Hybrid Parallel Programming: Performance Problems and Chances on Cray X1, NEC SX-6 and Other Platforms

Rolf Rabenseifner
rabenseifner@hlrs.de

University of Stuttgart,
High Performance Computing Center Stuttgart (HLRS)

CUG SUMMIT 2003
May 12–16, Columbus, Ohio, USA

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

optimal usage of the hardware
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?

Figures: Richard D. Loft, Stephen J. Thomas, John M. Dennis:
Terascale Spectral Element Dynamical Core for Atmospheric General Circulation Models.
Fig. 9 and 10.
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&Reserved
    - reserved thread for communication

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 ???????
  (... 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

pure MPI: Σ = 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

Ratio on several platforms

- IBM SP and SR 8000
  - Masteronly: MPI cannot saturate inter-node bandwidth
  - Pure MPI 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 glmem 4x8 CPUs, 1 CPU Masteronly

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**

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

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
Multiple inter-node communication paths: Hitachi SR8000

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

![Graph showing bandwidth per node for different configurations of CPUs and strides.]

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.

© Rolf Rabenseifner: Hybrid Parallel Programming: Performance Problems and Chances. CUG SUMMIT 2003, May 12–16, Columbus, Ohio, USA. Page 8
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 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

The second CPU doubles the accumulated bandwidth

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

With 3 CPUs similar to pure MPI

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 SX6 (with MPI_Alloc_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, 3/8 CPUs Stride 1
- 4x8 CPUs, Hybrid Multiple, 2/8 CPUs Stride 4
- 4x8 CPUs, Pure MPI, 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.

© Rolf Rabenseifner: Hybrid Parallel Programming: Performance Problems and Chances.
CUG SUMMIT 2003, May 12–16, Columbus, Ohio, USA. Page 9
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

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

Highest parallel bandwidth: 12.0 GF/s

Intra-node messages do not count for bandwidth

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

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

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

accumulated message size from node to node

Cray X1 results are preliminary

Comparison (as percentage of maximal bandwidth and #CPUs)

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

NEC SX6 glmem / 7680 kB
Hitachi SR8000 / 7680 kB
IBM SP/Power3+ / 7680 kB

NEC SX6 glmem / 1920*4 kB
Hitachi SR8000 / 1920*4 kB
Hitachi SR8000 / 240*4 kB
IBM SP/Power3+ / 240*4 kB

NEC SX6 glmem / 240*4 kB
IBM SP/Power3+ / 240*4 kB

NEC SX6 glmem / 30*4 kB
IBM SP/Power3+ / 30*4 kB

accumulated message size from node to node

Cray X1 results are preliminary

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

*) Bandwidth per node: totally transferred bytes on the inter-node network / wall clock time / number of nodes
Comparison (only 960 kB aggregated message size)

- Inter-node bandwidth per SMP node, accumulated over its CPUs
- Communicating CPUs per SMP node as percentage of the total number of CPUs per SMP node
- Accumulated bandwidth as percentage of the peak bandwidth

Similar behavior on Cray X1 and NEC SX-6

Cray X1 results are preliminary

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
Hybrid Parallel Programming

Overlapping Communication and Computation

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

• the application problem:
  – one must separate application into:
    • code that can run before the halo data is received
    • code that needs halo data
  ➔ very hard to do !!!

• the thread-rank problem:
  – comm. / comp. via thread-rank
  – cannot use work-sharing directives
  ➔ loss of major OpenMP support

• the load balancing problem

If (my_thread_rank < 1) {
  MPI_Send/Recv....
} else {
  my_range = (high-low-1) / (num_threads-1) + 1;
  my_low = low + (my_thread_rank+1)*my_range;
  my_high=high+ (my_thread_rank+1)*my_range;
  my_high = max(high, my_high)
  for (i=my_low; i<my_high; i++) {
    ....
  }
}

Overlapping communication and computation (cont’d)

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

  Funneled & Reserved
  reserved thread
  for communi.

  Multiple & Reserved
  reserved threads
  for communi.

  ➔ a good chance !!! ... see next slide

  Funneled with Full Load Balancing

  Multiple with Full Load Balancing

  ➔ very hard to do !!!
Performance ratio (theory)

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

Good chance of funneled & reserved:

$\varepsilon_{\text{max}} = 1 + m(1 - \frac{1}{n})$

Small risk of funneled & reserved:

$\varepsilon_{\text{min}} = 1 - \frac{m}{n}$

$m = \# \text{reserved threads for MPI communication}$

$n = \# \text{threads per SMP node}$

- $f_{\text{comm}}$ is the communication time as a percentage.

- $f_{\text{comp}}$ is the computation time.

- $f_{\text{comp, overlap}}$ is the overlap time.

- $f_{\text{comp, non-overlap}}$ is the non-overlap time.

- $n = \# \text{threads per SMP node}$

- $m = \# \text{reserved threads for MPI communication}$

- $\varepsilon > 1$ funneled & reserved is faster.

- $\varepsilon < 1$ masteronly is faster.

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 peak CPU performance

<table>
<thead>
<tr>
<th>System</th>
<th>Master only, inter-node [GB/s]</th>
<th>pure MPI, inter-node [GB/s]</th>
<th>Master only / max. intra-node bw</th>
<th>Memo -ry intra-node bandwidth [GB/s]</th>
<th>Peak performance 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 preliminary results</td>
<td>9.27</td>
<td>12.34</td>
<td>75 %</td>
<td>33.0</td>
<td>136</td>
<td>51.2</td>
<td>8 * 4 MSPs</td>
</tr>
<tr>
<td>Cray X1, MPI preliminary results</td>
<td>4.52</td>
<td>5.52</td>
<td>82 %</td>
<td>19.5</td>
<td>136</td>
<td>51.2</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>4 * 8 CPUs</td>
</tr>
<tr>
<td>NEC SX-5Be local memory</td>
<td>2.27</td>
<td>2.50</td>
<td>91 %</td>
<td>35.1</td>
<td>512</td>
<td>64</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>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</td>
<td>28 %</td>
<td>2.0</td>
<td>24</td>
<td>0.023</td>
<td>8 *16 CPUs</td>
</tr>
<tr>
<td>SGI Origin 3000 preliminary results</td>
<td>0.10</td>
<td>0.30</td>
<td>3%</td>
<td>3.2</td>
<td>16</td>
<td>4.8</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>4 *24 CPUs</td>
</tr>
</tbody>
</table>

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

Conclusions

• Cray X1 with MSPs (1 node = 4 MSPs) and NEC SX-5/6:
  – well designed hybrid MPI+OpenMP masteronly scheme

• Cray X1 with SSPs (1 node = 16 SSPs)
  – hybrid programming: 1 SSP cannot saturate inter-node bandwidth

• Other platforms
  – masteronly style cannot saturate inter-node bandwidth

• Pure MPI and hybrid masteronly:
  – idling CPUs (while one is communicating)

• 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