A Cost Effective Spatial Redundancy with Data-Path Partitioning

Matsusaka, Sigeharu
Dept. of Elec. and Computer Science, Fukuoka University

Inoue, Koji
Dept. of Informatics, Kyushu University

https://doi.org/10.15017/6233

出版情報：Proc. of the International Conference on Information Technology and Applications, pp.51-56, 2005-07. International Conference on Information Technology and Applications
バージョン：
権利関係：

A Cost Effective Spatial Redundancy with Data-Path Partitioning

Shigeharu Matsusaka† Koji Inoue‡§
†*Dept. of Elec. and Computer Science, Fukuoka University*
8-19-1 Nanakuma, Jonan-ku Fukuoka 814-0180 Japan
‡*Dept. of Informatics, Kyushu University*
6-1 Kasuga-Koen, Kasuga Fukuoka 816-8580 Japan
§*PRESTO, Japan Science and Technology Agency,*
4-1-8 Honcho Kawaguchi, Saitama 332-0012 Japan

†*matsusaka@v.tl.fukuoka-u.ac.jp* ‡§*inoue@i.kyushu-u.ac.jp*

Abstract

In order to maintain the high reliability of a computer system, it is necessary to detect the failure leading to a fault. In general, fault can be detected by exploiting time redundancy or spatial redundancy. However, it negatively affects on either hardware cost or processor performance. To solve the cost-performance issue, in this paper, we propose a concept of cost-effective approach to achieve spatial redundancy for dependable processors. In addition, we perform a primly evaluation for the impact of our method on processor performance.

1. Introduction

In recent years, dependability of computer systems is one of the most important design constraints. Especially, critical applications, such as online transaction processing, an artificial satellite, medical treatment, and traffic control require much higher reliability. Furthermore, in mobile computing environment, computing devices will deal with much important data, for instance electric money and private information. Therefore, such mobile devices must be reliable regardless of its execution environment.

In order to maintain higher reliability of computer systems, it is required to detect a fault which causes a failure. The fault can be classified into two types: permanent faults and temporal faults. The former affects in a long period of time, while the later appears only in very short time. The permanent fault is caused by an internal factor of systems, such as destruction of semiconductor junction, short circuit, and disconnection. On the other hand, the temporary fault, or transient fault, is forced by an external factor of the system, such as unexpected temperature, vibration, alpha particle, cosmic ray, and so on.

To detect the fault at run-time, we need to exploit temporal or spatial redundancy. The temporal redundancy can be achieved by executing the same program several times on the same processor, and comparing the results generated by each execution. There are two main drawbacks for this approach; degrading the performance and missing the permanent faults. On the other hand, the spatial redundancy can be realized by using several processors and executing simultaneously the target program on each processor. This approach does not have the negative effects of the temporal approach. However, we need to waste large amount of transistor budget.

In this paper, we propose the concept of a cost-effective approach to achieve spatial redundancy for dependable processors. In addition, we perform a primly evaluation for the impact of our method on processor performance. Our approach attempts to exploit effectively un-used, or redundant, hardware resources inherent in the processor. Particularly we focus on the data-path. In our scheme, the data-path of processor is physically or virtually (depends on implementation) partitioned into several segments as SIMD computation. For example, 32-bit data-path can be partitioned into 2 parts with 16-bit width. If an instruction requires only less than 16-bit precision for the calculation, it can be executed on the 16-bit data-paths in parallel. This means that we can achieve a spatial redundancy without any hardware and performance increases.

This paper is organized as follows: Section 2 explains the concept of our approach. Section 3 proposes two implementation methods for the data-path partitioning, and the coverage of the failure in this technique is defined. Section 4 evaluates the

Table 1. Comparison with Conventional Approach

	Conventional (Redundancy)		Proposed (Redundancy)	
	Temporal	Spatial	DPSM	DPCM
Temporal Fault	✓	✓	✓	✓
Permanent Fault		✓	✓	✓
Execution Time	↑	→	↑	↗
Hardware Cost	→	↑	→	→

performance overhead caused by the proposed approach. Section 5 describes related work. In Section 6, we conclude this paper.

