↯	↯	↯	↯	
Separation of (a) halo data and (b) inner data based calculations				↯↯	↯↯	
OpenMP work sharing only partially usable				↯↯	↯↯	
Load balancing problem due to hybrid programming model				↯	↯	

therefore irrelevant aspects

Maybe a candidate with limited programming expense

Conclusions

- **Constellations** (>4 CPUs per SMP node)**:**
  - **Only a few platforms**
    - e.g., Cray X1 in MSP mode, NEC SX-6
    - are well designed hybrid MPI+OpenMP masteronly scheme
  - **Other platforms**
    - masteronly style cannot saturate inter-node bandwidth
    - **Several multi-threaded MPI processes** per SMP node may help
- **Clusters with small SMP nodes:**
  - **Simple masteronly style** is a good candidate
  - although some CPU idle **(while one is communicating)**
- **DSM systems** (pure OpenMP, e.g Intel Cluster OpenMP)**:**
  - may help for **some** (**but only some**) applications
- **Optimal performance:**
  - overlapping of communication & computation → extreme programming effort
- **Pure MPI:**
  - often the cheapest and (nearly) best solution

See also www.hlrs.de/people/rabenseifner → list of publications

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)

Rolf Rabenseifner, HLRS, Germany
Georg Hager, University of Erlangen-Nuremberg, Germany
Gabriele Jost, Sun Microsystems, Germany
Rainer Keller, HLRS, Germany

Abstract. Most HPC systems are clusters of shared memory nodes. Such systems can be PC clusters with dual or quad boards, but also "constellation" type systems with large SMP nodes. Parallel programming must combine the distributed memory parallelization on the node inter-connect 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 nodes. Various hybrid MPI+OpenMP programming models are compared with pure MPI. Benchmark results of several platforms are presented. A hybrid-masteronly programming model can be used more efficiently on some vector-type systems, but also on clusters of dual-CPUs. On other systems, one CPU is not able to saturate the inter-node network and the commonly used masteronly programming model suffers from insufficient inter-node bandwidth. The thread-safety quality of several existing MPI libraries is also discussed. Case studies from the fields of CFD (NAS Parallel Benchmarks and Multi-zone NAS Parallel Benchmarks, in detail), Climate Modelling (POP2, maybe) and Particle Simulation (GTC, maybe) will be provided to demonstrate various aspect of hybrid MPI/OpenMP programming.

Another option is the use of distributed virtual shared-memory technologies which enable the utilization of "near-standard" OpenMP on distributed memory architectures. The performance issues of this approach and its impact on existing applications are discussed. This tutorial analyzes strategies to overcome typical drawbacks of easily usable programming schemes on clusters of SMP nodes.

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. 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. 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-Nürnberg. 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.

In his dissertation he developed a shared-memory parallel density-matrix renormalization group algorithm for ground-state calculations in strongly correlated electron systems. Recent work 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.

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.

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