HIGH PERFORMANCE CODE GENERATION FROM SYMBOLIC COMPUTING

Renato Miceli[1], Felippe Vieira[2], Marcos de Aguiar[3]

[1]SENAI CIMATEC, E-mail: renato.miceli@fieb.org.br

[2]SENAI CIMATEC, E-mail: felippe.vieira@fieb.org.br

[3]SENAI CIMATEC, E-mail: marcos.aguiar@fieb.org.br

RESUMO

Novas e radicais mudanças nas arquiteturas de computadores possibilitarão muitas oportunidades no campo de aplicações computacionais de alto desempenho, ao mesmo tempo em que irão também demandar mudanças radicais do ponto de vista de desenvolvimento de software, para tirar o máximo de proveito dessas novas plataformas. Esse cenário traz a seguinte questão: como conseguir portabilidade de desempenho para diferentes (e em constante evolução) arquiteturas? Normalmente existe uma clara distinção de desempenho entre códigos de fácil manutenção e portabilidade, escritos em linguagens de alto nível, e códigos paralelos altamente optimizados para certa arquitetura. A solução proposta pelo OPESCI (Open portablE Seismic Imaging) framework é tirar proveito da tecnologia de linguagens de domínio especifico (DSL) e de geração de código, para introduzir múltiplas camadas de abstração de software. No nível mais alto de abstração, desenvolvedores de aplicação irão escrever algoritmos de forma clara, semelhante a fórmulas matemáticas escritas a mão em um papel. Enquanto que no nível mais baixo, compiladores fonte para fonte irão transformar essa DSL em código altamente optimizado para uma arquitetura alvo e deve executar com desempenho próximo ao máximo, realista, possível para aquela aplicação. Diferentes geradores de códigos podem existir, enquanto a camada de alto nível consegue manter portabilidade de desempenho. O resultado é uma separação de conceitos, onde outras abordagens numéricas podem ser avaliadas e obter desempenhos compatíveis ou melhor do que códigos optimizados manualmente.

Palavras-Chaves: Imageamento sísmico; Geração de código; Stencil; Hpc; Otimização de código;

ABSTRACT

Upcoming disruptive changes to computer architectures offer many new opportunities for developing high performance applications, but it also demanding disruptive changes in software to achieve the full potential of the new hardware. Therefore, the question now is: how we achieve an acceptable degree of performance portability across different (and rapidly evolving) architectures? There is in general a sharp trade-off between easy to maintain,

extensible portable software written using high-level languages, and highly optimized parallel code for a target architecture. The solution proposed by the OPESCI (Open portable Seismic Imaging) framework is to leverage domain specific languages (DSL) and code generation software technologies to introduce multiple layers of software abstraction. At the highest level of abstraction application developers will be able to write algorithms in a clear and concise manner akin to how the algorithm might be written mathematically on paper. While at the lower levels source-to-source compilers will explore a rich implementation space to transform this DSL code into highly optimized code that can be compiled for a target platform to run at near-to-peak performance. It will provide layers that decouple domain experts from code tuning specialists, where different optimized code generator back ends can be replaced, and the high level code attains its portability. The result is a separation of concerns where new numerical approaches are readily evaluated and are capable of matching or outperforming hand tuned code.

Keywords: Seismic imaging; Code generation; Stencil; Hpc; Code optimization;

1. INTRODUCTION

High performance computing is getting more and more relevant every day. It went out of the academy and is becoming main stream and strategic in the high tech industry. It allows researchers to simulate experiments that would be too expensive, or too risky, or unethical or any combination of those. Big machines, also provides the computing power to solve very large problems, like predicting orbits and gravitational interactions among billions of celestial bodies.

Computing power keeps increasing at an accelerated pace, but because of the current technology limits, this new performance comes with the price of greater architectural complexity. In the past, the advance in speed was restricted mostly to a higher clock speed. The software would just get faster automatically when the frequency and bus speeds were incremented. Nowadays, because of the thermal properties of the materials used to build processor and memories, the clock speed is halted, and the performance improvements are mostly being achieved by the means of parallelization. Heterogeneity is also a big factor today. Performance opportunities are also offered in the use of different hardware components like GPU's (graphics processing unity), coprocessors (many core architectures) and FPGA (software programmable circuits)[1]. The learning curve to use all those components effectively is already a big challenge, and, to aggravate the issue, the optimization techniques can be quite different for each of these architectures. As the time passes, techniques that were once successful, won't work with new releases of the hardware.

