A Power-Aware Cache and Register File Design Space Exploration

Mehdi Alipour , Mostafa E. Salehi

Department of Electrical & Computer, Islamic Azad University, Qazvin Branch, Qazvin, Iran.

Received 10 December 2010; revised 18 March 2011; accepted 2 April 2011

Abstract

In the near future, embedded processors need to support more network applications, computation-intensive packet-processing tasks will become heavier, and new performance bottlenecks will be introduced in the embedded system designs. Since memory access delay and also the number of processor registers significantly affect processor performance, cache and register file are two major parts in designing embedded processor architectures. Increasing the sizes of cache and register file leads to the performance improvement in packet-processing tasks in high traffic networks with too many packets, but the increased area, power consumption, and memory access delay are the overheads of these techniques. Therefore, implementing these components in the optimum size is of great interest in the design of embedded processors. In this regard, this paper explores the effect of cache and register file size on the performance of processors while considering power consumption in calculating the optimum size of these components for embedded applications. The results show that although having bigger caches and register files helps with the performance improvement in embedded processors, increasing the size of these parameters beyond the threshold level makes the performance improvement saturated and finally decreased. Furthermore, a major part of the power of embedded processors is consumed in the memory.

Keywords: Embedded processor, design space exploration, cache, register file, performance, power consumption.

1. Introduction

In recent years, embedded applications and internet traffic have become sophisticated and heavier, respectively. In this regard, since future embedded processors will encounter more computation-intensive embedded applications, designing high performance processors is recommended. By decreasing the feature size of applications and having the technology of chip multiprocessors (CMP) that are usually multi-thread processors, the user’s performance is somehow guaranteed. Key components in designing these processors are cache and register file because the performance of a processor is closely related to cache access and having enough registers.

Recently in numerous studies, multi-thread processors are used to design fast processors especially in network processors [4], [9], [11], [23], [25], and [26]. In [3] a Markov model based on fine grain multithreading was developed. The resulting Analytical Markov model was faster than the simulation and had dispensable inaccuracy. In the chain, stalled threads defined as states and transitions were based on the cache contention between threads.

Cache memories are usually used to improve the performance and power consumption of processors by bridging the gap between the speed and power consumption of the main memory and CPU. Therefore, the system performance and power consumption depends on the average memory access time and power consumption which make cache an important part in designing embedded processor architectures.

In [4] cache misses were introduced as a factor for reducing memory level parallelism between threads. In [5] thread criticality prediction was used, and for better performance, resources were given to the threads that had higher L2 cache misses called the most critical threads.

To improve packet-processing in network processors, [6]-[8] used direct cache access (DCA). In [9] the processor architecture was based on simultaneous multithreading (SMT) and the cache miss rate was used to assess the improvement in performance.

To find out the effect of cache access delay on performance, a comparison between multi-core and multi-thread processors was made in [10]. Likewise, victim cache was used to improve the performance of a multi-thread processor [11].

Most of the recent studies are based on comparisons between single-core single-thread processors and multi-thread processors. Since multi-thread processors are the heir to the single-thread processors [23], [25], and [26], evaluating some effective parameters like cache and register file size is necessary for designing a multi-thread processor. The main purpose of this paper is, therefore, to study the effect of cache size on the performance of processors because embedded processors process computation and data intensive applications and larger cache sizes give a better performance.

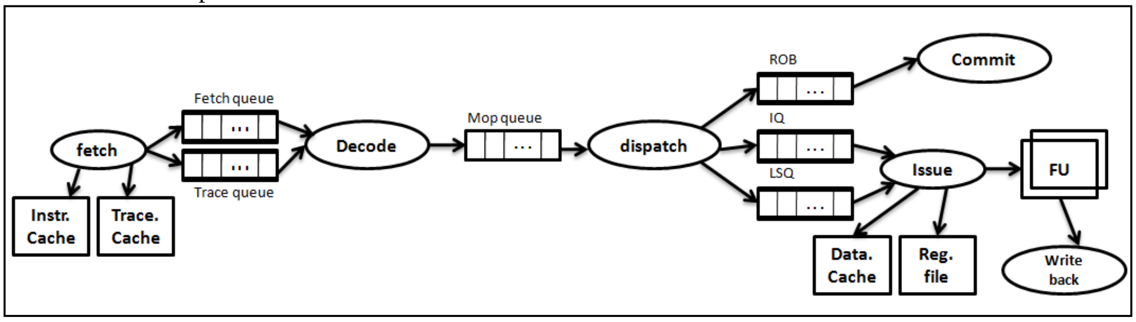


Fig. 1. The processor pipeline of Multi2sim simulator [19]

Generally, one of the easiest way to improve the performance of embedded and network processors is increasing the cache size [2], [12], [13], [14], and [22]-[26] but this improvement significantly increases the occupied area and power consumption of the processor. Hence, it is necessary to find a cache size that makes tradeoffs between the performance, power, and area of the processor.

