Fuzzy Memoization for Floating-Point Multimedia Applications

Carlos Álvarez, Jesús Corbal, and Mateo Valero, *Fellow*, *IEEE*

Abstract—Instruction memoization is a promising technique to reduce the power consumption and increase the performance of future low-end/mobile multimedia systems. Power and performance efficiency can be improved by reusing instances of an already executed operation. Unfortunately, this technique may not always be worth the effort due to the power consumption and area impact of the tables required to leverage an adequate level of reuse. In this paper, we introduce and evaluate a novel way of understanding multimedia floating-point operations based on the fuzzy computation paradigm: Performance and power consumption can be improved at the cost of small precision losses in computation. By exploiting this implicit characteristic of multimedia applications, we propose a new technique called tolerant memoization. This technique expands the capabilities of classic memoization by associating entries with similar inputs to the same output. We evaluate this new technique by measuring the effect of tolerant memoization for floating-point operations in a low-power multimedia processor and discuss the trade-offs between performance and quality of the media outputs. We report energy improvements of 12 percent for a set of key multimedia applications with small LUT of 6 Kbytes, compared to 3 percent obtained using previously proposed techniques.

Index Terms—Low-power design, special-purpose and application-based systems, real-time and embedded systems.

1 Introduction

Multimedia applications are one of the driving forces of computer architecture today, especially in the low-end domain [2], [3], [4]. Hand-held systems keep including better and larger displays/speakers and more media processing power. Current PDAs are able to play MP3 audio files or MPEG-1/AVI video files [5] and commodity hand-held game systems already run 3D engines to display in-game graphics [4]. Furthermore, with the coming increase in bandwidth, third generation digital mobile phones (UMTS) [6] are expected to deal with computational-intensive media protocols such as MPEG-4 [7], where enough raw power is required to decode a wide assortment of media sources (audio, speech, video, and 3D image synthesis).

In order to attain the performance levels required for new applications, it is unlikely that the embedded processor domain may adopt the same evolution that characterized general-purpose processors during the last 15 years. Performance improvements in conventional processors were partly achieved by increasing the instruction level parallelism (increasing the number of instructions fetched and executed per cycle, introducing speculative execution, and providing out-of-order execution capabilities) [8], [9], [10], [11]. Unfortunately, these features may provide diminishing returns from the point of view of performance/power as marginal performance improvements are provided at the cost of complexity and, thus, increased power.

- *C. Álvarez and M. Valero are with the Departament d'Arquitectura de Computadors, Campus Nord, Edifici D6, Jordi Girona, 1-3, 08034 Barcelona, Spain. E-mail: {calvarez, mateo}@ac.upc.es*
- *J. Corbal is with BSSAD, VSSAD, Intel Labs Barcelona, Campus Nord, Edifici Nexus II, Jordi Girona, 29, 08034 Barcelona, Spain. E-mail: jesusx.corbal@intel.com.*

Manuscript received 6 Aug. 2003; revised 17 June 2004; accepted 20 Jan. 2005; published online 16 May 2005.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0120-0803.

A proposed technique for the low-end domain should focus, hence, on reducing the number of instructions (or the cost of the operations) since it is one of the most efficient ways of improving, simultaneously, the execution time and the energy consumption of a given application.

1.1 Instruction Memoization

Among the techniques focused on reducing instructions or operations, instruction/region reuse [12], [3], [13], [14], [15] arises as a very promising technique as it allows the processor to skip the execution of an instruction/group of instructions by simply accessing a memorization table, thus providing the potential to reduce both execution time and power/energy consumption.

In particular, some recent studies [3], [15] have shown that instruction memoization is a potentially powerful technique to avoid computation in long latency instructions such as multiplication and division. Every time such an operation is invoked, its operands and its result are stored in a *Reuse Table*. When there is another instance of the same operation with the same inputs, then the computation can be avoided. This technique not only improves the performance of long latency instructions, but also produces savings in energy consumption.

