Improving the Reliability of Embedded Systems with Cache and SPM

Meng Wang, Yi Wang, Duo Liu, and Zili Shao
Department of Computing
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
{csmewang,csywang,csdliu,cszlshao}@comp.polyu.edu.hk

Abstract

In this paper, we develop a compiler-assisted thermal-aware data allocation algorithm to improve the reliability of embedded systems with cache and SPM (Scratch-pad Memory). Our basic idea is to distribute the workload evenly between the cache and SPM in order to alleviate the temperature hot spots in the on-chip memory system. In the algorithm, considering the size of SPM, we first divide the loop iterations into two parts, and put the accessed data of the first part into SPM. Then we perform code transformation based on the partitioning of iterations. By alternatively using the data cache and SPM, the peak temperature is reduced. We implement our technique and simulate them using the Trimaran infrastructure with power models for cache and SPM, and the thermal simulator, HotSpot, on a set of benchmarks from DSPstone and MiBench. The experimental results show that our technique can significantly improve the reliability of the on-chip memory system.

I. Introduction

Embedded systems have become a pervasive part of daily life, used for applications ranging from mobile consumer electronics to vehicle controllers. These systems are often designed to operate for long periods in critical environments without human intervention. However, due to various hardware errors and failures, the lifetime of the embedded system is limited. The recent study [7] shows that most hardware failure mechanisms are exponentially related to temperature. Therefore, it becomes an important research problem to reduce the on-chip peak temperature for improving the reliability of embedded systems.

In most embedded processors (e.g. ARM10E), both on-chip caches and scratch-pads are included in the memory subsystems. Cache is designed to improve the performance by exploiting the available locality in the program. The on-chip cache typically consumes 20-50% of the processor's area and energy consumption, a fraction that is increasing with time [5]. SPM (Scratch-pad Memory) refers to the memory residing on-chip that is mapped into the memory address space disjoint from the off-chip memory. The main difference between the cache and SPM is that the SPM guarantees single-cycle access latency whereas an access to the cache may result in a miss thereby incurring longer latency due to off-chip access. Moreover, compared to a hardware-managed cache of the same capacity, the recent study in [5] shows that a SPM has 34% smaller area and 40% lower power consumption. However, in such hybrid on-chip memory system, we cannot allocate all data to SPM due to its limited size. To take full advantage of such hybrid architecture with both data cache and SPM, in this paper, we focus on a compiler-based approach to minimize its peak temperature. Since loops are usually the most time and power consuming parts of embedded applications, we develop a novel loop data allocation technique to make the thermal behavior more balanced in order to improve the reliability of the system.

For on-chip memory systems, a lot of research efforts have been put to reduce power and temperature for both cache and SPM [9], [10], [8]. For multi-bank memory architectures, Hu et al. [9] proposed an algorithm to iteratively find the variable partition such that the maximum energy saving is achieved while satisfying the given performance constraint. Yang et al. [10] developed a dynamic cache sub-array permutation scheme using crossbars in the address predecoder to alleviate the thermal stress on high-leakage area. Steinke et al. [8] proposed an algorithm to analyze the application code and to select the program and data to be replaced into the SPM. However, in this work, as the partitioning of data is fixed, SPM consumes significant power due to the placement of frequently accessed data. Different from the previous work, we propose an iteration-

978-1-4244-5113-5/09/$25.00

level data partition method to alternatively use data cache and SPM. In our approach, both data cache and SPM can be cooled down for some iterations to reduce peak temperature of both sides.

In this paper, we propose a compiler-directed thermal-aware loop data allocation algorithm for embedded systems with both on-chip data cache and SPM. Our basic idea is to distribute the workload evenly between the cache and SPM in order to alleviate the temperature hot spots in the on-chip memory system. In the algorithm, considering the size of SPM, we first divide the loop iterations into two parts, and put the accessed data of the first part into the SPM. Then we perform code transformation based on the partitioning of iterations. To the best of our knowledge, this is the first compiler-directed work to explore both cache and SPM for thermal-aware optimization. We have implemented this work in Trimaran [2] and conducted experiments using a set of benchmarks from DSPstone [11] and MiBench [4]. We simulate the results on the cycle-accurate VLIW simulator of Trimaran [2] based on the power models [5], [6], the HotSpot [3] and the reliability model [7]. The experimental results reveal that our technique can significantly improve the reliability of embedded systems with cache and SPM.

