FPGA Implementation of Pipelined On-line Scheme for 3-D Vector Normalization

Zhijun Huang and Miloš D. Ercegovac
Computer Science Department
Univ. of California at Los Angeles
{zjhuang, milos}@cs.ucla.edu

Abstract

A three-dimensional vector normalizer based on pipelined on-line arithmetic is presented. The clock period is kept small by the use of redundant adders and low-precision estimates. The throughput is greatly improved by unfolding and pipelining on-line units and the area is reduced due to left-to-right processing. Assuming the same area/delay metric, the proposed scheme can improve throughput by 89% compared with PROVEN scheme, an ASIC normalizer, and by 10.2X compared with VU, a vector-processing unit for 3-D graphics computing. When implemented as an FPGA-based hardware accelerator, our scheme allows 85% more throughput than VU, and 2.3X more throughput than Pentium III SSE.

1. Introduction

Vector normalization is a basic operation in many computer graphics algorithms. The well-known Phong illumination model [1] is such an example which needs fast and frequent vector normalization. In this model, the light intensity I_λ of a point on an object surface is given by

$$I_\lambda = I_{A\lambda} k_a C_\lambda + I_{L\lambda}(k_d C_\lambda(\vec{G}_N \vec{L}_N) + k_s(\vec{G}_N \vec{H}_N)^n) \quad (1)$$

where $\vec{G}_N$ is the normalized surface normal, $\vec{L}_N$ is the normalized light vector, $\vec{H}_N$ is the halfway vector which in turn is the normalized sum of the light vector $\vec{L}$ and the viewer vector $\vec{V}$. Phong shading performs per-pixel illumination computations to determine pixels' colors using Equation (1). Despite of its superior effect, Phong shading is still uncommon in commercial graphics hardware because of its per-pixel computation complexity [2].

With the continuous improvement in IC technologies, direct hardware support for fast vector normalization, the bottleneck operation in Phong shading and many other graphics applications, is being considered. Knittel [3] presented a deep pipeline structure for vector normalization, PROVEN, which could produce one normalized vector each clock cycle. Since the functional units in PROVEN are based on conventional arithmetic design, the clock period is relatively large. Takagi and Kuwahara [4] [5] proposed a digit-recurrence algorithm for computing Euclidean norm of a 3-D vector. The algorithm combines the operations of three squares, two additions and a square root into one digit-recurrence, which is similar to composite on-line arithmetic [13]. Because it is a digit-recurrence algorithm, the throughput is low. A composite on-line arithmetic scheme for vector normalization (NOL) is proposed by Ercegovac and Lang [6]. Similarly, this on-line scheme has the throughput limited by the latency. Compared to a standard floating-point scheme for 3-D normalization, these two schemes have a speedup of 3. No implementations are given for the above two digit-recurrence algorithms.

An existing implementation is the 2.44-GFLOPS 300-MHz vector-processing unit (VU) for high-performance 3-D graphics computing designed by Ide *et al.* [7] and Suzuoki *et al.* [8]. VU is a SIMD-VLIW structure with parallel execution of multiply-accumulate unit and divide/square root unit. We will explore the execution of vector normalization on VU later in Section 3.4.

Other approaches are FPGA-based hardware accelerators and field-programmable custom computing machines for graphics applications [9][10][11][12]. Considering that graphics algorithms keep evolving and more computationally demanding, FPGA-based approaches have important advantage in flexibility over ASIC solutions and performance advantage over microprocessor implementations. There is little previous work on FPGA-based accelerators for 3-D vector normalization.

In this paper, we present a pipelined on-line (POL) scheme for 3-D vector normalization (TDVN), POL-TDVN, and study its FPGA-implementation efficiency. POL-TDVN is an extension of the non-pipelined on-line (NOL) scheme in [6]. The throughput problem caused by digit-serial nature of NOL arithmetic is overcome by unfolding and pipelining recursive on-line units. Unlike conventional arithmetic, on-line arithmetic allows digit-level pipelining of recursive computations. The clock period is kept small by the use of redundant adders and low-precision estimates. The area of POL scheme is reduced by truncating unnecessary digit-modules.

The rest of the paper is organized as follows. Sec-

Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)
0-7695-2667-5 $20.00 © 2001 IEEE

tion 2 defines the problem and general scheme. Section 3 presents implementation details of the NOL scheme, which is the basis of the POL scheme. Section 4 illustrates how to unfold and pipeline on-line arithmetic units. Section 5 gives theoretical evaluation and comparison with PROVEN and VU assuming the same area/delay metric. Section 6 shows FPGA implementation results and speedups over processor approaches. Conclusions are given in Section 7.

2. Problem Definition and General Scheme

A 3-D vector normalizer accepts a vector $\vec{V} = [\ v_x \quad v_y \quad v_z\ ]^T$ and produces the normalized vector $\vec{N} = [\ n_x \quad n_y \quad n_z\ ]^T$, which satisfies

$$\vec{N} = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} = \begin{bmatrix} v_x/d \\ v_y/d \\ v_z/d \end{bmatrix} \tag{2}$$

and

$$d = \sqrt{{v_x}^2 + {v_y}^2 + {v_z}^2} \tag{3}$$

The core operations of the vector normalization are sum of three squares (SO3S), square root (SQRT), and division (DIV). Since each vector element is usually represented in the floating-point format, a few extra processing steps are needed before and after the core operations.

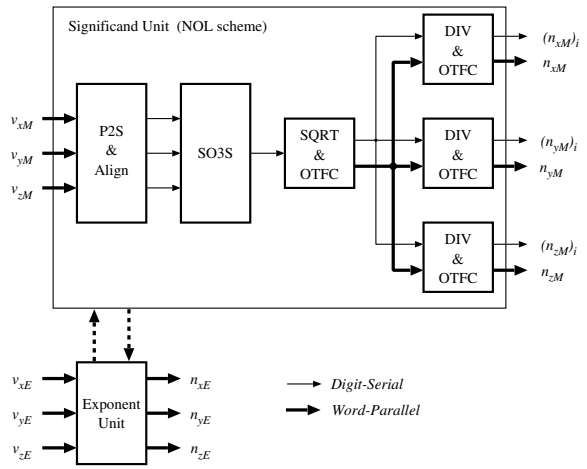


Figure 1: General on-line scheme

On-line arithmetic is useful in many computation-intensive applications such as function evaluation, matrix computations and DSP implementations [13][15][16][17][19]. In conventional arithmetic, all digits of the result must be computed before the next operation can be started. In on-line arithmetic, however, the operands, as well as the results, flow through the arithmetic units in a most-significant-digit-first (MSDF) serial manner. Successive operations execute in an overlapped manner as soon as a small number of the input digits is available. The major properties of on-line arithmetic network are: (a) digit-level operation overlapping, (b) high modularity, (c) simple interconnection requirements, and (d) precision-independent clock period. These properties make on-line arithmetic suitable in reconfigurable systems such as FPGA-based custom computing machines. The main disadvantage of on-line arithmetic is the latency determined by the serial nature of operations.

