

# Optimization of MPI Applications

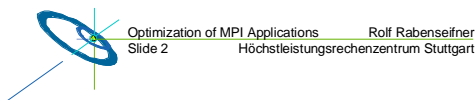
Rolf Rabenseifner

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



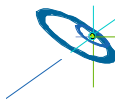
## Optimization and Standardization

- Issues
  - one programming problem has different solutions with MPI
  - which is the best solution?
- General rule:
  - MPI targets portable and efficient message-passing programming but  
**efficiency of MPI application-programming is not portable!**
- ==> Most of the following slides need not to change in future, but may change in future!



## Outline

- Communication = Overhead
  - transfer time
  - synchronization time = idle time
- Programming problems
  - deadlocks
  - buffer contention
- Other performance problems
  - collective routines
  - MPI-I/O bandwidth
  - recompute or communicate
  - cluster of SMPs
  - configuration
  - profiling / statistics
- Summary
- Practical

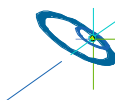


Optimization of MPI Applications Rolf Rabenseifner  
Slide 3 Höchstleistungsrechenzentrum Stuttgart

H L R I S

## Communication = Overhead

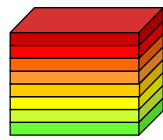
- Simplest model:  
$$\text{Transfer time} = \text{latency} + \text{message length} / \text{bandwidth}$$
- Latency: Startup for message handling
- Bandwidth: Transfer of bytes
- n messages:  
$$\text{Transfer time} = n * \text{latency} + \text{total message length} / \text{bandwidth}$$
  - **Send one big message instead of several small messages!**
  - **Reduce the total amount of bytes!**
  - **Bandwidth depends on protocol**



Optimization of MPI Applications Rolf Rabenseifner  
Slide 4 Höchstleistungsrechenzentrum Stuttgart

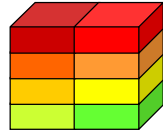
H L R I S

## Communication = Overhead — Decomposition

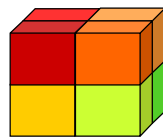


### Splitting in

- **one** dimension:  
communication  
 $= n^2 * 2 * w * 1$



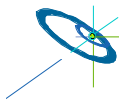
- **two** dimensions:  
communication  
 $= n^2 * 2 * w * 2 / p^{1/2}$



- **three** dimensions:  
communication  
 $= n^2 * 2 * w * 3 / p^{2/3}$

$w$  = width of halo  
 $n^3$  = size of matrix  
 $p$  = number of processors  
 cyclic boundary  
 → **two** neighbors  
 in each direction

optimal for  $p > 11$



Optimization of MPI Applications Rolf Rabenseifner  
 Slide 5 Höchstleistungsrechenzentrum Stuttgart

HLRS

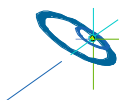
[You can find a copy of this slide in the Chapter „A Heat-Transfer Example with MPI“, slide 10]

## Communication = Overhead — Different protocols, I.

- Internal protocols for standard MPI\_Send
  - short protocol: envelope + message data:  
buffered in pre-allocated slot at receiver
  - eager protocol: message envelope: buffered at receiver,  
message data: buffered in temporarily  
allocated buffer at receiver or sender
  - rendezvous protocol: sender blocked until destination calls  
receiving routine, no buffering

Decision based on message size.

==> **Should be configured appropriately, if necessary.**

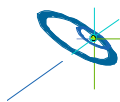


Optimization of MPI Applications Rolf Rabenseifner  
 Slide 6 Höchstleistungsrechenzentrum Stuttgart

HLRS

## Communication = Overhead — Different protocols, II.

- Latency: **short protocol** < eager protocol < rendezvous protocol
- Bandwidth: short protocol  $\approx$  eager protocol < **rendezvous protocol (best values)**
- Bandwidth for one message  
 $\approx$  network bandwidth / number of parallel messages  
on same hardware connection
- Benchmarks, e.g., [www.hlr.de/mpi/b\\_eff/](http://www.hlr.de/mpi/b_eff/)

