Microprocessors and Microsystems 38 (2014) 581–584

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Exploiting processor features to implement error detection in reduced precision matrix multiplications

Pedro Reviriego [a,*], Serdar Zafer Can [b,2], Çağrı Eryılmaz [c,3], Juan Antonio Maestro [a,1], Oğuz Ergin [b,2]

[a] Universidad Antonio de Nebrija, C/Pirineos, 55 E-28040 Madrid, Spain
[b] TOBB University of Economics and Technology, Ankara, Turkey
[c] Middle East Technical University, Ankara, Turkey

ARTICLE INFO

Article history:
Received 18 June 2013
Revised 17 February 2014
Accepted 1 May 2014
Available online 10 May 2014

Keywords:
Matrix multiplication
Error detection
Soft errors

ABSTRACT

Modern processors incorporate complex arithmetic units that can work with large word-lengths. Those units are useful for applications that require high precision. There are however, many applications for which the use of reduced precision is sufficient. In those cases, one possibility is to use the large word-length arithmetic units to implement reduced precision operations with additional error detection. In this paper, this idea is explored for the case of matrix multiplications. A technique is presented and evaluated. The results show that it can detect most errors and that for large matrixes the overhead in terms of execution time is small.

1. Introduction

Matrix multiplication is an operation commonly found in many computing applications. When implemented in a processor, matrix multiplications make an intensive use of the arithmetic processing units. Those, as the rest of the elements of the processor suffer soft errors [1] that can affect the functionality of the system. These errors are especially important in complex circuits such as multicore processors. Ensuring the dependability of advanced multicore processors at nanoscale is critical to enable their use in applications such as automotive, medical or space systems [2]. To mitigate those errors, protection techniques can be used at different levels [3]. For example, the modifications can be introduced in the manufacturing process or in the low level design of the processor. Mitigation can also be applied on a given processor by implementing redundancy and checks in software [4]. For applications that have a regular structure and algorithmic properties, another option is to exploit those features to mitigate errors. This is known as Algorithmic Based Fault Tolerance (ABFT) and has been applied to different applications like for example signal processing [5]. ABFT has also been used for matrix multiplications, see for example [6,7]. In most of the proposed techniques, the objective is to detect errors with the minimum number of additional operations. The ABFT techniques are well suited to multicore processors in which hardware resources are abundant and some of them can be dedicated to implement error detection and correction when the application is critical.

In this paper, the protection of matrix multiplications is considered from this perspective. A processor with an arithmetic unit that supports large word-lengths is used for the implementation of reduced precision matrix multiplications. The use of these narrow values is common in many applications. In this case, one possibility is to exploit these narrow values to provide error detection and correction features. This has already been done for register files [8]. In this paper, the same idea is explored for matrix multiplications. Namely, the use of the large word-length arithmetic units to implement reduced precision matrix multiplications with error detection capabilities is explored.

The rest of the paper is organized as follows. In Section 2, the proposed scheme is presented. Then in Section 3 it is evaluated using a CPU simulator to show the effectiveness both in terms of error detection and required overhead. Finally the conclusions are summarized in Section 4.

2. Proposed scheme

Let us assume that the two matrixes A, B to be multiplied are composed of integers represented with f bits and have a size $n \times n$. The product is also a matrix C of size $n \times n$ whose elements are obtained as follows:

* Corresponding author. Tel.: +34 914521100; fax: +34 914521110.
E-mail addresses: previrie@nebrija.es (P. Reviriego), serzafcan@etu.edu.th (S.Z. Can), eryilmazcg@gmail.com (Ç. Eryılmaz), jmaestro@nebrija.es (J.A. Maestro), oergin@etu.edu.th (O. Ergin).
[1] Tel.: +34 914521100; fax: +34 914521110.
[2] Tel.: +90 3122924290.
[3] Tel.: +90 5078677682.

http://dx.doi.org/10.1016/j.micpro.2014.05.001
0141-9331/

$$c_{ij} = \sum_{l=1}^{n} a_{il} \cdot b_{lj} \quad (1)$$

As the elements of the product matrix C have typically also f bits, a truncation (right shift by f bits) is done as the final step to obtain the values c_{ij}. This truncation is the same as multiplying by 2^{-f}.

