

Hybrid Programming – Outline

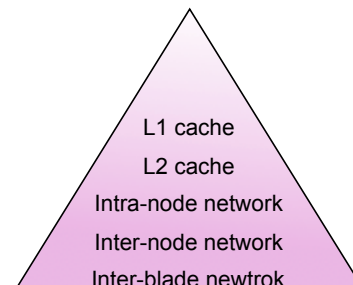
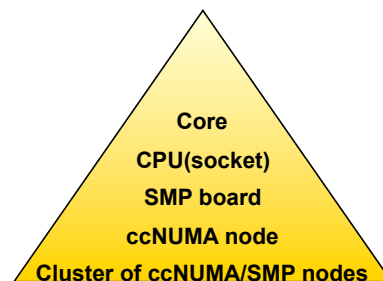
Author:
Hager

- Introduction / Motivation
- Programming Models on Clusters of SMP nodes
- Practical “How-To” on hybrid programming & Case Studies
- Mismatch Problems & Pitfalls
- Application Categories that Can Benefit from Hybrid Parallelization/Case Studies
- Summary on hybrid parallelization

Author:
seifner

Goals of this part of the tutorial

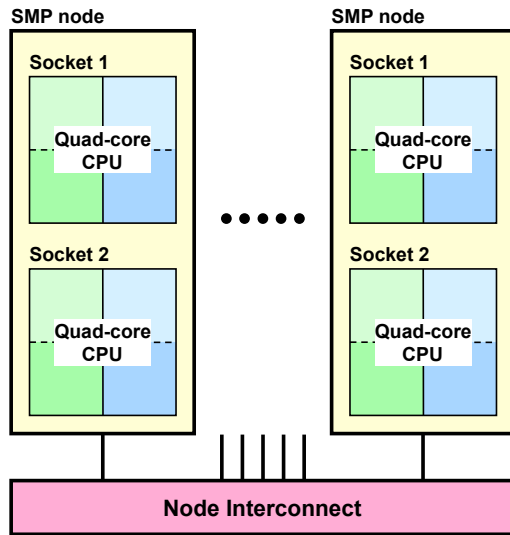
- **Effective methods for clusters of SMP node**
→ Mismatch problems & Pitfalls
- **Technical aspects of hybrid programming**
→ Programming models on clusters
→ “How-To”
- **Opportunities with hybrid programming** → Application categories that can benefit from hybrid parallelization
→ Case studies



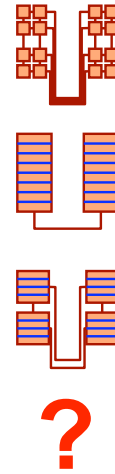
Author:
seifner

Motivation

Hybrid MPI/OpenMP programming seems natural

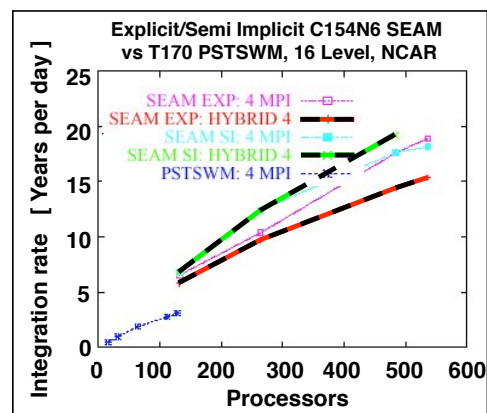
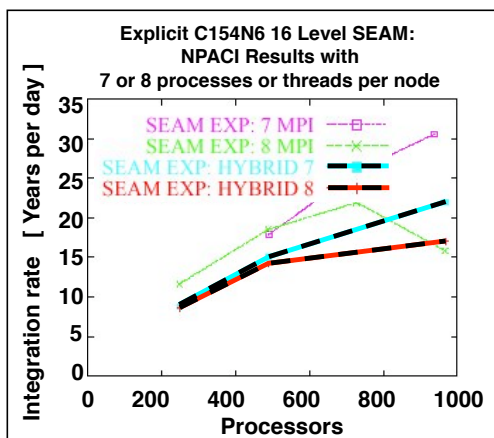


- Which programming model is fastest?
- MPI everywhere?
- Fully hybrid MPI & OpenMP?
- Something between? (Mixed model)
- Often hybrid programming **slower** than pure MPI
 - Examples, Reasons,
 - ...



Example from SC

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



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

— skipped —

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

Hybrid Programming – Outline

- Introduction / Motivation

- Programming Models on Clusters of SMP nodes

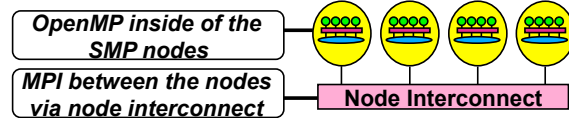
- Practical “How-To” on hybrid programming & Case Studies
- Mismatch Problems & Pitfalls
- Application Categories that Can Benefit from Hybrid Parallelization/ Case Studies
- Summary on hybrid parallelization

Programming Models for Hierarchical Systems

- **Pure MPI** (one MPI process on each CPU)

- **Hybrid MPI+OpenMP**

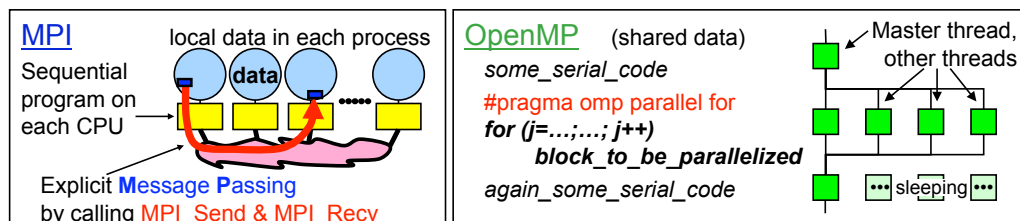
- shared memory OpenMP
- distributed memory MPI



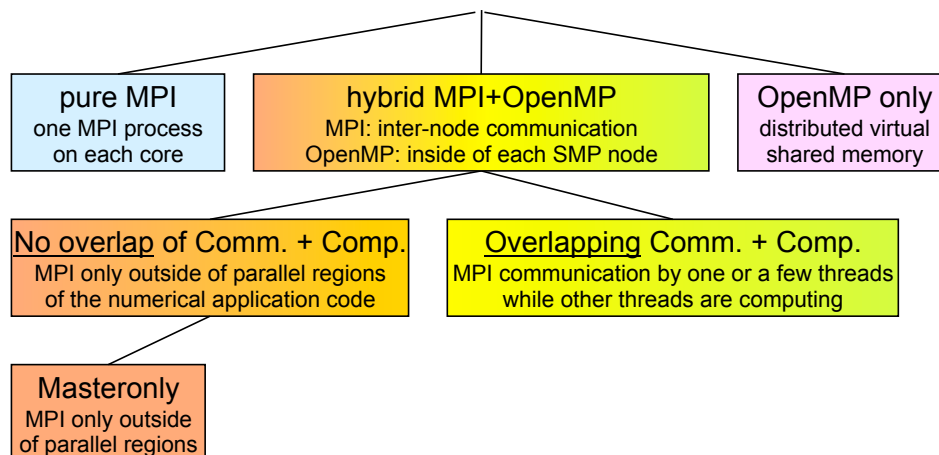
- **Other: Virtual shared memory systems, PGAS, HPF, ...**

- **Often hybrid programming (MPI+OpenMP) slower than pure MPI**

- why?



MPI and OpenMP Programming Models



Pure MPI

pure MPI
one MPI process
on each core

Advantages

- MPI library need not to support multiple threads

Major problems

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

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

■

Hybrid Masteronly

Masteronly
MPI only outside
of parallel regions

Advantages

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

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

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

Major Problems

- All other threads are sleeping while master thread communicates!
- Which inter-node bandwidth?
- MPI-lib must be thread-safe

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)

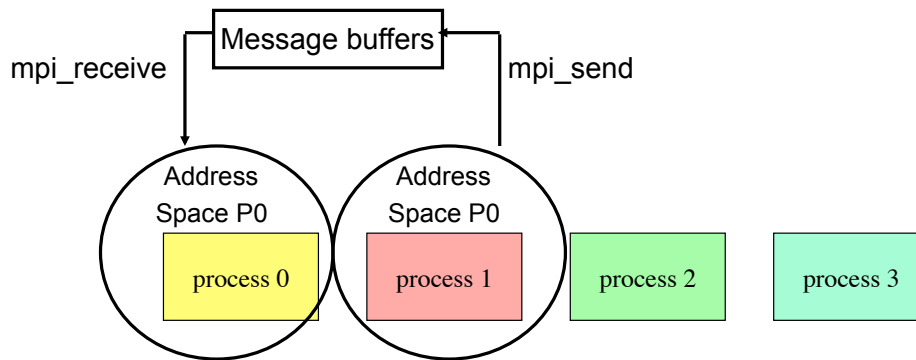
 by rule of thumb:
**Communication
may be
10 times slower
than with MPI**

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

■

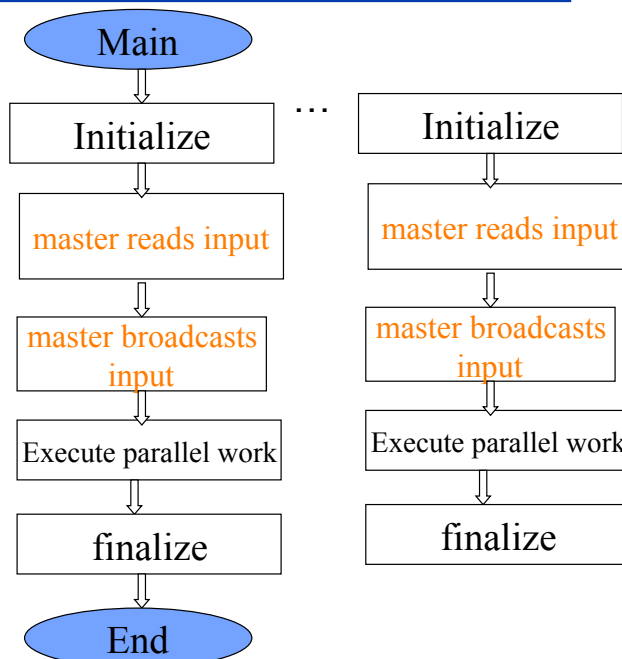
MPI Memory Model

- **Message Passing Interface**
- **Memory Model:**
 - MPI assumes a private address space
 - Private address space for each MPI Process
 - Data needs to be explicitly communicated
- **Applies to distributed and shared memory computer architectures**



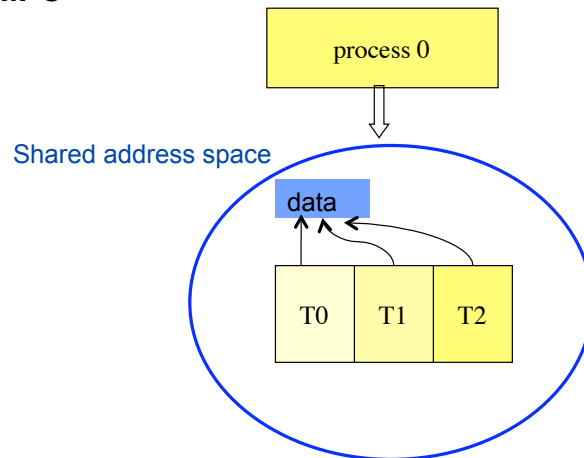
MPI Program General Structures

- In MPI/OpenMP all processes start up at the same time
- Two ways to handle input:
 - The master process reads the input data and broadcasts it to the other processes
 - Parallel I/O