2. A Data-Path Partitioning for Dependable Processors

In this section, we propose two dependable approaches; a simple multiplexing and a compressed multiplexing. In this section, we assume that the original data-path has 32 bit-width, and each 32-bit data execution is completed in one clock cycle. The concept explained here can easily be extended to wider data-paths, e.g., 64-bit data-path processors. Furthermore, it is assumed that the degree of redundancy to be realized is two (or four). Namely, each instruction of the target program needs to be executed two (or four) times in sequential or parallel.

2.1. Simple Multiplexing

The data-path partitioning with simple multiplexing, called DPSM, is a straightforward way to realize the spatial redundancy. The 32-bit data-path is partitioned into two of 16-bit data-paths. Therefore, all of the instructions are executed by consuming two clock cycles. Figure 1 depicts the concept of DPSM. Here, we assume that three instructions are executed. In this figure, "H1" indicates the highest 8-bit data position in the data-path, while "L0" shows the lowest one. The leftmost figure shows the execution on the original 32-bit data-path without any hardware redundancy. In this case, three clock cycles (3CC) are consumed. SR2 is a DPSM implementation with double-multiplexed structure, i.e. the degree of redundancy is two. The execution of each instruction is divided into two steps: the calculation for lower 16 bits and that for higher 16 bits. First, the execution of lower 16 bits is performed at both two of 16-bit data-paths. Then the higher 16 bits are executed. By comparing the execution results generated from the partitioned data-paths, we can detect faults. Therefore, the execution time becomes

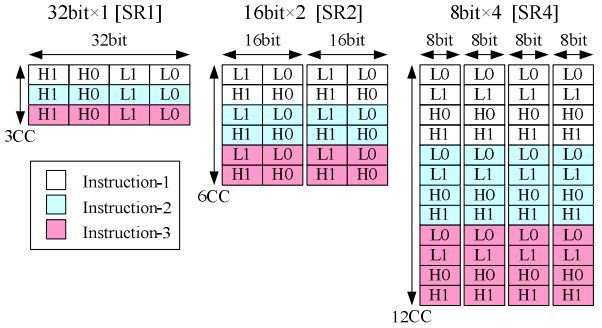

Figure 1. DPSM

double (6CC). SR4 is a case that the degree of redundancy is four, and it requires 12 clock cycles to execute. As shown in Table 1, the DPSM has the same characteristics as the conventional temporal redundancy in terms of the performance and cost. However, there is an advantage over the conventional approach, i.e., the ability to detect the permanent fault.

2. 2. Compressed Multiplexing

As explained in Section 2.1, although DPSM can support to detect both temporal and permanent faults, the main drawback is the negative impact on performance. Here, we call the bit-width to be required for each instruction execution the ***effective bit-width***. For instance, if an ADD instruction is executed with 8-bit char-type variables in the C programming language, the maximum of the effective bit-width is nine (if we would like to keep the carry generated at the MSB position). However, since the DPSM assumes pessimistically that the effective bit-width of all instructions is 32 bits, each instruction consumes several clock cycles for its execution.

It is known that a wide range of applications have small operand sizes. For example, in SPECint95, roughly 50% of instructions have both operands less than or equal to 16-bits [2]. Another example is a picture encoding/decoding, such as MPEG and JPEG. These media applications mainly deal with 8-bit pixel values. The data-path partitioning with compressed multiplexing, called DPCM, exploits this kind of narrow bit-width operations.

Figure 2 shows the execution of the three instructions on DPCM. In the case that the degree of redundancy is two, if the effective bit-width of these instructions are less than or equal to 16 bits, DPCM does not perform the calculation for higher 16 bits. Namely, we can complete the execution of each instruction in one clock cycle if its effective bit-width is smaller than or equal to the bit-width of the partitioned data-path. As long as this condition is

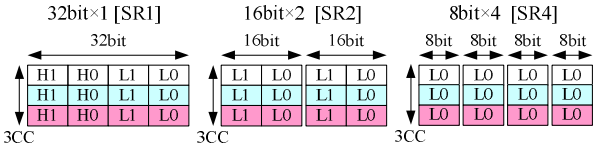

Figure 2. DPCM

satisfied, unlike the DPSM, DPCM does not cause performance degradation, as shown in Figure 2.

3. Implementation Alternatives

In order to implement the data-path partitioning for dependable computer systems, we can consider at least two approaches: static and dynamic. This section shows the concept of these two approaches.