The proposed scheme is based on arithmetic residues that have been used for error detection in multipliers and other arithmetic operations [9]. The residue modulo m of a number x is defined as:

$$\langle x \rangle_m = \mathrm{mod}(x, m) \quad (2)$$

One way to detect errors using arithmetic residues is to implement a redundant version of the operation using the residues instead of the values. In the case of matrix multiplication this can be computed as follows:

$$\langle c_{ij} \rangle_m = \left\langle \sum_{l=1}^{n} \langle a_{il} \rangle_m \cdot \langle b_{lj} \rangle_m \right\rangle_m \quad (3)$$

This is a much simpler computation than (1) as the input values can only take m values. Then the result of (3) can be compared with the residue of the value obtained in (1)

$$\langle c_{ij} \rangle_m = \left\langle \sum_{l=1}^{n} a_{il} \cdot b_{lj} \right\rangle_m \quad (4)$$

When the two values are different an error is detected. This approach has been used for multipliers [9] and also for more complex operations like filters [10]. The scheme is effective in detecting most errors when $m = 2^k - 1$. Typical values for m are 3 and 7.

In our case, since the idea is to use the large word-length arithmetic unit to add error detection features, the previous scheme is not directly applicable. The proposed scheme is based on modifying one of the matrixes, for example A to obtain a new matrix A′ such that each element of the new matrix has a residue of zero:

$$\langle a'_{ij} \rangle_m = 0 \quad (5)$$

This is easily done by making:

$$a'_{ij} = a_{ij} - \langle a_{ij} \rangle_m \quad (6)$$

Then matrix multiplication can be done to obtain C′ = A′ × B

$$c'_{ij} = \sum_{l=1}^{n} a'_{il} \cdot b_{lj} \quad (7)$$

and by construction:

$$\langle c'_{ij} \rangle_m = 0 \quad (8)$$

Therefore, errors can be detected by simply checking if the residue values of the obtained matrix C′ are different from zero.

This scheme can be effective in detecting errors, but introduces some errors in the computation of the matrix multiplication as the result obtained is C′ and not C. The use of large word-length arithmetic units can mitigate this issue. More precisely, if an arithmetic unit with words of $2f$ bits is used, the matrix A can be modified as follows:

$$\langle a'_{ij} \rangle_m = 2^f \cdot a_{ij} - \left\langle 2^f \cdot a_{ij} \right\rangle_m \quad (9)$$

which has now a length of $2f$ bits.

Then the matrix C′ would have elements for which the residue is also zero. Therefore errors can be detected by checking if the residue is zero. Finally a matrix very similar to the desired C matrix can be obtained by doing:

$$c''_{ij} = c'_{ij} \cdot 2^{-f} \quad (10)$$

In summary, the proposed scheme is as follows:

1. Multiply the elements of A by 2^f (left shift).
2. Modify the elements obtained in (1) such that their residue modulo m is zero.
3. Perform matrix multiplication using the arithmetic unit with $2f$ bits.
4. Check if the residues modulo m of the matrix elements obtained in (3) are zero. If not an error is detected. Note that the check is done on the elements c'_{ij} (left hand side of (7)) and not on each multiplication operation (right hand side of (7)).
5. Divide the elements obtained by 2^f (right shift) to obtain the final result.

The key observation is that now the error introduced to make the residue of the elements in A′ equal to zero is divided by 2^f in step 5 and therefore its impact is greatly reduced. This means that after the final truncation of the results to f bits, the error will be in most cases eliminated. An upper-bound on the probability that the error is not eliminated can be easily obtained. For a multiplication of matrixes of size $n \times n$ each element is the addition of n products. The maximum error in the final value of each product is $m - 1$. Since n multiplications are needed to compute an element in C, the maximum error in the computation of an element will be $n \times (m - 1)$. This error is before the division by 2^f in step 5. Therefore, the final error will be at most $n \times (m - 1) \times 2^{-f}$. In many cases, $2^f > n \times (m - 1)$ and therefore, the error can only affect the least significant bit of the computed element. Assuming that the computed elements are uniformly distributed, this will occur with a probability of $n \times (m-1) \times 2^{-f}$. The magnitude of this upper bound can be illustrated with an example, when $f = 16$, $m = 3$ *and* $n = 500$, the upper bound on the error probability on the least significant bit would be 0.0153. The actual probability of error will be much lower than the upper bound as in the derivation of the upper bound it is assumed that all errors add and have the maximum value which is very unlikely. This will be corroborated by the evaluation results presented in the following section.

