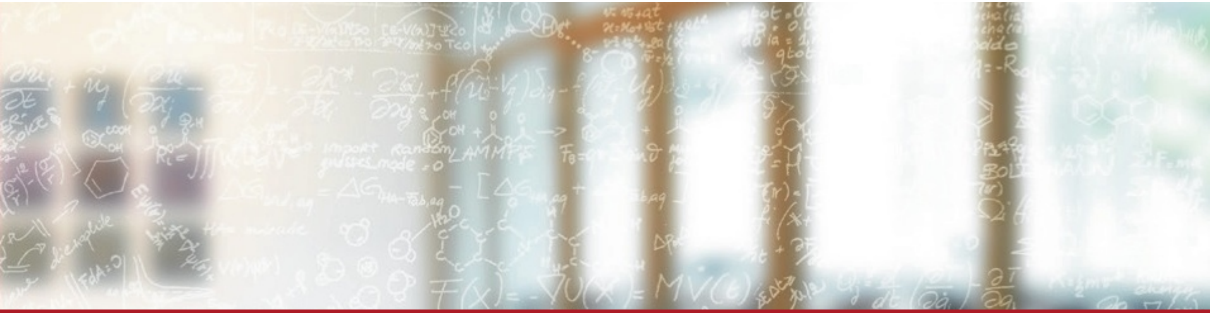
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



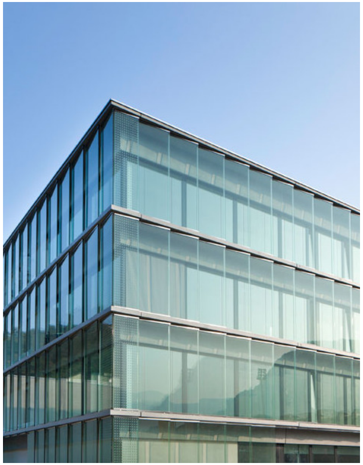
Collective communication for message-passing systems

CSCS User Lab Day

Andreas Jocksch (CSCS)

1th September, 2020

Collective communication for message-passing systems



- Motivation
- The message passing interface (MPI)
- Collective communication: blocking, non-blocking, persistent
- Shared memory on the node
- Our prototype library
- Benchmarks
- Conclusions and outlook

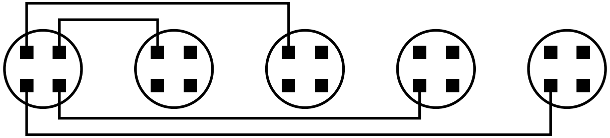
Motivation

- Modern supercomputers equipped with multicore CPUs distributed to many nodes
- Hybridly — OpenMP + MPI — or pure MPI programmed, focus on MPI only
- Collective communication operations provide fast message transfers for certain communication patterns
- Assumption of a fully connected network, described by latency and bandwidth
- For small messages store and forward algorithms, e.g., Bruck’s algorithm
- Persistent but blocking collectives $\rightarrow$ “plan” routines and separate execution routine

Message passing library

- Mostly MPI used, currend version 3.1 plus more recent drafts
- Established implementations MPICH, MVAPICH, OpenMPI
- Supports blocking and non-blocking collective communication, e.g. MPI_Alltoall, MPI_Ialltoall
- Persistent collectives as experimental feature, according to the standard non-blocking

Basic shared memory algorithm (all-to-all)



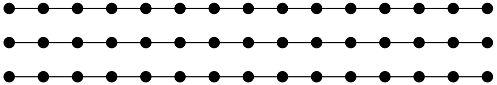
- Communication on the node using shared memory
- Communication between nodes done by multiple (e.g. all) cores
- Setup before execution (generation of bytecode)
- Execution of bytecode

Application programming interface

- Routines are written in ANSI C with C and Fortran interfaces

```
C/C++ :  C interface
int EXT_MPI_Alltoall_init_general (void *sendbuf, int sendcount,
    MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype
    recvtype, MPI_Comm comm_row, int cores_per_node_row, MPI_Comm
    comm_column, int cores_per_node_column, int *handle);
int EXT_MPI_Alltoall_init (void *sendbuf, int sendcount, MPI_Datatype
    sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype,
    MPI_Comm comm, int *handle);
int EXT_MPI_Exec (int handle);
int EXT_MPI_Done (int handle);
```

Advanced features

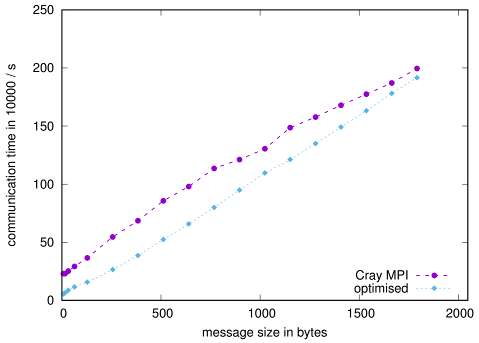


- Multiple collective communication, e.g. for FFTs with pencil decomposition tasks communicate in multiple groups independent from each other
- CUDA aware support for part of our collectives

