Initial Feasibility Study for Using a Hybrid MPI/OpenMP Approach in Elastic Plastic Impact Computation (EPIC)

by Andrew Tonge
NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer’s or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.
Initial Feasibility Study for Using a Hybrid MPI/OpenMP Approach in Elastic Plastic Impact Computation (EPIC)

by Andrew Tonge

Weapons and Materials Research Directorate, ARL
1. REPORT DATE (DD-MM-YYYY) | June 2018
2. REPORT TYPE | Memorandum Report
3. DATES COVERED (From - To) | January 2018–May 2018

4. TITLE AND SUBTITLE

Initial Feasibility Study for Using a Hybrid MPI/OpenMP Approach in Elastic Plastic Impact Computation (EPIC)

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

6. AUTHOR(S)

Andrew Tonge

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

US Army Research Laboratory
ATTN: RDRL-WMP-C
Aberdeen Proving Ground, MD 21005

8. PERFORMING ORGANIZATION REPORT NUMBER

ARL-MR-0979

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR’S ACRONYM(S)

11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report documents initial testing of a hybrid MPI/OpenMP implementation in EPIC to enable improved scalability for large simulations using modern high-performance computing systems.

15. SUBJECT TERMS

MPI, high-performance computing, EPIC, OpenMP, scaling study

16. SECURITY CLASSIFICATION OF:

| a. REPORT | Unclassified |
| b. ABSTRACT | Unclassified |
| c. THIS PAGE | Unclassified |

17. LIMITATION OF ABSTRACT

UU

18. NUMBER OF PAGES

15

19a. NAME OF RESPONSIBLE PERSON

Andrew Tonge

19b. TELEPHONE NUMBER (Include area code)

(410) 278-1069
## Contents

List of Figures iv

List of Tables iv

1. Introduction 1

2. Methods 2
   2.1 Implementation of OpenMP Threading in EPIC 2
   2.1 Timing Study Setup 2

3. Results 4

4. Discussion 5

5. Conclusions and Future Work 6

References 7

List of Symbols, Abbreviations, and Acronyms 8

Distribution List 9
List of Figures

Fig. 1  Starting configuration for large impact problem used for EPIC timing study .................................................................................................................. 3

List of Tables

Table 1  Timing study results for large simulation run................................................. 4
Table 2  Timing results for vary large simulation.......................................................... 5
1. Introduction

Modern high-performance computing (HPC) clusters consist of multiple compute nodes linked together with a high-speed network. Each node typically has many central processing unit cores and the trend for several years has been increasing the number of cores per node. A modern programming approach to take advantage of this hierarchical structure of the HPC clusters is to use a shared memory threaded programming model for parallel computations within a node and then use a distributed memory programming model to handle the communication between different nodes. For this work OpenMP is used as the shared memory programming model because it is portable to multiple compilers and programming languages and can be added in a way that is easily disabled without any modifications to the source code. In many cases, by using OpenMP within a node and MPI to communicate between nodes, the performance of a given code can be significantly enhanced. The benefits of OpenMP are reduced memory usage because data can be shared among processors within a node; easier load balancing; and enhanced communication performance among the MPI tasks because there are fewer tasks that will tend to send fewer larger messages.

The simulation code that this work targets is an explicit dynamics finite element code designed for impact calculations called EPIC (Elastic Plastic Impact Computation).\(^1\) This code has a long history of use in the Department of Defense (DOD) for armor/anti-armor calculations. EPIC has the capability to perform Lagrangian finite element computations where elements are converted to mesh-free particles when they become too distorted to provide accurate calculations. Additionally, EPIC has a variety of very robust contact algorithms. The current capability maintained by Southwest Research Institute (SwRI) is a parallel implementation using MPI to provide explicit communication. This implementation, depending on the problem type and size, provides very good scaling out to 128 processors and moderate scaling performance to 256 or 512 processors. The goal of this work is to investigate the benefits of using a hybrid OpenMP/MPI approach to enable much larger calculations using EPIC and enable scaling performance to thousands of processors when performing very large calculations.
2. Methods

