COMPILER GENERATED SYSTOLIC ARRAYS FOR WAVEFRONT ALGORITHM ACCELERATION ON FPGAS

Betul Buyukkurt
Sandbridge Technologies, Inc.
Tarrytown, NY 10591
abuyukku@cs.ucr.edu

Walid A. Najjar
Computer Science and Engineering
University of California Riverside
Riverside, CA 92521
najjar@cs.ucr.edu

ABSTRACT
Wavefront algorithms, such as the Smith-Waterman algorithm, are commonly used in bioinformatics for exact local and global sequence alignment. These algorithms are highly computationally intensive and are therefore excellent candidates for FPGA-based code acceleration. However, there is no standard form of these algorithms, they are used in a wide variety of situations with various constraints. It is therefore not practical to have a standard kernel that can be mapped to an FPGA, hence the importance of being able to compile such codes from a high level language. ROCCC is a C to VHDL compiler, which optimizes and parallelizes the most frequently executed kernel loops in applications such as in multimedia, scientific and high-performance computing. In this paper we describe the transformations performed by ROCCC, which transformed the kernel of the Smith-Waterman algorithm into a hardware systolic array that is mapped onto the FPGA on the SGI Altix RASC blade. We report a throughput increase by over 3,000X over a 2.8 GHz Xeon.

1. INTRODUCTION
Fast sequence alignment computation for DNA sequences has become one of the most challenging computational problems of the decade due to the exponential growth in the size of the biosequence databases. To keep up with this growth, heuristic methods have been adopted in place of exact ones for sequence alignment algorithms to perform faster but potentially inaccurate searches. There are many instances, however, when an exact search must be done at the expense of very large compute times. Furthermore, heuristic search algorithms are known to perform very badly on short strings and it is very common to have sequencing machines generate short strings of 30 to 40 base pairs.

Wavefront algorithms, such as the Smith-Waterman [1] algorithm (shown in Figure 1), are commonly used in bioinformatics for exact local and global sequence alignment. These computations are extremely regular, highly computationally intensive and are applied to a very large set of data. They are excellent candidates for hardware code acceleration. Mapping such computations to a circuit can drastically improve its efficiency as compared to running it on a traditional microprocessor. Wavefront algorithms are usually computed in software using a 2D matrix where the enclosing loop advances in a fixed direction to compute the current cell using neighboring values computed in previous iterations. A fast implementation of the same algorithm in hardware would be a systolic array on which the computations along the back diagonal are implemented.

In this paper we describe a set of compiler transformations and code generation steps that implement a systolic array on an FPGA, in VHDL, starting from its loop nest expressed in C. These transformations are developed within the context of the ROCCC (Riverside Optimizing Compiler for Configurable Computing) tool set [2]. We evaluate our systolic array generation approach using the Smith-Waterman [1] algorithm which is commonly used in DNA and protein string matching. The generated systolic array was mapped on to the Xilinx Virtex 4 LX200 of the SGI RASC RC100 Blade in the Altix 4700 multiprocessor. The results show a throughput increase larger than 3,000X over a 2.8 GHz Xeon using a 1,024 element systolic array.

2. ROCCC OVERVIEW
ROCCC is a C to RTL VHDL compilation framework for mapping application programs to FPGAs. The focus of ROCCC is generating highly parallel and optimized circuits rather than statement-by-statement translation of C programs to VHDL.

ROCCC is not designed to compile a whole program. In order to identify the kernel codes, ROCCC only requires the user to mark the candidate loop-nest by placing it between...
computationaly expensive sequence alignment algorithm, which computes a matching score of two strings - $S$ and $T$ of sizes $N$ and $M$ respectively - using insertions and deletions. The strings, $S$ and $T$, are formed from a finite alphabet. The cardinality of that alphabet is four for DNA (and RNA) strings and 20 for protein strings.

The algorithm has three steps: the initialization, matrix-fill and the trace back. Among the three stages matrix-fill is the one that is most computationally expensive and the hardest to parallelize. It is a wavefront dynamic programming algorithm on a 2D matrix. Once the first row and the first column of the matrix are initialized, a simple local rule is used to update cell $(i, j)$. Entry $(i, j)$ represents the total score, or edit distance, of aligning a prefix of $S$ of size $i$ against a prefix of $T$ of size $j$. The trace back stage is used to identify the location of the best match if a match is found. However, in the vast majority of runs no match is found. The most commonly used SW computation scheme comes in the following format:

$$d = \min \left\{ \begin{array}{ll}
a + \text{substitution} & \text{if } S_i = T_i \\
b + \text{insertion} & \text{if } S_i \neq T_i \\
c + \text{deletion} & \end{array} \right. \quad (1)$$

In the above formula $d$ is the cell being computed, $a$ is the cell’s northwest, $b$ is its north and $c$ is the cell’s west neighbors. The most typical adjustment scores for the SW implementation on DNAs are insertion and deletion scores being set to 1 and the substitution score being set to 2. This particular choice of scores would lead into a special property of the SW algorithm, proven in [10], that the horizontal and vertical neighbors would always differ by $\pm 1$ from one another and are guaranteed to be within $\pm 1$ of their neighbour cell $a$. This property leads into the fact that transferring only the two low order bits among processing elements within the systolic array would be enough for any length string comparison.