From another point of view, due to the performance per area parameter, higher performance in a specified area budget is one of the most important needs of a high performance embedded processor. A shortcoming of the recent researches is that they don't place any constraints on the cache size. Thus, because of the limited area budget in embedded processors, in this paper the optimum sizes of L1 and L2 cache are computed, and also because of the longer latency of bigger caches, the best size of the memory hierarchy in relation to this parameter is calculated.

As mentioned above, another essential part in designing embedded processors is register file. Like cache, size of this parameter has a fundamental effect on the processor performance. To improve the performance of an embedded processor, a large register file must be implemented. However, larger register files occupy more area and make a worse critical path [18]. Therefore, exploring the optimum size of the register file is the second purpose of this paper. This issue is highly important because some parameters encourage designers to have large register files. Generally, embedded processors are implemented in multi-issue architectures and out of order (OOO) instruction execution that has renaming logic [16]-[18], [23], [25], and [26]. On the other hand, register files are shared in multi-thread processors, thus they force the designer to have a larger register file [1]. This further shows the great importance of the size of register file. In [15] effects of the size of register file on SMT processors were studied. However, high budget for the number of registers was used. As in recent researches concurrent effects of register file and cache size are not studied, in this paper this issue is investigated. In some way, the present paper is an extended version of [29] that considers the issue of power consumption too. In [29] only performance criteria were explored by the authors. Yet in this study, using the cache and register file range reported in [29] and using a new tool for power consumption, we present power aware optimum sizes of cache and register file for embedded applications.

2. Simulation environment

2.1 Performance Exploration Simulator

For simulation, we use Multi2sim version 2.3.1 [19], a superscalar multi-thread multi-core simulation platform which has 5 stages of pipeline named *fetch, decode, dispatch, issue, writeback, and commit*, and executes x86 instruction sets. Figure 1 shows a block diagram of the processor pipeline modeled in Multi2Sim. In the fetch stage, instructions are read from the instruction or the trace cache. Depending on their origin, the instructions are placed in either the fetch queue or the trace queue. The former contains raw macroinstruction bytes while the latter stores pre-decoded microinstructions (uops). In the decode stage, the instructions are read from the queues, and decoded if necessary. Then, uops are placed in the program order into the uop queue. The fetch and decode stages form the front-end of the pipeline [19]. The dispatch stage takes the uops from the uop queue, renames their source and destination registers, and places them into the reorder buffer (ROB) and the instruction queue (IQ) or the load-store queue (LSQ). The issue stage is in charge of searching both the IQ and LSQ for the instructions with ready source operands, which are scheduled for the corresponding functional unit or data cache. When a uop is completed, the writeback stage stores its destination operand back in the register file. Finally, the completed uops at the head of the ROB are taken by the commit stage and their changes are confirmed. Thus the commit stage is where we can log and count the number of committed instructions for performance comparison. This simulation is described in detail in the simulation method and results section.

To evaluate the requirements of each thread, we use the single issue model for comparison although Multi2sim can run programs in multi-issue platforms. We changed and compiled the source code of the simulator on a 2.4GHz dual core processor with 4GB of RAM and 6MB of cache that runs fedora 10 as the operating system. Using this configuration, the average time of each simulation is about 20 minutes.

2.2 Power Exploration Tool

Since power is as important as performance in embedded processors, we explore power parameters using MCPAT [30] tool, an integrated power, area, and timing modelling framework that supports comprehensive design space exploration for multi-core and many-core processor configurations ranging from 90nm to 22nm and beyond. MCPAT [30] builds both reservation-station and physical register-file models based on real architectures.

