Parallel Programming Models on Hybrid Systems

MPI + OpenMP and other models on clusters of SMP nodes

Rolf Rabenseifner
rabenseifner@hlrs.de
University of Stuttgart
High-Performance Computing-Center Stuttgart (HLRS)
www.hlrs.de

Lecture at the University of Heidelberg, June 20, 2003

Outline

• Motivation [slides 3–7]
• Major parallel programming models [8–14]
• Programming models on hybrid systems [15–56]
  – Overview [15]
  – Technical aspects with thread-safe MPI [16–18]
  – Mismatch problems with pure MPI and hybrid MPI+OpenMP [19–40]
    • Topology problem [20]
    • Unnecessary intra-node comm. [21]
    • Inter-node bandwidth problem [22–35]
    – Comparison I: Two experiments
    • Sleeping threads and saturation problem [35]
    • Additional OpenMP overhead [36]
    • Overlapping comm. and comp. [41–47]
    – Comparison II: Theory + experiment
  – Pure OpenMP [48–56]
    – Comparison III
• No silver bullet / optimization chances / other concepts [58–62]
• Acknowledgments & Conclusions [63–64]
Motivation

- HPC systems
  - often clusters of SMP nodes
  - i.e., hybrid architectures
- Using the communication bandwidth of the hardware
- Minimizing synchronization = idle time

} optimal usage of the hardware

- Appropriate parallel programming models / Pros & Cons

Hitachi SR 8000-F1/112 (Rank 5 in TOP 500 / June 2000)

- System:
  - 168 nodes,
  - 2.016 TFLOP/s peak
  - 1.65 TFLOP/s Linpack
  - 1.3 TB memory
- Node:
  - 8 CPUs, 12 GFLOP/s
  - 6 GB, SMP
  - pseudo-vector
  - ext. b/w: 950 MB/s
- CPU:
  - 1.5 GFLOP/s, 375 MHz
  - 4 GB/s memory b/w
- Installed: 1.Q 2000 at LRZ
- Extended: 1.Q. 2002 (from 112 to 168 nodes)
Earth Simulator Project ESRDC / GS 40 (NEC)

- Virtual Earth - simulating
  - Climate change (global warming)
  - El Niño, hurricanes, droughts
  - Air pollution (acid rain, ozone hole)
  - Diastrophism (earthquake, volcanism)
- Installation: 2002
  http://www.es.jamstec.go.jp/

- System: 640 nodes, 40 TFLOP/s
  - 10 TB memory
  - optical 640x640 crossbar
  - 50m x 20m without peripherals
- Node: 8 CPUs, 64 GFLOP/s
  - 16 GB, SMP
  - ext. b/w: 2x16 GB/s

- Single-stage crossbar 640x640 (!)

Virtual Earth
- Climate change (global warming)
- El Niño, hurricanes, droughts
- Air pollution (acid rain, ozone hole)
- Diastrophism (earthquake, volcanism)

Installation: 2002
http://www.es.jamstec.go.jp/

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?
Example from SC 2001

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


Explicit C154N6 SEAM vs T170 PSTSWM, 16 Level, NCAR

Major Parallel Programming Models

- OpenMP (standardized since 1997)
  - Shared Memory Directives
    - to define the work decomposition
    - no data decomposition
    - synchronization is implicit (can be also user-defined)
    - mainly loops can be parallelized
    - compiler translates OpenMP directives into thread-handling
    - All data is shared / parallel execution threads on the same memory

- MPI (Message Passing Interface) (standardized since 1994)
  - User specifies how work & data is distributed
  - User specifies how and when communication has to be done
  - by calling MPI communication library-routines
  - compiler generates normal sequential code (running in each process)
  - typically domain decomposition with communication across domain boundaries
  - Each process has its private variables / Data exchange with messages
Shared Memory Directives – OpenMP, I.

Real :: A(n,m), B(n,m)  

!$OMP PARALLEL DO  
do j = 2, m-1  
do i = 2, n-1  
B(i,j) = ... A(i,j)  
  ... A(i-1,j) ... A(i+1,j)  
  ... A(i,j-1) ... A(i,j+1)  
end do  
end do  
!$OMP END PARALLEL DO

Data definition
Loop over y-dimension
Vectorizable loop over x-dimension
Calculate B, using upper and lower, left and right value of A

Shared Memory Directives – OpenMP, II.