The rest of the paper is organized as follows. In Section II, we introduce the reliability model. Our algorithm is presented in Section III. The experimental results are provided and analyzed in Section IV. The conclusions are shown in Section V.

II. Reliability Model

Processor errors can be generally classified into two categories, soft errors and hard errors. Soft errors are errors in processor execution due to electrical noise or external radiation. Hard errors physically damage the chips when they occur, and are viewed as the lifetime reliability concern. In this paper, we address the reliability problems of on-chip memory systems with both data cache and SPM based on the RAMP model in [7]. The MTTF (Mean-Time-To-Failure or the expected lifetime of the processor) is used as a metric for evaluation. The RAMP model consists of four critical hard-failure mechanisms, Electro-migration, stress migration, gate-oxide breakdown or time dependent dielectric breakdown (TDDB), and thermal cycling. And, three of them have exponential dependencies on temperature. The total MTTF of a component is the inverse of the sum of all four failure rates. It can be calculated as,

$$MTTF_{cache} = \frac{1}{\sum_{i=1}^{4} x_i} \quad (1)$$

$$MTTF_{SPM} = \frac{1}{\sum_{j=1}^{4} x_j} \quad (2)$$

where x_i and x_j represent the failure rate of the ith and jth failure of cache and SPM, respectively. We assume the four hard failures contribute equally to $MTTF_{cache}$ and $MTTF_{SPM}$ at 80 degrees.

III. Thermal-Aware Loop Data Allocation Algorithm

Algorithm III.1 Thermal-aware loop data allocation algorithm.

Require: Intermediate code, SPM_Size, Iter_Num.

Ensure: Thermal-aware code with predicate control.

1: Divide the loop iterations into two groups based on the size of SPM and the reliability constraint, in which one is for execution with SPM and the other one is for data cache;

2: Call function **Code_Transformation()** to perform thermal-aware code transformation based on the analysis result of the above step.

The overall algorithm is shown in algorithm III.1. In the algorithm, SPM_Size is the size of the SPM, and the $Iter\_Num$ is the number of iterations of the loop which can be obtained by using program profiling of the compiler. Considering the size of SPM and the reliability constraint, we first divide the loop iterations into several parts, and put the accessed data of the first part into the SPM. Then we call function $Code\_Transformation()$ to perform code transformation based on the partitioning of iterations. The function $Code\_Transformation()$ is shown in algorithm III.2.

A. Reliability-aware Iteration Partition

In this section, we show how we divide the iterations based on the reliability constraint.

In this paper, we use $cacti$ [6] to determine the power per access for caches. The power per access for scratch-pad memories is determined using the model presented in [5], [8].

The power $P(x_i)$ consumed by a memory operation x_i is expressed as:

$$P(x_i) = \begin{cases} P_{SPM}(x_i) \\ P_{Cache}(x_i) \end{cases} \quad (3)$$

where P_{Cache} can be calculated as follows,

$$P_{Cache}(x_i) = Hit(x_i) * P_{hit} + Miss(x_i) * P_{miss} \quad (4)$$

where functions $Hit(x_i)$ and $Miss(x_i)$ return the number of hits and misses, respectively, while fetching the instructions of memory operation x_i. P_{hit} is the power of a hit and P_{miss} is the power of a miss in the data cache. Since

there are no misses when a memory operation x_i is to access the SPM, we can directly have $P_{SPM}(x_i)$ as the power per access of the SPM.

The thermal model used in this paper is based on [3]. For simplicity, we assume the cache and SPM are isolated blocks. The hypothetical temperature T_{cache} and T_{SPM} of the data cache and SPM are determined out of its power consumption by the formula,

$$T_{cache} = T_{icache} + \theta_{cache} \times P_{cache} \quad (5)$$

$$T_{SPM} = T_{ispm} + \theta_{SPM} \times P_{cache} \quad (6)$$

Here, T_{icache} and T_{ispm} are the initial temperature of the data cache and SPM, respectively. θ_{cache} and θ_{SPM} are the thermal resistance of the cache and SPM with their cooling systems, respectively. Note that T_{icache} and T_{ispm} equal to the environmental temperature when we first calculate the temperature of the on-chip memory system, and they equal to the temperature of the state at which we have to use the data cache or the SPM, respectively.

