REDUCTION OF REGISTER FILE DELAY DUE TO PROCESS VARIABILITY IN VLIW EMBEDDED PROCESSORS

Praveen Raghavan[1], *José L. Ayala*[2], *David Atienza*[3,4]
Francky Catthoor[2], *Giovanni De Micheli*[4], *Marisa López-Vallejo*[1]
[1]DIE/UPM, Spain. [2] IMEC vzw and KULeuven, Leuven, Belgium.
[3]DACYA/UCM, Spain. [4] LSI/EPFL, Switzerland.

ABSTRACT

Process variation in future technologies can cause severe performance degradation since different parts of the shared Register File (RF) in VLIW processors may operate at various speeds. In this paper we present a complete approach that handles speed variability of the RF proposing different compile-time and run-time design alternatives. The first alternative extends current RF architectures and uses a compile-time variability-aware register assignment algorithm. The second alternative presents a fully-adjustable pure run-time approach, which overcomes the variability loss as well, but at the extra cost of cycles and area. However, the savings achieved and the run-time management of the register delay variations without any support from the user, show a very promising application field. Our results in embedded system benchmarks show that variability can be tackled without significant performance penalty, and trade-offs between performance and area are possible thanks to the whole design spectrum provided by the two presented alternatives.

I. INTRODUCTION

New multimedia consumer applications require high performance embedded platforms due to their intensive processing requirements. In this area of embedded systems, *Very Large Instruction Word (VLIW)* processors provide a promising solution to achieve suitable performance-power trade-offs. However, transistor scaling in future technology nodes is accompanied by an increase of variability in process technology. Two types of variations exist: functional variation and parametric variation. Functional variation leads to loss in component functionality, while parametric variation leads to timing issues in the working component with respect to its originally designed speed [1], [2], [3].

In this paper we present several solutions to handle parametric variation in shared register files of embedded VLIW processors, which provide also trade-offs of pre-characterization (design time) effort and dynamic (run-time) overhead for self-tuning. The first solution (Section III) is a *hardware/software (HW/SW)* approach that relies on limited HW extensions in the *Register File (RF)*, and a modified register assignment phase at compiler level to prevent using marked slow registers. In the latter two approaches (Section IV), we propose further extensions of the RF HW architecture to create completely run-time schemes against dynamic variation in the RF. These run-time approaches do not require additional compilations for every target device. Our results with several embedded multimedia benchmarks (Section V) show that these approaches can overcome in most of the cases the expected variability in future technologies without performance penalties compared to the ideal case (i.e. without speed variations), and provide several design alternatives of shared RFs against parametric variation.

II. RELATED WORK

Effects of intra-die stochastic variability on the performance of CMOS processor blocks, as ALUs, have been recently studied [4], [5]. For on-chip memories, the focus has been in functional yield and reliability issues, where the use of SRAM cell stability (e.g. signal to noise margin) and design rules to compensate for performance issues were proposed [2], [6]. However, to the best of our knowledge there has been no work which addresses variability problems in RFs at compiler level.

Regarding reliability in future technologies, *Single-Error-Correcting-Multiple-Error-Detecting (SEC-MED)* codes are already integrated in on-chip memories and some processors [7], [3] to decrease the probability of error. Also, [8] shows that program behavior patterns can be used to generate custom-error correction mechanisms for memory portions. Nevertheless, these techniques are not suitable for RFs since they impose significant area and energy consumption overheads.

In addition, system-level fault tolerant management mechanisms have been proposed for soft errors [3].Then, [9] uses redundant components and self-checking for embedded systems to extend processor lifetime in case of process variation. [10] presents self-repairing mechanisms via pre-existing processors. These works are complementary to our approach since they deal with functional variation in on-chip memories and processors, while the goal of our work considers parametric variation in the RF.

