Dynamic Compilation in HPC

Stefano Cherubin[1], Giovanni Agosta[2]

DEIB, Politecnico di Milano, via Ponzio 34/5, 20133, Milano, Italy

ABSTRACT

High Performance Computing (HPC) infrastructures are crucial to the competiveness of companies in a range of sectors whose trend is expanding in recent years. As a consequence of this trend, an increasing share of programmers deal with HPC architectures. However, there are optimizations which are common in general-purpose computing – such as profiling, ahead-of-time code specialization – that may not be supported in HPC scenarios due to the vastity of the input data. We propose an easy-to-use approach to dynamic compilation, guidelines, and case studies to effectively exploit it on HPC architectures. In particular, we discuss continuous program optimization, and the case of dynamic precision tuning of fixed-point representations.

KEYWORDS: Dynamic Compilation; Continuous Optimization; Precision Tuning

1 Introduction

Designing and implementing HPC applications is a difficult and complex task. The availability of effective APIs and programming languages is crucial to guarantee to developers the ability to work effectively on HPC platforms. Profile-guided code transformations and specialization at compile-time usually provide a good optimization level in general-purpose scenarios. However, the large data sets employed in HPC scenarios prevent proper profiling. In these cases, dynamic optimizations can prove more effective.

Dynamic compilation allows source code to be (re-)compiled at runtime. This allows the compiler to perform extra code specialization based on constant values or particular conditions which are known only at runtime. However, dynamic compilation techniques described in the literature usually employ ad-hoc compilation frameworks [A+96].

LIBVERSIONINGCOMPILER (LIBVC) [CA18] provides simple C++ APIs to perform dynamic compilation without the need to support customized frameworks nor just-in-time compilers. Indeed, it relies on industry-grade compilers that are already available on the host machine.

2 Continuous Optimization

The practice of improving the application code at runtime via dynamic recompilation is known as *continuous program optimization*. Automatic tools do not guarantee a good level of

[1]E-mail: stefano.cherubin@polimi.it
[2]E-mail: agosta@acm.org

software maintainability whereas manual implementation is not trivial. Thus, this practice is not widely adopted.

LIBVC allows different versions of the executable code of a computational kernel to be transparently generated at runtime. Hence, it can be used to perform continuous program optimization. Continuous program optimization with LIBVC can be performed by dynamically enabling or disabling code transformations, and changing compile-time parameters according to the decisions of other software tools such as a generic application autotuner like mARGOt [GPS15].

2.1 Case Study: Geometrical Docking Miniapp

To assess the impact of the proposed tool on a real-world application we employ a miniapp developed within the ANTAREX project [S+16], which emulates the workload of the geometric approach to molecular docking. This class of application is useful in the in-silico drug-discovery process, which is an emerging application of HPC, and aims to find the best fitting ligand molecule with a pocket in the target molecule [BCBC13]. This process is performed by approximating the chemical interactions with the proximity between atoms. The evaluation of every ligand molecule-pocket pair is independent with respect to the other pairs. Therefore, we implemented an MPI-based version of the same miniapp. The input dataset is partitioned among the slave processes. The integration of LIBVC required to add or modify a total of 60 lines of code over an original code size of 1300 lines of code, which is less than 5% of the code size.

After the integration the miniapp gained a speedup of $2.44\times$ with respect to the baseline – including the overhead for dynamic compilation. The speedup is achieved by exploiting code specialization on geometrical functions. Although the overhead of performing dynamic compilation on every parallel process increases the whole execution time, the speedup we obtained in the serial version of the miniapp is confirmed also in the parallel case. We run the MPI-based miniapp using 4, 8, 16, and 32 parallel processes. We obtained a speedup of $2.39\times$, $2.24\times$, $1.99\times$, and $1.63\times$ respectively.

2.2 Case Study: OpenModelica Compiler

To demonstrate the effectiveness of LIBVC also on legacy code we employ the C code which is automatically generated by a state-of-the-art compiler for Modelica. Modelica is a widely-used object-oriented language for modeling and simulation of complex systems. OpenModelica [F+06] is an open source compiler for the Modelica language. It translates Modelica code into C code, which is later compiled with `clang` and linked against an external equation solver library.

As test case, we simulated a transmission line model [Cas15] of 1000 elements. We modified the C and Makefile code automatically generated by the OpenModelica compiler to integrate the simulation C source code with LIBVC and properly compile it. It took two hours of work to integrate the automatically generated code with the LIBVC. The integration required to add or modify a total of 65 lines of C code and 5 lines of Makefile code over an original code size of 633390 lines of code, which is less than 0.015% of the code size.

The baseline code took 374.25 seconds before the integration. After the integration the simulation took 295.00 seconds – including the overhead for dynamic compilation – for a speedup of $1.27\times$ with respect to the baseline. The speedup is achieved by recompiling the

C code which implements the model description by using a deeper optimization level (`-O3`) with respect to the default one (`-O0`). In this case, the compilation time that it is spent on optimizations is widely paid back by a faster execution time.

