A Performance Exploration of 4D-VAR at High Resolution

Dale Roberts

December 2016
• 4D-VAR Performance Characteristics
  • Scaling
  • Partially Committed Nodes
• New approaches
  • Non-blocking Reductions
  • Non-blocking Consensus Algorithm
• 4D-VAR version 2016.03.0
• N320 Resolution
  • Scaling tests performed with a full 36 iterations
  • Unit testing was performed with a shorter 3 iteration job
• Intel Compilers v15.0.3
• Intel MPI v5.1.2
• Score-P profiling suite v1.4.2
• All tests run on Raijin
  • 2x Intel Xeon E5-2670 CPU/node, 8 Core 2.6GHz
  • 3592 nodes
Performance Characteristics - Scaling

4D-VAR Scaling - Broadwell

Scaling Factor vs. Cores

128 256 512 1024 2048

Cores

0 0.2 0.4 0.6 0.8 1 1.2

Scaling Factor
• Running 4D-VAR on partially committed nodes is known to offer better performance
• What parts of the code in particular?

**Effect of Partially Committed Nodes on 4D-VAR Performance**

- **Relative Runtimes**
  - y-axis: Relative Runtime
  - x-axis: Tasks per node

Comparison between:
- 12 Tasks per node
- 16 Tasks per node

Tasks include:
- gC_elliptic_operator
- gC_elliptic_operator_adj
- mpp_tri_solve_exec
- cubic_lagrange_adj
- gC_precon_adj_exec_tri_adj
- gC_precon_adj_exec_trisolve
- overall
This is more evidence of memory bound code
The CPU frequency increases by 4% when 6 of the 8 available cores are in use
Memory bandwidth per MPI task increases by 33%
Key features of the previous functions are lots of loops over the entire subgrid
• Non-blocking communication has been a part of the MPI standard for a long time
• In 2012, new non-blocking features were added to the standard as a part of MPI3
  • Specifically, non-blocking collective operations
• Best utilised in an area of code where a result is calculated before its needed
• In 4D-VAR, look to the evaluation of the elliptic operator at the poles
• At the poles, the operator terms need to be averaged across the rows.
• This requires a reduction operation only on the processors on which the polar row sections reside.
• There are two terms that end up being averaged across the polar rows
• Perform the communication required for the averaging of the first term while the second term is being calculated.
New Approaches – Non-blocking Polar Reductions

- Almost 50% reduction in communication over the MPI tasks containing the polar rows
- Couldn’t mitigate the cost of the second term that needs to be reduced
• A note on GCOM and MPI2/3...
• GCOM is strictly MPI2 compliant, and looks likely to remain so for the time being
• Asynchronous reductions can be written in an MPI2-compliant fashion
  • Need a ‘progress’ step on one rank
  • Lose access to the various optimized algorithms that an MPI library will use within an MPI_Allreduce
  • Will look almost exactly the same as the ‘deterministic’ version of gcg_rvecsumf
• A new algorithm for ‘Dynamic Sparse Data Exchange’ (DSDE)
  • Receivers don’t know how much data they’re going to get, or where it’s going to come from
• Usual method for handling DSDE:
  • 1) Construct a vector of ‘Data to send’ on each rank
  • 2) ‘Transpose’ these vectors among the ranks
  • 3) Perform point-to-point sends and receives
  • 4) Continue with the model
• This algorithm is implemented in the Interpolation subroutine in the Semi-Lagrangian solver.
Would like to take advantage of the metadata encoded within the point-to-point messages to avoid the explicit exchange
  - MPI_Iprobe can access message metadata without receiving the message
    - This data includes message size and the sender rank
  - Use this data to construct the corresponding receive on the target MPI tasks.

How can we be sure all data has been received?
  - Individual ranks can track that their messages have been received by the intended target
  - Once all ranks agree that their messages have been received, the model continues
‘Non-Blocking Consensus’ algorithm looks like this:

1) Construct a vector of ‘Data to send’ on each rank
2) Start point-to-point synchronous mode sends
3) Check for incoming messages
   3a) If an incoming message is found, query the metadata and post the receive
4) Check if the sends are complete
   4a) If they are, signal that they are complete to the other ranks
   4b) If they aren’t, go to (3)
5) If the sends have already completed, check the status of the global ‘sends complete’ signal
   5a) If all ranks have signaled their sends are complete, continue with model
   5b) If they haven’t, go to (3)
New Approaches – Non Blocking Consensus

• Results

![Time spent in all MPI calls in Interpolation.F90](image)

• Consistently faster, but only by ~5%
• Why only 5%?
  • It turns out that the algorithm behind an MPI_barrier is essentially a zero-sized allreduce
  • N-to-1, then 1-to-N
  • Synchronous-mode send more costly than standard send
  • At 960 cores, the reduced cost of the 1-to-N barrier compared to the N-to-N Alltoall only just mitigates the extra expense of the synchronous-mode sends
  • Should scale better
• Looked at the overall performance characteristics of 4D-VAR
  • Confirmed that scaling is reasonable to a point
  • Very sensitive to decomposition
  • Memory bandwidth bound
• Tried 2 new approaches to MPI communication in key areas of the code
  • One showed a good reduction in the cost of MPI communication
  • One not so much...