



FP6 Project IST-034869
Deliverable D3.6

List of Relevant Loop Transformations

IST-034869

Public

Project Number	:	IST-034869
Project Title	:	Advanced Compiler Technologies for Embedded Streaming
Deliverable Type	:	List

Deliverable Number	:	D3.6
Title of Deliverable	:	List of relevant loop transformations
Nature of Deliverable	:	Public
Internal Document version	:	1.1
Contractual Delivery Date	:	30 November 2006
Actual Delivery Date	:	15 January 2007, updated 26 June 2007
WP(s)	:	WP 3 High-level transformation parameterised by ASM
Author(s)/Affiliation	:	Albert Cohen (INRIA)

Table of Contents

Table of Contents	2
0 Introduction	3
1 References	4
2 Detailed list.....	6
2.1 Loop tiling or blocking or partitioning	6
2.2 Loop fusion or merging, and loop fission or distribution.....	6
2.3 Loop pipelining	6
3 Important ongoing and future work.	7

0 Introduction

The goal of the ACOTES project is to substantially improve the programmer's efficiency in developing applications that process massive streams of data on programmable, parallel embedded architectures. The target application areas of ACOTES are video processing and advanced radio communications for consumer systems.

Based on the SPM and the ASM, the work in WP3 (High level transformations parameterised by Abstract Streaming Machine) will partition the streaming application at the functional block level, and also identify and implement optimisation strategies at this level.

This document discusses relevant loop transformations.

1 References

We give a list of important references for the ongoing work on loop-level, data-parallelism and locality optimization in WP3.

- R. Allen and K. Kennedy, *Optimizing Compilers for Modern Architectures*, Morgan and Kaufman (publisher), 2002. (<http://portal.acm.org/citation.cfm?id=502981>)
Almost all loop transformations described in this book are of interest for ACOTES. We will focus on the locality-enhancing ones, and to the transformations that enable vector parallelism extraction. We will also extend some of these transformations to parallel streaming programs, allowing to optimize across parallel constructs.
- *High-Level Loop Optimizations for GCC*: GCC Developer's Summit, 2004. (http://www.gccsummit.org/2004/view_abstract.php?content_key=9)
State-of-the-art loop transformations in GCC. The scalar-oriented ones (induction variables) are very mature, but the ones targeting locality-enhancement, vectorisation or ILP are still in a preliminary stage, even in 2007.
- *GRAPHITE: Loop optimizations based on the polyhedral model for GCC*. GCC Developer's Summit, 2006. (http://www.gccsummit.org/2006/view_abstract.php?content_key=42)
First announcement and roadmap to implement a very innovative LNO into GCC. In a first version, the project will focus on single-threaded optimization (e.g., for vectorisation and locality).
- *Semi-Automatic Composition of Loop Transformations for Deep Parallelism and Memory Hierarchies*. International Journal of Parallel Programming, 2006. (<http://portal.acm.org/citation.cfm?id=1165153>)
This article provides a list of advanced loop transformations, with several examples, as well as the mapping of sequences of these transformations to a polyhedral representation of the program. It is a recent survey on polyhedral compilation, describing several achievements in the composability and scalability of those techniques. A preprint of the article is provided for internal use on the ACOTES Wiki. The references of this article are a rich source of technical information on the polyhedral model, from foundations to diverse applications (pioneering articles by Feautrier are still the reference on multidimensional affine scheduling). In addition, a list of references is also provided on stream-computing which involve a rich set of loop-level optimizations (for sequential and parallel code).
- *Space-Time Scheduling of Instruction-Level Parallelism on a Raw Machine*, ASPLOS 1998. (<ftp://ftp.cag.lcs.mit.edu/pub/raw/documents/Lee-ASPLOS-1998.pdf>)
- *Optimizing Compiler for the Cell Processor*, PACT 2005. (<http://cag.csail.mit.edu/crg/papers/eichenberger05cell.pdf>)
- *Exploiting Coarse-Grained Task, Data, and Pipeline Parallelism in Stream Programs*, ASPLOS 2006. (<http://cag.csail.mit.edu/commit/papers/06/gordon-asplos06.pdf>)
- *Sequoia: Programming the Memory Hierarchy*, Supercomputing 2006. (http://graphics.stanford.edu/~kayvonf/papers/sequoia_sc06.pdf)
See also PPOPP'07 paper on Sequoia.

