Performance Optimization of Scientific Software

Dr. Pekka Manninen
CSC - IT Center for Science
Finland

Part III: Improving Application Scaling

CSC Webinar Nov 20, 2018

Recall: Identifying scalability bottlenecks from performance analysis data

- Signature: User routines scaling but MPI time blowing up
  - Issue: **Not enough to compute in a domain**
    - Weak scaling could still continue
  - Issue: **Expensive collectives**
  - Issue: **Communication increasing as a function of tasks**
- Signature: MPI_Sync times increasing
  - Issue: **Load imbalance**
    - Tasks not having a balanced role in communication?
    - Tasks not having a balanced role in computation?
    - Synchronous (single-writer) I/O or stderr I/O?

IMPROVING LOAD BALANCE

Issue: Load imbalances

- Identify the cause by additional measurements and tests
  - Decomposition, communication design, additional duties (i.e. I/O)?
- Unfortunately algorithmic, decomposition and data structure revisions are often needed to fix load balance issues
  - Dynamic load balancing schemas
  - MPMD style programming

Hybrid programming

- Shared memory programming (OpenMP) inside a node, message passing between nodes
- Reduces the number of MPI tasks - less pressure for load balance
- May be doable with very little effort
  - However, in many cases large portions of the code has to be hybridized to outperform flat MPI
    - In order to reach very big core counts, one needs to be ready to start tackling this
- Needs experimentation with the best threads-per-task-ratio, care with thread affinities, etc

REDUCING PARALLEL OVERHEAD

Rank placement

- Remote access (over the interconnect) is far from homogeneous
  - Three-level network on Cray XC, islands on Infiniband etc
- Rank placement does matter: place the ranks that communicate the most onto the same node
- Changing rank placement happens via environment variables on the batch job script
  - So easy to experiment with that it should be tested with every application
  - For example: CrayPAT is able to make suggestions for optimal rank placement, enabled with the environment variable `MPICH_RANK_REORDER_METHOD`

Optimizing point-to-point communication

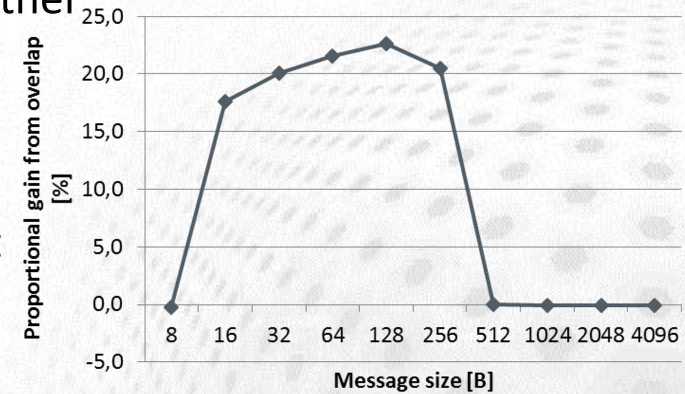
- Use non-blocking operations and try to *overlap* communication with other work
  - Post MPI_Irecv calls before the MPI_Isend calls to avoid unnecessary buffer copies and buffer overflows
- Bandwidth and latency depend on the used protocol
  - *Eager* or *rendezvous*
    - Latency *and* bandwidth higher in rendezvous
  - Rendezvous messages usually do not allow for overlap of computation and communication, even when using non-blocking communication routines
  - The platform will select the protocol basing on the message size, these limits can be adjusted
    - E.g. on Cray XC `MPICH_GNI_MAX_EAGER_MSG_SIZE`

Issue: Expensive collectives

- Reducing MPI tasks by hybridizing with OpenMP is likely to help here as well
- See if you can live with the basic version of a routine instead of a vector version (`MPI_Alltoallv` etc)
  - May be faster even if some tasks would be receiving unrefenced data
- In case of very sparse MPI_Alltoallv's, point-to-point or one-sided communication may outperform the collective operation

Issue: Expensive collectives

- Use non-blocking collectives (MPI_Ialltoall,...)
  - Allow for overlapping collectives with other operations, e.g. computation, I/O or other communication
  - May be faster than the blocking corresponds even without the overlap
  - Replacement is trivial



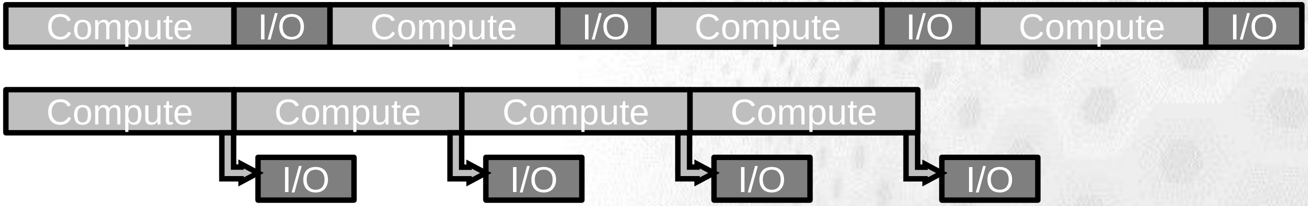
MPI_Ialltoall, 1024 cores Cray XC30

- See the documentation of your MPI library for tunable parameters, and test the impact of them
  - E.g. on Cray XC: increase the value of `MPICH_ALLTOALL_SHORT_MSG`