3 Precision Tuning

Approximate computing is an emerging research field. It suggests to trade off accuracy of the computed results in return for faster execution, which generally entails a lower energy-to-solution. Such trade-off can be managed to some extent within the boundaries of the floating point representation by selecting among single, double and quadruple precision representations for the intermediate values of a computation. However, there is also the option to explore fixed point arithmetic, with different bit-widths, as a further step in the trade-off. As fixed-point arithmetic are generally simpler to implement in terms of logical circuits, it is expected that more savings can be achieved. We employ floating to fixed point conversion within a dynamic compilation framework that allows to generate and load at runtime different versions of a kernel, to explore the space of solutions.

3.1 Proposed approach

We propose an approach that enables the programmer to trade off precision for a more efficient implementation of C/C++ application kernels. The main advantage of our contribution is the capability of adjusting the precision level at runtime according to the input data.

Our solution is built upon four main components: an equivalence function defined on the space of the input, a fixed point data type representation, a dynamic compilation library, and a runtime tuning policy.

Equivalence Function Our approach relies on a classification of the input of the application kernel. We perform continuous optimization on the application kernel for each class of input data, which are known only at runtime. The equivalence function defines which inputs of the application kernel can be considered equivalent in terms of time-to-solution and quantization error. We propose a general purpose classification based on the range of the input data. Therefore, we approximate the quantization error and the time-to-solution for the given application kernel as functions of the range of the input data. Although this assumption does not hold in every case, it allows us to target a broad range of HPC application kernels.

Dynamic Compilation The dynamic compilation allows us to explore at runtime how the input data range impacts the precision on the output. When input data range is relatively stable, it is possible to periodically explore the impact of precision and re-tune it. We rely on LIBVC [CA18], which is a C++ library that allows us to exploit any industry-grade compiler to dynamically generate versions of the application.

Tuning Policy We develop a policy which allows us to dynamically select which version to run according to the range of the input data we have to process. We start by considering the floating point version as the baseline version. Our policy keeps a list of versions to be tested and it is invoked every iteration. Every invokation it decides whether to start running

an exploration burst or to defer it. Every exploration burst investigate one or more possible optimum points by running one or more versions among the ones in the list of versions to be tested with the given data range. In the current implementation, each iteration takes one element at random from the list of candidates. It is possible to replace this implementation with more sophisticated alternatives, such as a binary search within the candidate space or by sorting the candidate versions by a pre-characterization.

4 Related Works and Future Directions

Recent works targeting HPC architectures exploit precision tuning of fixed point representations via a template-based C++ library. The approach discribed in [C+17] relies on the same aforementioned C++ library implementation.

The road to support heterogeneous platforms is currently open. Researchers recently explored mixed precision tuning in *OpenCL* kernels. Although the implementation described in [N+18] supports only static tuning, research on dynamic exploration is ongoing.

Future directions include exploiting LIBVC to enable the dynamic re-targeting of a given kernel over multiple architectures via dynamic compilation [A+18].

References

[A+96] Joel Auslander et al. Fast, effective dynamic compilation. *SIGPLAN Not.*, 31(5):149–159, May 1996.

[A+18] Giovanni Agosta et al. Managing heterogeneous resources in hpc systems. In *Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*, PARMA-DITAM '18, pages 7–12, New York, NY, USA, 2018. ACM.

[BCBC13] Andrea R. Beccari, Carlo Cavazzoni, Claudia Beato, and Gabriele Costantino. LiGen: a high performance workflow for chemistry driven de novo design, 2013.

[C+17] Stefano Cherubin et al. Implications of Reduced-Precision Computations in HPC: Performance, Energy and Error. In *International Conference on Parallel Computing (ParCo)*, Sep 2017.

[CA18] Stefano Cherubin and Giovanni Agosta. libVersioningCompiler: An easy-to-use library for dynamic generation and invocation of multiple code versions. *SoftwareX*, 7:95 – 100, 2018.

[Cas15] Francesco Casella. Simulation of large-scale models in modelica: State of the art and future perspectives. In *LINKÖPING ELECTRONIC CONFERENCE PROCEEDINGS*, pages 459–468, 2015.

[F+06] P. Fritzson et al. Openmodelica - a free open-source environment for system modeling, simulation, and teaching. In *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*, pages 1588–1595, Oct 2006.

[GPS15] Davide Gadioli, Gianluca Palermo, and Cristina Silvano. Application autotuning to support runtime adaptivity in multicore architectures. In *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, pages 173–180, July 2015.

[N+18] Ricardo Nobre et al. Aspect-driven mixed-precision tuning targeting gpus. In *Proceedings of the 9th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and the 7th Workshop on Design Tools and Architectures For Multicore Embedded Computing Platforms*, PARMA-DITAM '18, Jan 2018.

[S+16] Cristina Silvano et al. The antarex approach to autotuning and adaptivity for energy efficient hpc systems. In *Proceedings of the ACM International Conference on Computing Frontiers*, CF '16, pages 288–293, 2016.