Parallel Programming Models on Hybrid Systems

MPI + OpenMP and other models on clusters of SMP nodes

Rolf Rabenseifner
rabenseifner@hlrs.de

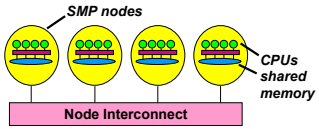
University of Stuttgart
High-Performance Computing-Center Stuttgart (HLRS)
www.hlrs.de

Lecture at the University of Heidelberg, June 20, 2003

Outline

Motivation

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



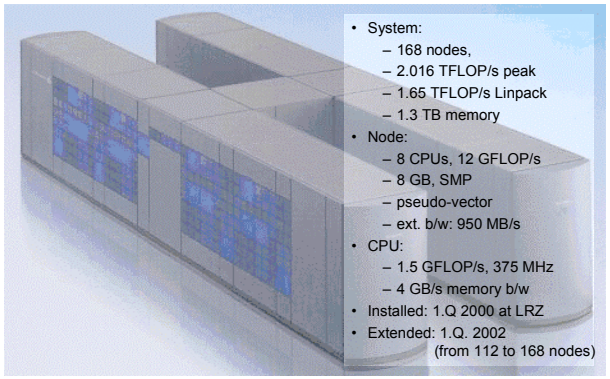
- Using the communication bandwidth of the hardware
- Minimizing synchronization = idle time

} **optimal usage of the hardware**

- Appropriate parallel programming models / Pros & Cons

skipped

Hitachi SR 8000-F1/112 (Rank 5 in TOP 500 / June 2000)

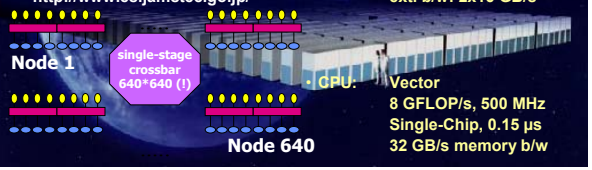


- System:
  - 168 nodes,
  - 2.016 TFLOP/s peak
  - 1.65 TFLOP/s Linpack
  - 1.3 TB memory
- Node:
  - 8 CPUs, 12 GFLOP/s
  - 8 GB, SMP
  - pseudo-vector
  - ext. b/w: 950 MB/s
- CPU:
  - 1.5 GFLOP/s, 375 MHz
  - 4 GB/s memory b/w
- Installed: 1.Q 2000 at LRZ
- Extended: 1.Q. 2002 (from 112 to 168 nodes)

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/

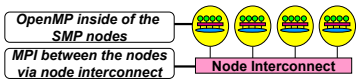
- System: 640 nodes, 40 TFLOP/s
  10 TB memory
  optical 640x640 crossbar
  50m x 20m without peripherals
- Node: 8 CPUs, 64 GFLOP/s
  16 GB, SMP
  ext. b/w: 2x16 GB/s



- CPU: Vector
  8 GFLOP/s, 500 MHz
  Single-Chip, 0.15 µs
  32 GB/s memory b/w

Major Programming models on hybrid systems

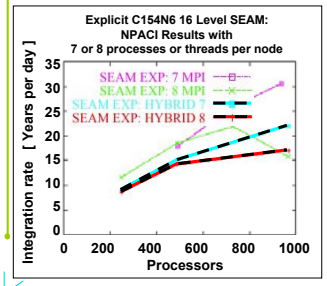
- Pure MPI (one MPI process on each CPU)
- Hybrid MPI+OpenMP
  - shared memory OpenMP
  - distributed memory MPI

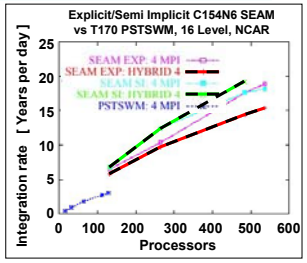


- Other: Virtual shared memory systems, HPF, …
- Often **hybrid programming (MPI+OpenMP)** slower than **pure MPI**
  - why?

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.
Proceedings of SC2001, Denver, USA, Nov. 2001.
http://www.sc2001.org/papers/pap.pap189.pdf
Fig. 9 and 10.