The overhead required to implement the scheme is basically two modulo operations per matrix element (steps 2 and 4) and a subtraction in step 2. The rest of the operations are simply shifts or comparisons. Therefore, the overhead per element is constant regardless of the matrix size. Since the complexity per element of the computation of matrix multiplication grows with the matrix size, this means that the relative overhead will be lower for large matrixes.

The effectiveness of the scheme is expected to be good for soft errors in the multipliers of the ALU as the residue codes have been proved to be effective to protect multipliers [9]. The effectiveness will depend on the type of error that is inserted. In the following for simplicity, a single bit error at the output of the multiplier is assumed but the error types described in [9] should also be detected. Permanent errors can affect several terms of the computation of an element, and therefore error masking can occur in that case. The study of the effectiveness to protect against permanent errors is not considered further in this paper and is left for future work.

Another important aspect is that the result obtained may contain small errors due to the modification introduced in step 2. Therefore we need to ensure that this effect is negligible.

In order to evaluate the effectiveness, overhead and accuracy of the proposed scheme, an evaluation is presented in the next section.

3. Evaluation

The proposed scheme is evaluated in terms of effectiveness and overhead through a case study. To that end, the gem5 simulator

[11] with 32-bit ARM instruction set architecture (ISA) is used. The proposed matrix multiplication procedure has been implemented in C++ language and compiled with a cross-compiler to be able to run on the simulator. The standard matrix multiplication is also implemented for comparison.

Faults are injected when the multiplications are executed. One error is injected per matrix multiplication run and affects one bit of one of its multiplications. The operation on which the error is inserted and the bit affected are randomly selected. The matrix multiplications are performed with $f = 16$ bits using the 32 bit ALU. The module operations are done with $m = 3$. Three different matrix sizes are used, $n = 10$,100 and 500. For each of the configurations, two hundred simulations were run.

The test sequence is as follows, in the first phase, the program takes a seed to generate the elements of the matrix randomly for the given size. These matrixes are used for both the traditional and the modified matrix multiplications. The numbers are generated with 16 bits. After generation, matrixes are multiplied following the five steps procedure described in Section 2 to detect errors. Then the results are compared with those of a traditional multiplication.

The results are evaluated in three aspects:

1. Overhead in terms of execution time.
2. Effectiveness in terms of error detection.
3. Effects of the modified procedure on the results.

To assess the overhead of the proposed scheme, the execution time was measured for both the modified and the traditional matrix multiplication. The ratio of both for different matrix sizes are summarized in Table 1. It can be observed that the overhead is large for small matrixes but small for large matrixes. In fact, for 500×500 matrixes the overhead is only 1%. This is expected as the overhead of the technique per matrix element is constant and the complexity of matrix multiplication for each element grows with matrix size as mentioned in the previous section. Therefore, the proposed technique is effective in terms of overhead for matrixes larger than 100×100.

To assess the effectiveness of the proposed error detection mechanism in each simulation, whether or not the inserted error was detected was logged. The results show that all errors are detected. This is consistent with the error model used that was a single bit error as explained before. When $m = 2^l - 1$, all single bit errors are detected as $\mathrm{mod}(2^i, m) \neq 0$.

In Section 2, it was mentioned that the proposed scheme can introduce some minor errors on the result matrix. These are due to the modification introduced in matrix A to ensure that modulo m of its elements is zero (see Eq. (9)). To measure the importance of this effect, an additional simulation run was done with the modified procedure but without inserting errors. The results were then compared with those of the standard matrix multiplication. It was observed that the maximum value of the observed differences was one. This corresponds to an error in the least significant bit. In Fig. 1, the percentage of matrix multiplications affected by at least one error is shown. It can be observed that the percentage grows with matrix size. This is expected as larger matrixes have more elements and therefore more chances of experience errors. Also the upper-bound derived in the previous section grows with matrix size.