Floating-point instructions are important for multimedia applications because, although a lot of algorithms use fixed-point computation for multimedia, there is a constantly growing group of programs that do not. Nowadays, the ITU standards [16] are written only in floating-point and many current DSPs incorporate floating-point units [17]. The need for floating-point computation grows mainly due to the growing complexity of the algorithms. It is nearly impossible to find any speech recognition software in fixed-point (due to the wide dynamic range of the data needed to perform the computation) and the same applies to voice synthesis and 3D rendering. Furthermore, floating-point algorithms are generally more efficient than integer ones and applications are beginning to migrate to them (e.g., wavelet video compression).

The major drawback of reuse techniques, applied to FP operations, is the need to implement very large *reuse tables* to achieve acceptable reuse rates due to the wide range of possible operands. This variability produces low hit rates with realistic tables and results in diminishing returns from the point of view of power consumption.

1.2 Fuzzy Computation

JPEG, MPEG2, or MP3 are examples of broadly extended "lossy" compression algorithms. All of them offer the possibility of choosing between output quality and output size: An increase in the compression ratio implies a reduction in the quality of the output data.

Fuzzy computation is a novel way to perform computation that, relying on the inherent tolerance of typical media data, introduce the processing speed factor into the previously mentioned compromise (see Fig. 1). That is, it allows faster and cheaper computation at the cost of some loss in accuracy that human senses could not distinguish [26]. This is possible by exploiting an intrinsic characteristic of multimedia algorithms: tolerance. By tolerance, we mean the robustness of typical media types (such as image pixels or audio samples) to losses in their precision. This property can be used to significantly increase the value locality of floating-point instructions [1] at the cost of modest degradations in the quality of the outputs.

The concept of tolerance is absolutely intrinsic to media applications. In sharp contrast to most other kinds of workload (such as any given SPECint or SPECfp benchmark), media applications exhibit a high tolerance to the accuracy of the outputs because the final destinations are human senses, which are tolerant by definition. The results of 3D, video, image, or audio applications

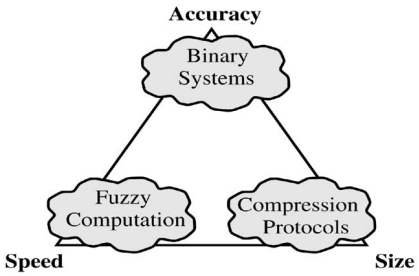

Fig. 1. Basis for the fuzzy computation paradigm.

may present differences that are not visually (or audibly) perceptible and, so, output "errors" are filtered by the human senses, a property that is actually exploited by most of the audio and image encoders as MP3 or JPEG [26].

This paper is organized as follows: Section 2 introduces the methodology used in the paper. Section 3 evaluates the viability of tolerant memoization in the context of low-end systems. It discusses the potential and the trade-offs and proposes a specific implementation. Section 4 is a performance evaluation of this technique and, finally, Section 5 summarizes the paper.

2 Methodology

The focus of this paper is to evaluate an instruction memoization technique for the mobile domain. Our target system would be a future generation hand-held system able to display audio, image, and video that, at the same time, addresses key applications such as 3D rendering and speech recognition. This kind of system could be representative of what PDAs, videogame hand-helds, pocket PCs, and even mobile phones will look like.

Although there are other architectures that are being widely used in the mobile domain (such as the StrongARM, commonly found in PDAs and videogame hand-held systems), we want to adopt as a baseline architecture a more aggressive processor also targeted at low power consumption but with floating-point support. Our final choice was the Hitachi SH4 [18]. The Hitachi SH4 is an embedded processor that is targeted at consumer multimedia applications like home video games (Sega *Dreamcast* [19]) and handheld PCs. The basic core consists of a 2-way superscalar core with in-order issue and three different pipelines: integer, branch, and floating-point. SH4 at 200 MHz has a reported power consumption of only 1.2 Watts.

The Simplescalar [20] and Wattch [21] simulators have been configured to model an SH4-like processor [18] with 0.25 micron technology, 200 MHz, and 1.8 Voltage supply as our basic framework. Table 1 shows the hardware parameters of the model.

