Parallel Programming Models on Hybrid Systems

Rolf Rabenseifner
rabenseifner@hlrs.de

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

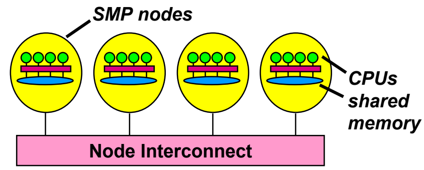
Tutorial at the HLRS Results and Review Workshop
“High Performance Computing in Science and Engineering”

Oct. 6–7, 2003, Stuttgart, Germany

Outline

Motivation

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



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

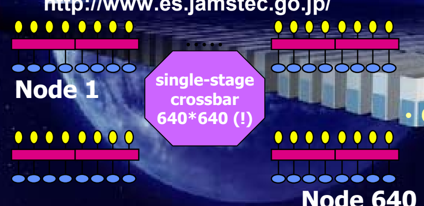
} **optimal usage of the hardware**

- Appropriate parallel programming models / Pros & Cons

Earth Simulator Project ESRDC / GS 40 (NEC)

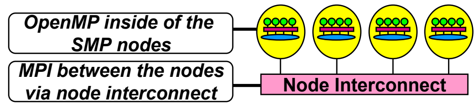
- **Virtual Earth - simulating**
  - **Climate change (global warming)**
  - **El Niño, hurricanes, droughts**
  - **Air pollution (acid rain, ozone hole)**
  - **Diastrophism (earthquake, volcanism)**
- **Installation: 2002**
  http://www.es.jamstec.go.jp/

- **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

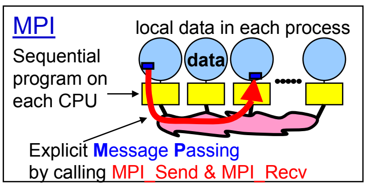


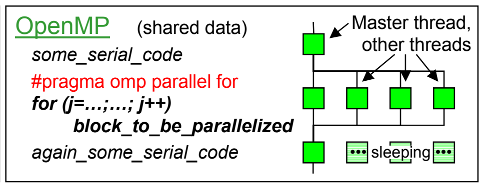
Major Programming models on hybrid systems

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



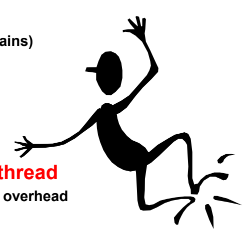
- Other: Virtual shared memory systems, HPF, …
- Often **hybrid programming (MPI+OpenMP)** slower than **pure MPI**
  - why?





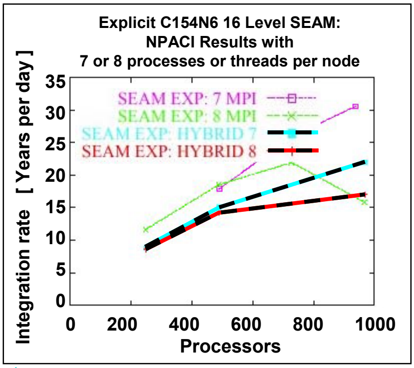
But results may surprise!

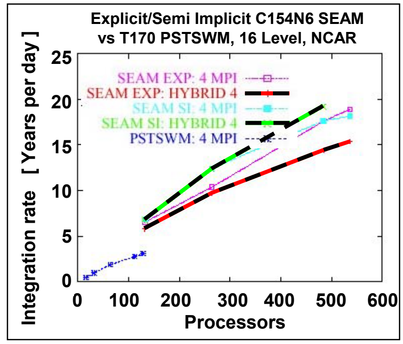
- Example code - HYDRA
- Domain-decomposed hydrodynamics
  - (almost) independent mesh domains with ghost cells on boundaries
  - ghost cells communicate boundary information ~40-50 times per cycle
- Parallelism model: single level
  - MPI divides domains among compute nodes
  - OpenMP further subdivides domains among processors
  - domain size set for cache efficiency
    - **minimizes memory usage, maximizes efficiency**
    - **scales to very large problem sizes (>10^7 zones, >10^3 domains)**
- Results:
  - **MPI** (256 proc.) **~20% faster than MPI / OpenMP** (64 nodes x 4 proc./node)
  - domain-domain communication not threaded, i.e., **MPI communication is done only by main thread**
    - **accounts for ~10% speed difference, remainder in thread overhead**



Example from SC 2001

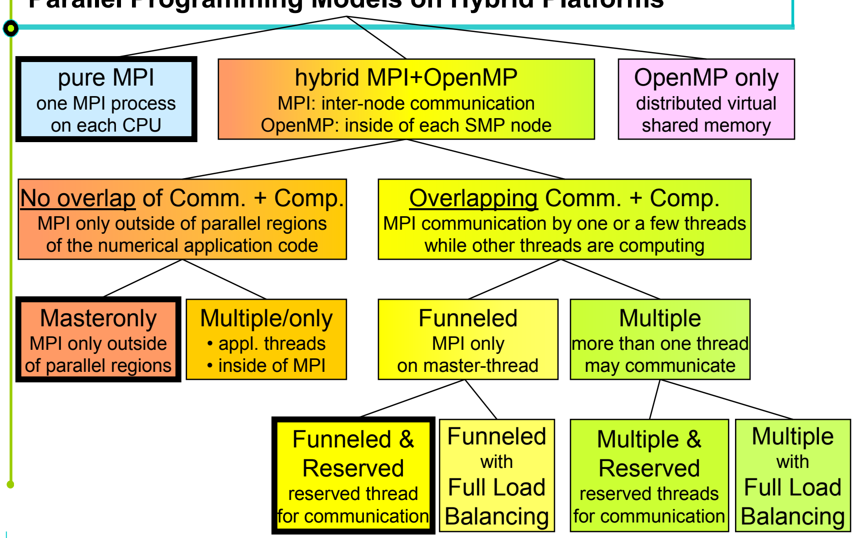
- Pure MPI versus
  Hybrid MPI+OpenMP (Masteronly)
- What's better?
  → it depends on?