III. VARIABILITY-AWARE HW/SW COMPILATION

The baseline VLIW architecture used in our work is provided by the CRISP framework [11] which is a cycle-accurate extension of the Trimaran framework [12] including memories. It consists of a cycle-accurate simulator of a selectable number of *Functional Units (FUs)* and *RFs (RFs)* that model the desired *Digital Signal Processor (DSP)*. It includes a retargetable compiler based on the Trimaran framework [12]. In

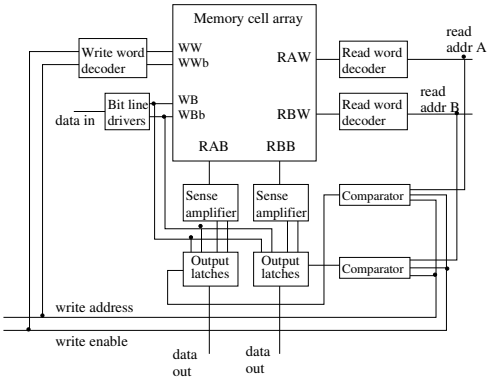

Fig. 1. RF block diagram

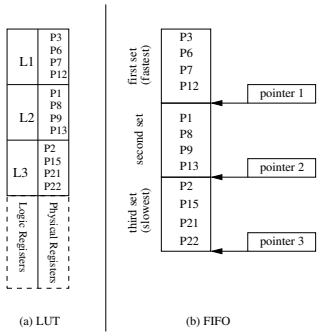

Fig. 2. Implementation of (a) LUT HW module (b) FIFO HW module

the following, the extensions applied to the CRISP framework are described.

A. HW Extensions

In a typical configuration, the RF is an array of N words by M bits. Any of the N words can be simultaneously accessed by any of the ports of the RF (read or write). Figure 1 shows that the RF contains seven distinct types of functional blocks [13]. In this design, the memory cell array stores the data bits, and is arranged in a grid of N rows by M columns of memory cells. When any of the ports accesses it, the read/write operation is performed simultaneously on every memory cell in the selected row.

Technology variations during device manufacturing can create delays in any component shown in Figure 1. In our approach, the write/read delay of each RF register with respect to the original specification is stored in the HW architecture using some extra bits per register (in our notation, label bits). In case a 1-bit label is used, a '0' would indicate the register satisfies the timing requirement and '1' would assume that it does not (therefore we would have to assume a worst case, i.e., 4 cycles) delay. In case a 2-bit label is used, then a more fine-grained labeling would be done, i.e, '00' would mean satisfying a 1-cycle delay, '01' would imply 2-cycle latency to operate and so on. This classification is done in an initial characterization phase after production and requires the area overhead of extra timing circuitry and storage of the label bits. Since the size of this circuitry is very small, the timing circuitry overhead will be not noticeable and can be neglected. Furthermore, this circuitry is activated only once after fabrication time; thus, its dynamic and leakage power overhead can be neglected because it can be Vdd gated during the normal processor execution. The storage overhead for the label bits has been calculated using a TSMC *90nm* technology and our results indicate that it is very limited with respect to the area of the baseline RF (Section V). We suppose no process variation on this extra bits due to two assumptions: (1) the extra bits required account for a very limited area; hence, the probability of any variation in their storage is very low. (2) this storage can be over-designed, so that it is more tolerant to process variation (without a significant overhead in the overall area of the RF). Finally, the number of sets used to classify the registers enables trade-offs between accuracy in the classification and the area of extra label bits (see Section V for examples in real benchmarks).

B. Compiler Support for Variability

The Trimaran compiler of the CRISP framework has been modified to include manage the described HW architecture, including label bits. In prticular, the register allocation phase of the compiler has been rewritten to incorporate this extra knowledge.

Traditionally, register assignment algorithms choose registers from the whole set of free registers without any constraint. In the case of Trimaran, as many other compilers, it retrieves the first register from a *First Input First Output (FIFO)* list of free registers. The order of registers inside the list is not representative and no restriction on selecting the registers exists, so the assigned registers in the original register assignment can easily belong to different *delay sets*. Our register assignment policy modifies this original register assignment by first selecting the registers labeled as faster registers. The new compiler initializes internally FIFO lists for each delay set in the RF by a first reading phase of the stored label bits, and performs a *variability-aware* assignment giving preference to registers of the fast sets for the loops of multimedia applications. These applications are usually loop-dominated [14] and the execution of loops seriously affect the overall processor performance.