Major Parallel Programming Models

- **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**
- **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**

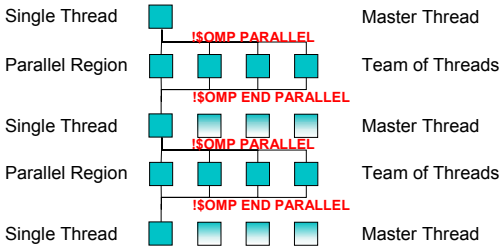
Shared Memory Directives – OpenMP, I.

```
Real :: A(n,m), B(n,m)

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

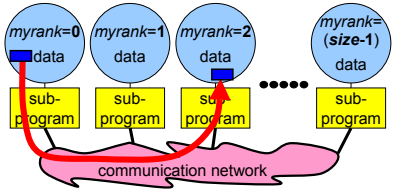
- Data definition
- Loop over y-dimension
- Vectorizable loop over x-dimension
- Calculate B,
  - using upper and lower,
  - left and right value of A

Shared Memory Directives – OpenMP, II.

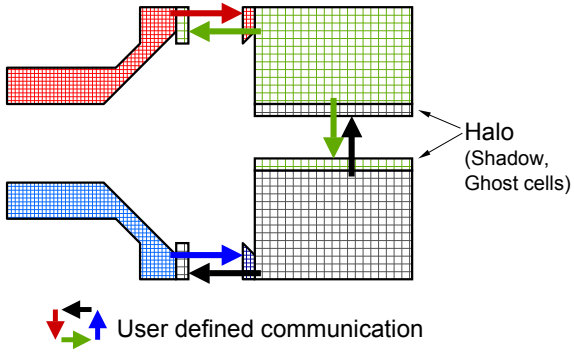


Message Passing Program Paradigm – MPI, I.

- 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.



Message Passing – MPI, III.

```
Call MPI_Comm_size(MPI_COMM_WORLD, size, ierror)
Call MPI_Comm_rank(MPI_COMM_WORLD, myrank, ierror)
m1 = (m+size-1)/size;   ja=1+m1*myrank;   je=max(m1*(myrank+1), m)
jax=ja-1;  jex=je+1   // extended boundary with halo

Real :: A(n, jax:jex), B(n, jax:jex)
do j = max(2,ja), min(m-1,je)
  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
```

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

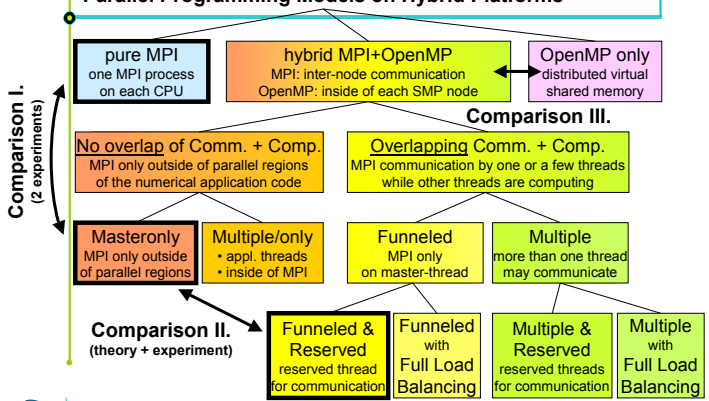
Limitations of the Major Programming Models

- MPI
  - standardized distributed memory parallelism with message passing
  - process-based
  - the application processes are calling MPI library-routines
  - compiler generates normal sequential code

  Limitations:
  - **the amount of your hours available for MPI programming**
  - **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**

Parallel Programming Models on Hybrid Platforms



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*)
  MPI_INIT_THREAD(REQUIRED, *PROVIDED, IERROR*)

- REQUIRED values (increasing order):
  - **MPI_THREAD_SINGLE:** **Only one thread will execute**
  - **THREAD_MASTERONLY:** (virtual value, not part of the standard) **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!

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

```
#pragma omp barrier
#pragma omp master
      MPI_Xxx(...);