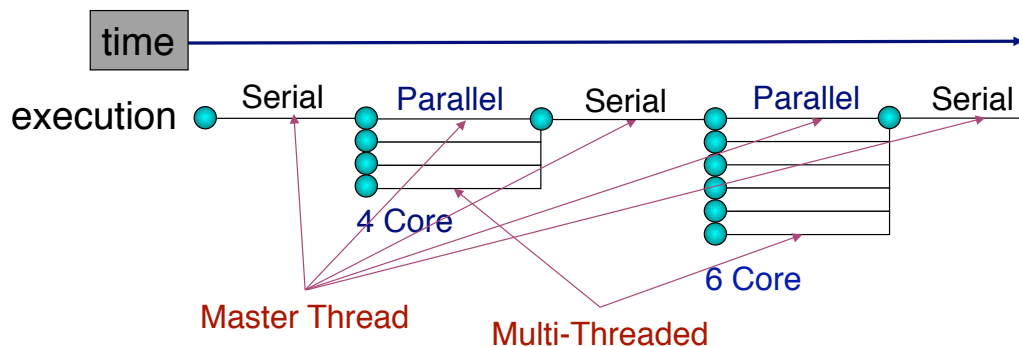
OpenMP Memory Model

- OpenMP assumes a shared address space
- No communication is required between threads
- Thread Synchronization is required when accessing shared data
- Applies to shared memory or distributed shared memory, e.g. Intel's Cluster OpenMP®™



OpenMP Code General Structure

- Fork-Join Model:
- Execution begins with a single “Master Thread”
- A team of threads is created at each parallel region
- Threads are joined at the end of parallel regions
- Execution is continued after parallel region by the Master Thread until the beginning of the next parallel region



Comparison of MPI and OpenMP

- | | |
|--|--|
| <ul style="list-style-type: none">• MPI• Memory Model<ul style="list-style-type: none">– Data private by default– Data accessed by multiple processes needs to be explicitly communicated• Program Execution<ul style="list-style-type: none">– One start and beginning• Parallelization<ul style="list-style-type: none">– Domain decomposition– Explicitly programmed by user | <ul style="list-style-type: none">• OpenMP• Memory Model<ul style="list-style-type: none">– Data shared by default– Access to shared data requires synchronization– Private data needs to be explicitly declared• Program Execution<ul style="list-style-type: none">– Fork-Join Model• Parallelization<ul style="list-style-type: none">– Typically on loop level– Based on compiler directives |
|--|--|

Support of Hybrid Programming

- | | |
|--|--|
| <ul style="list-style-type: none">• MPI<ul style="list-style-type: none">– MPI-1 no concept of threads– MPI-2:<ul style="list-style-type: none">– Thread support– MPI_Init_thread | <ul style="list-style-type: none">• OpenMP<ul style="list-style-type: none">– None– API only for one execution unit, which is one MPI process– For example: No means to specify the total number of threads across several MPI processes. |
|--|--|

MPI2 MPI_Init_thread

Syntax:

```
call MPI_Init_thread(                                irequired,    iprovided, ierr)
int MPI_Init_thread(int *argc, char ***argv, int required, int *provided)
int MPI::Init_thread(int& argc, char**& argv, int required)
```

Support Levels	Description
MPI_THREAD_SINGLE	Only one thread will execute.
MPI_THREAD_FUNNELED	Process may be multi-threaded, but only main thread will make MPI calls (calls are "funneled" to main thread). Default
MPI_THREAD_SERIALIZE	Process may be multi-threaded, any thread can make MPI calls, but threads cannot execute MPI calls concurrently (all MPI calls must be "serialized").
MPI_THREAD_MULTIPLE	Multiple threads may call MPI, no restrictions.

If supported, the call will return provided = required.
Otherwise, the highest level of support will be provided.

Funneling through Master

Fortran

```
include 'mpif.h'
program hybmas

call mpi_init_thread(...)

!$OMP parallel

!$OMP barrier
!$OMP master

call MPI_<whatever>(...,ierr)
!$OMP end master

!$OMP barrier

!$OMP end parallel
end
```

C

```
#include <mpi.h>
int main(int argc, char **argv){
    int rank, size, ierr, i;
    ierr = MPI_Init_thread(..)
    #pragma omp parallel
    {
        #pragma omp barrier
        #pragma omp master
        {
            ierr=MPI_<Whatever>(...)
        }

        #pragma omp barrier
    }
}
```

Serialize through Single

Fortran

```
include 'mpif.h'
program hybsing
  call mpi_init_thread(MPI_THREAD_SINGLE,
                      iprovided,ierr)

  !$OMP parallel

    !$OMP barrier
    !$OMP single

      call MPI_<whatever>(...,ierr)
      !$OMP end single

    !!OMP barrier

  !$OMP end parallel
end
```

C

```
#include <mpi.h>
int main(int argc, char **argv){
  int rank, size, ierr, i;
  mpi_init_thread(MPI_THREAD_SINGLE,
                  iprovided)
  #pragma omp parallel
  {
    #pragma omp barrier
    #pragma omp single
    {
      ierr=MPI_<Whatever>(...)
    }

    //pragma omp barrier
  }
}
```

Overlapping Communication and Work

- One core can saturate the PCI-e \leftrightarrow network bus. Why use all to communicate?
- **Communicate with one** or several cores.
- **Work with others** during communication.
- Need at least **MPI_THREAD_FUNNELED** support.
- Can be difficult to manage and load balance!

Overlapping Communication and Work

Fortran

```
include 'mpi.h'
program hybover

call mpi_init_thread(MPI_THREAD_FUNNELED,...)

!$OMP parallel

  if(ithread .eq. 0) then
    call MPI_<whatever>(...,ierr)
  else
    <work>
  endif

!$OMP end parallel
end
```

C

```
#include <mpi.h>
int main(int argc, char **argv){
  int rank, size, ierr, i;

  ierr= MPI_Init_thread(...)

  #pragma omp parallel
  {
    if (thread == 0){
      ierr=MPI_<Whatever>(...)
    }
    if(thread != 0){
      work
    }
  }
}
```

Thread-rank Communication

```
:
call mpi_init_thread( MPI_THREAD_MULTIPLE, iprovided,ierr)
call mpi_comm_rank(MPI_COMM_WORLD, irank, ierr)
call mpi_comm_size( MPI_COMM_WORLD,nranks, ierr)
:
!$OMP parallel private(i, ithread, nthreads)
:
  nthreads=OMP_GET_NUM_THREADS()
  ithread =OMP_GET_THREAD_NUM()
  call pwork(ithread, irank, nthreads, nranks...)
  if(irank == 0) then
    call mpi_send(ithread,1,MPI_INTEGER,1,ithread,MPI_COMM_WORLD, ierr)
  else
    call mpi_recv(      j,1,MPI_INTEGER,0,ithread,MPI_COMM_WORLD, istatus,ierr)
    print*, "Yep, this is ",irank," thread ", ithread," I received from ", j
  endif

!$OMP END PARALLEL
end
```

Communicate between ranks.

Threads use tags to differentiate.

Running Hybrid Codes

- **Running the code**
 - Highly non-portable! Consult system docs
 - Things to consider:
 - Is environment available for MPI Processes:
 - E.g.: `mpirun -np 4 OMP_NUM_THREADS=4 ...`
`a.out` instead of your binary alone may be necessary
 - How many MPI Processes per node?
 - How many threads per MPI Process?
 - Which cores are used for MPI?
 - Which cores are used for threads?
 - Where is the memory allocated?

Hybrid Programming – Outline

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

Author:
Hager

- Practical “How-To” on hybrid programming & Case Studies
- Mismatch Problems & Pitfalls
- Application categories that can benefit from hybrid parallelization
- Summary on hybrid parallelization

Hybrid Programming How-To: Overview

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

How to compile, link and run

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

Some examples for compilation and execution (1)

- **Standard Intel Xeon cluster:**

- Intel Compiler
- `mpif90 -openmp ...`
- Execution (handling of `OMP_NUM_THREADS`, see next slide):

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

Some examples for compilation and execution (2)

Handling of `OMP_NUM_THREADS`

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

Some examples for compilation and execution (3)

Handling of OMP_NUM_THREADS (continued)

Author:
Rabenseifner

- with support by OpenMPI `-x` option:

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

Author:
Gle Jost

- **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)

Some examples for compilation and execution (4)

- **Cray XT4:**
 - `ftn -fastsse -tp barcelona-64 -mp=nonuma ...`
 - `aprun -n nprocs -N nprocs_per_node a.out`
- **NEC SX8**
 - NEC SX8 compiler
 - `mpif90 -C hopt -P openmp ... # -ftrace for profiling info`
 - Execution:

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


Interlude: Advantages of mpiexec

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

Running the code

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

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

- **Same but 2 MPI processes per node:**

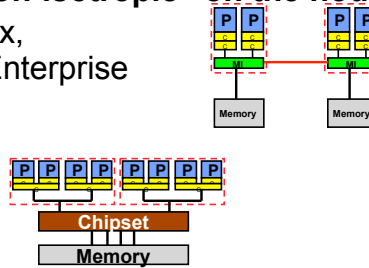
```
$ export OMP_NUM_THREADS=2
$ mpiexec -npernode 2 ./a.out
```

- **Pure MPI:**

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

Running the code *efficiently*?

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

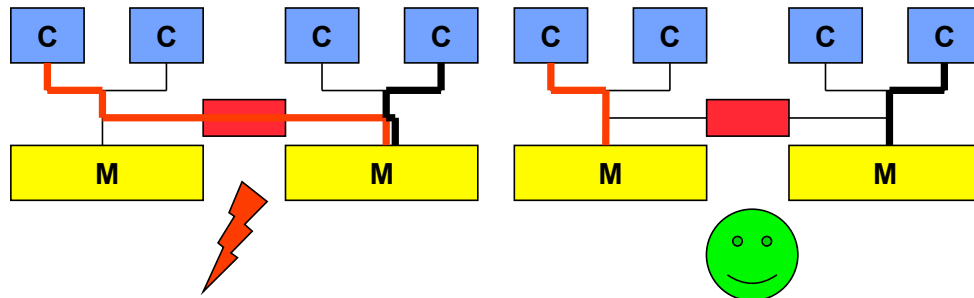


Issues for running code efficiently on “non-isotropic” nodes

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

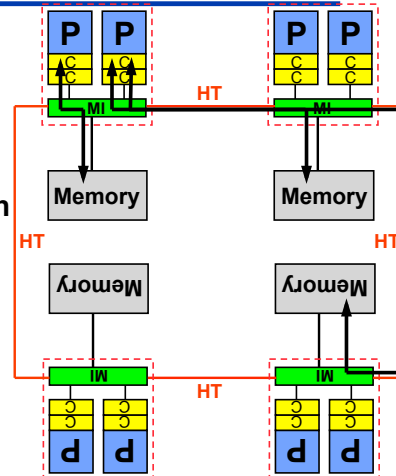
A short introduction to ccNUMA

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

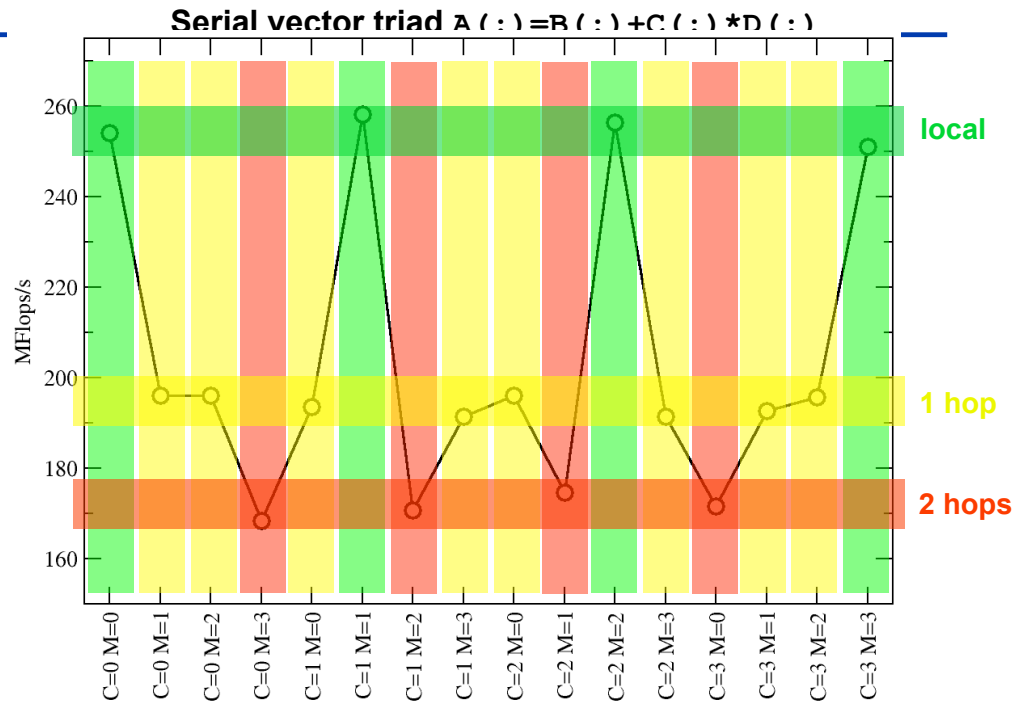


Example: HP DL585 G5 4-socket ccNUMA Opteron 8220 Server

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



Effect of non-local access on HP DL585 G5:

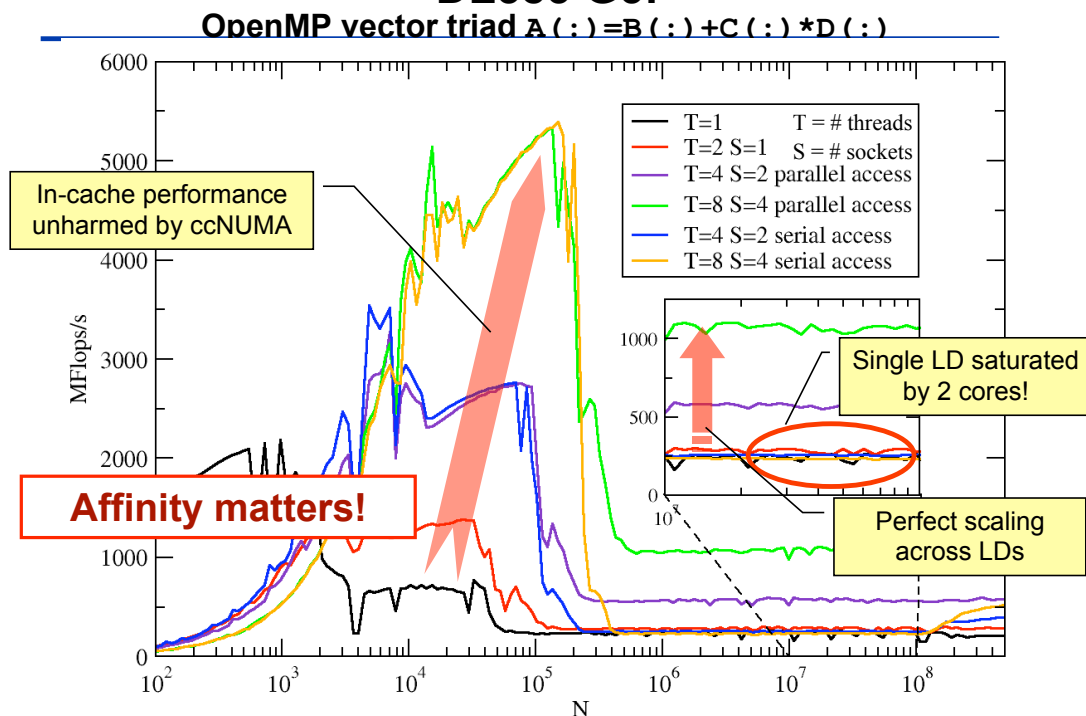


Author:
Hager

ParCFD09 Tutorial © Jost, Koniges, Wellein, Hager, Rabenseifner, Lusk & Others

39

Contention vs. parallel access on HP DL585 G5:



Author:
Hager

ParCFD09 Tutorial © Jost, Koniges, Wellein, Hager, Rabenseifner, Lusk & Others

40

ccNUMA Memory Locality Problems

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

Avoiding locality problems

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

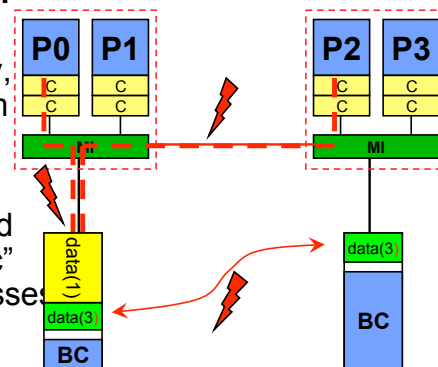
Solving Memory Locality Problems: First Touch

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

```
double *huge = (double*)malloc(N*sizeof(double));
// memory not mapped yet
for(i=0; i<N; i++) // or i+=PAGE_SIZE
    huge[i] = 0.0; // mapping takes place here!
```
- It is sufficient to touch a single item to map the entire page

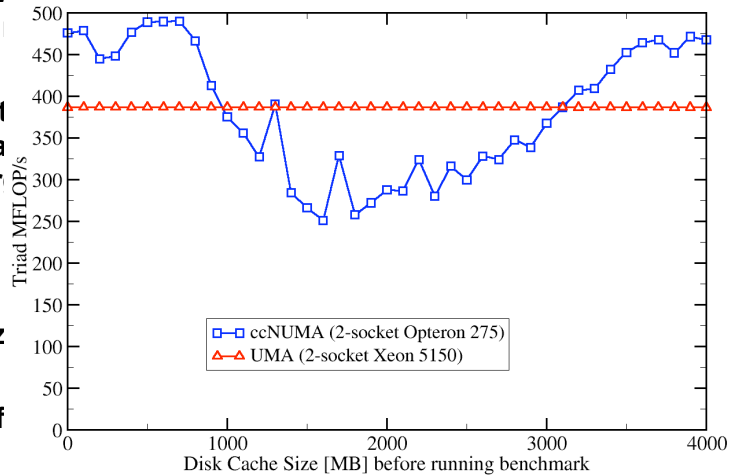
ccNUMA problems beyond first touch

- **OS uses part of main memory for disk buffer (FS) cache**
 - If FS cache fills part of memory, apps will probably allocate from foreign domains
 - → non-local access!
 - Locality problem even on hybrid and pure MPI with "asymmetric" file I/O, i.e. if not all MPI processes perform I/O
- **Remedies**
 - Drop FS cache pages after user job has run (admin's job)
 - Only prevents cross-job buffer cache "heritage"
 - "Sweeper" code (run by user)
 - Flush buffer cache after I/O if necessary ("sync" is not sufficient!)



ccNUMA problems beyond first touch

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



Author:
Hager

ParCFD09 Tutorial © Jost, Koniges, Wellein, Hager, Rabenseifner, Lusk & Others

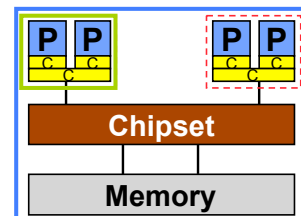
45

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`

Author:
Hager

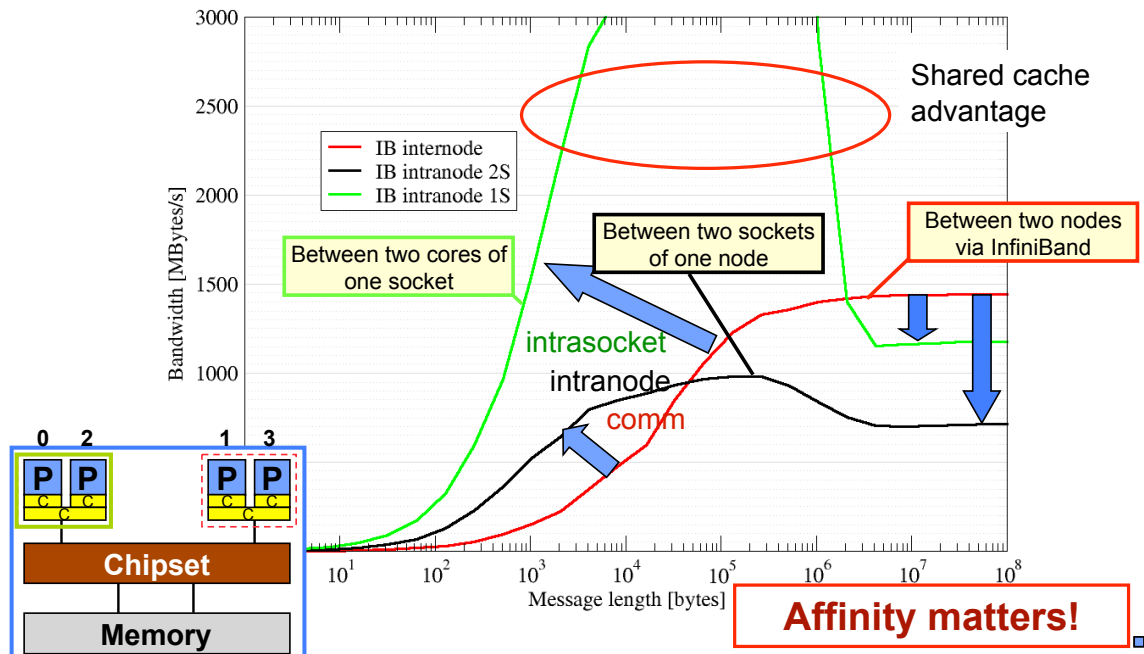
ParCFD09 Tutorial © Jost, Koniges, Wellein, Hager, Rabenseifner, Lusk & Others

Courtesy of Georg Hager (RRZE)

46

IMB Ping-Pong on DDR-IB Woodcrest cluster: Bandwidth Characteristics

Intra-Socket vs. Intra-node vs. Inter-node



Author:
Hager

ParCFD09 Tutorial © Jost, Koniges, Wellein, Hager, Rabenseifner, Lusk & Others

Courtesy of Georg Hager (RRZE)

47

Hybrid Programming – Outline

- Introduction / Motivation
- Programming models on clusters of SMP nodes
- Case Studies / Practical “How-To” on hybrid programming
- **Mismatch Problems & Pitfalls**
- Opportunities: Application categories that can benefit from hybrid parallelization
- Summary on hybrid parallelization

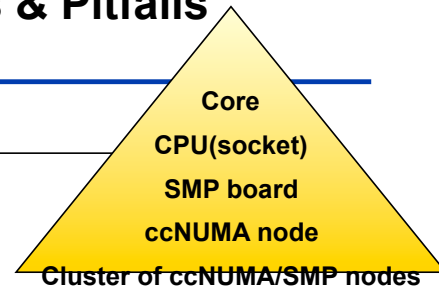
Author:
seifner

ParCFD09 Tutorial © Jost, Koniges, Wellein, Hager, Rabenseifner, Lusk & Others

48

Mismatch Problems & Pitfalls

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



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

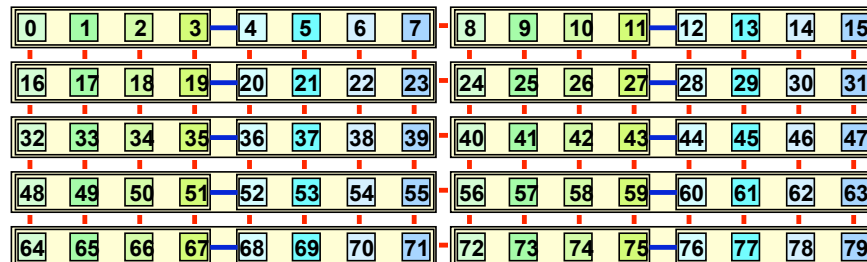
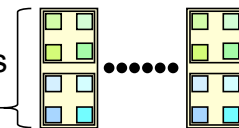
In most cases:
Both categories!

The Topology Problem with pure MPI

one MPI process
on each core

Application example on 80 cores:

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



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

Sequential ranking of
MPI_COMM_WORLD

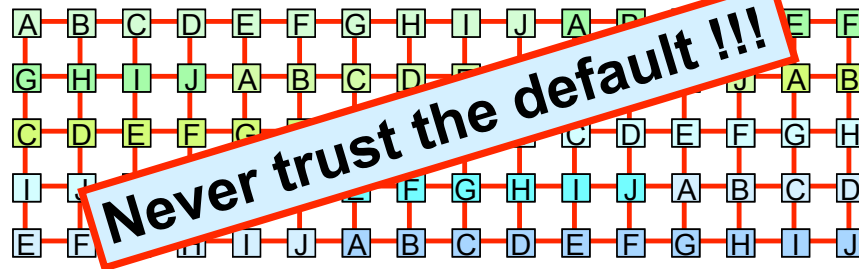
Does it matter?

