Parallel Programming Models on Hybrid Systems

Rolf Rabenseifner
rabenseifner@hlrs.de

University of Stuttgart, High Performance Computing Center Stuttgart (HLRS)
www.hlrs.de

Tutorial at the HLRS Results and Review Workshop
“High Performance Computing in Science and Engineering”
Oct. 6–7, 2003, Stuttgart, Germany

Outline

• Motivation [slides 3–7]
• Programming models on hybrid systems [8–50]
  – Overview [8]
  – Technical aspects with thread-safe MPI [9–11]
  – Mismatch problems with pure MPI and hybrid MPI+OpenMP [12–46]
    • Topology problem [13]
    • Unnecessary intra-node comm. [14]
    • Inter-node bandwidth problem [16–34]
    – Comparison I: Two experiments
    • Sleeping threads and saturation problem [35]
    • Additional OpenMP overhead [36]
    – Comparison II: Theory + experiment
  – Pure OpenMP [46–50]
    – Comparison III
• No silver bullet / optimization chances / other concepts [54–59]
• Acknowledgments & Conclusions [60–61]
Motivation

- HPC systems
  - often clusters of SMP nodes
  - i.e., hybrid architectures
- Using the communication bandwidth of the hardware
- Minimizing synchronization = idle time
- Appropriate parallel programming models / Pros & Cons

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
- CPU: Vector
  8 GFLOP/s, 500 MHz
  Single-Chip, 0.15 µs
  32 GB/s memory b/w
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?

```
some_serial_code
#pragma omp parallel for
for (j=...; ...; j++)
block_to_be_parallelized
again_some_serial_code
```

Master thread, other threads

OpenMP (shared data)

MPI local data in each process

Explicit Message Passing by calling MPI_Send & MPI_Recv

Node Interconnect

OpenMP inside of the SMP nodes

MPI between the nodes via node interconnect

But results may surprise!

- Example code - HYDRA
- Domain-decomposed hydrodynamics
  - (almost) independent mesh domains with ghost cells on boundaries
  - ghost cells communicate boundary information ~40-50 times per cycle
- Parallelism model: single level
  - MPI divides domains among compute nodes
  - OpenMP further subdivides domains among processors
  - domain size set for cache efficiency
    - minimizes memory usage, maximizes efficiency
    - scales to very large problem sizes (>10^7 zones, >10^3 domains)
- Results:
  - MPI (256 proc.) ~20% faster than MPI / OpenMP (64 nodes x 4 proc./node)
  - domain-domain communication not threaded,
    i.e., MPI communication is done only by main thread
    - accounts for ~10% speed difference, remainder in thread overhead
### Example from SC 2001

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

#### Figures: Richard D. Loft, Stephen J. Thomas, John M. Dennis:
Terascale Spectral Element Dynamical Core for Atmospheric General Circulation Models.
Fig. 9 and 10.

#### Explicit C154N6 16 Level SEAM:
NPACI Results with 7 or 8 processes or threads per node

<table>
<thead>
<tr>
<th>Processors</th>
<th>Integration rate [Years per day]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>200</td>
<td>5</td>
</tr>
<tr>
<td>400</td>
<td>10</td>
</tr>
<tr>
<td>600</td>
<td>15</td>
</tr>
<tr>
<td>800</td>
<td>20</td>
</tr>
<tr>
<td>1000</td>
<td>35</td>
</tr>
</tbody>
</table>

#### Explicit/Semi Implicit C154N6 SEAM vs T170 PSTSWM, 16 Level, NCAR

<table>
<thead>
<tr>
<th>Processors</th>
<th>Integration rate [Years per day]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>5</td>
</tr>
<tr>
<td>200</td>
<td>10</td>
</tr>
<tr>
<td>300</td>
<td>15</td>
</tr>
<tr>
<td>400</td>
<td>20</td>
</tr>
<tr>
<td>500</td>
<td>25</td>
</tr>
<tr>
<td>600</td>
<td>30</td>
</tr>
</tbody>
</table>

### Parallel Programming Models on Hybrid Platforms

- **pure MPI**
  - one MPI process on each CPU
  - No overlap of Comm. + Comp.
  - MPI only outside of parallel regions of the numerical application code

- **hybrid MPI+OpenMP**
  - MPI: inter-node communication
  - OpenMP: inside of each SMP node
  - Overlapping Comm. + Comp.
  - MPI communication by one or a few threads while other threads are computing