Supported collective operations

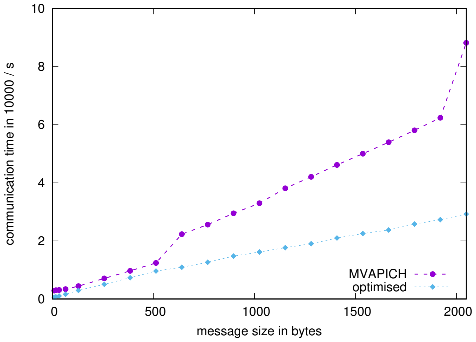
- Alltoall, Alltoallv
- Scatter, Gather (wrappers to Alltoallv)
- Allgather, Allgatherv, Reduce_scatter, Reduce_scatter_block
- Bcast, Reduce (wrappers to Allgatherv and Reduce_scatter)
- Allreduce
- Scan, Exscan

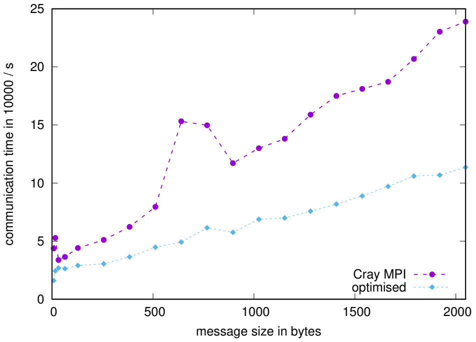
Benchmarks



Alltoall on 156 nodes with 12 tasks per node

Benchmarks





Alltoall on Infiniband (Meteoswiss) 5 nodes with 12 cores per node (left) and on Cray using CUDA aware MPI on 12 nodes with 12 cores per node (right), Cray

Literature

- Andreas Jocksch, Noe Ohana and Emmanuel Lanti and Vasileios Karakasis and Laurent Villard: Towards an optimal allreduce communication in message-passing systems, EuroMPI/USA'20, 2020, accepted
- Andreas Jocksch, Noe Ohana and Emmanuel Lanti and Vasileios Karakasis and Laurent Villard: Optimised allgatherv, reduce_scatter and allreduce communication in message-passing systems, arXiv, 2020
- Andreas Jocksch, Matthias Kraushaar and David Daverio: Optimised all-to-all communication on multicore architectures applied to FFTs with pencil decomposition, Concurrency Computat Pract Exper., 2018

Availability of the code

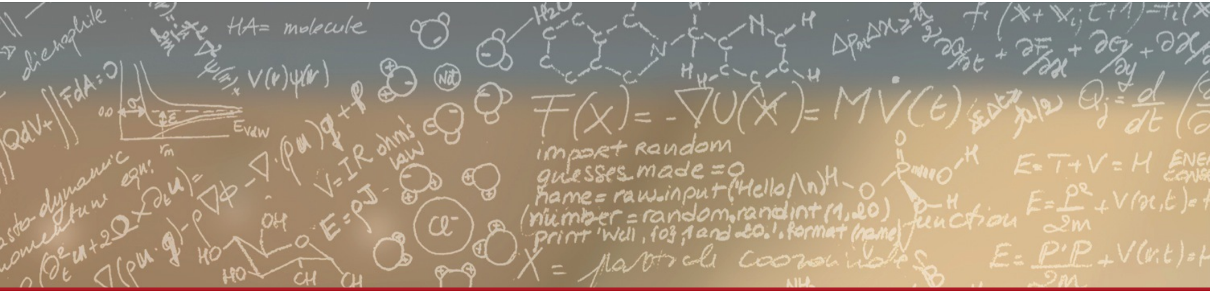
- Work in progress
- ETH-CSCS GitHub — repository “ext_mpi_collectives”
- On request “andreas.jocksch__at__cscs.ch”

Conclusions and outlook

- Collective communication can be higher optimised than the established default case if implemented as persistent communication
- Shared memory on the nodes and complex algorithmic features can be exploited since the setup is hidden in an initalisation phase
- Prototype implementation of collective communication is persistent and blocking

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich


HA= molecule
V(r)ψ(r)
F(x) = -∇U(x) = MV(t)
import random
guesses_made = 0
name = raw_input('Hello\n')
number = random.randint(1, 20)
print 'Well, {0}, I am thinking of a number between 1 and 20.'.format(name)
V=IR ohm's law
E = T + V = H
function
E_vdW

Thank you for your attention.