

Hybrid programming

Jussi Enkovaara

High Performance Computing

CSC – Tieteen tietotekniikan keskus Oy
CSC – IT Center for Science Ltd.

Outline



- What is hybrid programming
- Hybrid programming in theory
- Hybrid programming in practice

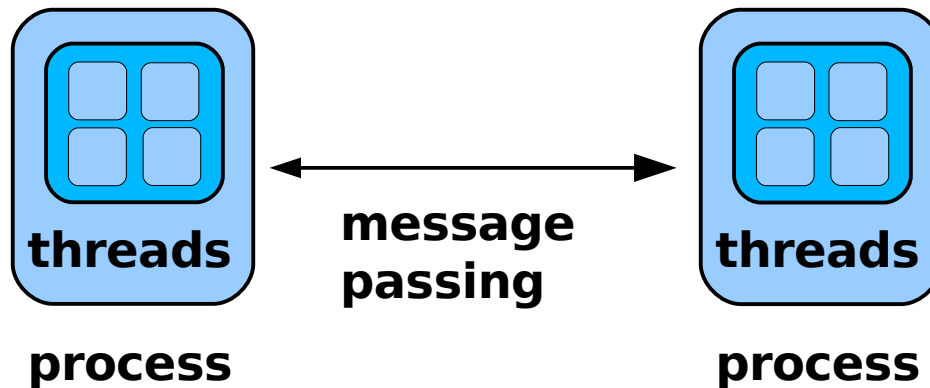
Parallel programming models



- Threads model
 - single process with multiple concurrent execution paths
 - communication through global memory
 - OpenMP, Pthreads
- Message passing model
 - Multiple independent processes
 - communication by passing messages
 - MPI

Hybrid programming

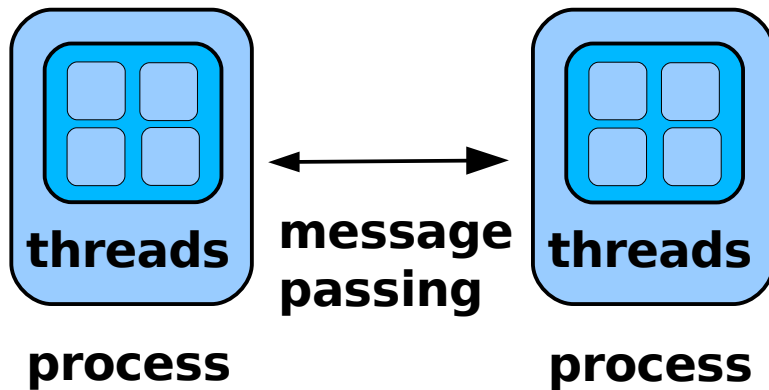
- Parallel programming model combining threads and message passing models
- Message passing between processes
- Threads model within the processes
 - each process can launch multiple threads



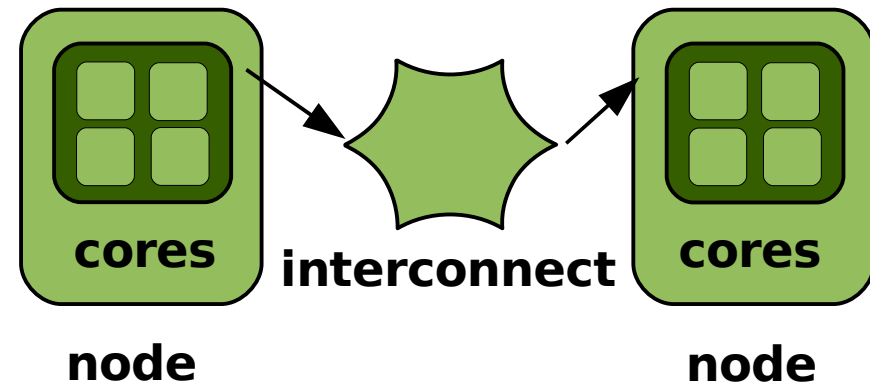
Why hybrid programming?



