CS6250 Final Exam

Chinmay Chandak, Tapan Sahni

December 14, 2019

Abstract

The paper provides a detailed experimentation of Loop Unrolling and Function Inlining on the Samsung Reconfigurable Processor. The performance improvement due to the above optimization has been tested on two benchmarks. Experimental details shows 83% and 11% gain on performance for the 2 benchmarks when compared to the original code. We also look at the recent work in loop optimization on SRC.

1 Introduction

The Samsung Reconfigurable Processor (SRP) has the flexibility of providing application-specific hardware configuration. The SRP has two modes for running applications: The computationally code-critical sections of application software run on the Coarse Grain Reconfigurable Array (CGRA) mode of SRP and the non-code critical sections of software run on Very Large Instruction Word (VLIW) mode of SRP.

Though the SRP has multiple Functional Units, the maximal performance of the application depends on the ILP possible in the application and modulo scheduling (software pipelining) to exploit the Loop Level Parallelism. Application-specific code needs to be optimized in each module. Function inlining and loop unrolling are well known optimizations applied for this purpose.

As the application software is becoming more modular, there tend to be more and more function calls for sets of operations. However, if the loop has function calls inside, it will be disqualified for the CGRA mapping due to software pipelining exploration limitations in SRP. Hence, the functional calls inside innermost loop bodies have to be inlined to enable the CGRA mapping. Thus, in SRP compiler, only innermost loops containing no function calls are candidates for CGRA mapping on the SRP. This limitation can be quite severe as most application programmers prefer a modular style of coding.

2 Loop Unrolling and Inlining

Loop unrolling can be used to improve the performance of the loops either to increase the ILP of loops to run in VLIW mode or to increase LLP opportunity of loops to run in CGRA mode to explore the modulo scheduling on a larger loop body. If the intra-iteration of the loop doesnt have sufficient ILP, the ILP can be exploited with inter-iterations either through software pipelining or through loop unrolling.

There were two major challenges to overcome for SRP specific inlining, in order to gain considerable improvement through function inlining. The first challenge is, to consider the loops which contain call site, which adhere to the constraints imposed, for mapping loops to CGRA, such as innermost loop. The second challenge is to map the inlined code to CGRA. SRP compiler uses the LLVM infrastructure as front-end to leverage on the optimizations supported by LLVM.

SRP selects the call-sites for inlining, if callee is a leaf function and if the call site is in innermost loop. Cost corresponding to these, SRP applicable call-sites, is calculated and compared with the threshold limit identified for inlined function. The selected call-sites are finally inlined using the LLVM inlining utility.

Inlining is performed with the following limitations:

- inlining across multiple compilation units.
- inlining recursive procedures.
- indirect calls using function pointers.
- callee definitions, which calls a function, which is not recursive.

It is evident from the state-of-the-art loop unrolling that, the loop unrolling factor is very loop specific. Moreover, it is threshold based. If the loop size is big (150 instructions), it couldnt unroll the loop by default. But, the SRP have more number of functional units and it can support more operations than the number of functional units in its configuration memory. Hence, this restriction is not applicable for the SRP. If the loop is not rotated, the present LLVM loop unrolling pass ignores the loop to unroll. However, as SRP supports predicate instructions, this can be handled with conditional predicate instructions and the loop still qualifies for CGRA mapping.

In the VLIW Mapping of Loop, the unrolled loop shows the improved performance gains only upto a certain unroll factor. This is due to the sub-optimal selection of VLIW instructions by the compiler when the unrolled code becomes too huge.

It is known that the increasing unroll factor will affect the number of stages in the software pipelining of CGRA loops data flow graph of operations and also the size of prologue and epilogue of software pipeline. If the unroll factor becomes large, the performance can be degraded and it depends on the size of prologue and epilogues, and the loop iterations. When the number of iterations is very large, the effect of prologue and epilogue are dominated by the high loop iterations, and hence no negative gains in performance are seen.

On the other hand, the higher prologue and epilogue sizes for the higher unrolling factor dominate the software pipeline kernel execution if there are less number of loop iterations. Hence, we have seen negative performance gains after increasing the unroll factor to a certain extent. The higher unrolling factor also increases the complexity of CGRA mapping due to larger loop body. Hence, it may provide sub-optimal results for the loop with higher unrolling factor due to the CGRA mapping exploration limitation of SRP compiler.

3 Recent Improvements on SRP Loop Pipelining

The computation-intensive portions of applications, i.e., loops, are often implemented on CGRAs for acceleration. The loop pipelining techniques are usually used to exploit the parallelism of loops. However, for nested loops, the existing loop pipelining methods often result in poor hardware utilization and low execution performance. To tackle this problem, Shouyi Yin, Dajiang Liu [2] proposed the following:

- They proposed the use of affine transformation to facilitate nested loop pipelining.
- Based on polyhedral model, they presented a precise and general formulation of the nested loop pipelining problem on a CGRA.
- They designed a joint affine transformation and multipipeline merging approach to improve the performance of nested loop on CGRA.

The experimental results show that our approach can improve the performance of nested loops up to 35% on average, compared with the state-of-the-art techniques that included loop unrolling and inlining improvements.

Jeongho Nah, Jun Lee [3] introduced a vectorization technique to translate a coalesced loop into a vectorized version. Their compiler maps the parallel loop into the CGRA mode and translates the code to a vectorized version. We have shown the effectiveness of our OpenCL optimizing They evaluated its performance using 17 kernel programs. Evaluation results showed that their approach is effective and promising for the SRP.

4 Conclusion

It can be said that although the SRP faces the semantic constraints and limitations that LLVMs optimization passes face, the multiple Functional Units, capacity of software pipelining which is enhanced in the CGRA mode, the speeding up of conventional, time-consuming, expensive invocations and instructions in the VLIW mode, and the provision of conditional predicate instructions make it more efficient (in terms of speed and power consumption) than the normal production processors.

References

[1] N. R. Miniskar, P. S. Gode, S. Kohli, D. Yoo, Function inlining and loop unrolling for loop acceleration in reconfigurable processors *Proc. Int. Conf. Compilers Archit. Synth. Embedded Syst., pp. 101-110, 2012.*

[2] Shouyi Yin, Xianqing Yao, Dajiang Liu, Leibo Liu, Shaojun Wei, Memory-Aware Loop Mapping on Coarse-Grained Reconfigurable Architectures *Very Large Scale Integration (VLSI) Systems IEEE Transactions on, vol. 24, pp. 1895-1908, 2016, ISSN 1063-8210..*

[3] J Nah, J Lee, H Kim, J Lee, SJ Hwang, An OpenCL optimizing compiler for reconfigurable processors, *2013 International Conference on Field-Programmable Technology (FPT)*