A general scheme using non-pipelined on-line (NOL) arithmetic is shown in Fig. 1. Due to the floating-point representation, it has exponent and significand units. The exponents of the results are calculated in parallel by the exponent unit. This exponent unit also provides the alignment distances for the alignment function in significand unit and executes final significand normalization into the standard range $[0.5, 1)$ if necessary. In the following analysis, we neglect the exponent unit. The significand unit performs the on-line computation. The bit-parallel inputs are converted into serial form and aligned by P2S-and-align unit [6]. The serial inputs then flow through three main on-line arithmetic units: SO3S, SQRT and DIV. Each on-line unit is an one-dimensional linear array of identical modules. On-the-fly converters (OTFC) are used to convert signed-digit partial results into conventional 2's complement forms [14]. OTFC is performed concurrently with the digit-by-digit generation of redundant results so that the conversion does not cause extra delay.

When the NOL scheme is unfolded and pipelined to produce high-throughput POL scheme, each linear array is extended into two-dimensional stair-shape array and each digit-serial signal becomes an array of digit-serial signals. When fully pipelined, a POL scheme consists of n stages. In the steady state, it computes up to n different normalized vectors, the first stage producing the first digit of the i-th result and the last stage producing the last digit of the $(i-n)$-th result.

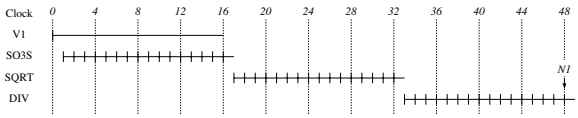


Figure 2: Timing diagram of conventional scheme.

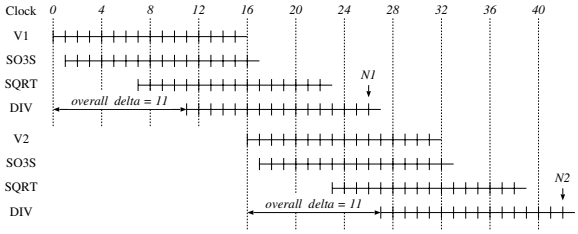


Figure 3: Timing diagram of NOL scheme.

To demonstrate the advantages of (pipelined) on-line arithmetic, the timing diagrams of different

Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)
0-7695-2667-5 $20.00 © 2001 IEEE

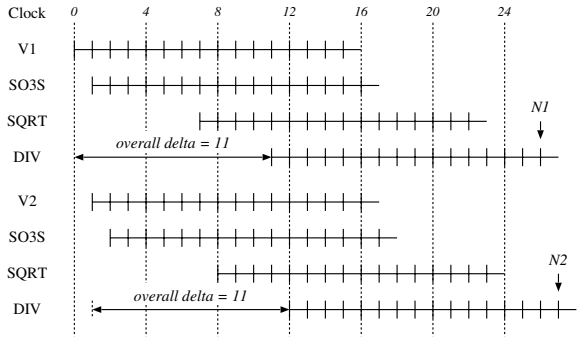


Figure 4: Timing diagram of POL scheme.

schemes are considered. Assume 16-bit precision with 15-bit fractions in the following analysis. Each timing diagram is marked by the available clocked digits. In a conventional (LSD-first) digit-serial scheme, usually all digits must be known before a successive operation begins, as shown by the timing digram in Fig. 2. In this case, the full-precision normalized vector in parallel from is available at 48-th clock cycle. Moreover, the clock frequency is relatively low because of the full-precision carry propagation. In a NOL scheme, successive operations execute in an overlapped manner as soon as δ input digits are available, as shown in Fig. 3. From Fig. 3, it can be seen that the on-line delay δ of SO3S is 1, δ of SQRT is 6 and δ of DIV is 4 and that the overall on-line delay is 11. The final result is available at 26-th cycle, in contrast with 48-th cycle in conventional scheme. Each operation unit in NOL accepts a new vector input after the previous computation is completely done. Fig. 4 shows the timing diagram of the POL scheme. POL has the same on-line delay as NOL, it accepts one vector per cycle and produces one normalized vector per cycle when fully pipelined.

3. Non-pipelined On-line (NOL) Scheme

Since NOL scheme is the basis of POL scheme, we first present the implementation details of NOL scheme. The theoretical part has been presented in [6]. As shown in Fig. 1, the core units are: P2S for parallel-to-serial conversion, SO3S for sum of three squares, SQRT for square root, DIV for division, and OTFC for on-the-fly conversion from the serial signed-digit form to the parallel 2's complement form. P2S can be easily realized by a shift register with parallel loading. To implement alignment, a shift start signal is used to control the shift register [16][6]. Other units are explained in detail in this section.

The inputs and outputs have $(n+1)$-digit precision with n fractional digits in radix-2 form. For a signed-digit d in $\{-1, 0, 1\}$, a suitable coding in bit level is described by the arithmetic expression $d = d1 - d0$, where $d1$ and $d0$ are two bits of the representation. We shall treat "11" as don't-care input to simplify the design.

3.1 Sum of Three Squares

Instead of separate units for multiplication and addition, a composite unit, SO3S, is designed to simplify the implementation. The NOL algorithm for SO3S, Fig. 5, accepts three vector elements, X, Y and Z ($X, Y, Z < 1$ and at least one of them is no less than 0.5), and produces the sum of three squares S ($0.25 \leq S < 3$) in a serial manner. The recurrence is implemented using carry-save additions to keep a short cycle time. The on-line delay is chosen to be $\delta = 1$ and the output digit s_j is in the range of $\{0, 1, .., 8\}$. Output digit s_j corresponds to the integer part of the j-th intermediate residual $W[j]$. The fractional part of $W[j]$ is the next residual $R[j]$.

