Efficient Hardware Generation for Dynamic Programming Problems

Zubair Nawaz #1, Todor Stefanov *2, Koen Bertels #3

# Computer Engineering Lab, Delft University of Technology
The Netherlands
1 z.nawaz@tudelft.nl
3 k.l.m.bertels@tudelft.nl

* Leiden Embedded Research Center, Leiden University
The Netherlands
2 stefanov@liacs.nl

Abstract—**Optimization problems are known to be very hard problems requiring a lot of CPU time. Dynamic Programming (DP) is a powerful method, which is typically used to compute large number of discrete optimization problems. This paper presents an improved approach called RVEP (RVE with pre-computation) that allows to design highly parallel hardware accelerators for wide range of DP problems. We applied our approach to three representative DP problems. We estimate speedups to 200% compared to a pure dataflow approach and at least 25% to previous RVE approach.**

I. Introduction

Optimization problems are usually very important problems and take considerable amount of time to compute. There is always a need to solve them quickly, possibly through parallel computation. Given the great need for CPU power, such problems are good candidates for hardware acceleration. Dynamic Programming (DP) is a method to compute a large number of discrete optimization problems in various fields. Few examples of optimization problems for which dynamic programming is applied are the Knapsack, Traveling salesman problem, Smith-Waterman, shortest path, Viterbi algorithm and Planner's problem.

This paper presents an approach, which extends the *Recursive Variable Expansion* (RVE) algorithm to some representative DP problems, call it RVEP (RVE with pre-computation). The equations for these DP problems are algebraically manipulated to generate highly parallel hardware accelerators using Reconfigurable systems. This approach exposes more parallelism than any other parallel technique at the cost of extra area on FPGA. In case of Smith-Waterman, we estimate a 2x speedup at the cost of around 4x more hardware as compared to dataflow approach. This approach is especially suitable for cases where high performance is a priority and extra area can be used to achieve this.

Our acceleration is based on RVE [1], which is similar to techniques like back substitution [2], look ahead computation [3], [4] and block back-substitution [5]. In all of these techniques, the recurrence is iterated M times, expanded and rearranged to calculate the result of M iterations of the original recurrence. All these transformations produce a lot of redundant computations, however the critical path is reduced by using the *recursive doubling decomposition* algorithm [4].

The first known implementation of look ahead in DP problems was done in [6], where it was applied to the viterbi algorithm and showed its potential for DP problems. It was shown that the add-compare-select (ACS) operation, which is nonlinear in nature, is difficult to parallelize. Later, this work was extended to DP problems [7], [8], again showing only the viterbi algorithm example. As resources were very limited at that time, there was less exploited parallelism. No hint was given to tackle conditional statements in generic DP formulations.

Some of the ideas in Section III has been reported in our previous work [9], applied only to Smith-Waterman algorithm and some simplification relevant to Smith-Waterman were made. We now term it as RVENP (RVE with no pre-computation). It was shown that RVENP is a useful method for accelerating the Smith-waterman algorithm.

The work presented in this paper is related to the Delft Workbench (DWB) project[1] [10]. The DWB is a semi-automatic toolchain platform for integrated hardware-software co-design targeting the Molen[11] reconfigurable architecture. The proposed approach is currently being implemented as an extension of the DWARV HW compiler [12] and focuses on DP problems. [13], [14], [15] are other examples of efficient but highly focused/restricted HW compilers generating efficient hardware for e.g. perfectly nested loops.

This paper extends the work described in [9] by applying it to more DP problems, which represent a broad range of such problems. The main contributions of the paper are:

1) an approach called RVEP, which is described by applying to 3 representative problems.
2) extension of the algorithm with arbitrary number of conditional statements.
3) an estimation of speedup and hardware cost and its comparison with other best available approaches.

The paper is organized as follows. Section II describes 3

[1]Delft Workbench is sponsored by the hArtes (IST-035143), the MORPHEUS (IST-027342) and RCOSY (DES-6392) projects.

