Hardware Efficient Architectures for Eigenvalue Computation

Yang Liu[1], Christos-Savvas Bouganis[1], Peter Y. K. Cheung[1]
Philip H.W. Leong[2] and Stephen J. Motley[1]
[1]Department of Electrical & Electronic Engineering, Imperial College London
[2]Department of Computing, Imperial College London
{yang.liu, christos-savvas.bouganis, p.cheung, philip.leong, stephen.motley}@imperial.ac.uk

Abstract

Eigenvalue computation is essential in many fields of science and engineering. For high performance and real-time applications, this may need to be done in hardware. This paper focuses on the exploration of hardware architectures which compute eigenvalues of symmetric matrices. We propose to use the Approximate Jacobi Method for general case symmetric matrix eigenvalue problem. The paper illustrates that the proposed architecture is more efficient than previous architectures reported in the literature. Moreover, for the special case of 3×3 *symmetric matrices, we propose to use an Algebraic Method. It is shown that the pipelined architecture based on the Algebraic Method has a significant advantage in terms of area.*

1. Introduction

Eigenvalue computation for symmetric matrices plays an important role in science and engineering. It is the core engine of many algorithms such as the Principal Component Analysis and a variety of applications including real-time operations such as optical flow computation [1]. Jacobi-like methods for eigenvalue computation have inherent parallelism and this underlying property makes them particular suitable for distributed resource system. Systolic array of processors [2] was originally proposed for VLSI and the advent of FPGA popularized this architecture [3, 4]. While previous implementations employed the *Exact Jacobi Method*, Gotze et al [5] proposed an approximation to the *Exact Jacobi method* and claimed that it is more efficient. Besides the Jacobi-based method, the *Algebraic Method* [6] offers an alternative approach that may be more efficient for some restricted classes of eigenvalue problems. All of these reported hardware architectures lack solid comparative data to allow a system designer to choose between them with confidence. In particular, modern FPGAs have rich hardware resources that are particularly suitable for high throughput eigenvalue computation. This paper provides the first comparative evaluation of these three different methods for computing eigenvalues in terms of resource utilization, performance and accuracy. It demonstrates that the *Approximate Jacobi Method* is more efficient than the exact Jacobi method. The novel contributions of this paper are: 1) The proposed architecture achieves one order of magnitude speed up compared with previous implementation for eigenvalue computation and it requires less area in the meantime; 2) A high throughput architecture is developed based on the *Algebraic Method* which is area efficient; 3) On implementing the *Algebraic Method*, we propose a hybrid algorithm to efficiently solve the $\arcsin(q/p)$ subproblem.

2. Hardware efficient eigenvalue computation methods

Three algorithms for eigenvalue computation using hardware are described in this section. These are the *Exact Jacobi*, the *Approximate Jacobi* and the *Algebraic Method*.

2.1. Exact Jacobi Method

Given a symmetric $n \times n$ matrix $\mathbf{A}$, Jacobi methods [7] work by systematically reducing the "norm" of the off-diagonal elements (1).

$$\mathcal{F}(\mathbf{A}) = \sqrt{\sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} a_{ij}^2} \tag{1}$$

This is accomplished by a sequence of orthogonal similarity transformations as in (2).

$$\begin{aligned} \mathbf{A}^{(k+1)} &= \mathbf{J}_{pq}^T \mathbf{A}^{(k)} \mathbf{J}_{pq}, \quad k = 0, 1, 2, ... \\ \text{with} \quad \mathbf{A}^{(0)} &= \mathbf{A} \end{aligned} \tag{2}$$

$\mathbf{J}_{pq}$ is called *Jacobi rotation* and is defined by the parameters $(c, s, -s, c)$ in the (pp, pq, qp, qq) entries of an $n \times n$ identity matrix. $c = \cos(\theta)$ and $s = \sin(\theta)$, where θ is

3-9810801-0-6/DATE06 © 2006 EDAA

the rotation angle. By applying (2) iteratively to make off-diagonal element zero, the diagonal elements tend to the eigenvalues of $\mathbf{A}$, denoted as λ_i.

