Loop Optimizations for Reconfigurable Architectures

Ozana Silvia Dragomir*,[1],
Todor Stefanov*,[1], Koen Bertels*,[1]

* *TU Delft, Mekelweg 4, 2628CD, Delft, The Netherlands*

ABSTRACT

Loops are an important source of optimization. In this paper, we show how traditional loop transformations such as loop unrolling and loop shifting can be applied in the context or reconfigurable computing. By applying unrolling and shifting to a loop containing a hardware kernel, we relocate the function calls contained in the loop body such that in every iteration of the transformed loop, software functions (running on GPP) execute in parallel with multiple instances of the kernel (running on FPGA). We illustrate the transformations and present experimental results for a loop extracted from MPEG2 encoder containing the DCT kernel. The maximum speedup that can be achieved is 19.65x over the software execution, using the combination of unrolling and shifting.

KEYWORDS: loop optimizations, reconfigurable computing

1 Introduction

In many real life applications, loops represent an important source of optimization. A number of loop transformations (such as loop unrolling, software pipelining, loop shifting, loop distribution, loop merging, or loop tiling) can be used successfully to maximize the parallelism inside the loop and improve the performance. The applications we target in our work have loops that contain kernels inside them. One challenge we address is to improve the performance for such loops, by applying standard loop transformations such as the ones mentioned above. We also keep in mind that there are loop transformations that are not beneficial in most compilers because of the large overhead that they introduce when applied at instruction level, but at a coarse-level (*i.e.*, function), they show a great potential for improving the performance.

Loop unrolling is traditionally used to eliminate loop overhead, improving cache hit rate and reducing branching by replicating the loop body. We use unrolling to expose the loop parallelism, allowing us to execute concurrently multiple kernels on the reconfigurable hardware.

Loop shifting is a transformation that moves operations from one iteration of the loop body to the previous iteration. The operations are shifted from the beginning of the loop body to the end of the loop body and a copy of these operations is also placed in the loop

[1]E-mail: {O.S.Dragomir, T.P.Stefanov, K.L.M.Bertels}@tudelft.nl
This work is supported by the FP6 EU project hArtes, with integrated project number 035143.

Table 1: General and Molen-specific assumptions

Loop nest ⋆ no data dependencies between different iterations; ⋆ loop bounds are known at compile time; ⋆ loops are perfectly nested;
Memory accesses ⋆ memory reads in the beginning, memory writes in the end; ⋆ on-chip memory shared by the GPP and the CCUs is used for program data; ⋆ all necessary data are available in the shared memory; ⋆ all transactions on shared memory are performed sequentially; ⋆ kernel's local data are stored in the FPGA's local memory, not in the shared memory;
Area & placement ⋆ shape of design is not considered; ⋆ placement is decided by a scheduling algorithm such that the configuration latency is hidden; ⋆ interconnection area needed for CCUs grows linearly with the number of kernels.

prologue. In our research, loop shifting means moving a function from the beginning of the loop body to the end and we use it to eliminate the data dependencies between software and hardware functions, allowing concurrent execution on the GPP and FPGA.

2 Study case

We study the effect on performance of applying loop unrolling and loop shifting to a loop nest containing the DCT kernel. Our target architecture is Molen [VWG+04], which allows running multiple kernels/applications at the same time on the reconfigurable hardware. The unroll factor is computed (at compile time) taking into consideration profiling information about memory transfers, execution times for the kernel in hardware and in software (in GPP cycles), area requirements for the kernel, and memory bandwidth. Our assumptions regarding the application and the framework are summarized in Table 1.

The original loop. The loop in Fig. 1a) has been extracted from the MPEG2 encoder multimedia benchmark. It consists of two functions: `CPar` — which is executed always on the GPP — and `DCT`, which is the application's kernel and will be executed on the reconfigurable hardware in order to speed up the application.

In each iteration, data dependencies between `CPar` and `DCT` exist. In this case, `CPar` is the code that computes the parameters for the kernel instance to be executed in the same iteration.

Transforming the loop with loop unrolling ([DMPBW08]). The loop in Fig. 1a) contains **no** data dependencies between `DCT(`i`)` and `DCT(`j`)`, for any iterations i and j, $i \neq j$, therefore loop unrolling can be applied. Figure 1b) presents a simplified case of applying the unrolling method, when $N \mod u = 0$. Each iteration consists of u sequential executions of the function `CPar()` followed by the parallel execution of u kernel instances (there is an implicit synchronization point at the end of the parallel region).