Our benchmark suite is composed of four different programs that are representative of four key application domains for future hand-held media systems: audio/music, image, 3D rendering, and speech recognition. `Lame` [22] is an open-source MP3 music encoder. `Epic` is an experimental image compression protocol included in the *mediabench* suite [2]. `Texgen` is a Mesa (Open-GL) application which generates a texture-mapped object, also from the *mediabench* suite. Finally, `Speech Rec` is an experimental speech recognition application from the Digital Signal Department at the Universitat Politecnica de Catalunya [23] , which recognizes numbers from 0 to 9. The programs' characteristics are summarized in Table 2.

3 Memoization Framework

Fig. 2a shows the scheme under study. We have the conventional floating-point functional unit together with an LUT (look-up table). This LUT stores the results of floating-point operations, together with the values of the source operands and one or two bits indicating the type of operation. When a floating-point instruction is going to be executed, the LUT is accessed to determine if there has been a previous instance of that instruction with the same source operands. In that case, the result can be directly obtained from the table. Otherwise, the instruction needs to be executed and, once it is finished, the LUT is updated with the result.

As we are focusing on the low power domain, the tables have been implemented sequentially with the FPALU. In other words, we first consider whether there is a hit or miss in the LUT before dispatching the instruction to the corresponding functional unit. Therefore, a miss in the table will increase the operation latency by one cycle, but a hit will make FPALU not consume any power. Moreover, this implementation has inherently lower complexity, thus being more suitable for the low-end domain.

A typical problem when evaluating the potential of memoization is that, in fact, a relevant percentage of the operations is trivial, that is, the result can be easily deduced before executing the operations (for instance, a multiply operation when one of the

TABLE 1
Configuration of the Base Processor

Processor Core		Memory Hierarchy		Floating Point Latencies (issue latencies)	
Issue	In order	L1 Dcache Size	16 K	Addition (Float)	2 (1)
Physical Registers	32	L1 Dcache Assoc.	2-way	Addition (Double)	4 (1)
Fetch width per cycle	2	L1 Icache Size	16 K	Multiplication (Float)	4 (1)
Decode width per cycle	2	L1 Icache Assoc.	2-way	Multiplication (Double)	8 (6)
Issue width per cycle	2	DTLB Size (full assoc)	32	Division (Float)	12 (12)
Commit width per cycle	2	ITLB Size (full assoc)	32	Division (Double)	24 (24)
FP units	1	L2 Cache	none		
Integer units	1				
Branch prediction	Not Taken				

TABLE 2
Benchmarks

Program	Description	Data set	Characteristics
Epic	Dyadic Wavelet-based Image compression	pamela.pgm	256x256 grey-scale 8 bitmap showing a girl
Texgen	MESA 3D: an open-GL 3D-graphics API	teapot.ppm	500x500 texture mapped version of Utah teapot
SpeechRec	Speech Recognition	numbers.wav	audio sample with a sequence of 1000 numbers
Lame	MP3 encoder	fugue.wav	Star Wars soundtrack fragment, 10 seconds

