Hybrid MPI and OpenMP Programming on Clusters of Multi-core Nodes

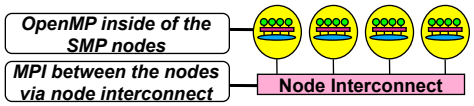
Gabriele Jost – Texas Advanced Computing Center, The University of Texas at Austin
*Rolf Rabenseifner – High Performance Computing Center Stuttgart (HLRS), Germany
*Georg Hager – Erlangen Regional Computing Center (RRZE), University of Erlangen-Nuremberg, Germany
*Bob Robins, NorthWest Research Associates, Inc, Redmond, WA.
*author only—not speaking

Hybrid Programming MPI/OpenMP

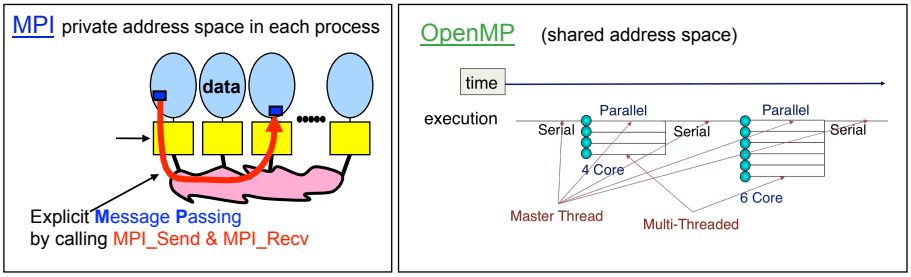
- Programming Models on Clusters of SMP nodes
- Remarks on MPI and OpenMP
- Hybrid MPI/OpenMP Case Studies:
  - Pitfalls and Opportunities
- Summary on Hybrid Parallelization

Programming Models for Hierarchical Systems

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

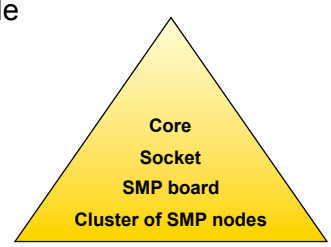


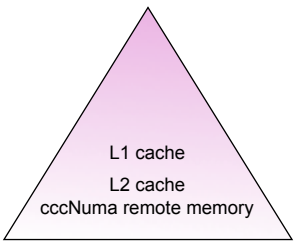
- **Other: Virtual shared memory systems, PGAS, HPF, …**



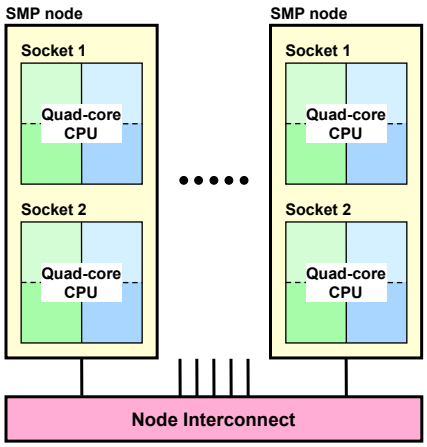
MPI on Clusters of Multi-Core Nodes

- Common Characteristics of Large Scale Parallel Systems:
  - Hierarchical systems
  - Different characteristics for intra-node and inter-node communication
  - Different latencies for shared memory access
- Application specific issues, eg:
  - MPI Scalability Limitations
  - Memory Consumption
- MPI Performance is affected by:
  - Data and process layout across node
  - Network Latency and Bandwidth
  - MPI Stack Latency and Bandwidth
  - Shared memory access
  - Resource contention for network and memory access within a node

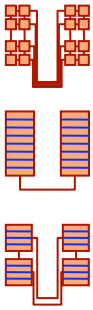




Hybrid MPI/OpenMP Programming



- Seems like the natural choice, but…..
- … which model is the fastest?
- MPI everywhere?
- Fully hybrid MPI & OpenMP?
- Something between? (Mixed model)
- Often hybrid programming slower than pure MPI



?