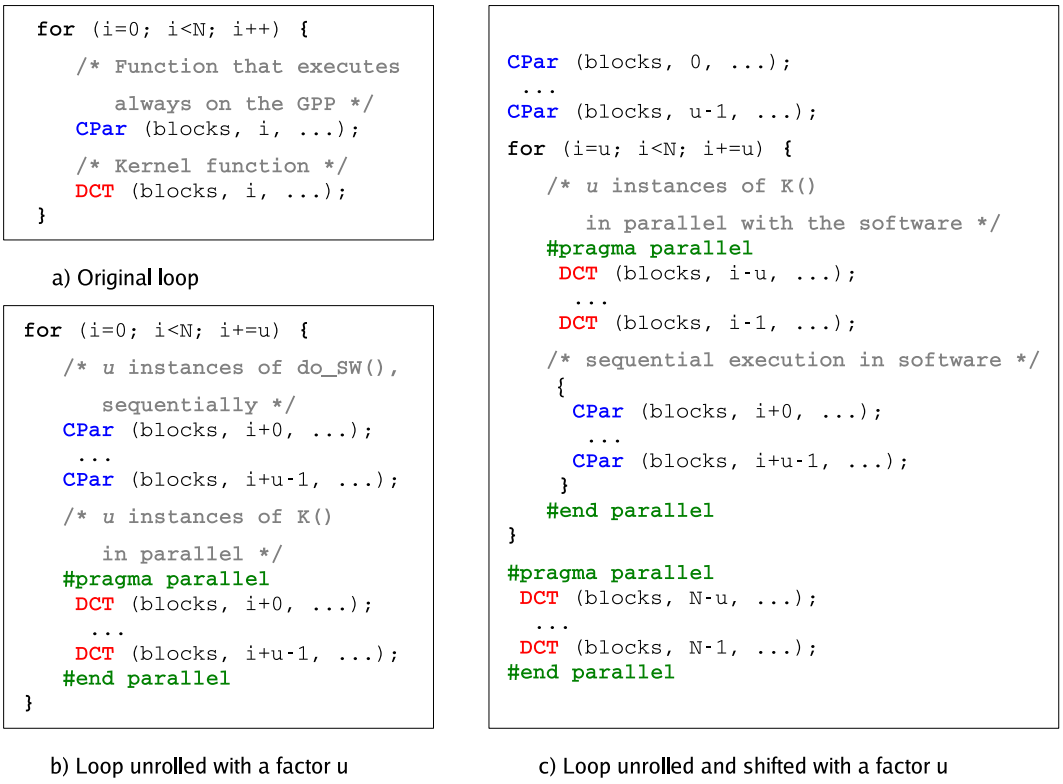

Figure 1: Loop containing a kernel call

Transforming the loop with loop unrolling and shifting ([DSB08]). In order to perform loop shifting, one more constraint needs to be satisfied: there should be **no** data dependencies between `CPar(`i`)` and `DCT(`j`)`, for any iterations i and j, $i \neq j$. This condition is satisfied by the loop in Fig. 1a).

The loop unrolling is extended in Fig. 1c) by shifting the software part of the loop to the end of the loop body, such that in each iteration u sequential executions of the function `CPar` are executed in parallel with u identical kernel instances. The loop body has one iteration less than in the previous case (when applying only unrolling), as the first u calls of `CPar` are executed before the loop d the last u kernel instances are executed after the loop body.

Experimental results. The loop nest presented in Example 1a) containing the DCT kernel (2-D integer implementation) was extracted from MPEG2 encoder multimedia benchmark and executed on the Virtex II Pro board. The VHDL code for DCT was **automatically** generated with DWARV [YKB+07] tool. The experiment was performed with one instance of the kernel running on the FPGA and the execution times have been measured using the PowerPC timer registers. The results for the execution time of the loop for higher unroll factors were computed based on the profiling information.