Single Thread  
Parallel Region  
Single Thread  
Parallel Region  
Single Thread  

ISOMP PARALLEL  
ISOMP END PARALLEL

Master Thread  
Team of Threads  
Master Thread  
Team of Threads  
Master Thread

Message Passing Program Paradigm – MPI, I.

- Each processor in a message passing program runs a sub-program
  - written in a conventional sequential language, e.g., C or Fortran,
  - typically the same on each processor (SPMD)
- All work and data distribution is based on value of myrank
  - returned by special library routine
- Communication via special send & receive routines (message passing)

Additional Halo Cells – MPI, II.

Halo
(Shadow, Ghost cells)

User defined communication
Message Passing – MPI, III.

Call MPI_Comm_size(MPI_COMM_WORLD, size, ierr)
Call MPI_Comm_rank(MPI_COMM_WORLD, myrank, ierr)
m1 = (m+size-1)/size; ja=1+m1*myrank; je=max(m1*(myrank+1), m)
jax=ja-1; jex=je+1 // extended boundary with halo

Real :: A(n, jax:jex), B(n, jax:jex)
do j = max(2,ja), min(m-1,je)
do i = 2, n-1
B(i,j) = ... A(i,j) ... A(i-1,j) ... A(i+1,j) 
... A(i,j-1) ... A(i,j+1) ... A(i-1,j) ... A(i+1,j) 
end do
end do

Call MPI_Send(.......) ! - sending the boundary data to the neighbors
Call MPI_Recv(.......) ! - receiving from the neighbors,
! storing into the halo cells

Limitations of the Major Programming Models

• MPI
  – standardized distributed memory parallelism with message passing
  – process-based
  – the application processes are calling MPI library-routines
  – compiler generates normal sequential code

Limitations:
  • the amount of your hours available for MPI programming
  • can be used on any platform, but communication overhead on shared memory systems

• OpenMP
  – standardized shared memory parallelism
  – thread-based
  – compiler translates OpenMP directives into thread-handling

Limitations:
  • only for shared memory and ccNUMA systems
  • mainly for loop parallelization via OpenMP-directives
  • only for medium number of processors
  • explicit domain decomposition also via rank of the threads
Parallel Programming Models on Hybrid Platforms

Comparison I.

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

Comparison II.

- No overlap of Comm. + Comp.: MPI only outside of parallel regions of the numerical application code
- Overlapping Comm. + Comp.: MPI communication by one or a few threads while other threads are computing

Comparison III.

- Master only: MPI only outside of parallel regions
- Multiple/only: MPI only on master-thread
- Funneled & Reserved: reserved thread for communication
- Funneled with Full Load: balancing

MPI rules with OpenMP / Automatic SMP-parallelization (2)

- Special MPI-2 Init for multi-threaded MPI processes:
  ```c
  int MPI_Init_thread(int * argc, char *((**argv)[ ]), int required, int* provided)
  MPI_INIT_THREAD(REQUIRED, PROVIDED, IERROR)
  ```

- REQUIRED values (increasing order):
  - MPI_THREAD_SINGLE: Only one thread will execute
  - THREAD_MASTERONLY: (virtual value, not part of the standard)
    MPI processes may be multi-threaded, 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!

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

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

… the barrier is necessary – example with MPI_Recv

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

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]
- 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

→ no silver bullet
  - each parallelization scheme has its problems

The Topology Problem with Pure MPI

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

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

- Pure MPI
  - Vertical and horizontal messages
  - Intra-node 6-8 x 1MB: 2.0 ms
  - Inter-node 8 x 1MB: 9.6 ms
  - Timing: Hitachi SR8000, MPI_Sendrecv
  - 8 nodes, each node with 8 CPUs
  - Pure MPI: \[ \sum = 11.6 \text{ ms} \]