Comparison of MPI and OpenMP

MPI

- **Memory Model**
  - Data private by default
  - Data accessed by multiple processes needs to be explicitly communicated
- **Program Execution**
  - Parallel execution from start to beginning
- **Parallelization**
  - Process based
  - Domain decomposition
  - Explicitly programmed by user

- OpenMP
- **Memory Model**
  - Data shared by default
  - Access to shared data requires synchronization
  - Private data needs to be explicitly declared
- **Program Execution**
  - Fork-Join Model
- **Parallelization**
  - Thread based
  - Typically on loop level, incremental
  - Based on compiler directives

Support of Hybrid Programming

MPI

- MPI-1 no concept of threads
- MPI-2:
  - Thread support
  - MPI_Init_thread

OpenMP

- 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

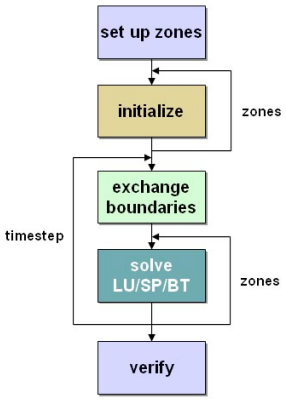
Syntax: call MPI_Init_thread(irequired, iprovided, ierr)
int MPI_Init_thread(int *argc, char ***argv, int required, int *provided)
int MPI::Init_thread(int& argc, char**& argv, int required)

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.

eg: overlap communication and

eg: p2p thread sync. Across MPI procs

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 D: 1632 x 1216 x 34 grid points
Class E: 4224 x 3456 x 92 grid points

Expectations:

BT-MZ: (Block tridiagonal simulated CFD application)
Alternative Directions Implicit (ADI) method
#Zones: 1024 (D), 4096 (E)
Size of the zones varies widely:
large/small about 20
requires multi-level parallelism to achieve a good load-balance

Pure MPI: Load-balancing problems!
Good candidate for MPI+OpenMP

LU-MZ: (LU decomposition simulated CFD application)
SSOR method (2D pipelined method)
#Zones: 16 (all Classes)
Size of the zones identical:
no load-balancing required
limited parallelism on outer level

Limitted MPI Parallelism:
→ MPI+OpenMP increases Parallelism

SP-MZ: (Scalar Pentadiagonal simulated CFD application)
#Zones: 1024 (D), 4096 (E)
Size of zones identical
no load-balancing required

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

Using MPI/OpenMP

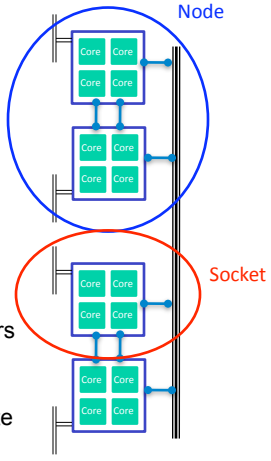
```
call omp_set_numthreads (weight)
do step = 1, itmax
  call exch_qbc(u, qbc, nx,…)

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

end do
  ...
```

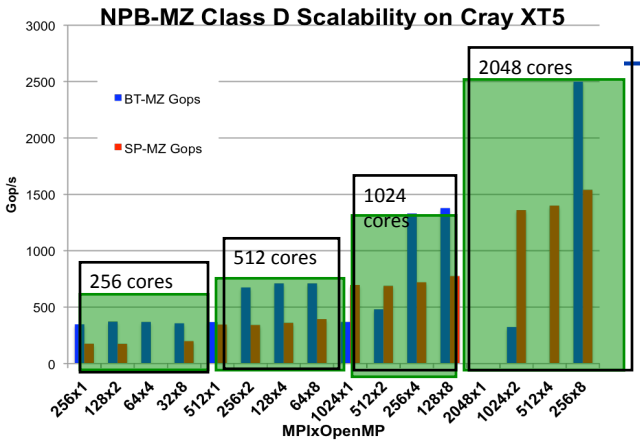
call mpi_send/recv

```
subroutine zsolve(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
          u(m,i,j,k)=
            dt*rsd(m,i,j,k-1)
        end do
      end do
    end do
```