For one iteration of the loop, we obtain the power P_{iter} consumed by all n memory operations which access the data cache according to equation 3. In this paper, we assume that every loop iteration consumes an equal amount of power and thus generates an equal amount of heat. As data cache consumes more power than SPM and is more vulnerable to hardware failures which will lead to big performance loss, we give high priority to reduce temperature of data cache when we perform iteration partition.

Based on equations 3, 5, 6, we set the value of $T_{threshold}$ as a temperature constraint to maintain low failure rate of the data cache. We determine the maximum number of iterations $iter_{max}$ which we can use the data cache for data accessing while achieving the desired reliability requirement as follows,

$$iter_{max} = \frac{T_{threshold} - T_{icache}}{\theta_{cache} \times \sum_{i=1}^{n} P_{Cache}(x_i)} \quad (7)$$

Here, $iter_{max}$ is the maximum number of iterations for execution with data cache. In our technique, we try to fully utilize the SPM and "cool down" the data cache. However, as the size of SPM is limited, we have to use data cache for some iterations whose number is at most $iter_{max}$ to prevent unpredictable cache behavior due to its high operating temperature. When the program is executed with data cache for $iter_{max}$ iterations, we put the data into the SPM and reset the predicate value to change the program behavior. When the SPM is full, we recalculate $iter_{max}$ and change the program to execute with the data cache. Note that, the initial temperature T_{icache} of the data cache is reduced when the SPM is used since no more power is consumed by the data cache during this time interval.

B. Code Transformation

Algorithm III.2 Function Code_Transformation().

Require: Intermediate code, loop iteration partitioning result.
Ensure: Thermal-aware code with predicate control.

1: **Step 1:** Generate the prologue to initialize for execution with SPM before entering the loop body. In the prologue, the following items are added
- Flush out the cache;
- Copy the accessed data of the first part of the iteration partitioning result into the SPM;
- Assign value true to the predicate register;
- Use registers to hold the value of time for changing to execute with data cache;

2: **Step 2:** Generate the new loop body as follows,
- Replace each memory operation with two new memory operations, one is for SPM and the other one is for data cache. The memory operations are extended with predicate operands, and the operation for SPM will be run when the predicate is set as true.
- When the loop enter the iteration for changing to data cache, assign value false to the predicate register.
- Copy the the accessed data of the first part of the iteration partitioning result back to the off-chip memory DRAM;

In Code_Transformation(), our basic idea is to control the program execution by extending the memory operation with predicate operand. It consists of two steps. In step 1, we generate the prologue to perform initialization for execution with SPM before entering the loop body. In step 2, we change the loop body based on the iteration partitioning result. In this paper, we the intermediate code generated by Trimaran [2], as the input. We choose Trimaran because it is an open-source compiler infrastructure with full support from the open-source community. Note that our technique is general enough and can be applied in different compilers.

In the prologue, we first copy the corresponding accessed data of first several loop iterations into the SPM. Then we assign the true value to the predicate to guarantee the SPM-based memory operation being executed in the loop body. We also use registers to hold the value of time for changing the execution status from SPM to data cache.

In the loop body, we design two new sets of memory operations, one is for SPM and the other one is for data cache. The memory operations are extended with predicate operands. We first replace the original memory operations with ours. Then, an instruction for predicate register assignment is created to control the execution status. The predicate defining instruction is of the form "pred_eq Pout, src1, src2". This instruction assigns value to Pout according to a comparison of src1 and src2 specified

by equal relation. When the loop enter the second part of the iteration partitioning, we assign false value to the predicate register to assure the execution with data cache. And, we copy the data from SPM to the off chip memory to guarantee the correctness of the program before accessing the data cache.

In the following, we show how our technique works using a running example. A real DSP (Digital Signal Processing) application, a FIR (Finite Impulse Response Filter) program, and its corresponding intermediate code generated by the Trimaran compiler [2] are shown in Figure 1.