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

Programming Models on Hybrid Platforms: Hybrid Masteronly

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

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

Experiment: Orthogonal parallel communication

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

- Only half of the transferred bytes
- Less latencies due to 8x longer messages

→ 1.6x slower than with pure MPI, although

- Only half of the transferred bytes
- Less latencies due to 8x longer messages

**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 is faster
- MPI+OpenMP (masteronly) is faster

Ratio on several platforms

IBM SP and SR 8000
Masteronly: MPI cannot saturate inter-node bandwidth

Pure MPI
is faster

Hybrid
is faster

IBM SP
8x16 CPUs,
1 CPU Masteronly

SGI O3000
16x4 CPUs,
1 CPU Masteronly

Hitachi SR8000
8x8 CPUs,
1 CPU Masteronly

Pure MPI,
horizontal + vertical

Cray X1
8x4 MSPs,
1 MSP Masteronly

NEC SX6
4x8 CPUs,
1 CPU Masteronly

Pure MPI
is faster

Hybrid
is faster

Cray X1 and NEC SX are well prepared for hybrid masteronly programming

IBM SP and SR 8000 Masteronly: MPI cannot saturate inter-node bandwidth

Pure MPI
is faster

Hybrid
is faster

Cray X1 and NEC SX are well prepared for hybrid masteronly programming

Cray X1 and SGI results are preliminary

Pure MPI versus Hybrid-masteronly

Data transfer

Pure MPI:

- Typically in message passing epochs
- inter-node network saturated by a few processes per node
- Best case: only one additional cutting plane in each dimension (e.g., 8-way SMP) compared to hybrid MPI+OpenMP
- only doubling the total amount of transferred bytes

Hybrid-masteronly:

- other threads are sleeping while master thread calls MPI routines
- node-to-node communication time therefore weighted by number of processors/node !!!
- node-to-node bandwidth = 1 GB/s is reduced to 125 MB/s (on 8 CPUs/node)
- latency = 20 µs explodes to 160 µs
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

2nd Experiment: Multiple inter-node communication paths

- Multiple vertical communication paths, e.g.,
  - 3 of 8 CPUs in each node
  - stride 2

Following benchmark results: with one MPI process on each CPU

- MPI+OpenMP: only vertical
- pure MPI: vertical AND horizontal messages
- pure MPI: intra- + inter-node (= vert. + horizontal)
Multiple inter-node communication paths: Hitachi SR8000

Inter-node bandwidth per SMP node, accumulated over its CPUs, *)

To spend more than 3 CPUs/node for communication makes no sense

Intra-node messages do not count for bandwidth

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

Multiple inter-node communication paths: Hitachi SR 8000

Hybrid communication time / pure MPI communication time on Hitachi SR 8000

Hybrid is faster than pure MPI if ≥ 2 CPUs/node are used for intra-node communication in hybrid programming model

Multiple inter-node communication paths: **IBM SP**

Inter-node bandwidth per SMP node, accumulated over its CPUs, *)
on IBM at NERSC (16 Power3+ CPUs/node)

- 8x16 CPUs, Hybrid Multiple, 12/16 CPUs Stride 1
- 8x16 CPUs, Hybrid Multiple, 6/16 CPUs Stride 1
- 8x16 CPUs, Hybrid Multiple, 4/16 CPUs Stride 1
- 8x16 CPUs, Hybrid Multiple, 3/16 CPUs Stride 1
- 8x16 CPUs, Hybrid Multiple, 2/16 CPUs Stride 1
- 8x16 CPUs, Hybrid Multiple, 2/16 CPUs Stride 4
- 8x16 CPUs, Pure MPI, horizontal + vertical
- 8x16 CPUs, Hybrid Masteronly, MPI: 1 of 16CPUs

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

More than 4 CPUs per node needed to achieve full inter-node bandwidth

With 3 CPUs similar to pure MPI

The second CPU doubles the accumulated bandwidth

---

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

Inter-node bandwidth per SMP node, accumulated over its CPUs, *)
on NEC SX6 (with MPI_Aloc_mem)

- 4x8 CPUs, Hybrid Multiple, 8/8 CPUs Stride 1
- 4x8 CPUs, Hybrid Multiple, 6/8 CPUs Stride 1
- 4x8 CPUs, Hybrid Multiple, 4/8 CPUs Stride 1
- 4x8 CPUs, Hybrid Multiple, 2/8 CPUs Stride 1
- 4x8 CPUs, Hybrid Multiple, 2/8 CPUs Stride 4
- 4x8 CPUs, Hybrid Masteronly, MPI: 1 of 8 CPUs
- 4x8 CPUs, Hybrid Masterlet, horizontal + vertical

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

Inverse: More CPUs = less bandwidth

Intra-node messages do not count for bandwidth

Measurements: Thanks to Holger Berger, NEC.
Multiple inter-node communication paths:
Cray X1, used with 4 MSPs/node (preliminary results)

Inter-node bandwidth per SMP node, accumulated over its CPUs, *) on Cray X1, 4 MSPs / node (1 MSP = 4 CPUs)

1 MSP achieves already 80% of full inter-node bandwidth

Intra-node messages do not count for bandwidth

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

Measurements:
Thanks to Monika Wierse and Wilfried Oed, CRAY.

