Dynamic Compilation in HPC

Stefano Cherubin, Giovanni Agosta

{name.surname}@polimi.it

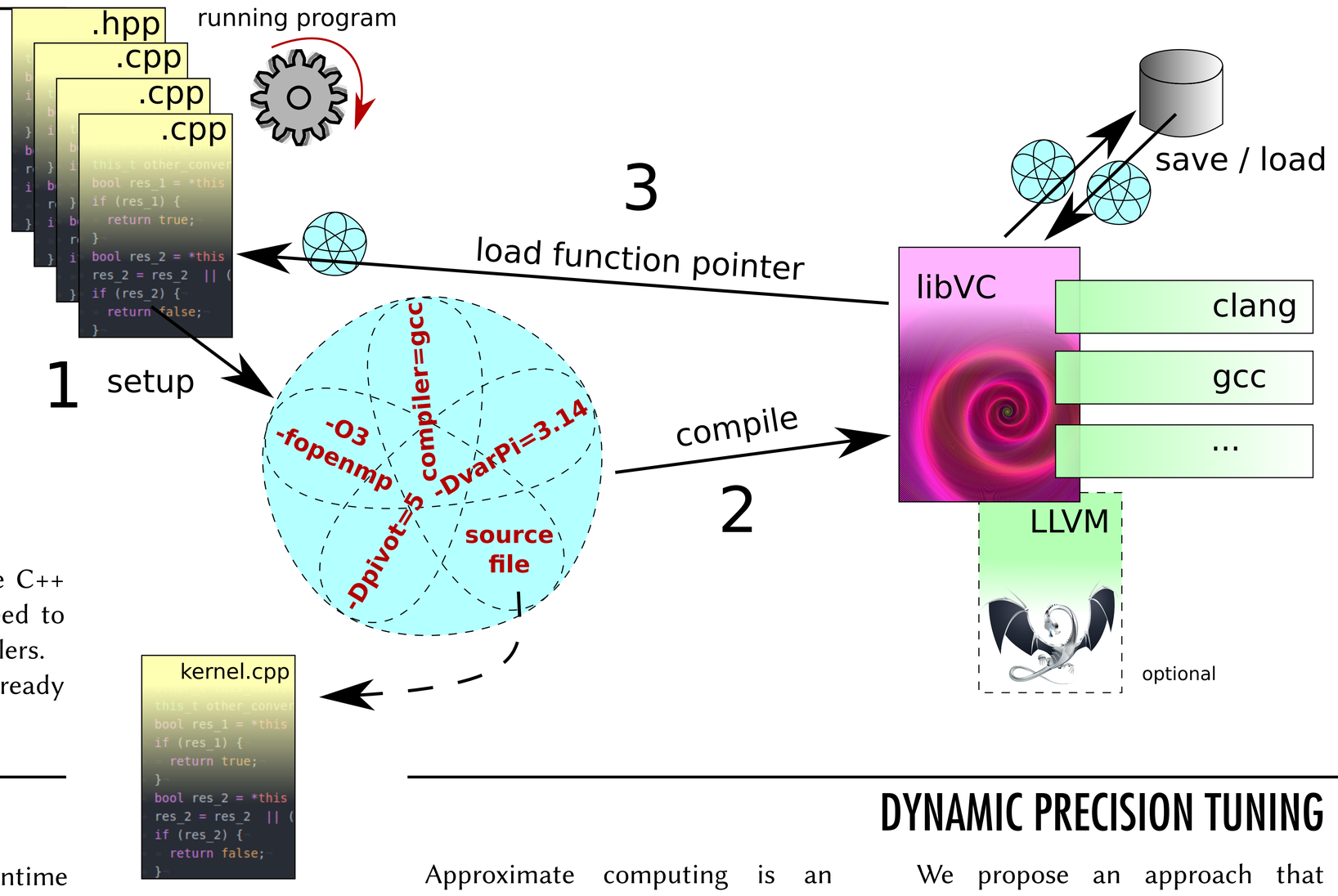
POLITECNICO MILANO 1863

HEAP LAB

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.

GO DYNAMIC!

Dynamic compilation allows source code to be (re-)compiled at runtime. This paradigm allows the compiler to perform extra code specialization based on constant values or particular conditions which are known only at runtime.
Traditionally dynamic compilation techniques require ad-hoc compilation frameworks.
We overcame this issue: now you can have the benefits of these technologies within your C++ applications. Just try it!



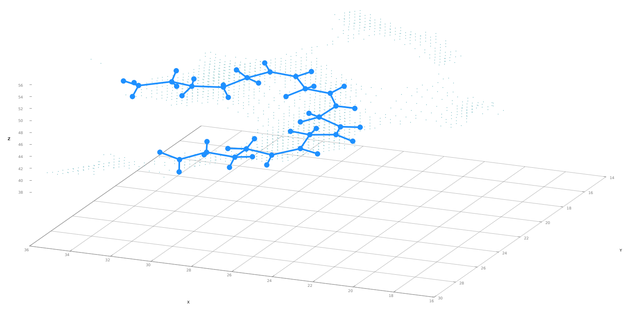
Key Technology: LibVersioningCompiler

LibVersioningCompiler (aka libVC) [1] 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.

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 software maintainability whereas manual implementation is not trivial. Thus, this practice is not widely adopted.
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 [2].

CASE STUDY: Geometrical Docking Miniapp



This application aims to find the best fitting ligand molecule with a pocket in the target molecule [4]. It approximates the chemical interactions between a ligand molecule and the pocket of a target molecule with the proximity between their atoms.
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 (less than 5%).
After the integration the miniapp gained a speedup of 2.44× with respect to the baseline – including the overhead for dynamic compilation.
The speedup is achieved by exploiting code specialization on geometrical functions.

