Par4All: From Convex Array Regions to Heterogeneous Computing

Mehdi Amini[1,2] Béatrice Creusillet[2] Stéphanie Even[2] Ronan Keryell[2]
Onig Goubier[2] Serge Guelton[2] Janice Onanian McMahon[2]
François-Xavier Pasquier[2] Grégoire Péan[2]
Pierre Villalon[2]
[1]MINES ParisTech *firstname.lastname*@mines-paristech.fr
[2]HPC Project *firstname.lastname*@hpc-project.com

Keywords
Heterogeneous computing, convex array regions, source-to-source compilation, polyhedral model, GPU, CUDA, OpenCL.

ABSTRACT
Recent compilers comprise an incremental way for converting software toward accelerators. For instance, the PGI Accelerator [14] or HMPP [3] require the use of directives. The programmer must select the pieces of source that are to be executed on the accelerator, providing optional directives that act as hints for data allocations and transfers. The compiler generates all code automatically.

JCUDA [15] offers a simpler interface to target CUDA from JAVA. Data transfers are automatically generated for each call. Arguments can be declared as IN, OUT, or INOUT to avoid useless transfers, but no piece of data can be kept in the GPU memory between two kernel launches. There have also been several initiatives to automate transformations for OpenMP annotated source code to CUDA [10, 11]. The GPU programming model and the host accelerator paradigm greatly restrict the potential of this approach, since OpenMP is designed for shared memory computer. Recent work [6, 9] adds extensions to OpenMP that account for CUDA specificity. These make programs easier to write, but the developer is still responsible for designing and writing communications code, and usually the programmer have to specialize his source code for a particular architecture.

Unlike these approaches, Par4All [13] is an automatic parallelizing and optimizing compiler for C and Fortran sequential programs funded by the HPC Project startup. The purpose of this source-to-source compiler is to integrate several compilation tools into an easy-to-use yet powerful compiler that automatically transforms existing programs to target various hardware platforms. Heterogeneity is everywhere nowadays, from the supercomputers to the mobile world, and the future seems to be promised to more and more heterogeneity. Thus adapting automatically programs on targets such as multicore systems, embedded systems, high performance computers and GPUs is a critical challenge.

Par4All is mainly based on the PIPS [7, 1] source-to-source compiler infrastructure and benefits from its interprocedural capabilities like memory effects, reduction detection, parallelism detection, but also polyhedral-based analyses such as convex array regions [4] and preconditions.

The source-to-source nature of Par4All makes it easy to integrate third-party tools into the compilation flow. For instance, we are using PIPS to identify parts that are of interest in a whole program, and we rely on the PoCC [12] polyhedral loop optimizer to perform memory accesses optimizations on these parts, in order to exhibit locality for instance.

The combination of PIPS' analyses together and the insertion of other optimiser in the middle of the compilation flow is automated by Par4All using a programmable pass manager [5] to perform whole program analysis, spot parallel loops and generate mostly OpenMP, CUDA or OpenCL code.

To that end, we mainly face two challenges: parallelism detection and data transfer generation. The OpenMP directives generation relies on coarse grain parallelization and semantic-based reduction detection [8]. The CUDA and OpenCL targets add the difficulty of data transfer management. We tackle it using convex array regions that are translated into optimized, interprocedural data transfers between host and accelerator as described in [2].

The demonstration will provide the assistance with a global understanding of Par4All internals compilation flow, going through the interprocedural results of PIPS analyses, parallelism detection, data transfer generation and resulting code execution. Several benchmark examples and some real-world scientific applications will be used as a showcase.

IMPACT 2012
Second International Workshop on Polyhedral Compilation Techniques
Jan 23, 2012, Paris, France
In conjunction with HiPEAC 2012.

http://impact.gforge.inria.fr/impact2012

1. REFERENCES
[1] Mehdi Amini, Corinne Ancourt, Fabien Coelho, Béatrice Creusillet, Serge Guelton, François Irigoin, Pierre Jouvelot, Ronan Keryell, and Pierre Villalon.