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.

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** [with masteronly]
  saturation problem [with pure MPI]
- **Additional OpenMP overhead** [with hybrid MPI+OpenMP]
  - Thread startup / 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

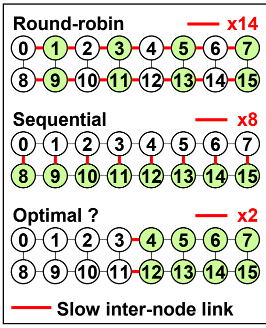
→ **no silver bullet**
  - **each parallelization scheme has its problems**

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

The Topology Problem with Pure MPI

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

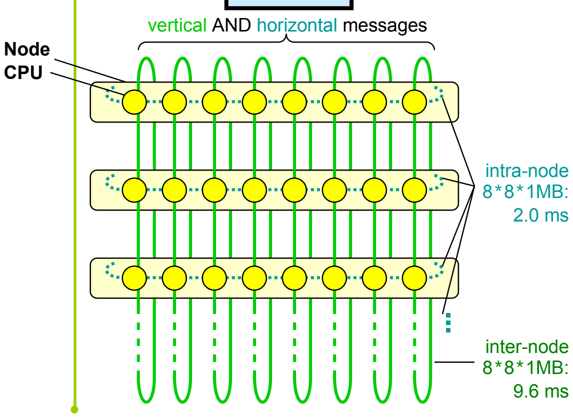
Solutions for Cartesian grids:
- E.g. choosing ranks in MPI_COMM_WORLD ???
  - **round robin (rank 0 on node 0, rank 1 on node 1, ...)**
  - **Sequential (ranks 0-7 on 1st node, ranks 8-15 on 2nd …)**

… 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**

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

Unnecessary intra-node communication

pure MPI



pure MPI: **Σ=11.6 ms**

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

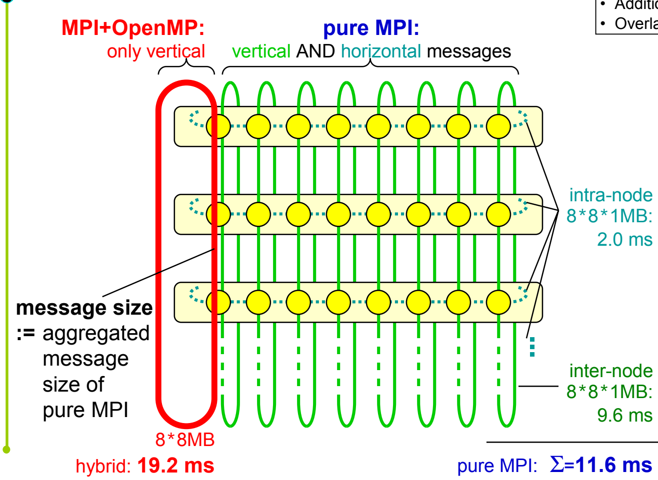
→Reason for implementing overlapping of communication & computation

pure MPI
Masteronly

Experiment: Orthogonal parallel communication

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

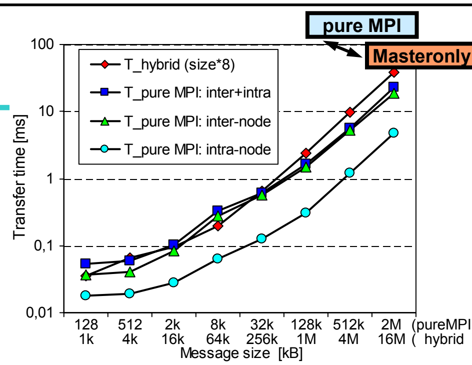
Hitachi SR8000
- 8 nodes
- each node with 8 CPUs
- MPI_Sendrecv

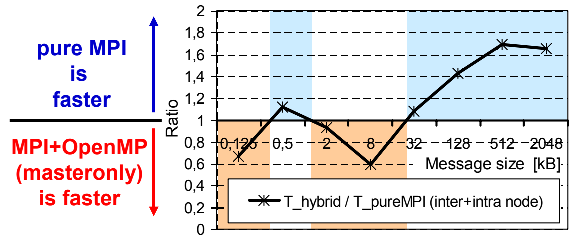


→ **1.6x slower** than with pure MPI, **although**
- only half of the transferred bytes
- and less latencies due to 8x longer messages

