Challenges of Performance Portability for Fortran Unstructured Mesh Codes

David Neill Asanza\textsuperscript{1}, Abigail Hsu\textsuperscript{2}

Mentors: Neil Carliosn\textsuperscript{1}, Zach Jibben\textsuperscript{2}

\textsuperscript{1}Grinnell College, \textsuperscript{2}Stonybrook University, Los Alamos National Laboratory

Introduction

This project investigates how different approaches to parallel optimization impact the performance portability of unstructured mesh Fortran codes. In addition, we explore the productivity challenges due to the software tool limitations unique to Fortran.

As a case study, we optimized one of the key computational kernels of Truchas\textsuperscript{1}, a 3D multiphysics application for simulating metal casting and processing. Truchas has been in development for over 20 years, is written in modern Fortran, and uses unstructured meshes for modeling complex geometries.

Challenges

Kernel optimization is challenging due to available tools, Fortran features, and unstructured meshes.

- Limited compiler support for Fortran 2008
- Limited compiler support for OpenMP GPU
- Lack of performance-portable libraries for Fortran
- No CUDA Fortran compiler compatible with Truchas, forced to rewrite computational kernel in CUDA C
- Fortran array syntax requires expanding to do loops for OpenMP optimization
- Indirect addressing for unstructured meshes causes ineffective caching, in contrast to regular access patterns of structured meshes

Methodology

We modified and optimized the key kernel to investigate the performance portability of three parallelization approaches. To determine portability, we ran on a variety of different node configurations. To measure productivity, we recorded number of line changes.

Parallelization approaches:

- OpenMP CPU: Directive-based parallel runtime with a work-sharing model
- CUDA: GPU-specific language, only runs on Nvidia hardware
- OpenMP GPU: New directives available in OpenMP 4.0+ enable GPU computing

Hardware resources:

<table>
<thead>
<tr>
<th>Label</th>
<th>CPU model</th>
<th>S/C/T</th>
<th>GPU model*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Haswell 2S</td>
<td>Intel® Xeon® E5-2698 v3 2.30 GHz</td>
<td>2:16:2</td>
<td>N/A</td>
</tr>
<tr>
<td>Haswell 4S</td>
<td>Intel® Xeon® E7-8880 v3 2.30 GHz</td>
<td>4:18:2</td>
<td>N/A</td>
</tr>
<tr>
<td>KNL</td>
<td>Intel® Xeon Phi® 7250 0.14-0.40 GHz</td>
<td>1:08:4</td>
<td>N/A</td>
</tr>
<tr>
<td>Broadwell</td>
<td>Intel® Xeon® E5-2695 v4 2.60GHz</td>
<td>2:18:2</td>
<td>NVIDIA® TITAN V®</td>
</tr>
<tr>
<td>Power9</td>
<td>IBM® POWER9™ 3.80 GHz</td>
<td>2:20:4</td>
<td>NVIDIA® Tesla® V100 SXM2</td>
</tr>
</tbody>
</table>

* Sockets : Cores per socket : Threads per core
** Kernel only ran on one of the on-node GPUs

Acknowledgements

Many thanks to our mentors: Rob Bobey, Hai Ah Nam, Kris Garrett, Doug Jacobsen, Neil Carliosn, and Zach Jibben.

Support provided by ASC Integrated Codes Telluride Project.

Support provided by U.S. Department of Energy at Los Alamos National Laboratory supported by Contract No. DE-AC52-06NA25396


References


Performance and Productivity Analysis

Figure 3. OpenMP CPU performance across node types (lower is better). Kernel performance scales well initially as the number of cores increases, but scales poorly with more than one thread per core. Only KNL and Power9 gfortran continue to speed-up with more than one thread per core. With the most physical cores (72), Haswell 4S is the best performing hardware.

Figure 4. Vector length impact on run time across Intel® processors (lower is better). For small vector sizes, Haswell 2S performs better than KNL due to its higher clock speed. KNL has more than twice as many vector units as Haswell 2S [2], resulting in a lower run time for 256-bit vectors. The performance for 512-bit vectors likely decreased due to indirect addressing.

Figure 5. Performance relative to original serial code run on Haswell 2S (higher is better). Each approach speeds-up the kernel by at least a factor of 4. OpenMP on the CPU performs best due to powerful CPUs, namely Power9 and Haswell 4S. OpenMP on the GPU and CUDA perform similarly on one GPU. Power9 xlf runs 4 times faster than Power9 gfortran.

Conclusion

- Compiler choice can dramatically impact performance on the same architecture.
- OpenMP on the CPU is the most productive approach, requiring the least programmer effort for at least a factor of 4 speed-up. It is also the most portable due to widespread compiler and hardware support.
- OpenMP on the GPU is a viable optimization approach, requiring little programmer effort and providing comparable performance to CUDA. This approach is currently limited to running on IBM hardware (Power9) using the xlf compiler, but it may become more portable as compilers adopt the standard.
- CUDA is the least productive approach because it requires adding a Fortran-C interface and rewriting the computational kernel in CUDA C. More optimization effort may yield increased performance. Portability is limited because the CUDA API can only run on Nvidia hardware.
- Fully utilizing all available on-node GPUs could provide a significant performance improvement, especially given the increasing adoption of GPUs in high-performance computing.