Cray XT5

- **Results obtained by the courtesy of the HPCMO Program and the Engineer Research and Development Center Major Shared Resource Center, Vicksburg, MS (http://www.erdc.hpc.mil/index)**
- **Cray XT5 is located at the Arctic Region Supercomputing Center (ARSC) (http:// www.arsc.edu/resources/pingo)**
  - 432- Cray XT5 compute nodes with
    - 32 GB of shared memory per node (4 GB per core)
    - 2 - quad core 2.3 GHz AMD Opteron processors per node.
    - 1 - Seastar2+ Interconnect Module per node.
  - Cray Seastar2+ Interconnect between all compute and login nodes





Expected: #MPI Processes limitted to 1024

- **Performance reported for Class D 256-2048 cores**
  - SP-MZ pure MPI scales up to 1024 cores
  - SP-MZ MPI/OpenMP outperforms MPI: scales to 2048 cores
  - SP-MZ MPI/OpenMP outperforms pure MPI for 1024 cores
  - BT-MZ MPI/OpenMP scales to 2048 cores, outperforms pure MPI

Unexpected? Why?

Expected: Load-Imbalance for pure MPI

BT-MZ Load-Balance 32x4 vs 128x1

```
Table 2:  Load Balance across PE's by FunctionGroup

 Time % |     Time | Calls |Experiment=1
        |          |       |Group
        |          |       | PE[mmm]
        |          |       |  Thread

 100.0% | 1.782603 | 18662 |Total
|-----------------------------------
|  86.1% | 1.535163 |  7783 |USER
||----------------------------------
||   2.7% | 1.535987 |  6813 |pe.0
|||---------------------------------
3||   0.7% | 1.535987 |  6188 |thread.1
3||   0.7% | 1.535871 |  6188 |thread.3
3||   0.7% | 1.535829 |  6188 |thread.2
3||   0.7% | 1.466954 |  6813 |thread.0
|||=================================
||   2.7% | 1.535147 |  7783 |pe.18
|||---------------------------------
3||   0.7% | 1.535147 |  7072 |thread.1
3||   0.7% | 1.534995 |  7072 |thread.3
3||   0.7% | 1.534968 |  7072 |thread.2
3||   0.6% | 1.290502 |  7783 |thread.0
|||=================================
||   2.7% | 1.534239 |  7783 |pe.16
|||---------------------------------
3||   0.7% | 1.534239 |  7072 |thread.1
3||   0.7% | 1.534101 |  7072 |thread.3
3||   0.7% | 1.534076 |  7072 |thread.2
3||   0.6% | 1.268085 |  7783 |thread.0
|||=================================
```

bt-mz-C.32x4

```
Table 2:  Load Balance across PE's by FunctionGroup

 Time % |     Time | Calls |Group
        |          |       | PE[mmm]

 100.0% | 24.277514 | 38258 |Total
|-----------------------------------
|  54.2% | 13.166225 |  4545 |MPI
||----------------------------------
||   0.5% | 16.454993 |  4846 |pe.91
||   0.5% | 14.058598 |  2434 |pe.29
||   0.0% |  0.289479 |  2434 |pe.0
||==================================
|  44.9% | 10.894808 | 17983 |USER
||----------------------------------
||   0.7% | 23.205797 |  9093 |pe.0
||   0.3% | 10.084200 | 26873 |pe.110
||   0.3% |  8.070997 | 17983 |pe.91
|==================================
```

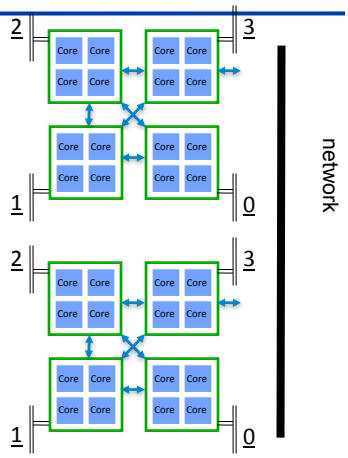
bt-mz-C.128x1

- CrayPat profile shows:
  - maximum, median, minimum PEs
  - bt-mz.C.128x1 large imbalance in User and MPI time
  - bt-mz.C.32x4 well balanced times