3.1. Static data-path partitioning

The static approach virtually partitions the processor's data-path. Namely, a dedicated compiler explicitly generates the object codes for realizing the data-path partitioning. For example, for the execution of a 16-bit add instruction, the compiler generates the instructions 1) to set the operands to both higher and lower bit positions and 2) to perform the add operation. Of cause, the compiler need to ensure that there is no carry bits from the lower partitioned data-path to the higher one. Figure 3 shows the concept of the static approach.

There are two main advantages. First, we do not require drastic hardware modification in order to achieve spatial redundancy. Therefore, it may be possible to apply this concept to already embedded processors. Next, a special software like OS can manage the degree of redundancy. Based on user requirements or execution environment, for instance, OS can select an object code to achieve the best efficiency for performance, energy, and dependability.

The main challenge of the compiler-based optimization is to estimate the effective bit-width at off-line. To solve this issue, the technique proposed in [6] may be acceptable. Another issue is to develop the special compiler which supports the sub-word optimization in order to generate the data-path partitioned object code. We believe that the Valen-C compiler which has been developed to optimize the data-path bit-width can be exploited for our purpose [7].

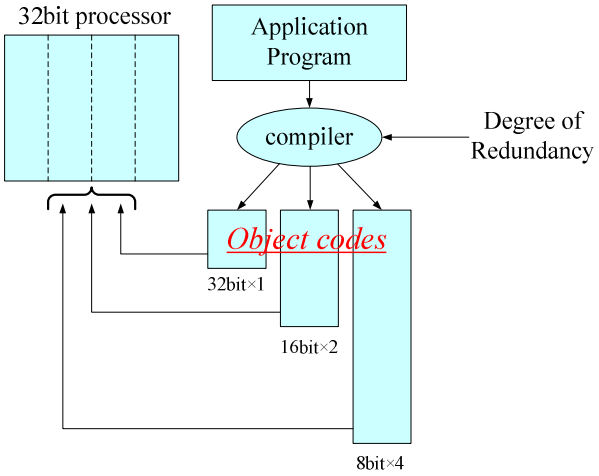

Figure 3. Static Optimization for Data-path Partitioning

3.2. Dynamic data-path partitioning

The dynamic approach examines the effective bit-width at run-time, and attempts to realize a given redundancy. In DPCM, the element which analyzes effective bit-width dynamically is required. For example, in order to realize spatial redundancy, a hardware which divides the data read from the register file is required. To implement this kind of special function, the dynamic analysis technique proposed by [2] may be applicable. Before writing a data into the register file, we compare the generated results on the partitioned higher data-path and the lower one.

Unlike static approach, it is possible to analyze effective bit-width at run-time. Thus, the compatibility of object code can be maintained.

3.3. Coverage

In this section, we discuss the coverage of failure detection ensured by our data-partitioning method. Here, we assume a simple five-stage pipeline processor as shown in Figure 4. In the shaded portions, we can detect any error. Fundamentally, our method can be adapted to the data-path. Therefore, the memory sub-systems such as instruction and data caches need to be protected by using parity or ECC (Error-correcting code). In addition, our approach does not cover the control logics, the comparators used to check the error, branch prediction units. Therefore, they need to be protected by employing other technique, e.g., duplicating the logics or layout designs with enough margins. In addition, for the dynamic data-path partitioning, the special hardware to divide the operand data need to be protected.

