Performance Portable Parallel Programming

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

Michel Müller
Tokyo Institute of Technology
Global Scientific Information and Computing Center
Aoki Laboratory

***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 changes and allows keeping 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 or no slowdown on CPU while enabling high speedups on GPU.*

Keywords—GPGPU, CUDA, OpenACC, Hybrid, Fortran, HPC

I. 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.

Often the biggest change is going from coarse-grained parallelism (order of 10-100 threads per processor) to fine grained parallelism (order of 10'000 - 100'000 threads per processor). This is particularly challenging for code that is parallelized at a point in the program that is far removed from the actual computations. The most prominent examples are physical cores for weather and climate models. Our motivation, the next generation Japanese weather prediction model ‚ASUCA', contains ~20k lines of code in its physical core - parallelized in a single loop.

The usual approach (using OpenACC / OpenMP for Intel MIC) is to privatize such code in the parallel domains in order to split up into multiple smaller kernels. This leads to

1. Substantial performance losses when executing this code on CPUs. It is therefore not performance portable.
2. A complete rewrite of the computational code, with lots of mechanical work for simply inserting additional domains in declarations and accessors. This is bug prone and leads to less readable code.

II. 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.

III. METHOD

Hybrid Fortran[1] is an Open Source preprocessor framework and a Fortran language extension developed for the task of allowing such hybridized parallelizations as described in (2) and transforming such unified codes into standard 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 Fortr
- an can be used
- No directive code duplication

IV. Performance Results

Table I gives an overview over the current performance results (details please see the poster).

	Performance Characteristics	*Speedup HF on 6 Core vs. 1 Core*	*Speedup HF on GPU vs 6 Core*	*Speedup HF on GPU vs 1 Core*
1. ASUCA Physical Weather Prediction Core (121 Kernels) [2]	Mixed, Coarse Grain Parallelism	4.47x	3.63x	16.22x
2. 3D Diffusion (Source on Github) [1]	Memory Bandwidth Bound, Fine Grained Parallelism	1.06x	10.94x	11.66x
3. Particle Push (Source on Github) [1]	Computationally Bound, Sine/Cosine operations, Fine Grained Parallelism	9.08x	21.72x	152.79x
4. Poisson on FEM Solver with Jacobi Approximation (Source on Github) [1]	Memory Bandwidth Bound, Fine Grained Parallelism	1.41x	5.13x	7.28x
5. MIDACO Ant Colony Solver with MINLP Example (Source on Github) [1] [3]	Computationally Bound, Divisions, Coarse Grain Parallelism	5.26x	10.07x	52.99x

TABLE I. Performance Results

V. Conclusion and Future Work

The preprocessor framework "Hybrid Fortran" has been developed and shown to

1. be performance portable,
2. require minimum 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.

Acknowledgment

We'd like to thank Dr. Johan Hysing, Tokyo Institute of Technology, for his contributions on the Poisson Solver. Many thanks also to Dr. Martin Schlueter for the collaboration on porting a sample problem for the MIDACO solver onto Hybrid Fortran.

References

[1] M. Müller "Hybrid Fortran Github Repository", [Website, Accessed 2014-10-17] http://github.com/muellermichel/Hybrid-Fortran

[2] T. Hara et. al "Development of the Physics Library and its application to ASUCA", 2012

[3] M. Schlueter "MIDACO-Global Optimization Software for Mixed Integer Nonlinear Programming", 2009