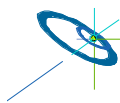


Optimization of MPI Applications Rolf Rabenseifner  
Slide 7 Höchstleistungsrechenzentrum Stuttgart

H L R I S

## Communication = Overhead — Send routines

- Send / Bsend / Ssend / Rsend – which is the best?
  - **Send**
    - internally chooses **best protocol**  
==> may be synchronous ==> slide about serialization, see later  
==> slide about deadlocks, see later
  - **Ssend**
    - should be used only if internal rendezvous (barrier) synchronization is necessary!
  - **Bsend** (buffered send)
    - to reduce synchronization time
    - to avoid deadlocks
    - but not scalable with message length  
==> Choose lsend+Wait or Sendrecv
  - **Rsend** (ready send)
    - use never, except you have a 200% guarantee that Recv is already called in the current version and all future versions of your code

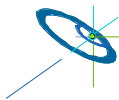


Optimization of MPI Applications Rolf Rabenseifner  
Slide 8 Höchstleistungsrechenzentrum Stuttgart

H L R I S

Communication = Overhead — **non-blocking comm.**

- Non-blocking
  - latency hiding / overlap of communication and computation,
    - Problem: most MPI implementations communicate only while MPI routines are called
    - Exception: Metacomputing libraries
    - ==> **Do not spent too much effort in such overlap**
  - used to avoid deadlocks (see later)
  - used to avoid waiting until sender **and** receiver are ready to communicate, i.e., to avoid idle time (see later)



Communication = Overhead  
**Comparing latencies with “heat” application**

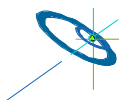
Communication-time	T3E 900	Hitachi SR2201	HP-V HP-V
(number of PEs)	(16)	(16)	(8)
MPI non-blocking	3.4	5.3	2 [sec]
MPI_SENDRECV	1.9	5.8	2 [sec]
MPI_ALLTOALLV	0.8 *)	15.4	2 [sec]
Computation-Time	0.65	1.9	2 [sec]

MPI targets portable and efficient message-passing programming  
but

**efficiency of MPI application-programming is not portable!**

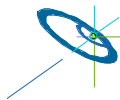
\*) up 128 PEs:  
**ALLTOALLV  
is better than  
SENDRECV**

(measured April 29, 1999, with heat-mpi1-big.f and stride 179,  
CRAY T3E: sn6715 hwwt3e.hww.de 2.0.4.48 unicosmk CRAY T3E mpt.1.3.0.0.6,  
Hitachi: HI-UX/MPP hitachi.rus.uni-stuttgart.de 02-03 0 SR2201,  
HP: HP-UX hp-v.hww.de B.11.00 A 9000/800 75859)



## Communication = Overhead — Strided Data, I.

- Theory about transfer of strided data:
    - give all information to MPI, and MPI will optimize your transfer
  - Experience:
    - many MPI implementations do not optimize transfers of strided data
  - Different solutions:
    - MPI\_Type\_vector (MPI-1), MPI\_Type\_create\_subarray (MPI-2)
      - ==> MPI library may internally copy the data to a scratch buffer, and the copy operation may not be optimized!
    - copy strided data into a scratch array and transfer the scratch array and vice versa
      - ==> Compiler can optimize the copy operation, but always an additional scratch array is used.
- And: May solve the “corner problem” (MPI-1, page 40, lines 44-45: a memory location must not be transferred in parallel by several Isends)
- Rule: **If in the time critical path, then implement both and compare!**

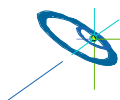


Optimization of MPI Applications Rolf Rabenseifner  
Slide 11 Höchstleistungsrechenzentrum Stuttgart