Since $\mathbf{J}_{pq}$ is an orthogonal transformation, (3) holds,

$$\|\mathbf{A}^{(k+1)}\|_F = \|\mathbf{A}^{(k)}\|_F \tag{3}$$

where $||...||_F$ denotes the Frobenius norm. Each update in (2) affects only the p and q rows and columns of $\mathbf{A}^{(k)}$, thus

$$[\mathcal{F}(\mathbf{A}^{(k+1)})]^2 = [\mathcal{F}(\mathbf{A}^{(k)})]^2 - 2[(a_{pq}^{(k)})^2 - (a_{pq}^{(k+1)})^2] \tag{4}$$

With the exact Jacobi rotation, the coefficients c and s at each iteration are computed using (5) and (6) so that $a_{pq}^{(k+1)}$ becomes zero and hence maximal reduction of $\mathcal{F}(\mathbf{A}^{(k)})$ is achieved.

$$\begin{aligned} t &= \tan\theta = \frac{\text{sign}(\tau)}{|\tau| + \sqrt{1+\tau^2}} && (5)\\ c &= \frac{1}{\sqrt{1+t^2}}, \quad s = t \cdot c && (6)\\ \text{where} \quad \tau &= (a_{pq}^{(k)} - a_{pp}^{(k)})/2a_{pq}^{(k)} \end{aligned}$$

2.2. Approximate Jacobi Method

Approximate Jacobi Method [5] do not annihilate off-diagonal element $a_{pq}^{(k+1)}$ but reduce it by d as in (7).

$$\begin{aligned} a_{pq}^{(k+1)} &= d a_{pq}^{(k)} && (7)\\ \text{where} \quad d &= \frac{1 - 2\tau t - t^2}{1 + t^2} && (8) \end{aligned}$$

The maximal value of $|d|$, denoted as $|d|_{\max}$, is a measure for the quality of the approximation. *Exact Jacobi Method*, with t computed as in (5), yields $d = 0$ at each update. On the contrary, *Approximate Jacobi Method* approximate t with hardware efficient operators and hence

$$\tilde{t} = \text{sign}(\tau) \cdot 2^{-l} \approx t, \quad l \in \{0, 1, 2, ..., b\} \tag{9}$$

where b is the wordlength of the hardware system. The aim of the *Approximate Jacobi Method* is to choose l such that the approximate angle $\tilde{\theta} = \arctan(2^{-l})$ is the closest to the exact rotation angle θ and $|d|_{\max}$ is minimized.

Gotze et al [5] provide the following formulae which require at most three comparisons to find l while guaranteeing $|d|_{\max} \leq 1/3$.

$$k = \exp(|a_{qq} - a_{pp}|) - \exp(|a_{pq}|) \tag{10}$$

$$\begin{cases} \text{if } k \leq -2, & \text{then } l = 0 \\ \text{if } -2 < k \leq 0, & \text{then } l \in \{0, 1\} \\ \text{if } k > 0, & \text{then } l \in \{k-1, k, k+1\} \end{cases} \tag{11}$$

where exp denotes the exponent of a number.

2.3. Algebraic Method for 3×3 symmetric matrices

For a special case of 3×3 symmetric matrix $\mathbf{M}$, the eigenvalues can be calculated algebraically.

$$\mathbf{M} = \begin{pmatrix} a & d & e \\ d & b & f \\ e & f & c \end{pmatrix}$$

The eigenvalues can be expressed as roots of a third degree characteristic polynomial $p(\lambda) = \det(\mathbf{M} - \lambda\mathbf{I})$ [7]. Let this characteristic equation be

$$\lambda^3 + k\lambda^2 + l\lambda + m = 0 \tag{12}$$

where

$$\begin{aligned} k &= -(a + b + c) && (13)\\ l &= ab + bc + ac - f^2 - e^2 - d^2 \\ m &= af^2 - abc - 2fde + be^2 + cd^2 \end{aligned}$$

We substitute $\lambda = x - k/3$ to remove the λ^2 term in (12).

$$x^3 + px + q = 0 \tag{14}$$

where

$$p = -\frac{1}{3}k^2 + l, \;\; q = \frac{2}{27}k^3 - \frac{1}{3}lk + m \tag{15}$$