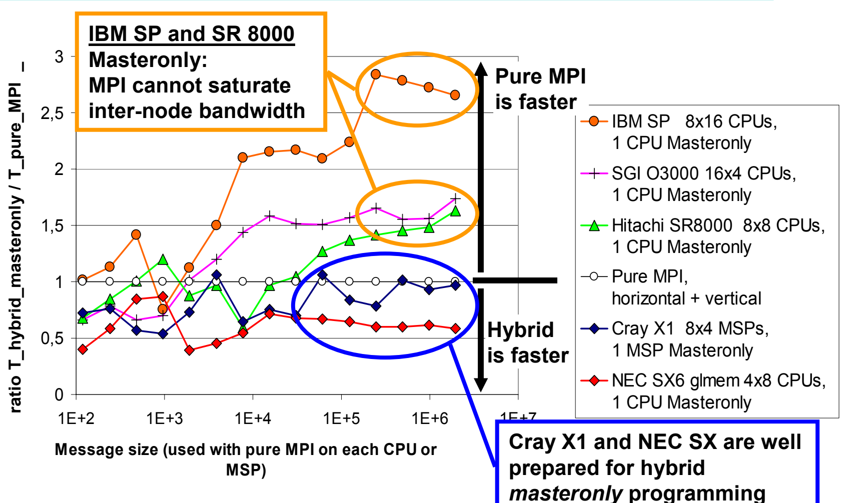
Results of the experiment

- pure MPI 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



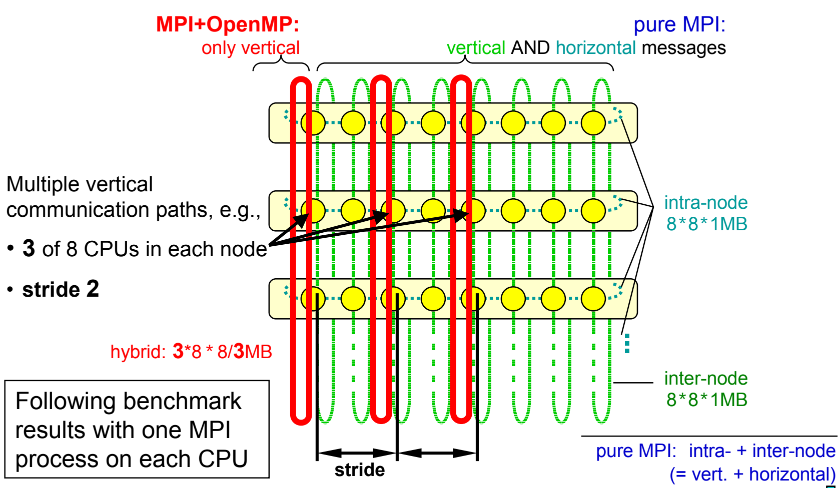
Cray X1 and SGI results are preliminary

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

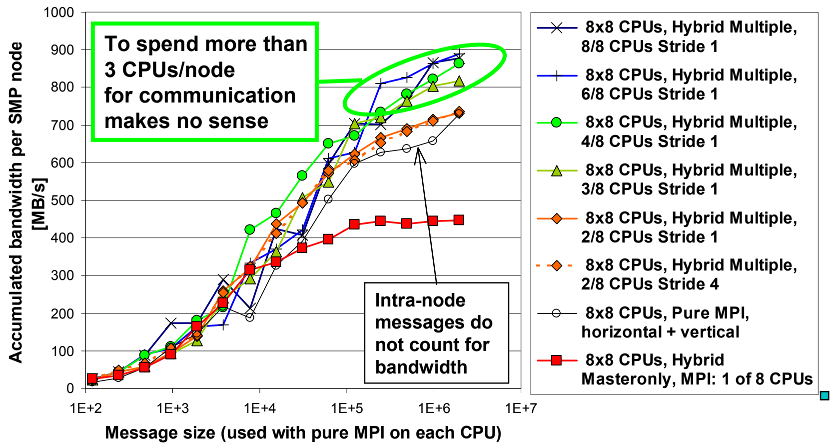
Multiple inter-node communication paths



Following benchmark results with one MPI process on each CPU

Multiple inter-node communication paths: Hitachi SR8000

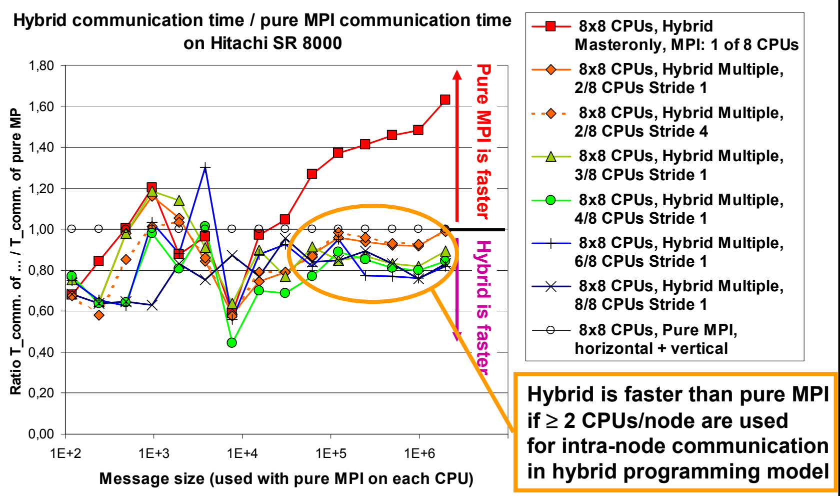
Inter-node bandwidth per SMP node, accumulated over its CPUs, *)
on Hitachi SR8K



H L R S

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

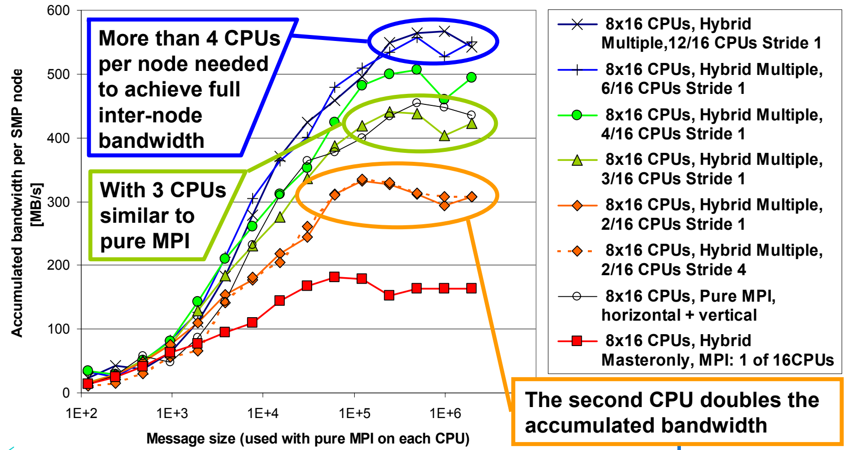
Multiple inter-node communication paths: Hitachi SR 8000