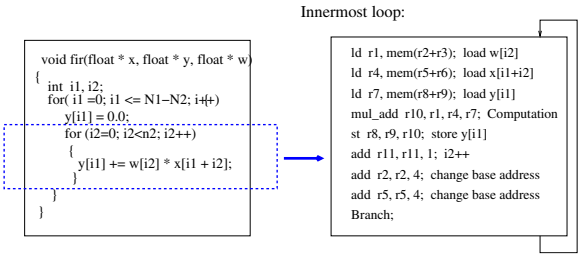

Fig. 1. C code for FIR program and its corresponding intermediate code.

We assume the innermost loop has 1000 iterations, and the size of SPM is large enough to hold the accessed data of array x, y and w for 500 iterations. Using our technique, we divide the loop into two halves and put the accessed data of the first half into SPM. The intermediate code generated by our technique is shown in Figure 2.

In this example, we first put the accessed data of the first 500 iterations into the SPM, and use registers to hold the value of time for changing to execution with cache. We generate the prologue for the above transformations before entering the loop body.

In the transformed loop body, we create two sets of memory operations, one is for accessing data from SPM, and the other one remains the same as the original one which utilizes data cache. The predicate operand P1 is added into the memory operation to control the exchanging between the SPM and the data cache. The instruction "$\mathtt{pred\_eq}\ -\mathtt{P1}, \mathtt{r11}, \mathtt{r12}$" denotes that the predicate register "-P1" is assigned 1 when the value of register "r11" equals to that of "r12". When the loop is analyzed to be executed with data cache, we set 0 for the predicate operand P1 and perform initialization for the cache. Thus, we can balance the thermal profile of the on-chip memory system by distribute the workload evenly between the SPM and cache.

IV. Experiments

We have implemented this work in Trimaran [2] and conducted experiments using a set of 18 benchmarks from DSPstone [11] and MiBench [4]. The benchmarks includes $\mathtt{dot\_product}$, $\mathtt{n\_complex\_updates}$, fir, fir2dim, lms, matrix1*3, matrix, matrix2, $\mathtt{fft\_stage\_scaled}$, $\mathtt{fft\_input\_scaled}$, bfencrypt, bfdecrypt, cjpeg, djpeg, gsmencode, gsmdecode, rawcaudio and rawdaudio.

We simulate the results on the cycle-accurate VLIW simulator Trimaran [2] based on the power models [5], [6] and the HotSpot [3]. In this section, we first discuss our implementation and simulation environment in section IV-A, and then give the experimental results and discussion in section IV-B2.

A. The Implementation and Simulation Platform

Our technique is integrated into Elcor, the back end of Trimaran [2]. Major modifications are performed to integrate our technique into the loop optimization module of Trimaran. In the experiments, we configure the simulator to simulate the floorplan shown in Figure 3. This floorplan is taken from the TI's data sheet [1] as an approximation with on-chip SPM. The detailed configurations of the Trimaran simulator and its memory system is shown in Table I.

Processor technology	65nm
V_{dd}	1.0V
Processor Frequency	4.0GHz
Processor core size	$20.2\mathrm{mm}^2 (4.5\mathrm{mm} \times 4.5\mathrm{mm})$
SPM	64KB
L1 data cache	64KB, 4way 32B blocks, 2cycles
L1 instruction cache	64KB, 4way 32B blocks, 2cycles
L2 cache	512KB,8way 128 blocks, 16cycles
Memory access time	First access: 250 cycles Subsequently: 6cycles

TABLE I. Processor parameters.

In order to track the thermal behavior of the on-chip memory system, we incorporate the power models in [5], [6] into the Trimaran simulator to capture energy per access of memory operations. With the provided power consumption, we use the HotSpot [3], an architectural-level thermal model, to estimate the temperature of the data cache and the SPM. The parameters for HotSpot are shown in Table II.

B. Results and Discussion

In the experiments, we obtain the results of MTTF improvement and code size expansion on the code generated