		G	A	C	G	G	A
	0	-2	-4	-6	-8	-10	-12
G	-2	**1**	-1	-3	-5	-7	-9
A	-4	-1	**2**	0	-2	-4	-6
T	-6	-3	**0**	1	-1	-3	-5
C	-8	-5	-2	**1**	0	-2	-4
G	-10	-7	-4	-1	**2**	1	-1
G	-12	-9	-6	-3	0	**3**	1
A	-14	-11	-8	-5	-2	1	**4**

(a) Needleman-Wunsch

		a	b	a	**c**	**d**	a	c	**c**
	0	0	0	0	0	0	0	0	0
c	**0**	0	0	0	1	1	1	1	1
a	0	**1**	1	1	1	1	2	2	2
d	0	**1**	**1**	**1**	1	2	2	2	2
c	0	1	1	1	**2**	2	2	3	3
d	0	1	1	1	**2**	3	3	3	3
d	0	1	1	1	2	**3**	**3**	**3**	3
c	0	1	1	1	2	3	3	4	**4**

(b) LCS

Figure 1. Scoring Matrix, when $g = -2$ and $x[i,j] = 1$ when $S[i] = T[j]$ otherwise -1. Elements in bold show the traceback.

representative DP problems. Then in Section III, we describe the steps and apply on each of the problems. In Section IV, we estimate the hardware acceleration as compared to dataflow and RVENP as given in [9]. Finally, Section V concludes the paper with future work.

II. Representative Problems

In this section, we will describe representative problems encompassing a broad range of DP problems.

A. Needleman-Wunsch (NW) Algorithm

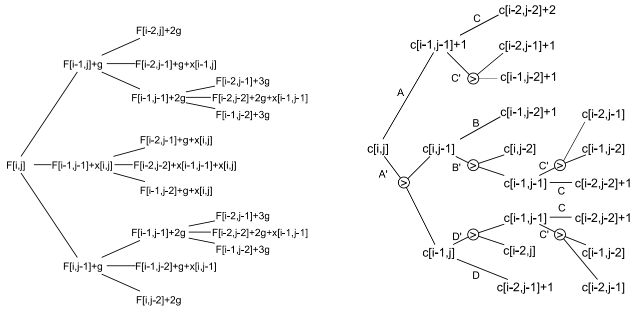
NW is a global alignment algorithm for two biological sequences [16]. The optimal alignment score $F[i,j]$ for two sequences $S[1..i]$ and $T[1..j]$ is given by the following recurrence equation.

$$F[i,j] = \max \begin{cases} F[i,j-1] + g \\ F[i-1,j-1] + x[i,j] \\ F[i-1,j] + g \end{cases} \tag{1}$$

where $F[0,0] = 0, F[0,j] = g \times j$ and $F[i,0] = g \times i$, for $1 \leq i \leq n, 1 \leq j \leq m$, n and m are lengths of S and T respectively. The $x[i,j]$ is the score for match/mismatch, depending upon whether $S[i] = T[j]$ or $S[i] \neq T[j]$. The g is some constant penalty for inserting a gap in any sequence. A table is filled using Equation 1 for the two sequences. The traceback to find the optimal solution is always started from bottom right corner of the table. For most of the DP problems, traceback is done in a similar way. Figure 1a shows a table-fill-and-traceback example. The global alignment as a result of traceback shown in Figure 1a, is

B. Smith-Waterman (SW) Algorithm

It is a local alignment algorithm for two biological sequences. It has similar formulation as NW with three changes. First is the addition of fourth term 0 in the max equation of NW, second is the different boundary condition and the third is that traceback starts from the highest value any where in the table till it reaches a certain threshold value or 0. The local optimal alignment score *F[i,j]* is as following.



(a) NW (b) LCS

Figure 2. RVE Expanded

$$F[i,j] = \max \begin{cases} F[i,j-1] + g \\ F[i-1,j-1] + x[i,j] \\ F[i-1,j] + g \\ 0 \end{cases} \tag{2}$$

where $F[0,0] = F[0,j] = F[i,0] = 0$, for $1 \leq i \leq m$ and $1 \leq j \leq n$.