- **OpenMP only**
  - distributed virtual shared memory
  - Multiple/only
  - more than one thread may communicate

- **Masteronly**
  - MPI only outside of parallel regions
  - Funneled & Reserved
  - reserved thread for communication
  - Funneled with Full Load Balancing

- **Multiple/only**
  - appl. threads inside of MPI
  - Funneled
  - MPI only on master-thread
  - Funneled with Full Load Balancing

- **Multiple & Reserved**
  - reserved threads for communication
  - Multiple with Full Load Balancing
  - 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
  - MPI_THREAD_MASTERONLY: MPI processes may be multi-threaded, but only master thread will make MPI-calls
    (virtual value, not part of the standard) AND only while other threads are sleeping
  - MPI_THREAD_FUNNELED: Only master thread will make MPI-calls
  - MPI_THREAD_SERIALIZE: Multiple threads may make MPI-calls, but only one at a time
  - MPI_THREAD_MULTIPLE: Multiple threads may call MPI, with no restrictions

- Returned PROVIDED may be less than REQUIRED by the application

---

Calling MPI inside of OMP MASTER

- Inside of a parallel region, with "OMP MASTER"
- Requires MPI_THREAD_FUNNELED, i.e., only master thread will make MPI-calls

- Caution: There isn’t any synchronization with "OMP MASTER"!
  Therefore, "OMP BARRIER" normally necessary to guarantee, that data or buffer space from/for other threads is available before/after the MPI call!