H L R I S

## Communication = Overhead — Strided Data, II.

- **On a cluster of SMPs:**
    - MPI\_Type\_vector (MPI-1), MPI\_Type\_create\_subarray (MPI-2)
      - ==> MPI library may internally copy the data to a scratch buffer, and the copy operation may not be optimized!
- May be very slow if application is multi-threaded and MPI is single threaded**
- copy strided data into a scratch array and transfer the scratch array and vice versa
    - ==> Compiler can optimize the copy operation, but always an additional scratch array is used.
- May be automatically parallelized (multithreaded) and vectorized!**
- Rule: **If in the time critical path, then implement both and compare!**



Optimization of MPI Applications Rolf Rabenseifner  
Slide 12 Höchstleistungsrechenzentrum Stuttgart

H L R I S

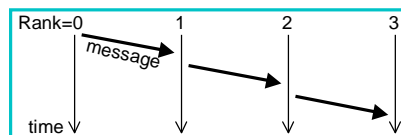
## Synchronization time = idle time

- Transfer time = latency + message length / bandwidth + sync.time
- Synchronization time:
  - receiver waits until message is sent
  - sender waits until receive is posted
  - how to avoid serialization
  - how to avoid idle time
  - methods:
    - non-blocking routines can avoid waiting on communication routines
    - but waiting for freeing the request (and buffers!)
    - three internal protocols

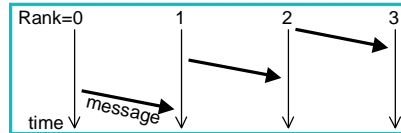
## Synchronization time — How to avoid serialization

- Synchronization may cause serialization:

**MPI\_Recv(left\_neighbor)  
MPI\_Send(right\_neighbor)**



**MPI\_Send(right\_neighbor)  
MPI\_Recv(left\_neighbor)**

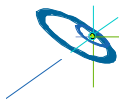


- Solutions:
  - MPI\_I..... (non-blocking routines)
  - MPI\_Bsend
  - MPI\_Sendrecv

## Synchronization time — Non-blocking communication

How to avoid synchronization time:

- **receiver waits until message is sent**
  - no MPI tricks available
- **sender waits until receive is posted**
  - non-blocking routines can avoid waiting on communication routines
  - but waiting for freeing the request (and buffers!)
  - therefore double buffering may be needed
  - three internal protocols, also in combination with non-blocking comm.

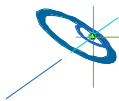


Optimization of MPI Applications Rolf Rabenseifner  
Slide 15 Höchstleistungsrechenzentrum Stuttgart

H L R I S 

## Synchronization time — Non-blocking communication

- Non-blocking
  - latency hiding / overlap of communication and computation,
    - Problem: most MPI implementations communicate only while MPI routines are called
    - Exception: Metacomputing libraries
    - ==> Do not spent too much effort in such overlap
  - used to avoid deadlocks (see later)
  - used to avoid waiting until sender **and** receiver are ready to communicate, i.e., **to avoid idle time**



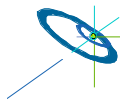
Optimization of MPI Applications Rolf Rabenseifner  
Slide 16 Höchstleistungsrechenzentrum Stuttgart

H L R I S 



## Outline

- Communication = Overhead
  - transfer time
  - synchronization time = idle time
- **Programming problems**
  - **deadlocks**
  - **buffer contention**
- Other performance problems
  - collective routines
  - MPI-I/O bandwidth
  - recompute or communicate
  - cluster of SMPs
  - configuration
  - profiling / statistics
- Summary
- Practical

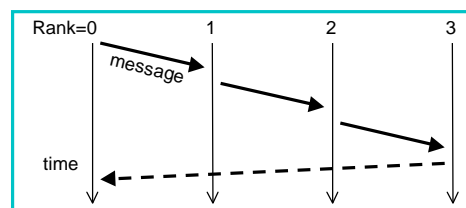


Optimization of MPI Applications Rolf Rabenseifner  
Slide 17 Höchstleistungsrechenzentrum Stuttgart

HLRIS

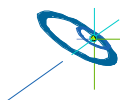
## Point-to-point: Avoiding Deadlocks

- Several strategies to avoid deadlocks:
  - Reordering of the messages



==> Normally implies a serial execution ==> **worst performance**

- Without last message: No deadlock cycle, **but same performance problem**
- Bsend – not scalable for large message sizes