ADDRESSING I/O BOTTLENECKS

General considerations

- Parallelize your I/O !
  - MPI I/O, I/O libraries (HDF5, NetCDF), hand-written schemas,...
  - Without parallelization, I/O will be a scalability bottleneck in every application
- Try to hide I/O (asynchronous I/O)

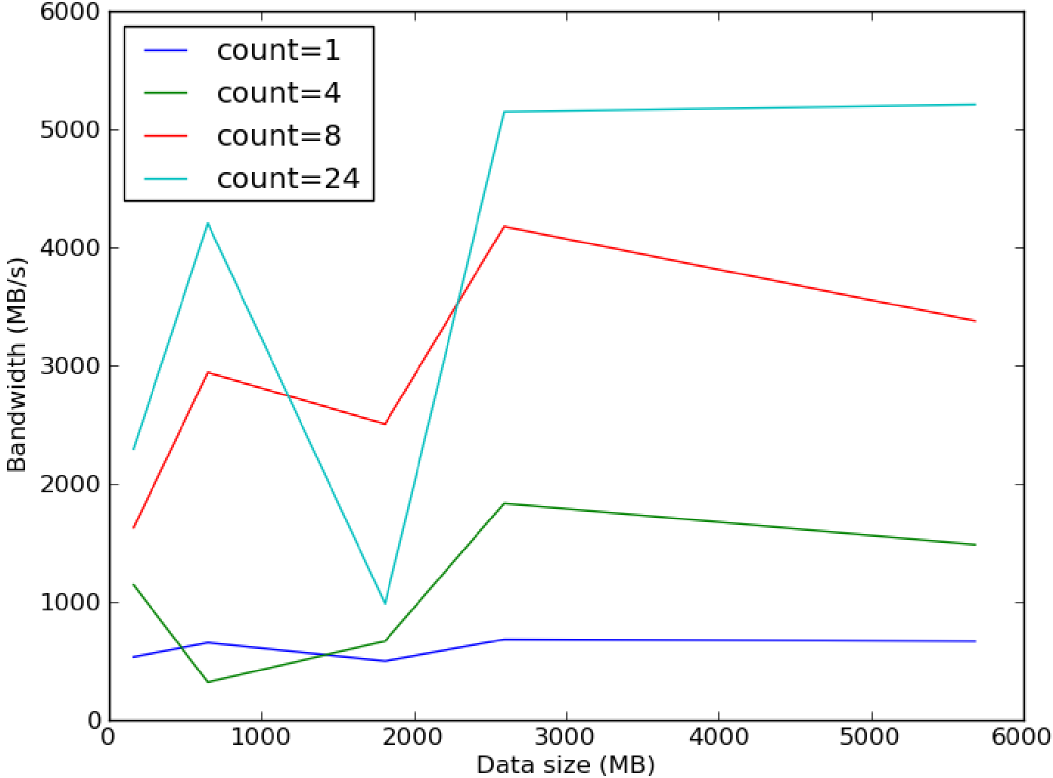


Lustre file striping

- Striping pattern of a file/directory can queried or set with the **lfs** command
- **lfs getstripe** <*dir*|*file*>
- **lfs setstripe** –c *count dir*
  - Set the default stripe count for directory *dir* to *count*
  - All the new files within the directory will have the specified striping
  - Also stripe size can be specified, see **man lfs** for details
- Proper striping can enhance I/O performance a lot

Filesystem parameters

Writing a single file on a Cray XC40 (4 PB DDN Lustre, 141 OSTs)



Summary

- Find the optimal decomposition & rank placement
  - Load balance is established at algorithmic and data structure level
- Use non-blocking communication operations for p2p and collective communication both
- Hybridize (mix MPI+OpenMP) the code to improve load balance and alleviate bottleneck collectives
- All large-scale file I/O needs to be parallelized
  - I/O performance is sensitive to the platform setup
  - Dedicated I/O ranks needed even for simple I/O

Performance Optimization of Scientific Software

Webinar Series Wrap-up

Four easy steps towards better application performance

- Find best-performing *compilers* and *compiler flags*
- Employ *tuned libraries* wherever possible
- Find suitable settings for *environment parameters*
- Mind the *I/O*
  - Do not checkpoint too often
  - Do not ask for the output you do not need

Performance engineering: take-home messages

- Mind the application performance: it is for the benefit of you, other users and the service provider
- Profile the code and identify the performance issues first, before optimizing *anything*
  - "Premature code optimization is the root of all evil"
- Serial optimization is mostly about helping the compiler to optimize for the target CPU
  - Good cache utilization crucial for performance, together with vectorization
- Quite often algorithmic or intrusive design changes are needed to improve parallel scalability
  - To utilize cutting-edge supercomputers, one must be ready to start tackling these

Don't stop here

- Try to apply this stuff yourself!
  - E.g. do the last section from the optional labs
- CSC runs an exhaustive set of HPC courses, e.g.
  - Advanced Parallel Programming (next run in February 2019)
  - Advanced Threading and Optimization (next run in April 2019)
  - see **www.csc.fi/training**
- The PRACE Training Center network provides HPC training opportunities elsewhere in Europe, see **www.training.prace-ri.eu**