– Initialize–
$R[-1] = 0;$
$X[-1] = Y[-1] = Z[-1] = 0;$
– Recurrence –
for $j = 0, 1, 2, ..., n$ **do:**
$W[j] = 2R[j-1] + (2X[j-1] + x_j 2^{-j})x_j$
$+(2Y[j-1] + y_j 2^{-j})y_j + (2Z[j-1] + z_j 2^{-j})z_j;$
$s_j \leftarrow csint(W[j]);$
$R[j] \leftarrow csfrac(W[j]);$
$X[j] \leftarrow append(X[j-1], x_j);$
$Y[j] \leftarrow append(Y[j-1], y_j);$
$Z[j] \leftarrow append(Z[j-1], z_j);$
end for

Figure 5: NOL Algorithm for SO3S

A block diagram of implementation is given in Fig. 6. FLAG block in the diagram is a shift register generating an n-bit *flag* signal with a shifting '1'. The flag signal is used to indicate the current input digit position and working precision. REG/APPEND block appends each new input digit to form a partial input value in parallel form. DGT-MULT block produces the digit multiplication results of $(2X[j-1] + x_j 2^{-j})x_j$, $(2Y[j-1] + y_j 2^{-j})y_j$ and $(2Z[j-1] + z_j 2^{-j})z_j$. 5-to-2 CSA generates the carry-save form of intermediate residual, $W[j]$. S-SEL is a 2-digit CPA to retrieve the integer value of $W[j]$, $2(to_1+to_2+cout)+(WS_0+WC_0)$.

The above block diagram can be organized in a linear array structure of digit modules as shown in Fig. 7. Except the serial input signal which is connected to all modules, all other signals are between immediate neighbor modules. Fig. 8 gives the internal circuit of each digit module, which contains registers, input data appending, digit multiplication, and 5-to-2 carry-save addition (CSA). The function of digit multiplication slice for one vector element, Vx, is further given in Table 1. The precision of the computation can be increased by adding more modules to the array, while the clock period and on-line delay remain the same if wire delay is neglected.

The critical path in SO3S consists of register, DGT-MULT, 5-to-2 CSA and S-SEL. Each 5-to-2 CSA has 3 full adders (FAs). The maximum clock period of SO3S is

$$t_{reg} + t_{DGT-MULT} + t_{5-2-CSA} + t_{S-SEL}$$

Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)
0-7695-2667-5 $20.00 © 2001 IEEE

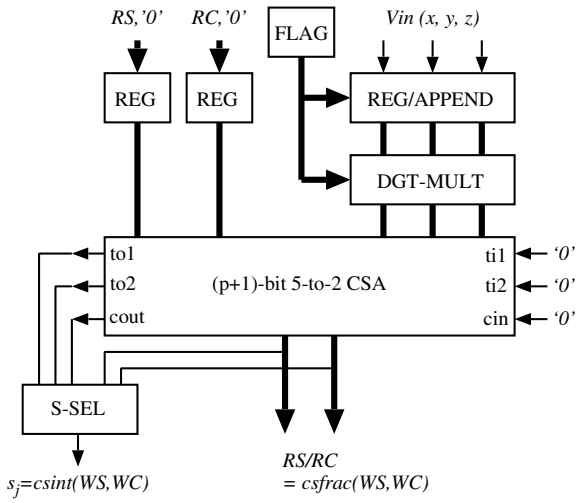


Figure 6: Block diagram of SO3S.

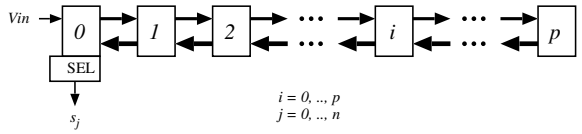
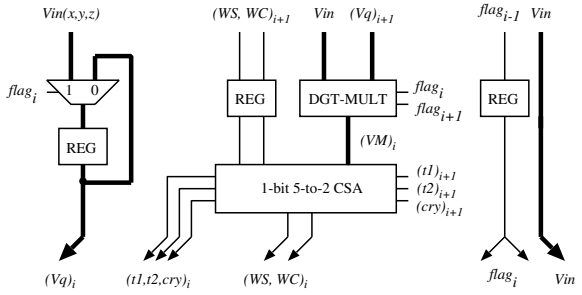


Figure 7: Linear array organization of SO3S.

which is about

$$t_{so3s} = t_{reg} + (t_{MUX} + t_{AND}) + 3t_{FA} + t_{3bCPA} \quad (4)$$

Since the digit selection function only depends on few most significant digits and the residual is propagated only single position each step towards the digit selection block, the number of digit modules can be reduced without affecting the result. Denote p as the the maximum number of fractional digit modules needed for generating results with n fractional digits. For



Vin, Vq, VM (in thick lines) are vectors with three elements x,y,z.
t1, t2 and cry are intermediate signals of 5-to-2 CSA.
$(Vq)_i$, $(t1,t2,cry)_i$ and $(WS,WC)_i$ flow left into (i-1)-th slice;
$flag_i$ flows both left into (i-1)-th slice and right into (i+1)-th slice;
Vin is connected to all slice modueles and can be viewed as flowing right.

Figure 8: Digit module of SO3S.

Table 1: Bit-level DGT-MULT in SO3S

V_x	$flag_i$	$flag_{i+1}$	$(V_{xM})_i$
0	–	–	0
1	0	0	$(V_q)_{i+1}$
1	0	1	0
1	1	0	1
1	1	1	–

$j \leq p$, j modules are actively working for the j-th recurrence step. For $j > p$, only p modules are available, which introduces a truncation error. In the 5-to-2 CSA case, an truncation at j-th bit introduces an error at the $(j-2)$-th bit of the result. Moreover, the error is shifted one bit left with the residual shifting. Overall, the truncation error propagates three bits left each recurrence step. In order not to affect the final result, p should be chosen to avoid the truncation error affecting the selection of the last (n-th) result digit:

$$p = \left\lceil \frac{3n-1}{4} \right\rceil \quad (5)$$

For a precision of 15 fractional bits, 11 fractional-digit modules are needed.