C. Longest Common Subsequence (LCS) problem

Given a string of characters, if some of the characters are deleted from that string, then the resulting string is called a *subsequence*. For example, $Z = \langle a,d,c \rangle$ is a subsequence of $X = \langle a,b,a,c,d,a,c,d \rangle$. Given two subsequences X and Y, we say that Z is a *longest common subsequence* of X and Y, if Z is longest among all subsequences common to both X and Y [17]. Let *c(i,j)* is the length of the LCS for sequences X_i and Y_j, then its formulation for $i,j > 0$ is given by

$$c[i,j] = \begin{cases} c[i-1,j-1]+1 & \text{if } x_i = y_j, \\ \max\{c[i,j-1], c[i-1,j]\} & \text{if } x_i \neq y_j. \end{cases} \tag{3}$$

where $c[i,j] = 0$ for $i = 0$ or $j = 0$. Similar to NW, traceback is started from bottom right corner of the table. The LCS as we get from Figure 1b is $\langle a,c,d,c \rangle$. The condition in the recursive formulation along with max make it different from the previous two examples. Another very well known problem which has a similar structure is the Knapsack problem.

III. Recursive variable expansion for DP Problems

In this section, we describe the steps of our approach by applying it on the examples in Section II. Since NW and SW are very similar, we will only show the steps for SW where it is different from NW. We have termed this approach as *RVEP*. The transformed equations when mapped on hardware exhibit more parallelism than dataflow and RVENP [9] alone. A detailed description is as follows.

A. Apply RVE

We apply RVE partially on Equation 1 of NW to expose three levels of parallelism. The recursion tree after the application of RVE is shown in Figure 2a. $F[i,j]$ can be written as max of the leaf nodes in Figure 2a. Similarly, we get the recursion tree shown in Figure 2b when RVE is partially applied on LCS. The edge labels in Figure 2b define the condition as A defines $x_i = y_j$, B defines $x_i = y_{j-1}$, C defines $x_{i-1} = y_{j-1}$, D define $x_{i-1} = y_j$ and A', B', C', D' are the complement of A, B, C, D respectively.

B. Remove redundant sub-equations

In case of NW, there are some leaf nodes which are redundant, the reduced equation after removing the redundant nodes is the following.

$$F[i,j]=\max\begin{cases} i & F[i,j-2]+2g \\ ii & F[i-1,j-2]+g+x[i,j-1] \\ iii & F[i-1,j-2]+3g \\ iv & F[i-1,j-2]+g+x[i,j] \\ v & F[i-2,j-2]+2g+x[i-1,j-1] \\ vi & F[i-2,j-2]+x[i-1,j-1]+x[i,j] \\ vii & F[i-2,j-1]+3g \\ viii & F[i-2,j-1]+g+x[i,j] \\ ix & F[i-2,j-1]+g+x[i-1,j] \\ x & F[i-2,j]+2g \end{cases} \tag{4}$$

The 13 leaf nodes in Figure 2a are reduced to 10 sub-equations in Equation 4.

When the conditional statements are mixed in a max statement, then it is not obvious to remove the redundancies as in the case of LCS, shown in Figure 2b. The non-associative nature of conditional statements make it difficult to get an additional benefit from applying RVE. It is algebraically correct to take the maximum of all the unique nodes and any statement will be only effective when its accompanying conditional statement is also true, otherwise it will be 0. The *c[i,j]* after RVE expansion can thus be written as follows:

$$c[i,j] \quad = \quad \begin{array}{l} \qquad\quad c[i-2,j-2]+2 \qquad A1 \\ \max\begin{cases} c[i-1,j-2]+1 & A2 \\ c[i-2,j-1]+1 & A3 \\ c[i,j-2] & A4 \\ c[i-2,j-2]+1 & A5 \\ c[i-1,j-2] & A6 \\ c[i-2,j-1] & A6 \\ c[i-2,j] & A7 \end{cases}\end{array} \tag{5}$$

where $A1 = A \wedge C$, $A2 = (A \wedge C') \vee (A' \wedge B)$, $A3 = (A \wedge C') \vee (A' \wedge D)$, $A4 = (A' \wedge B')$, $A5 = (A' \wedge B' \wedge C) \vee (A' \wedge D' \wedge C)$, $A6 = (A' \wedge B' \wedge C') \vee (A' \wedge D' \wedge C')$ and $A7 = A' \wedge D'$. Here $A \wedge C$ means A AND C, $A \vee C$ means A OR C. The Equation 5 has only 8 sub-equations as compared to 13 leaf nodes in Figure 2b.