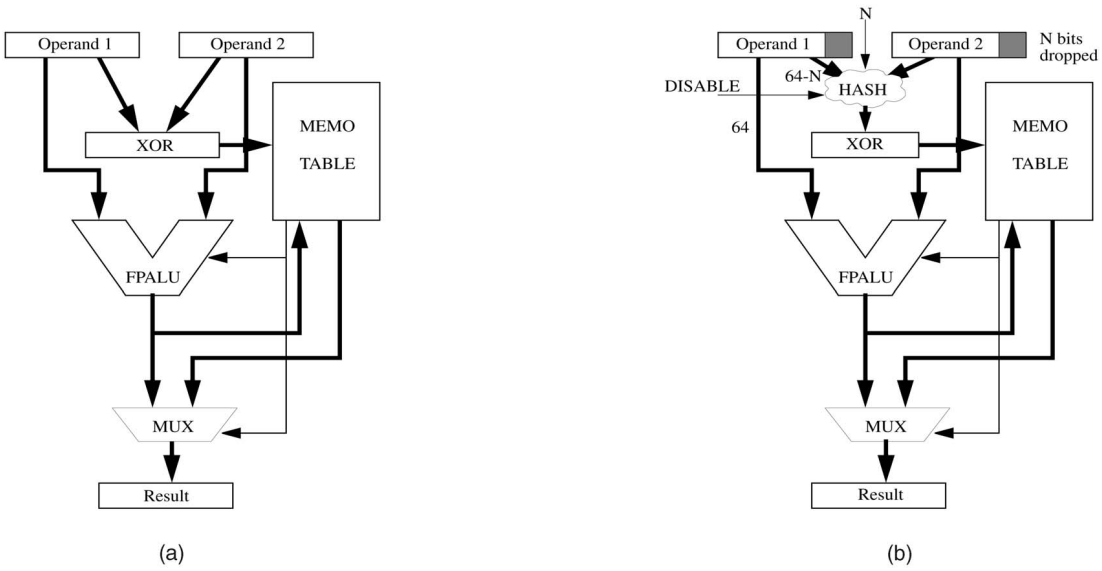

Fig. 2. Hardware configuration of a sequential LUT for (a) classical memoization and (b) tolerant memoization.

TABLE 3
LUT Characteristics

LUT Characteristics	
Number of independent LUTs	1
Memoized operations	multiplication / division
LUT type	2-way set associative
Hashing scheme	XOR of less significant bits of operands mantissas
LUT Sizes	
Low-cost	6 KBytes
Aggressive	24 KBytes

source operands is 0). In order to allow a fair evaluation, we have also implemented the method proposed by Richardson [24] to catch the trivial operations so that they will not pollute the table. Power consumed in the LUT has been modeled with Cacti [25] and the effect of the LUT delayed update in case of a miss (due to the intrinsic latency of the operation) has been modeled. Leakage power has also been measured in our simulations.

We have made some preliminary evaluations to determine which kinds of FP operations to memoize, which is the most cost-effective configuration table (direct-mapped, n-way set associative), the hashing mechanism of the source operands to index the table, and how many independent LUTs it is worth implementing. An in-depth discussion of these results is out of the scope of this paper, but the final choices are summarized in Table 3. As can be observed, we have only implemented one reuse table and memoized only multiplications and divisions, as we have observed that memoizing addition and subtraction is not cost-effective because of the low hit rate and the low latency of these two operations.

3.1 Tolerant Memoization Implementation

Fig. 2b shows the scheme we used to implement fuzzy computation. We call it fuzzy or tolerant memoization. Tolerant memoization is implemented by taking off the N least significant bits from the mantissa of every operand before it is used to access the table. So, not only will identical inputs hit, but also similar ones will hit. The number of dropped bits, N, is what we call the tolerance level.

As happens with other lossy methods, a practical limitation exists for fuzzy memoization as very high losses can produce diminishing results. In the JPEG algorithm, for example, a compression ratio of near 25 produces quite good quality images, but an increase in the compression ratio of up to 30 quickly yields very poor quality images. A similar behavior would be expected in fuzzy memoization. To measure the error introduced by our system, we have used the Signal to Noise Ratio (SNR), which is broadly used in the signal processing environment. The SNR is defined as:

$$SNR = 10 \log \frac{S}{N},$$

where S is the Signal power and N is the Noise power. An SNR above 30 dB means that the error is nearly indistinguishable by the human senses [26].

Fig. 3 shows the reuse table hit rates compared with the associated SNR measured for every benchmark as a function of tolerance level. Results were obtained with a 1,024 entries table (2-way set associative).

Results from Fig. 3 show two different behaviors of hit rate and SNR, depending on the tolerance range. From a tolerance level of 0 (or, in other words, classical memoization) to 30, we observe that there are no distinguishable changes in either the hit rate or the SNR (i.e., we introduce no error in the output). In this region, which we call the "redundant region," tolerant memoization takes advantage of redundancy in the representation of the data. This is a secondary effect of performing tolerant memoization and, although it does not give us benefits (there is no use of tolerance here), it allows us to use thinner (and thus smaller) tables. As a result of this, we could wrongly think about achieving the same or better result by redefining all double data to single data. This is not possible because:

First of all, some algorithms (such as Lame or Texgen) simply do not work with simple-precision mantissas. Doing tolerant

