Communication Bandwidth of Parallel Programming Models on Hybrid Architectures

Rolf Rabenseifner
rabenseifner@hlrs.de

This talk is given by Matthias Müller on the WOMPEI 2002

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

WOMPEI 2002 at ISHPC-IV, Kansai Science City, Japan, May 15-17, 2002

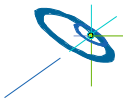
Motivation

- HPC systems
  - often clusters of SMP nodes
  - i.e., hybrid architectures
- Hybrid programming models, e.g.,
  - MPI & OpenMP
  - MPI & automatic loop parallelization
- Often hybrid programming slower than pure MPI programming
  - why?
  - bandwidth problems?

Goals

- Using the communication bandwidth of the hardware (that I have bought)
- Appropriate parallel programming models
- Pros & Cons of existing programming models

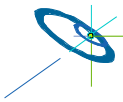
- Work horses for legacy & new parallel applications
- Optimization of the middleware



Programming Large Scale Hybrid Systems

- Interconnect
  - ccNUMA
  - Remote Direct Memory Access (RDMA)
  - only message passing

- Programming models
  - OpenMP on ccNUMA
  - OpenMP cluster extensions
  - MLP (Multi Level Parallelism)
  - Co-Array Fortran & UPC
  - One-sided communication (MPI-2 & shmem)

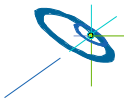


On ccNUMA

- OpenMP on large partitions
  - single level of parallelism
  - Nested parallelism
    - **not yet implemented in most OpenMP compilers**
- OpenMP cluster extensions
  - first touch mechanism
  - data distribution extensions

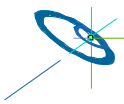
  → **may optimize the data locality**

  → **may reduce communication on interconnect**
- Multi Level Parallelism (MLP) from NASA/Ames
  - a Fortran wrapper to System V shared memory (shm)
  - multi-threaded processes access variables in shared memory segments
  - cheap load balancing:
    changing the number of threads of each process



RDMA (Remote Direct Memory Access)

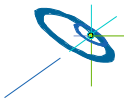
- Co-Array Fortran / Unified Parallel C (UPC)
  - access to variables in other threads/processes is done via additional array subscript in brackets, addressing the thread/process by its rank
  - multi-dimensional ranking is possible
  - not tailored for clusters of SMP, but usable
  - without overhead for • additional message passing
    • additional temporary data copies
  - Lack of portable / public compiling system
  - Architecture may allow **separation** of optimization
    - **Communication:** • **Compiled into ›› RDMA+synchronization and ›› normal sequential code**
    - **Computation:** • **Optimizing compiler for sequential code**
  - This **separation** was the beginning of the **success of MPI**
  - But **not yet done** for Co-Array Fortran / UPC



RDMA platforms (continued)

- MPI-2 one-sided operations
- shmem

- all RDMA programming models can be used also on ccNUMA platforms

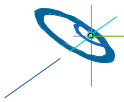


Neither NUMA nor RDMA required

- MPI / PVM on node interconnect
- Inside of the SMP node:
  - MPI → pure MPI (the MPP-MPI model)
  - OpenMP → hybrid programming MPI+OpenMP

focus of this report

- Other models
  - HPF
  - OpenMP and DSM



MPI + OpenMP

- MPI_Init_threads(required, &provided) — MPI 2.0: provided=
- categories of thread-safety: — MPI_THREAD_...
  - no thread-support by MPI — ..._SINGLE
  - MPI process may be *sometimes* multi-threaded, (parallel regions) and MPI is called only if only the master-thread exists — ~~..._SINGLE~~
  - Same, but the other threads may sleep — ~~..._SINGLE~~
  - MPI may be called only outside of OpenMP parallel regions — ~~..._SINGLE~~ **..._MASTERONLY** { proposal for MPI 2.1
  - Same, but all other threads may compute — ..._FUNNELED
  - Multiple threads may call MPI, but only one thread may execute an MPI routine at a time — ..._SERIALIZED
  - MPI may be called from any thread — ..._MULTIPLE

MPI + OpenMP

- using OMP MASTER ➔ MPI_THREAD_FUNNELED needed
  - no implied barrier !
  - no implied cache flush !
- using OMP SINGLE ➔ MPI_THREAD_SERIALIZED needed
- A[i] = ...
  OMP BARRIER
  OMP MASTER / SINGLE
  MPI_Send(A, ...)
  OMP END MASTER / SINGLE
  OMP BARRIER
  A[i] = new value
- Same problem as with MPI_THREAD_MASTERONLY:
  - **all application threads are sleeping while MPI is executing**