2.1 Implementation of OpenMP Threading in EPIC

For this proof of concept study, the use of OpenMP threading was limited to loops that are exercised when running a simulation using standard tetrahedral elements with conversion into particles using the Generalized Particle Algorithm (GPA),\textsuperscript{1} in three dimensions, without heat transfer. For these calculations most of the computational work is done in the element calculation loop, the particle calculation loop, and the nodal calculation loop. Internally within EPIC, the elements and particles are divided into blocks for the computations. Each block contains a limited number of elements or nodes. Typically there are many more blocks than MPI processes so the loop over element and particle blocks is a convenient place to add the OpenMP threading. Additionally, EPIC is already set up so that blocks are independent of each other, and there should be no dependence on which order the computations on the blocks occurs.

Within the nodal loop routine there are some sets of calculations that occur over a set of nodal blocks; however, the most expensive portion of the nodal loop seems to be the contact calculation. Portions of the contact calculation were parallelized using OpenMP; however, due to the complexity of the algorithm and associated data flow, some portions of the contact calculation were not distributed among threads. In particular the initial bucket creation (BUCK3) was not parallelized. Most of the OpenMP loops use the guided scheduling option because the execution time for each iteration through the loop is unlikely to take the same amount of time. In all cases, MPI calls are performed only by the master thread so the complexity and cost associated with maintaining thread-safe MPI calls can be avoided. This strategy with the master thread also minimizes the impact of these modifications on the remainder of the code. All OpenMP commands are implemented using compiler directives so that the code can still be compiled without OpenMP to use the MPI only or serial implementation. OpenMP is activated by including a compiler flag during the compile process. When OpenMP is enabled, both the version of the code with MPI and without MPI will be compiled with thread support. This will cause the serial version of EPIC to support shared memory threading parallelism and the parallel version of EPIC to support hybrid OpenMP/MPI calculations.

2.1 Timing Study Setup

EPIC was compiled using the Intel 2017.1.132 compilers with level 3 optimizations enabled and machine-dependent vector extensions. The MPI implementation was SGI MPT version 2.15. These timing studies were run on the DOD Centennial...
system. Each node on Centennial has two Intel Xeon E5-2698v4 Broadwell processors with 20 cores each for a total of 40 cores per node. On Centennial, standard nodes have 128 GB of random access memory (RAM) while a limited number of large memory nodes have 512 GB of RAM.

The impact problem that was used for the studies was a conically tipped cemented tungsten carbide projectile with 12% cobalt binder impacting a boron carbide disk backed by Lexan. These timing studies looked at two problem sizes. A large problem containing about 44 million elements and 10.5 million nodes was restarted at 15 µs of simulated time and run to 16 µs. Of the 10.5 million nodes, 1.1 million were meshless particles. The geometry of the large problem at the time of the initial restart is shown in Fig. 1. A very large problem containing 184 million elements and 38.3 million nodes was run from 0 to 1 µs. The very large problem was not restarted from a pre-existing restart file because attempts to do so resulted in an error and the code failing. As a consequence, the very large problem is dominated by the element computations while the smaller problem provides a more balanced test of a typical impact simulation that is part of the way through the computation. The hybrid OpenMP/MPI simulations were run using 2 MPI ranks per node and 20 OpenMP threads per MPI rank.

![Fig. 1 Starting configuration for large impact problem used for EPIC timing study](image)

Approved for public release; distribution is unlimited.
3. Results

The results from the timing study using the large simulation with about 44 million elements are summarized in Table 1. Timing is reported based on EPIC’s internal timers, and the memory usage is the per node memory use reported by the Centennial job submission system after the job completes. In these results the sliding time is included as part of the nodal loop time. All of the simulations, except for the 160-processor MPI-only simulation, were run using standard compute nodes. For this simulation, the total run time is still decreasing at 1,280 processors.