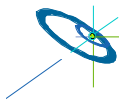


Optimization of MPI Applications Rolf Rabenseifner  
Slide 18 Höchstleistungsrechenzentrum Stuttgart

HLRIS

## Point-to-point: Avoiding Deadlocks (continued)

- Several strategies to avoid deadlocks: (continued)
  - Using non-blocking routines
    - Irecv + Send + Wait(all)
      - ==> the application specifies a fixed sequence of send operations
      - ==> **may lead to** network contention and **delays in some processes**
      - ==> but the slowest process may determine the whole system
    - Isend + Recv + Wait(all) ==> same problems
    - Irecv + Isend + Waitall
      - ==> should be the **best solution with non-blocking routines**
  - MPI\_Sendrecv
    - ==> **best solution on most platforms for regular communication patterns!**
  - MPI\_Alltoallv – the collective alternative
    - ==> does not scale for large number of processes, high latency on most platforms
  - MPI-2: Use One-sided communication



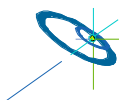
Optimization of MPI Applications Rolf Rabenseifner  
Slide 19 Höchstleistungsrechenzentrum Stuttgart

H L R I S 

## Buffer contention

- Contention of buffer or message slots:

```
do i=1,1000
  if (rank /= 0) MPI_Send(1 byte or 2 kb or 10 Mb)
  else receive the message from each process
enddo
```
- Solutions:
  - use Gather/Gatherv if the receiver knows the message sizes
  - use MPI\_**S**send
    - **does not prohibit overflow of envelope queue**
  - configure a large message queue
  - serialization
    - **token circles around the sending processes & MPI\_Ssend**
    - **token sent by receiver**
    - **token circled with MPI\_Barrier, called for each sending process (worst solution)**

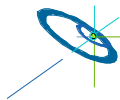


Optimization of MPI Applications Rolf Rabenseifner  
Slide 20 Höchstleistungsrechenzentrum Stuttgart

H L R I S 

## Outline

- Communication = Overhead
  - transfer time
  - synchronization time = idle time
- Programming problems
  - deadlocks
  - buffer contention
- **Other performance problems**
  - **collective routines**
  - **MPI-I/O bandwidth**
  - **recompute or communicate**
  - **cluster of SMPs**
  - **configuration**
  - **profiling / statistics**
- Summary
- Practical

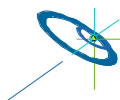


Optimization of MPI Applications Rolf Rabenseifner  
Slide 21 Höchstleistungsrechenzentrum Stuttgart

H L R I S 

## Collective operations

- Should be optimized by vendor of MPI library
- Example: Bcast
  - Tree algorithms on distributed memory platforms
    - **binary tree ==> load balanced, pipelined execution of a sequence of bcasts, but total execution time is not optimal**
    - **unbalanced tree ==> minimal total execution time**
  - Parallel execution on all processes
    - **on shared memory architectures or with hardware broadcast**
- Rules:
  - **Always use collective operations, if fitting to your application's needs**
  - Avoid all-to-all communication
  - **Never use MPI\_Barrier**, except for debugging without debugger

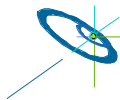


Optimization of MPI Applications Rolf Rabenseifner  
Slide 22 Höchstleistungsrechenzentrum Stuttgart

H L R I S 

## MPI – I/O

- Best throughput with
  - large size of data,
  - accessed with one (collective) MPI\_IO call
  - optimization is extremely platform-**dependent**
- Benchmark results, see
  - [www.hlrs.de/mpi/b\\_eff\\_io/](http://www.hlrs.de/mpi/b_eff_io/)
  - further benchmarks are evaluated on [www.top500clusters.org](http://www.top500clusters.org)

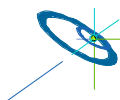


Optimization of MPI Applications Rolf Rabenseifner  
Slide 23 Höchstleistungsrechenzentrum Stuttgart

H L R I S 

## Recomputation versus communication

- optimization, if same data can be computed on several / all processes
  - parallel equivalent computation
  - single computation + broadcast  
while other processes can do other work
  - single computation + broadcast  
while other processes idle (worst solution!)