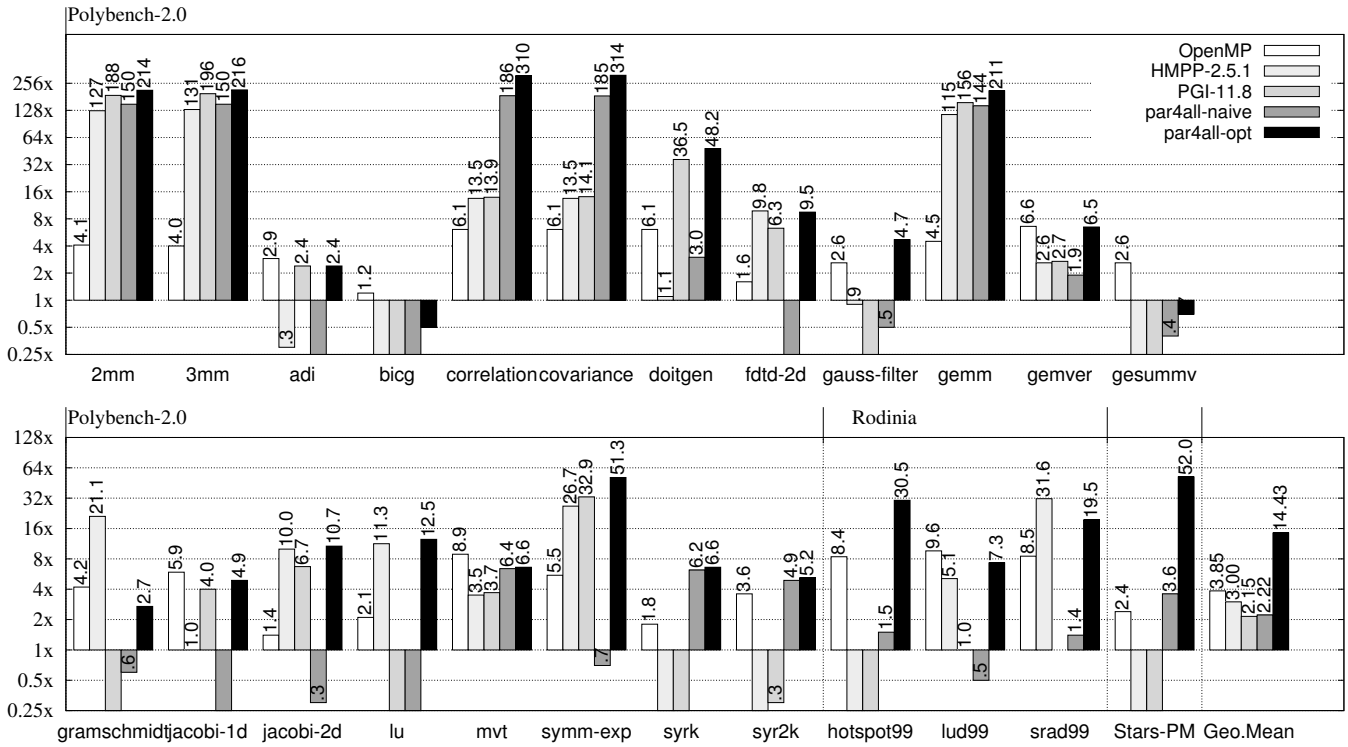

Figure 1: Speedup relative to naive sequential version for an OpenMP version, a version with basic PGI and HMPP directives, a naive CUDA version, and an optimized CUDA version, all automatically generated from the naive sequential code.

PIPS Is not (only) Polyhedral Software. In *First International Workshop on Polyhedral Compilation Techniques*, IMPACT, April 2011.

[2] Mehdi Amini, Fabien Coelho, François Irigoin, and Ronan Keryell. Static compilation analysis for host-accelerator communication optimization. In *Workshops on Languages and Compilers for Parallel Computing*, LCPC, 2010.

[3] Francois Bodin and Stephane Bihan. Heterogeneous multicore parallel programming for graphics processing units. *Sci. Program.*, 17:325–336, December 2009.

[4] Béatrice Creusillet and Francois Irigoin. Interprocedural array region analyses. *International Journal of Parallel Programming*, 24(6):513–546, 1996.

[5] Serge Guelton. *Building Source-to-Source compilers for Heterogenous targets*. PhD thesis, Télécom Bretagne, 2011.

[6] Tianyi David Han and Tarek S. Abdelrahman. hiCUDA: a high-level directive-based language for GPU programming. In *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, pages 52–61, New York, NY, USA, 2009. ACM.

[7] François Irigoin, Pierre Jouvelot, and Rémi Triolet. Semantical interprocedural parallelization: an overview of the PIPS project. In *International Conference on Supercomputing*, ICS, pages 244–251, 1991.

[8] Pierre Jouvelot and Babak Dehbonei. A unified semantic approach for the vectorization and parallelization of generalized reductions. In *International Conference on Supercomputing*, ICS, pages 186–194, 1989.

[9] Seyong Lee and Rudolf Eigenmann. OpenMPC: Extended OpenMP programming and tuning for GPUs. In *SC '10*, pages 1–11, 2010.

[10] Seyong Lee, Seung-Jai Min, and Rudolf Eigenmann. OpenMP to GPGPU: a compiler framework for automatic translation and optimization. In *PPoPP*, 2009.

[11] Satoshi Ohshima, Shoichi Hirasawa, and Hiroki Honda. OMPCUDA : OpenMP execution framework for CUDA based on omni OpenMP compiler. In *Beyond Loop Level Parallelism in OpenMP: Accelerators, Tasking and More*, volume 6132 of *Lecture Notes in Computer Science*, pages 161–173. Springer Verlag, 2010.

[12] Louis-Noel Pouchet, Cédric Bastoul, and Uday Bondhugula. PoCC: the Polyhedral Compiler Collection, 2010. `http://pocc.sf.net`.

[13] HPC Project. Par4All initiative for automatic parallelization. `http://www.par4all.org`, 2010.

[14] Michael Wolfe. Implementing the PGI accelerator model. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, GPGPU, pages 43–50, New York, NY, USA, 2010. ACM.

[15] Yonghong Yan, Max Grossman, and Vivek Sarkar. JCUDA: A programmer-friendly interface for accelerating Java programs with CUDA. In *Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, 2009.