#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

```
!$OMP PARALLEL
!$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
```

```
#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] [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

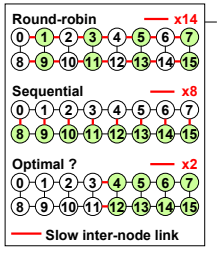
→ **no silver bullet**
  - **each parallelization scheme has its problems**

The Topology Problem with Pure MPI

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

pure MPI
one MPI process on each CPU

Exa.: 2 SMP nodes, 8 CPUs/node



Advantages
- No modifications on existing MPI codes
- MPI library need not to support multiple threads

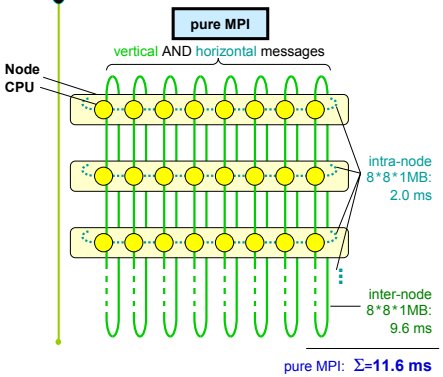
Problems
- To fit application topology on hardware topology

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

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



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

Programming Models on Hybrid Platforms: Hybrid Masteronly

Masteronly
MPI only outside of parallel regions