Figure 2 presents the speedup obtained for different unroll factors, for both unrolling and unrolling plus shifting. The maximum speedup is achieved for loop unrolling and shifting with unroll factor $u = 8$. As it can be seen from the graph, the theoretical performance for the maximum unroll factor tends to be the same for both techniques, as $S_{\text{loop}}(u)$ is a monotonously increasing function and $S_{\text{shift}}(u)$ oscillates within 10% of the maximum value (19.65x) for $u \geq 8$.

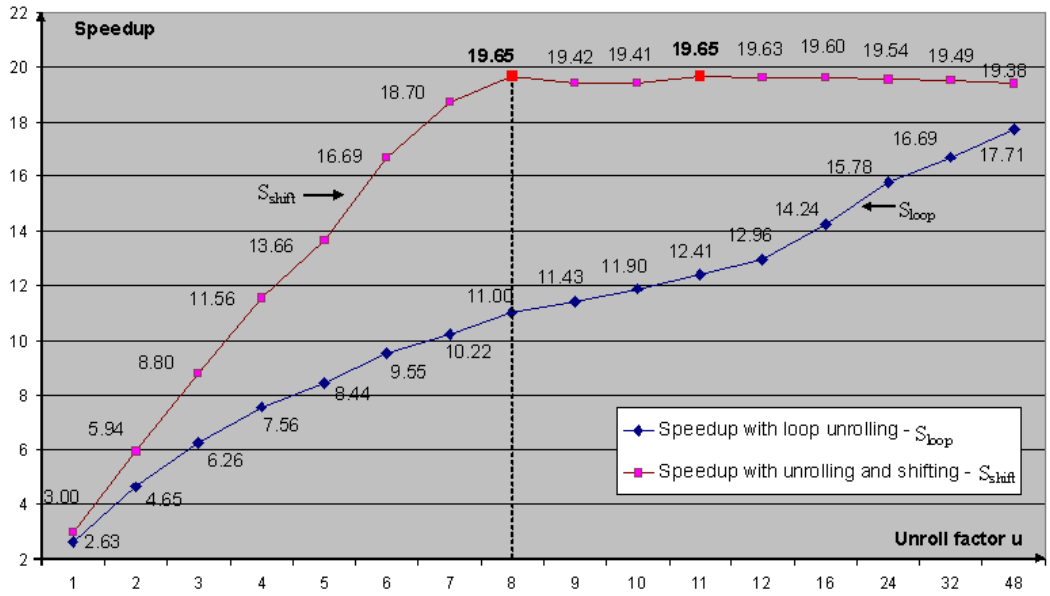

Figure 2: Speedup with loop unrolling and shifting vs. loop unrolling

3 Conclusion

The results achieved for a loop nest extracted from a real-life application (*e.g.*, the MPEG2 encoder) show that loop unrolling and loop shifting are suitable for reconfigurable architectures. Our method that transforms loops automatically with loop unrolling and/or loop shifting (if constraints are met) decreases the time for design-space exploration and exploits the architectural capabilities more efficiently. The input data consists of profiling information about area utilization, memory transfers and execution times in software and in hardware for the kernel implementation. An advantage of this method is that it can be applied to any kernel hardware implementation, from automatically-generated to aggressively-optimized VHDL code.

References

[DMPBW08] Ozana Silvia Dragomir, Elena Moscu-Panainte, Koen Bertels, and Stephan Wong. Optimal unroll factor for reconfigurable architectures. In *Proceedings of the 4th International Workshop on Applied Reconfigurable Computing (ARC'08)*, pages 4–14, March 2008.

[DSB08] Ozana Silvia Dragomir, Todor Stefanov, and Koen Bertels. Loop unrolling and shifting for reconfigurable architectures. In *Proceedings of the 18th International Conference on Field Programmable Logic and Applications (FPL'08) (to appear)*, September 2008.

[VWG+04] Stamatis Vassiliadis, Stephan Wong, Georgi Gaydadjiev, Koen Bertels, Georgi Kuzmanov, and Elena Moscu Panainte. The MOLEN polymorphic processor. *IEEE Transactions on Computers*, 53(11):1363–1375, November 2004.

[YKB+07] Y. D. Yankova, G. Kuzmanov, K. Bertels, G.N. Gaydadjiev, J. Lu, and S. Vassiliadis. DWARV: DelftWorkbench automated reconfigurable VHDL generator. In *Proceedings of the 17th International Conference on Field Programmable Logic and Applications (FPL'07)*, pages 697–701, August 2007.