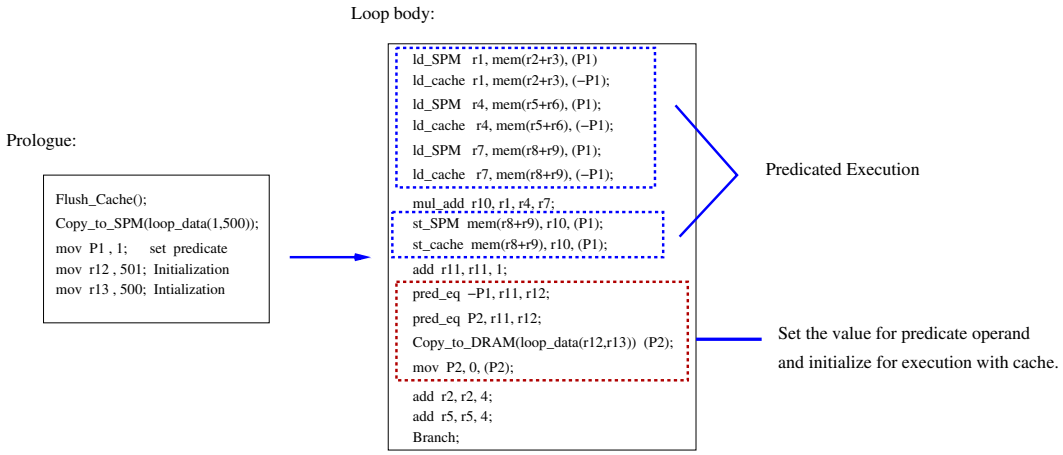

Fig. 2. The intermediate code generated by our technique.

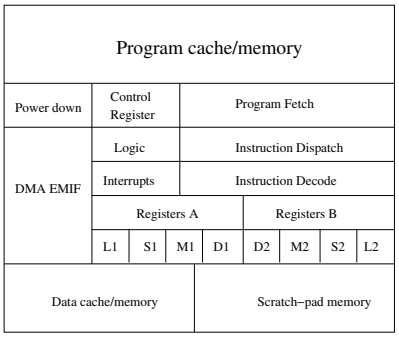

Fig. 3. Floor-plan of chip simulated.

Initial Temperature	60 degree
Ambient temperature	45 degree
Package thermal resistance	0.8 K/W
Die	0.5mm thick, 7.52mm*7.52mm
Heat spreader	1mm thick, 1cm*1cm
Heat sink	7mm thick, 6cm*6cm

TABLE II. The parameters for HotSpot.

by our technique. We compare these results with that of the original code without thermal-aware compilation support. We assume that the data cache miss ratio is 10%. In the experiment, we only apply our technique for the loops whose number of iterations is more than 100 since they are the critical part for power and heat generation.

1) **Reliability Improvement***:* In Figure 4 and Figure 5, we show the result of reliability improvement of the data cache and SPM when running different benchmarks, respectively. Based on the power consumption, and temperature estimates obtained from HotSpot, we calculate the MTTF of each component based on the reliability model, RAMP, in [7]. On average, our technique contributes to an improvement of 7.33% on the cache lifetime, and leads to an improvement of 5.42% on the reliability of SPM.

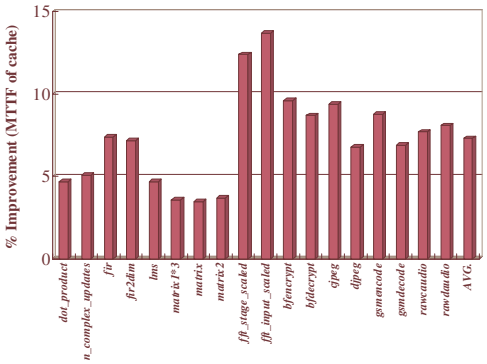

Fig. 4. MTTF improvement of the data cache.

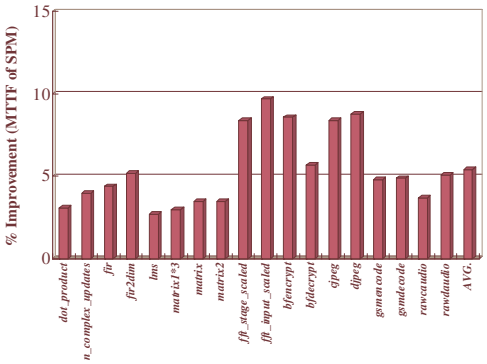

Fig. 5. MTTF improvement of SPM (scratch-pad memory).

The discussion is as follows. The reason for improv-

ing the reliability of the memory subsystems is that our technique reduces the peak temperature of both the cache and SPM, and thus the hard failure rate is becoming lower accordingly. Our technique reduces the peak temperature by distributing the workload evenly between the data cache and the SPM. With our algorithm, the cache access rate of the programs is reduced to a low level, and the dynamic and leakage power of the data cache is reduced accordingly. And, the data cache and the SPM are alternatively cooled down with different program behaviors. Thus, our technique alleviates the temperature hot spots of the on-chip memory system. The proposed approach works particular well with data intensive benchmarks.