3.2 Square Root

The NOL algorithm for square root (Fig. 9) works on the output of SO3S, S ($0.25 \leq S < 3$), and generates the square root D ($0.5 \leq S < \sqrt{3}$). The input digits, s_j, are in the set of $\{0, 1, .., 8\}$ (radix-2). Note that $s_k = 0$ for $k > n$. The on-line delay is $\delta = 6$ and the digit selection function $dsel$ is

$$d_j = dsel(\{W[j]\}_2) = \begin{cases} 1 & \text{if } \{W[j]\}_2 \geq 0 \\ 0 & \text{if } \{W[j]\}_2 = -\frac{1}{4} \\ -1 & \text{if } \{W[j]\}_2 \leq -\frac{1}{2} \end{cases} \quad (6)$$

where $\{W[j]\}_2$ is an estimate with two fractional bits of $W[j]$.

Table 2: Bit-level DGT-MULT in SQRT

d_1d_0	$flag_i$	$flag_{i+1}$	M_i
00	–	–	0
10	0	0	$\overline{a_{i+1}}$
10	0	1	1
10	1	0	1
10	1	1	–
01	0	0	b_{i+1}
01	0	1	1
01	1	0	1
01	1	1	–
11	–	–	–

Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)
0-7695-2667-5 $20.00 © 2001 IEEE

– *Initialize* –
$W[-6] = 0$; $D[-1] = 0$;
– *Accumulate 5 input digits* –
for $j = -5, .., -1$ **do:**
$W[j] = 2(W[j-1] + s_{j+5}2^{-6})$;
end for
$R[-1] \leftarrow W[-1]$;
– *Recurrence* –
for $j = 0, 1, 2, ..., n$ **do:**
$W[j] = R[j-1] + s_{j+5}2^{-6}$;
$d_j \leftarrow dsel(\{W[j]\}_2)$;
$R[j] = 2W[j] - 2d_j D[j-1] - d_j^2 2^{-j}$;
$D[j] \leftarrow convert(D[j-1], d_j)$;
end for

Figure 9: NOL Algorithm for SQRT

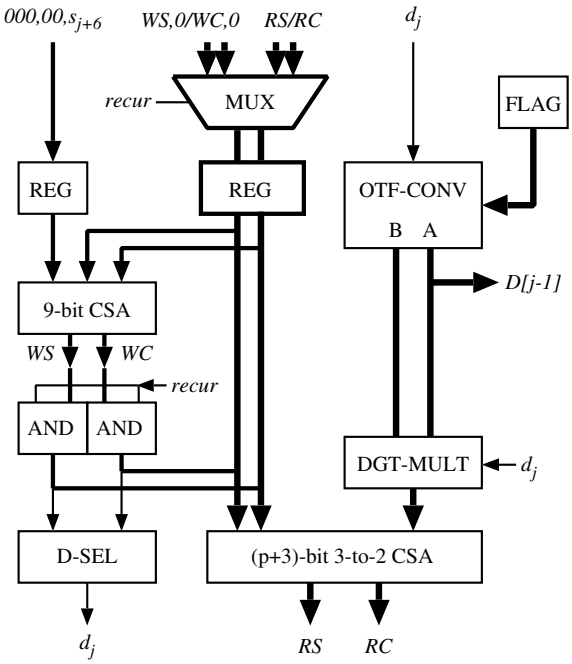


Figure 10: Block diagram of SQRT.

A block diagram of implementation with carry-save additions is given in Fig. 10. It combines the initial accumulating steps and recurrence steps. The signal "recur" controls whether the circuit is accumulating or recurring. Since the residual $R[j]$ is in carry-save form, a 9-bit 3-to-2 CSA is first needed to produce $W[j]$. Another $(p+1)$-bit 3-to-2 CSA is used to generate the residual $R[j]$. When "recur" is 0, the circuit accumulates the first 5 input digits using only the 8-bit CSA and registers. When "recur" is 1, the circuit executes the normal recurrence. D-SEL calculates the residual estimate $\{W[j]\}_2$ using a 5-bit CPA and makes digit selection decision using two-level combinational logic. OTF-CONV block converts the serial signed-digit output into parallel result, as further explained in Section 3.4. DGT-MULT block produces the digit multiplication results of $M[j] = -2d_j D[j-1] - d_j^2 2^{-j}$. It takes advantage of the two registered values from OTF-CONV, A and B. The bit-level function of DGT-MULT is given in Table 2.

The critical path in SQRT consists of a register, first 3-t-2 CSA, recurrence control logic, D-SEL, DGT-MULT and second 3-to-2 CSA, which determines the clock period

$$t_{sqrt} = t_{reg} + 4t_{AND} + 2t_{MUX} + 2t_{FA} + t_{5bCPA} \quad (7)$$

The maximum number of fractional-digit modules, p, is deduced in the same way as in the SO3S case and we obtain

$$p = \left\lceil \frac{2n+2}{3} \right\rceil \quad (8)$$

Note that p is not applicable to OTF-CONV since (n+1)-bit OTF-CONV is needed to produce results of $(n+1)$-bit precision.

3.3 Division

In division, the dividends, X, Y, or Z, are available in parallel form and the divisor, D, is in on-line form produced by SQRT. The NOL algorithm with such semi-online inputs is given in Fig. 11. Note that $d_k = 0$ for $k > n$. The on-line delay is $\delta = 4$ and the quotient digit selection function $qsel$ is

$$q_j = qsel(\{W[j]\}_3) = \begin{cases} 1 & \text{if } \{W[j]\}_3 \geq \frac{2}{8} \\ 0 & \text{if } -\frac{2}{8} \leq \{W[j]\}_3 = -\frac{1}{8} \\ -1 & \text{if } \{W[j]\}_3 \leq -\frac{3}{8} \end{cases} \quad (9)$$

where $\{W[j]\}_3$ is an estimate with three fractional bits of $W[j]$.