3. Benchmarks

The aim of this paper is to calculate the optimum size of cache and register file. Because embedded applications are too pervasive and homogenous, they cannot be a good choice for DSE. Hence we apply our DSE on heterogeneous applications, such that in some of them the data cache is more important while in the others the instruction cache is more significant. To this end, we apply PacketBench [20] and MiBench [27], respectively. PacketBench is a good platform to evaluate the workload characteristics of network processors. It reads and writes packets from and to real packet traces, manages packet memory, and implements a simple application programming interface API. This involves reading and writing trace files and placing packets into the memory data structures used internally by PacketBench.

Many of these functions are implemented on a network processor by specialized hardware components and therefore should not be considered part of the application. There are three categories of programs in this tool: *1- IP forwarding*, which is corresponding to current internet standards. *2- Packet classification,* which is commonly used in firewalls and monitoring systems. *3- Encryption,* a function that modifies the entire payload of the packet. Specific applications that we use from each category are IPv4-Lctrie, Flow-Classification and IPSec, respectively. IPv4-trie performing RFC1812-based packet forwarding is derived from an implementation for the Intel IXP1200 network processor. This application uses a multi-bit trie data structure to store the routing table, which is more efficient in terms of storage space and lookup complexity [20]. Flow classification, another application used in the study, is a common part of various applications such as firewalling, NAT, and network monitoring. The packets passing through the network processor are classified into the flows which are defined by a 5-tuple consisting of the IP source and destination addresses, source and destination port numbers, and transport protocol identifier. The 5-tuple is used to compute a hash index into a hash data structure that uses link lists to resolve collisions [20]. Fulfilling different functions, IPSec is an implementation of the IP Security Protocol [27], where the packet payload is encrypted using the Rijndael Advanced Encryption Standard (AES) algorithm [28]. This is the only application where the packet payload is read and modified.

MiBench is a combination of six different categories. Three of them are selected here: *1- Dijkstar* from the network category, *2- Susan (corners)* from the automotive and industrial control category, and *3- String-search* from the office category. The Dijkstra benchmark constructs a large graph in an adjacency matrix representation and then calculates the shortest path between every pair of nodes using repeated executions of Dijkstra’s algorithm [49]. *Susan* is an image recognition package. It is developed for recognizing corners and edges in magnetic resonance images of the brain [27]. *String-search* searches for given words in phrases using a case insensitive comparison algorithm.

4. Simulation method and results

The main goal of this study is to evaluate the optimum size of cache and register file. At first, we describe the methodology for extracting the proper size of cache. To do so, it is necessary to configure the simulator in a way that the only parameter affecting performance is the size of cache. Thus, for each application the execution number of the main function is calculated in different sizes of L1 and L2 caches. This is done by making changes to some parts of the simulator source code to calculate the cycles of sending a packet (the cycles that are used to execute the main function of each application).

To calculate the beginning address and the end address of the main function, we disassemble the executable code of each benchmark application and extract these addresses. Then these parameters are back annotated to the commit.c and processor.h files of the Multi2sim simulator where a thread is executed.

Having made these changes, we can calculate the number of x86 instructions and macroinstructions and the execution cycles for each specific function.

The second step is to run the simulator with different cache sizes. In this regard it should be pointed out that while based on some recent studies the cache size should be doubled to improve the performance of a processor, it is also important to consider key parameters like area power and cache access delay during the process of doubling the cache size. In this study, therefore, we have used CACTI 5.0 [21], a tool from HP that is a platform for extracting parameters relevant to cache size in fabrication technology. Table 1 shows most of the important parameters used in this research.

To compare the performance based on the cache size, the results extracted from cacti (L1 and L2 cache access delay) are back annotated to Multi2sim. This is done by calculating the simulator cycle time and comparing it to the results of cache access time obtained from CACTI. In this way when the cache size is changed, actual cache access delays are taken into account.

As illustrated by Figure 2, increasing the cache size leads to more cache access delays. For exploring the cache size, the other simulator parameters are set to the default value because the purpose is to find the best cache size for a single-thread single-core processor for embedded applications. That is, the width of the pipeline stages must be one (issue-width =1).