C. Group sub-equations

In NW, Equation 4 can be rearranged and simplified to the following.

$$F[i,j] = \max\begin{cases} i & (F[i,j-2] \succ F[i-2,j]) + 2g \\ ii & F[i-1,j-2] + C_1 \\ iii & F[i-2,j-2] + C_2 \\ iv & F[i-2,j-1] + C_3 \end{cases} \tag{6}$$

where $C_1 = ((g + (x[i,j-1] \succ x[i,j])) \succ 3g)$, $C_2 = ((2g + x[i-1,j-1]) \succ (x[i-1,j-1] + x[i,j])) = (2g \succ x[i,j]) + x[i-1,j-1]$ and $C_3 = (3g \succ (g + (x[i,j] \succ x[i-1,j])))$ for Equation 4. Here $\succ$ is defined as the max operator.

In LCS, Equation 5 can be rearranged and simplified to the following.

$$c[i,j] \quad = \quad \begin{array}{l} \qquad\quad c[i-2,j-2]+2 \qquad A1 \\ \max\begin{cases} c[i-1,j-2]+C'_1 & A2 \vee A6 \\ c[i-2,j-1]+C'_2 & A3 \vee A6 \\ c[i,j-2] & A4 \\ c[i-2,j-2]+1 & A5 \\ c[i-2,j] & A7 \end{cases}\end{array} \tag{7}$$

where $C'_1=\max\begin{cases} 1 & \text{if } A2 \\ 0 & \text{if } A6 \end{cases}$ and $C'_2=\max\begin{cases} 1 & \text{if } A3 \\ 0 & \text{if } A6 \end{cases}$.

It is possible that $A2$ and $A6$ are true at the same time. Similarly for $A3$ and $A6$.

D. Precompute cost function

Precomputation for an iteration means to do a part of computation for the current iteration in some earlier iteration where its contents are known. This can further increase the parallelism without increasing the hardware. This is an extension to the work described in [9], which reduces the critical path without an increase in area on an FPGA.

In NW, while $F[i,j]$ is being computed C_1, C_2 and C_3 for next iteration of (i,j) values defined as (i',j') can be computed in parallel as shown in Figure 5.

Similarly in LCS, the contents of C'_1 and C'_2 in Equation 7 is known a priori, therefore C'_1,C'_2, $A4$ and $A7$ for the next iteration *of* (i,j) defined as (i',j') can be computed in parallel with computation of $c[i,j]$ for the current iteration (i,j). The circuit for LCS as given by Equation 7 is shown in Figure 6. C'_1 and C'_2 are also optimized to $C^*[i,j]$ in Figure 6.

E. Fill the block and mix with dataflow

In NW, Equation 6 only computes $F(i,j)$ (i.e. O1) as shown in Figure 3a. We can compute $F(i,j-1)$, $F(i-1,j)$ and $F(i-1,j-1)$ (i.e. O2, O3 and O4 as shown in Figure 3a) using the same steps as followed for finding $F(i,j)$. All these unknown variables in a block can be computed in parallel, as there are no dependencies among them. The whole table can be filled as shown in Figure 3c, which is like dataflow

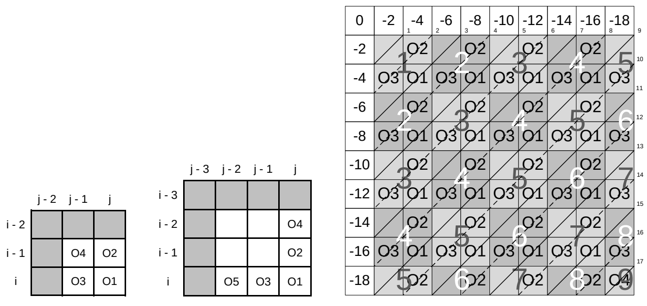

(a) block b=2 (b) block with b=3 (c) sequence of fill

Figure 3. Filling the table

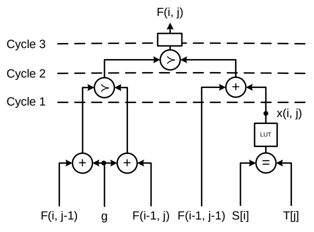
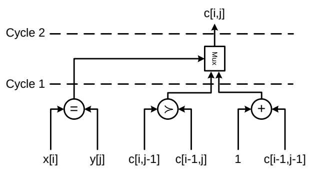


