INL/EXT-14-32979

Memory Optimization for Phase-field Simulations

Derek Gaston
John Peterson
Andrew Slaughter
Cody Permann
David Andrs

August 2014

The INL is a U.S. Department of Energy National Laboratory
operated by Battelle Energy Alliance

INL/EXT-14-32979

Memory Optimization for Phase-field Simulations

Derek Gaston
John Peterson
Andrew Slaughter
Cody Permann
David Andrs

August 2014

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

CONTENTS

Memory Optimization for Phase-field Simulations

1. INTRODUCTION

Phase-field simulations are computationally and memory intensive applications. Many of the phase-field simulations being conducted in support of NEAMS were not capable of running on "normal clusters" with 2-4GB of RAM per core, and instead required specialized "big-memory" clusters with 64GB per core. To address this issue, the MOOSE team developed a new Python-based utility called MemoryLogger, and applied it to locate, diagnose, and eradicate memory bottlenecks within the MOOSE framework.

MemoryLogger allows for a better understanding of the memory usage of an application being run in parallel across a cluster. Memory usage information is captured for every individual process in a parallel job, and communicated to the head node of the cluster. Console text output from the application itself is automatically matched with this memory usage information to produce a detailed picture of memory usage over time, making it straightforward to identify the subroutines which contribute most to the application's peak memory usage. The information produced by the MemoryLogger quickly and effectively narrows the search for memory optimizations to the most data-intensive parts of the simulation.

2. OPTIMIZATIONS

The first memory optimization study conducted in support of phase-field simulations investigated the memory usage of the Metis mesh partitioning library which is called from MOOSE. Fig. 1 shows the output from MemoryLogger for a two processor run with a 2003 mesh of hexahedral elements. In this case, the application simply read a mesh and partitioned it using the Metis interface in MOOSE, but this particular example is relevant to actual phase-field simulations because they also employ large hexahedral grids on "cube-like" computational domains.

The green line in Fig. 1 depicts memory consumption vs. simulation time prior to the implementation of any memory optimizations. The red line shows the effect of an initial memory optimization which replaced a data structure based on the C++ std::map container with a more memory-efficient data structure based on std::vector. The cyan line shows the memory usage after replacing code that was initially run (duplicated) on all processors with code executed on a single processor followed by a broadcast of the results to the other processors. Finally, the dark blue line shows the memory usage after a final optimization in which a dense two-dimensional data structure used to pass information to the partitioner was replaced with a sparse compressed row storage data structure. The cumulative effect of these memory optimizations was a nearly 2GB decrease in the total peak memory consumed across all processors, and a 10% decrease in execution time for this particular example.

