Performance Portable Parallel Programming

Compile-Time Defined Parallelization and Storage Order for Accelerators and CPUs

Abstract
Performance portability between CPU and accelerators is a major challenge for coarse grain parallelized codes. Hybrid Fortran offers a new approach in porting for accelerators that requires minimal code change compared to the performance of CPU optimized loop structures and storage orders. This is achieved through a compile-time code transformation where the CPU and accelerator cases are treated separately. Results show minimal performance losses compared to the fastest non-portable solution on both CPU and GPU. Using this approach, five applications have been ported to accelerators, showing minimal slowdown on CPU while enabling high-speeds on GPU.

1. Motivation
When porting real world HPC applications for accelerators, performance portability is often one of the main goals - it is imperative that code can be executed on different architectures with at least reasonable performance. Achieving this for accelerators is a major challenge since their architecture is so different from CPUs.

2. Proposal
In order to (1) ensure performance portability and (2) minimize code portation, we propose the following solution:
1. Allow both coarse-grained and fine-grained parallelization in the same codebase through directives. This enables optimal parallelization for both CPU and accelerator architectures.
2. Automate the privatization of symbols where needed, such that the original code can be kept with a low number of dimensions.

3. Method
Hybrid Fortran[1] is an Open Source preprocessor framework and a Fortran language extension developed for the task of allowing such hybrid parallelization as described in (2) and transforming such unified codes into standardized x86 Fortran and Accelerator enabled Fortran. So far, OpenMP, OpenACC and CUDA Fortran parallelizations are implemented. Hybrid Fortran currently supports any data parallel code that can be implemented on shared memory systems. Storage order is abstracted and can be defined in a central location without any changes to array accessors and declarations.

Advantages over pure OpenMP / OpenACC:
• No manual privatization of callgraph necessary (this saves ~20k LOC changes in case of ASUCA)
• Less overhead on GPU than OpenACC since CUDA Fortran can be used
• No directive code duplication

1D Computational Code (in Z - No changes needed)

4. Performance Results

<table>
<thead>
<tr>
<th>Performance Characteristic</th>
<th>Speedup CPU vs. Core</th>
<th>Speedup on GPU vs. 6</th>
<th>Speedup on CPU vs 1 Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ASUCA Physical Weather Prediction Core (121 Kernels) [2]</td>
<td>4.47x</td>
<td>3.63x</td>
<td>16.22x</td>
</tr>
<tr>
<td>2. 3D Diffusion (Source on Github) [1]</td>
<td>1.06x</td>
<td>10.94x</td>
<td>11.66x</td>
</tr>
<tr>
<td>3. Particle Push (Source on Github) [1]</td>
<td>9.08x [B]</td>
<td>21.72x</td>
<td>152.79x</td>
</tr>
<tr>
<td>4. Changes on FIM Solve with Jacob Approximation (Source on Github) [1][2][3]</td>
<td>1.41x</td>
<td>5.13x</td>
<td>7.28x</td>
</tr>
<tr>
<td>5. ASUCA All Solve with Jacob Approximation (Source on Github) [1][2][3]</td>
<td>5.26x</td>
<td>10.07x</td>
<td>52.99x</td>
</tr>
</tbody>
</table>

4.1 ASUCA Physical Weather Prediction Core - Speedup

6. Conclusion and Future Work
The preprocessor framework “Hybrid Fortran” has been developed and shown to:
1. ... be portable, performable, ...
2. ... require minimal code changes for porting CPU code to accelerators, ...
3. ... be general purpose capable for various data parallel problems.

Until Early 2015 we will extend the ASUCA on Hybrid Fortran implementation to include the entire model (Dynamical + Physical Core) and integrate the multi-node parallelization using MPI. ASUCA on Hybrid Fortran is expected to become production ready and operational in 2015.

---