Kommunikations- und Optimierungsaspekte paralleler Programmiermodelle auf hybriden HPC-Plattformen

Rolf Rabenseifner
rabenseifner@hlrs.de
Universität Stuttgart, Höchstleistungsrechenzentrum Stuttgart (HLRS)
www.hlrs.de

Gerhard Wellein
gerhard.wellein@rrze.uni-erlangen.de
Friedrich-Alexander-Universität Erlangen-Nürnberg, Regionales Rechenzentrum Erlangen (RRZE)

Slide 1
High Perf. Comp. Center, Univ. Stuttgart

Motivation

• HPC systems
  – often clusters of SMP nodes
  – i.e., hybrid architectures

• Using the communication bandwidth of the hardware = optimal usage of the hardware

• Minimizing synchronization = idle time

• Appropriate parallel programming models / Pros & Cons
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

**Comparison I.** (experiment)
- Masteronly
  - MPI only outside of parallel regions

**Comparison II.** (theory + experiment)
- Overlapping Comm. + Comp.
  - MPI communication by one or a few threads while other threads are computing
- Funneled & Reserved
  - reserved thread for communication
- Funneled with Full Load Balancing
- Multiple & Reserved
  - reserved threads for communication

**Comparison III.**
- Funneled
  - MPI only on master-thread
- Multiple
  - more than one thread may communicate

**Parallel Programming Models on Hybrid Platforms:**

**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
    - 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, ...)
- **Disadvantages**
  - Message passing overhead inside of SMP nodes (instead of simply accessing the data via the shared memory)!
  - **Reason for using hybrid programming models**

**Exa.:** 2 SMP nodes, 8 CPUs/node

Round-robin
- 0-1-2-3-4-5-6-7
- 8-9-10-11-12-13-14-15
- x14

Sequential
- 0-1-2-3-4-5-6-7
- 8-9-10-11-12-13-14-15
- x8

Optimal ?
- 0-1-2-3-4-5-6-7
- 8-9-10-11-12-13-14-15
- x2

- Slow inter-node link
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
- 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

纯 MPI: Σ=11.6 ms

纯 MPI: only vertical

MPI+OpenMP: only vertical

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

Ratio $T_{hybrid \text{ MPI+OpenMP}} / T_{pure \text{ MPI}}$ – 16*16 CPUs

IBM RS/6000 SP, 208 SMP nodes, each SMP node with 16 POWER3+ CPUs, at NERSC

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

Additional slides (measured at NERSC)
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?
Optimizing the hybrid masteronly model

- By the MPI library:
  - Using multiple threads
    - using multiple memory paths (e.g., for strided data)
    - using multiple floating point units (e.g., for reduction)
    - using multiple communication links
      (if one link cannot saturate the hardware inter-node bandwidth)
  - requires knowledge about free CPUs,
    e.g., via new MPI_THREAD_MASTERONLY

- By the user-application:
  - unburden MPI from work,
    that can be done by the application
    - e.g., concatenate strided data in parallel regions
      instead of using MPI derived datatypes

PROs & CONs “Hybrid”

- PROs “hybrid”
  - more easy to fit hybrid hardware topology
  - no communication overhead inside of the SMP nodes
  - reduced #MPI messages & larger message sizes
    - reduced latency-based overheads
  - save memory
  - reduced #MPI processes
    - better speedup (Amdahl’s law)
    - faster convergence, e.g., if multigrid numeric is computed only on a partial grid

- CONs “hybrid”
  - You may lose your cache (cache flushes inside of OpenMP)
  - Requires additional level of parallelism in problem/program
    - Programming complexity increases & additional synchronization overhead
  - Requires support by MPI library & mature OpenMP compiler
  - Problem to saturate the inter-node network
  - Idling CPUs in the Masteronly scheme
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. (2 experiments)

Comparison II. (theory + experiment)

Comparison III.