```
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
```

Advantages
- No message passing inside of the SMP nodes
- No topology problem

Problems
- MPI-lib must support MPI_THREAD_FUNNELED

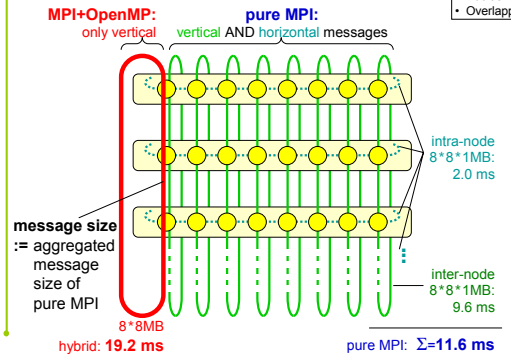
Disadvantages
- do we get full inter-node bandwidth? **… next slide**
- all other threads are sleeping while master thread communicates

→Reason for implementing overlapping of communication & computation

Experiment: Orthogonal parallel communication

pure MPI
Masteronly

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



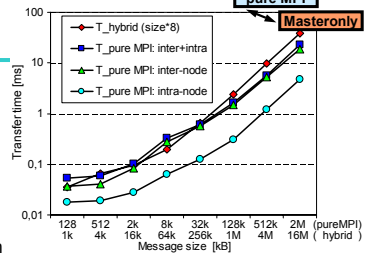
Hitachi SR8000
- 8 nodes
- each node with 8 CPUs
- MPI_Sendrecv

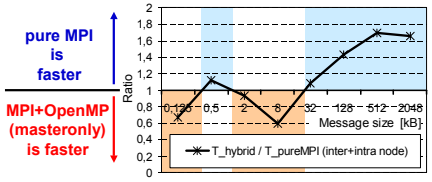
→ **1.6x slower** than with pure MPI, **although**
- only half of the transferred bytes
- and less latencies due to 8x longer messages

Results of the experiment

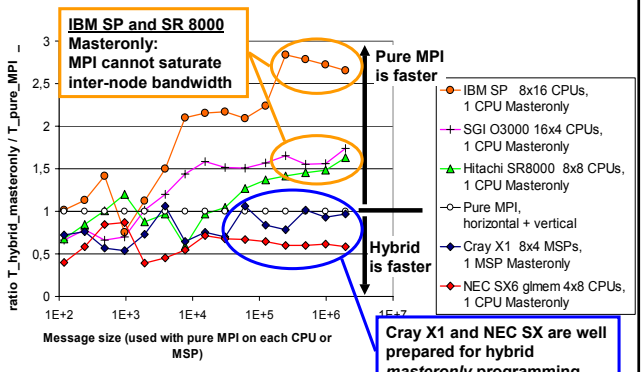
pure MPI
Masteronly

- pure MPI is better for message size > 32 kB
- long messages: $T_{hybrid} / T_{pureMPI} > 1.6$
- OpenMP master thread cannot saturate the inter-node network bandwidth





Ratio on several platforms



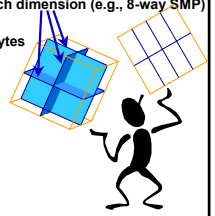
Cray X1 and NEC SX are well prepared for hybrid *masteronly* programming

Cray X1 and SGI results are preliminary

Pure MPI versus Hybrid-masteronly

pure MPI → Masteronly

- Data transfer
  - Pure MPI:
    - **Typically in message passing epochs**
    - **→ inter-node network saturated by a few processes per node**
    - **Best case: only one additional cutting plane in each dimension (e.g., 8-way SMP) compared to hybrid MPI+OpenMP**
    - **→ only doubling the total amount of transferred bytes**
  - Hybrid-masteronly:
    - **other threads are sleeping while master thread calls MPI routines**
    - **node-to-node communication time therefore weighted by number of processors/node !!!**
    - **→ node-to-node bandwidth = 1 GB/s is reduced to 125 MB/s (on 8 CPUs/node)**
    - **latency = 20 µs explodes to 160 µs**

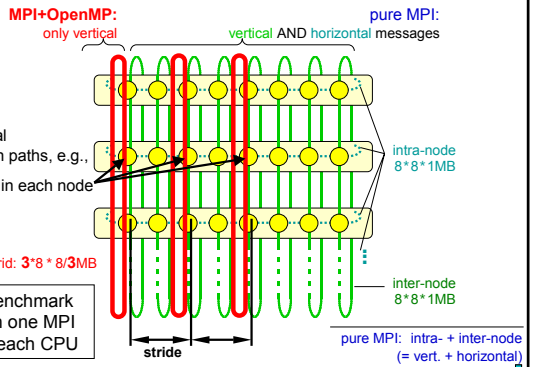


Possible Reasons

- 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?

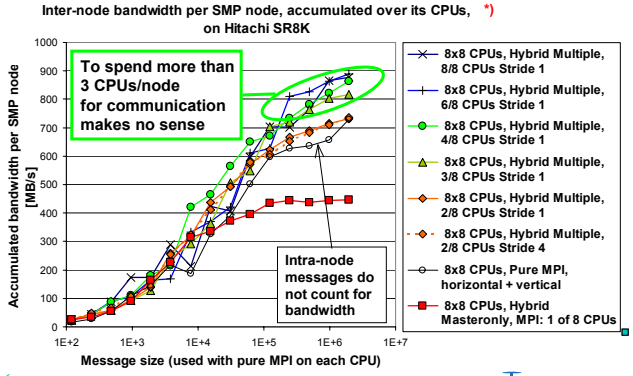
➔ Let's look at parallel bandwidth results