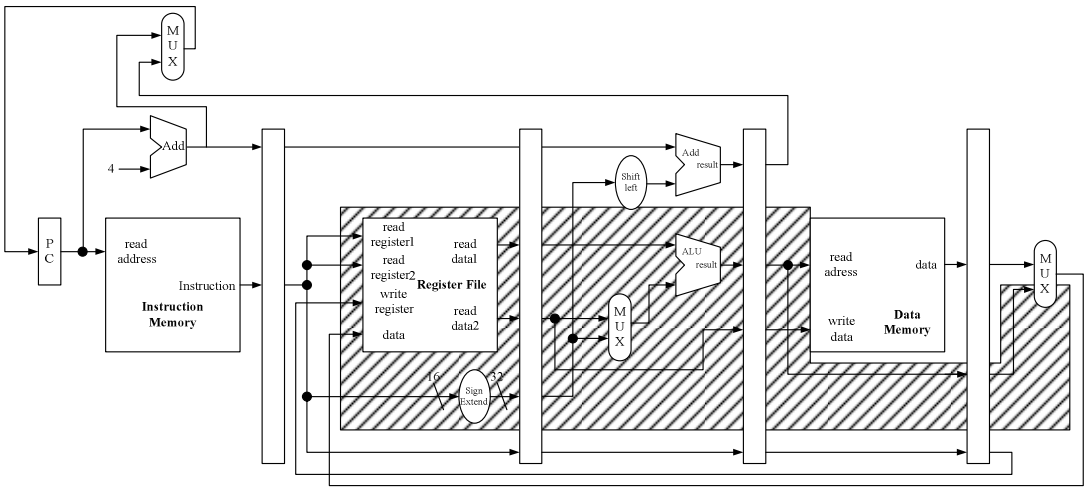

Figure 4. Coverage of data-path partitioning

4. Evaluation

4.1 Experimental Setup

In order to evaluate the negative effect of the proposed data-path partitioning, we performed instruction-level processor simulations to measure execution time. We extended SimpleScalar simulation tool set (ver.3.0d) [8] and executed seven benchmark programs from the SPEC2000 benchmark suite [9]. The small input data sets provided by the SPEC were used. We estimate the execution time as follows:

*Execution Time = IC * CPI * CCT,*

- *IC* : instruction count,
- *CPI* : clock per instruction,
- *CCT* : clock cycle time.

In this experiment, we assume that CPI for any instruction is 1. In addition, CCT is assumed to be a fixed value. Therefore, the execution time depends only on the total number of instructions executed, *IC*. To model the execution time of the data-path partitioning, we use the following equations:

$IC_{SR2} = IC_{org} + IC_{gt16b} * 1,$

$IC_{SR4} = IC_{org} + IC_{gt8b} * 3,$

- IC_{org} : Instruction Counts with the 32-bit data-path,
- IC_{SR2} : Instruction Counts for SR2,
- IC_{SR4} : Instruction Counts for SR4,
- IC_{gt16b} : The number of instructions with greater than or equal to 16 bits of the effective bit-width,
- IC_{gt8b} : The number of instructions with greater than or equal to 8 bits of the effective bit-width.

To evaluate the potential of the proposed approach, we assumed perfect caches. In the DPSM, IC_{gt16b} and IC_{gt8b} are the same as IC_{org} , because it does not consider the effective bit-width. Therefore, the execution time becomes *X* times longer compared with the original data-path model, where *X* is the degree of redundancy required. On the other hand, for the DPCM, we measured IC_{org}, IC_{SR2}, and IC_{SR4} based on the instruction-level processor simulations.

4.2 Results

Figure 5 shows the simulation results. All of the results are normalized to the execution time with the non-partitioned 32-bit original data-path. From the figure, we see that the DPCM can restrain the performance overhead around 50% when the degree of redundancy is two. This penalty is relatively large when we compare it with the conventional organization. However, the DPCM particularly executes each program two times, and we can detect both temporal and permanent faults. In addition, as explained in Section 3, the DPCM does not produce large hardware overhead. Therefore, we consider that the DPCM is promising.

4.3 Discussion

To analyze the observed results in detail, we measured the execution frequency and the rate of compressible instructions, as shown in Table 2. This result is the average of all benchmarks. We see that 41.59% of all executed instructions are ALU instructions, and 52.62% and 36.32% of them can be executed in one cycle if we realize the degree of redundancy 2 and 4, respectively. However, the load/store instructions which occupy about 37% of the executed instructions are hardly uncompressible. This

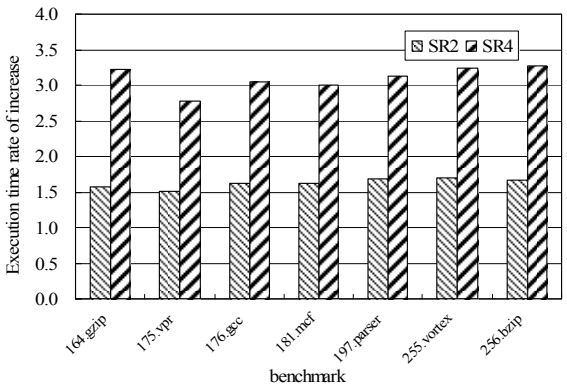

Figure 5. Execution time overhead

comes from the fact that usually the base address is set to a large value. For instance, if we partition the memory space into two parts, instruction and data segments, the MSB for the data address is set to one, loosing the opportunity to execute with narrow bit-width data-paths. Therefore, in order to alleviate the negative performance impact of the proposed approach, another technique for load/store instructions will be required. Note that for branch instructions, our method can be exploited only for the branch-condition examination. The address calculation for the next PC is out of the coverage.