With this current scenario, the software community must find new disruptive ways to harness the great performance that modern hardware offers, and at the same time be able to maximize the investment made in application code development.

Taking this situation under consideration, and also other issues that troubles the scientific and high performance computing community, the OPESCI (Open Performance portablE Seismic Imaging) initiative was created. The main goal of the project is to achieve high performance code portability across different architectures. At the highest levels, domain experts will use high level languages to describe solutions to their problem of study, while at the lowest one, high performance coding specialist will write fine-tuned code to specific architectures.

The domain experts (i.e. scientists, engineers) will write code using DSL (domain specific languages)[2] without having to know in which platform this code will be executed. High performance specialists will use every language and compiler that will help get the most of the hardware. In between OPESCI will provide the abstraction layers to transform the DSL code in the most optimized form possible, exploring every aspect of optimization automatically. New backend plugins can be added to the platform to target different architectures and resolution methods.

In the beginning the OPESCI framework will focus primarily in stencil computation. Stencil is an integral part of applications in a number of scientific computing domains where each point of a *d*-dimensional grid uses the value of itself and some groups of neighboring elements to be repeatedly updated. It is an important computational pattern used in a variety of domains such as electromagnets, solutions of partial differential equations using finite difference or finite volume discretization, image processing and etc [3].

2. METHODOLOGY

The Project team is composed of different profiles: mathematicians, physicists, and computer scientists.

Domain Expert main tasks are:

Develop and validate models in high level portable languages.
Test the platform.
Provide real world data to benchmark the software.

Computer Scientists main tasks are:

Benchmark codes that solves the models created by the domain expert team.
Look for ready to use tools that already exploit performance on modern hardware for a given numerical method (i.e. finite difference, finite element).

Devise a roofline model to have an expectation of how much performance improvement can be achieved for the current code.
Tune code to get as close as possible to the roofline value, taking in consideration that this model is not perfect and 100% performance matching is unlikely to be achieved (due to different memory speeds and cache hierarchies not normally taken into account by the available roofline models).
Design code generation interfaces that will allow contributors to plug in various back-ends for different architectures.

The domain experts will help by developing high level code that will serve as the base for the source generation. They will also check the correctness and effectivity of the generated code and the results it calculates. Their input will also influence in which direction the project will go, as they are the target audience to use the final product.

When the point comes that the abstraction interfaces are stable. The software development team, will focus on porting, tuning and developing more and more back-ends to the platform. New benchmarks will be made, and the tuning cycle will keep going for every new hardware platform release.

The authors are currently trying out optimization strategies based on the latest academic publications, as well as special code generators, with the goal to develop the first back end to this framework. It's being tested both in conventional CPU as well as in a Many-Core Coprocessor. This module is platform specific and will be plugged in the low level layer of the system.

Another important point to make is that this framework is being developed as a IPCC (Intel Parallel Computing Center) initiative, and the resulting work will be made available to the community as open source software. Other projects and researchers will be able to plugin they're own back ends, as well as to provide their high level modeling codes.

3. RESULTS

So far the Project is still at the beginning but some of these steps were already executed, and are now being refined.

As first step we benchmarked existing solutions for stencils optimization. The goal was to leverage existing solutions or improve them to use as back-ends for framework's finite difference kernel generator.

Three different solutions for stencil optimization were tested, as seen in figure 1. The SDSLC[3] stencil compiler which applies a set of compiler transformations to generate efficient code for multicore processors with short-vector SIMD instructions. The pluto[4][5] a fully automatic source-to-source transformation framework that can optimize regular programs (sequences of possibly imperfectly nested loops) for parallelism and locality

simultaneously. Finally, the pochoir[6], which optimization technique translates the code on an efficient parallel cache algorithm.