### 3.2. Systolic Array Generation in ROCCC

The outer loop in the C code shown in Figure 1 is unrolled to form the processing elements of the systolic array cells. The unrolling factor defines the number of processing elements. To optimize for performance, the code should be unrolled along the shorter edge allowing the elements along the longer edge to be streamed in and the hardware systolic array to compute the 2D matrix in bands of length equal to the length of the systolic
array as illustrated in Figure 5. If we assume that the length of the systolic array is \(k\) and the length of the shorter sequence is \(m\), this arrangement will require \(\lceil m/k \rceil\) systolic arrays to be processed on the hardware one after another, where the values generated by one systolic array are used to initialize the computation of the next one.

Fig. 3. Code to be mapped to hardware after loop unrolling, scalar replacement, feedback reference elimination and loop invariant code motion

Next, the unrolled loop is scalar replaced. Scalar replacement decouples the memory accesses from the actual computation, hence generating a memory access free datapath preceded by all the array reads and followed by all array stores. The array feedback elimination pass is the major pass that generates the systolic array description at the HLL level. It takes in the scalar replaced code and analyzes the type of dependency therein for every load-store pair. It determines whether an array address stored into during a given iteration is read again the following iteration. Such load-store pairs are replaced with scalar temporaries and any initialization code is copied above the innermost loop. Later, these temporaries are replaced by on-chip registers whose values are computed and kept on-chip to be used by future iterations. The number of on-chip registers needed for a given application is linear in the amount of the unrolling factor.

Figure 3 shows the code after loop unrolling, scalar replacement, feedback reference elimination and loop invariant code motion. Figure 4 shows the final output of the high level optimization passes of ROCCC. The ROCCC_load_prev and ROCCC_store2next are macros internal to our compiler. These two macros are inserted into the final code to reflect the effects of the eliminated feedback load/store pairs. ROCCC_load_prev loads a value from the iteration before and ROCCC_store2next saves its input to be read during the next iteration. Both macros are translated into special machine instructions defined in MachineSUIF, followed by VHDL code at later stages. In the generated code, both macros read/write from/to on-chip registers.

In the unaltered wavefront code (Figure 1) each cell computation requires at least three array reads from neighbours and one write. Once unrolled \(k\) times, out of a total of \(4k\) memory accesses per loop iteration only a total of three per iteration are left in the resultant code. \(4k-3\) memory accesses are eliminated through the array feedback elimination pass and transferred through wires and on-chip registers. These figures does not include the \(1k\) reads from \(T\), which are lifted above the \(j\) loop due to invariant code motion, as well as any reads from \(S\).

Fig. 4. Generated datapath

Next, the unrolled loop is scalar replaced. Scalar replacement decouples the memory accesses from the actual computation, hence generating a memory access free datapath preceded by all the array reads and followed by all array stores. The array feedback elimination pass is the major pass that generates the systolic array description at the HLL level. It takes in the scalar replaced code and analyzes the type of dependency therein for every load-store pair. It determines whether an array address stored into during a given iteration is read again the following iteration. Such load-store pairs are replaced with scalar temporaries and any initialization code is copied above the innermost loop. Later, these temporaries are replaced by on-chip registers whose values are computed and kept on-chip to be used by future iterations. The number of on-chip registers needed for a given application is linear in the amount of the unrolling factor.

Figure 3 shows the code after loop unrolling, scalar replacement, feedback reference elimination and loop invariant code motion. Figure 4 shows the final output of the high level optimization passes of ROCCC. The ROCCC_load_prev and ROCCC_store2next are macros internal to our compiler. These two macros are inserted into the final code to reflect the effects of the eliminated feedback load/store pairs. ROCCC_load_prev loads a value from the iteration before and ROCCC_store2next saves its input to be read during the next iteration. Both macros are translated into special machine instructions defined in MachineSUIF, followed by VHDL code at later stages. In the generated code, both macros read/write from/to on-chip registers.

In the unaltered wavefront code (Figure 1) each cell computation requires at least three array reads from neighbours and one write. Once unrolled \(k\) times, out of a total of \(4k\) memory accesses per loop iteration only a total of three per iteration are left in the resultant code. \(4k-3\) memory accesses are eliminated through the array feedback elimination pass and transferred through wires and on-chip registers. These figures does not include the \(1k\) reads from \(T\), which are lifted above the \(j\) loop due to invariant code motion, as well as any reads from \(S\).