(a) NW (b) LCS

Figure 4. Circuit for one element

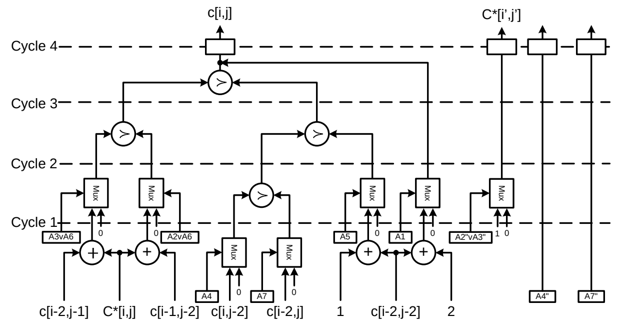


Figure 6. Circuit for LCS as given by Equation 7. $A2$”,$A3$” and $A4$”,$A7$” are based on (i',j').

at block level. In Figure 3c, the number on the blocks shows the sequence of fill and all blocks with the same number are executed in parallel.

In DP problems, we fill the complete table, find the maximum value, which is mostly at the bottom right corner of the table and then trace back to find the solution. We can avoid filling the whole table for all those problems whose maximum is at bottom right corner. $F(i,j)$ is chosen from any of the sub-equation, therefore can be traced back to the respective element. In a block, we only need boundary elements to compute, therefore computation of O4 can be avoided in Figure 3a. This saves area on FPGA. This saving can be increased, when we apply RVE with a larger blocking factor as shown in Figure 3b.

We cannot apply this technique for SW, as according to the algorithm, the traceback starts from the maximum value in the table, that can be anywhere in the table. Therefore we have to completely fill the table. This reduction technique can be used for most of the DP problems, as in most of them, the traceback starts from bottom right corner.

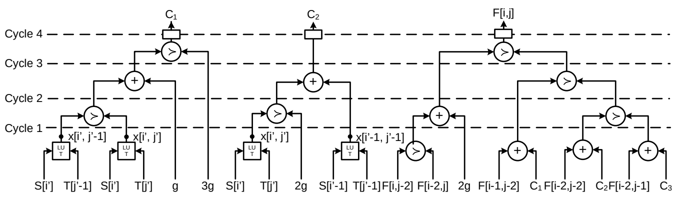


Figure 5. Circuit for computing an element of RVEP for NW problem using Equation 6.

IV. Estimate for Hardware Acceleration and Cost

This section presents the estimate of the time and hardware for 3 monadic DP problems using dataflow, RVENP and RVEP. Even though this section presents the derived hardware circuits for the RVEP equation, embedding them in the DWARV compiler [12] is a mere implementation issue and not a conceptual one.

The respective equations after applying the precomputation will result in circuits shown in Figure 5 and Figure 6 for NW and LCS respectively. Similarly circuits for NW and LCS with dataflow implementation are shown in Figure 4a and Figure 4b respectively. In Figure 5 and 4a , LUT denotes the table used to get the value of $x[i,j]$ in Equation 1 and 6 respectively.

