Communication and Optimization Aspects of Parallel Programming Models on Hybrid Architectures

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

Gerhard Wellein
gerhard.wellein@rrze.uni-erlangen.de
University of Erlangen
Regionales Rechenzentrum RRZE

EWOMP 2002, Rome, Italy, Sep. 18–20, 2002

Motivation & Goals

- HPC systems
  - often clusters of SMP nodes = hybrid architectures
- Often hybrid programming (MPI+OpenMP) slower than pure MPI
  - why?
- Using the communication bandwidth of the hardware
- Minimizing synchronization = idle time
- Appropriate parallel programming models
- Pros & Cons
- Work horses for legacy & new parallel applications
- Optimization of the middleware

optimal usage of the hardware
Cluster Programming Models

Rolf Rabenseifner
Höchstleistungsrechenzentrum Stuttgart

Slide 3 / 23

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)

- Masteronly
  MPI only outside of parallel regions
- Overlapping Comm. + Comp.
  MPI communication by one or a few threads
  while other threads are computing
- Funneled
  MPI only on master-thread
- Funneled & Reserved
  reserved thread for communication
- Funneled with Full Load Balancing
- Multiple & Reserved
  reserved threads for communication
- Multiple with Full Load Balancing

Comparison III.

- Multiple
  more than one thread may communicate
- Multiple & Reserved
  reserved threads for communication
- Multiple
  more than one thread may communicate
- Multiple & Reserved
  reserved threads for communication
- Multiple with Full Load Balancing

Comparison I.
(2 experiments)

Comparison II.
(theory + experiment)

Comparison III.

MPI + OpenMP

- MPI_Init_threads(required, &provided)
- categories of thread-safety:
  - no thread-support by MPI
  - MPI process may be sometimes multi-threaded,
    (parallel regions) and
    MPI is called only if only the master-thread exists
  - Same, but the other threads may sleep
  - MPI may be called
    only outside of OpenMP parallel regions
  - Same, but all other threads may compute
  - Multiple threads may call MPI, but only one
    thread may execute an MPI routine at a time
  - MPI may be called from any thread

MPI 2.0: provided=

MPI_THREAD...

..._SINGLE
..._SINGLE
..._SINGLE
..._MASTERONLY
..._FUNNELED
..._SERIALIZED
..._MULTIPLE
**MPI + OpenMP**

- using OMP MASTER ➔ MPI_THREAD_FUNNELED needed
  - no implied barrier!
  - no implied cache flush!

- using OMP SINGLE ➔ MPI_THREAD_SERIALIZED needed

- \[ \text{A}[i] = \ldots \]
  - OMP MASTER / SINGLE
  - MPI_Send(A, \ldots)
  - OMP END MASTER / SINGLE
  - \[ \text{A}[i] = \text{new value} \]
  - OMP BARRIER
  - OMP BARRIER

- Same problem as with MPI_THREAD_MASTERONLY:
  - all application threads are sleeping while MPI is executing

---

**Pure MPI on hybrid architectures**

- Optimizing the communication
  - best ranking of MPI processes on the cluster
    - based on MPI virtual topology
    - sequential ranking: \[0, 1, 2, 3, \ldots, 7, 8, 9, 10, \ldots, 15, 16, 17, 18, \ldots, 23\]
    - round robin: \[0, N, 2N, \ldots, 7N, N+1, \ldots, 7N+7, 2N+2, \ldots, 7N+7\]
      (Example: \(N = \text{number of used nodes, 8 threads per node}\)
Cluster Programming Models
Rolf Rabenseifner
Hochleistungsrechenzentrum Stuttgart

Slide 7 / 23

Pure MPI on hybrid architectures (continued)

- Additional message transfer inside of each node
  - compared with MPI+OpenMP
  - Example: 3-D (or 2-D) domain decomposition
    - e.g. on 8-way SMP nodes
    - one (or 1–3) additional cutting plane in each dimension
    - expecting same message size on each plane
      - outer boundary (pure MPI)
      - inner plane (pure MPI)
      - outer boundary (MPI+OpenMP)

- pure MPI compared with MPI+OpenMP: only doubling the total amount of transferred bytes

Benchmark results

- On Hitachi SR8000, b_eff 1) benchmark on 12 nodes

<table>
<thead>
<tr>
<th></th>
<th>pure MPI</th>
<th>b_eff</th>
<th>3-d-cyclic</th>
<th>3-d-cyclic</th>
</tr>
</thead>
<tbody>
<tr>
<td>(per node)</td>
<td>[MB/s]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>aggregated bandwidth</td>
<td>Hybrid</td>
<td>1535</td>
<td>5565</td>
<td>1604</td>
</tr>
<tr>
<td></td>
<td>[MB/s]</td>
<td>(128)</td>
<td>(464)</td>
<td>(134)</td>
</tr>
<tr>
<td>(per process)</td>
<td>pure MPI</td>
<td>5299</td>
<td>16624</td>
<td>5000</td>
</tr>
<tr>
<td></td>
<td>[MB/s]</td>
<td>(55)</td>
<td>(173)</td>
<td>(52)</td>
</tr>
<tr>
<td>bw_pure MPI / bw_hybrid (measured)</td>
<td>3.45</td>
<td>2.99</td>
<td>3.12</td>
<td>3.27</td>
</tr>
<tr>
<td>size_pure MPI / size_hybrid (assumed)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>T_hybrid / T_pure MPI (concluding)</td>
<td>1.73</td>
<td>1.49</td>
<td>1.56</td>
<td>1.64</td>
</tr>
</tbody>
</table>