**Overlapping computation & communication**

- **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

Implications:
- all other threads are computing while master thread communicates
  - better CPU usage
- MPI-lib must provide at least MPI_THREAD_FUNNELED thread-safety
- inter-node bandwidth may be still used only partially
- Major drawback: load balancing necessary
  - alternative: reserved thread for communication ➔ next slide
Overlapping computation & communication

Alternative:
- Funneled
  MPI only on master-thread
- Funneled & Reserved
  reserved thread for communication

i.e., reserved tasks on threads:
- master thread: 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)

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

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

Small risk of funneled & reserved:
\( \epsilon_{\text{min}} = 1 - \frac{m}{n} \)

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

\( n = \# \text{ threads per SMP node, } m = \# \text{ reserved threads for MPI communication} \)
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 $n_{\text{loc}}$ ⇒ 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

Hybrid MVM implementation

MASTERONLY mode:
- Automatic parallel. of inner i loop (data parallel)
- Single threaded MPI calls

FUNNELED mode:
- Functional parallelism: Simulate asynchronous data transfer ! (OpenMP)
- Release list - LOCK
- Single threaded MPI calls

Parallel Programming Models on Hybrid Platforms

Comparison I. (experiment)

Comparison II. (theory + experiment)

Comparison III.
Comparing other methods

Memory copies from remote memory to local CPU register and vice versa

<table>
<thead>
<tr>
<th>Access method</th>
<th>Copies</th>
<th>Remarks</th>
<th>Bandwidth ( b(message\ size) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-sided MPI</td>
<td>2</td>
<td>Internal MPI buffer + app. rec. buf.</td>
<td>( b(size) = \frac{b_{\infty}}{1 + (1 + \frac{T_{\text{latency}}}{size})} )</td>
</tr>
<tr>
<td>1-sided MPI</td>
<td>1</td>
<td>Application receive buffer</td>
<td>Same formula, but probably better ( b_{\infty} ) and ( T_{\text{latency}} )</td>
</tr>
<tr>
<td>Compiler based:</td>
<td>1</td>
<td>Page based transfer</td>
<td>Extremely poor, if only parts are needed</td>
</tr>
<tr>
<td>OpenMP on DSM</td>
<td>0</td>
<td>Word based access</td>
<td>( \frac{8\ \text{byte}}{0.33\ \mu\text{s}} = 24\text{MB/s} )</td>
</tr>
<tr>
<td>or with cluster extensions, UPC, Co-Array Fortran, HPF</td>
<td>0</td>
<td>Latency hiding with pre-fetch</td>
<td>( \frac{b_{\infty}}{2} )</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Latency hiding with buffering</td>
<td>See 1-sided communication</td>
</tr>
</tbody>
</table>

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?
  - OpenMP Source (Fortran / C) with optimization directives
    1. OMNI Compiler
    2. C-Code + Library calls
    2. Optimizing native compiler

- Optimization of the computation more important than optimization of the communication
Comparison I & II – Some conclusions

Comparison I. (experiment)
- pure MPI
  one MPI process on each CPU

Comparison II. (theory + experiment)
- hybrid MPI + OpenMP
  Masteronly
  MPI only outside of parallel regions

Hybrid masteronly may not saturate the inter-node bandwidth
- Experiments: 1.5 ... 2.7 times faster message transfer with pure MPI (only communication part!)
- hard topology problem with pure MPI

Performance chance $\epsilon < 2$
(with overlapping computation & communication and one communication thread per SMP)
- up to 50% total performance benefit with real matrix-vector-multiply
- cons: overlapping is hard & tricky

Conclusions

• **Pure MPI** versus **hybrid masteronly** model:
  - Topology problem with pure MPI may be hard, e.g., with unstructured grids
  - Communication is bigger with pure MPI, but may be nevertheless faster
  - On the other hand, typically communication is only some percent $\Rightarrow$ relax

• Efficient hybrid programming:
  - one may overlap communication and computation $\Rightarrow$ hard to do!
  - using simple **hybrid funneled & reserved** model,
    you maybe up to 50% faster (compared to masteronly model)

• If you want to use **pure OpenMP** (based on virtual shared memory)
  - try to use still the full bandwidth of the inter-node network
    (keep pressure on your compiler/DSM writer)
  - be sure that you do not lose any computational optimization
    • e.g., best Fortran compiler & optimization directives should be usable

• $\Rightarrow$ optimal parallel programming model on clusters of SMP nodes,
  depends on many factors $\Rightarrow$ no general recommendation