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

Lecture at the University of Heidelberg, June 20, 2003

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

Outline

- Motivation [slides 3–7]
- Major parallel programming models [8–14]
- Programming models on hybrid systems [15–56]
  - Overview [15]
  - Technical aspects with thread-safe MPI [16–18]
  - Mismatch problems with pure MPI and hybrid MPI+OpenMP [19–46]
    - Topology problem [19]
    - Unnecessary intra-node comm. [20]
    - Inter-node bandwidth problem [21–24]
    - Comparison I: Two experiments
    - Sleep thread problem and saturation problem [25–38]
    - Additional OpenMP overhead [39–41]
    - Overlapping comm. and comp. [42–46]
  - Pure OpenMP [47–56]
  - Comparison III
- No silver bullet / optimization chances / other concepts [58–62]
- Acknowledgments & Conclusions [63–64]
Earth Simulator Project ESRDC / GS 40 (NEC)

- Virtual Earth - simulating
  - Climate change (global warming)
  - El Nino, hurricanes, droughts
  - Air pollution (acid rain, ozone hole)
  - Droughtism (earthquakes, volcanism)
- Node: 8 CPUs, 64 GFLOP/s
  - 16 GB, SMP
  - ext. b/w: 2x16 Gb/s
  - System: 640 nodes, 40 TFLOP/s
  - 10 TB memory
  - optical 640x640 crossbar
  - 50m x 20m without peripherals

- System: Single-Chip, 32 GB/s memory b/w

Node 1

Node 640

- Virtual Earth - simulating
  - Climate change (global warming)
  - El Nino, hurricanes, droughts
  - Air pollution (acid rain, ozone hole)
  - Droughtism (earthquakes, volcanism)

- Node: 8 CPUs, 64 GFLOP/s
  - 16 GB, SMP
  - ext. b/w: 2x16 Gb/s

- System: Single-Chip, 32 GB/s memory b/w

Node 640

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?
- Explicit C154N6 SEAM vs T170 PSTSWIM, 16 Level, NCAR

- MPI (Message Passing Interface) (standardized since 1994)
  - User specifies how work & data is distributed
  - User specifies how and when communication has to be done
  - by calling MPI communication library-routines
  - compiler generates normal sequential code (running in each process)
  - typically domain decomposition with communication across domain boundaries
  - Each process has its private variables / Data exchange with messages

OpenMP (standardized since 1997)

- Shared Memory Directives
  - to define the work decomposition
  - no data decomposition
  - synchronization is implicit (can be also user-defined)
  - mainly loops can be parallelized
  - compiler translates OpenMP directives into thread-handling
    - All data is shared / parallel execution threads on the same memory

Major Parallel Programming Models
Shared Memory Directives – OpenMP, I.

```fortran
Real :: A(n,m), B(n,m)
!
OMP PARALLEL DO
    do j = 2, m-1
        do i = 2, n-1
            B(i,j) = ... A(i,j) ...
                ... A(i-1,j) ...
                ... A(i+1,j) ...
                ... A(i-1,j) ...
                ... A(i+1,j) ...
        end do
    end do
!
OMP END PARALLEL DO
```

Data definition
Loop over y-dimension
Vectorizable loop over x-dimension
Calculate B, using upper and lower, left and right value of A

Shared Memory Directives – OpenMP, II.

Single Thread
Parallel Region
Team of Threads
Master Thread

Single Thread
Parallel Region
Team of Threads
Master Thread

Message Passing Program Paradigm – MPI, I.

- Each processor in a message passing program runs a sub-program
  - written in a conventional sequential language, e.g., C or Fortran,
  - typically the same on each processor (SPMD)
- All work and data distribution is based on value of myrank
  - returned by special library routine
- Communication via special send & receive routines (message passing)

Additional Halo Cells – MPI, II.

Halo
(Shadow, Ghost cells)

User defined communication
Message Passing — MPI, Ill.