Table 1
Execution time overhead.

Matrix size	10×10	100×100	500×500
Ratio of modified vs standard	1.40	1.04	1.01

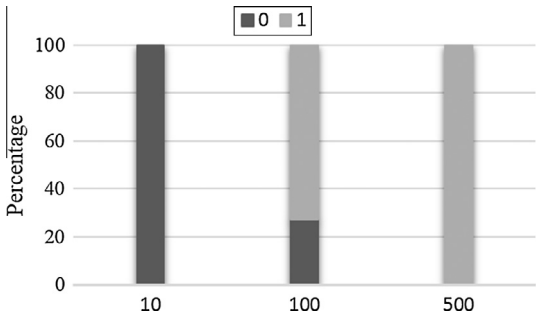

Fig. 1. Percentage of matrix multiplications affected by at least one error in the modified procedure.

Table 2
Probability of errors in the proposed scheme.

Matrix size	10×10	100×100	500×500
Measured probability of error	0.0%	0.01%	0.03%
Upper bound $(n \times (m-1) \times 2^{-f})$	0.03%	0.31%	1.53%

To provide a more detailed analysis of this effect, the probability of error for each element is given in Table 2 and compared with the upper bound derived in Section 2. It can be observed that the probability of failure grows with matrix size but is very small in all cases. Therefore, its effect would be negligible for most applications.

The upper-bound is much larger than the actual values obtained in simulation. This is again expected as the upper-bound assumes a worst case. A more elaborated theoretical analysis could be done to derive a tighter upper-bound based on assumptions about the statistical distribution of the elements of the matrixes. The development of this refined upper bound is left for future work.

4. Conclusions

In this paper a technique to implement error detection in matrix multiplications has been presented. The scheme considers the implementation of reduced precision matrix multiplication on a processor with extended precision arithmetic units. In that case, it is shown that by using arithmetic residue codes it is possible to efficiently implement error detection. The proposed technique modifies one of the input matrixes to ensure that the product matrix has a zero residue in all its elements. This is then used to detect errors.

The proposed scheme has been evaluated to show its effectiveness in a realistic case study. The results show that the required overhead is small for large matrixes and 100% detection rate is achieved in all the configurations studied. Therefore, the proposed technique can be of interest to implement error detection in reduced precision matrix multiplications.

Future work will focus on extending the proposed scheme to other common computer applications that make an intensive used of the processors arithmetic units.

Acknowledgements

This work was supported by the Spanish Ministry of Science and Education under Grant AYA2009-13300-C03 and by the Scientific and Technological Research Council of Turkey (TUBITAK) under research grant 112E004. This paper is part of a collaboration in the framework of COST ICT Action 1103 "Manufacturable and Dependable Multicore Architectures at Nanoscale".

References

[1] S.K. Reinhardt, S.S. Mukherjee, J. Emer, The soft error problem: an architectural perspective, in: Proc. of the 11th International Symposium on High-Performance Computer, Architecture, 2005.

[2] COST Action IC1103: Manufacturable and dependable multicore architectures at nanoscale "MEDIAN" <http://www.median-project.eu/>.

[3] M. Nicolaidis, Design for soft error mitigation, IEEE Trans. Device Mater. Reliab. 5 (3) (2005).

[4] G.A. Reis, J. Chang, N. Vachharajani, R. Rangan, D.I. August, SWIFT: software implemented fault tolerance, in: Proc. of the International Symposium on Code Generation and Optimization, 2005, pp. 243–254.

[5] A. Reddy, P. Banarjee, Algorithm-based fault detection for signal processing applications, IEEE Trans. Comput. 39 (10) (1990) 1304–1308.

[6] K.-H. Huang, J.A. Abraham, Algorithm-based fault tolerance for matrix operations, IEEE Trans. Comput. 33 (6) (1984) 518–528.

[7] C. Argyrides, C.A.L. Lisbôa, D.K. Pradhan, L. Carro, A fast error correction technique for matrix multiplication algorithms, in: Proc of the IEEE International On Line Test Symposium, 2009, pp. 133–137.