2nd Experiment: Multiple inter-node communication paths



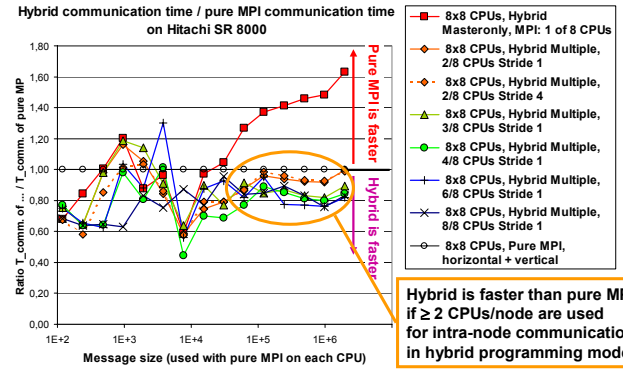
Following benchmark results: with one MPI process on each CPU

Multiple inter-node communication paths: Hitachi SR8000



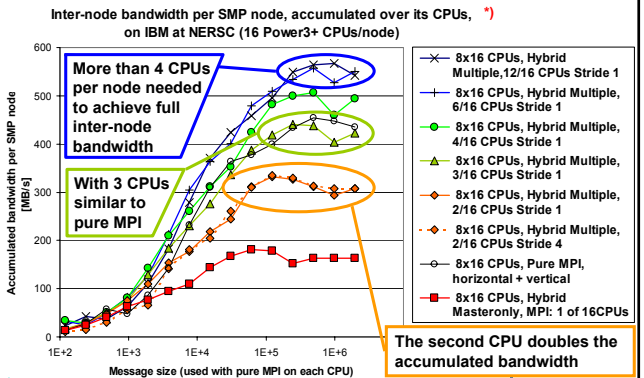
*) Bandwidth per node: totally transferred bytes on the inter-node network / wall clock time / number of nodes

Multiple inter-node communication paths: Hitachi SR 8000



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: IBM SP

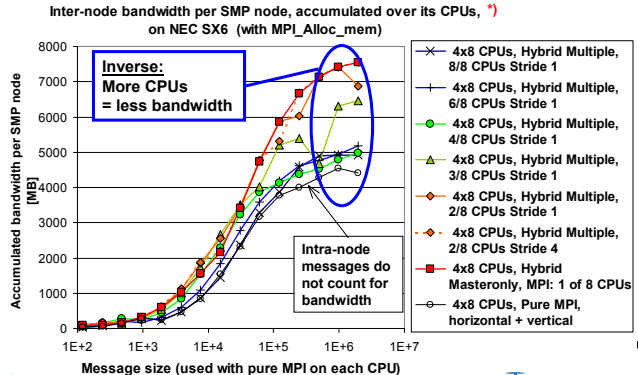


The second CPU doubles the accumulated bandwidth

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

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

Multiple inter-node communication paths: NEC SX-6 (using global memory)

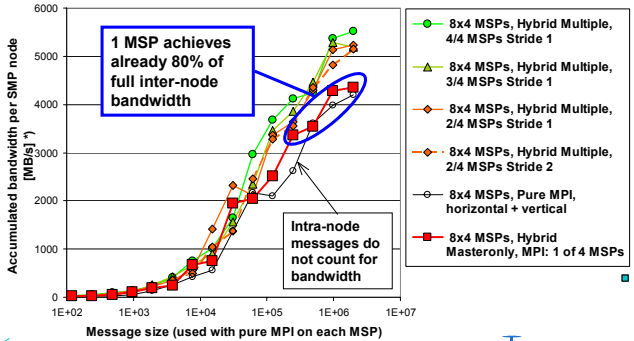


*) Bandwidth per node: totally transferred bytes on the inter-node network / wall clock time / number of nodes

Measurements: Thanks to Holger Berger, NEC.

Multiple inter-node communication paths: Cray X1, used with 4 MSPs/node (preliminary results)

Inter-node bandwidth per SMP node, accumulated over its CPUs, *)
on Cray X1, 4 MSPs / node (1 MSP = 4 CPUs)



H L R S

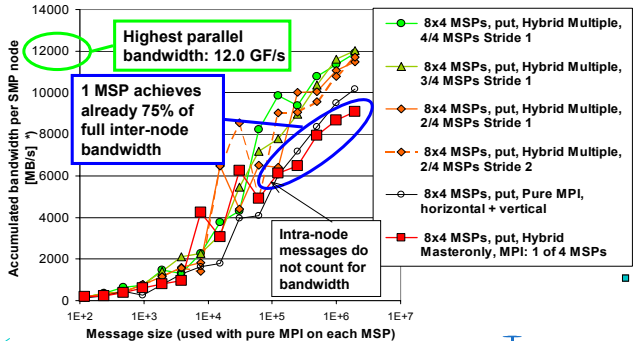
*) Bandwidth per node: totally transferred bytes on the inter-node network / wall clock time / number of nodes

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

— skipped —

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



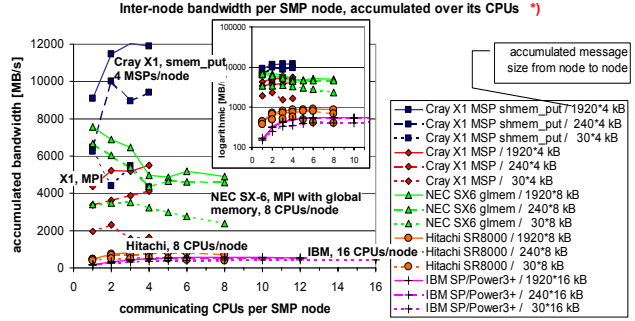
H L R S

*) 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.

Comparison

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



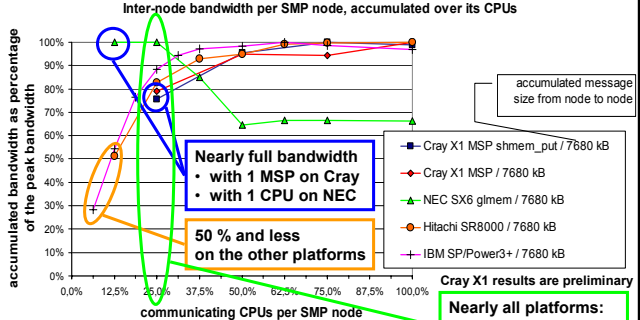
Cray X1 results are preliminary

H L R S

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

Comparison (as percentage of maximal bandwidth and #CPUs)

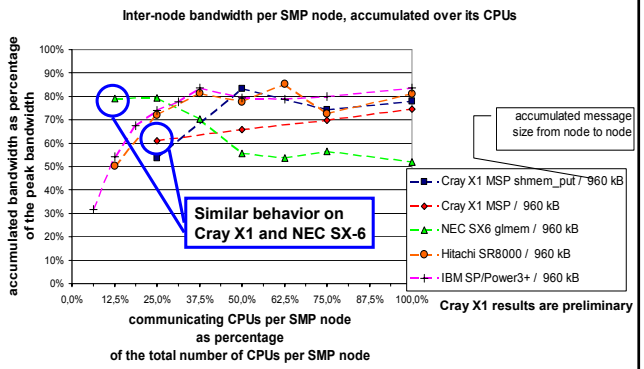
Inter-node bandwidth per SMP node, accumulated over its CPUs



H L R S

skipped

Comparison (only 960 kB aggregated message size)



Comparing inter-node bandwidth with peak CPU performance

All values: aggregated over one SMP nodes. *) mess. size: 16 MB +) 2 MB	Master-only, inter-node [GB/s]	pure MPI, inter-node [GB/s]	Master-only bw / max. intra-node bw	pure MPI, intra-node [GB/s]	memory bandwidth [GB/s]	Peak performance Gflop/s	max. inter-node bw / peak perf. B/Flop	nodes*CPUs