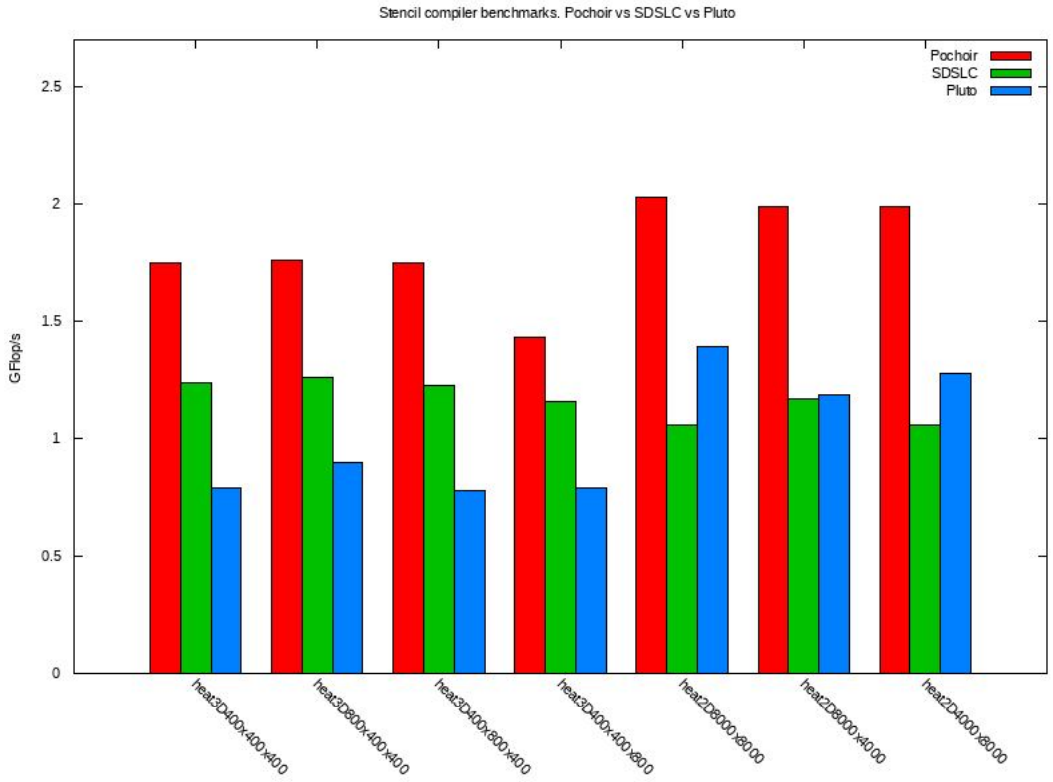

Figure 1. Stencil compilers benchmark.

The algorithms used were the stencil to simulate the heat propagation in a medium. Its 3d and 2d implementations were used, varying its grid size and keeping the time steps uniform across all tests. In all tests, the pochoir stencil compiler was better than the other ones, because its optimization technique consider both time and spatial domain that leverage a better performance of stencil algorithms in modern architectures.

Even though pochoir outperformed other stencil compilers, manually developed code with OpenMP had better results, as seen on figure 2. The data leads to the path of generating regular OpenMP kernels, instead of pochoir DSL Stencils. Starting from these initial results, the next steps to assembly the lower level code generation are being developed.