<table>
<thead>
<tr>
<th>Processors</th>
<th>40</th>
<th>80</th>
<th>160</th>
<th>160 (MPI)</th>
<th>320</th>
<th>640</th>
<th>1280</th>
<th>2560</th>
</tr>
</thead>
<tbody>
<tr>
<td>Walltime (s)</td>
<td>11,973</td>
<td>7,382</td>
<td>4,381</td>
<td>6,623</td>
<td>2,695</td>
<td>1,780</td>
<td>1,409</td>
<td>1,532</td>
</tr>
<tr>
<td>Max mem (GB)</td>
<td>70</td>
<td>62</td>
<td>61</td>
<td>156</td>
<td>58</td>
<td>57</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>Min mem (GB)</td>
<td>70</td>
<td>18</td>
<td>10</td>
<td>101</td>
<td>6.1</td>
<td>5</td>
<td>4.8</td>
<td>4.7</td>
</tr>
<tr>
<td>Cycles</td>
<td>2,100</td>
<td>2,094</td>
<td>2,095</td>
<td>2,095</td>
<td>2,092</td>
<td>2,100</td>
<td>2,172</td>
<td>2,212</td>
</tr>
<tr>
<td>Element loop (s)</td>
<td>2,692</td>
<td>1,966</td>
<td>1,299</td>
<td>3,922</td>
<td>726</td>
<td>474</td>
<td>364</td>
<td>217</td>
</tr>
<tr>
<td>Node loop (s)</td>
<td>4,897</td>
<td>2,520</td>
<td>1,236</td>
<td>58</td>
<td>625</td>
<td>311</td>
<td>157</td>
<td>82</td>
</tr>
<tr>
<td>Sliding (s)</td>
<td>4,844</td>
<td>2,489</td>
<td>1,220</td>
<td>40</td>
<td>616</td>
<td>307</td>
<td>154</td>
<td>80</td>
</tr>
<tr>
<td>Particle loop (s)</td>
<td>2,504</td>
<td>1,253</td>
<td>629</td>
<td>991</td>
<td>338</td>
<td>197</td>
<td>131</td>
<td>336</td>
</tr>
<tr>
<td>Comm (s)</td>
<td>1,502</td>
<td>1,335</td>
<td>1,025</td>
<td>1,529</td>
<td>885</td>
<td>710</td>
<td>688</td>
<td>838</td>
</tr>
</tbody>
</table>

The results from the timing study using the very large problem are shown in Table 2. All of these computational runs used one large memory node for the first two MPI ranks and then used standard nodes for the remaining processes. For the pure MPI timing run, the memory requirements limited the number of processors per node to 20 instead of the 40 that are available. Additionally, the pure MPI run used the memory scalable preprocessor and implementation instead of the standard implementation. As a result, the preprocessing time may not be reported accurately and can be done in parallel, while the preprocessing for all of the other runs was done in serial. In these simulations the limited amount of simulation time results in limited penetration in the problem and there are not many particles generated. As a result, by the end of the problem there are only enough particles for EPIC to spread the particle calculations among 13 MPI ranks instead of all of the available ranks.
Table 2  Timing results for vary large simulation