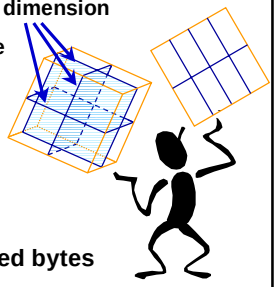
Pure MPI on hybrid architectures

- Optimizing the communication
  - best ranking of MPI processes on the cluster
    - **based on MPI virtual topology**
    - **sequential ranking:** **0, 1, 2, 3, ... 7** **8, 9, 10, ... 15** **16, 17, 18 ... 23**
    - **round robin:** **0, N, 2N ... 7N** **1, N+1, ... 7N+1** **2, N+2, ... 7N+2**

(Example: N = number of used nodes, 8 threads per node)

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**
    - **<u>one</u> (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

	b_eff	b_eff Lmax[2]	3-d-cyclic average	3-d-cyclic Lmax[2]
aggregated bandwidth – **hybrid** [MB/s] (per node)	**1535** (128)	**5565** (464)	**1604** (134)	**5638** (470)
aggregated bandwidth – **pure MPI** [MB/s] (per process)	**5299** (55)	**16624** (173)	**5000** (52)	**18458** (192)
$bw_{\textbf{pure MPI}}$ / $bw_{\textbf{hybrid}}$ [MB/s] (measured)	**3.45**	**2.99**	**3.12**	**3.27**
$size_{\textbf{pure MPI}}$ / $size_{\textbf{hybrid}}$ (assumed)	2 (based on last slide)			
$T_{\textbf{hybrid}}$ / $T_{\textbf{pure MPI}}$ (concluding)	**1.73**	**1.49**	**1.56**	**1.64**

➔ communication in this hybrid model is about 60% slower than with pure MPI

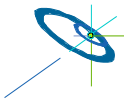
[1] www.hlrs.de/mpi/b_eff/ [2] message size = 8MB

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 $bw_{\textbf{pure MPI}}$ / $bw_{\textbf{hybrid}} > 2$ & $size_{\textbf{pure MPI}}$ / $size_{\textbf{hybrid}} < 2$
  ➔ faster communication with pure MPI
- If $bw_{\textbf{pure MPI}} = bw_{\textbf{hybrid}}$ & $size_{\textbf{pure MPI}} > size_{\textbf{hybrid}}$
  ➔ faster communication with hybrid MPI+OpenMP

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

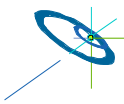


Workaround for single-threaded MPI implementations in the hybrid MPI+OpenMP model

- Work of MPI routines is done by single thread
  - other processors of the SMP nodes may sleep

Communication aspects on last slides
Now, local work of the MPI routines, executed by the CPUs:

- **Workaround for single-threaded MPI implementations**
  - concatenation of strided message data:
    - not by MPI with derived datatypes
    - by multi-threaded user code
  - 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



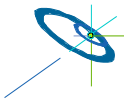
Overlapping computation & communication

The model:

- Hybrid MPI+OpenMP
- At least MPI_THREAD_FUNNELED
- While master thread calls MPI routines:
  - all other threads are computing!

The implications:

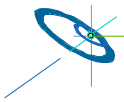
- no communication overhead inside of the SMP nodes
- better CPU usage
  - although inter-node bandwidth may be used only partially
- 2 levels of parallelism:
  - additional synchronization overhead
- Major drawback: load balancing necessary



Comparing other methods

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

Access method	Copies	Remarks	bandwidth *b(message size)*
2-sided MPI	2	internal MPI buffer + application receive buf.	b(size) = b_∞ / (1+ $b_\infty T_{latency}$/size)
1-sided MPI	1	application receive buffer	same formula, but probably better b_∞ and $T_{latency}$
Compiler based: UPC, Co-Array Fortran, HPF, OpenMP on DSM or with cluster extensions	1	page based transfer	extremely poor, if only parts are needed
	0	word based access	8 byte / $T_{latency}$, e.g, 8 byte / 0.33µs = 24MB/s
	0	**latency hiding with pre-fetch**	**b_∞**
	1	latency hiding with buffering	see 1-sided communication

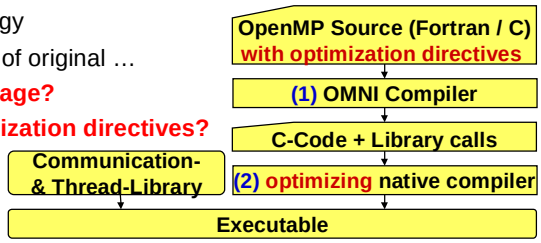


Compilation and Optimization

- 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**

Summary

- **Programming models on hybrid architectures** (clusters of SMP nodes)
  - Pure MPI / hybrid MPI+OpenMP / 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**
  - latency hiding with compiler based parallelization
- **Optimization and compilation**
  - separation of optimization
  - we must not lose **optimization of computation**
- **Conclusion:**
  - Pure MPI → MPI+OpenMP → OpenMP on clusters
  - a roadmap full of stones!