5. Analysis of the simulation results

5.1 Performance Analysis

Figure 3 shows the results of our simulations. In this figure, each axis has a label and the vertical axis (per-pen) shows the performance penalty of the related cache size configuration. According to these results, by increasing the cache size, we can achieve more hit rates,

Table 1
The most important parameters used in cacti

	L1 cache	L2 cache
Cache size	Variable	Variable
Cache line size	Variable	Variable
Associatively	Variable	Variable
Number of banks	1	1
Technology node (nm)	90nm	90nm
Read/write ports	1	1
Exclusive read ports	0	0
Exclusive write ports	0	0
Change tag	No	No
Type of cache	Fast	normal/serial
Temperature (K)	300-400	300-400
RAM cell/transistor type in data array	ITRS-HP	Global
RAM cell/transistor type in tag array	ITRS-HP	Global

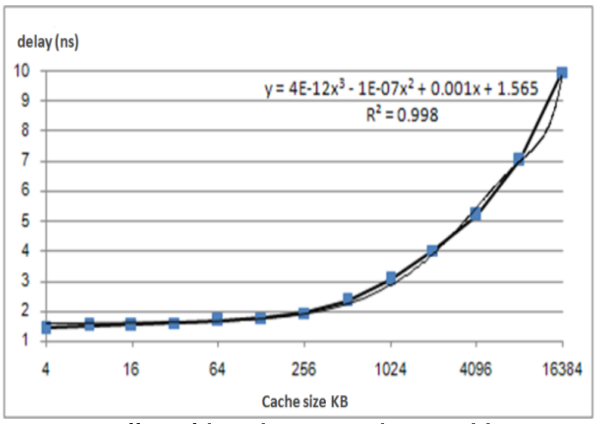


Fig. 2. Effects of the cache size on cache access delay

but because of the longer cache access time of larger caches, from a specific point (the best cache size) the performance improvement is saturated and then even decreased. In other words, doubling the cache size cannot always improve the performance. In fact, area budget is limited and we can't always have a large cache. Hence, we should consider smaller sizes for the cache especially close to the best cache size so that performance degradations would be negligible (3% on average).

To calculate the optimum size of register file, we have applied the parameters used for calculating the best cache size; however, to find out just the effect of register file size on the performance, we have used the best cache size (L1 and L2) concluded in the previous section for the cache size and run the simulator accordingly. Figure 4 indicates the findings of this part of analysis, where the vertical column shows the performance effect (performance penalty) of register file size. Numbers in the vertical column are relative to the best size of register file. Accordingly, although for all applications, on average, the best size of register file is 68 and above but in sizes near the half of this size performance penalty is lower that 5%. The figure also shows that reducing the register file size always decreases the performance but sometimes by doubling the register file size we don't have a noticeable performance improvement. As a result, the first point that the highest performance is reached is determined as the best size for register file.

Another relevant issue is that based on recent researches [23, 25, 26], to have a multi-thread architecture, we need more area budget, and then to run the architecture in the best performance that can be met, a multi issue architecture with renaming logic , ROB, LSQ, IQ and other OOO components which occupy a large area budget are needed. Base on these simulations, we calculated 2 points for the sizes of cache and register file: *1- the best size* that has no performance penalty and occupies a bigger area budget and *2- the optimum size* that has about 3% performance penalty and occupies a smaller area budget.