- Call MPI_Comm_size(MPI_COMM_WORLD, size, ierror)
- Call MPI_Comm_rank(MPI_COMM_WORLD, myrank, ierror)

\[ m_1 = \text{size} + 1; \]
\[ m = \text{size}; \]
\[ j = \text{size} + 1; \]
\[ j = \text{size} + 1; \]

\[ j = j + 1 \]

\[ \text{extended boundary with halo} \]

Real :: A(n, jax:jex), B(n, jax:jex)

do j = 2, m-1

do i = 2, n-1

B(i,j) = ... A(i,j)

... A(i-1,j) ... A(i+1,j)... A(i,j-1) ... A(i,j+1)

end do
end do

Call MPI_Send(........) ! - sending the boundary data to the neighbors
Call MPI_Recv(........) ! - receiving from the neighbors,

! storing into the halo cells

Limitations of the Major Programming Models

- MPI
  - standardized distributed memory parallelism with message passing
  - can be used on any platform, but communication overhead on shared memory systems

- OpenMP
  - standardized shared memory parallelism
  - thread-based
  - compiler translates OpenMP directives into thread-handling

Limitations:
- only for shared memory and ccNUMA systems
- mainly for loop parallelization via OpenMP-directives
- only for medium number of processors
- explicit domain decomposition also via rank of the threads

MPI rules with OpenMP / Automatic SMP-parallelization (2)

- Special MPI-2 Init for multi-threaded MPI processes:
  - int MPI_Init_thread(int * argc, char ** argv[], int required, int * provided)

  - REQUIRED values (increasing order):
    - MPI_THREAD_SINGLE: Only one thread will execute
    - MPI_THREAD_MASTERONLY: MPI processes may be multi-threaded, but only master thread will make MPI-calls
    - MPI_THREAD_FUNNELED: Only master thread will make MPI-calls
    - MPI_THREAD_SERIAL: 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

```bash
!$OMP BARRIER
!$OMP MASTER
    call MPI_Xxx(...) MPI_Xxx(...);
!$OMP END MASTER
!$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

```bash
!$OMP PARALLEL
    !$OMP DO
    do i=1,1000
        a(i) = buf(i)
    end do
    !$OMP END DO NOWAIT!
    !$OMP BARRIER!
    !$OMP MASTER
        call MPI_RECV(buf,...)
    !$OMP END MASTER
    !$OMP BARRIER!
!$OMP DO
    do i=1,1000
        c(i) = buf(i)
    end do
!$OMP END DO NOWAIT
!$OMP 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 pure MPI]
- 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

The Topology Problem with Pure MPI

Ex. 2 SMP nodes, 8 CPUs/node

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

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 (rank 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

Experiment:
Orthogonal parallel communication

Results of the experiment
- pure MPI is better for message size > 32 kB
- long messages:
  \( \frac{T_{\text{hybrid}}}{T_{\text{pure MPI}}} > 1.6 \)
- OpenMP master thread cannot saturate the inter-node network bandwidth

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
Ratio on several platforms

<table>
<thead>
<tr>
<th>Platform</th>
<th>Ratio T_hybrid_masteronly / T_pure_MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM SP</td>
<td>1</td>
</tr>
<tr>
<td>Hitachi SR8000</td>
<td>1.5</td>
</tr>
<tr>
<td>NEC SX6</td>
<td>2</td>
</tr>
<tr>
<td>Cray X1</td>
<td>2</td>
</tr>
<tr>
<td>SGI O3000</td>
<td>2.5</td>
</tr>
<tr>
<td>Cray X1</td>
<td>3</td>
</tr>
</tbody>
</table>

Pure MPI is faster than 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.