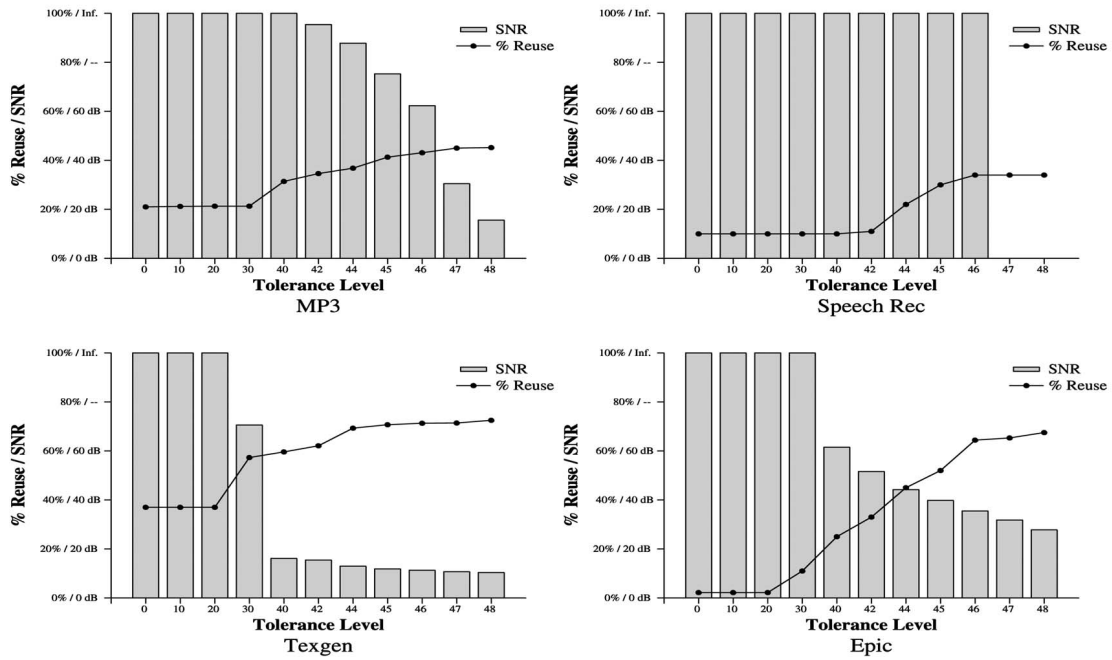

Fig. 3. Error versus hit rate in tolerant memoization.

memoization, some results are really computed in high precision (every operation in which a miss is passed to the FPALU) and, so, the output is very close to the original. Introducing shorter FP multipliers will result in every operation adding some error and this leads to unacceptable quality or even program crashes. Second, we wanted to study the algorithms as they are because, as time to market decreases, specific optimizations are less affordable in a realistic environment. Furthermore, as can be seen in Fig. 3, dismissing 29 bits of double-precision values (and thus making them some kind of "double-range simple-precision" values) does not increase the reuse. So, if it was possible to use simple-precision floating point, the results obtained with the original algorithms would not change significantly.

From a tolerance level of 31 to $46-48$ (depending on the benchmark), we observe the "tolerant region." In this area, we are effectively doing fuzzy computation, that is, the system is reusing different input instances and giving them the same output. Therefore, both the reuse rate and the error are increased, but we observe that, at the beginning of this range, the reuse hit rate grows faster than the degradation of the SNR. Then, the hit rate saturates while the SNR starts dropping dramatically. This behavior allows our method to provide great energy savings without introducing noticeable errors in the output data.

4 Evaluation

Fig. 4 summarizes the results provided by fuzzy memoization. It shows the resulting savings in energy, execution time, and energy-delay product for five different configurations: a *trivial operations* mechanism where only the execution of trivial operations is skipped, a *6 KB table* with classical memoization (i.e., no tolerance), a *24 KB table* also with classical memoization and two LUTs (also of 6 KB and 24 KB) with tolerance. The tolerance level for every benchmark was chosen to keep the SNR above 30 dB.