Multiple inter-node communication paths:
Cray X1, used with 4 MSPs/node, shmem put (instead MPI)

Inter-node bandwidth per SMP node, accumulated over its CPUs, *) on Cray X1, 4 MSPs / node (1 MSP = 4 CPUs), shmem put

1 MSP achieves already 75% of full inter-node bandwidth

Intra-node messages do not count for bandwidth

Highest parallel bandwidth: 12.0 GF/s

Measurements:
Thanks to Monika Wierse and Wilfried Oed, CRAY.
Comparison

Inter-node bandwidth per SMP node, accumulated over its CPUs *)

accumulated message size from node to node

Cray X1, smem_put, 4 MSPs/node
NEC SX-6, MPI with global memory, 8 CPUs/node
Hitachi, 8 CPUs/node
IBM, 16 CPUs/node

Cray X1 results are preliminary

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

Comparison (as percentage of maximal bandwidth and #CPUs)

Nearly full bandwidth
• with 1 MSP on Cray
• with 1 CPU on NEC

50 % and less on the other platforms

Nearly all platforms:
>80% bandwidth with >25% of CPUs/node

Cray X1 results are preliminary
Comparing inter-node bandwidth with peak CPU performance

<table>
<thead>
<tr>
<th>System</th>
<th>Master-only bw</th>
<th>pure MPI, inter-node [GB/s]</th>
<th>Master-only bw</th>
<th>pure MPI, intra-node [GB/s]</th>
<th>memo-ry band-width [GB/s]</th>
<th>Peak perf-omance GFlop/s</th>
<th>max. inter-node bw / peak perf. B/Flop</th>
<th>nodes*CPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cray X1 shmem_put</td>
<td>9.27</td>
<td>12.34</td>
<td>75%</td>
<td>33.0</td>
<td>136</td>
<td>51.2</td>
<td>0.241</td>
<td>8 * 4 MSPs</td>
</tr>
<tr>
<td>Cray X1, MPI</td>
<td>4.52</td>
<td>5.52</td>
<td>82%</td>
<td>51.2</td>
<td>256</td>
<td>64</td>
<td>0.118</td>
<td>8 * 4 MSPs</td>
</tr>
<tr>
<td>NEC SX-6 global memory</td>
<td>7.56</td>
<td>4.98</td>
<td>100%</td>
<td>78.7</td>
<td>256</td>
<td>64</td>
<td>0.039</td>
<td>4 * 8 CPUs</td>
</tr>
<tr>
<td>NEC SX-5Be local memory</td>
<td>2.27</td>
<td>2.50 (a)</td>
<td>91%</td>
<td>25.1</td>
<td>512</td>
<td>64</td>
<td>0.039</td>
<td>2 * 16 CPUs</td>
</tr>
<tr>
<td>Hitachi SR8000</td>
<td>0.45</td>
<td>0.91</td>
<td>49%</td>
<td>5.0</td>
<td>32 store</td>
<td>8</td>
<td>0.114</td>
<td>8 * 8 CPUs</td>
</tr>
<tr>
<td>IBM SP Power3+</td>
<td>0.16</td>
<td>0.57 (a)</td>
<td>28%</td>
<td>2.0</td>
<td>16</td>
<td>24</td>
<td>0.023</td>
<td>8 * 16 CPUs</td>
</tr>
<tr>
<td>SGI Origin 3000</td>
<td>0.10</td>
<td>0.30 (a)</td>
<td>33%</td>
<td>0.39</td>
<td>3.2</td>
<td>4.8</td>
<td>0.063</td>
<td>16 * 4 CPUs</td>
</tr>
<tr>
<td>SUN-fire (prelim.)</td>
<td>0.15</td>
<td>0.85</td>
<td>18%</td>
<td>1.68</td>
<td></td>
<td></td>
<td></td>
<td>4 * 24 CPUs</td>
</tr>
</tbody>
</table>