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

- But this implies that all other threads are sleeping!
- The additional barrier implies also the necessary cache flush!
... the barrier is necessary – example with MPI_Recv

```c
!$OMP PARALLEL
!$OMP DO
do i=1,1000
    a(i) = buf(i)
end do
!$OMP END DO NOWAIT
!$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 startup / join
  - Cache flush (data source thread – communicating thread – sync. → flush)
- Overlapping communication and computation [with hybrid MPI+OpenMP]
  - an application problem → separation of local or halo-based code
  - a programming problem → thread-ranks-based vs. OpenMP work-sharing
  - a load balancing problem, if only some threads communicate / compute

→ 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

Alternative:
- Hybrid MPI+OpenMP
- No intra-node messages
- Longer inter-node messages
- Really faster ???????

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

pure MPI

Masteronly

Hybrid Parallel Programming
Rolf Rabenseifner
High Perf. Comp. Center, Univ. Stuttgart

pure MPI: only vertical
MPI+OpenMP: vertical AND horizontal messages

message size := aggregated message size of pure MPI

8+8MB: 2.0 ms
8+8+1MB: 9.6 ms
intra-node 8+8+1MB: 2.0 ms
inter-node 8+8+1MB: 9.6 ms
hybrid: 19.2 ms
pure MPI: 11.6 ms

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

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

![Graph showing the ratio of hybrid to pure MPI transfer times](image)

**Ratio on several platforms**

- IBM SP and SR 8000: Pure MPI is faster
- SGI O3000: Hybrid is faster
- Hitachi SR8000: Hybrid is faster
- Cray X1: Hybrid is faster
- NEC SX6: Pure MPI is faster

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

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

pure MPI: intra- + inter-node messages

pure MPI: intra- + inter-node messages

hybrid: 3*8 + 8/3MB

intra-node 8*8*1MB

inter-node 8*8*1MB

vertical AND horizontal messages

MPI+OpenMP:
only vertical
Multiple inter-node communication paths: Hitachi SR8000

Inter-node bandwidth per SMP node, accumulated over its CPUs, on Hitachi SR8K

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

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)

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

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

Measurements: Thanks to Gerhard Wellein, RRZE, and Horst Simon, NERSC.

---

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

**Inter-node bandwidth per SMP node, accumulated over its CPUs,** *)
on NEC SX-6 (with MPI_Alloc_mem)

- More CPUs = less 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 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

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

Highest parallel bandwidth: 12.0 GF/s

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

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

Intra-node messages do not count for bandwidth

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

© Rolf Rabenseifner: Parallel Programming Models on Hybrid Systems.
Comparison

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

* 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
Comparison (only 960 kB aggregated message size)

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

accumulated bandwidth as percentage
of the peak bandwidth

communicating CPUs per SMP node
as percentage
of the total number of CPUs per SMP node

Similar behavior on Cray X1 and NEC SX-6

Cray X1 MSP shmem_put / 960 kB
Cray X1 MSP / 960 kB
NEC SX6 glmem / 960 kB
Hitachi SR8000 / 960 kB
IBM SP/Power3+ / 960 kB

Myrinet Cluster

Myrinet Cluster

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

- 128x2 CPUs, Hybrid Multiple, 2/2 CPUs Stride 1
- 128x2 CPUs, Hybrid Masteronly, MPI: 1 of 2 CPUs
- 128x2 CPUs, Pure MPI, horizontal + vertical
Hybrid Parallel Programming

Myrinet Cluster (only 64 nodes)

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

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

![Graph showing bandwidth](image)

Hybrid Programming on Cray X1: MSP based usage

- pure MPI or hybrid masteronly MPI+OpenMP
  -> same communication time
- 1 MSP already achieves 80% of maximum bandwidth (contiguous data)
  - Are CPU-intensive MPI routines (Reduce, strided data) efficient & multi-threaded?
- Hybrid programming -> 4 layers of parallelism
  - MPI between nodes (e.g. domain decomposition)
  - OpenMP between MSPs (e.g. outer loops)
  - Automatic parallelization (e.g. inner loops)
  - Vectorization (e.g. most inner loops)
-> risk of Amdahl's law on each level!
- Hybrid & overlapping communication and computation
  - horrible programming interface (but standardized)
  - but chance to use sleeping MSPs while master MSP communicates
Hybrid Programming on Cray X1: SSP based

- Communication is hardware-bound to SSP
  - 1 SSP can get only 1/4 of 1 MSP’s inter-node bandwidth
  - with shmem put:
    all SSPs of a node can together achieve full inter-node bandwidth
    (12.3 GB/s of 12.8 GB/s hardware specification)
- Hybrid MPI+OpenMP, masteronly style
  - optimized MPI library needed with same bandwidth as on 1 or 4 MSP
    - e.g., internally thread-parallel
- Multiple communicating user-threads are not supported
- pure MPI
  - efficient MPI implementation under development

Comparing inter-node bandwidth with CPU performance

<table>
<thead>
<tr>
<th></th>
<th>Master-only, inter-node</th>
<th>pure MPI, inter-node</th>
<th>Master-only bw / max. intra-node bw</th>
<th>pure MPI, intra-node bw</th>
<th>memo-ry bandwidth [GB/s]</th>
<th>Peak &amp; Linpack perfor- mance [Gflop/s]</th>
<th>max.inter-node bw / peak &amp; Linpack 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>45.03</td>
<td>0.241 0.274</td>
</tr>
<tr>
<td>Cray X1, MPI</td>
<td>4.52</td>
<td>5.52</td>
<td>82 %</td>
<td>19.5</td>
<td>136</td>
<td>51.2</td>
<td>45.03</td>
<td>0.108 0.123</td>
</tr>
<tr>
<td>NEC SX-6</td>
<td>7.56</td>
<td>4.98</td>
<td>100 %</td>
<td>78.7</td>
<td>256</td>
<td>64</td>
<td>61.83</td>
<td>0.118 0.122</td>
</tr>
<tr>
<td>NEC SX-5Be</td>
<td>2.27</td>
<td>2.50</td>
<td>91 %</td>
<td>35.1</td>
<td>512</td>
<td>64</td>
<td>60.50</td>
<td>0.039 0.041</td>
</tr>
<tr>
<td>Hitachi SR8000</td>
<td>0.45</td>
<td>0.91</td>
<td>49 %</td>
<td>5.0</td>
<td>8</td>
<td>0.114</td>
<td>0.133</td>
<td>8 * 8 CPUs</td>
</tr>
<tr>
<td>IBM SP Power3+</td>
<td>0.16</td>
<td>0.57</td>
<td>28 %</td>
<td>2.0</td>
<td>16</td>
<td>0.023</td>
<td>0.040</td>
<td>8 *16 CPUs</td>
</tr>
<tr>
<td>SGI O3000, 600MHz</td>
<td>0.43 *)</td>
<td>1.74 *)</td>
<td>25 %</td>
<td>1.73 *)</td>
<td>4.8</td>
<td>3.64</td>
<td>3.64</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>2.80</td>
<td>1.61</td>
<td>0.043</td>
<td>0.074</td>
</tr>
<tr>
<td>HELICS Dual-PC cluster</td>
<td>0.118 *)</td>
<td>0.119 *)</td>
<td>100 %</td>
<td>0.104 *)</td>
<td>2.80</td>
<td>1.61</td>
<td>0.043</td>
<td>0.074</td>
</tr>
</tbody>
</table>

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

All values: aggregated over one SMP node. *) mess. size: 16 MB *) 2 MB

© Rolf Rabenseifner: Parallel Programming Models on Hybrid Systems.
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

Overlapping Communication and Computation

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

- The application problem:
  - One must separate application into:
    - Code that can run before the halo data is received
    - Code that needs halo data
  → very hard to do !!!

- The thread-rank problem:
  - Comm. / comp. via thread-rank
  - Cannot use work-sharing directives
  → Loss of major OpenMP support

- The load balancing problem