The Topology Problem with pure MPI

one MPI process
on each core

Application example on 80 cores:

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



- + 32 x inter-node connections per node
- 0 x inter-socket connection per node

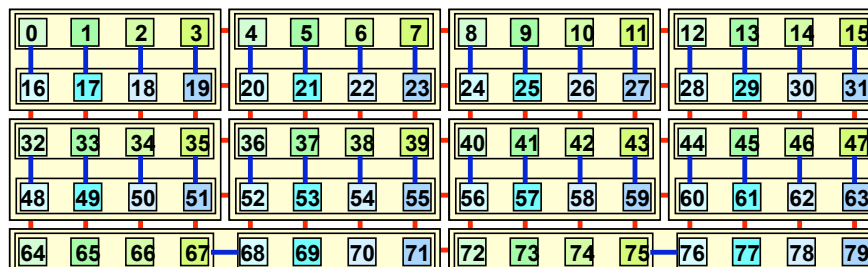
Round robin ranking of
MPI_COMM_WORLD

The Topology Problem with pure MPI

one MPI process
on each core

Application example on 80 cores:

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



- + 10 x inter-node connections per node
- + 4 x inter-socket connection per node

Two levels of
domain decomposition

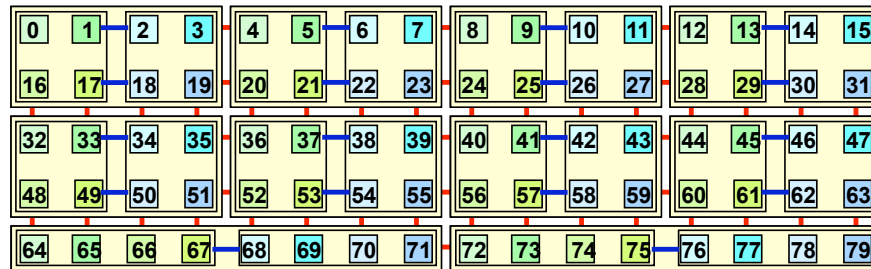
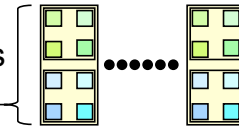
Bad affinity of cores to thread ranks

The Topology Problem with pure MPI

one MPI process
on each core

Application example on 80 cores:

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



+ 10 x inter-node connections per node

+ 2 x inter-socket connection per node

Two levels of
domain decomposition

Good affinity of cores to thread ranks

Author:
seifner

ParCFD09 Tutorial © Jost, Koniges, Wellein, Hager, Rabenseifner, Lusk & Others

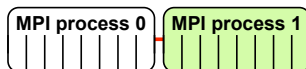
53

The Topology Problem with hybrid MPI+OpenMP

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

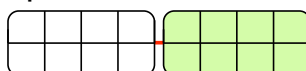
Exa.: 2 SMP nodes, 8 cores/node

Optimal ?



Loop-worksharing
on 8 threads

Optimal ?



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

Problem

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

Solutions:

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

Successful examples:

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

Author:
seifner

ParCFD09 Tutorial © Jost, Koniges, Wellein, Hager, Rabenseifner, Lusk & Others

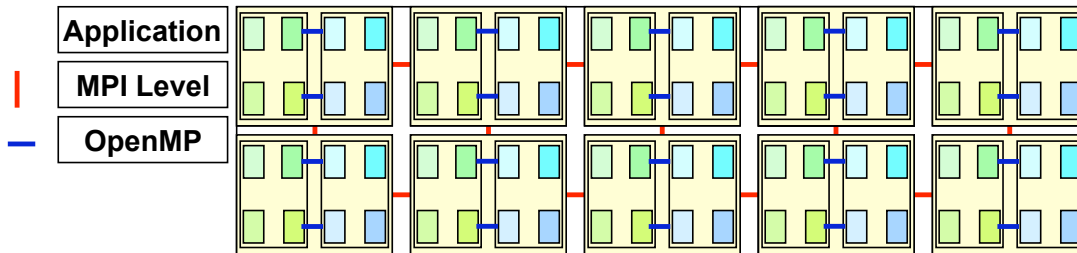
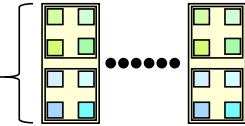
54

The Topology Problem with hybrid MPI+OpenMP

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

Application example:

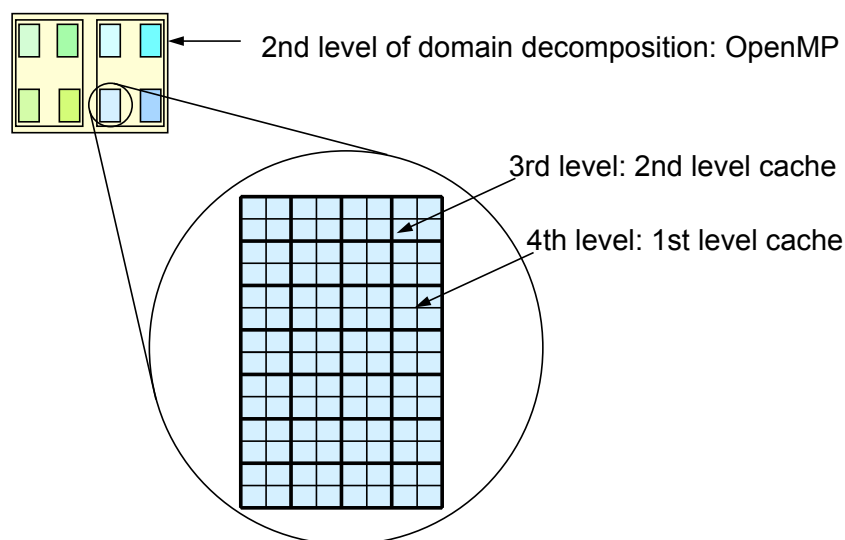
- Same Cartesian application aspect ratio: 5 x 16
- On system with 10 x dual socket x quad-core
- 2 x 5 domain decomposition



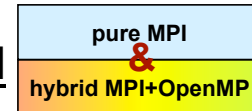
- + 3 x inter-node connections per node, but ~ 4 x more traffic
- + 2 x inter-socket connection per node

Affinity of cores to thread ranks !!!

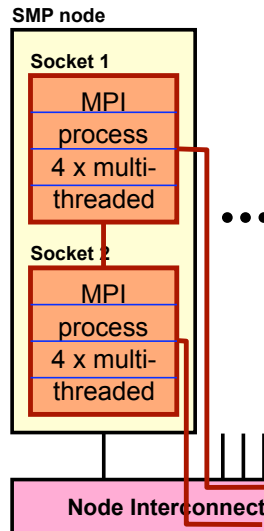
Inside of an SMP node



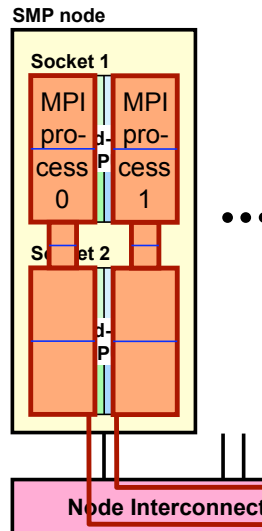
The Mapping Problem with mixed model



Do we have this?



... or that?



Several multi-threaded MPI process per SMP node:

Problem

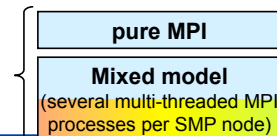
- Where are your processes and threads really located?

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

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

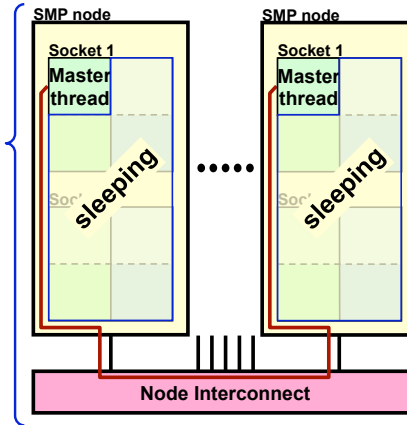
Quality aspects of the MPI library

Sleeping threads and network saturation with Masteronly

MPI only outside of
parallel regions

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

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



Problem 1:

- Can the master thread saturate the network?

Solution:

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

Problem 2:

- Sleeping threads are wasting CPU time

Solution:

- Overlapping of computation and communication

Problem 1&2 together:

- Producing more idle time through lousy bandwidth of master thread

OpenMP: Additional Overhead & Pitfalls

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

Overlapping Communication and Computation

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

Three problems:

- the application problem:

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

→ **very hard to do !!!**

- the thread-rank problem:

- comm. / comp. via thread-rank
- cannot use work-sharing directives

→ **loss of major OpenMP support**
(see next slide)

- the load balancing problem

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

Jacobi Solver

Basic implementation (2 arrays; no blocking etc...)

```

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

```

Performance Measure:
 Million Lattice Site Updates per second: MLUPs

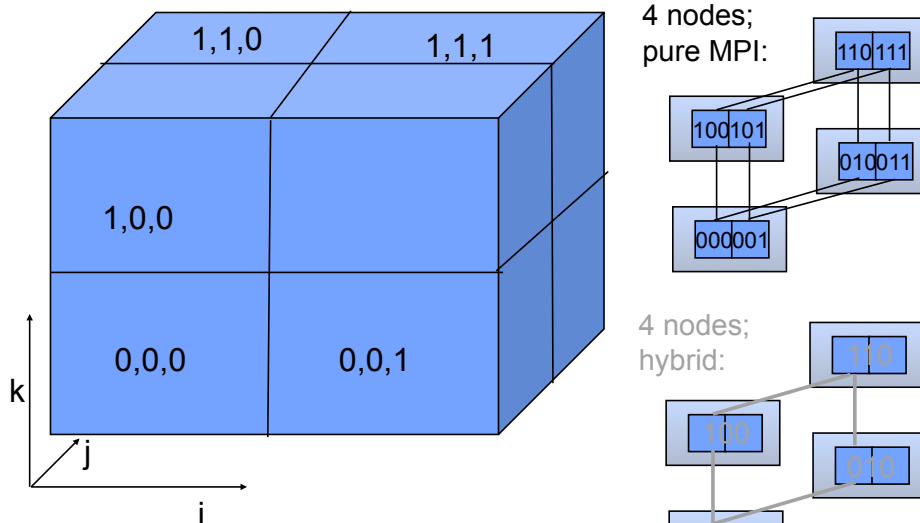
Equivalent MFLOPs:
 8 FLOP/LUP * MLUPs

Parallelization through

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

Parallelization – 3-D Jacobi

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



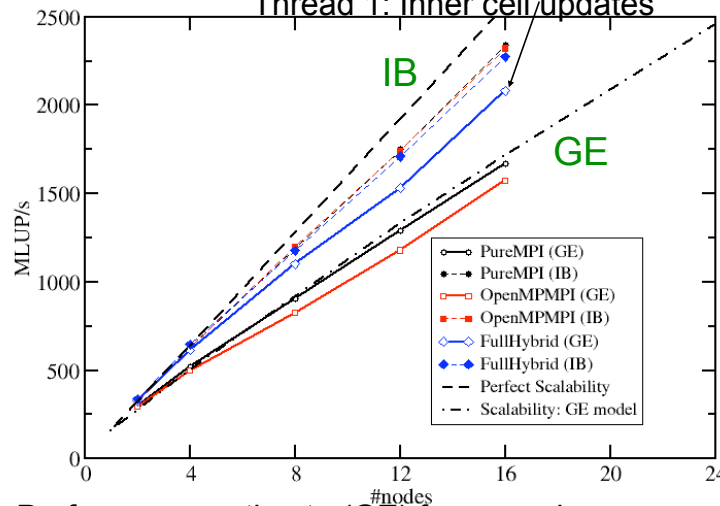
Strong scaling:

$$N^3 = 480^3$$

Hybrid on GE:

Thread 0: Communication + Boundary cell updates

Thread 1: Inner cell updates



Performance model

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

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

$$T_{\text{COMM}} = \text{DaVo} / \text{BW}$$

$$P_0 = 150 \text{ MLUP/s}$$

$$\text{BW(GE)} = 100 \text{ MBit/s}$$

Data volume of halo exchange

Performance estimate (GE) for no nodes:

$$P(\text{no}) = N^3 / ((T_{\text{COMP}}/\text{no}) + T_{\text{COMM}}(\text{no}))$$