2nd Experiment: Multiple inter-node communication paths

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

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

Hybrid is faster than pure MPI if ≥ 2 CPUs/node are used for intra-node communication in hybrid programming model

Multiple inter-node communication paths: Hitachi SR8000

Hybrid communication time / pure MPI communication time on Hitachi SR8000

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

With 3 CPUs similar to pure MPI

The second CPU doubles the accumulated bandwidth

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

The second CPU doubles the accumulated bandwidth

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

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

Measurements: Thanks to Holger Berger, NEC.
Multiple inter-node communication paths:

Cray X1, used with 4 MSPs/node, accumulated over its CPUs (preliminary results)

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

- 1 MSP achieves already 80% of full inter-node bandwidth

Intra-node messages do not count for bandwidth

Highest parallel bandwidth: 12.0 GF/s

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)

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

- Cray X1: SMP / 960 kB
- NEC SX6: global mem / 960 kB
- Hitachi SR8000: 960 kB
- IBM SP/Power3+ / 960 kB

Communicating CPUs per SMP node as percentage of the total number of CPUs per SMP node

Cray X1 results are preliminary

Comparing inter-node bandwidth with peak CPU performance

<table>
<thead>
<tr>
<th>System</th>
<th>Master-only, intra-node bw (GB/s)</th>
<th>pure MPI, intra-node bw (GB/s)</th>
<th>pure MPI, inter-node bw (GB/s)</th>
<th>memroy bandwidth (GB/s)</th>
<th>Peak performance (Gflop/s)</th>
<th>max. inter-node bw / peak perf. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cray X1, preliminary</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>Cray X1, MPI</td>
<td>4.52</td>
<td>5.52</td>
<td>92</td>
<td>19.5</td>
<td>136</td>
<td>51.2</td>
</tr>
<tr>
<td>NEC SX-6</td>
<td>7.56</td>
<td>4.98</td>
<td>100</td>
<td>79</td>
<td>256</td>
<td>64</td>
</tr>
<tr>
<td>NEC SX-5Is</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>Hitachi SR8000</td>
<td>0.45</td>
<td>0.91</td>
<td>49</td>
<td>5.0</td>
<td>16</td>
<td>24</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>SGI Origin 3000</td>
<td>0.20</td>
<td>0.60</td>
<td>33</td>
<td>3.2</td>
<td>48</td>
<td>4.8</td>
</tr>
<tr>
<td>SUN-Iro (prelim.)</td>
<td>0.15</td>
<td>0.85</td>
<td>18</td>
<td>1.68</td>
<td></td>
<td>4 * 24 CPUs</td>
</tr>
</tbody>
</table>

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

The sleeping-threads and the saturation problem

- Master-only:
  - 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
Mismatch Problems

- Topology problem
- Unnecessary intra-node communication
- Inter-node bandwidth problem
- Sleeping threads and saturation problem
- Additional OpenMP overhead
  - Thread fork/join
  - Cache flush
- Overlapping communication and computation
  - 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

- The load balancing problem:
  - Some threads communicate, others do not
  - Balance work on both types of threads
- Strategies:
  - Funneled & reserved
  - Multiple & reserved
- Pros:
  - More easy programming scheme than with full load balancing
  - Chance for good performance!
- Cons:
  - Bad load balance, if
    - Communication vs. computation
    - Communication_threads ≠ Computation_threads
  - Very hard to do!!!

Parallel Programming Models on Hybrid Platforms

- Comparison I.
- Comparison II.
- Comparison III.

Overlapping computation & communication (cont’d)

- Funneled & reserved or Multiple & reserved:
  - Reserved tasks on threads:
    - One thread or some threads: communication
    - All other threads: computation
  - Pros:
    - More easy programming scheme than with full load balancing
    - Chance for good performance!
Performance ratio (theory)

\[ \varepsilon = \frac{\text{Hybrid, funneled & reserved}}{\text{Hybrid, masteronly}} \]

- Good chance of funneled & reserved: \( \varepsilon_{\text{max}} = 1 + \omega(1 - f_{\text{comm}}) \)
- Small risk of funneled & reserved: \( \varepsilon_{\text{min}} = 1 - \frac{m}{n} \)

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

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

Comparison II.
- Funneled & Reserved
- MPI only outside of parallel regions
- reserved thread for communication

Comparison III.
- Multiple & Reserved
- more than one thread may communicate
- Full Load Balancing