2) **Code Size Expansion***:* The percentage of the code size expansion is shown in Figure 6. On average, the results show that our technique leads to an expansion of 2.32% for the benchmarks. The reason of the expansion is that our approach generates prologue for initialization and replaces the original memory operation with new instructions. However, in the experiment, we only apply technique for the loop whose number of execution iteration is more than 100 since they are the critical part for power and heat generation. With such small code size expansion, our technique is suitable for embedded systems.

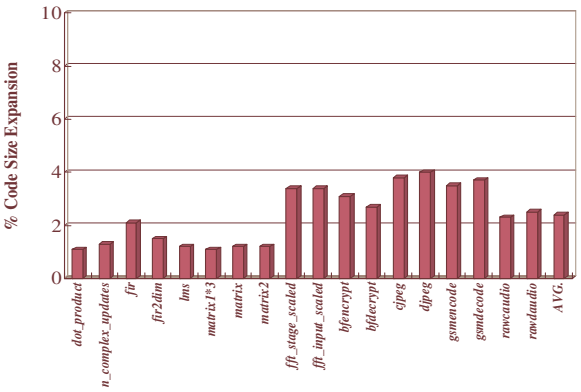

Fig. 6. Code size expansion.

V. Conclusions

This paper studied the thermal-aware loop data allocation problem for embedded processors with both on-chip data cache and SPM. We developed a compiler-directed approach to distribute the workload evenly between the cache and SPM in order to improve the reliability of the on-chip memory system. We implemented our technique and simulate them using the Trimaran infrastructure, power models, and the HotSpot on a set of benchmarks from DSPstone and MiBench. The experimental results show that our technique can significantly improve the reliability of embedded systems.

Acknowledgment

The work described in this paper is partially supported by the grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (GRF PolyU 5269/08E) and PolyU 1-ZV5S.

References

[1] *Texas Instruments*. http://www.ti.com.

[2] *The Trimaran Compiler Research Infrastructure*. http://www.trimaran.org/.

[3] K.Skadron, M.R.Stan, W.Huang, S.Velusamy, K.Sankaranarayanan, and D.Tarjan. Temperature-aware microarchitecture. In *The 30th annual International Symposium on Computer Architecture*, pages 2–13, 2003.

[4] M.R.Guthaus, J.S.Ringenberg, D.Ernst, T.M.Austin, T.Mudge, and R.B.Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the IEEE International Workshop on Workload Characterization*, pages 3–14, 2001.

[5] R.Banakar, S.Steinke, and B.-S.Lee. Scratchpad memory: A design alternative for cache on-chip memory in embedded systems. In *The 10th IEEE International Symposium on Hardware/Software Codesign*, 2002.

[6] S.J.E.Wilton and N.P.Jouppi. Cacti: An enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits*, pages 677–688, 1996.

[7] J. Srinivasan, S. V.Adve, P. Bose, and J. A.Rivers. The case for lifetime reliability-aware microprocessors. In *The 31st Annual International Symposium on Computer Architecture (ISCA 2004)*, 2004.

[8] S. Steinke, L. Wehmeyer, B. Lee, and P. Marwedel. Assigning program and data objects to scratchpad for energy reduction. In *The 2002 Design, Automation and Test in Europe Conference and Exhibition (DATE 2002)*, pages 409–415, 2002.

[9] Z. Wang and X. S. Hu. Energy-aware variable partitioning and instruction scheduling for multibank memory architectures. In *ACM Transactions on Design Automation of Electronic Systems*, pages 369–388, 2005.

[10] W. Wu, J. Yang, S. X.-D. Tan, and S.-L. Lu. Improving the reliability of on-chip data caches under process variations. In *The 2007 International Conference on Computer Design (ICCD 2007)*, 2007.

[11] V. Zivojnovic, J.Martinez, C.Schlager, and H.Meyr. Dspstone: A dsp-oriented benchmarking methodology. In *Proceedings of the 1994 International Conference on Signal Processing Applications and Technology*, 1994.