- Matches the modern hardware model
 - shared memory nodes connected with a high speed interconnects



Programming model



Hardware

Possible benefits



- Load balance
 - hybrid approach decreases the number of processes
- Communication inside node replaced by direct memory access
- Aggregated messages
 - increased communication bandwidth

Possible benefits

- Number of messages are reduced
- Additional parallelization levels with threading
- Decreased memory requirements
 - less replicated data

Possible disadvantages

- Overhead of thread creation and administration
- More complicated program development
- Limited parallelism

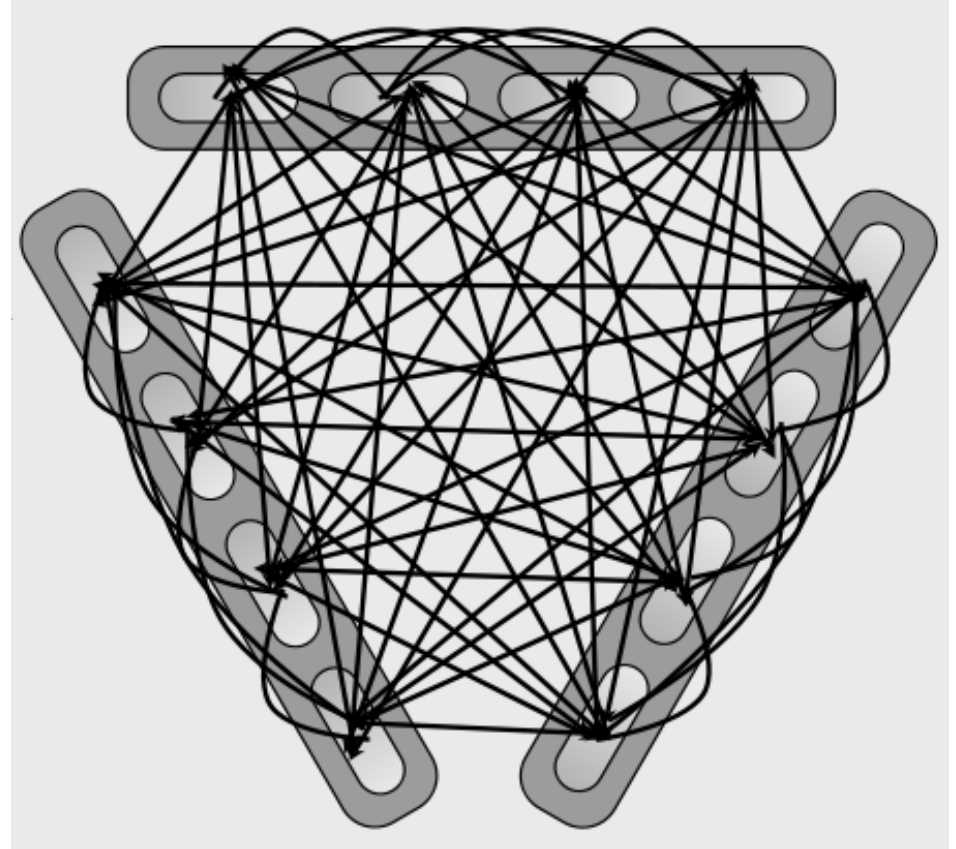
Case study: Gromacs



- Molecular dynamics program for biological simulations
- Main computational load is calculation of short and long ranged forces
- Long ranged forces
 - particle mesh Ewald (PME)
 - uses all-to-all communication