Hybrid Programming – Outline

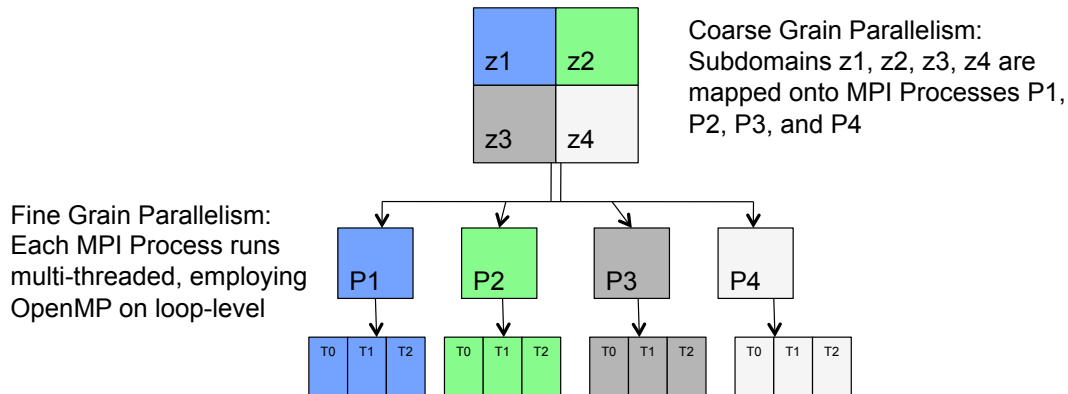
- Introduction / Motivation
- Programming Models on Clusters of SMP nodes
- Practical “How-To” on hybrid programming & Case Studies
- Mismatch Problems & Pitfalls
- Application Categories that Can Benefit from Hybrid Parallelization/Case Studies
- Summary on hybrid parallelization

Author:
Hager

Author:
seifner

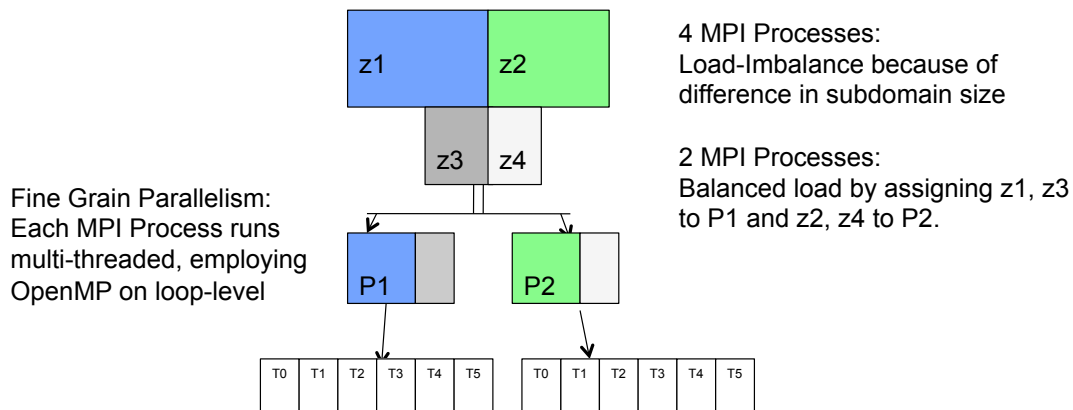
Multi-Level Parallelism in Applications

- **Extract additional Parallelism in case of Limited coarse grain Parallelism**



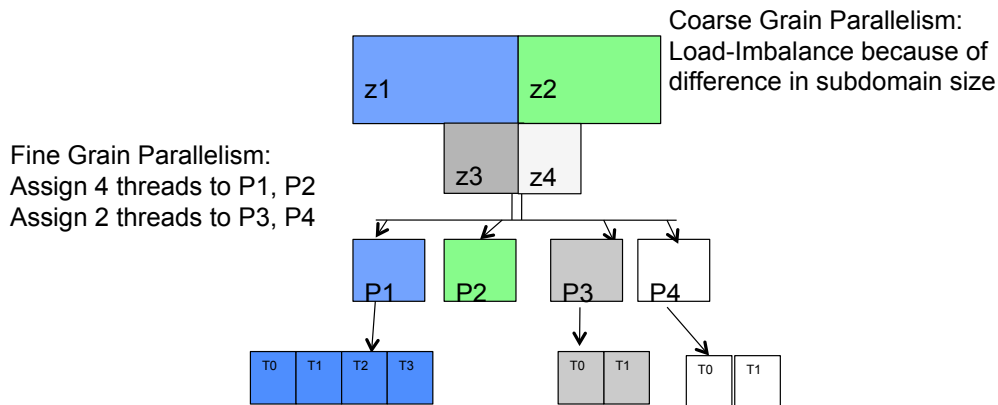
Coarse Grain Load-Balancing

- **Improve Load-Balance**
 - Restrict #MPI Processes
 - Exploit loop level parallelism instead

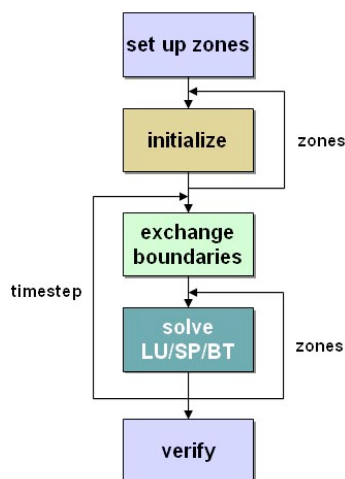


Fine Grain Load-Balancing

- **Improve Load-Balance on Fine Grain**
 - Assign more threads to MPI Process with high workload



The Multi-Zone NAS Parallel Benchmarks



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

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

Benchmark Characteristics

- **Aggregate sizes:**

- 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

Expectations:

- **BT-MZ: (Block-tridiagonal Solver)**

- #Zones: 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

Pure MPI: Load-balancing problems!

Good candidate for MPI+OpenMP

- **LU-MZ: (Lower-Upper Symmetric Gauss Seidel Solver)**

- #Zones: 16 (C, D, and E)
- Size of the zones identical:
 - no load-balancing required
 - limited parallelism on outer level

Limited MPI Parallelism:
→ MPI+OpenMP increases Parallelism

LU not used in this study because of small number of cores on the systems

- **SP-MZ: (Scalar-Pentadiagonal Solver)**

- #Zones: 256 (C), 1024 (D), 4096 (E)
- Size of zones identical
 - no load-balancing required

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

BT-MZ based on MPI/OpenMP

Coarse-grain MPI Parallelism

```
call omp_set_numthreads (weight)
do step = 1, itmax
  call exch_qbc(u, qbc, nx,...)
  call mpi_send/rcv
  do zone = 1, num_zones
    if (iam .eq. pzone_id(zone))
    then
      call comp_rhs(u,rsd,...)
      call x_solve (u, rhs,...)
      call y_solve (u, rhs,...)
      call z_solve (u, rhs,...)
      call add (u, rhs,...)
    end if
  end do
end do
...
```

Fine-grain OpenMP Parallelism

```
subroutine x_solve (u, rhs,
!$OMP PARALLEL DEFAULT(SHARED)
!$OMP& PRIVATE(i,j,k, isize,...)
isize = nx-1
!$OMP DO
  do k = 2, nz-1
    do j = 2, ny-1
      ...
      call lhsinit (lhs, isize)
      do i = 2, nx-1
        lhs(m,i,j,k)= ..
      end do
      call matvec ()
      call matmul ()....
    end do
  end do
end do
!$OMP END DO nowait
!$OMP END PARALLEL
```

NEC SX8:MPI/OpenMP/Vectorization

- Located at HLRS, Stuttgart, Germany
- 72 SX8 vector nodes with 8 CPUs each
- 12 TFlops peak performance
- Node-node interconnect IXS 16 GB/s per node
- **Compilation:**
sxmpif90 -C hopt -P openmp
- **Execute:**
export MPIMULTITASK=ON
export OMP_NUM_THREADS=<#num threads pr MPI proc>
mpirun -nn <#nodes> -nnp <#MPI procs per node> a.out
- **Vectorization is required to achieve good performance**
- **A maximum of 64 nodes (512 CPUs) were used for the study**

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
- **Caution:**
 - Data dependences can prevent vectorization
 - OpenMP parallelization might interfere with vectorization!

BT-MZ Cache Optimized Version

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

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

BT-MZ Vectorizable

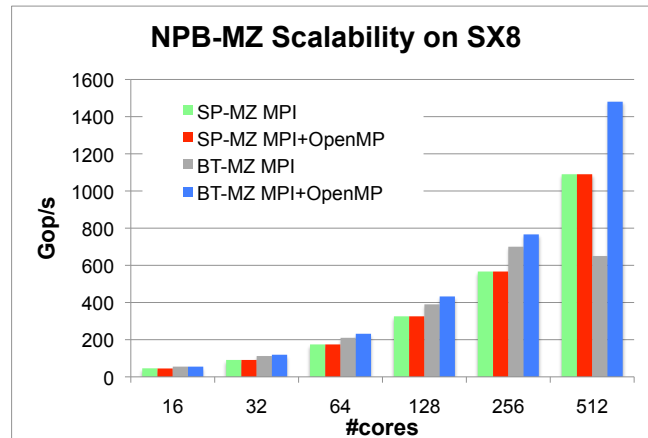
- SX8 requires vectorization:
 - Re-introduce 3D arrays
 - Loop interchange to remove data dependence from inner loop
 - manual procedure in-lining to allow vectorization
 - Note: OpenMP directives within routines prevented automatic inlining