H L R S

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)



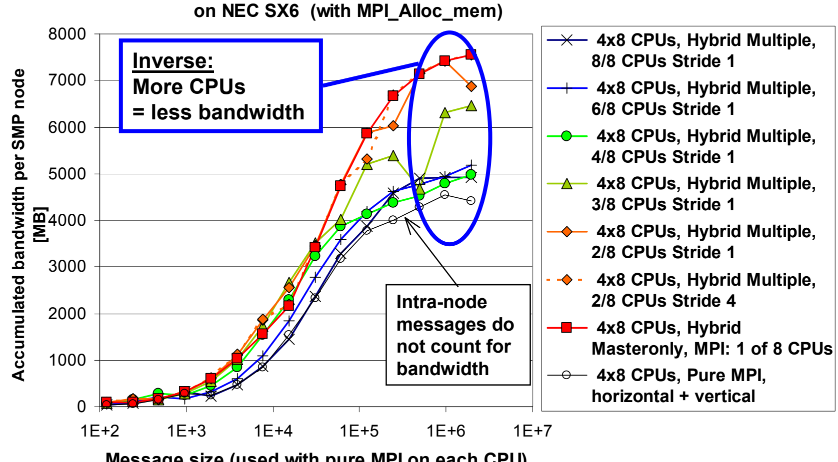
Hybrid Parallel Programming Rolf Rabenseifner
Slide 23 / 61 High Perf. Comp. Center, Univ. Stuttgart

*) 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)

Inter-node bandwidth per SMP node, accumulated over its CPUs, *)
on NEC SX6 (with MPI_Alloc_mem)



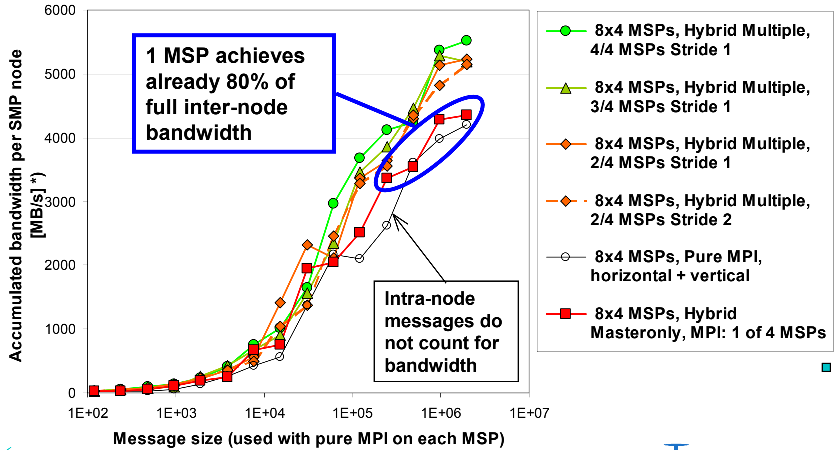
Hybrid Parallel Programming Rolf Rabenseifner
Slide 24 / 61 High Perf. Comp. Center, Univ. Stuttgart

*) 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)



Hybrid Parallel Programming Rolf Rabenseifner
Slide 25 / 61 High Perf. Comp. Center, Univ. Stuttgart

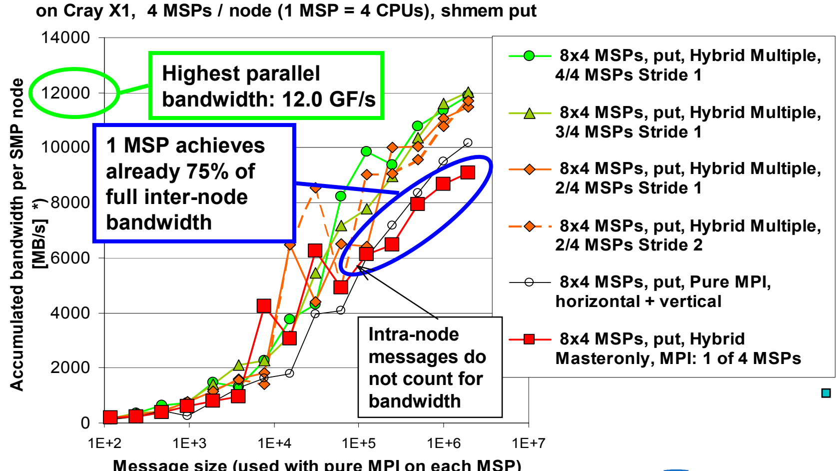
H L R | S

*) 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.