– *Initialize* –
$R[-1] = x \times 2^{-1}$; $Q[-1] = 0$;
– *Recurrence* –
for $j = 0, 1, 2, ..., n$ **do:**
$W[j] = 2R[j-1] - Q[j-1]2^{-3}d_{j+3}$;
$q_j \leftarrow qsel(\{W[j]\}_3)$;
$R[j] = W[j] - D[j+3]q_j$;
$Q[j] \leftarrow convert(Q[j-1], q_j)$;
end for

Figure 11: NOL Algorithm for DIV (semi-online)

Table 3: Bit-level DGT-MULT in DIV

$d_1 d_0$	m_j	$carry$
01	0	0
10	$\overline{v_j}$	1
01	v_j	0
11	–	–

The implementation diagram is given in Fig. 12. The residual computation in this algorithm is divided

Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)
0-7695-2667-5 $20.00 © 2001 IEEE

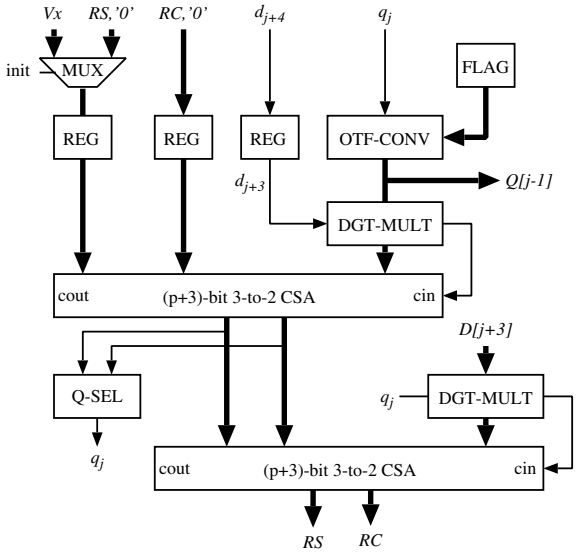


Figure 12: Block diagram of DIV.

into two steps. Therefore, two DGT-MULTs and two $(p+1)$-bit 3-to-2 CSAs are required. Each DGT-MULT block produces digit multiplication results of the form $M[j] = -dV[j]$. The bit-level function of DGT-MULT is shown in Table 3. Q-SEL selects quotient digit based on the residual estimate $\{W[j]\}_3$, using a 6-bit CPA and two-level combinational logic.

The critical path in DIV consists of a register, first DGT-MULT, and 3-to-2 CSA, Q-SEL, second DGT-MULT, and second 3-to-2 CSA. The clock period is

$$t_{div} = t_{reg} + 2t_{MUX} + 4t_{AND} + 2t_{FA} + t_{6bCPA} \quad (10)$$

The maximum number of fractional modules, p, is

$$p = \left\lceil \frac{3n+1}{4} \right\rceil \quad (11)$$

and p is also not applicable to OTF-CONV.

3.4 On-the-fly converter

In the NOL algorithms of SQRT and DIV, partial results $D[j-1]$ and $Q[j-1]$ in conventional form are needed during the recurrence step j. To avoid the variable-length shift operation, an appending form of on-the-fly converter (OTFC) is designed using the flag signal available in the design and thus requires less hardware. Two registers, A and B, are needed to store X and $X^* = X - 1$, respectively.

The radix-2 conversion algorithm is given in Table 4. For each bit i on the implementation level, the new values of a_i and b_i are set according to the input digit value and the flag value, as shown in Table 5. Fig. 13 gives the implementation detail of one bit slice.

Table 4: Radix-2 OTF-CONV

x_{k+1}	$A[k+1]$	$B[k+1]$
0	$(A[k], 0)$	$(B[k], 1)$
1	$(A[k], 1)$	$(A[k], 0)$
-1	$(B[k], 1)$	$(B[k], 0)$

Table 5: Bit-level OTF-CONV

$flag_i$	x_1x_0	a_i	b_i
1	$--$	$x_1 + x_0$	$\overline{(x_1 + x_0)}$
0	00	a_i	b_i
0	10	a_i	a_i
0	01	b_i	b_i
0	11	–	–

4. Pipelined on-line (POL) scheme

The potential problem with NOL scheme is its throughput limited by its latency. The throughput of the above NOL scheme is one vector every 16 cycles, which may not be acceptable in real-time graphics applications if the clock frequency is not high enough. To improve throughput, NOL scheme can be unfolded and pipelined to become POL scheme. The large number of vectors to be processed sequentially in graphics applications permits the use of a deep pipeline structure without any performance penalty [3].

For vector with $(n+1)$-bit precision, a fully unfolded POL scheme consists of $(n+1)$ stages. In the steady state, it computes up to $n+1$ different normalized vectors, the first stage producing the first digit of the k-th result and the last stage producing the last digit of the $(k-n)$-th result. Section 3 has shown that, in NOL scheme, only $p+1$ $(p < n)$ slice modules are needed to produce results with the precision of n fractional bits. For $j \leq p$, $j+1$ modules are actively involved in computation at stage j and therefore only $j+1$ modules are needed in POL's j-th stage. For $j > p$, a straightforward way of unfolding is to keep $p+1$ modules for all j-th stages. Since the truncation error propagates a few bits left in each recurrence step as explained in Section 3, the number of digit modules can be reduced

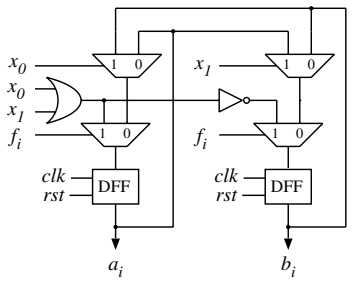


Figure 13: Digit slice of on-the-fly converter.

Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)
0-7695-2667-5 $20.00 © 2001 IEEE

accordingly. In POL scheme for SO3S (POL-SO3S), the number of digit modules can be reduced by 3 stage by stage when $j+1 > p$ because the truncation error propagates 3-bit left each stage. For $(n+1)$-bit POL-SO3S, the number of modules, NM_j, at stage j is

$$NM_j = \begin{cases} j+1 & \text{if } 0 \le j \le p \\ 3(n-j)+3 & \text{if } p < j \le n \end{cases} \quad (12)$$

For $(n+1)$-bit POL-SQRT, the number of modules, NM_j, at stage j is

$$NM_j = \begin{cases} j+1 & \text{if } 0 \le j \le p \\ 2(n-j)+6 & \text{if } p < j \le n \end{cases} \quad (13)$$

In the DIV case, the NOL scheme assumes that the inputs are in a semi-online way: the dividend is available in parallel form while the divisor is in serial on-line form. To apply proper truncation, the dividend also needs inputting in an on-line way. This requires minor modifications to the semi-online NOL-DIV algorithm in Fig. 11. The modified NOL-DIV algorithm is given in Fig. 14.