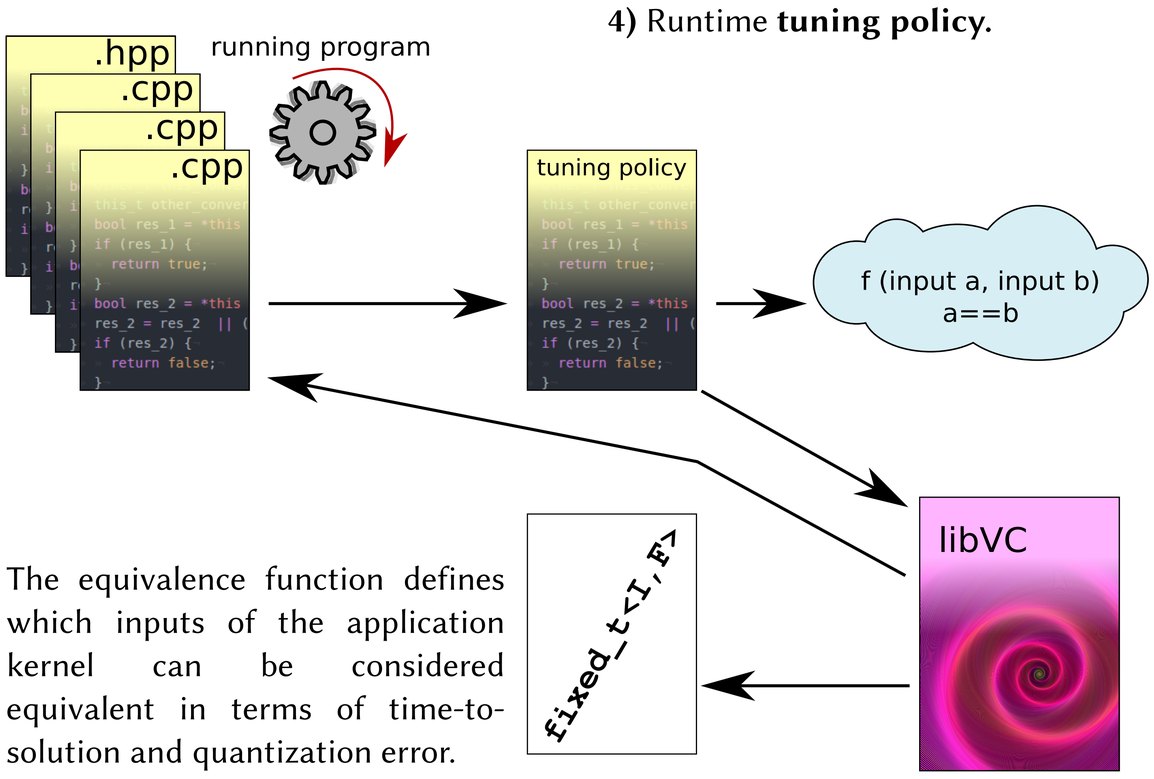
CASE STUDY: OpenModelica Compiler

To demonstrate the effectiveness of libVC on legacy code we employ the C code which is automatically generated by a state-of-the-art compiler for Modelica (OpenModelica [4]).
The integration required to add or modify a total of 70 lines of code over an original code size of 633390 lines of code (less than 0.015%).
After the integration the simulation achieved a speedup of 1.27× with respect to the baseline – including the overhead for dynamic compilation.
The speedup is achieved by recompiling the C code by using a deeper optimization level (-O3) with respect to the default one (-O0).

DYNAMIC 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.
We employ floating to fixed point conversion within a dynamic compilation frame-work that allows to generate and load at runtime different versions of a kernel, to explore the space of solutions.

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:
1) Equivalence function,
2) Fixed point data type,
3) Dynamic compilation,
4) Runtime **tuning policy.**



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. Although this classification is not guaranteed to be exact in every case, it allows us to target a broad range of HPC application kernels.
We employ an approximate computing-oriented library for fixed point data types [3].
We rely on libVC as dynamic compilation framework.

The policy dynamically selects which version to run according to the class of the input data we currently have to process. It keeps a list of versions to be tested for each class of input data, and it is invoked every iteration. Every invokation it decides whether to start running an exploration burst or to defer it.

REFERENCES

[1] S. Cherubin, G. Agosta
libVersioningCompiler: An easy-to-use library for dynamic generation and invocation of multiple code versions
SoftwareX, Volume 7, January–June 2018, Pages 95-100
ISSN 2352-7110. DOI: 10.1016/j.softx.2018.03.006

[2] D. Gadioli, G. Palermo, and C. Silvano
Application autotuning to support runtime adaptivity in multicore architectures
2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation

[3] S. Cherubin. G. Agosta, I. Lasri. E. Rohou, O. Sentieys
Implications of Reduced-Precision Computations in HPC: Performance, Energy and Error
International Conference on Parallel Computing, Sep 2017

[4] A. R. Beccari, C. Cavazzoni, C. Beato, and G. Costantino
LiGen: a high performance workflow for chemistry driven de novo design
Journal of Chemical Information and Modeling, May 2013
DOI: 10.1021/ci400078g

[5] P. Fritzson et al.
OpenModelica - A free open-source environment for system modeling, simulation, and teaching
2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, Oct 2006

ACKNOWLEDGEMENTS

This work is supported by the European Union's Horizon 2020 research and innovation programme, under grant agreement No 671623, FET-HPC ANTAREX.