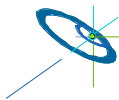


Optimization of MPI Applications Rolf Rabenseifner  
Slide 24 Höchstleistungsrechenzentrum Stuttgart

H L R I S 

## Clusters of SMP nodes

- MPI on clusters of shared memory parallel (SMP) nodes
- MPI processes — three solutions:
  - (a) One MPI process on each processor of each node
    - **How are they ranked? (configurable, i.e., on SR8000)**
      - Contiguous on each node
      - round robin over all nodes
  - (–) One MPI process on each node
    - (b) automatically parallelized by the compiler on all processors of a node
    - (c) parallelization on each SMP node with OpenMP
      - call MPI only from OpenMP root thread!
      - cache coherence must be guaranteed by OpenMP programming → OMP FLUSH directive

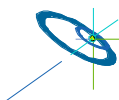


Optimization of MPI Applications Rolf Rabenseifner  
Slide 25 Höchstleistungsrechenzentrum Stuttgart



## Configuring MPI

- Essential on some MPI implementations
- via linkage, program start (mpirun/mpiexec options), or environment variables (analyzed at runtime)
- examples: **(default may not be the fastest/cheapest solution!)**
  - enable/disable internal error checking
  - enable counter profiling
  - enable user profiling PMPPI interface (`-lpmppi`)
  - T3E: `MPI_BUFFER_MAX`
    - = maximal message size for eager protocol
    - (default is infinite buffering with eager protocol!!!)
  - SR8000: ranking of MPI processes, if 8 processes per node  
(default is currently round-robin ranking)
  - SX-4/5: `MPI_SUSPEND=ON`
    - switches from spin-wait to suspend/resume
  - MPI-I/O: which filesystem interface (T3E, Fujitsu, IBM)

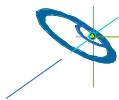


Optimization of MPI Applications Rolf Rabenseifner  
Slide 26 Höchstleistungsrechenzentrum Stuttgart



## Statistics / Profiling

- Measured between MPI\_Init and MPI\_Finalize
- Counter-profiling examples
  - T3E: module switch mpt mpt.1.2.1.2.p  
setenv MPIPROFOUT stdout
  - SX-4/5: setenv MPIPROGINF YES | DETAIL | ALL | ALL\_DETAIL
  - hp: mpirun -i *profiling\_prefix* -np *size* *program*  
ASCII: view *profiling\_prefix.instr*  
Graphical: mpiview *profiling\_prefix.mpiview*
- Trace-based profiling
  - see VAMPIR



Optimization of MPI Applications Rolf Rabenseifner  
Slide 27 Höchstleistungsrechenzentrum Stuttgart

H L R I S

## Optimization / Summary

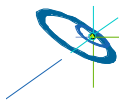
### We discussed

transfer time, protocols, latency & bandwidth, B/S/Rsend, non-blocking, strided data,  
synchronization time = idle time, serialization, non-blocking routines,  
deadlocks, buffer contention,  
collective routines, MPI-I/O bandwidth, recompute or communicate,  
cluster of SMPs, configuration, profiling & statistics.

### Never forget

MPI targets portable and efficient message-passing programming  
but

**efficiency of MPI application-programming is not portable!**

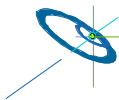


Optimization of MPI Applications Rolf Rabenseifner  
Slide 28 Höchstleistungsrechenzentrum Stuttgart

H L R I S

## Optimization Practical

- `cp ~/MPI/course/F/Ch7/ring.f .`  
`C      ring.c .`
- make two-dimensional topology
  - splitting “size” with `MPI_DIMS_CREATE()`
  - cyclic in the first dimension
  - linear in the second dimension
- compute and print the sum of the original cartesian ranks separately in each ring
- please, discuss and implement the best choice for large scale systems, i.e., expect that the computation is repeated very often and hundreds of processors are used
- your trainer will come to look at your decisions