This compiler phase has no extra overhead in terms of area, performance or power consumption of in-order processors. However, it requires the recompilation of the sources for every target processor since the profile of access delays varies between each instance of the target architecture. This limitation makes difficult its extended use. Therefore, we have also designed two additional run-time HW techniques to solve this previous limitation at the potential cost of reducing the overall potential savings of the compile-time approach. The compiler approach can be considered as the baseline proposal wich the HW approach is compared to.

IV. HW RUN-TIME APPROACHES

The initial register assignment performed by the compiler can be redefined at run-time to assign first the registers with a reduced access delay. For that purpose, we have first defined a technique using a *Look-Up Table (LUT)*-based HW. The LUT included in the system translates the logic registers (software

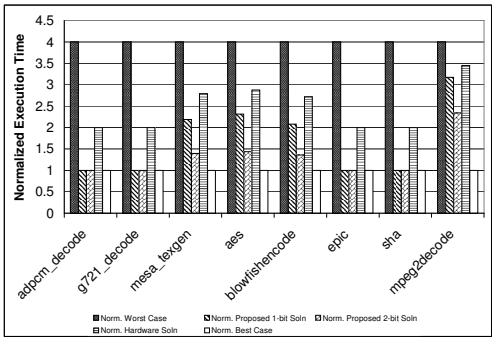

Fig. 3. Normalized performance with 50% stochastic faulty registers with 1-bit and 2-bit classification and HW based solution

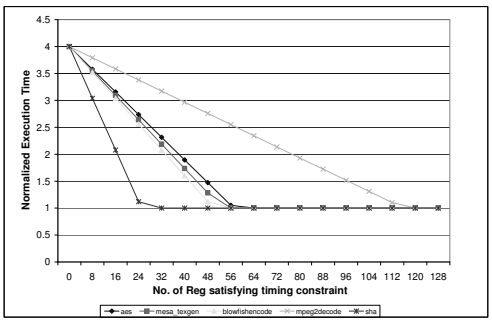

Fig. 4. Evolution of performance due to variability with compiler based 1-bit classification

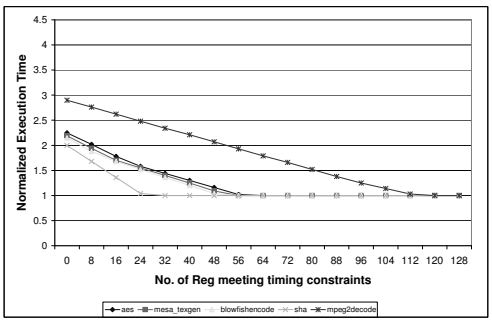

Fig. 5. Evolution of performance due to variability with compiler based 2-bit classification

assigned) into the physical registers (HW assigned) trying to minimize the access delay. The granularity of the assignment in the current version we have developed can be chosen to be low (just 2 sets, i.e., slow and fast registers) or high (4 sets), see Figure 2(a). Since the compiler assigns the logic registers in an increasing manner (i.e., 1, 2, etc), the smallest indexes will be the most frequently used, and they will be translated into fast registers. Then, the processor, which can have the profile of register speeds stored in the BIOS of the system, accesses the LUT to write the translation information during the boot of the system. Once the translation information is communicated to the LUT, the table works as a normal translation device at run-time.

Also, since the LUT approach can expose a negative impact in access time to the memory device due to the overhead of finding a free register to be assigned, we have defined a simplified version of this HW technique, which uses a *First-Input First-Output (FIFO)* list of free registers. In this case, the complete set of indexes of free registers is stored in a FIFO mode in an ordered way according to the order given by the access delay (i.e. according to their reduced access delay). As in the LUT-based technique, the profile information of the register delays can be stored in the BIOS of the system and loaded into the FIFO at boot. Then, the module acts as a FIFO whenever is read (if all the fastest registers are assigned, then the slower ones are taken), but some extra pointers are needed to identify the end of every set where the freed registers should be taken back (see Figure 2(b)). Thus, the granularity level is determined by the number of pointer registers included, and trade-offs between area and granularity exist. Finally, LUT and FIFO approaches introduce an extra access time that could represent a performance penalty if the savings achieved in the register translation do not overcome this delay (see Section V for more details).