[8] O. Ergin, O. Unsal, X. Vera, A. González, Exploiting narrow values for soft error tolerance, IEEE Comput. Architec. Lett. 5 (2006).

[9] U. Sparmann, S.M. Reddy, On the effectiveness of residue code checking for parallel two's complement multipliers, in: proc of the 24th International Symposium On Fault-Tolerant, Computing (FTCS-24), 1994, pp. 219–228.

[10] Z. Gao, P. Reviriego, W. Pan, Z. Xu, M. Zhao, J. Wang, J.A. Maestro, Efficient arithmetic residue based SEU-tolerant FIR filter design, IEEE Trans. Circ. Syst. II 60 (8) (2013) 497–501.

[11] N. Binkert et al., The gem5 Simulator, ACM SIGARCH Comput. Architec. News 39 (2) (2011).

Pedro Reviriego received the M.Sc. and Ph.D. degrees (with honors) in telecommunications engineering from the Technical University of Madrid, Madrid, Spain, in 1994 and 1997, respectively. From 1997 to 2000, he was an R&D Engineer with Teldat, Madrid, working on router implementation. In 2000, he was with Massana to work on the development of 1000Base-T transceivers. During 2003, he was a Visiting Professor with University Carlos III, Leganés, Madrid. From 2004 to 2007, he was a Distinguished Member of Technical Staff with LSI Corporation working on the development of Ethernet transceivers. He is currently with the Universidad Antonio de Nebrija, Madrid. His research interests are soft errors and reliability, fault-tolerant systems, and energy-efficient communication networks. He has authored numerous papers in international conferences and journals. He has also participated in the IEEE 802.3 standardization for 10GBase-T.

Serdar Zafer Can received the BS degree in electrical and electronics engineering from TOBB University of Economics and Technology, Ankara, Turkey, in 2012 and he is now MS student in computer engineering in TOBB University of Economics and Technology, Ankara. His current research interests include computer architecture, reliability, VLSI design and reconfigurable architectures.

Çağrı Eryılmaz is pursuing BSc degree in Electrical and Electronics Engineering from Middle East Technical University (METU), Ankara, Turkey. He also pursues Information Systems Minor Degree of Computer Engineering Department from METU. He has been a productive researcher during his BSc degree; he joined Kasirga Informatics and BSC-Microsoft Research Centre as a research intern. As of June 2014, he will finish his studies at Middle East Technical University and he will attend to the University of Texas at Austin for M.Sc. and Ph.D. studies. Some of the research interests of him include fault tolerant computing, high performance systems, low power microarchitectures and heterogeneous architectures.

Juan Antonio Maestro received the M.Sc. degree in physics and the Ph.D. degree in computer science from Universidad Complutense de Madrid, Madrid, Spain, in 1994 and 1999, respectively. He has served both as a Lecturer and a Researcher at several universities, such as the Universidad Complutense de Madrid; the Universidad Nacional de Educación a Distancia (Open University), Madrid; Saint Louis University, Madrid; and the Universidad Antonio de Nebrija, Madrid, where he currently manages the Computer Architecture and Technology Group. His current activities are oriented to the space field, with several projects on reliability and radiation protection, as well as collaborations with the European Space Agency. Aside from this, he has worked for several multinational companies, managing projects as a Project Management Professional and organizing support departments. He is the author of numerous technical publications, both in journals and international conferences. His areas of interest include high-level synthesis and cosynthesis, signal processing, and real-time systems, fault tolerance, and reliability.

Oguz Ergin received the BS degree in electrical and electronics engineering from the Middle East Technical University, Ankara, Turkey, in 2000 and the MS and PhD degrees in computer science from the State University of New York, Binghamton, in 2003 and 2005, respectively. He was a senior research scientist at the Intel Barcelona Research Center for 15 months during 2004 and 2005. He is currently an assistant professor in the Department of Computer Engineering, TOBB University of Economics and Technology, Ankara. His current research interests include VLSI design, energy-efficient and high-performance computer architectures, and dependable/reliable systems.