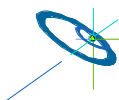


Optimization of MPI Applications    Rolf Rabenseifner  
Slide 29                                    Höchstleistungsrechenzentrum Stuttgart

H L R I S 

## Optimization Practical — Background, I.

- `MPI_Dims_create`:
  - `int MPI_Dims_create(int nnodes, int ndims, int *dims);`
  - SUBR. `MPI_DIMS_CREATE(nnodes, ndims, dims, ierror)`  
`INTEGER nnodes, ndims, dims(ndims), ierror`
  - `ndims` := number of dimensions in `dims`, e.g., := 2
  - `dims(...)` must be initialized with zero, e.g., (0,0)
  - `nnodes` := size of `MPI_COMM_WORLD`, e.g., := 12
  - result: `dims` contains a balanced distribution, e.g., (4,3)
- `MPI_Cart_create`:
  - „`dims`“ and „`periods`“ are now arrays!
- Expected results:
  - `size=4` => `dims=(2,2)` => `sums = (2,4)`
  - `size=6` => `dims=(3,2)` => `sums = (6,9)`
  - `size=12` => `dims=(4,3)` => `sums = (18,22,26)`

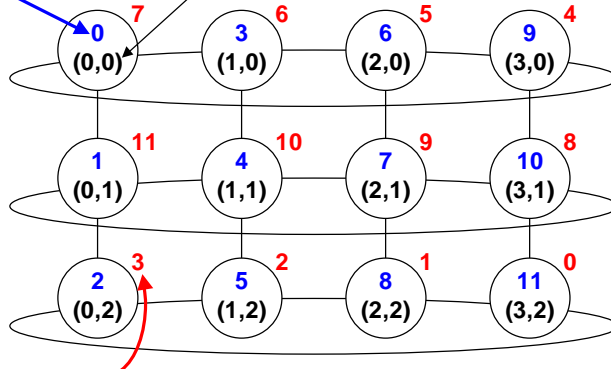


Optimization of MPI Applications    Rolf Rabenseifner  
Slide 30                                    Höchstleistungsrechenzentrum Stuttgart

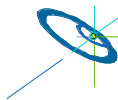
H L R I S 

## Optimization Practical — Background, II.

- Ranks and Cartesian process coordinates in `comm_cart`



- Ranks in `comm` and `comm_cart` may differ, if `reorder = 1` or `.TRUE.`
- This reordering can allow MPI to optimize communications

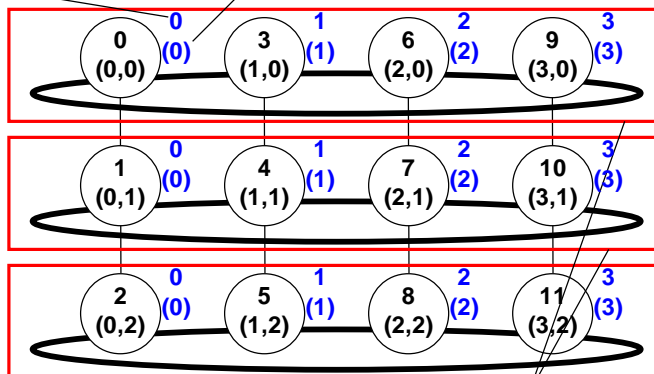


Optimization of MPI Applications Rolf Rabenseifner  
Slide 31 Höchstleistungsrechenzentrum Stuttgart

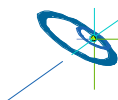
H L R I S

## Optimization Practical — Background, III.

- Ranks and Cartesian process coordinates in `comm_sub`



- `MPI_Cart_sub(comm_cart, remain_dims, comm_sub, ierror)`  
(true,false)



Optimization of MPI Applications Rolf Rabenseifner  
Slide 32 Höchstleistungsrechenzentrum Stuttgart

H L R I S