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



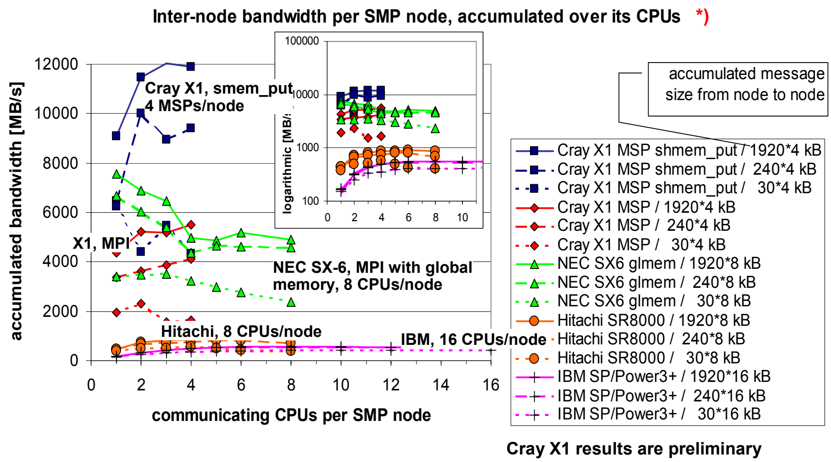
Hybrid Parallel Programming Rolf Rabenseifner
Slide 26 / 61 High Perf. Comp. Center, Univ. Stuttgart

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

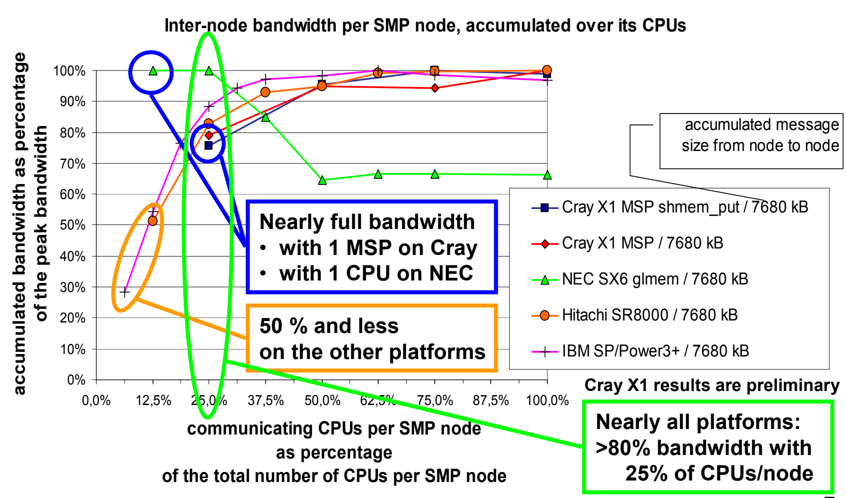


Cray X1 results are preliminary

Hybrid Parallel Programming — Rolf Rabenseifner
Slide 27 / 61 — High Perf. Comp. Center, Univ. Stuttgart

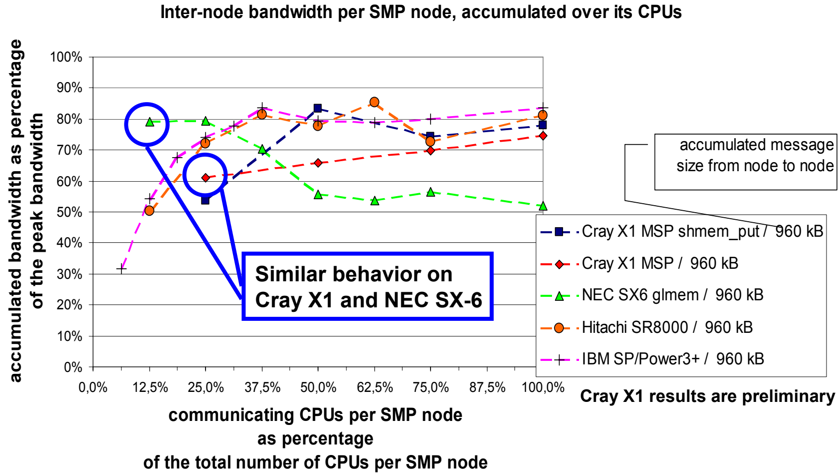
*) Bandwidth per node: totally transferred bytes on the inter-node network / wall clock time / number of nodes

Comparison (as percentage of maximal bandwidth and #CPUs)

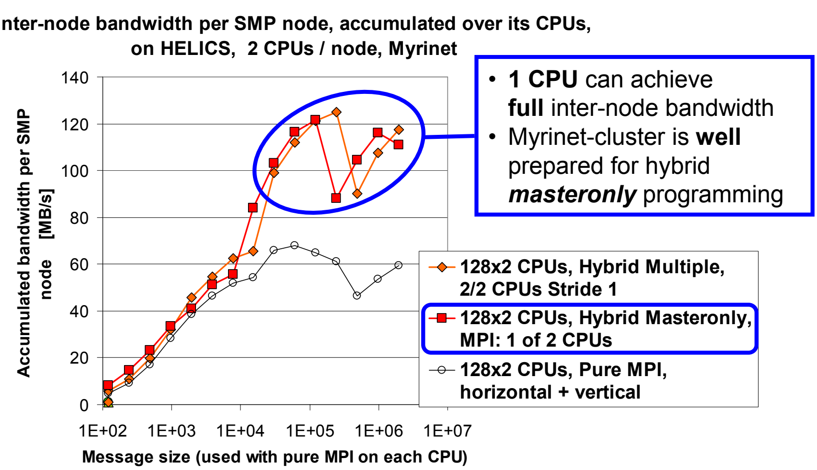


Hybrid Parallel Programming — Rolf Rabenseifner
Slide 28 / 61 — High Perf. Comp. Center, Univ. Stuttgart

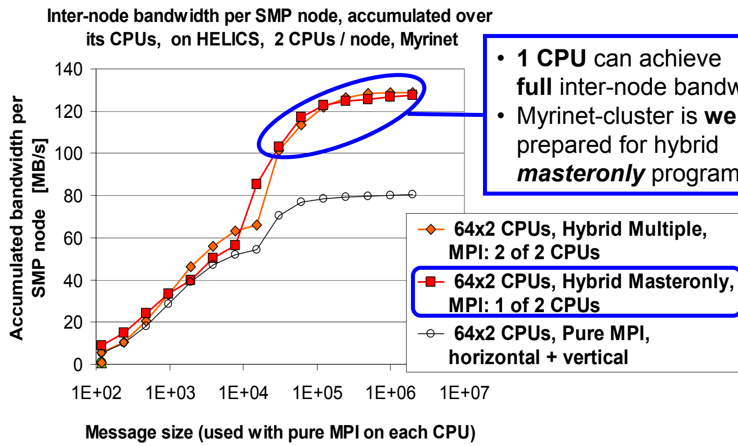
Comparison (only 960 kB aggregated message size)