Cray X1, shmem_put preliminary results	9.27	**12.34**	75 %	33.0	136	**51.2**	0.241	8 * 4 MSPs
Cray X1, MPI preliminary results	4.52	**5.52**	82 %	19.5	136	**51.2**	0.108	8 * 4 MSPs
NEC SX-6 global memory	**7.56**	4.98	100 %	78.7 93.7+)	256	**64**	0.118	4 * 8 CPUs
NEC SX-5Be local memory	2.27	**2.50** a)	91 %	35.1	512	**64**	0.039	2 *16 CPUs a) only with 8
Hitachi SR8000	0.45	**0.91**	49 %	5.0	32 store 32 load	**8**	0.114	8 * 8 CPUs
IBM SP Power3+	0.16	**0.57+)**	28 %	2.0	16	**24**	0.023	8 *16 CPUs
SGI Origin 3000 preliminary results	0.10	**0.30+)**	33 %	0.39+)	3.2	**4.8**	0.063	16 *4 CPUs
SUN-fire (prelimi.)	0.15	**0.85**	18 %	1.68				4 *24 CPUs

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

The sleeping-threads and the saturation problem

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

- 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**

Additional OpenMP Overhead

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

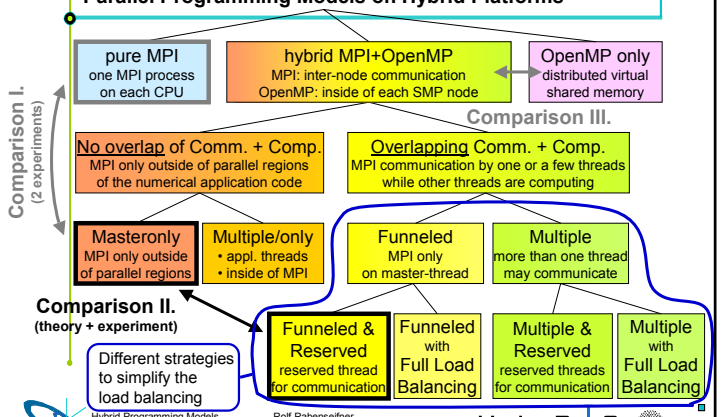
- Thread fork / join
- Cache flush
  - synchronization between *data source thread* and *communicating thread* implies → a cache flush
- Amdahl's law for each level of parallelism

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] [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**

Parallel Programming Models on Hybrid Platforms



Overlapping communication and computation

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

- the load balancing problem:
  - some threads communicate, others not
  - balance work on both types of threads
  - strategies:

Funneled & Reserved reserved thread for communi.	Multiple & Reserved reserved threads for communic.

  - 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

Funneled with Full Load Balancing	Multiple with Full Load Balancing

→ very hard to do !!!

Overlapping computation & communication (cont'd)

funneled & reserved

Funneled & reserved or Multiple & reserved:

- reserved tasks on threads:
  - master thread or some threads: communication
  - all other threads …….. : computation
- cons:
  - bad load balance, if