Results from Fig. 4 show that fuzzy memoization provides very good energy savings compared with the rest of the techniques: More than 15 percent of the energy consumption is saved for three of the benchmarks and 6 percent for the other one. This is especially significant for `Epic`, where the conventional memoization technique is actually counterproductive. Furthermore, some performance speed-up is achieved as the average latency of the floating-point instructions is reduced. Reductions on the execution time of up to 6 percent are reported for the 6 KB table. This translates into significant gains in the energy-delay product (up to 25 percent). We should note that reductions in the execution time can be converted into additional energy savings as the clock frequency of the processor can be reduced. Since power is proportional to the square of the working voltage, great gains may come as a result of this approach.

Another very interesting observation from the results of Fig. 4 is that the 6 KB fuzzy memoization table achieves gains that are already very close to those of the more costly 24 KB alternative. Furthermore, the results of a table with an oracle access method have been measured. In this table, only the input pairs that are going to hit are used to access the table. The input pairs that miss are directly bypassed to the FPALU and, so, no penalty is introduced. This oracle table gains less than an additional 0.5 percent for fuzzy memoization. Classical memoization is more than 1 percent away from the oracle.

Fig. 5 shows the energy savings in the FP unit only. In this figure, it can be seen that, while our proposed scheme presents gains up to 60 percent in energy (35 percent on average), the classical memoization scheme only can save 9 percent on average. All the results presented include the power consumption of the LUT and the leakage. These results lead us to think about designing a simple FP ALU that, with the help of tolerant memoization, performs as fast as the aggressive one studied in this paper while consuming less power.

Finally, we perform a comparison between the Energy*Delay savings achieved in serial and parallel configurations. In the serial configuration, if there is a hit in the table, the FPU is not used and, so, if there is a miss, an extra latency cycle is used. In the parallel configuration, the LUT and the FPU are accessed in the same cycle and, so, although there is a hit, some power is expended in the FPUs. On the other hand, a miss results in no time penalties. Results are shown in Fig. 6 and show a compromise effect. When only a low hit rate is achieved (classical reuse and speech), parallel configuration works better as it saves some energy but does not

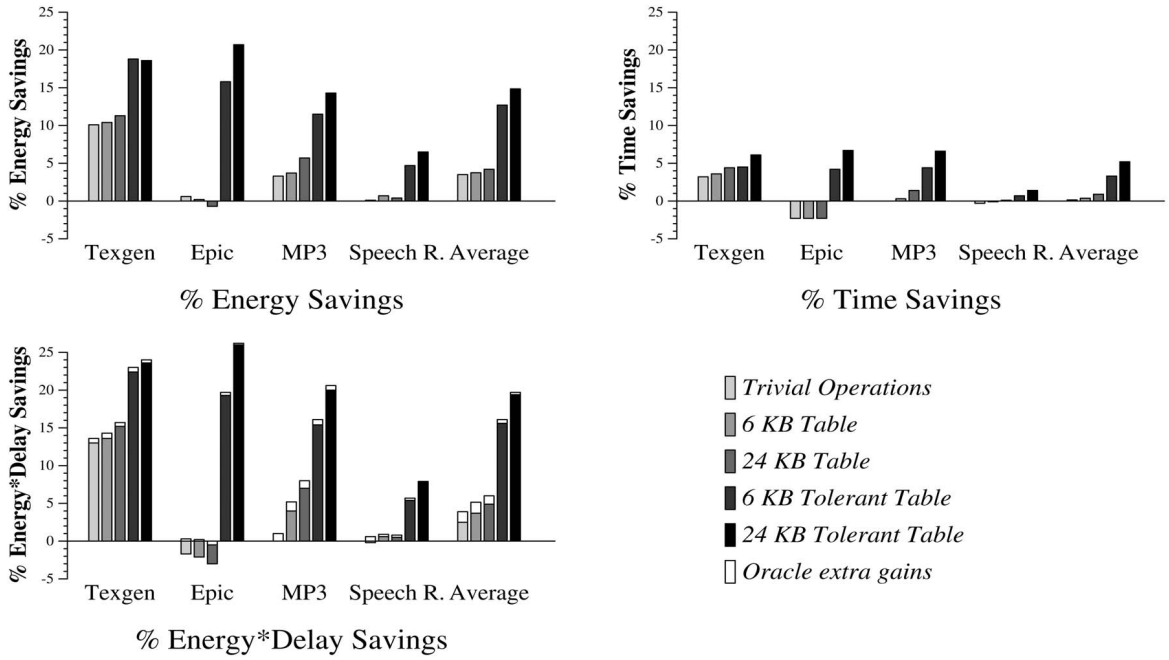

Fig. 4. Results of fuzzy FP memoization.