V. CASE STUDIES AND EXPERIMENTAL RESULTS

The CRISP framework was used to simulate a 32-bit, 4-issue VLIW processor, including 12 ports (8 read and 4 write) 128 entries deep RF. We simulated various MediaBench [14] benchmarks on this architecture. The area of the original RF was $3.04 \times 10^5 \mu m^2$ using the TSMC 90nm standard cell synthesis process. We assumed that process variations of access time delays in the whole pool of registers follows an homogeneous distribution, as suggested by [1]. The RF was annotated to inject such a distribution. Therefore, when the benchmarks compilation process has finished in CRISP, the percentage of utilization of the RF can be obtained. Then, after the execution of the benchmarks, CRISP provides us the number of registers that meet the performance constraints and overall performance results with the injected timing variations (based on the register allocation).

A. Experiment 1

In a first set of simulations we have studied the normalized execution time when 50% of the registers do not meet the timing constraint of 1 cycle of operation (at 300 MHz), i.e. they do not respond in less than 3.33 ns. The execution time has been normalized to this best case, where all registers respond in one cycle. We have assumed that 32 out of 128 registers meet the performance constraints (read/written in one cycle), while 32 registers require 2 cycles, 32 require 3 cycles, and the last 32 need 4 cycles. No further process variation is considered. Any register that requires more than 4 cycles is marked as unusable.

Figure 3 shows the normalized performance results in terms of average number of clock cycles to access the RF using one/two bits for the label bits, and for the worst case simulation (i.e. all registers belong to the slowest set with 4 cycles of latency) and the best case simulations. These results indicate that our compiler-based approach achieves the optimal point of the best case without variability in 4 of the benchmarks, namely, 57% better than the worst case for the other benchmarks on average with the 1-bit labeling and 67% better in the case of 2-bit labeling.

Both types of HW-based solutions also provide the same performance improvement, and only one bar is shown for both in Figure 3 (*38%* faster than the worst process corner). Note that an extra cycle is needed for every RF access due to the overhead of the FIFO or the mapping table. Hence, the HW based technique cannot perform as well as the compiler-based

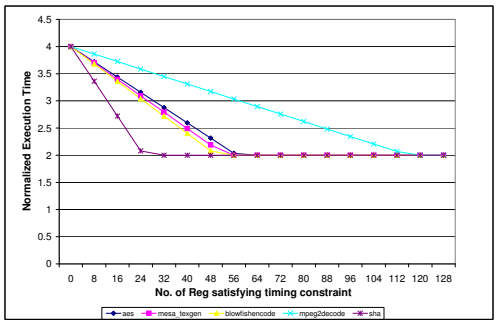

Fig. 6. Evolution of performance due to variability with HW based solution (both FIFO based and mapping table based)

Technique	Area (in μm^2)	Power (in μW)	Avg. Perf. (w.r.t W.C.)	Compile Time
Compiler -based (2-bit)	1.16×10^3	0	67%	Compile needed for every chip
Mapping Table	5.47×10^3	54.67	38%	Once
FIFO-based	2.54×10^3	29.19	38%	Once

TABLE I
COMPARISONS OF RELIABILITY-AWARE RF DESIGN TECHNIQUES

technique for these in-order processors. The performance of the HW-based technique is at best two times worse than the ideal (best) case, as a translation is always needed[1].

Also, the results of Figure 3 show that large benchmarks with strong register pressure (e.g. *aes* and *blowfishencode*) always need to use registers that do not meet the timing constraints. Therefore, the solutions using two bits for the label bits (i.e. classifying the registers into four delay sets) achieve better results than using only one label bit. The only benchmark that does not benefit to the same extent with any of these configurations is *mpeg2dec*, since it demands a large amount of registers during the whole execution. Therefore, for this benchmark we have tested the potential benefits of increasing the number of label bits. Our results indicate that with 5 or more bits we can improve the case of 2 bits, reaching its maximum with 7 bits (only 25% worse than the best case), which outlines the existence of trade-offs between area and performance even with tight system requirements.

