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

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 [12–49]
    • Topology problem [15]
    • Unnecessary intra-node comm. [14]
    • Inter-node bandwidth problem [16–18]
    – Comparison I: Two experiments
    – Sleeping threads and saturation problem [36]
    • Additional OpenMP overhead [38]
    – Overlapping comm. and comp. [39–46]
    – Comparison II: Theory + experiment
  – Pure OpenMP [47–49]
    – Comparison III
• No silver bullet / optimization chances / other concepts [50–55]
• Acknowledgments & Conclusions [56–57]
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

- GPU: 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?

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?


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

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

Integration rate [Years per day]

Processors

0      200      400      600      800      1000

Integration rate [Years per day]

Processors

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

Parallel Programming Models on Hybrid Platforms

- 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

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

core

Multiple/only
- appl. threads • inside of MPI

Funneled: MPI only on master-thread

Multiple
- more than one thread may communicate

Funneled & Reserved reserved thread for communication

Funneled with Full Load Balancing

Multiple & Reserved reserved threads for communication

Multiple 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: 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
!$OMP BARRIER #pragma omp barrier
!$OMP MASTER #pragma omp master
call MPI_Xxx(...) MPI_Xxx(...);
!$OMP END MASTER #pragma omp barrier

!$OMP BARRIER #pragma omp barrier
!$OMP MASTER #pragma omp master
call MPI_Xxx(...) MPI_Xxx(...);
!$OMP END MASTER #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
#pragma parallel
{
#pragma for nowait
for (i=0; i<1000; i++)
a[i] = buf[i];
#pragma omp barrier
#pragma omp master
MPI_Recv(buf,...);
#pragma omp barrier
#pragma for nowait for (i=0; i<1000; i++)
c[i] = buf[i];
}
#pragma 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 ???????
  (... wait 2 slides)

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

pure MPI: \( \sum = 11.6 \text{ ms} \)
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

- Topology problem
- Unnecessary intra-node comm.
- Inter-node bandwidth problem
- Sleeping threads and saturation problem
- Additional OpenMP overhead

→ 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: \( T_{\text{hybrid}} / T_{\text{pure MPI}} > 1.6 \)
- OpenMP master thread cannot saturate the inter-node network bandwidth

Ratio on several platforms

IBM SP and SR 8000
Mastersonly: MPI cannot saturate inter-node bandwidth
Pure MPI is faster

Hybrid is faster

Cray X1 and NEC SX are well prepared for hybrid mastersonly 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
  - vertical AND horizontal messages
  - multiple vertical communication paths, e.g.,
    - 3 of 8 CPUs in each node
    - stride 2

- **pure MPI:**
  - intra-node 8×8×1MB
  - inter-node 8×8×1MB
  - hybrid: 3×8×8×3MB

Following benchmark results with one MPI process on each CPU
Multiple inter-node communication paths: IBM SP

Inter-node bandwidth per SMP node, accumulated over its CPUs, on IBM at Juelich (32 Power4+ CPUs/node, FederationSwitch with 4 adapters per node)

- More than 4 CPUs per node needed to achieve full inter-node bandwidth
- With 3-4 CPUs similar to pure MPI

Measurements: Thanks to Bern Mohr, ZAM, FZ Jülich

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

The second CPU doubles the accumulated bandwidth
But only if second process is located on CPU connected with 2nd adapter!

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

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

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

The second CPU doubles the accumulated bandwidth
But only if second process is located on CPU connected with 2nd adapter!
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)

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

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

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

- 8x4 MSPs, put, Hybrid Multiple, 4/4 MSPs Stride 1
- 8x4 MSPs, put, Hybrid Multiple, 3/4 MSPs Stride 1
- 8x4 MSPs, put, Hybrid Multiple, 2/4 MSPs Stride 2
- 8x4 MSPs, put, Pure MPI, horizontal + vertical
- 8x4 MSPs, put, Hybrid Masteronly, MPI: 1 of 4 MSPs

*) 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: Hitachi SR8000

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

- 8x8 CPUs, Hybrid Multiple, 8/8 CPUs Stride 1
- 8x8 CPUs, Hybrid Multiple, 6/8 CPUs Stride 1
- 8x8 CPUs, Hybrid Multiple, 4/8 CPUs Stride 1
- 8x8 CPUs, Hybrid Multiple, 3/8 CPUs Stride 1
- 8x8 CPUs, Hybrid Multiple, 2/8 CPUs Stride 1
- 8x8 CPUs, Pure MPI, horizontal + vertical
- 8x8 CPUs, Hybrid Masteronly, MPI: 1 of 8 CPUs

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

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

Intra-node messages do not count for bandwidth

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

Message size (used with pure MPI on each CPU)
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.