$$\frac{T_{communication}}{T_{computation}} \neq \frac{n_{communication\_threads}}{n_{computation\_threads}}$$

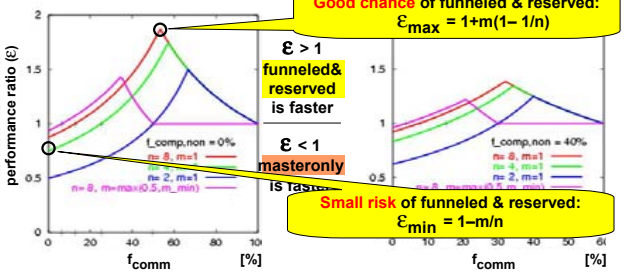
- pros:
  - more easy programming scheme than with full load balancing
  - chance for good performance!

Performance ratio (theory)

Masteronly — funneled & reserved

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

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



$\varepsilon > 1$ funneled& reserved is faster

$\varepsilon < 1$ masteronly is faster

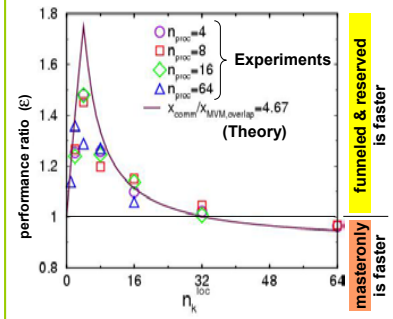
Small risk of funneled & reserved: $\varepsilon_{min} = 1-m/n$

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

n = # threads per SMP node, m = # reserved threads for MPI communication

Experiment: Matrix-vector-multiply (MVM)

Masteronly — funneled & reserved

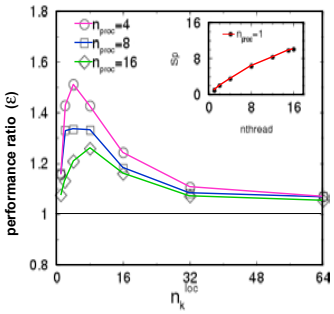


- Jacobi-Davidson-Solver
- Hitachi SR8000
- 8 CPUs / SMP node
- JDS (Jagged Diagonal Storage)
- vectorizing
- n_{proc} = # SMP nodes
- $D_{Mat} = 512*512*(n_k^{loc}*n_{proc})$
- Varying n_k^{loc} ⇒ Varying $1/f_{comm}$
- $\frac{f_{comp,non\text{-}overlap}}{f_{comp,overlap}} = \frac{1}{6}$

Source: R. Rabenseifner, G. Wellein: Communication and Optimization Aspects of Parallel Programming Models. EWOMP 2002, Rome, Italy, Sep. 18–20, 2002

Experiment: Matrix-vector-multiply (MVM)

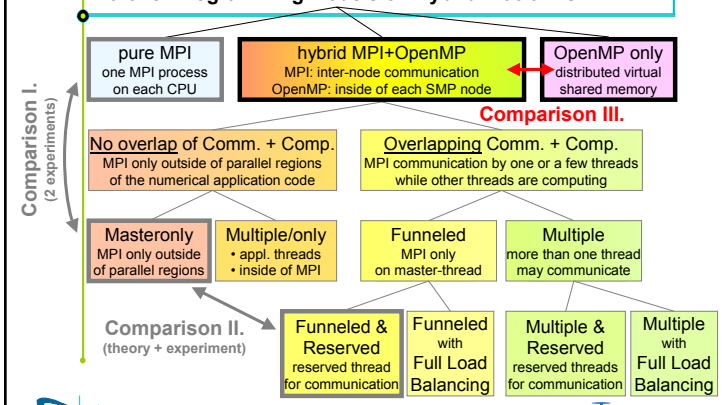
Masteronly — funneled & reserved



- 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

Source: R. Rabenseifner, G. Wellein: Communication and Optimization Aspects of Parallel Programming Models on Hybrid Architectures. International Journal of High Performance Computing Applications, Vol. 17, No. 1, 2003, Sage Science Press .

Parallel Programming Models on Hybrid Platforms

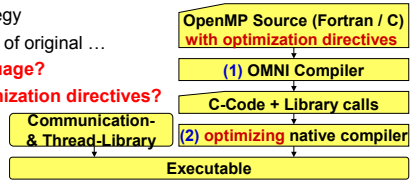


hybrid MPI+OpenMP ↔ OpenMP only

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**

pure MPI ↔ OpenMP only

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

pure MPI ↔ OpenMP only

Case Studies

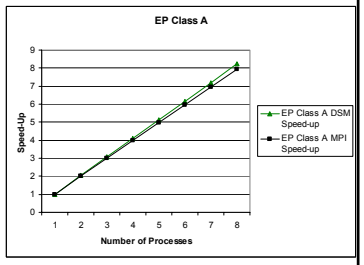
- NAS Parallel Benchmarks EP, FT, and CG:
  - Message passing and sequential version
- Automatically generate OpenMP directives for sequential code using CAPO (www.nas.nasa.gov/Groups/Tools/CAPO)