Myrinet Cluster



Myrinet Cluster (only 64 nodes)



- **1 CPU** can achieve **full** inter-node bandwidth
- Myrinet-cluster is **well** prepared for hybrid ***masteronly*** programming

Hybrid Programming on Cray X1: MSP based usage

- pure MPI or hybrid masteronly MPI+OpenMP

  → same communication time
- 1 MSP already achieves 80% of maximum bandwidth (**contiguous data**)
  - **Are CPU-intensive MPI routines (Reduce, strided data) efficient & multi-threaded ?**
- Hybrid programming → 4 layers of parallelism

– MPI between nodes	(e.g. domain decomposition)
– OpenMP between MSPs	(e.g. outer loops)
– Automatic parallelization	(e.g. inner loops)
– Vectorization	(e.g. most inner loops)

  → risk of Amdahl's law on each level!
- Hybrid & overlapping communication and computation
  - **horrible programming interface (but standardized)**
  - **but chance to use sleeping MSPs while master MSP communicates**

Hybrid Programming on Cray X1: SSP based

- Communication is hardware-bound to SSP
  - 1 SSP can get only 1/4 of 1 MSP's inter-node bandwidth
  - with shmem put:
    all SSPs of a node can together achieve full inter-node bandwidth
    (12.3 GB/s of 12.8 GB/s hardware specification)
- Hybrid MPI+OpenMP, masteronly style
  - optimized MPI library needed with same bandwidth as on 1 or 4 MSP
  - e.g., internally thread-parallel
- Multiple communicating user-threads are not supported
- pure MPI
  - efficient MPI implementation under development

Comparing inter-node bandwidth with CPU performance

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

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 band-width [GB/s]	**Peak** & *Linpack* performance Gflop/s	max.inter-node bw / peak & *Linpack* perf. B/Flop	nodes*CPUs
Cray X1, shmem_put preliminary results	9.27	**12.34**	**75 %**	33.0	136	**51.2** *45.03*	**0.241** *0.274*	**8 * 4 MSPs**
Cray X1, MPI preliminary results	4.52	**5.52**	**82 %**	19.5	136	**51.2** *45.03*	**0.108** *0.123*	**8 * 4 MSPs**
NEC SX-6 global memory	**7.56**	4.98	**100 %**	78.7 93.7^{+})	256	**64** *61.83*	**0.118** *0.122*	**4 * 8 CPUs**
NEC SX-5Be local memory	2.27	**2.50 a)**	**91 %**	35.1	512	**64** *60.50*	**0.039** *0.041*	**2 *16 CPUs** a) only with 8
Hitachi SR8000	0.45	**0.91**	**49 %**	5.0	32 store 32 load	**8** *6.82*	**0.114** *0.133*	**8 * 8 CPUs**
IBM SP Power3+	0.16	**0.57^{+})**	**28 %**	2.0	16	**24** *14.27*	**0.023** *0.040*	**8 *16 CPUs**
SGI O3000, 600MHz	0.43^{+})	**1.74^{+})**	**25 %**	1.73^{+})		**4.8** *3.64*	**0.363** *0.478*	**16 *4 CPUs**
SUN-fire (prelimi.)	0.15	**0.85**	**18 %**	1.68				**4 *24 CPUs**
HELICS Dual-PC cluster with Myrinet	0.118 $^{+}$)	**0.119 $^{+}$)**	**100 %**	0.104 $^{+}$)		**2.80** *1.61*	**0.043** *0.074*	**128 *2 CPUs**

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 [with masteronly]
  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**

Overlapping Communication and Computation

MPI communication by one or a few threads while other threads are computing

- the application problem:
  – one must separate application into:
    - **code that can run before the halo data is received**
    - **code that needs halo data**

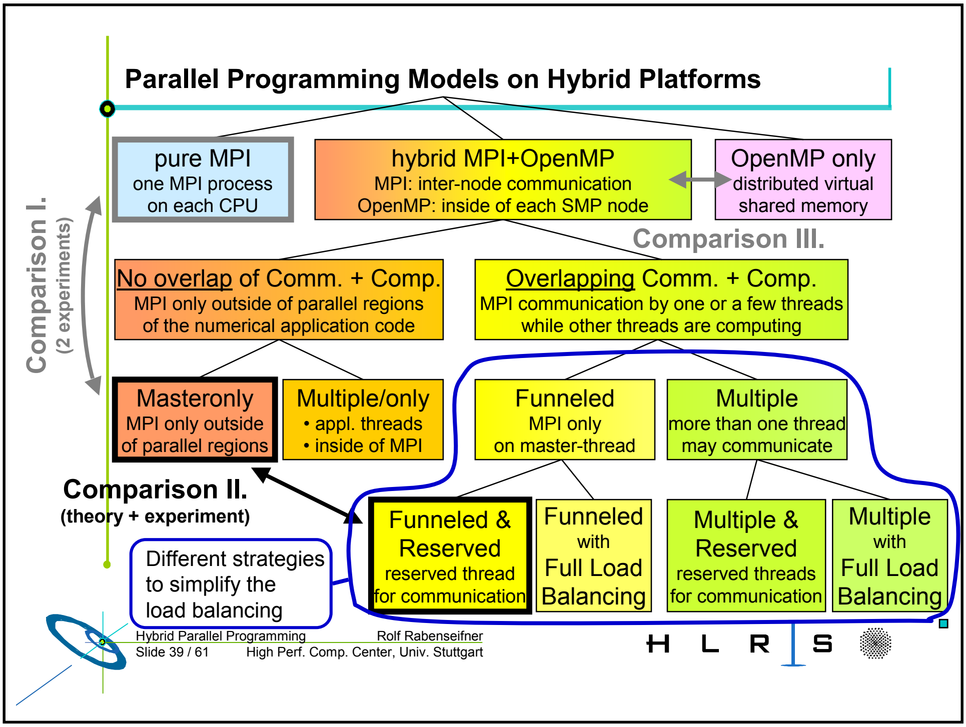
  ➔ very hard to do !!!