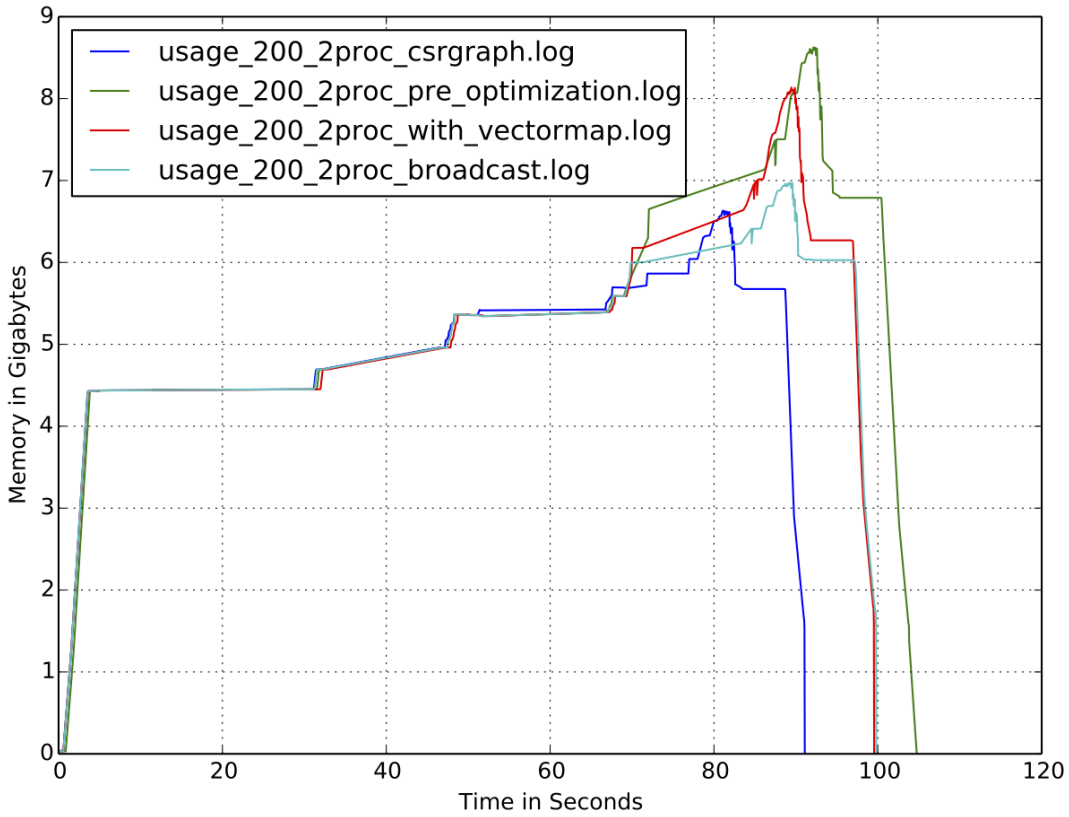

Figure 1: Stages of memory optimization for the Metis partitioner used within MOOSE.

The next memory peak we identified and ameliorated with MemoryLogger was related to serialization of data structures during solution output. The "solution" of a MOOSE-based application is a vector containing floating point numbers representing the value of each variable at every physical point within the domain. Phase-field simulations require many variables: generally 15-20 for a two-dimensional calculation while a three-dimensional calculation may need 40 or more. The need for many variables places a heavy burden on the memory system of a cluster.

Using MemoryLogger, we were able to identify large increases in peak memory usage during the output of solution vectors. Further investigation discovered that the solution vector, which is normally distributed across all processors in a parallel job, was being "serialized" (i.e. gathered) to *every* processor during the output phase of all MOOSE-based phase-field simulations. During serialization, a full copy of the entire solution vector is created on every processor. The memory required by this serialization step would often cause phase-field jobs to run out of memory and crash, even though the spike in memory itself was transient.

Since solution output is a serial process in this particular case, the serialization step is completely unnecessary. Therefore, an obvious memory optimization is to communicate the solution vector to a single processor, and subsequently have that processor write the solution to file. For example, if the solution vector is 4GB in size and the simulation is using four processors, each processor will require approximately 1GB of memory to store its "local" part of the vector. Each processor would then receive an additional full copy of the solution vector during the output phase, increasing memory utilization to 5GB per processor (20GB total). If all four theoretical processes in this example were running on a compute node with, say, 4GB of RAM available per core, then the simulation would crash with an out of

memory error. On the other hand, bringing a full copy of the solution vector to only one processor means that the total memory being consumed by all four processes is only 8GB; well under the 16GB assumed in this example.

Figure 2 demonstrates the effectiveness of this optimization for a problem with a solution vector with two million entries (approximately 16GB) distributed across 800 processors. The green line represents the memory log before the memory optimization just discussed, and the blue line after. In the beginning of the simulation, both versions of the code consume a similar amount of memory. In the pre-optimized version of the code, there is a prodigious spike in memory consumption while writing out the initial solution. The memory consumption in the two cases is once again comparable until the next solution output step is reached, at which point the green line jumps again. Note that MemoryLogger has annotated the lines with the console output from the application, allowing the developer to see what code was being executed during the memory spikes. In this case, the spikes are between "Beginning Exodus Write" and "Ending Exodus Write". Here, "Exodus" is the name of the output file format used in the simulation.

