Optimisation and Benchmarking

Part 3 – MPI Optimisation

28. Nov 2014|

Alan O‘Cais
a.ocais@fz-juelich.de

Live notes:
http://supercomputing.cyi.ac.cy/index.php/live

Minimizing Communication

- **Communication = Overhead**
- Transfer time = latency + message length / bandwidth
  - Latency: Startup for message handling
  - Bandwidth: Transfer of bytes
- For n messages:
  - Transfer time = n * latency + total message length / bandwidth

Send one big message instead of several small messages!

Reduce the total amount of bytes!

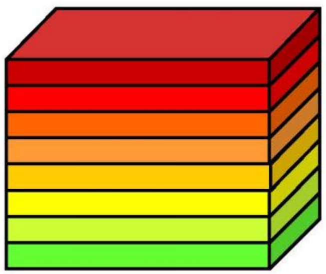
Bandwidth depends on protocol

Courtesy of Rolf Rabenseifner

Minimizing Communication

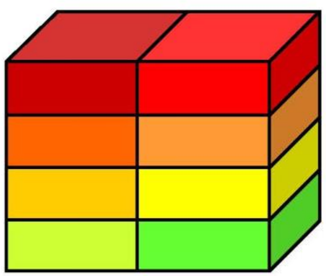
Decomposition:

<u>Splitting in</u>



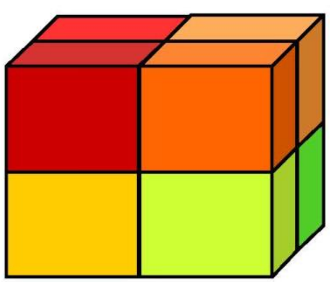
- **one** dimension:

  communication
  $= n^2 * 2 * w * 1$



- **two** dimensions:

  communication
  $= n^2 * 2 * w * 2 / p^{1/2}$



- **three** dimensions:

  communication
  $= n^2 * 2 * w * 3 / p^{2/3}$

optimal for p>11

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

Courtesy of Rolf Rabenseifner

Minimizing Communication

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!)

Courtesy of Rolf Rabenseifner

Minimizing Communication

Clusters of SMP nodes

MPI processes — three solutions:

(a) One MPI process on each processor/HWT of each node

- *How are they ranked?*

(–) One MPI process on each node/socket

(b) automatically parallelized by the compiler on all processors of a node

(c) parallelization on each SMP node/socket with OpenMP

– call MPI only from OpenMP root threads!

– cache coherence must be guaranteed by OpenMP

Courtesy of Rolf Rabenseifner

JÜLICH FORSCHUNGSZENTRUM

Minimizing Impact of Communication

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

Courtesy of Rolf Rabenseifner

Minimizing Impact of Communication

- **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
- Need to avoid serialization
- Need to avoid idle time
- Methods:
  - non-blocking routines can avoid waiting on communication routines
  - but still waiting for freeing the request (and buffers!)

Courtesy of Rolf Rabenseifner

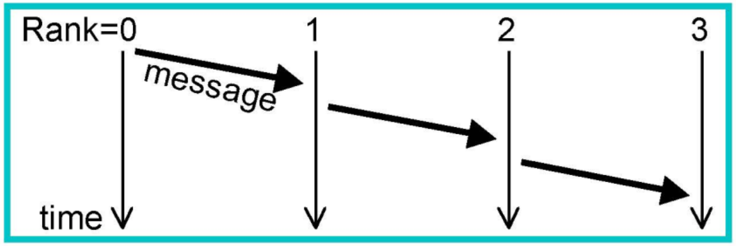
Hands-on: MPI, Pt. 1

- Go to the MPI directory
- Read over `mpi_blocking.c`
- Compile the code with `./compile.sh`, run it with `./run.sh` (use `2048` as the argument)
- Browse the output file

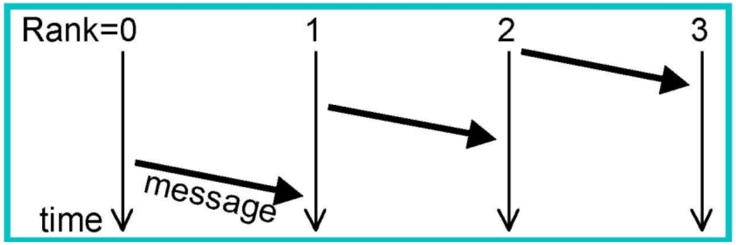
Minimizing Impact of Communication

- 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

Courtesy of Rolf Rabenseifner

Hands-on: MPI, Pt. 2

- Read over `mpi_blocking.c` again
  - Focus on the `Communication` block
- What are the issues with the `Communication` block?
  - Try to solve (one of) them using `MPI_Sendrecv`
  - Use google if you need help!
  - Make a copy of the code and work on the original

Minimizing Impact of Communication

- **Non-blocking communication:**
  - latency hiding / overlap of communication and computation,
    - *Problem: most MPI implementations communicate only while MPI routines are called*

      ==> Do not spend too much effort in such overlap
  - used to avoid deadlocks
  - ***used to avoid waiting until sender and receiver are ready to communicate, i.e., to avoid idle time***

Courtesy of Rolf Rabenseifner

Hands-on: MPI, Pt. 3

- Go to the MPI directory
- Read over `mpi_nonblocking.c`
  - Focus on the `Communication` block
- Compile the code, run it with `run.sh`
- How does the performance compare to `mpi_blocking.c`?
  - Assume the code had more work to do in this case after sending the message, are there any additional advantages?

Special thanks: Rolf Rabenseifner

- Slides are a shortened version of his presentation available at
  - https://fs.hlrs.de/projects/par/par_prog_ws/pdf/mpi_optimize_3.pdf

- Please also see "HLRS Online Parallel Programming Workshop“
  - URL: http://www.hlrs.de/training/par-prog-ws/
  - Course program at
    - http://www.hlrs.de/events/
    - http://www.hlrs.de/training/course-list/

Courtesy of Rolf Rabenseifner