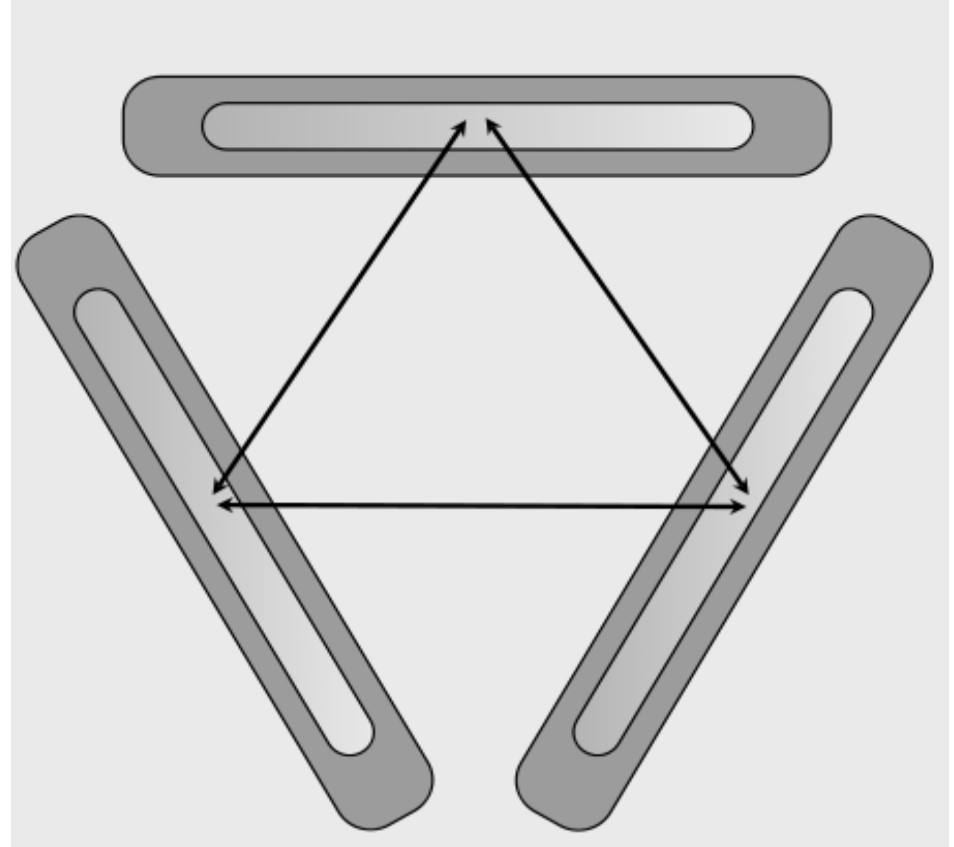
Optimizing all-to-all communication

- Collective all-to-all communication is often performance bottleneck



Optimizing all-to-all communication

- Hybrid implementation
 - number of messages decreases by n_{threads}^2
 - size of messages increases by n_{threads}
 - speed-up depends on message size and number of cores
 - up-to 4-5 times faster



Gromacs: summary



- Hybridization can be useful with current hardware
 - optimized all-to-all communication and threading in PME part gave speed-up of **1.4** with 704 cores
 - the latest code version (4.5) has more thorough threading support

Case study: GPAW



- Electronic structure simulations within density-functional theory
- Implemented in Python and C programming languages
- Good performance (scaling and GFLOPS)
 - can the performance or scalability be improved with hybrid approach?

Algorithms



- Finite-differences in uniform grid
 - 3D nearest neighbor communication with small messages (~tens of kB)
- Parallel matrix-matrix products
 - 1D nearest neighbor communication with large messages (~MBs)
- Integrals over non-uniformly distributed spheres
 - load balance some times problematic

Possible threading benefits for GPAW



- Finite-differences
 - no large benefits expected
- Matrix-matrix products
 - optimized, threaded BLAS library should outperform MPI-based implementation
- Integrals over spheres
 - improved load balance

GPAW: practice



- Finite-difference
 - hybrid version a little bit slower
- Matrix-matrix products
 - only a small improvement over MPI
 - overlapping communication and computation effective in MPI version
- Integrals over spheres
 - individual integrals sped-up
 - no improvement in load balance seen in practice

GPAW: summary

- Currently, a pure MPI implementation is faster
- Problems with threading
 - threading overhead
 - no coarse-grained enough parallelism
 - to be investigated...

Summary



- Hybrid programming model matches the modern hardware
- Hybrid program is not always faster
- An efficient hybrid program can be difficult to implement
- Ever-increasing number of cores per CPU is making hybrid approach more important