Fine-tuning Loop-level Parallelism for Increasing Performance of DSP Applications on FPGAs

Emre Özer, Andy P. Nisbet, and David Gregg

Department of Computer Science
Trinity College, Dublin, Ireland
e-mail: {emre.øzer, andy.nisbet, david.gregg}@cs.tcd.ie

Abstract

This paper discusses the balance between loop-level parallelism and clock rate for enhancing the performance of DSP applications fully implemented on FPGAs. Loop-level parallelism reduces the total cycles of an application at the cost of increased routing complexity that often results in lower clock rates. We analyze loops that can be fully parallelized and show that it is possible to achieve better performance by controlling the number of parallel iterations of the loops than using fully parallel loops. We have implemented loop parallelism in our compilation framework and fine-tune them to enhance the performance of DSP applications that target Xilinx Virtex-II FPGA chip. Our experimental results show that it is possible to reach a performance equilibrium point where the total number of cycles and the overall clock frequency can be adjusted to maximize the overall performance of an application.

1. Introduction

FPGAs have the potential to execute an application faster than general-purpose microprocessors because they can be customized to match the computational requirements of the application. It is possible to speed up computationally intensive programs such as DSP applications implemented onto FPGAs due to available loop-level parallelism existing in DSP applications. The overall clock speed of the FPGA may decrease as the parallelism extracted from a program increases because the increased complexity of routing the design may result in longer routing delays. There are two reasons why this situation occurs. 1) When the iterations of loops are executed in parallel, each iteration is instantiated separately at the same logic level or cycle. This increase in the FPGA area usage may lengthen the clock period as the router has less flexibility in the routing strategies. 2) Several functional units to be used in the same cycle are instantiated to extract sufficient parallelism from the program. Although there is no communication between the functional units at the same level or cycle, the logic blocks at earlier levels may need to communicate with the later levels and a wide logic level in between may lead to longer routes. This long routing delay may change the critical path of the final circuit. As a result, the minimum clock cycle period to drive the FPGA increases. Even though the total number of cycles is reduced with higher parallelism, the total execution time can be slower than the unoptimized program because of the increased clock cycle period.

Our goal is to maximize the performance by fine-tuning loops in order to balance the overall clock rate of the FPGA and the total number of execution cycles. Loops that can be fully parallelized are detected and analyzed by varying the number of parallel iterations per loop to increase the clock frequency so that performance equilibrium can be reached at a point where the overall performance is maximized.

2. Loop Parallelism

A loop can be converted into a fully parallel loop if there are no loop-carried dependencies. Parallel loops consume a large portion of area in the FPGA due to several parallel functional units. As more FPGA resources are used, the routes that connect the resources become longer. Hence, longer routes can dominate and reduce the overall clock cycle time of the application running on the FPGA. Small variations in the clock cycle time can have significant impact on the overall performance of an application. For instance, let us assume an application takes n cycles with the overall clock period t after parallel loop transformations. The total execution time is expressed:

\[ T_{\text{parallel}} = n \cdot t \]

Now, let us assume a certain number of iterations for the same loops are parallelized while the rest of iterations are serialized. The total number of cycles is increased to \( c_1 \cdot n \) and the overall clock period is reduced to \( t / c_2 \) where \( c_1 \) and \( c_2 \) are real numbers. The new total execution time is defined as:

\[ T_{\text{new}} = D \cdot n \cdot t = D \cdot T_{\text{parallel}} \quad , \quad D = \frac{c_1}{c_2} \]

\( D \) is the deceleration coefficient. An application slows down if \( D \) is greater than one and speeds up if it is between zero and one. It is possible to reach a \( D \) value that is less than one by varying the number of parallel iterations per loop. For instance, we can limit the number of parallel iterations for loops by partitioning a loop into two sub-loops where one is a parallel loop while the other is a serial loop. We can expect an improvement on the clock rate because there will be better placement opportunities since less FPGA resources are used. This further can lead to much shorter routes between the logic blocks, and speed up the overall clock rate.

