Securing Execution in Simultaneous Multithreaded Processors

Bernhard Fechner[1]

[1] FernUniversität Hagen, Fachbereich Informatik,
Lehrgebiet Parallelität und VLSI, 58084 Hagen
Bernhard.Fechner@fernuni-hagen.de

1 Introduction

In today's sub-micron, ultra-low voltage and high clock rate technologies, circuits are much more vulnerable to single event upsets (SEUs) causing mostly (90%) transient but also permanent hardware (10%) faults. For future processors it will therefore be essential to secure execution against SEUs. SEUs are modeled by bit-flips of the corresponding latches or memory cells. Newer recovery and roll-forward schemes, like [2] and [5] use simultaneous multithreaded processors [1] to increase performance by using multiple hardware threads. To apply these recovery schemes, hash values for each hardware thread must be computed.

2 Signatures for Multithreaded Pipeline Execution

We present a technique to compute thread signatures on a micro-architectural level, exploiting the deep-pipeline execution scheme on a minimal multithreaded system (two hardware threads). Let P be an S-stage pipelined $(s_1,\ldots,s_s)$, superscalar 2-way multithreaded processor with a finite instruction set of $B\neq\emptyset$ instructions and two sets of instruction streams m, $n\in\mathrm{IN}$. $i_1,\ldots,i_n$, $j_1,\ldots,j_m\in B$, $\forall a\leq n\wedge b\leq m.|i_a|=|j_b|=C$, $I=\{i_1,\ldots,i_n\}\neq\emptyset$, $J=\{j_1,\ldots,j_m\}\neq\emptyset$ which are fetched by two hardware threads. For simplicity, we set I=J. Instruction streams do not have to be necessarily finite, because of program loops. In each stage of the pipeline there can one or multiple parts of decoded instructions from both threads, so $t_1\rightarrow i_a\rightarrow s_i$, $t_2\rightarrow j_b\rightarrow s_j$, where i=j for (multiple) out-of-order resources, $i\neq j$ else. Each stage has a little storage, where the processor saves the thread-ID. Hardware redundancy can be used to store multiple copies of the thread IDs, where the correct id is chosen according to a majority voting. The signature computation involves the well known cyclic redundancy check codes (CRCs) [3,4]. Suppose as a message m(x) the content of the latches of pipeline stages, e.g. the fetch stage. We use thread IDs of active instruction streams for each stage, so that we *can* have two different generator polynomials g(x),h(x) where $\forall x.\exists t.c(x)g(x)\oplus t=c(x)h(x)$. Since it is I=J, we set t=0. The computation of data checksums d(x) is enabled by inserting a multiplexer. We compute the signature for block multithreading according to Fig. 1:

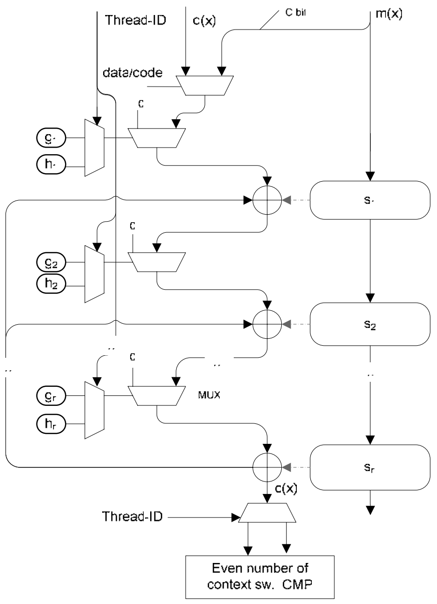

Fig. 1. Computation for block multithreading

A decrementer is initialized with the length of the pipeline. If it reaches zero, the part of the instruction stream in the pipeline before the context switch will participate to the corresponding checksum. After an even number of context switches (we support two threads) the signatures are compared. If the results are not equal an error in the pipeline execution is detected. Additionally of taking the first stage into checksum computation, we can use information out of any pipeline stage (arrows from $s_2,\ldots,s_n$ to XOR-gates). For fine-grained multithreading, the circuit described in Fig. 1 must be modified, so that a multiplexer for each polynomial factor selects the correct instruction stream. Common branch prediction schemes will not hurt the scheme because speculated instructions of both threads will be in both checksums, but schemes based on pre-recognition of branches or pre-computation by threads [5], will lead to different checksums.

References

1. D. Tullsen, S. Eggers, and H. Levy, *Simultaneous Multithreading: Maximizing On-chip Parallelism, 22nd Annual International Symposium on Computer Architecture*, June 1995
2. B. Fechner, J. Keller, P. Sobe. *Performance Estimation of Virtual Duplex Systems on Simultaneous Multithreaded Processors*. In Proc. 9th IEEE Workshop on Fault-Tolerant Parallel, Distributed and Network-Centric Systems, Santa Fe, April 2004
3. S. Lin, D. Costello, *Error Control Coding*, Prentice-Hall, 1983
4. Peterson, W. & E. Weldon. *Error-Correcting Codes*, MIT Press, Second Edition, 1972
5. S.K. Reinhardt, S.S. Mukherjee. *Transient Fault Detection via Simultaneous Multithreading*. In Proc. of the 27th annual International Symposium on Computer Architecture, pp. 25 – 36, 2000