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

$OMP PARALLEL DO
do k
  do j
    do i
      ...
      rhs_3d(i,j,k) = c * rhs_3d(i-1,j,k) + ....
```

Loop interchange yields vectorizable inner loop

NPB-MZ Class D Scalability on SX8

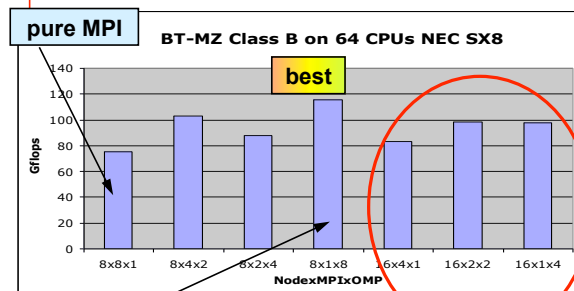


- Three dimensions of variation: Nodes, Processes per Node, Threads per Process
- Hybrid: Reported is the best performance for a given number of CPUs on a combination of Nodes x MPI x OMP
- SP-MZ performs best for pure MPI
- BT-MZ benefits from hybrid

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

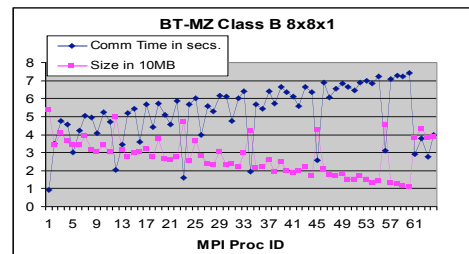
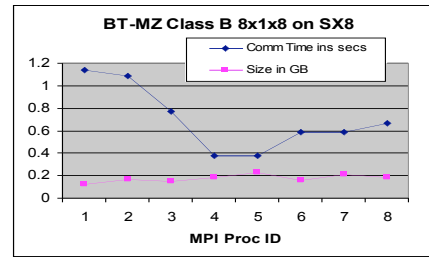


hybrid MPI+OpenMP

Does not use all available cores:
Bad!

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

- 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



Sun Constellation Cluster Ranger (1)

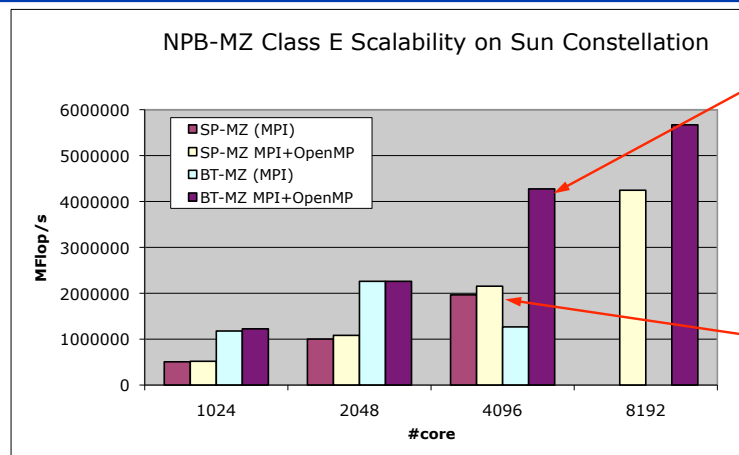
- Located at the Texas Advanced Computing Center (TACC), University of Texas at Austin (<http://www.tacc.utexas.edu>)
- 3936 Sun Blades, 4 AMD Quad-core 64bit 2.3GHz processors per node (blade), 62976 cores total
- 123TB aggregate memory
- Peak Performance 579 Tflops
- InfiniBand Switch interconnect
- 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 MMAPICH
 - setenv OMP_NUM_THREADS NTHREAD
 - ibrun numactl.sh bt-mz.exe
- numactl controls
 - Socket affinity: select sockets to run
 - Core affinity: select cores within socket
 - Memory policy: where to allocate memory

Default script for process placement available on Ranger

NPB-MZ Class E Scalability on Ranger



BT-MZ

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

SP-MZ

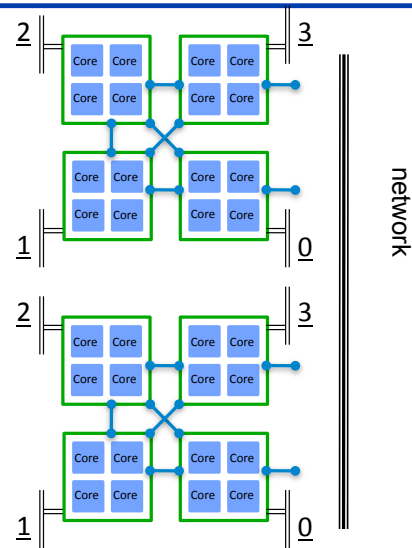
Pure MPI is already load-balanced.
But hybrid programming 9.6% faster

Unexpected!

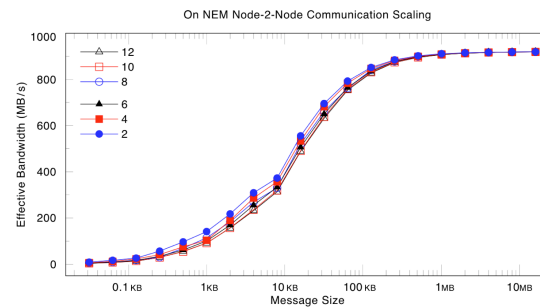
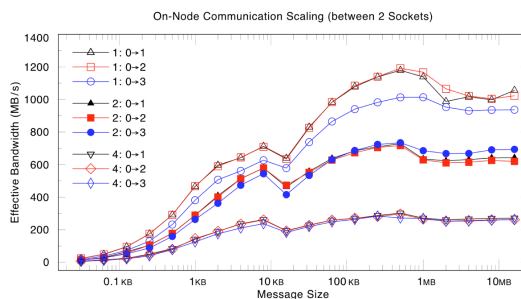
- Scalability in Mflops with increasing number of cores
- MPI/OpenMP: Best Result over all MPI/OpenMP combinations for a fixed number of cores
- Use of numactl essential to achieve scalability

Sun Constellation Cluster

- **Highly hierarchical**
- **Shared Memory:**
 - Cache-coherent, Non-uniform memory access (ccNUMA) Blade
- **Distributed memory:**
 - Network of ccNUMA blades
 - Core-to-Core
 - Socket-to-Socket
 - Blade-to-Blade
 - Chassis-to-Chassis

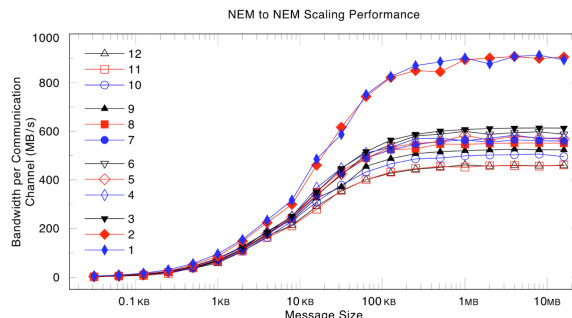


Ranger Network Bandwidth



MPI ping-pong micro benchmark results

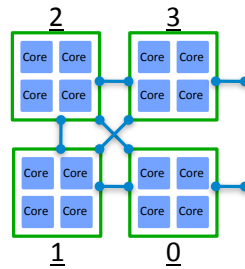
“Exploiting Multi-Level Parallelism on the Sun Constellation System”.
L. Koesterke, et. al., TACC,
TeraGrid08 Paper



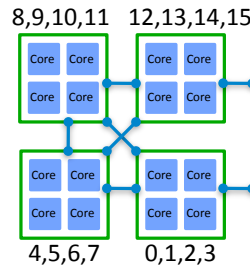
NUMA Control: Process Placement

- Affinity and Policy can be changed externally through `numactl` at the socket and core level.

Command: `numactl <options> ./a.out`

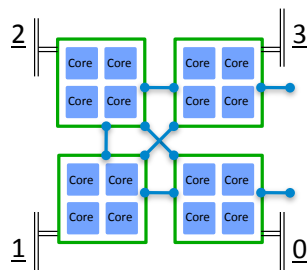


Socket References



Core References

NUMA Operations: Memory Placement



Memory: Socket References

- Memory allocation:**
- MPI – local allocation is best**
- OpenMP**
 - Interleave best for large, completely shared arrays that are randomly accessed by different threads
 - local best for private arrays
- Once allocated, a memory structure's is fixed**

NUMA Operations (cont. 3)

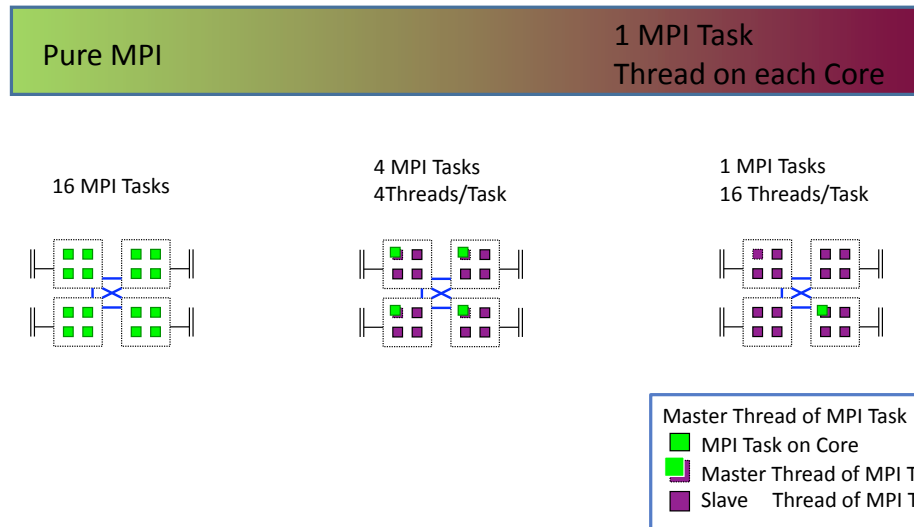
	cmd	option	arguments	description
Socket Affinity	numactl	-N	{0,1,2,3}	Only execute process on cores of this (these) socket(s).
Memory Policy	numactl	-l	{no argument}	Allocate on current socket.
Memory Policy	numactl	-i	{0,1,2,3}	Allocate round robin (interleave) on these sockets.
Memory Policy	numactl	--preferred=	{0,1,2,3} select only one	Allocate on this socket; fallback to any other if full .
Memory Policy	numactl	-m	{0,1,2,3}	Only allocate on this (these) socket(s).
Core Affinity	numactl	-C	{0,1,2,3, 4,5,6,7, 8,9,10,11, 12,13,14,15}	Only execute process on this (these) Core(s).

Hybrid Batch Script 4 tasks, 4 threads/task

for mvapich2

job script (Bourne shell)	job script (C shell)
<pre>... #! -pe 4way \$2 ... export OMP_NUM_THREADS=4 ibrun numa.sh</pre>	<pre>... #! -pe 4way 32 ... setenv OMP_NUM_THREADS 4 ibrun numa.csh</pre>
<pre>numa.sh #!/bin/bash export MV2_USE_AFFINITY=0 export MV2_ENABLE_AFFINITY=0 export VIADEV_USE_AFFINITY=0 #TasksPerNode TPN=`echo \$PE sed 's/way//'` [! \$TPN] && echo TPN NOT defined! [! \$TPN] && exit 1 socket=\$((\$PMI_RANK % \$TPN)) numactl -N \$socket -m \$socket ./a.out</pre>	<pre>numa.csh #!/bin/tcsh setenv MV2_USE_AFFINITY 0 setenv MV2_ENABLE_AFFINITY 0 setenv VIADEV_USE_AFFINITY 0 #TasksPerNode set TPN = `echo \$PE sed 's/way//'` if(! \${%TPN}) echo TPN NOT defined! if(! \${%TPN}) exit 0 @ socket = \$PMI_RANK % \$TPN numactl -N \$socket -m \$socket ./a.out</pre>

Modes of Hybrid Operation



Numactl: Using Threads across Sockets

bt-mz.1024x8 yields
best load-balance

```
-pe 2way 8192
export OMP_NUM_THREADS=8

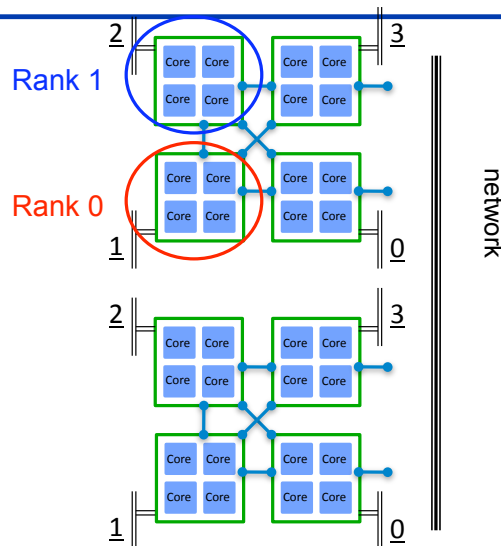
my_rank=$PMI_RANK
local_rank=$(( $my_rank % $myway ))
numnode=$(( $local_rank + 1 ))
```

Original:

```
numactl -N $numnode -m $numnode $*
```

Bad performance!

- Each process runs 8 threads on 4 cores
- Memory allocated on one socket



Numactl: Using Threads across Sockets