3. Experimental Results

We performed the experimental analysis on two DSP benchmarks: huffman encoder and 2D convolution. These two

---

1 The research described in this paper is supported by an Enterprise Ireland Research Innovation Fund Grant IF/2002/035.
applications have loops that can be fully parallelized, and therefore the number of parallel iterations can be controlled by the compiler. For each benchmark, our experimental is compiler carries out loop parallelism and generates Handel-C [1] HDL code. The final HDL code is then passed to the Handel-C compiler, which in turn generates an EDIF netlist for Xilinx Virtex-II XC2V6000 [7] FPGA device. Xilinx EDA tools generate the final mapping of the application for the FPGA chip after placing and routing the design.

The original benchmark without loop parallelism is used as a baseline model. Our metric for performance comparison is the total execution time for each benchmark. The total execution time is determined by multiplying the total number of cycles to execute the benchmark by the overall minimum clock period or maximum clock frequency of the FPGA. The overall minimum clock period is the minimum clock period required to operate the FPGA after each benchmark is placed and routed onto the FPGA with the highest effort levels.

Table 1: Performance statistics

<table>
<thead>
<tr>
<th>Cycles</th>
<th>Clock (MHz)</th>
<th>Time (ns)</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>huff 2D</td>
<td>huff 2D</td>
<td>huff 2D</td>
<td>huff 2D</td>
</tr>
<tr>
<td>Orig.</td>
<td>2,514</td>
<td>19,329</td>
<td>33.1</td>
</tr>
<tr>
<td>Fully</td>
<td>1,052</td>
<td>4,045</td>
<td>22.5</td>
</tr>
<tr>
<td>90%</td>
<td>1,071</td>
<td>6,569</td>
<td>23.8</td>
</tr>
<tr>
<td>70%</td>
<td>1,113</td>
<td>11,412</td>
<td>24.3</td>
</tr>
<tr>
<td>50%</td>
<td>1,149</td>
<td>13,838</td>
<td>24.2</td>
</tr>
</tbody>
</table>

Table 1 compares the results of the original huffman encoder and 2D convolution to the results of fully parallel loops. The total number of cycles is reduced from 2514 cycles to 1052 while the clock frequency drops from 33.1 MHz to 22.5 MHz with fully parallel loops. Despite this drop in the frequency, the optimized program speeds up by about 40%. For 2D convolution, the total number of cycles is reduced by 80% while the clock frequency drops by only 14%. Thus, the overall speedup is about 75%. Our intuition tells us that it is possible to reach a higher clock frequency than the clock frequency of the fully parallel loops if we gradually limit the number of parallel iterations for all loops while keeping the total number of cycles relatively closer to the total number of cycles of the fully parallel loops. That means a certain number of parallel iterations in a loop is restricted to a proportion of the loop iteration space. The rest of the loop is kept as a serial loop. Thus, we select 90%, 70% and 50% proportions of the loop iteration space and tabulate the clock frequency and the total number of cycles for huffman encoder and 2D convolution, respectively.

90% and 70% parallel loops for huffman encoder have a deceleration coefficient value less than 1, and therefore they both outperform the fully parallel loops. However, 50% parallel loops have a deceleration coefficient greater than 1, which means that its overall performance is worse than the fully parallel loops. As loop parallelism is reduced to 50% or less, the performance drops with respect to fully parallel loops. As for 2D convolution, none of 90%, 70% and 50% parallel loops can outperform the fully parallel loops because they all have a deceleration coefficient of greater than 1. The total execution rises as the degree of parallelism decreases. This means that reducing the number of parallel loop iterations has a negative impact on the overall performance of the program. Hence, the best execution time-clock rate product can be achieved by fully parallelizing loops in 2D convolution.

4. Related Work

Callahan et al. [3] selects loops from the program and executes them in a reconfigurable coprocessor while the rest of the program is executed in a RISC processor. Their compiler exploits loop parallelism to reduce the number of execution cycles. In [2], [5] and [6] several loop optimizations are performed to increase parallelism. So et al. [4] examines the balance ratio between the total execution time and FPGA area by tuning the loop unrolling factor for nested loops in the application. However, none of these works analyzed the effects of these optimizations on the overall clock cycle period or examined the issue of fine-tuning loop-level parallelism to improve the overall clock rate of the final circuit.

5. Conclusion

This paper discussed the issue of fine-tuning loop-level parallelism to find a balance between the overall clock rate and the total number of clock cycles to increase the overall performance of DSP applications. Our experimental results concluded that it was possible to reach a performance equilibrium point where the total number of cycles and the overall clock frequency could be adjusted to maximize the overall performance by controlling parallel iterations in the DSP applications.

References