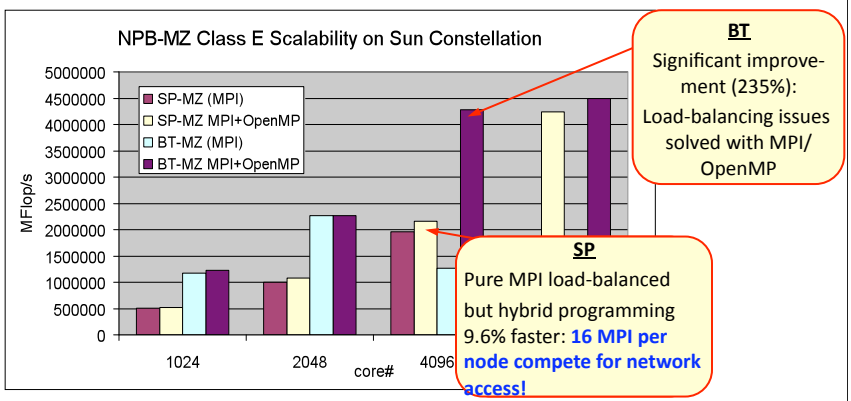
Sun Constellation Cluster Ranger

Sun Constellation (UT-TACC):

- Located at the Texas Advanced Computing Center, The University of Texas at Austin
- 4 2.3GHz AMD Quad-core Opteron per node
- 16 cores per node
- 32 Gbyte shared memory per node
- Infiniband Interconnect



SUN Constellation: NPB-MZ Class E Scalability



BT
Significant improve-ment (235%): Load-balancing issues solved with MPI/ OpenMP

SP
Pure MPI load-balanced but hybrid programming 9.6% faster: **16 MPI per node compete for network access!**

- **Performance reported for 1024 to 8192**
  - SP-MZ MPI/OpenMP outperforms MPI **WHY?**
  - BT-MZ MPI/OpenMP does not scale to 8192 **WHY?**

Hybrid:
SP: still scales
BT: does not scale **Bad default process placement!**

Linux *numactl*: Socket, Core, Memory Placement

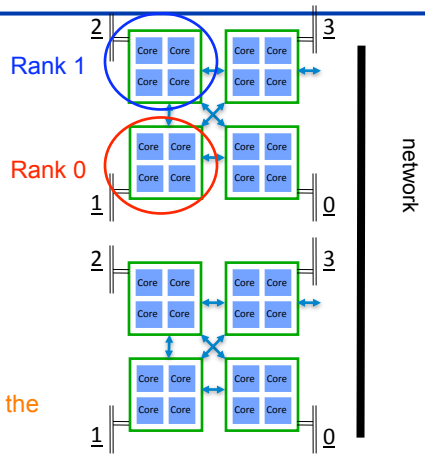
bt-mz.1024x8 yields
best load-balance

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

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

Default:

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

Bad performance:
- MPI process uses only cores within the local socket.
- 8 threads on 4 cores
- Memory allocated on one socket



numactl: Using Corres across Sockets

bt-mz.1024x8

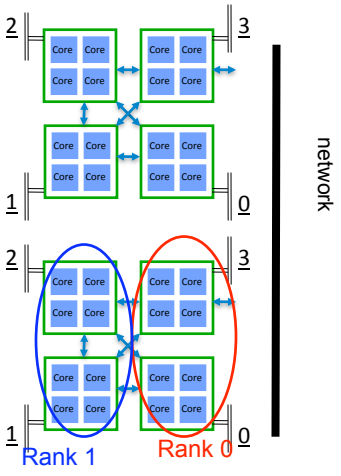
```
export OMP_NUM_THREADS=8

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

Default:

```
--------
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
```

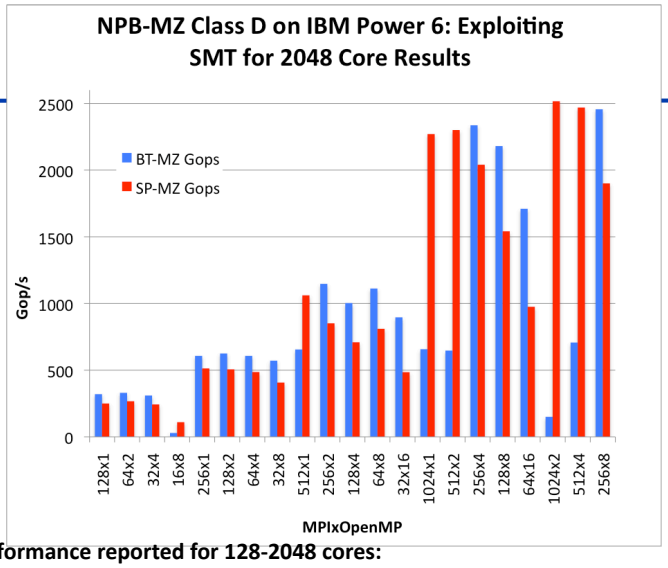


Achieves Scalability:
- Process uses cores and memory across 2 sockets
- Suitable for 8 threads

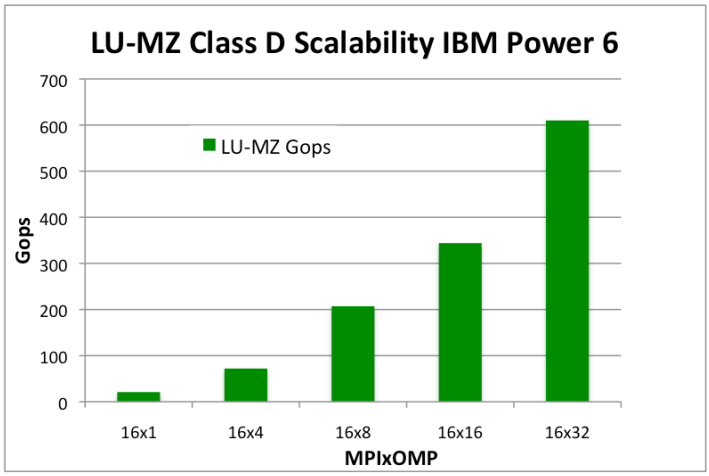
IBM Power 6

- Results obtained by the courtesy of the HPCMO Program and the Engineer Research and Development Center Major Shared Resource Center, Vicksburg, MS (http://www.erdc.hpc.mil/index)
- The IBM Power 6 System is located at (http://www.navo.hpc.mil/davinci_about.html)
- 150 Compute Nodes
- 32 4.7GHz Power6 Cores per Node (4800 cores total)
- 64 GBytes of dedicated memory per node
- QLOGOC Infiniband DDR interconnect
- IBM MPI: MPI 1.2 + MPI-IO
  - mpxlf_r -O4 -qarch=pwr6 -qtune=pwr6 -qsmp=omp

Flag was essential to achieve full compiler optimization in presence of OMP directives!



- **Performance reported for 128-2048 cores:**
  - BT-M, SP-MZ behave as expected
  - Only 1024 cores were available for the experiments
  - BT-MZ and SP-MZ show benefit from SMT: 2048 threads on 1024 cores

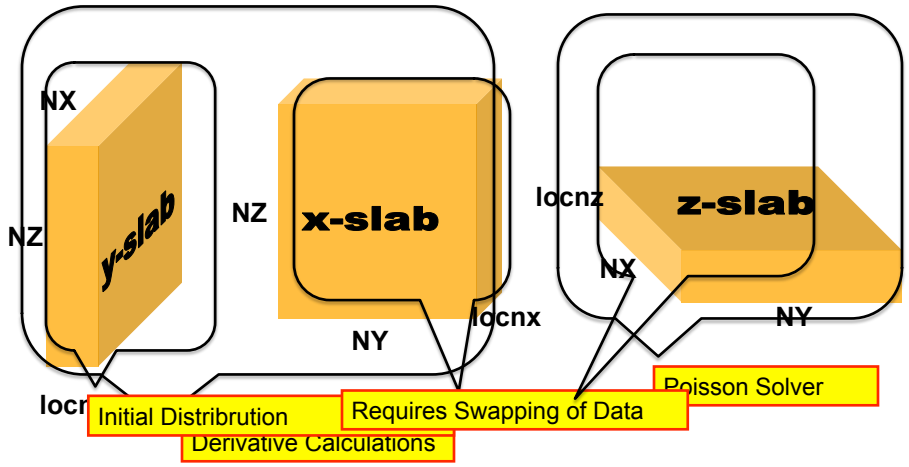


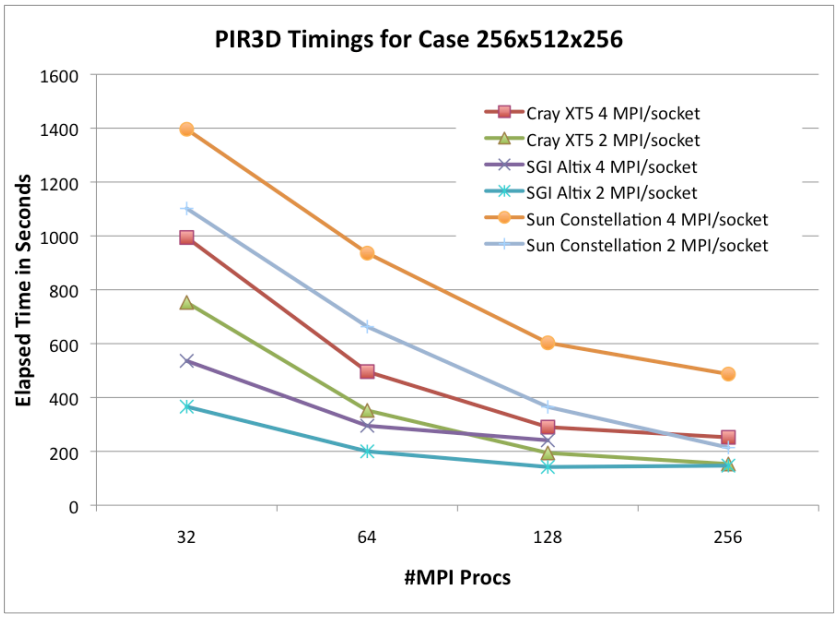
- LU-MZ significantly benefits from hybrid mode:
- Pure MPI limited to 16 cores, due to #zones = 16

PIR3D: Hybrid Programming in the Real World

- **PIR3D code developed by Bob Robins, Northwest Research Associates**
- **Physical problem:**