Comparison

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

Hybrid Parallel Programming  Rolf Rabenseifner
Slide 27 / 57  High Perf. Comp. Center, Univ. Stuttgart

Hybrid Parallel Programming  Rolf Rabenseifner
Slide 28 / 57  High Perf. Comp. Center, Univ. Stuttgart
Parallel Programming Models on Hybrid Systems

MPI+OpenMP and other models on clusters of SMP 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

Comparison (only 960 kB aggregated message size)

Similar behavior on Cray X1 and NEC SX-6

Cray X1 results are preliminary
Myrinet Cluster

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

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
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></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></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>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.241</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.274</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>19.5</td>
<td>136</td>
<td>51.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.108</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.123</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8 * 4 MSPs</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>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.118</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.122</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4 * 8 CPUs</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>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.039</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.041</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2 * 16 CPUs only with 8</td>
</tr>
<tr>
<td>Hitachi SR8000</td>
<td>0.45</td>
<td>0.91</td>
<td>49%</td>
<td>5.0</td>
<td>22 store/52 load</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.114</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.133</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>24</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.023</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.040</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.363</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.478</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>2.0</td>
<td>14.27</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.040</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4 * 24 CPUs</td>
</tr>
<tr>
<td>HELICS Dual-PC cluster with Myrinet</td>
<td>0.118</td>
<td>0.119</td>
<td>100%</td>
<td>0.104</td>
<td>2.80</td>
<td>1.61</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.074</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>128 * 2 CPUs</td>
</tr>
</tbody>
</table>

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**
Example with Sweep3d – analyzed with Kojak

- Expert: part of Kojak project at FZ Jülich,
  - Contact: Dr. Bernd Mohr
  - www.fz-juelich.de/zam/kojak/
- Sweep3d
  - ASCI Benchmark (MPI-Version)
  - A first, insufficient (straight-forward) hybrid “masteronly” MPI+OpenMP implementation
- Demo:
  - Icon starts D:\EigeneDateien\expert_Mohr\src\presenter.py
  - File ➔ Open ➔ D:\EigeneDateien\expert_Mohr\reports\sweep3d.eap.dat
  - Left mouse button: open more details
  - Right mouse double-click: choose this event class for details in next window

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++) {
        ....
    }
}
```
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
    \[ \frac{T_{\text{communication}}}{T_{\text{computation}}} \neq \frac{n_{\text{communication threads}}}{n_{\text{computation threads}}} \]
- pros:
  - more easy programming scheme than with full load balancing
  - chance for good performance!

Performance ratio (theory)

- \[ \varepsilon = \left( \frac{T_{\text{hybrid, funneled & reserved}}}{T_{\text{hybrid, master only}}} \right)^{-1} \]
  - Good chance of funneled & reserved:
    \[ \varepsilon_{\text{max}} = 1 + m(1 - 1/n) \]
  - Funneled & reserved is faster
    \[ \varepsilon > 1 \]
  - Small risk of funneled & reserved:
    \[ \varepsilon_{\text{min}} = 1 - m/n \]
  - Master only
    \[ \varepsilon < 1 \]
Experiment: Matrix-vector-multiply (MVM)

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


---

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

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

Comparison II.
- Master only: MPI only outside of parallel regions
- Multiple/only: appl. threads inside of MPI

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

Compilation and Optimization
- Library based communication (e.g., MPI)
  - clearly separated optimization of
    1. communication \rightarrow MPI library
    2. computation \rightarrow 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

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 [36]
    - Additional OpenMP overhead [38]
    - Overlapping comm. and comp. [39–46]
      - Comparison II: Theory + experiment
    - Pure OpenMP [47–49]
      - Comparison III
- No silver bullet / optimization chances / other concepts [50–56]
- Acknowledgments & Conclusions [56–57]
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 SysV 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):
  ```
  integer a[*], b[*] ! Replicate a and b on all processes
  dimension (n,n) :: u[*] ! Allocates the n*n array u
  ! on each of the 3* 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