```c
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++) {
        ....
    }
}
```
Hybrid Parallel Programming

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

Comparison II.
(theory + experiment)

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

Comparison III.

- 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

Overlapping communication and computation (cont’d)

- 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

  ➔ 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 \( T_{\text{communication}} \neq \frac{n_{\text{communication_threads}}}{T_{\text{computation}}} \)
  \( T_{\text{computation}} \neq \frac{n_{\text{computation_threads}}}{} \)
- pros:
  - more easy programming scheme than with full load balancing
  - chance for good performance!

Performance ratio (theory)

- \( \varepsilon = \left( \frac{T_{\text{hybrid, funneled\&reserved}}}{T_{\text{hybrid, masteronly}}} \right)^{-1} \)
- Good chance of funneled & reserved: \( \varepsilon_{\text{max}} = 1+m(1-1/n) \)
- Small risk of funneled & reserved: \( \varepsilon_{\text{min}} = 1-m/n \)

\( f_{\text{comm}} \)% performance ratio (\( \varepsilon \))

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

\( n = \# \text{ threads per SMP node}, \quad m = \# \text{ reserved threads for MPI communication} \)
Experiment: Matrix-vector-multiply (MVM)

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

Source: R. Rabenseifner, G. Wellein:
Communication and Optimization Aspects of Parallel Programming Models on Hybrid Architectures.

© Rolf Rabenseifner: Parallel Programming Models on Hybrid Systems.
Parallel Programming Models on Hybrid Platforms

Comparison I.
Pure MPI
- one MPI process on each CPU
- No overlap of Comm. + Comp.
- MPI only outside of parallel regions of the numerical application code

Comparison II.
Multiple/only appl. threads inside of MPI
- Funneled & Reserved reserved thread for communication
- Funneled with Full Load Balancing

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

Hybrid Parallel Programming Models

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 Gabrielle 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.

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

Number of Processes

Speedup

Omni
Intel
Guide

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]
- Programming models on hybrid systems [8–50]
  - Overview [8]
  - Technical aspects with thread-safe MPI [9–11]
  - Mismatch problems with pure MPI and hybrid MPI+OpenMP [12–46]
    - Topology problem [13]
    - Unnecessary intra-node comm. [14]
    - Inter-node bandwidth problem [16–34]
      - Comparison I: Two experiments
    - Sleeping threads and saturation problem [35]
    - Additional OpenMP overhead [36]
    - Overlapping comm. and comp. [37–44]
      - Comparison II: Theory + experiment
  - Pure OpenMP [45–50]
    - Comparison III
- No silver bullet / optimization chances / other concepts [54–59]
- Acknowledgments & Conclusions [60–61]
No silver bullet

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

Chances for optimization

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

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):
  ```
  integer a[*], b[*] ! Replicate a and b on all processes
  
  dimension (n,n) :: u[3,*] ! Allocates the n*n array u
  ! on each of the 3x* processes
  
  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
  - Reiner Vogelsang, SGI
  - Gabriele Jost, NASA
  - Dieter an Mey, RZ Aachen
  - Horst Simon, NERSC
  - Matthias Müller, HLRS
  - my colleges at HLRS
Conclusions

- Only a few platforms
  - e.g., Cray X1 in MSP mode, NEC SX-6, and Myrinet-cluster
  - 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
  - single threaded, vectorized, low-priority, small-medium memory needs
See also www.hlrs.de/people/rabenseifner -> list of publications