– Initialize –
$R[-1] = x_2 2^{-3} + x_1 2^{-2} + x_0 2^{-1}$; $Q[-1] = 0$;
– Recurrence –
for $j = 0, 1, 2, ..., n$ **do:**
$W[j] = 2R[j-1] + x_{j+3} 2^{-3} - Q[j-1] 2^{-3} d_{j+3}$;
$q_j \leftarrow qsel(\{W[j]\}_3)$;
$R[j] = W[j] - D[j+3] q_j$;
$Q[j] \leftarrow convert(Q[j-1], q_j)$;
end for

Figure 14: NOL Algorithm for DIV

For $(n+1)$-bit POL-DIV based on the above NOL-DIV algorithm, the number of modules, NM_j, at stage j is

$$NM_j = \begin{cases} j+1 & \text{if } 0 \le j \le p \\ 3(n-j)+7 & \text{if } p < j \le n \end{cases} \quad (14)$$

Similar to the restriction of p in Section 3.2 and 3.3, the above formulas of NM_j are also not applicable to OTF-CONV in POL-SQRT and POL-DIV. OTF-CONV has to be fully unfolded and pipelined without truncation (i.e., $(j+1)$ module modules are needed at stage j for $0 \le j \le n$).

An example of 16-bit POL scheme for SO3S (POL-SO3S) is given in Fig. 15. It is a two-dimensional array structure with 16 stages. The first module of each row includes the output digit selection module, S-SEL. Now, the residual signals flow in the vertical direction from one linear array to the next one, instead of shifting left as in Fig. 7 and 8. No recurrence indication signals (*flag*) or control signals (*recur* in SQRT) are needed since each stage itself indicates the proper digit input position and working position. The CSA internal signals and digit input signals still flow horizontally.

To match the pipeline on-line flow, a series of parallel vector inputs, $\{V_0, V_1, .., V_k\}$, need to be shifted in a stair-case way. This is done by a stair-case shifter array as shown in Fig. 16. The i-th digit of a vector is delayed by a i-bit shift register. Such a shifter array is only required at the beginning of a series of POL operation blocks. The output of POL-SO3S can be directly fed into POL-SQRT and the output of POL-SQRT into POL-DIV. Since on-the-fly converter, OTFC, is also pipelined, there is no need of an inverted stair-case shift array to re-align the final output. The normalized vector can be obtained directly from the final stage of the pipelined OTFC, at the rate of one vector per cycle.

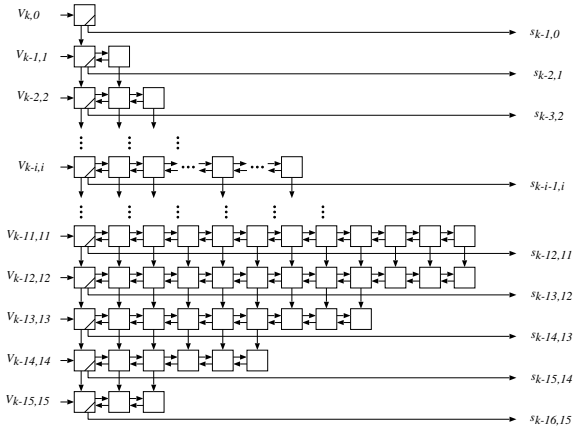


Figure 15: 16-bit POL scheme for So3S.

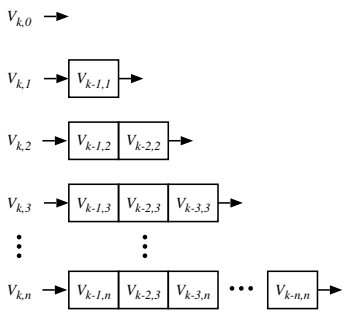


Figure 16: Stair-case input shifter array.

5. Theoretical evaluation

To get a quick evaluation of the proposed scheme, we now estimate the delays and areas of different schemes under the same metric. These delays and areas can vary somewhat depending on the technology and implementation, but the relative values should not vary significantly since all schemes use the same estimation metric.

The clock periods are estimated in terms of τ – the delay of a single complex gate, such as one FA. The basic gates like MUX2 and NAND2, have the delay of 0.5τ. The register delay, including setup time and clock-to-Q delay, is $t_{reg} = \tau$. A CPA is a carry lookahead adder (CLA) and a 6-bit CPA has a delay of 3τ.

The maximum clock period in POL-TDVN is similar to the DIV critical path delay expressed in Equation

Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)
0-7695-2667-5 $20.00 © 2001 IEEE

(10) and is estimated as 9τ. Since no recurrence indication or control signals are needed in POL, the related logic can be simplified, which can help reduce the cycle time of POL slightly compared to the NOL scheme and will not be considered in the comparisons.

The area is estimated as the number of equivalent NAND2 gates. The area estimation depends greatly on implementation technologies. Here we assume one FA has the area of 6.7 and a D-type and a D-type flip-flop (DFF) has the area of 4.0 equivalent gates [18]. For 16-bit POL scheme, there are about 3,743 DFFs, 968 FAs and 1,256 other miscellaneous gates, which are 22,714 equivalent gates in total. For 24-bit POL, there are about 7,523 DFFs, 1,812 FAs and 2,484 other miscellaneous gates for a total of 44,716 equivalent gates. In addition, the exponent unit are estimated to have 688 gates (72 DFFs and 400 combinational gates), assuming 8-bit exponent for both 16-bit and 24-bit POL schemes.

5.1 Comparison with PROVEN

PROVEN [3] is a 36-stage pipeline structure which also accepts one vector and produces one normalized vector each cycle. The input operands are treated as 33-bit 2's complement integers and the outputs are 16-bit 2's complement fixed-pointed numbers. The operations of square and sum are separated in two pipeline stages. Square unit uses standard array multipliers with exploitation of the equality of both operands. We assume that the carry-save results from square units are registered directly into the successive summing stage to reduce critical path delay. Its square-root unit is a pipelined 14-stage unit, producing one digit for every two input digits. The critical path in PROVEN is estimated as: register and 16-bit array multiplier, which has the delay of 17τ. PROVEN has about 1,812 DFFs, 1,238 FAs and 1,102 miscellaneous gates, which are equivalent to 16,645 NAND2 gates. The result of comparison with PROVEN is given in Table 6. Due to the use of on-line arithmetic, our POL scheme reduces the clock period by 47%, latency by 57%, and improves throughput by 89%, at the cost of 41% area increase.