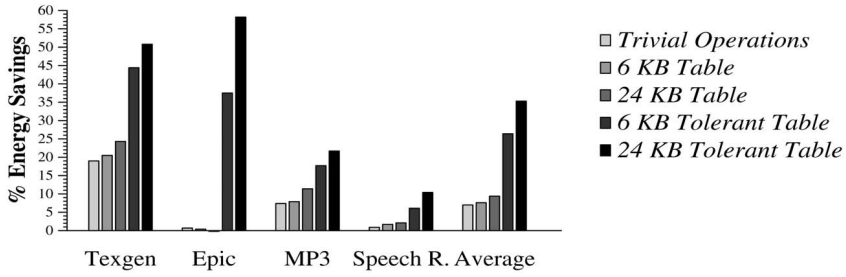

Fig. 5. Energy savings in the FP unit.

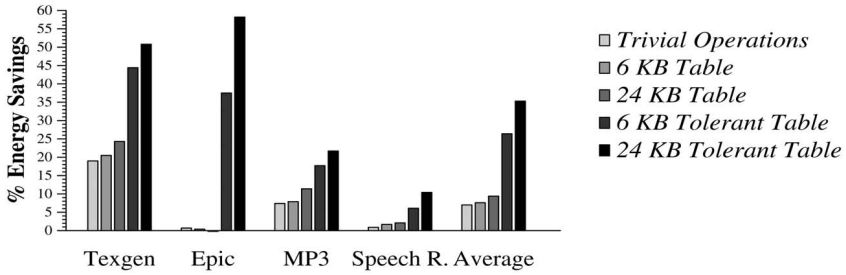

Fig. 6. Parallel versus serial configurations.

increase the operation latency. When the hit rate grows, serial configuration arises as the best solution because it only infrequently uses one more cycle, but often saves the entire energy of the FPU, therefore, serial configuration is the best choice for tolerant reuse.

5 Summary

In this paper, we have performed an extensive evaluation of the potential of floating-point memoization in the context of hand-held oriented multimedia processors. In such processors, energy consumption is the major constraint. We have shown that conventional memoization may provide diminishing returns in such architectures due to the intrinsic cost of the tables required to provide significant reuse rates. Furthermore, we have shown that simple techniques to detect trivial operations provide very similar gains with considerably less cost.

We propose applying the concept of fuzzy computation to the memoization technique. By considering similar instances as acceptable candidates to reuse, we take advantage of the trade-off between output quality and energy consumption. The mechanism proposed is simple and could be implemented without either compiler or ISA support.

Fuzzy computation significantly improves the performance of FP operations. With tolerant memoization and realistic table sizes, the reuse hit rate is raised and, as a result, considerable power and time savings are achieved (up to a 25 percent improvement in the energy-delay product for some of the benchmarks) at the cost of introducing some errors in the output data that are negligible in the context of hand-held devices.

ACKNOWLEDGMENTS

This work has been supported by the Ministry of Education of Spain under contract TIN-2004-07739-C02-01, the HiPEAC European Network of Excellence, and CEPBA. The authors would like to thank Jose A.R. Fonollosa, Esther Salamí, and Josh Fisher, who really helped them in improving the paper.