- communication with pure MPI model is about 60% faster than with the hybrid-masteronly model

1) www.hlrs.de/mpi/b_eff
2) message size = 8MB
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

\[ \begin{align*}
T_{\text{hybrid}} & = 19.2 \text{ ms} \\
T_{\text{pure MPI}} & = 9.6 \text{ ms} \\
T_{\text{MPI+OpenMP}} & = 11.6 \text{ ms}
\end{align*} \]

---

Cluster Programming Models

Rolf Rabenseifner

Höchstleistungsrechenzentrum Stuttgart
Interpretation of the benchmark results

• If the inter-node bandwidth cannot be consumed by using only one processor of each node
  ➔ the pure MPI model can achieve a better aggregate bandwidth

• If \( \frac{bw_{\text{pure MPI}}}{bw_{\text{hybrid}}} > 2 \) \& \( \frac{\text{size}_{\text{pure MPI}}}{\text{size}_{\text{hybrid}}} < 2 \)
  ➔ faster communication with pure MPI

• If \( \frac{bw_{\text{pure MPI}}}{bw_{\text{hybrid}}} = 1 \) \& \( \frac{\text{size}_{\text{pure MPI}}}{\text{size}_{\text{hybrid}}} > 1 \)
  ➔ faster communication with hybrid MPI+OpenMP

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
    • reduction operations (MPI_reduce / MPI_Allreduce):
      • numerical operations by user-defined multi-threaded call-back routines
      • no rules in the MPI standard about multi-threading of such call-back routine
Other Advantages of Hybrid MPI+OpenMP

- No communication overhead inside of the SMP nodes
- Larger message sizes on the boundary
  - reduced latency-based overheads
- Reduced number of MPI processes
  - better speedup (Amdahl’s law)
  - faster convergence,
    e.g., if multigrid numeric is computed only on a partial grid

Comparison I. (2 experiments)

Comparison II. (theory + experiment)

Comparison III.
Overlapping computation & communication

The model: communication funneled through master thread
- Hybrid MPI+OpenMP
- Requires at least MPI_THREAD_FUNNELED
- While master thread calls MPI routines:
  - all other threads are computing!

The implications:
- no communication overhead inside of the SMP nodes
- better CPU usage
  - although inter-node bandwidth may be used only partially
- 2 levels of parallelism (MPI and OpenMP):
  - additional synchronization overhead
- Major drawback: load balancing necessary
  - alternative: reserved thread for communication → next slide

Overlapping computation & communication

Alternative:
- 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)

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

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

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

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

Experiment: Matrix-vector-multiply (MVM)

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

- One MPI thread should achieve full inter-node bandwidth
- For MPI 2.1: MPI_THREAD_MASTERONLY
  - allows MPI-internal multi-threaded optimizations:
    - e.g., handling of derived data
    - reduction operations
- Application should overlap computation & communication
- Performance chance $\epsilon < 2$
  (with one communication thread per SMP)
- ~50% performance benefit with real CG solver on Hitachi SR8000

Parallel Programming Models on Hybrid Platforms

Comparison I.
(2 experiments)

Comparison II.
(theory + experiment)

Comparison III.

hybrid MPI+OpenMP
MPI only outside of parallel regions

OpenMP only
distributed virtual shared memory

Overlapping Comm. + Comp.
Funneled & Reserved
reserved thread for communication

pure MPI
one MPI process on each CPU
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 + application receive buf.</td>
<td>b(size) = b_∞ / (1 + b_∞T_{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_∞ and T_{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>8 byte / T_{latency}, e.g., 8 byte / 0.33µs = 24MB/s</td>
</tr>
<tr>
<td>or with cluster</td>
<td>0</td>
<td>latency hiding with pre-fetch</td>
<td></td>
</tr>
<tr>
<td>extensions, UPC,</td>
<td></td>
<td>latency hiding with buffering</td>
<td>see 1-sided communication</td>
</tr>
<tr>
<td>Co-Array Fortran,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HPF</td>
<td></td>
<td></td>
<td></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?

- Optimization of the computation more important than optimization of the communication
Conclusions

- **Pure MPI** versus *hybrid masteronly* model:
  - Communication is bigger with pure MPI, but may be nevertheless faster
  - On the other hand, typically communication is only some percent → relax

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

- → Coming from *pure MPI*, one should try to implement *hybrid funneled&reserved 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

Summary

- Programming models on hybrid architectures (clusters of SMP nodes)
  - Pure MPI / hybrid MPI+OpenMP masteronly / funneled-reserved / compiler based parallelization (e.g. OpenMP on clusters)

- Communication
  - difficulties with hybrid programming
    - multi-threading with MPI
    - bandwidth of inter-node network
    - overlapping of computation and communication, real chance: 50% performance benefit
  - latency hiding with compiler based parallelization

- Optimization and compilation
  - separation of optimization
  - we must not lose optimization of computation

- Conclusion:
  - Pure MPI → hybrid MPI+OpenMP → OpenMP on clusters
  - a roadmap full of stones!