The condition of symmetric matrix $\mathbf{M}$ assures that all eigenvalues are real numbers [7]. Following [6], we substitute $x = \sqrt{-\frac{4p}{3}}y$, yielding

$$4y^3 - 3y = \frac{3q}{p\sqrt{-\frac{4p}{3}}}$$

Using the trigonometric relation $\cos 3\alpha = 4\cos^3\alpha - 3\cos\alpha$ and substituting $y = \cos\alpha$, we have

$$\cos 3\alpha = \frac{3q}{p\sqrt{-\frac{4p}{3}}}$$

The three real solutions to (14) are computed and the eigenvalues of $\mathbf{M}$ are

$$\begin{aligned} \lambda_1 &= x_1 - \frac{k}{3} = \beta\cos\alpha - \frac{k}{3} \\ \lambda_2 &= x_2 - \frac{k}{3} = \beta\cos\left(\alpha - \frac{2\pi}{3}\right) - \frac{k}{3} \\ \lambda_3 &= x_3 - \frac{k}{3} = \beta\cos\left(\alpha + \frac{2\pi}{3}\right) - \frac{k}{3} \end{aligned}$$

where

$$\beta = \sqrt{-\frac{4p}{3}}, \quad \alpha = \frac{1}{3}\left(\frac{\pi}{2} - \arcsin\frac{3q}{p\beta}\right) \tag{16}$$

The range of α, $\alpha \in (0, \pi/3)$, imposes an ordering to the acquired eigenvalues which is $\lambda_1 \geq \lambda_2 \geq \lambda_3$. This provides the designer of the system the flexibility to choose a particular eigenvalue.

3. Hardware realization of the eigenvalue methods

All three methods described in the last sections rely on the CORDIC algorithm for trigonometric functions. The CORDIC algorithm rotates a vector $[x, y]$ by an arbitrary angle θ in a hardware friendly way since it requires only iterations of shifts and additions. This is obtained by performing successive iterations of elementary angles, $\pm \arctan 2^{-i}$ with $i = 0, 1, ...b$. The CORDIC iteration equations are shown in (17) to (19).

$$\begin{aligned} x_{i+1} &= x_i - d_i \cdot y_i \cdot 2^{-i} && (17)\\ y_{i+1} &= y_i + d_i \cdot x_i \cdot 2^{-i} && (18)\\ \tau_{i+1} &= \tau_i + d_i \cdot \arctan(2^{-i}) && (19)\\ \text{where} \quad d_i &= \text{sign}(\tau_i) \end{aligned}$$

The resulting vector after $b+1$ iterations has to be scaled by

$$\frac{1}{K_b} = \prod_{i=0}^{b} \frac{1}{\sqrt{1+2^{-2i}}} \qquad (20)$$

3.1. Scaling factor of exact Jacobi rotations

The *exact* Jacobi rotations employ the CORDIC algorithm faithfully and the scaling factor K_b in (20) depends only on the wordlength b and can be pre-computed. The scaling can be executed with approximately $b/4$ shifts and additions [4].

3.2. Scaling factor of approximate Jacobi rotations

From (9), an approximate Jacobi rotation is equivalent to one iteration of the CORDIC algorithm. The scaling factor varies with l and it is given in (21).

$$\frac{1}{K_l} = \frac{1}{\sqrt{1+2^{-2l}}} \qquad (21)$$

As l increases, the scaling factor converges to unity. The number of $1/K_l$ needs to be stored is around $b/2$.

3.3. Architecture of the Approximate Jacobi Method

We adopted an upper triangular array rather than a full square array of processors in order to take full advantage of the symmetry of the matrix as illustrated in Figure 1. Each diagonal processor processes 3 elements of the matrix and each off-diagonal processor processes 4. Processors are only interconnected to their nearest neighbors to prevent broadcasting of signals.

