Paper Presentation Notes

Pan An (A0134556A)

April 27, 2016

Contents

1 OPGEN

Every compiler has its optimizing rules. These rules, most of the time, are made by compiler developers with a good mathematical background. Hand crafted rules, however, can never guarantee correctness and completeness.

OPGEN solves this problem by generating all the local optimizations. Optgen generates optimization rules that work on the IR level. This allows a more abstract view on the program behavior than working on assembly level. The basic idea of OPGEN is to generate all instruction blocks and compares the output. If the generated expressions pass all the test cases AND at the same time has a smaller cost, these expressions will be selected.

[$N = 100$] [$N = 1000$] [Riferimento]

Testing of OPGEN on modern compilers has helped identifying more than 50 missing optimizations.

2 Exploring and Enforcing Security Guarantees via Program Dependence Graphs

PIDIN is a program analysis and understanding tool that enables the specification and enforcement of precise application-specific information security guarantees It enables regression testing of information security guarantees.

PIDIN uses program dependence graphs to precisely and intuitively capture the information flows within an entire program and a custom PDG query language to allow the exploration, specification, and enforcement of information security guarantees. Developers can understand how information flows within a program and express precise security guarantees.

3 GPUCC

GPUCC is an open-source gpgpu compiler. More precisely, an LLVM-based, fully open-source, CUDA compatible compiler for high performance computing.

The paper focuses on Clang frontend, IR generator and host code generator. For CUDA frontend, gpucc has the capacity to compile mixed code. gpucc increases the function inlining threshold to 1100, which we select based on the overall effects on our benchmarks.

It was designed as a competitive substitute of NVIDIA nvcc. The results of the paper show that gpucc produces code that is faster than or on par with nvcc. It also produces some modern features in modern compilers.

4 PSLP: Padded SLP Automatic Vectorization

Writing code that makes efficient use optimization methods like vectorizations and SIMDizations is hard and leads to platform-specific implementations. PSLP is a vectorization algorithm that can vectorize code containing non-isomorphic instruction sequences.

It injects a small number of redundant instructions into the code to transform non-isomorphic sequences into isomorphic ones. The padded instruction sequence can then be successfully vectorized.

The experiments shows that PSLP can decrease the execution time by up to 63%.

PSLP solves a major problem of existing SLP vectorization algorithms, without having to worry about the instruction sets.

5 Optimizing Large Programs at Small Price

Optimization of a large program during compiling might cost a lot of time and memory. This paper presents the analysis of the causes of the difficulty for optimization on large scale programs.

The common motive behind the unscalability problem is the large number of statements within a single static control part, which also amounts to large number of dependences.

The proposed method for solving this problem is through Statement Condensation and Dependence Condensation algorithms. Where the program is then represented with just super-statements and representative dependences, which still allows to correctly reason about the application of any particular loop optimization.

In order to speed up the construc- tion of PDG while not sacrificing precision, this paper presented scalefuse, to this problem. Scalefuse represents the entire hot region with a smaller set of statements (and dependences) that is sufficient for the compiler to perform loop optimizations. The proposed method however, might not be able to be very effective in solving the problem.

6 HELIX

HELIX is an automatic parallelization of irregular programs for chip multiprocessing. It was designed for loop parallelization that assigns successive iterations of a loop to separate threads.

HELIX applies code transformations to minimize the inefficiencies of sequential segments, data transfer, signaling, and thread management. It works on an intermediate representation of a program. Experimental results show that by reducing communication costs and increasing thread level parallelism, HELIX achieves a mean speedup of 2.25 (with a maximum of 4.12) over a broad range of applications on a real six-core system.

HELIX is fully automatic. It does not depend on source code annotations or modifications, or other forms of human intervention.

7 Reactive Tiling

This paper focusing on the discussion of manipulation and management of multicore system resources. In the real life application development, most of the effort were spent focusing on the OS and hardware problems. This paper shows a way for applications to react to OS and hardware management of the resources.

For a typical matrix tiling problem. The proposed system will generate code pieces for different tile sizes. Then it determines the best tile sizes for the cache allocations using curve-fitting. Safe point analysis was introduced in order to make sure the switching between different tile sizes are conducted seamlessly.

Our experimental results show that the proposed scheme improves the performance of applications by 8.4%, on average.

8 Symbolic Range Analysis of Pointers

Pointer analysis is one of the most fundamental techniques for compilers and stack based language program designs. Mainstream compilers still struggle to distinguish intervals within the same array.

This paper provides combination of pointer analysis with range analysis on the symbolic interval lattice including global and local pointer disambiguation. An abstract domain is introduced in order to associates pointers with symbolic ranges. The symbolic analysis handles the subtleties of pointer arithmetic and memory indexation in a theoretical framework.

The analysis framework achieved precision in the proposed algorithm by combining alias analysis with classic range analysis on the symbolic domain.