REFERENCES

[1] D. Goldberg, "What Every Computer Scientist Should Know about Floating-Point Arithmetic," *ACM Computing Surveys*, vol. 23, no. 1, pp. 5-48, 1991.
[2] C. Lee, M. Potkonjak, and W.H. Magione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communication Systems," *Proc. 30th Ann. ACM/IEEE Int'l Symp. Microarchitecture (MICRO 97)*, Dec. 1997.
[3] D. Citron, D. Feitelson, and L. Rudolph, "Accelerating Multi-Media Processing by Implementing Memoing in Multiplication and Division Units," *Proc. Eighth Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS '95)*, 1995.
[4] Graphic State, "GBA 3D Engine," http://www.graphic-state.com/press/n_engines.htm, 2002.
[5] Casio, "Cassiopea," http://www.casio.com/personalpcs/product.cfm?section=19&product=3553&display=15&cid=3949, 2001.
[6] UMTS Forum, "UMTS Forum," http://www.umts-forum.org/information.html, 2001.
[7] R. Koenen, "MPEG-4, Multimedia for Our Time," *IEEE Spectrum*, pp. 26-34, Feb. 1999.
[8] M.H. Lipasti and J.P. Shen, "Exceeding the Dataflow Limit," *Proc. 29th Ann. ACM/IEEE Int'l Symp. Microarchitecture (MICRO '96)*, pp 226-237, Dec. 1996.
[9] J.G. Steffan and T.C. Mowry, "The Potential for Using Thread-Level Data Speculation to Facilitate Automatic Parallelization," *Proc. Fourth Int'l Symp. High-Performance Computer Architecture (HPCA '98)*, Feb. 1998.
[10] H. Akkary and M. Driscoll, "A Dynamic Multithreaded Processor," *Proc. 31st Ann. ACM/IEEE Int'l Symp. Microarchitecture (MICRO '98)*, 1998.
[11] A. Roth and G.S. Sohi, "Speculative Data-Driven Multithreading," *Proc. Seventh Int'l Symp. High-Performance Computer Architecture (HPCA '01)*, 2001.
[12] A. Sodani and G.S. Sohi, "Dynamic Instruction Reuse," *Proc. 24th Ann. Int'l Symp. Computer Architecture (ISCA '97)*, 1997.
[13] D.A. Connors and W.M. Hwu, "Compiler-Directed Dynamic Computation Reuse: Rationale and Initial Results," *Proc. 32nd Ann. ACM/IEEE Int'l Symp. Microarchitecture (MICRO '99)*, 1999.
[14] S.S. Sastry, R. Bodik, and J.E. Smith, "Characterizing Coarse-Grained Reuse of Computation," *Proc. Third ACM Workshop Feedback-Directed and Dynamic Optmization*, 2000.
[15] M. Azam, P. Franzon, and W. Liu, "Low Power Data Processing by Elimination of Redundant Computations," *Proc. 1997 Int'l Symp. Low Power Electronics and Design*, pp. 259-264, 1997.
[16] Int'l Telecomm. Union, "Home Page," http://www.itu.int/, 2004.
[17] Texas Instruments, "DSP Developers' Village," http://dspvillage.ti.com/docs/dspvillagehome.jhtml, 2004.
[18] F. Arakawa, O. Nishii, K. Uchiyama, and N. Nakagawa, "SH4 Risc Multimedia Processor," *IEEE Micro*, Mar./Apr. 1998.
[19] S. Hagiware and I. Oliver, "Sega Dreamcast: Creating a Unified Entertainment World," *IEEE Micro*, Nov./Dec. 1999.
[20] D. Burger and T.M. Austin, "The SimpleScalar Tool Set, Version 2.0," Technical Report #1342, Computer Science Dept., Univ. of Wisonsin-Madison, 1997.
[21] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," *Proc. 27th Ann. Int'l Symp. Computer Architecture (ISCA '00)*, 2000.
[22] The LAME Project, "Home Page," http://www.mp3dev.org/mp3/, 2004.
[23] Dept. of Signal Theory and Comm., "Speech Processing Group," http://gps-tsc.upc.es/veu/, Universitat Politecnica de Catalunya, 2004.
[24] S.E. Richardson, "Exploiting Trivial and Redundant Computation," *Proc. 11th IEEE Symp. Computer Arithmetic*, 1993.
[25] P. Shivakumar and N.P. Jouppi, "Cacti 3.0: An Integrated Cache Timing, Power and Area Model," http://research.compaq.com/wrl/people/jouppi/CACTI.html, technical report, Compaq Computer Corp., 2001.
[26] E.B. Goldstein, *Sensation and Perception*, sixth ed. Univ. of Pittsburgh, 2002.