<table>
<thead>
<tr>
<th>Processors</th>
<th>640</th>
<th>1280</th>
<th>2560</th>
<th>5120</th>
<th>10240</th>
<th>640 (MPI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Walltime (s)</td>
<td>4352.56</td>
<td>3818.57</td>
<td>3520.505</td>
<td>3106.429</td>
<td>2867.675</td>
<td>6127.55</td>
</tr>
<tr>
<td>Max mem (GB)</td>
<td>162.42</td>
<td>213.05</td>
<td>212.2</td>
<td>159.01</td>
<td>211.77</td>
<td>157.7</td>
</tr>
<tr>
<td>Min mem (GB)</td>
<td>19.38</td>
<td>18.71</td>
<td>18.38</td>
<td>18.35</td>
<td>18.35</td>
<td>93.57</td>
</tr>
<tr>
<td>Cycles</td>
<td>2389</td>
<td>2399</td>
<td>2387</td>
<td>2362</td>
<td>2356</td>
<td>2375</td>
</tr>
<tr>
<td>Setup time (s)</td>
<td>1709.055</td>
<td>1711.48</td>
<td>1742.54</td>
<td>1708.374</td>
<td>1710.569</td>
<td>124.826</td>
</tr>
<tr>
<td>Element loop time (s)</td>
<td>726.7183</td>
<td>391.928</td>
<td>197.7213</td>
<td>99.3069</td>
<td>51.0786</td>
<td>1231.62</td>
</tr>
<tr>
<td>Node loop time (s)</td>
<td>150.9887</td>
<td>165.82</td>
<td>170.9699</td>
<td>56.573</td>
<td>32.3449</td>
<td>94.6551</td>
</tr>
<tr>
<td>Particle loop time (s)</td>
<td>241.358</td>
<td>217.191</td>
<td>245.6268</td>
<td>218.2173</td>
<td>173.6678</td>
<td>336.1974</td>
</tr>
<tr>
<td>Comm Time (s)</td>
<td>1358.091</td>
<td>1248.88</td>
<td>1113.786</td>
<td>990.8015</td>
<td>883.974</td>
<td>4451.228</td>
</tr>
</tbody>
</table>

4. Discussion

The timing results demonstrate that adding OpenMP for parallelism within an MPI rank provides both speed and total memory usage benefits for the EPIC code when used in large HPC systems for sufficiently large simulations. For the large problem that contained a significant number of particles, the particle and element calculations showed acceptable scaling all the way out to 1,280 cores. The 40-processor simulation may demonstrate better performance than the simulations that use multiple nodes because all of the MPI communication can be done in a shared memory environment and does not require communication over the high-speed network between nodes. In these calculations the nodal loop did not scale as well because the sliding calculation did not seem to scale particularly well. This is likely because one of the expensive loops in the sliding calculation was not properly parallelized using OpenMP. This conclusion is supported by comparing the sliding calculation time of the 160-processor hybrid run to the 160-processor MPI run. The sliding calculation is about 20 times faster for the pure MPI calculation because it is efficiently spread over all 160 MPI ranks. For the hybrid OpenMP approach, portions that were parallelized, like the neighbor search, performed very well. In future work the BUCK3 routine should be parallelized to use OpenMP when available; however, for a proof of concept, the current state of the parallelization is sufficient. The very large, element-dominated problem shows good scaling all the way out to 10,240 cores. For this large problem, the memory savings resulting from using OpenMP and reducing the number of MPI ranks on each node enables full utilization of each node, which was not possible for the pure MPI implementation.
5. Conclusions and Future Work

This work has demonstrated that hybrid OpenMP/MPI can provide performance benefits for the EPIC code both for elapsed wall time and the memory needed for a simulation. The hybrid implementation was capable of running a 184-million-element simulation and performed reasonably well up to 10,240 cores. Future work in this area should include adding OpenMP parallelism to additional routines that were not parallelized in this work. One of those routines was the BUCK3 bucket initialization for the master nodes in the contact calculation. Other routines that were not parallelized using the hybrid approach were the combined particle element loops and the loops that checked for an element’s proximity to damaged material or a free surface.
References

### List of Symbols, Abbreviations, and Acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOD</td>
<td>Department of Defense</td>
</tr>
<tr>
<td>EPIC</td>
<td>Elastic Plastic Impact Computation</td>
</tr>
<tr>
<td>GPA</td>
<td>Generalized Particle Algorithm</td>
</tr>
<tr>
<td>HPC</td>
<td>high-performance computing</td>
</tr>
<tr>
<td>RAM</td>
<td>random access memory</td>
</tr>
<tr>
<td>SwRI</td>
<td>Southwest Research Institute</td>
</tr>
</tbody>
</table>