5.2 Comparison with VU

VU [7] is a vector-processing unit for high-performance 3-D graphics computing. A VU has four major blocks: two execution unit blocks, a vector general-purpose register file (VGPR) and an integer register file. One execution unit block has four SIMD floating-point multiply-accumulate (fMAC) units. The other execution unit block has a floating-point divide/square root (fDIV) unit, an elementary function unit, a random number generation unit, a load/store unit, a branch, and a 16-bit integer unit. fMAC unit has a latency of 3 cycles and a throughput of 1 result every cycle. fDIV unit has a latency of 6 cycles for division or square root, and 12 cycles for composite operation $y/sqrt(x)$. fDIV unit has a throughput of 1 result every 7 cycles for division or square root, and 1 result every 13 cycles for composite operation $y/sqrt(x)$. The steps to compute 3-D vector normalization on VU are illustrated in Fig. 17. The latency is 20 cycles and the throughput is limited to 1 vector/13 cycles by fDIV unit.

The datapath in VU includes a register, a 48-bit adder and 24-bit shifter on the critical path [7]. The 48-bit adder is a carry-select adder with two-level CLA, which has an estimate delay of 6τ. The 24-bit shifter is constructed from multi-bit selectors such a 2-1, 4-1 and 8-1 multiplexer and its delay is estimated as 2τ. The total clock period is about 9τ. Most circuits in VU are implemented in precharged logic. Ide *et al.* [7] gives the number of transistors in VU: 64,000 for a fMAC and 30,500 for a fDIV. For the vector normalization to work, at least 3 fMACs and one fDIV are needed and hence the related area can be counted as 222,500 transistors or 55,625 gates in a conservative estimate. Such estimate might not be fair since VU is a graphics processor designed for multiple purposes. Here we only try to get a rough estimate of area. Table 7 gives the overall comparison result. To match the precision in VU, POL-TDVN is set to support data with significands of 24-bit precision and the 3-cycle latency for exponent processing is also included. The result indicates that POL can improve the throughput by 10.2X.

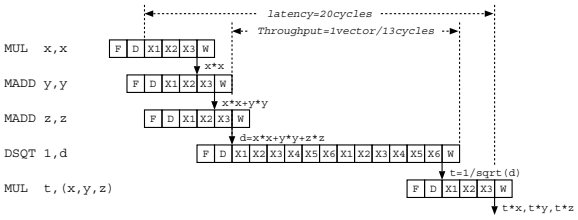


Figure 17: 3-D vector normalization on VU

If we assume that the throughput rate of VU has already met the requirements from graphics applications, it is interesting to see the effect of adjusting POL or NOL to have the same throughput ($1/117\tau$) as VU. This can be done either by slowing down POL's clock frequency or partial pipeline. If POL clock frequency is decreased from $1/9\tau$ to $1/117\tau$ (about 92%), the dynamic power dissipation will be decreased in the same percentage, 92%! Moreover, the circuits on the critical path can use much cheaper schemes, such as carry ripple adder (CRA) instead of CLA, which helps reduce the area. To have more area reduction, the original NOL can be partially pipelined. For the throughput of $1/117\tau$, a POL scheme only need to generate one normalized vector each 13 cycles. Recall that the NOL scheme has a throughput of 1 vector/37 cycles. Thus, NOL can be partially unfolded so that each stage recursively computes 3 cycles before pushing results into the next stage. The resulting partial POL scheme will have a throughput of 1 vector/13 cycles and only 1/3 area of the full POL.

6. FPGA implementation

The above evaluation and comparison show that POL scheme is superior under the same

Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)
0-7695-2667-5 $20.00 © 2001 IEEE

Table 6: Comparison with PROVEN (16-bit precision)

Design	Period	Latency	Throughput	Area (NAND2)
PROVEN	17τ	$36cycles = 612\tau$	$1vector/1cycle = 1/17\tau$	16,645
POL	9τ	$29cycles = 261\tau$	$1vector/1cycle = 1/9\tau$	23,402
Improvement	+47%	+57%	+ 89%	-41%

Table 7: Comparison with VU (24-bit precision)

Design	Period	Latency	Throughput	Area (NAND2)
VU	9τ	$20cycles = 180\tau$	$1vector/13cycle = 1/117\tau$	55,625
POL	9τ	$37cycles = 333\tau$	$1vector/1cycle = 1/9\tau$	45,404
Improvement	0	-85%	+10.2X	+18%

cost/performance metric. In practice, however, it is costly and time-consuming to develop an ASIC solution for only one operation. With the advance in FPGA technology, FPGA-based hardware accelerator is much more appealing. Such an accelerator can be field-programmed to speedup a different graphics operation or accommodate a new algorithm. Therefore, it is interesting to see the performance of FPGA-based TDVN accelerator versus general/special-purpose processor approaches.

Both NOL and POL schemes are specified in structural VHDL, simulated using ModelSim XE tool, and implemented in Xilinx FPGA environment [21]. NOL is mapped into xcv150-6-fg456 and POL is mapped into xcv1000-6-fg560. For 16-bit precision, the NOL scheme requires 928 4-input look-up tables (LUTs) and 462 DFFs. The POL scheme needs 4948 LUTs and 4043 DFFs. This implies that unfolding on-line arithmetic demands more sequential logic than combinational logic.

The performance of FPGA-based POL is compared with a graphics processor, VU, and a general-purpose processor, Intel Pentium III with Streaming SIMD extension (SSE), as shown in Table 8. It can be seen that FPGA-based POL scheme produces 85% more throughput than VU, and 2.3X more throughput than Pentium. However, the performance of FPGA-based NOL lags far behind the high-performance processor approach. From the table, we can also see that the clock period or frequency are improved when NOL is unfolded to become POL. This is because the critical path is simplified due to the absence of recurrence indication or control signals.