- the thread-rank problem:
  – comm. / comp. via thread-rank
  – cannot use work-sharing directives

  ➔ loss of major OpenMP support

- the load balancing problem

```
if (my_thread_rank < 1) {
   MPI_Send/Recv....
} else {
   my_range = (high-low-1) / (num_threads-1) + 1;
   my_low = low + (my_thread_rank+1)*my_range;
   my_high=high+ (my_thread_rank+1+1)*my_range;
   my_high = max(high, my_high)
   for (i=my_low; i<my_high; i++) {
          ....
   }
}
```



Overlapping communication and computation (cont'd)

- 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 !!!

Hybrid Parallel Programming Rolf Rabenseifner
Slide 40 / 61 High Perf. Comp. Center, Univ. Stuttgart
HLRS

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

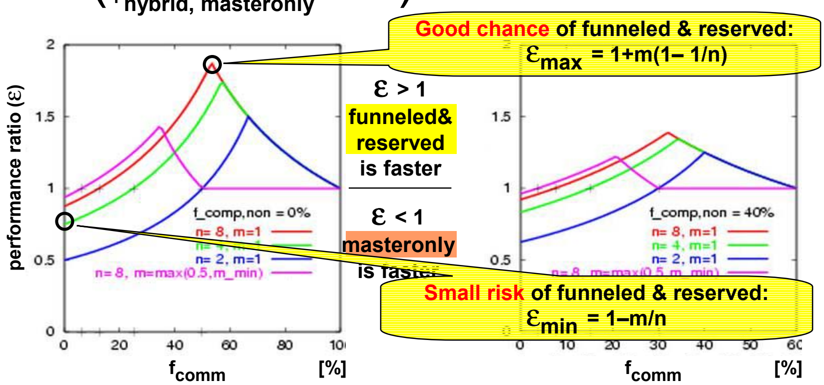
$$\frac{T_{\mathbf{communication}}}{T_{\mathbf{computation}}} \neq \frac{n_{\mathbf{communication\_threads}}}{n_{\mathbf{computation\_threads}}}$$

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

Performance ratio (theory)

Masteronly ↔ **funneled & reserved**

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

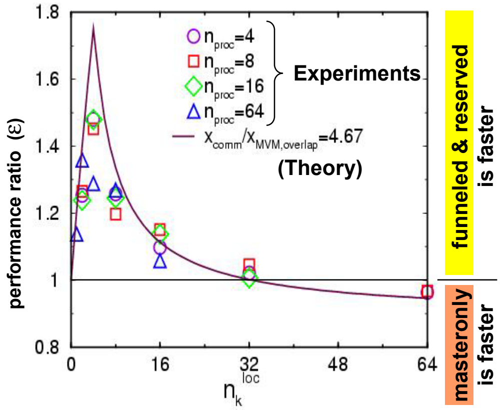


$T_{\text{hybrid, masteronly}} = (f_{\text{comm}} + f_{\text{comp, non-overlap}} + f_{\text{comp, overlap}})\ T_{\text{hybrid, masteronly}}$

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

Masteronly ↔ funneled & reserved

Experiment: Matrix-vector-multiply (MVM)

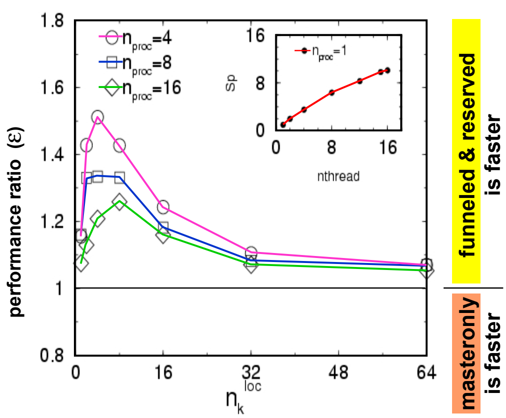


- Jacobi-Davidson-Solver
- Hitachi SR8000
- 8 CPUs / SMP node
- JDS (Jagged Diagonal Storage)
- vectorizing
- n_{proc} = # SMP nodes
- D_{Mat} = $512*512*(n_k^{loc}*n_{proc})$
- Varying n_k^{loc}
  $\Rightarrow$ Varying $1/f_{comm}$
- $\frac{f_{comp,non-overlap}}{f_{comp,overlap}} = \frac{1}{6}$