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

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

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]
- 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

→ no silver bullet
  - each parallelization scheme has its problems

Parallel Programming Models on Hybrid Platforms

Comparison I. (2 experiments)
- pure MPI
  - one MPI process on each CPU
- hybrid MPI+OpenMP
  - MPI: inter-node communication
  - OpenMP: inside of each SMP node
- OpenMP only distributed virtual shared memory

Comparison II. (theory + experiment)
- No overlap of Comm. + Comp.
  - MPI only outside of parallel regions of the numerical application code
- Overlapping Comm. + Comp.
  - MPI communication by one or a few threads while other threads are computing
- Masteronly
  - MPI only outside of parallel regions
  - Multiple/only
    - appl. threads
    - inside of MPI
- Funneled
  - MPI only on master-thread
  - Multiple & Reserved
    - reserved thread for communication
  - Funneled with Full Load Balancing
  - Multiple &
    - Reserved threads for communication
  - Full Load Balancing
    - Multiple with Full Load Balancing

Comparison III.
- Different strategies to simplify the load balancing

Overlapping communication and computation

- the load balancing problem:
  - some threads communicate, others not
  - balance work on both types of threads
- strategies:
  - 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 & Reserved
reserved thread for comm.

Multiple & Reserved
reserved threads for comm.

→ very hard to do !!!

Overlapping computation & communication (cont’d)

Funneled & reserved or Multiple & reserved:
- reserved tasks on threads:
  - master thread or some threads: communication
  - all other threads: computation
- cons:
  - bad load balance, if
    \[
    \frac{T_{\text{communication}}}{n_{\text{communication_threads}}} \neq \frac{T_{\text{computation}}}{n_{\text{computation_threads}}}
    \]
- pros:
  - more easy programming scheme than with full load balancing
  - chance for good performance!
Performance ratio (theory)

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

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

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

\[ n = \text{# threads per SMP node}, \quad m = \text{# reserved threads for MPI communication} \]

Good chance of funneled & reserved:
\[ \varepsilon_{\text{max}} = 1 + m(1 - \frac{1}{n}) \]

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

Experiment: Matrix-vector-multiply (MVM)

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

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


Parallel Programming Models on Hybrid Platforms

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

Comparison II.
Masteronly
MPI only outside of parallel regions
Multiple/only
• appl. threads
• inside of MPI

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

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

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

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/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., TreadMarks
Case Studies

- NAS Parallel Benchmarks EP, FT, and CG:
  - Message passing and sequential version
- Automatically generate OpenMP directives for sequential code using CAPO (www.nas.nasa.gov/Groups/Tools/CAPO)
- Omni Compiler
- Compare speedup of:
  - Message passing vs. OpenMP/DSM
  - OpenMP/DSM vs. OpenMP/SMP
- Hardware platforms:
  - DSM Test Environment
  - Use only one CPU per node
  - SMP 16-way NEC AzusA
- Case study was conducted with Gabrielle Jost from NASA/Ames and Matthias Hess, Matthias Mueller from HLRS

Slides courtesy of Gabriele Jost, NASA/AMES, and Matthias Müller, HLRS

The EP Benchmark

- Embarrassing Parallel:
  - Generation of random numbers
  - Loop iterations parallel
  - Global sum reduction at the end
- Automatic Parallelization without user interaction
- MPI implementation:
  - Global sum built via MPI_ALLREDUCE
  - Low communication overhead (< 1%)
- OpenMP/DSM:
  - OMP PARALLEL
  - OMP DO REDUCTION

Linear speedup for MPI and OpenMP/DSM. No surprises.