In the above experiment, little extra tuning for better performance was made except setting the option "optimize for speed" because our initial goal is to verify the design. Further improvements can be done in several ways. First, the implementation of shift registers can simplified using SRAMs in each Virtex slice [20]. In this way, shift register density can increase by approximately a factor of 16. This is meaningful to our schemes since NOL and POL utilize many serial-to-serial shift registers. Second, the fast carry chain in Virtex can be exploited to reduce carry propagation delay, which might be more efficient than CLA in FPGA technology. Third, the delay on speed critical outputs can be reduced by configuring them as fast outputs.

7. Conclusions

A pipelined on-line scheme for 3-D vector normalization, POL-TDVN, has been proposed. The clock period of this scheme is kept small by the use of redundant adders and low-precision estimates. The throughput is greatly improved by unfolding and pipelining the original recursive on-line scheme. The performance of POL-TDVN has been studied both theoretically and experimentally based on FPGA technology.

Assuming the same area/delay metric, POL scheme can perform vector normalization ten times faster than a recent vector-processing unit (VU) designed for 3-D graphics computing. When implemented as an FPGA-based hardware accelerator, POL improves throughput by 85% compared with VU, and by 2.3X compared with Pentium III SSE. This result implies that the bottleneck computation, 3-D vector normalization, can be moved from a processor into a field-programmable hardware accelerator to achieve better graphics performance. We have not analyzed the cost/performance of the interface between such an accelerator and the processor.

Acknowledgments. This research has been supported in part by Raytheon Corp. and Xilinx Corp. under California MICRO program and the NSF Grant MIP-9725771 "Effect of Redundancy in Arithmetic Operations on Processor Cycle Time, Architecture and Implementation."

References

[1] Bui-Tuong Phong, "Illumination for computer

Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)
0-7695-2667-5 $20.00 © 2001 IEEE

Table 8: FPGA-based Accelerator v.s. Processors

Scheme	Period(ns)	Freq(MHz)	Throughput(MVector/s)
NOL(FPGA)	26.1	38.3	2.4
POL(FPGA)	23.3	42.9	42.9
VU	3.33	300	23.1
Pentium	1.67	600	13.0

generated pictures", *Communications of the ACM*, vol.18, no.6, pp.311-317, June 1975.

[2] T. Moller, E. Haines, *Real-time Rendering*, A K Peters, Ltd., MA, 1999.

[3] G. Knittel, "PROVEN - Prompt vector normalizer", *Proc. 6th IEEE Int. ASIC Conf. and Exhibit*, pp.112-115, Sept. 1993.

[4] N. Takagi and S. Kuwahara, "Digit-recurrence algorithm for computing Euclidean norm of a 3D vector", *Proc. 14th Symp. Computer Arithmetic*, pp.86-93, Apr. 1999.

[5] N. Takagi and S. Kuwahara, "A VLSI algorithm for computing the Euclidean norm of a 3D Vector", *IEEE Trans. Computers*, vol.49, no.10, pp.1074-1082, Oct. 2000.

[6] M.D. Ercegovac and T. Lang, "On-line scheme for normalizing a 3-D vector", *Proc. 33rd Asilomar Conf. Signals, Systems, and Computers*, vol.2, pp.1460-1464 , Oct. 1999.

[7] N. Ide, etc, "2.44-GFLOPS 300-MHz floating-point vector-processing unit for high-performance 3-D graphics computing", *IEEE J. Solid-State Circuits*, vol.35, no. 7, pp.1025-1033, July 2000.

[8] M. Suzuoki, etc, "A microprocessor with a 128-Bit CPU, ten floating-point MAC's, four floating-point divisors, and an MPEG-2 decoder", *IEEE J. Solid-State Circuits*, vol.34, no.11, pp.1608-1618, Nov. 1999.

[9] Singh, S.; Bellec, P. "Virtual hardware for graphics applications using FPGAs", *Proc. IEEE Workshop on FPGAs for Custom Computing Machines*, pp.49-58, Apr. 1994.

[10] Culbertson, W.B.; Amerson, R.; Carter, R.J.; Kuekes, P.; Snider, G. "Exploring architectures for volume visualization on the Teramac custom computer", *Proc. IEEE Symposium on FPGAs for Custom Computing Machines*, pp.80-88, Apr. 1996.

[11] Ye, A.G.; Lewis, D.M. "Procedural texture mapping on FPGAs", *Proc. AGM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays (FPGA'99)*, pp.112-20, Feb. 1999.

[12] Styles, H.; Luk, W. "Customising graphics applications: techniques and programming interface", *Proc. 2000 IEEE Symposium on Field-Programmable Custom Computing Machines*, pp.77-87, Apr. 2000.

[13] M.D. Ercegovac, "On-line arithmetic: an overview", *Proc. SPIE (Real Time Signal Processing VII)*, vol.495, pp.86-93, Aug. 1984.

[14] M. D. Ercegovac and T. Lang, "On-the-fly conversion of redundant into conventional representations" *IEEE Trans. Computers*, vol.C-36, no.7, pp.895-897, July 1987.

[15] M.D. Ercegovac and T. Lang, "On-line arithmetic: a design methodology and applications in digital signal processing", *VLSI Signal Processing III*, pp.252-263, Nov. 1988.

[16] M. D. Ercegovac and T. Lang, "On-line scheme for computing rotation factors", *J. Parallel and Distributed Computing*, vol.5, no.3, pp.209-227, June 1988.

[17] P.K.-G. Tu, *On-line Arithmetic Algorithms for Efficient Implementation*, PhD Thesis, UCLA Comput. Sci. Dept., 1990.

[18] M. D. Ercegovac and T. Lang, *Digital Arithmetic*, Lecture Notes, UCLA Comput. Sci. Dept., 2000.

[19] P.K.-G. Tu and M. D. Ercegovac, "Gate array implementation of on-line algorithms for floating-point operations", *J. VLSI Signal Processing*, vol.3, no.4, pp.307-317, Oct. 1991.

[20] P. Alfke, "Efficient shifter registers, LFSR counters, and long pseudo-random sequence generators", *XAPP (Xilinx Application Notes)*, July 1996.

[21] *The Programmable Logic Data Book*, Xilinx Corporation, 1999.

Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)
0-7695-2667-5 $20.00 © 2001 IEEE