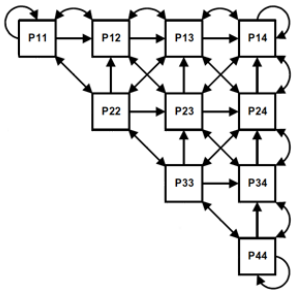

Figure 1. Upper triangular array of processors for 8×8 symmetric matrix eigenvalue computation

3.3.1. Diagonal Processor The block diagram of a diagonal processor is shown in Figure 2. On receiving an input matrix, all the diagonal processors compute the quantity l in parallel according to (10) and (11). l computed by each diagonal processor is transmitted to its neighboring off-diagonal processors in both horizontal and vertical directions. $\text{sign}(\tau)$ is transmitted in the identical way. Following analogous derivation to the update equations of *Exact Jacobi Method* [2], the approximate rotation blocks execute the update operations shown below.

$$\begin{aligned} a_{pp}^{(k+1)} &= a_{pp}^{(k)} - \text{sign}(\tau) 2^{-l+1} a_{pq}^{(k)} + 2^{-2l} a_{qq}^{(k)} \\ a_{pq}^{(k+1)} &= 2^{-2l} a_{pp}^{(k)} + \text{sign}(\tau) 2^{-l+1} a_{pq}^{(k)} + a_{qq}^{(k)} \\ a_{qq}^{(k+1)} &= \text{sign}(\tau) 2^{-l} (a_{pp}^{(k)} - a_{qq}^{(k)}) - 2^{-2l} a_{pq}^{(k)} + a_{pq}^{(k)} \end{aligned}$$

In the diagonal processor, the scaling factor $1/K_l$ has to be applied twice and hence the square root in (21) is removed. Therefore, we can execute scaling recursively using (22) with only shifts and additions [8].

$$1/K_{i+1}^2 = K_i^2 (1 + 2^{-2^{i+1} l}), \quad K_1^2 = 1 - 2^{-l} \qquad (22)$$

Given wordlength b, the recursion stops when $2^{i+1} l < b$. After the scaling, the updated matrix elements are exchanged with those in the adjacent processors.

3.3.2. Off-diagonal Processor The block diagram of an off-diagonal processor is shown in Figure 3. Each off-diagonal processor has to wait for the arrival of l and $\text{sign}(\tau)$ signals from the adjacent processors in both horizontal and vertical directions. In the next clock cycle, it passes on the signals to its vertical and horizontal neighboring off-diagonal processors.

Firstly, input matrix elements are subject to approximate rotation determined by the horizontally transmitted l_h and $\text{sign}(\tau)_h$ signals, which have been generated by the diagonal processor on the same row. The update operations consist of right-shifts and additions/subtrations are governed by the equations shown below.

$$a_{11h} = a_{11} - \text{sign}(\tau)_h 2^{-l_h} a_{21}$$

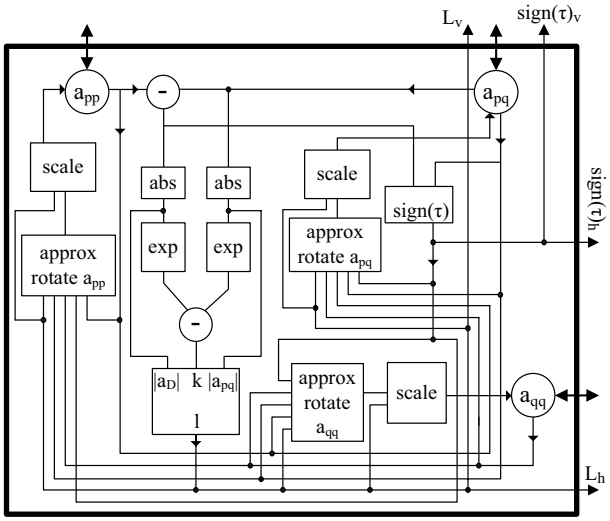

Figure 2. Block diagram of a Diagonal Processor

$$
\begin{aligned}
a_{12h} &= a_{12} - \mathrm{sign}(\tau)_h 2^{-l_h} a_{22} \\
a_{21h} &= a_{21} + \mathrm{sign}(\tau)_h 2^{-l_h} a_{11} \\
a_{22h} &= a_{22} + \mathrm{sign}(\tau)_h 2^{-l_h} a_{12}
\end{aligned}
$$

The scaling is carried out by multiplying one of the precomputed $1/K_l$ as in (21) selected by l_h via a multiplexer.

Secondly, the scaled results are subject to similar operations controlled by vertically transmitted signals to complete the matrix elements update. Finally, the updated matrix elements are exchanged with adjacent processors.