We have tried to estimate the time and hardware for 3 DP problems. It is assumed that the time for each cycle is equal to the latency of one adder, comparator, LUT or MUX operation. RVEP extension is applied to these problems, the circuits are drawn, then the hardware and time is estimated for it. When the dataflow approach is applied, then the maximum number of steps in a table of dimensions $m \times n$ is $m+n-1$. Therefore, if each step takes c cycles, then the total time to compute the whole table is $c(m+n-1)$. The maximum number of elements to be computed in parallel are equal to $l_d = \min(m,n)$. If h_e quantifies the amount of hardware used to compute a single element, then the maximum amount of hardware needed are $h_e \times l_d$. In case of RVENP and RVEP, the maximum number of steps in a table of dimensions $m \times n$ that needs to be computed are $s = \left\lceil \frac{m}{b} \right\rceil + \left\lceil \frac{n}{b} \right\rceil - 1$ and the number of blocks that needs to be computed in parallel are $n_b = \min(\left\lfloor \frac{m}{b} \right\rfloor, \left\lfloor \frac{n}{b} \right\rfloor) + \frac{\min(m,n) \bmod b}{b}$ [9]. Similarly, if each step take c cycles, then the total time to compute the whole table is $c \times s$. If h_b is the amount of hardware used to compute a block, then the maximum amount of hardware needed is $h_b \times n_b$. Results for time and hardware estimation for 3 problems by applying three different approaches are summarized in Table I.

When RVEP is applied to SW, it gives a 2x speedup as compared to the dataflow implementation of SW at the cost of around 4x more hardware. It is 25% better than RVENP at the cost of little extra hardware. Similarly, RVEP gives a speedup of 1.52x as compared to the dataflow implementation of NW at an extra cost of 4.25x the hardware. Precomputation

Table I

TIME AND HARDWARE ESTIMATION

		Time			Hardware											
		(cycles)			$+$			$\succ$			registers			LUT/MUX		
		variable	val[1]	speed-up[2]	var	val[1]	over-head[2]	var	val[1]	over-head[2]	var	val[1]	over-head[1]	var	val[1]	over-head[2]
SW	Dataflow	$4(m+n-1)$	396	1	$3\times l_d$	150	1	$3\times l_d$	150	1	$1\times l_d$	50	1	$1\times l_d$	50	1
	RVENP	$5(\lceil \frac{m}{2} \rceil + \lceil \frac{n}{2} \rceil -1)$	245	1.62	$18\times n_2$	450	3	$21\times n_2$	525	3.5	$4\times n_2$	100	4	$4\times n_2$	100	2
	RVEP	$4(\lceil \frac{m}{2} \rceil + \lceil \frac{n}{2} \rceil -1)$	196	2.02	$18\times n_2$	450	3	$21\times n_2$	525	3.5	$9\times n_2$	225	4.5	$4\times n_2$	100	2
NW	Dataflow	$3(m+n-1)$	297	1	$3\times l_d$	150	1	$2\times l_d$	100	1	$1\times l_d$	50	1	$1\times l_d$	50	1
	RVENP	$5(\lceil \frac{m}{2} \rceil + \lceil \frac{n}{2} \rceil -1)$	245	1.21	$18\times n_2$	450	3	$17\times n_2$	425	4.25	$4\times n_2$	100	4	$4\times n_2$	100	2
	RVEP	$4(\lceil \frac{m}{2} \rceil + \lceil \frac{n}{2} \rceil -1)$	196	1.52	$18\times n_2$	450	3	$17\times n_2$	425	4.25	$9\times n_2$	225	4.5	$4\times n_2$	100	2
LCS	Dataflow	$2(m+n-1)$	198	1	$1\times l_d$	50	1	$2\times l_d$	100	1	$1\times l_d$	50	1	$1\times l_d$	50	1
	RVENP	$5(\lceil \frac{m}{2} \rceil + \lceil \frac{n}{2} \rceil -1)$	245	0.81	$8\times n_2$	200	4	$17\times n_2$	425	4.25	$10\times n_2$	250	5	$14\times n_2$	350	7
	RVEP	$4(\lceil \frac{m}{2} \rceil + \lceil \frac{n}{2} \rceil -1)$	196	1.01	$8\times n_2$	200	4	$17\times n_2$	425	4.25	$12\times n_2$	300	6	$14\times n_2$	350	7

[1] values calculated for *m=50* & *n=50*, [2] with respect to the dataflow case , RVENP is without pre-computation, RVEP is with pre-computation.
$l_d=\min(m,n)=50,\ n_2=\min(\lfloor \frac{m}{2} \rfloor,\lfloor \frac{n}{2} \rfloor)+\frac{\min(m,n) \bmod 2}{2}=25$[1]

gives a boost of 26% as compared to RVENP at the cost of small hardware. However, RVEP does not improve the speed for LCS as compared to dataflow implementation, despite the fact it is using extra hardware. RVENP even slows it down. The reason for this slow down lies in the recurrence equation of LCS, which does not have enough associative operators. It is the presence of associative operators, which helps in reducing the depth of the circuit.