  - Simulate environmental effects on evolution of trailing vortices of underwater vehicles
- **Numerical Approach:**
  - Solve 3-D (or 2-D) Boussinesq equations for incompressible fluid
  - FFT's for horizontal derivatives (periodic BC)
  - Higher-order compact scheme for vertical derivatives
  - 2nd order Adams-Bashforth time-stepping
  - (projection method to ensure incompressibility –
  - requires solution to Poisson's Equation at every time step)
  - Periodic smoothing to control small-scale energy – compact approach in vertical, FFT approach in horizontal
  - Diagnostics – normalized divergence to check incompressibility, timings

1D Domain Decomposition





CrayPat MPI Performance Statistics for Cray XT5

4 cores per socket

Table 1: Profile by Function

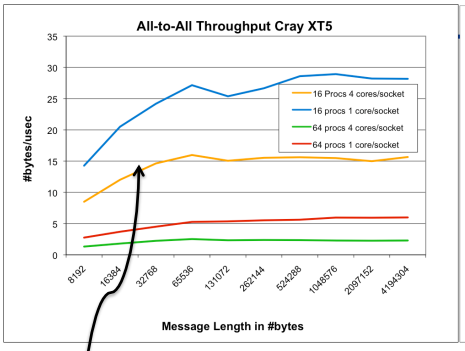
Samp %	Samp	Group
40.9%	279404	USER
8.4%	57437	dcalc_
5.0%	34240	getdiv_
4.1%	28323	rvcalc_
4.0%	27202	csfft_
2.3%	15693	swapyx_
1.5%	10051	swapxy_
29.9%	204411	MPI
16.1%	109624	mpi_waitall_
4.1%	28253	mpi_send_
3.9%	26565	mpi_ibsend_
3.3%	22363	mpi_irecv_
1.9%	13100	mpi_bsend_
29.1%	198881	ETC
6.9%	46856	dgtts2_
4.6%	31179	_c_mcopy8
4.3%	29496	daxpy_k
1.5%	10027	hc2cbdftv_8
1.2%	8117	dgbmv_n

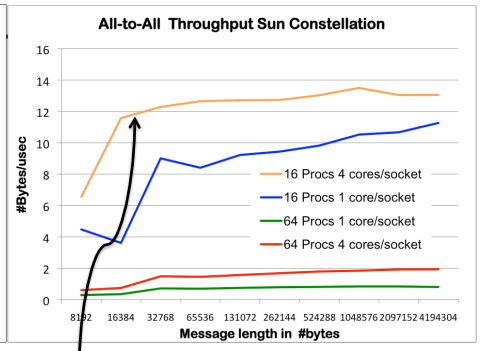
1 core per socket

Table 1: Profile by Function

Samp %	Samp	Group
100.0%	442157	Total
40.7%	179890	USER
10.9%	48416	dcalc_
4.9%	21543	getdiv_
4.3%	19064	rvcalc_
3.1%	13795	csfft_