```
bt-mz.1024x8
export OMP_NUM_THREADS=8

my_rank=$PMI_RANK
local_rank=$(( $my_rank % $myway ))
numnode=$(( $local_rank + 1 ))
```

Original:

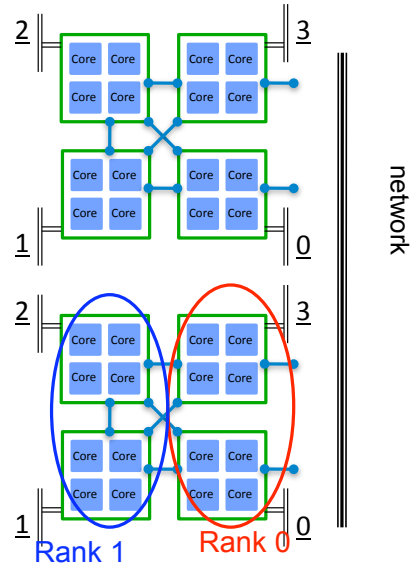
```
numactl -N $numnode -m $numnode $*
```

Modified:

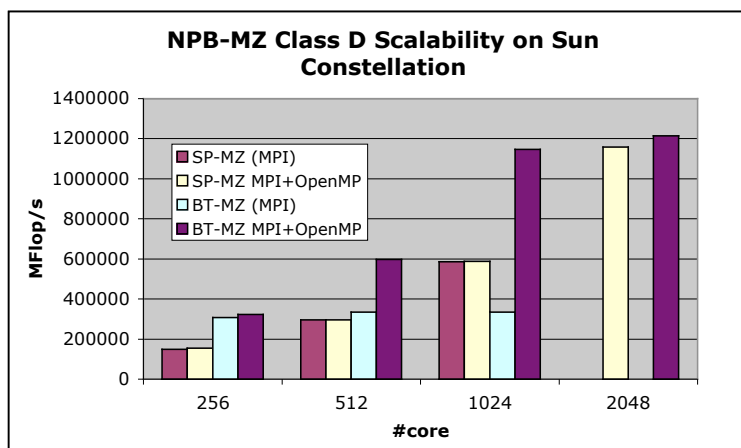
```
if [ $local_rank -eq 0 ]; then
    numactl -N 0,3 -m 0,3 $*
else
    numactl -N 1,2 -m 1,2 $*
fi
```

Achieves Scalability!

- Process uses cores and memory across 2 sockets
- Suitable for 8 threads



NPB-MZ Class D Scalability on Ranger

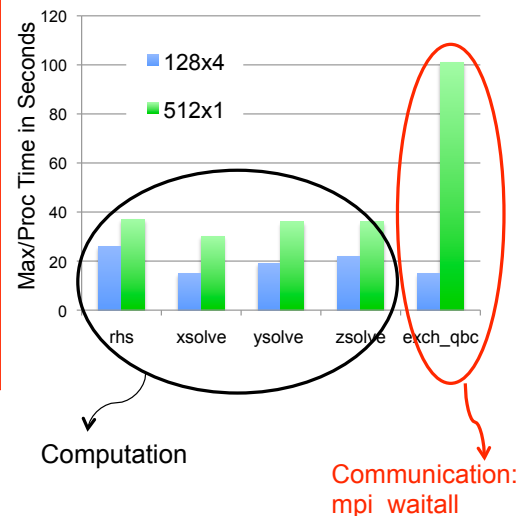


- SP-MZ hybrid outperforms SP-MZ pure MPI for
- Class D
- Does not meet expectations!

BT-MZ: Combining MPI and OpenMP

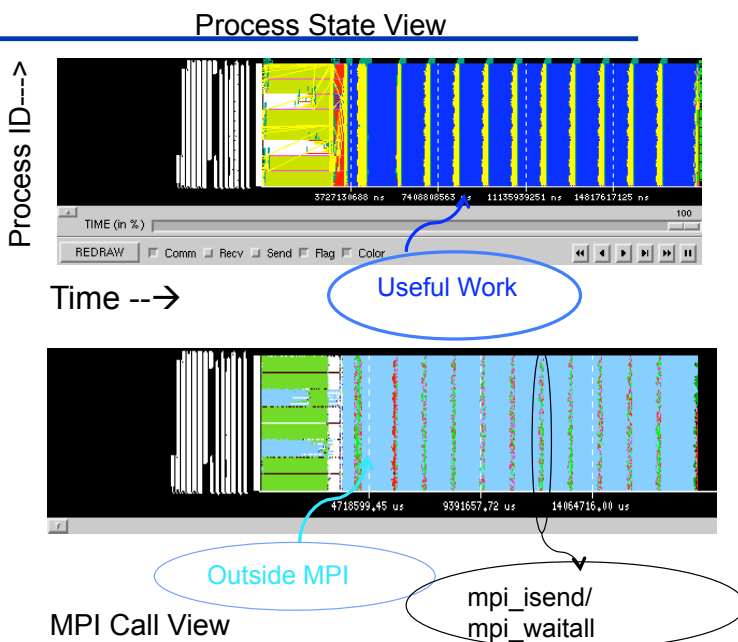
- Performance Metrics Class D
- 128x4 :
 - 4 MPI Processes per node
 - 1 MPI Process per socket
 - 595 Gflops
 - Total time: 86.5 sec
 - Workload: 536962/523124 points
- 512x1:
 - 16 MPI Processes per node
 - 4 MPI Processes per socket
 - 334 Gflops
 - Total time: 154 sec
 - Workload: 243236/14450 points

Subroutine Timings Class D



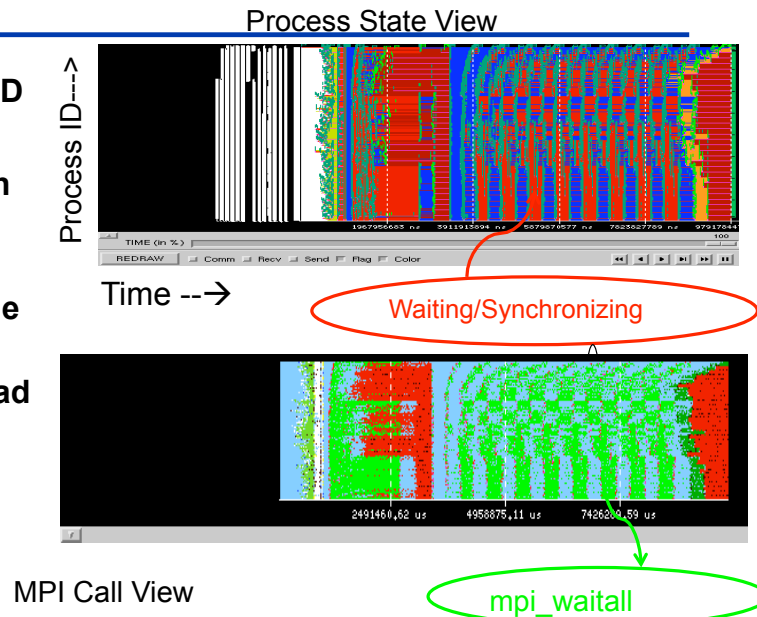
Execution Timelines for BT-MZ 128 MPI Processes

- Paraver Performance Analysis System
<http://www.cepba.upc.es/paraver/>
- 10 time steps Class D
- 128 MPI Processes
- Most of the time spent doing useful work
- Small amount of time in communication
- Well load-balanced

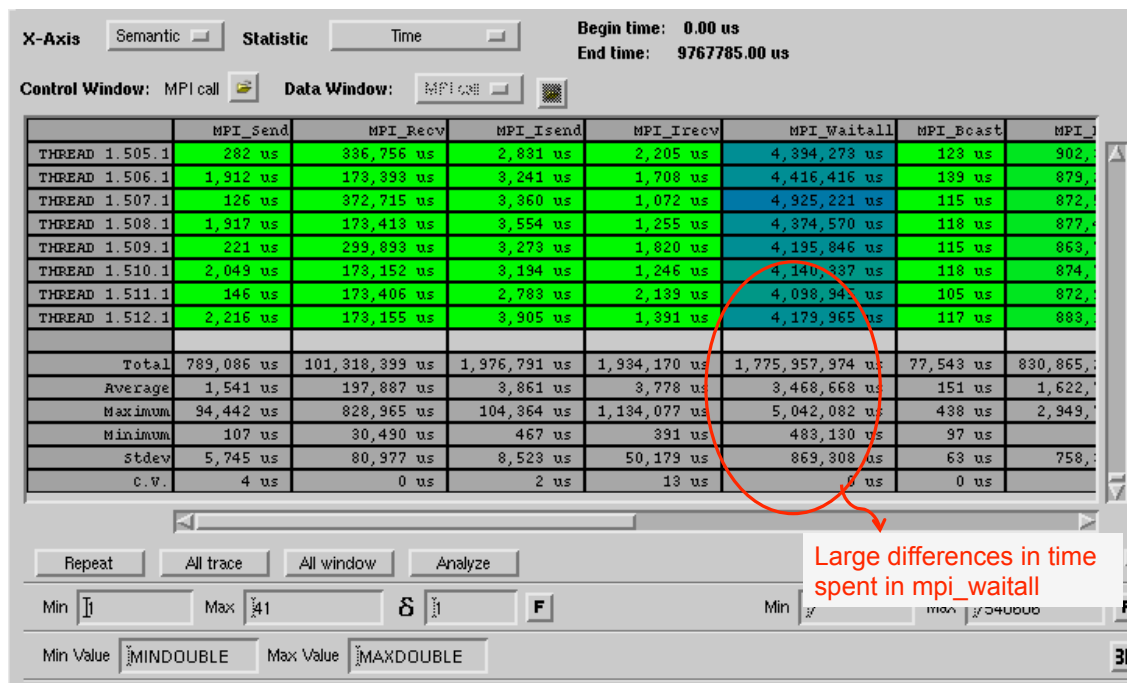


Execution Timelines for BT-MZ 512 MPI Processes

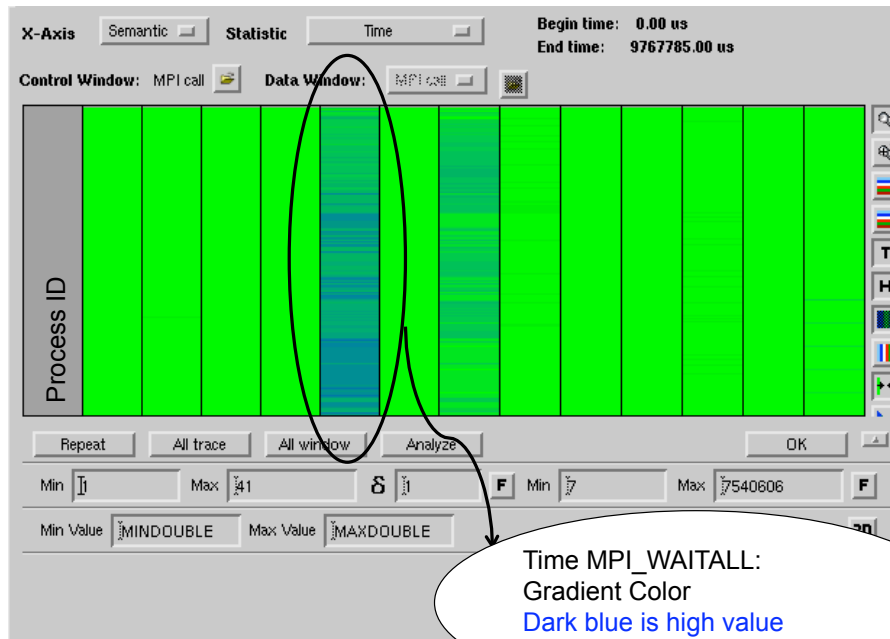
- 10 time steps Class D
- 512 MPI Processes
- A lot of time spent in Waiting and Synchronization
- Large amount of time spent in `mpi_waitall`
- Unbalanced Workload on MPI Level



Communication Timings BT-MZ Class D 512 Processes



Compressed View of MPI Calls BT-MZ 512 Processes



SP-MZ based on MPI/OpenMP

Coarse-grain MPI Parallelism

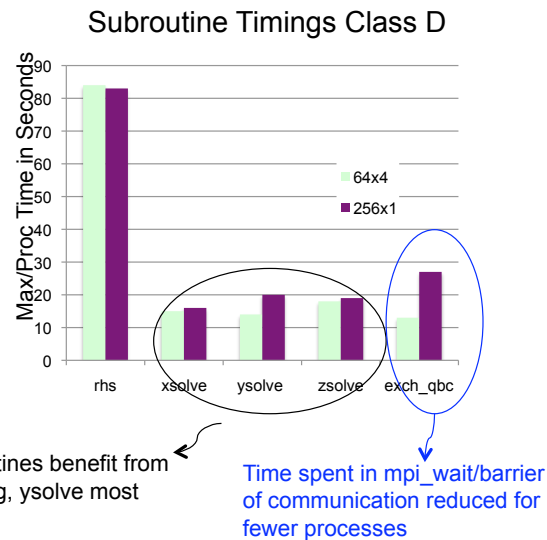
```
call omp_set_numthreads (weight)
do step = 1, itmax
  call exch_qbc(u, qbc, nx,...)
  call mpi_send/recv
  do zone = 1, num_zones
    if (iam .eq.pzone_id(zone))
    then
      call txinvr(u,rsd,...)
      call comp_rhs(u,rsd,...)
      call x_solve (u, rhs,...)
      call y_solve (u, rhs,...)
      call z_solve (u, rhs,...)
      call add (u, rhs,...)
    end if
  end do
end do
```

Fine-grain OpenMP Parallelism

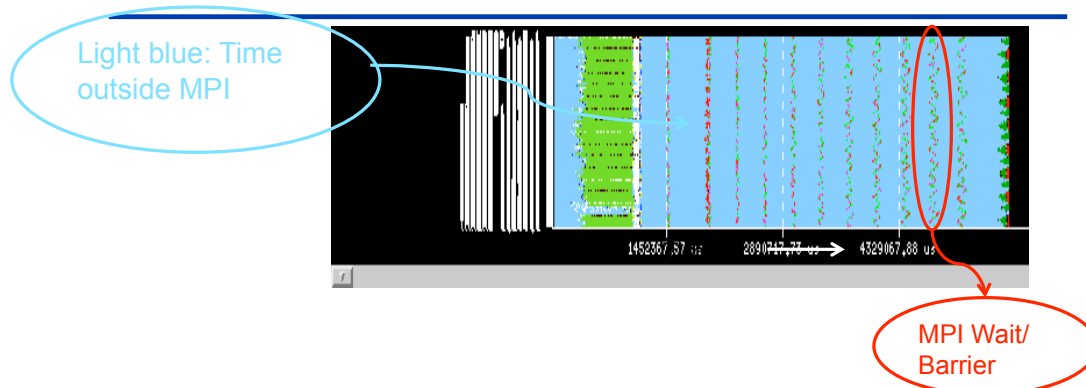
```
subroutine x_solve (u, rsd
!$OMP PARALLEL DEFAULT(SHARED)
!$OMP& PRIVATE(i,j,k, isize...)
!$OMP DO
  do k = 2, nz-1
    do j = 2, ny-1
      ...
      do i = 2, nx-1
        lhs(m,i,j,k)= ..
        rhs(m,i,j,k) =
      end do
    end do
  end do
!$OMP END DO nowait
!$OMP END PARALLEL
call ninvr (rhs,...)
```

SP-MZ: Combining MPI and OpenMP

- Performance Metrics Class D
- 64x4 :153 Gflops
 - Total time:169
 - Communication:
 - Count: 4531 isend /MPI Proc
 - Size: 802 MB / MPI Proc
 - Total Size:~51328MB
- 256x1: 148 GFlops
 - Total time:174
 - Communication:
 - Count: 2004 isend/MPI Proc
 - Size:436 MB/MPI Proc
 - Total Size:~110000MB



SP-MZ Execution on 256 Processes



- Timeline view of MPI calls for 10 iterations on 256 MPI Processes
- Little time spent in MPI calls
- No workload imbalance
- Light unbalance develops during the course of the execution: Time spent in MPI_Wait/Barrier increases over multiple iterations.

Analysis of SP-MZ Execution

	MPI_Isend	MPI_Irecv	MPI_Waitall
THREAD 1.251.1	339 us	308 us	11,418 us
THREAD 1.252.1	1,311 us	338 us	27,833 us
THREAD 1.253.1	1,072 us	183 us	12,868 us
THREAD 1.254.1	1,181 us	209 us	10,430 us
THREAD 1.255.1	960 us	171 us	14,584 us
THREAD 1.256.1	850 us	664 us	15,784 us
Total	189,493 us	83,496 us	3,218,334 us
Average	740 us	326 us	12,572 us
Maximum	2,206 us	1,775 us	32,609 us
Minimum	178 us	37 us	2,874 us
Stddev	380 us	238 us	5,912 us
C.V.	1 us	1 us	0 us

	MPI_Isend	MPI_Irecv	MPI_Waitall
THREAD 1.251.1	117 us	87 us	15,555 us
THREAD 1.252.1	103 us	130 us	31,489 us
THREAD 1.253.1	162 us	101 us	21,361 us
THREAD 1.254.1	109 us	154 us	18,423 us
THREAD 1.255.1	149 us	93 us	17,620 us
THREAD 1.256.1	108 us	146 us	28,191 us
Total	28,708 us	24,644 us	6,255,887 us
Average	112 us	96 us	24,437 us
Maximum	211 us	211 us	56,184 us
Minimum	45 us	36 us	3,614 us
Stddev	41 us	46 us	11,458 us
C.V.	0 us	0 us	0 us

Iteration 1

Iteration 10

Increased amount of time in MPI_waitall in later iterations!

IPM Performance Monitor

- **IPM:**
 - Integrated Performance Monitoring
 - <http://ipm-hpc.sourceforge.net/home.html>
- **Summary at end of program**
- **Detailed Information:**
 - [Example: BT-MZ 1024x1](#)
 - [Hostlist](#)
 - [Executable](#)

IPM Summary Information

SP-MZ 256x1

Replicated Data
MPI Message Buffer

SP-MZ 64x4