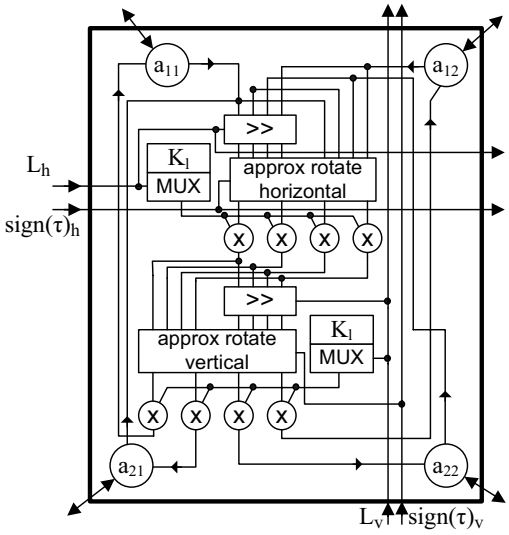

Figure 3. Block diagram of an Off-diagonal Processor

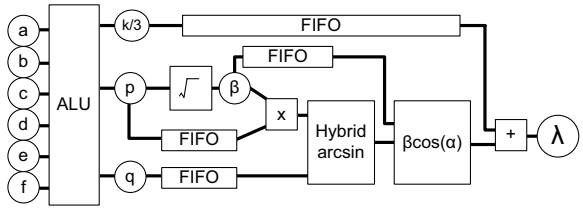

Figure 4. Overview of the eigenvalue computation system using the Algebraic Method.

3.4. Architecture of the Algebraic Method

We developed a pipelined architecture based on the *Algebraic Method* described in Section 2.3. Without loss of generality, the design computes the smallest eigenvalue of a 3×3 matrix. The overview of the design is shown in Figure 4.

3.4.1. ALU and Square Root modules ALU module contains adders and multipliers and it computes p, q and $k/3$ from (15) and (13) respectively. This module is the very first stage of the system so the wordlengths of output variables are determined such that early stage truncation error is avoided. Existence of common terms has been explored to reduce the number of operators in the ALU. The square root module adopts the successive approximation algorithm [9]. It is an iterative process that achieves one bit of accuracy per iteration.

3.4.2. Hybrid arcsin module We propose a hybrid method of general rotations and conventional CORDIC rotation to calculate $\arcsin(q/p)$ in (16), while avoiding the wide division. General rotations can be achieved by CORDIC rotations with the scaling factor compensated at every iteration as opposed to one scaling factor (20) at the end. Conventional arcsin CORDIC algorithm [8] has the same iteration equations as (17), (18) and (19) but different expression to calculate d_i and initial conditions as shown in (23) and (24).

$$
d_i = \begin{cases} +1, & y_i < K_b \cdot q \\ -1, & \text{otherwise} \end{cases} \tag{23}
$$

$$
x_0 = p, \quad y_0 = 0, \quad \tau_0 = 0 \tag{24}
$$

After b iterations, τ_b returns the value of $\arcsin(q/p)$. However, due to scaling factor at each iteration, incorrect decisions can be made and may lead to large errors as shown in Figure 5. The proposed hybrid algorithm ensures correct decision making for the first few rotations by compensating the scaling factors, i.e. general rotations, and applying the conventional CORDIC rotations in the subsequent rotations. With the proposed approach, a reasonable trade-off between computation and accuracy is achieved. In principle, accuracy is attained from the general rotation and the

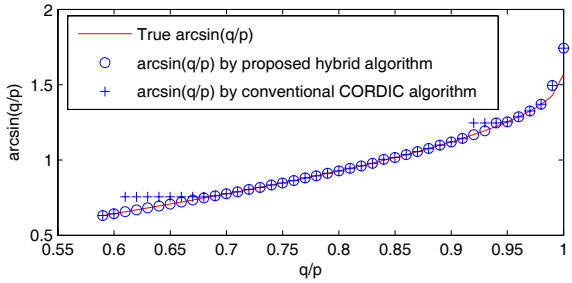

Figure 5. arcsin **by conventional CORDIC rotations and hybrid algorithm**

low computation cost is acquired from the CORDIC algorithm. Evidently, comparison from Figure 5 demonstrates the greatly reduced error achieved by the hybrid algorithm.