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

Masteronly ↔ funneled & reserved

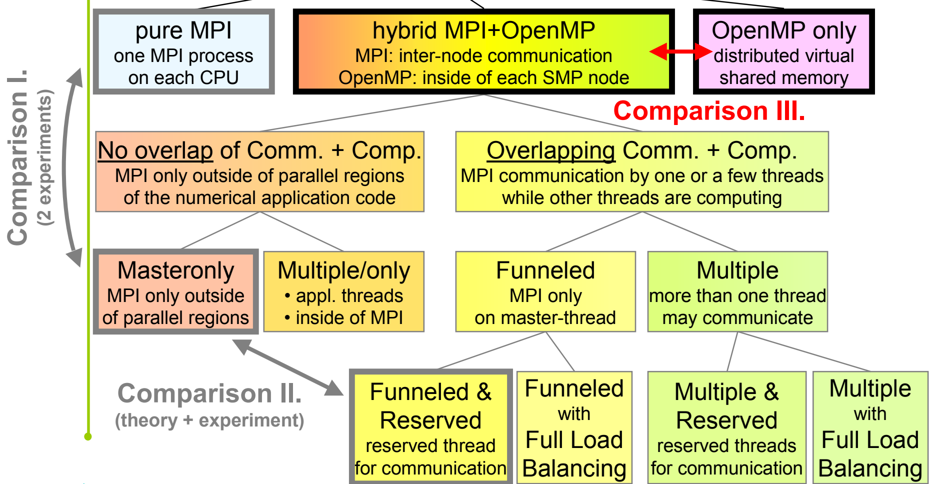
Experiment: Matrix-vector-multiply (MVM)



- 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

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?**

OpenMP Source (Fortran / C)
with optimization directives
↓
(1) OMNI Compiler
↓
C-Code + Library calls
↓
(2) optimizing native compiler
↓
Executable ← **Communication- & Thread-Library**

- **Optimization of the computation** **<u>more important than</u>** **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

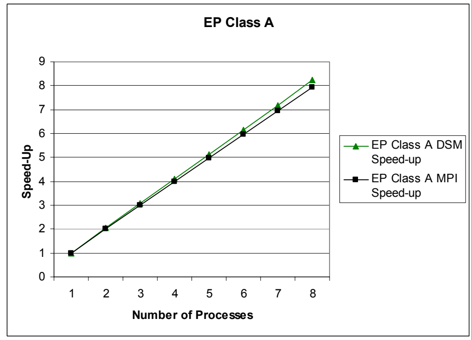
- NAS Parallel Benchmarks EP, FT, and CG:
  - Message passing and sequential version
- Automatically generate OpenMP directives for sequential code using CAPO (www.nas.nasa.gov/Groups/Tools/CAPO)
- Omni Compiler
- Compare speedup of:
  - Message passing vs. OpenMP/DSM
  - OpenMP/DSM vs. OpenMP/SMP
-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



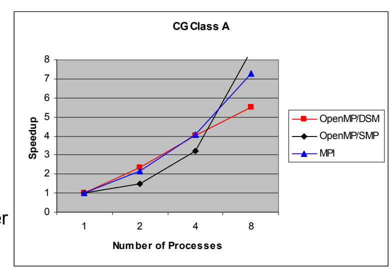
Linear speedup for MPI and OpenMP/DSM. No surprises.

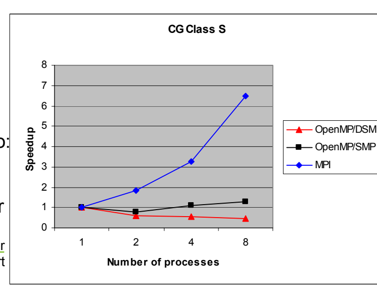
Slides courtesy of Gabriele Jost, NASA/AMES, and Matthias Müller, HLRS

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

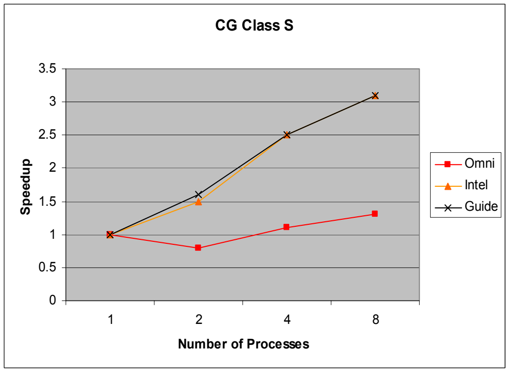




Slides courtesy of Gabriele Jost, NASA/AMES, and Matthias Müller, HLRS

pure MPI ↔ OpenMP only

CG Benchmark Results (2)



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

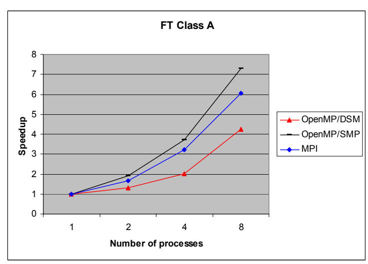
---

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

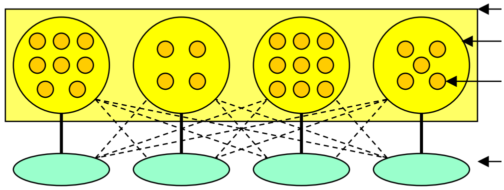
No standards!
Only on a few platforms!

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)

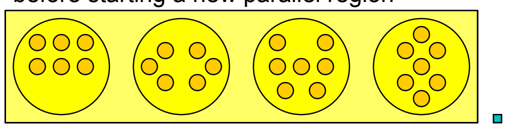


- program
- processes
- multiple threads inside of each process (**OpenMP**)
- data associated with each process
- **but shared (ccNUMA) access to other processes' data**

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
  – Reiner Vogelsang, SGI
  – Gabriele Jost, NASA
  – Dieter an Mey, RZ Aachen
  – Horst Simon, NERSC
  – Matthias Müller, HLRS
  – my colleges at HLRS

Conclusions

- **Only a few platforms**
  - e.g., Cray X1 in MSP mode, NEC SX-6, and Myrinet-cluster
  - 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
    - **single threaded, vectorized, low-priority, small-medium memory needs**

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