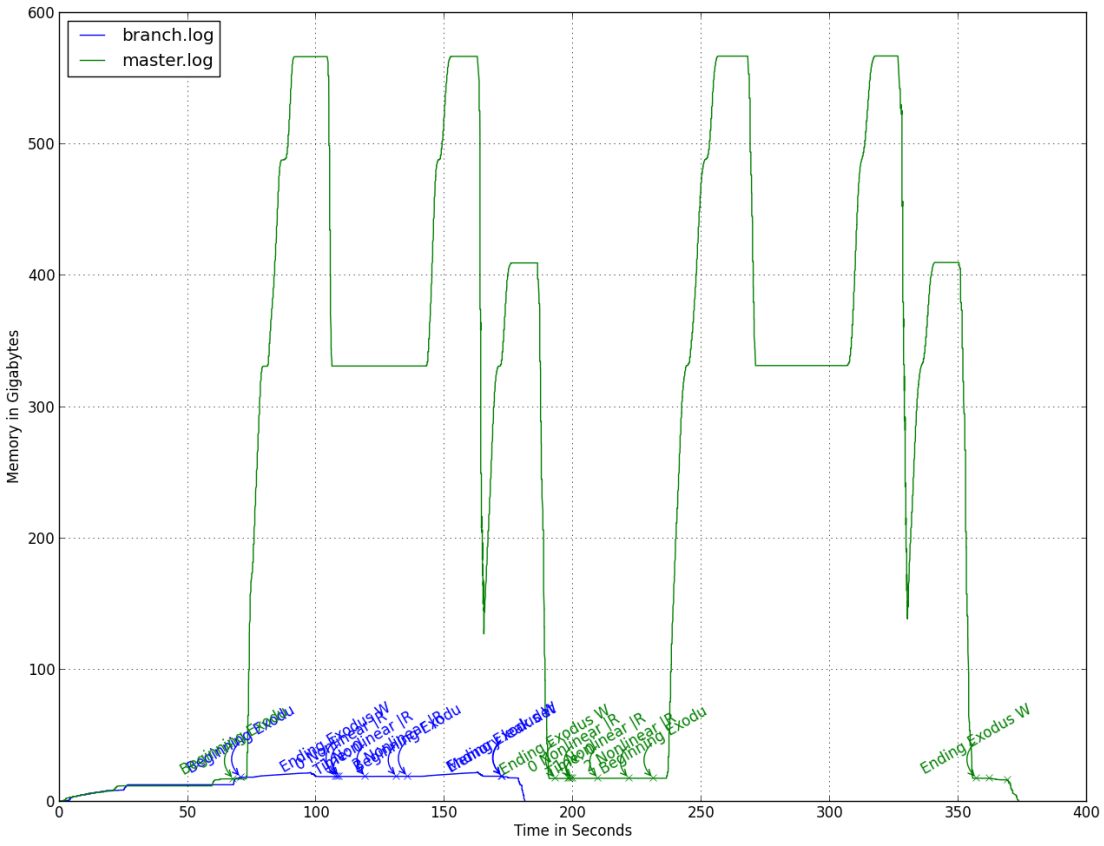

Figure 2: Aggregate memory usage plots for before (master.log) and after (branch.log) optimization of solution output.

In this particular example, the total memory usage was reduced from over 550GB to less than 20GB. This memory savings also scales directly with the number of processors used, allowing larger phase-field simulations to be run than were previously possible. In addition, avoiding the solution vector serialization step actually results in a shorter overall simulation time because the communication

overhead from serialization has been eliminated. In this case, the new code ran nearly twice as fast as the old code, as much less time was spent in writing the solution to file.

3. CONCLUSION

Creation of the new MemoryLogger utility allowed the MOOSE team to identify and abrogate several memory bottlenecks. The changes discussed in this report allow for much larger phase-field simulations, which are both more reliable and faster to execute. In the future, the MOOSE team will continue to work with application domain scientists to identify and address memory issues, and the MemoryLogger will be an invaluable tool for accomplishing this task.