These references should be considered particularly useful when extending the WP3 tools to a full-fledged restructuring of the parallelism exposed in the source program (through the programming model of WP2). In the short term, we propose to follow a pragmatic approach where thread-level parallelism issues (distributed and heterogeneous) will be considered in source-to-source tools, and decoupled from the rest of the compilation. Nevertheless, towards

the end of the project, all loop transformations should work on parallel code, leveraging the high-level, well defined semantics of the streaming constructs.

Other work on loop transformations targeted cost models and heuristics. They are not directly relevant for this deliverable.

See also the two HiPEAC GCC tutorials and the slides and patch about Kennedy and Allen's loop distribution algorithm: <http://www.hipeac.net/node/745>

2 Detailed list

In the short term, we thus address the sub problem associated with the exploitation of Data-Level Parallelism (DLP) and Instruction-Level Parallelism (ILP) in the middle-end and back-end compiler. We focus on media and signal processing. Although we assume that thread-level parallelism has been extracted, mapped and balanced at earlier stages of the compilation flow, we recognize the importance of modelling the full system when tuning single-threaded optimizations.

The three following transformations are the main ones of interest for ACOTES. They are detailed, formalized and applied to case studies in the above-mentioned references. We only list their specific occurrence in the context of the ACOTES project.

2.1 Loop tiling or blocking or partitioning

Partitioning the iteration space in blocks, and staging the traversal of this space, iterating first within then among blocks (like blocked matrix operations). The main application in ACOTES will be to coarsen the grain of parallelism of fine-grained tasks; it may also marginally help to improve locality (much less than on scientific codes).

2.2 Loop fusion or merging, and loop fission or distribution

Transforming a loop on A followed by a loop on B into a loop on A;B, and conversely. This has several caveats when loops have dependences and/or different trip counts. The main usage is to coarsen the grain of parallelism (fusion) or to enable vectorisation (distribution). Loop fusion is typically interesting when coupled to a higher-level merging of producer-consumer parallel tasks: it yields high savings in memory footprint (temporary arrays) and improves vastly memory locality.

2.3 Loop pipelining

A multi-level, non-perfectly nested form pipelining can be defined. It serves both as an enabling transformation to perform subsequent fusion or tiling, and as a parallelism-enhancing transformation at the vector or instruction level. Once again, it may be coupled with higher-level pipelining of streaming tasks, as proposed in the above-mentioned ASPLOS'06 paper.

3 Important ongoing and future work.

- The basic support for loop optimizations exists in GCC. Yet the GRAPHITE project to provide the middle-end of the compiler with a polyhedral representation will take several months until it reaches the maturity of the previous research prototypes developed at INRIA. It has finally reached a stage where collaborative work on is possible, shortly before milestone M3.1. Independent developers outside the project expressed interest in the project and started studying the code, but did not contribute anything yet.
- Building a cost model for these optimizations is a well known challenge, especially when sequences of transformations are considered. We will mostly follow a pragmatic approach, based on state-of-the-art, proven scalable heuristics, and based on the ASM design in WP2. Yet we will also consider more advanced methods based on iterative optimization or machine learning compilation, and integrate them if they mature enough (through independent developments, not sponsored by ACOTES).
- Towards the end of the project, the main goal is to extend GRAPHITE to support optimization of parallel code. Management of data-parallelism is well understood in the polyhedral model, but this is not the case for task and stream-oriented parallelism. As a first pragmatic step, we are studying how GRAPHITE can facilitate the generation of multiple parallel tasks from a single specification following the programming model of WP2. Technically, the loop fusion, tiling, pipelining and code generation techniques embedded into GRAPHITE can be easily reused and applied to the main corresponding optimization of streaming constructs. As a target, we wish to reproduce the optimizations demonstrated with StreamIt in the above-mentioned ASPLOS'06 paper.