2.6%	11531	swapyx_
1.6%	6941	swapxy_
1.3%	5679	scfft_
38.2%	169084	ETC
10.4%	46117	dgtts2_
6.7%	29648	daxpy_k
4.3%	18820	__c_mcopy8
2.1%	9194	hc2cbdftv_8
1.8%	8108	dgbmv_n
21.1%	93183	MPI
7.3%	32290	mpi_waitall_
4.7%	20944	mpi_ibsend_
3.5%	15558	mpi_irecv_
2.5%	10862	mpi_send_
2.2%	9755	mpi_bsend_

All-to-All Throughput

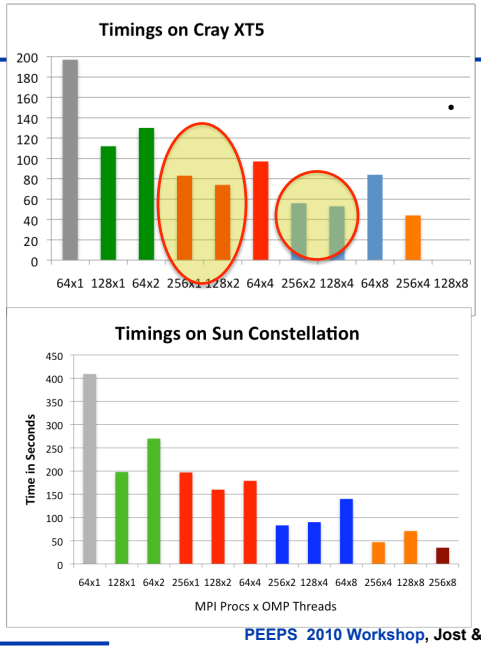




Inter-Node Communication requires network access.

Intra-Node Communication only! No network access required.

Hybrid Timings for Case 512x256x256



Adding OpenMP to loops in 3 routines:

- increases the number of usable cores
- 128x2 outperforms 256x1 on 256 cores,128x4 better than 256x2 on 512 cores
- Time distributed across many routines: Need to identify more time consuming loops!

Most of the performance due to "spacing" of MPI. About 12% improvement due to OpenMP

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 and fine
  - Enough fine-grained parallelism to allow OpenMP scaling to the number of cores per node
- **Pitfalls:**
  - Highly dependent on optimal process and thread placement
  - No standard API to achieve optimal placement
  - Optimal placement may not be be known beforehand (i.e. optimal number of threads per MPI process) or requirements may change during execution
  - OpenMP:
    - impact on compiler optimization
    - impact of ccNUMA remote memory access

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!