Communication Optimization for Medical Image Reconstruction Algorithms

Torsten Hoefler¹, Maraike Schellmann², Sergei Gorlatch² and Andrew Lumsdaine¹

¹Indiana University
²University of Münster

EuroPVM/MPI 2008
Dublin, Ireland

9th September 2008
Positron Emission Tomography

- used to create high resolution images of the inside of a body
- computationally intensive post-processing
- most common is the list-mode OSEM algorithm
- needs many hours on a single CPU
- parallelization is an option to achieve higher performance
PET Details

- radioactive substance is applied to the patient
- patient is placed inside a scanner
- detectors of the scanner count events
- radioactive material emits positrons
- positrons collide with an electron in the surrounding tissue
- collision emits gamma rays which are detected by scanner
PET Parameters

- A single measurement results in $10^7$ to $5 \times 10^8$ events.
- The algorithm computes a 3d image of the substance distribution.
- Ordered Subset Expectation Maximation algorithm is used.
- Image $f$ is vector of $N$ voxels.
- Block-iterative method ($m$ blocks of events).
- $i$-th row of $m \times N$ matrix $A$ represents interaction between event $i$ and a voxel.

for each (iteration $k$) {
  for each (subiteration $l$) {
    for (event $i \in S_l$) {
      compute $A_i$
      compute $c_l = (A_i)^t \frac{1}{A_i f_i^k}$
    }
    $f_{i+1}^k = f_i^k c_l$
    $f_{0}^{k+1} = f_{i+1}^k$
  }
}
Parallelization Options

- two strategies:
  - Projection Space Decomposition (PSD)
  - Image Space Decomposition (ISD)
- PSD distributes events, was shown to be better

1. read $m_s/P$ events
2. compute $c_{l,j} = \sum_{i \in S_{l,j}} (A_i)^t \frac{1}{A_i f_l}$. This includes the on-the-fly computation of $A_i$ for each event in $S_{l,j}$.
3. sum up $c_{l,j} \in \mathbb{R}^N \left( \sum_j c_{l,j} = c_l \right)$ with MPI_Allreduce
4. compute $f_{l+1} = f_l c_l$

- use OpenMP to parallelize computation of steps 2 and 4
- events are read with MPI/IO operations
- exclusive use of collective operations!
The algorithm (schematically)
Optimization Options

- collective operations are already used
- hide overhead? ("overlap" computation and communication)
  ➔ should be possible (at a small cost)!

1. read $m_s/P$ events in the first subset
2. compute $c_{l,j} = \sum_{i \in S_{l,j}} (A_i)^t \frac{1}{A_i f_l}$. This includes the on-the-fly computation of $A_i$ for each event in $S_{l,j}$ in the first subset. Beginning from the second subset, rows $A_i$ have already been computed in parallel with NBC_lallreduce
3. start NBC_lallreduce for $c_{l,j}$ ($\sum_j c_{l,j} = c_l$
4. in every but the last subset, each node reads the $m_s/P$ events for subset $l + 1$ and computes $A_i$ for subset $l + 1$
5. perform NBC_Wait to finish NBC_lallreduce
6. compute $f_{l+1} = f_l c_l$
The new algorithm (schematically)
Potential Overlap

- need enough computation to overlap communication
- but: read-time and computation-time decrease linearly with P
- computation time decreases linearly with number of threads T
  - but: OpenMP doesn't scale that well (investigating)
  - delivers speedup of approx. 2 on 4 cores
- overlap potential:
  - parallelization works against us!
- how much do we need?
  - as much time as the reduction takes!
- reduction-size is scanner dependent (approx. 48 MiB)
48 MiB Allreduce Options

- expect small communicators
- chunk data into P pieces
- reduce in ring: 2P-2 comm/comp cycles
What to expect?

- overhead nearly an order of magnitude lower
- two orders of magnitude with thread and spare core
- we expect the overlap to decrease with increasing P and T
- threaded progression will have problems without spare core
- 32-node application runtime results:
What is the Overhead?

- Allreduce overhead with a single thread per node
- Communication overhead is decreased
- Computation time slightly increased (cache misses)
Conclusions

- Non-blocking Allreduce is easy to apply
  - Needs small code-reorganization to maximize overlap
  - Might cause other slowdowns (cache misses)
- Analysis of overlap potential has to be done before!
  - Also analyze scaling behavior!
  - Parallel scaling works against overlap in some cases
- Progression issues remain complex
  - Threaded vs. Test-based progression
  - Progress thread might cause CPU congestion
- OpenMP and MPI can be combined (also with NBC)