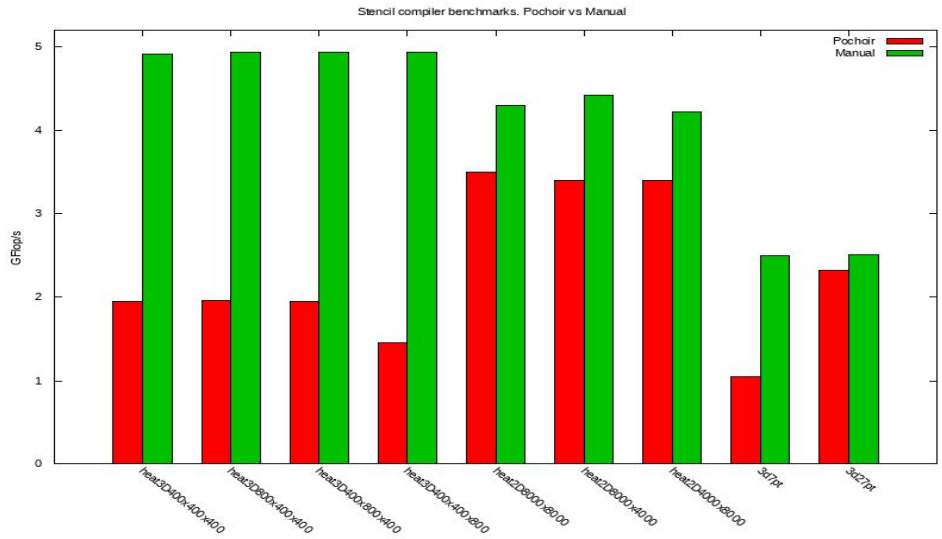

Figure 2. Performance comparison between pochoir and hand-made code with OpenMP.

First, the code generation tool using SymPy[7] (Python for symbolic computing) is being used to already transform high level symbolic language code into OpenMP finite difference kernel implementations. This tool also will enable develop plugins to translate its high level to others stencil compilers language.

While domain specialists are working on models, a reference implementation for elastic wave equation was developed and has been used as reference. The generated kernel for this elastic wave equation model from SymPy is being benchmarked against the reference hand-made code, as well as against other stencil compilers, like pochoir, to see how good it perform against existing optimization back end solutions.

Once this last step is done, the next task will be to make sure that the generated code is being fully vectorized and using every automatic optimization opportunity that the compiler offers. After that more advanced techniques will follow, like doing better use of cache memory and minimizing main memory access.

4. CONCLUSION

So far the Project showed good evolution and the team managed to have the basic workflow working. Based on the result obtained from comparing the stencil compilers, OpenMP code was chosen to be the first backend to attempt to be integrated with the framework. So, additional improvements will be done progressively in order to get higher performances from the generated code at specific architectures.

There is still more to be explored in terms of defining longer lasting interfaces for the framework as well as exploring or developing more efficient optimization engines for a given numerical method running for a specific architecture.

The team will also decide whether only SymPy and Firedrake UFL[8][9] DSL's will be used or other ones should also be adopted.

5. REFERENCES

[1]Barros, R. S., Geldermalsen, S., Boers, A. M. et al *"Heterogeneous Platform Programming for High Performance Medical Imaging." Lecture Notes in Computational Science* (Springer) Volume 8374 (**2014**).

[2]Taha, W. M. *"Domain-Specific Languages."* IFIP TC2 Working Conference (Oxford) (**2009**).

[3]T. Henretty, J. Holewinski, R. Veras, F. Franchetti, L.N. Pouchet, J. Ramanujam, A. Rountev, P. Sadayappan. "A Domain-Specific Language and Compiler for Stencil Computations on Short-Vector SIMD and GPU Architectures," Compilers for Parallel Computing Workshop (CPC), July 2013.

[4]Bandishti, V., Pananilath, I. and Bondhugula, U. *Tiling stencil computations to maximize parallelism*. In SC, 2012

[5]Bondhugula, U., Hartono, A., Ramanujam, J. and Sadayappan, P. *A practical automatic polyhedral program optimization system*. In PLDI, 2008.

[6]Tang, Y., Chowdhury, R., Luk, C., Leiserson, C. E. .*"Coding stencil computations using the Pochoir stencil-specification language". In 3rd USENIX Workshop on Hot Topics in Parallelism (HotPar'11)*, **2011**

[7]Joyner, D., Čertík, O., Muerer, A., Granger B. E. *"Open Source Computer Algebra Systems: Sympy."* ACM Communications in Computer Algebra *Volume 45* (**2011**).

[8]Alnæs, M. S. *"UFL: A Finite Element Form Language, Automated Solution of Differential Equations by the Finite Element Method." Lecture Notes in Computational Science and Engineering* (Springer) 84 (**2012**).

[9]Alnæs, M. S., A Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells. *"Unified Form Language: A domain-specific language for weak formulations of partial differential equations." ACM Transactions on Mathematical Software*, **2014**.