4. Performance Evaluation of Architectures

In this section, we firstly carry out architecture exploration on efficient implementation of the *Approximate Jacobi Method*. Then we compare pipelined architectures of the *Approximate Jacobi Method* and the *Algebraic Method*. The target system is the Xilinx XC2V6000-6 Virtex-II FPGA chip. In this paper, 16-bits system means that the inputs are 16-bits fixed-point numbers.

4.1. Area

Using the architecture in Section 3.3, systems that compute eigenvalues of matrices in various sizes have been generated and Table 1 shows that the proposed architecture uses less area than the *Exact Jacobi* architecture [4]. Our experimental data also shows that the area of the *Approximate Jacobi* architecture scales linearly with the wordlength of the input variables.

A diagonal processor occupies 1235 slices and an off-diagonal processor uses 2255 slices. Therefore, given a matrix size, we can predict the total area of the system for computing the eigenvalues by summing the processors area.

4.2. Speed

4.2.1. Exact Jacobi method The computation time for one step of exact Jacobi method is given by (25), where b is the wordlength, T_e is the latency for matrices interchange between processors and T_{sc} is the latency of the scaling compensation in (20) [4].

Matrix size	Exact Jacobi [4]	Approximate Jacobi
4×4	5507	4474
6×6	11296	10006
8×8	19013	17735

Table 1. Area comparison (in FPGA slices) of two architectures in 16-bits systems

$$T_{\text{exact}} = 23b + T_{sc} + T_e + 111 \tag{25}$$

4.2.2. Approximate Jacobi Method The *Approximate Jacobi Method* takes significantly less computation time. This is due to the fact that it is less dependent on b. Exact Jacobi method takes b CORDIC iterations to compute the cosine-sine pair in (6) and $2b$ CORDIC rotations to update the matrix. The *Approximate Jacobi Method* takes at most 3 comparisons to compute l in (11), which is the cosine-sine counterpart. Only one iteration of CORDIC is needed for every approximate rotation. The computation time for one step of our implementation of the *Approximate Jacobi Method* is given by (26).

$$T_{\text{approximate}} = 16 + \log_2\left(\frac{b}{2}\right) + T_e \tag{26}$$

For 16-bits systems under the same accuracy requirement for eigenvalues, the number of sweeps for the *Approximate Jacobi Method* is at most twice of that of the *Exact Jacobi Method* [5]. While using smaller area shown in Table 1, the proposed architecture is at least 12 times faster than the previously published exact Jacobi architecture [4] estimated from (25) and (26).

4.3. Truncation and Rounding Error of the Approximate Jacobi Method

With finite fixed-point accuracy, we discovered that although $\mathcal{F}(\mathbf{A})$ in (1) decreases to a certain minimum with approximate Jacobi rotations, the eigenvalues computed with excessive sweeps depart from the actual eigenvalues. This is due to truncation error propagating to the next sweep. If instead of executing truncation, we execute rounding, the eigenvalue instability is rectified. This improvement can be seen from Figure 6, where the maximum percentage error is calculated from (27).

$$\text{error}_{\max\,\%} = \frac{|\tilde{\lambda} - \lambda|_{\max}}{2^b} \tag{27}$$

$\tilde{\lambda}$ is the computed eigenvalue, λ is the quantized true eigenvalue and b is the system wordlength.

For 3000 random 4×4 matrices in 16-bits system, the maximum percentage errors are shown in Figure 6 for the proposed architecture. It is clear that the rounding stabilize the eigenvalue result.

However, the overhead in terms of area incurred by the rounding operation can not be ignored. For 4×4 16-bits systems, the area increase by 27% for the proposed architecture.

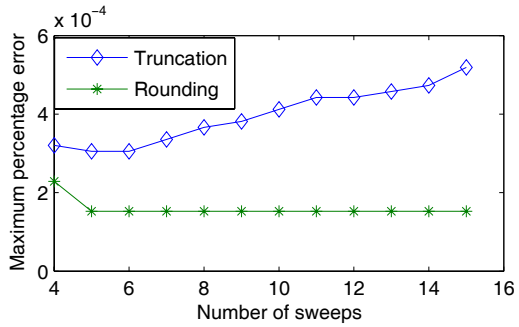

Figure 6. Max percentage error in the eigenvalues with the proposed architecture using rounding and truncation schemes.