B. Experiment 2

In a second set of experiments we have evaluated possible trade-offs between area overhead of label bits and achievable performance if the number of registers that fail the timing constraints varies (Figure 4 and 5). Our results indicate that the optimality in the number of label bits (1-bit or 2 bits) is largely determined by the variability in the RF and the desired performance. Also, another observed trend is the steeper degradation of execution time in the case of 1-bit label registers compared to the two-bit solution. This effect occurs due to the more fine-grained exploitation of the register delay with 2 bits, similarly as the performance effect explained before for the larger multimedia benchmarks. Moreover, these results indicate that using 2-bit for the label bits already achieves the best case performance bound in almost all tested multimedia applications, assuming a variability of up to 39% (i.e. 50 registers responding in 1 cycle). Figure 6 shows a similar plot for HW-based solutions. This figure illustrates that all curves saturate at a normalized execution time of 2. The trend in HW-based approaches is similar to the compiler-based approach. It was seen that when more than 50% of the registers satisfy the register timing requirement, most benchmarks give the required performance. Finally, the comparisons of HW- and compiler-based techniques are summarized in Table I. It shows that the area and power overhead of the mapping table and FIFO-based approaches is higher than in the compiler technique, but removes the problem of recompiling for each instance of the final device.

[1]It is assumed no process variation in the FIFO or the mapping table. This can be ensured by over-designing (i.e. worst-case design) this HW, whereas the RF can be designed at an average-case design.

VI. CONCLUSIONS

In this paper we have presented a HW/SW compile-time approach and two HW-based run-time design alternatives to handle speed variability of the RF. We have analyzed the advantages and disadvantages of these methods and their applicability to real-life applications. Our results indicate that the proposed approaches almost completely avoid performance penalties due to variability in VLIW processors. Moreover, these results have outlined that trade-offs between performance and area overhead are possible according to system requirements.

VII. ACKNOWLEDGEMENTS

This work is partially supported by the Spanish Government Research Grants TIN2005-05619 and TEC2006-00739/MIC.

VIII. REFERENCES

[1] Hua Wang, et al. "Systematic analysis of energy and delay impact of very deep submicron process variability effects in embedded sram modules," in *Proc of DATE*, 2005, pp. 914–919.
[2] A. J. Bhavnagarwala, et al. "The impact of intrinsic device fluctuations on cmos sram cell stability," *IEEE J. Solid-State Circuits*, vol. 36, no. 2, pp. 18–31, 2001.
[3] V. Agarwal, et al. "The effect of technology scaling on microarchitectural structures," Tech. Rep., TR2000-02, UTAustin, 2002.
[4] T. W. Chen et al. "A low cost individual-well adaptive body bias scheme for leakage power reduction and performance enhancement in the presence of intra-die variations.," in *Proc of DATE*, 2004, pp. 240–245.
[5] C. Visweswariah, et al. "First-order incremental block-based statistical timing analysis," in *Proc of DAC*, 2004, pp. 331–336.
[6] R. Heald, "Managing variability in SRAM designs," in *In Proc of ISSCC uP Forum*, February 2004.
[7] M. Blaum, et al., "The reliability of single-error protected computer memories," *IEEE Trans. Comput.*, vol. 37, no. 1, pp. 114–119, 1988.
[8] N. S. Bowen et al., "The effect of program behavior on fault observability," *IEEE Trans. Comput.*, vol. 45, no. 8, pp. 868–880, 1996.
[9] P. Shivakumar, et al. "Exploiting microarchitectural redundancy for defect tolerance," in *Proc of ICCD*, p. 481-487, 2003.
[10] C-L Su, et al., "A processor-based built-in self-repair design for embedded memories," in *Proc. of ATS*, 2003.
[11] P Op de Beeck, et al, "Crisp: A template for reconfigurable instruction set processors.," in *Proc of FPL*, pp. 296–305, 2001.
[12] Trimaran, "Trimaran: An infrastructure for research in instruction-level parallelism," 1999, http://www.trimaran.org.
[13] S A. Steidl, *A 32-Word by 32-Bit Three-Port Bipolar Register File Implemented Using a SiGe HBT BiCMOS Technology*, Ph.D. thesis, Rensselaer Polytechnic Institute, 2001.
[14] C. Lee, et al., "Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems," in *Proc of MICRO*, pp. 330–335, 1997.