```
##### IPMv0.922#####
#
# command : ./bin/sp-mz.D.256 (completed)
# host : i101-402/x86_64.Linux
# start : 09/25/08/00:04:30
# stop : 09/25/08/00:04:37
# gbytes : 3.96945e+01 total
#
# mpi_tasks : 256 on 16 nodes
# wallclock : 6.731512 sec
# %comm : 32.85
# gflop/sec : 3.66761e+02 total
#
#####
# region : * [ntasks] = 256
#
# [total] <avg> min max
# entries 256 1 1 1
# wallclock 1723.16 6.73109 6.7306 6.73151
# user 1910.09 7.46129 7.14445 7.53647
# system 43.8986 0.171479 0.068004 0.240015
# mpi 566.162 2.21157 2.01036 2.39116
# %comm 32.85 29.8662 35.5239
# gflop/sec 366.761 1.43266 1.14317 1.48659
# gbytes 39.6945 0.155057 0.154293 0.247147
#
# PAPI_RES_STL 2.46886e+12 9.64398e+09 7.69526e+09 1.0007e+10
# PAPI_TOT_CYC 3.72385e+12 1.45463e+10 1.01678e+10 1.46223e+10
# PAPI_L1_DCM 9.4849e+09 3.70504e+07 2.98643e+07 1.88519e+08
# PAPI_L2_DCM 2.73355e+09 1.03123e+07 1.02886e+07 1.56149e+07
#
# [time] [calls] <%api> <%wall>
# MPI_Barrier 465.088 512 82.15 26.99
# MPI_Waitall 50.3943 11264 8.90 2.92
# MPI_Recv 23.905 1275 4.22 1.39
# MPI_Bcast 13.6376 3072 2.42 0.79
# MPI_Reduce 12.0214 768 2.12 0.70
# MPI_Isend 0.286147 11264 0.05 0.02
# MPI_Sendrecv 0.222154 11856 0.04 0.01
# MPI_Comm_size 0.132859 1280 0.02 0.01
# MPI_Allreduce 0.132542 512 0.02 0.01
# MPI_Irecv 0.131822 11264 0.02 0.01
# MPI_Allgather 0.0920441 256 0.02 0.01
# MPI_Send 0.066374 1275 0.01 0.00
# MPI_Comm_rank 0.00157302 1808 0.00 0.00
#####
```

```
##### IPMv0.922#####
#
# command : ./bin/sp-mz.D.64 (completed)
# host : i111-403/x86_64.Linux
# start : 09/25/08/00:07:56
# stop : 09/25/08/00:08:04
# gbytes : 1.80950e+01 total
#
# mpi_tasks : 64 on 16 nodes
# wallclock : 4.912151 sec
# %comm : 6.31
# gflop/sec : 1.02274e+02 total
#
#####
# region : * [ntasks] = 64
#
# [total] <avg> min max
# entries 64 1 1 1
# wallclock 314.375 4.9121 4.91206 4.91215
# user 1228.28 19.1919 19.1292 19.2932
# system 16.789 0.262328 0.16401 0.32802
# mpi 19.8324 0.309881 0.18546 0.423585
# %comm 6.30845 3.77559 8.62331
# gflop/sec 102.274 1.59802 1.56701 1.62619
# gbytes 18.095 0.282735 0.281651 0.31538
#
# PAPI_RES_STL 5.02383e+11 7.84973e+09 7.69737e+09 7.98811e+09
# PAPI_TOT_CYC 7.02834e+11 1.09818e+10 1.08508e+10 1.10058e+10
# PAPI_L1_DCM 2.02927e+09 3.17073e+07 3.01005e+07 4.4795e+07
# PAPI_L2_DCM 7.6661e+08 1.19783e+07 1.14856e+07 1.26603e+07
#
# [time] [calls] <%api> <%wall>
# MPI_Waitall 9.60437 2816 48.43 3.06
# MPI_Barrier 4.7828 128 24.12 1.52
# MPI_Recv 2.77577 315 14.00 0.88
# MPI_Bcast 1.33887 768 7.05 0.44
# MPI_Reduce 0.802732 192 4.05 0.26
# MPI_Sendrecv 0.328601 2384 1.66 0.10
# MPI_Isend 0.052586 2816 0.27 0.02
# MPI_Send 0.0410128 315 0.21 0.01
# MPI_Irecv 0.0308591 2816 0.16 0.01
# MPI_Allreduce 0.00675132 128 0.03 0.00
# MPI_Allgather 0.00525232 64 0.03 0.00
# MPI_Comm_size 0.00245019 320 0.01 0.00
# MPI_Comm_rank 0.000310361 464 0.00 0.00
#####
```

SP-MZ:Hybrid vs Pure MPI

•Performance metrics for Class D:

•64x4:

- Workload: HW FP OPS:91G x 4 per MPI Process
- Communication:
 - Time (sec): 3.4sec max
 - Count: 4531 isend per MPI Process
 - Size: 802MB per MPI Process
 - Total size: ~51328MB

•256x1:

- Workload: HW FP OPS: 91G per MPI Process
- Communication:
 - Time (sec):17 sec Max
 - Count: 2004 isend per MPI Process
 - Size: 436 MB Max, 236MB Min
 - Total Size: ~110000MB.

mpi_waitall

Imbalance

•Performance issues for pure MPI:

- Large amount of data communicated (2 x hybrid)
- Imbalance in message size across processes

Hybrid Programming – Outline

Author:
J. Hager

- Introduction / Motivation
 - Programming Models on Clusters of SMP nodes
 - Practical “How-To” on hybrid programming & Case Studies
 - Mismatch Problems & Pitfalls
 - Application Categories that Can Benefit from Hybrid Parallelization/ Case Studies
- Summary on Hybrid Parallelization

Author:
Rabenseifner

Elements of Successful Hybrid Programming

- **System Requirements:**
 - Some level of shared memory parallelism, such as within a multi-core node
 - Runtime libraries and environment to support both models
 - Thread-safe MPI library
 - Compiler support for OpenMP directives, OpenMP runtime libraries
 - Mechanisms to map MPI processes onto cores and nodes
- **Application Requirements:**
 - Expose multiple levels of parallelism
 - Coarse-grained and fine-grained
 - Enough fine-grained parallelism to allow OpenMP scaling to the number of cores per node
- **Performance:**
 - Highly dependent on optimal process and thread placement
 - No standard API to achieve optimal placement
 - Optimal placement may not be known beforehand (i.e. optimal number of threads per MPI process) or requirements may change during execution
 - Memory traffic yields resource contention on multi-core nodes
 - Cache optimization more critical than on single core nodes

Recipe for Successful Hybrid Programming

- **Familiarize yourself with the layout of your system:**
 - Blades, nodes, sockets, cores?
 - Interconnects?
 - Level of Shared Memory Parallelism?
- **Check system software**
 - Compiler options, MPI library, thread support in MPI
 - Process placement
- **Analyze your application:**
 - Does MPI scale? If not, why?
 - Load-imbalance => OpenMP might help
 - Too much time in communication? Load-imbalance? Workload too small?
 - Does OpenMP scale?
- **Performance Optimization**
 - Optimal process and thread placement is important
 - Find out how to achieve it on your system
 - Cache optimization critical to mitigate resource contention

Hybrid Programming: Does it Help?

- **Hybrid Codes provide these opportunities:**
 - Lower communication overhead
 - Few multi-threaded MPI processes vs Many single-threaded processes
 - Fewer number of calls and smaller amount of data communicated
 - Lower memory requirements
 - Reduced amount of replicated data
 - Reduced size of MPI internal buffer space
 - May become more important for systems of 100's or 1000's cores per node
 - Provide for flexible load-balancing on coarse and fine grain
 - Smaller #of MPI processes leave room to assign workload more even
 - MPI processes with higher workload could employ more threads
 - Increase parallelism
 - Domain decomposition as well as loop level parallelism can be exploited

YES, IT CAN!