V. CONCLUSION

In this paper, we have improved our previous approach[9], named RVEP, that can generate a highly parallel circuit for DP problems. RVEP relaxes some constraints presented in previous work to make it suitable for wide range of DP problems. We have applied it to three representative DP problems, two among them show better speedups than dataflow approach at the expense of more area. In case of SW, we reported a 2x speedup compared to the dataflow approach and at least 25% faster compared to previous RVENP approach at the cost of around 4x area overhead to dataflow approach. As future work, we will propose a generic framework allowing a certain class of DP to be accelerated using our approach. We will also report on real implementations rather than estimations.

REFERENCES

[1] Z. Nawaz, O. Dragomir, T. Marconi, E. M. Panainte, K. Bertels, and S. Vassiliadis, "Recursive variable expansion: A loop transformation for reconfigurable systems," in *proceedings of International Conference on Field-Programmable Technology 2007*, December 2007.

[2] D. Kuck, Y. Muraoka, and S.-C. Chen, "On the number of operations simultaneously executable in fortran-like programs and their resulting speedup," *Transactions on Computers*, vol. C-21, pp. 1293– 1310, 1972.

[3] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *J. ACM*, vol. 27, no. 4, pp. 831–838, 1980.

[4] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Transactions on Computers*, vol. C-22, pp. 786–793, 1973.

[5] M. S. Schlansker and V. Kathail, "Acceleration of first and higher order recurrences on processors with instruction level parallelism," in *Proceedings of the 6th International Workshop on Languages and Compilers for Parallel Computing*, (London, UK), pp. 406–429, Springer-Verlag, 1994.

[6] G. Fettweis and H. Meyr, "Parallel viterbi algorithm implementation: breaking the acs-bottleneck," *IEEE Transactions on Communications*, vol. 37, pp. 785 – 790, 1989.

[7] K. Parhi, "Look-ahead in dynamic programming and quantizer loops," in *Circuits and Systems, 1989., IEEE International Symposium on*, pp. 1382–1387 vol.2, May 1989.

[8] K. Parhi, "Pipelining in dynamic programming architectures," *Signal Processing, IEEE Transactions on*, vol. 39, pp. 1442–1450, Jun 1991.

[9] Z. Nawaz, M. Shabbir, Z. Al-Ars, and K. Bertels, "Acceleration of smith-waterman using recursive variable expansion," in *11th Euromicro Conference on Digital System Design (DSD-2008)*, pp. 915–922, September 2008.

[10] "Delft workbench. online:http://ce.et.tudelft.nl/DWB/."

[11] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte, "The molen polymorphic processor," *IEEE Transactions on Computers*, pp. 1363– 1375, November 2004.

[12] Y. D. Yankova, G. Kuzmanov, K. Bertels, G. N. Gaydadjiev, Y. Lu, and S. Vassiliadis, "Dwarv: Delftworkbench automated reconfigurable vhdl generator," in *In Proceedings of the 17th International Conference on Field Programmable Logic and Applications (FPL07)*, pp. 697–701, August 2007.

[13] "Roccc. online: http://www.cs.ucr.edu/∼roccc/."

[14] "Spark: A parallelizing approach to the high level synthesis of digital circuits. online: http://mesl.ucsd.edu/spark/."

[15] A. Putnam, D. Bennett, E. Dellinger, J. Mason, P. Sundararajan, and S. Eggers, "Chimps: A c-level compilation flow for hybrid cpu-fpga architectures," in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, pp. 173–178, Sept. 2008.

[16] S. Needleman and C. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *J. Mol Biol.*, vol. 48, pp. 443–453, 1970.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, McGraw Hill, second ed., 2001.