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
  - Communication- & Thread-Library
  - Executable
- 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

Case Studies

- NAS Parallel Benchmarks EP, FT, and CG:
  - Message passing and sequential version
  - Automatically generate OpenMP directives for sequential code using CAPO
    (www.nas.nasa.gov/Groups/Tools/CAPO)
  - Omni Compiler
  - Compare speedup of:
    - Message passing vs. OpenMP/DSM
    - OpenMP/DSM vs. OpenMP/SMP
  - Hardware platforms:
    - DSM Test Environment
      - Use only one CPU per node
    - SMP 16-way NEC Azusa
  - Case study was conducted with Gabrielle Jost from NASA/Ames and Matthias Hess, Matthias Mueller from HLRS

The EP Benchmark

- Embarassing Parallel:
  - Generation of random numbers
  - Loop iterations parallel
  - Global sum reduction at the end
  - Automatic Parallelization without user interaction
  - MPI implementation:
    - Global sum built via MPI_ALLREDUCE
    - Low communication overhead (< 1%)
  - OpenMP/DSM:
    - OMP PARALLEL
    - OMP DO REDUCTION

Linear speedup for MPI and OpenMP/DSM.
No surprises.
**CG Benchmark Results (1)**

- Conjugate Gradient method to solve an eigenvalue problem
  - Stresses irregular data access
  - Major loops:
    - Sparse Matrix-Vector-Multiply
    - Dot-Product
    - AXPY Operations
  - Same major loops in MPI and OpenMP implementation
  - Automatic parallelization without user interaction
  - Class A:
    - Problem size: na=14000, nz=11
    - OpenMP/DSM efficiency about 75% of that of MPI
  - Class B:
    - Problem size: na=1400, nz=7
    - MPI about 20% communication.
    - No speedup for OpenMP/DSM due to:
      - Large Communication to Computation Ratio
      - Inefficiencies in the Omni Compiler

**CG Benchmark Results (2)**

- OpenMP/DSM efficiency about 70% of MPI
  - Extra communication introduced by DSM system (false page sharing?)
  - Remote data access required for FFT in z-dimension

**FT Benchmark Results (1)**

- Kernel of spectral method based on 3D Fast Fourier Transform (FFT)
  - 3D FFT achieved by a 1D FFT in x, y, and z direction
- MPI Parallelization:
  - Transpose of data for FFT in z-dimension
  - 15% in communication
- OpenMP Parallelization:
  - OpenMP parallelization required some user interaction
  - Privatization of certain arrays via the CAPO user interface
  - OMP DO PARALLEL
  - Order of loops changes for z-dimension

**OpenMP/DSM: Conclusions:**

- Rapid development of parallel code running across a cluster of PCs was possible
- OpenMP/DSM delivered acceptable speedup if the communication/computation ratio is not too high:
  - OpenMP/DSM showed between 70% and 100% efficiency compared to MPI for benchmarks of Class A
- Problems encountered:
  - High memory requirements for management of virtual shared memory (> 2GB)
  - Potential scalability problems
- Need for profiling tools
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 Sys V shared memory (data access between processes)
      - only on ccNUMA
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):

```fortran
integer a[*], b[*] ! Replicate a and b on all processes
a[1] = b[6] ! a on process 1 = b on process 6

dimension (n,n) :: u[*,*] ! Allocates the n x n array
u(1:n,1) = u(1:n,n) ! 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
  - Gabriele Jost, NASA
  - Dieter an Mey, RZ Aachen
  - Horst Simon, NERSC
  - My colleges at HLRS
  - Thomas Ludwig, University of Heidelberg

Conclusions

- Only a few platforms (e.g. Cray X1 in MSP mode, or NEC SX-6)
  - Are well-designed hybrid MPI+OpenMP master-only scheme
- Other platforms
  - Master-only style cannot saturate inter-node bandwidth
  - Optimization chances should be used
- Pure MPI and hybrid master-only:
  - 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

See also: www.hlrs.de/people/rabenseifner ➔ List of publications