- Omni Compiler
- Compare speedup of:
  - Message passing vs. OpenMP/DSM
  - OpenMP/DSM vs. OpenMP/SMP
- 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

Slides courtesy of Gabriele Jost, NASA/AMES, and Matthias Müller, HLRS

pure MPI ↔ OpenMP only

The EP Benchmark

- **E**mbarrassing **P**arallel:
  - 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



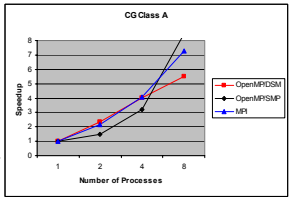
Linear speedup for MPI and OpenMP/DSM. No surprises.

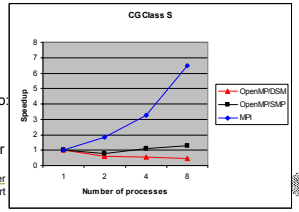
Slides courtesy of Gabriele Jost, NASA/AMES, and Matthias Müller, HLRS

pure MPI ↔ OpenMP only

CG Benchmark Results (1)

- **C**onjugate **G**radient 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 S:
  - 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

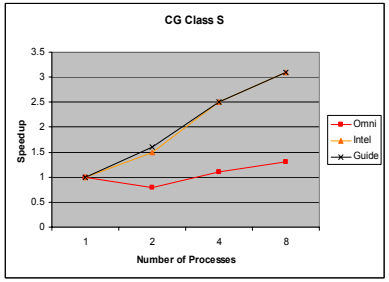




Slides courtesy of Gabriele Jost, NASA/AMES, and Matthias Müller, HLRS

pure MPI ↔ OpenMP only

CG Benchmark Results (2)



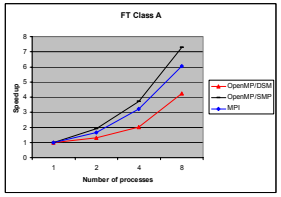
Slides courtesy of Gabriele Jost, NASA/AMES, and Matthias Müller, HLRS

skipped

pure MPI ↔ OpenMP only

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

Slides courtesy of Gabriele Jost, NASA/AMES, and Matthias Müller, HLRS

pure MPI ↔ OpenMP only

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

Slides courtesy of Gabriele Jost, NASA/AMES, and Matthias Müller, HLRS

Outline

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

No standards!
Only on a few platforms!

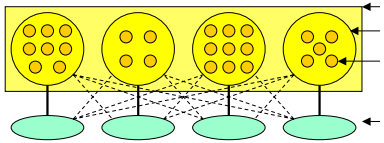
skipped

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
a[1] = b[6]                         ! a on process 1  :=  b on process 6
-------------------------------------------------------------------------
dimension (n,n) :: u[3,*]           ! 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,)
```

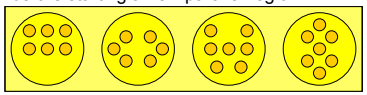

Multi Level Parallelism (MLP)



- 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
  - Gabriele Jost, NASA
  - Dieter an Mey, RZ Aachen
  - Horst Simon, NERSC
  - my colleges at HLRS

  and
  - 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 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

See also www.hlrs.de/people/rabenseifner → list of publications