4.4. Hardware multiplier

For some hardware architectures, e.g. FPGAs, that have logic and embedded multipliers, the available resources can be exploited for multiplication operations. This is beneficial since the scaling in the off-diagonal processors is achieved through multiplication. In a 4×4 16-bits system, two 18×18 embedded multipliers trade for 1343 slices.

4.5. Pipelined architectures for 3×3 symmetric matrices

In very high speed applications, such as real time optical flow computation [1], massive throughput of the smallest eigenvalues of 3×3 matrices is required. We compare the *Algebraic Method* and the *Approximate Jacobi Method*. Both architectures have been pipelined and designed to return the smallest eigenvalue of a matrix at every clock cycle. In order to minimize area, there is no need to implement the systolic array but to lay off the redundant operation of the off-diagonal processor.

Inputs of the optical flow computation are normally 8-bits. Empirically, 13-bits is sufficient for eigenvalue computation in order to render accurate optical flow. The comparison between the *Algebraic Method* and the *Approximate Jacobi Method* on such 13-bits systems is summarized in Table 2. The *Approximate Jacobi Method* is unrolled and pipelined and its behavior is equivalent to 3 sweeps of its un-pipelined version.

Scheme	Algebraic Method	Approximate Jacobi Method
Area (slices)	4556	20591
Max Frequency	62MHz	70MHz
Max % error	13.8×10^{-3}	2.44×10^{-3}

Table 2. Comparison of two pipelined architectures in 13-bits accuracy

The results show that the *Algebraic Method* implementation is considerably smaller than the *Approximate Jacobi Method*. However, due to its "open loop" characteristic, its maximum percentage error is 5.65 times that of the *Approximate Jacobi Method*.

5. Conclusion

This paper explores efficient architectures for eigenvalue computation based on the *Approximate Jacobi Method* and the *Algebraic Method*. In terms of area and speed, the proposed Approximate Jacobi architecture is smaller than the implementations that are reported in the literature while achieving one order of magnitude speed up. We also propose an architecture resorting to the *Algebraic Method* to compute the smallest eigenvalues for 3×3 symmetric matrices and successfully achieve very high throughput with much smaller area compared to the Jacobi methods.

Future work will focus on more thorough error analysis and numerical stability of the three methods in the context of efficient hardware architecture.

References

[1] G. Farneback, "Fast and accurate motion estimation using orientation tensors and parametric motion models," *ICPR Proceedings*, vol. 1, pp. 135 – 139, 2000.

[2] R. P. Brent, F. T. Luk, and C. Van Loan, "Computation of the singular value decomposition using mesh-connected processors.," *Journal of VLSI and Computer Systems*, vol. 1, no. 3, pp. 242 – 270, 1985.

[3] M. Kim, K. Ichige, and H. Arai, "Design of Jacobi EVD processor based on CORDIC for DOA estimation with MUSIC algorithm," *PIMRC Proceedings*, vol. 1, pp. 120 – 4, 2002.

[4] A. Ahmedsaid, A. Amira, and A. Bouridane, "Improved SVD systolic array and implementation on FPGA," *FPT Proceedings*, pp. 35 – 42, 2003.

[5] J. Gotze, S. Paul, and M. Sauer, "An efficient Jacobi-like algorithm for parallel eigenvalue computation," *IEEE Transactions on Computers*, vol. 42, no. 9, pp. 1058 – 65, Sept. 1993.

[6] Waerden and B. L. van der, *Algebra*. New York ; London: Springer-Verlag, 1991. translated by Fred Blum and John R. Schulenberger. Vol.1.

[7] G. H. Golub and C. F. Van Loan, *Matrix computations*. Baltimore; London: Johns Hopkins University Press, 3rd ed., 1996.

[8] J.-M. Delosme, "CORDIC algorithms: theory and extensions," *Proceedings of the SPIE - The International Society for Optical Engineering*, vol. 1152, pp. 131 – 45, 1989.

[9] D. M. Mandelbaum, "Method for calculation of the square root using combinatorial logic," *Journal of VLSI Signal Processing*, vol. 6, no. 3, pp. 233 – 242, 1993.