Slides courtesy of Gabriele Jost, NASA/AMES, and Matthias Müller, HLRS
CG Benchmark Results (1)

- Conjugate Gradient method to solve an eigenvalue problem
  - Stresses irregular data access
  - Major loops:
    - Sparse Matrix-Vector-Multiply
    - Dot-Product
    - AXPY Operations
  - Same major loops in MPI and OpenMP implementation
  - Automatic parallelization without user interaction
- Class A:
  - Problem size: na=14000, nz=11
  - OpenMP/DSM efficiency about 75% of that of MPI
- Class S:
  - Problem size: na=1400, nz=7
  - MPI about 20% communication.
  - No speedup for OpenMP/DSM due to:
    - Large Communication to Computation Ratio
    - Inefficiencies in the Omni Compiler

CG Benchmark Results (2)

- CG Class S

Slides courtesy of Gabriele Jost, NASA/AMES, and Matthias Müller, HLRS
FT Benchmark Results (1)

- Kernel of spectral method based on 3D Fast Fourier Transform (FFT)
  - 3D FFT achieved by a 1D FFT in x, y, and z direction
- MPI Parallelization:
  - Transpose of data for FFT in z-dimension
  - 15% in communication
- OpenMP Parallelization:
  - OpenMP parallelization required some user interaction
  - Privatization of certain arrays via the CAPO user interface
  - OMP DO PARALLEL
  - Order of loops changes for z-dimension

- OpenMP/DSM efficiency about 70% of MPI
  - Extra communication introduced by DSM system (false page sharing?)
  - Remote data access required for FFT in z-dimension

OpenMP/DSM: Conclusions:

- Rapid development of parallel code running across a cluster of PCs was possible
- OpenMP/DSM delivered acceptable speedup if the communication/computation ratio is not too high:
  - OpenMP/DSM showed between 70% and 100% efficiency compared to MPI for benchmarks of Class A
- Problems encountered:
  - High memory requirements for management of virtual shared memory (> 2GB)
  - Potential scalability problems
- Need for profiling tools
Outline

- Motivation [slides 3–7]
- Major parallel programming models [8–14]
- Programming models on hybrid systems [15–56]
  - Overview [15]
  - Technical aspects with thread-safe MPI [16–18]
  - Mismatch problems with pure MPI and hybrid MPI+OpenMP [19–46]
    - Topology problem [20]
    - Unnecessary intra-node comm. [21]
    - Inter-node bandwidth problem [22–33]
      - Comparison I: Two experiments
    - Sleeping threads and saturation problem [39]
    - Additional OpenMP overhead [40]
    - Overlapping comm. and comp. [41–47]
      - Comparison II: Theory + experiment
  - Pure OpenMP [48–56]
    - Comparison III
- No silver bullet / optimization chances / other concepts [58–62]
- Acknowledgments & Conclusions [63–64]

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 master-only (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

Other Concepts

- Distributed memory programming (DMP) language extensions
  - Co-array Fortran
  - UPC (Unified Parallel C)
    Idea: direct access to remote data via additional [rank] index

- Multi level parallelism (MLP)
  - combining OpenMP (inside of the processes)
  - with Sys V shared memory (data access between processes)
  - only on ccNUMA

No standards!
Only on a few platforms!
DMP Language Extensions

- Programmable access to the memory of the other processes
- Language bindings:
  - Co-array Fortran
  - UPC (Unified Parallel C)
- Special additional array index to explicitly address the process
- Examples (Co-array Fortran):
  
  ```fortran
  integer a[*], b[*] ! Replicate a and b on all processes
  
  dimension (n,n) :: u[*] ! Allocates the n x n array u
  
  p = THIS_IMAGE(u,1) ! first co-subscript of local process
  q = THIS_IMAGE(u,1) ! second co-subscript of local process
  
  u(1:n,1)[p+1,q] = u(1:n,n)[p,q] ! Copy right boundary u(1,) on process [p,] to right neighbor [p+1,] into left boundary u(n,)
  ```

Multi Level Parallelism (MLP)

- program
- processes
- multiple threads inside of each process (OpenMP)
- data associated with each process
- but shared (ccNUMA) access to other processes' data

Cheap load balancing
- by changing the number of threads per process
- before starting a new parallel region
Acknowledgements

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

  and
  - Thomas Ludwig, University of Heidelberg

Conclusions

- Only a few platforms (e.g. Cray X1 in MSP mode, or NEC SX-6)
  - are well designed hybrid MPI+OpenMP masteronly scheme
- Other platforms
  - masteronly style cannot saturate inter-node bandwidth
  - optimization chances should be used
- Pure MPI and hybrid masteronly:
  - idling CPUs (while one or some are communicating)
- DSM systems (pure OpenMP):
  - may help for some applications
- Optimal performance:
  - overlapping of communication & computation
    - extreme programming effort
  - optimal throughput
    - reuse of idling CPUs by other applications

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