4. PERFORMANCE EVALUATION

From the C implementation of SW, ROCCC automatically generates a systolic array implementation of the algorithm as described in the previous section. Our target is the Xilinx Virtex 4 LX200 FPGAs on the SGI RASC RC100 blades [11]. SGI RASC RC100 blade contains two Xilinx Virtex 4 LX200 FPGAs with up to 80MBs of QDR SRAM memory in four modules. Each FPGA is connected to two QDR SRAM memory modules with a bandwidth of 128bits/cycle at each connection. These FPGAs operate seamlessly with the SGI Altix Servers [12], which enables the applications to be run part on the high-performance server and part on the RASC blades.

Table 1 compares the performance of the code on an Intel Xeon 2.8 GHz with Hyperthreading and on an Intel Itanium 1.5 GHz to the performance of our ROCCC generated code on the Virtex 4 LX200. The code for the Xeon and Itanium are compiled using the GCC 4.0.2 and the ICC 9.1 compilers respectively. The CPU data is based on the average of five
Table 1. Throughput evaluation of Smith-Waterman codes in cell updates per second (CUPS)

<table>
<thead>
<tr>
<th>Platform</th>
<th>Speed</th>
<th>GCUPS</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Xeon</td>
<td>2.8 GHz</td>
<td>0.049</td>
<td>1</td>
</tr>
<tr>
<td>Intel Itanium-2</td>
<td>1.5 GHz</td>
<td>0.084</td>
<td>1.7</td>
</tr>
<tr>
<td>200-cells, 2.8% area</td>
<td>191 MHz</td>
<td>38.20</td>
<td>800</td>
</tr>
<tr>
<td>512-cells, 6.9% area</td>
<td>188 MHz</td>
<td>96.25</td>
<td>1964</td>
</tr>
<tr>
<td>1024-cells, 17% area</td>
<td>174 MHz</td>
<td>178.65</td>
<td>3,646</td>
</tr>
</tbody>
</table>

actual runs and computed as first timing the execution of the candidate loop that is actually mapped onto the hardware and then dividing the measured time by the number of computed cells. FPGA data is based on VHDL simulation (using Xilinx ISE 8.2) with the area and clock frequency obtained after place and route. The performance is measured in giga cell-updates per second (GCUPS) as is the literature for SW algorithms.

So far we have experimented with a systolic array size of up to 1024 cells. ROCCC generated 1024-cell systolic array occupies 17% of the area of the Virtex 4 LX200 FPGA and achieved a throughput of 178.65 GCUPS. To fill up 70% of the FPGA area¹, we can run up to 5 copies of the 1024-cell systolic array (for DNAs) simultaneously on an FPGA, which results in a total speedup of around 10,630 over the Itanium and 18,230 over the Xeon respectively. For the 512 and 200 cell systolic arrays, the throughputs obtained were 96.5 and 38.2 GCUPS while the circuit area was only 6.9% and 2.8% of the entire FPGA area respectively.

The ROCCC compiler-generated systolic array code required one element of the input array S and one value from the starting row of the 2D matrix to be fetched onto the FPGA every cycle. Assuming that the sequences under analysis are DNA nucleotides (represented using 2 bits per value), the bandwidth requirement is four bits/cycle. For the five copies of 1024-cell systolic arrays executing simultaneously, the entire computation would require a total of 20 bits/cycle which is well within the capabilities of our target platform.

Our work is the only compiled HDL implementation from C available. ROCCC generated SW circuit fits 1024 cells to 17% of the available FPGA area. Our implementation does not require reconfiguration. Circuit is configured once. The circuit reads the query in as a parameter from outside in the start and the database entries are streamed in one element per cycle after circuit’s initialization. Analyzing the area growth rates of our circuit from 200 to 512 to 1024, we expect to fit around 3000 cells on our target FPGA with a throughput estimate of over 500 GCUPS.

5. CONCLUSION

In this paper we have presented the high-level transformations performed by ROCCC that allows the automatic generation of systolic arrays from wavefront algorithms in C. To our knowledge, there is currently no other C-to-VHDL compiler that can do the systolic array generation transformation. We used Smith-Waterman, a highly computationally intensive bioinformatics sequence alignment algorithm, to evaluate the generated systolic array. We report a throughput increase by over 3,000X over a 2.8 GHz Xeon.

6. REFERENCES


¹The performance of a circuit degrades rapidly when its area exceeds about 80% of the FPGA area, thus in an efficient FPGA design, typically the maximum used area can not exceed 75% - 85%.