5. Related work

AR-SMT [4] is an architectural support to realize a temporal redundancy without producing serial performance degradation. AR-SMT executes each thread two times on SMT (simultaneous multithreading) structure, so that faults can be detected by comparing the generated results. DIVA [1] attempts to detect errors by means of chaining two execution cores, i.e., realizing a spatial redundancy. The main processor core is followed by a checker processor. Namely, the execution results generated by the main processor are examined by the checker processor. Compared with the previous approaches, our data-path partitioning requires neither supporting SMT execution which increases the complexity of hardware nor employing multi-core organization. Therefore, our method can easily be applied even if the design constraint for area is very tight.

Data-path partitioning has been originally proposed to reduce energy consumption. We can reduce energy reduction by gating or disenabling the unused parts of the data-path at run-time [2] or at design time [3]. In this paper, we attempt to exploit the data-path partitioning to improve reliability. Recently, a promising approach which exploits sub-word

Table 2. Rate of compressed instruction

	Execution Frequency	Rate of Compressed Instruction	
		SR2	SR4
Branch	18.23%	82.43%	75.61%
load/store	37.32%	0.00%	0.00%
ALU	41.59%	52.62%	36.32%
Others	2.86%	0.00%	0.00%

operations has been proposed [5]. In this approach, data-path partitioning is performed at run-time as dynamic DPCM scheme. In this paper, we also consider a static implementation for the data-path partitioning.

6. Conclusions

In this paper, we have proposed the concept of data-path partitioning for improving reliability of processors. The data-path partitioning with compressed multiplexing (DPCM) attempts to exploit narrow-width calculations in order to alleviate the negative effect to the performance. We have performed a primly evaluation, and have found that the proposed approach has some possibility to achieve doubled spatial redundancy with only 50% of performance degradation.

This paper has discussed only the concept of the data-path partitioning. Our ongoing work is to establish the complete microarchitecture to support the proposed idea. We are now designing a processor which supported the data-path partitioning. In addition, developing a dedicated compiler for supporting the static data-path partitioning is our future work.

Acknowledgements

I would like to thank Prof. Shingi Tomita, Prof. Hiroto Yasuura, and all other members of PREST “information infrastructure and applications” research group for discussing at technical meetings. This research was supported in part by the Grant-in-Aid for Creative Basic Research, 14GS0218.

Reference

[1] T.M. Austin. “DIVA: a reliable substrate for deep submicron microarchitecture design,” *In Proc. MICRO-32,* pp. 196–207, Nov. 1999.

[2] D. Brooks and M. Martonosi. “Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance.” *HPCA*, pp.13-22, 1999.

[3] Y. Cao and H. Yasuura. “A System-level Energy Minimization Approach Using Datapath Width Optimization,” *Int. Symp. on Low Power Electronics and Design*, pp.231-236, Aug. 2001.
[4] E. Rotenberg. “AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessors,” *29th International Symposium on Fault-Tolerant Computing*, June 1999.
[5] T. Sato, "Exploiting Sub-word Parallelism for Dependable Processors," *WSEAS Transactions on Information Science and Applications*, issue 6, vol.1, pp.1051-1056, December 2004.
[6] H. Yamashita, H. Yasuura, F. N. Eko, and C. Yun, “Variable Size Analysis and Validation of Computation Quality,” *Proc. of Workshop on High-Level Design Validation and Test* , pp.95-100, Nov. 2000.
[7] H. Yasuura, H. Tomiyama, A. Inoue, and F. N. Eko, “Embedded System Design Using Soft-Core Processor and Valen-C,” *IIS Journal of Information Science and Engineering, vol. 14, no. 3*, pp. 587-603, Sep.1998.
[8] SimpleScalar Tool Sets, http://www.simplescalar.com/.
[9] SPEC (Standard Performance Evaluation Corporation), http://www.specbench.org/.