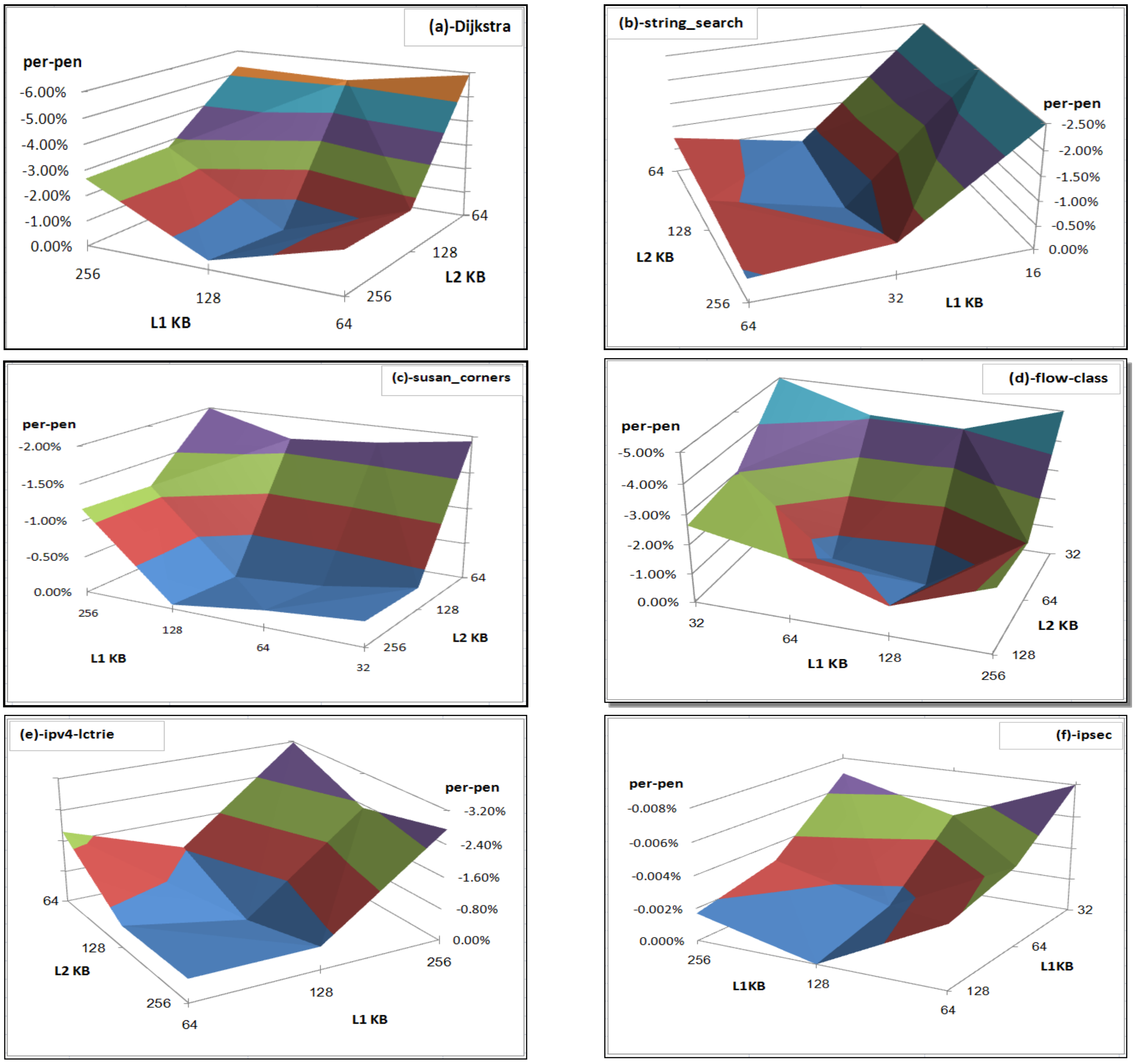


Fig. 3. Effects of the cache size on the performance (a):Dijkstra, (b): String_search, (c):Susan.corners, (d):flow_class, (e):ipv4_lctrie, (f): ipsec.

Consequently, we can deduce that in the optimum size of cache and register file we have saved the area budget of the processors and qualifications to run multi-threads in the higher issue widths is obtained. In other words, the lower area in which more performance is achieved causes the most important parameter of the embedded applications that is performance per area to develop.

As mentioned before, given that multi-thread processors are the heir of single thread processors, extracting the best size of important parameters like cache size and register file size is necessary.

5.2 Power Analysis

We did the back annotation method e.g. we put the results of performance analysis from multi2sim to the input of MCPAT [30]. The inputs are the cycle simulation, number of access to the cache (register file), hit rate, miss rate, number of committed instructions, number of dispatched instructions, etc.

Based on Figure 3, we can introduce 12 cache configurations for the selected embedded application. For L1 we can use 16, 32, 64 and 128 KB (so four points), and for L2 we can use 32, 64 and 128 Kb (so three points). Multiplying 3 by 4, we reach to 12 points for the cache hierarchy. For example, we can have L1=16 KB and L2=64 as configuration number 1. In this way we can reach 12 different cache configurations. By using MCPAT [30], we explored the power parameter of these configurations related to an embedded core to find out how much of power consumption of an embedded processor will be used in the cache hierarchy.

Figure 4 shows the percentage of the cache hierarchy total power (leakage + dynamic) consumption related to the core for all 12 configurations. The best cache hierarchy configuration is cfg number 1 because it consumes the lower percentage of the core total power. According to the results of the total power analysis, in 90nm, up to 32% of the core total power will be consumed in the cache hierarchy. Performance is also one of the most important parameters to tune a cache configuration for embedded applications.

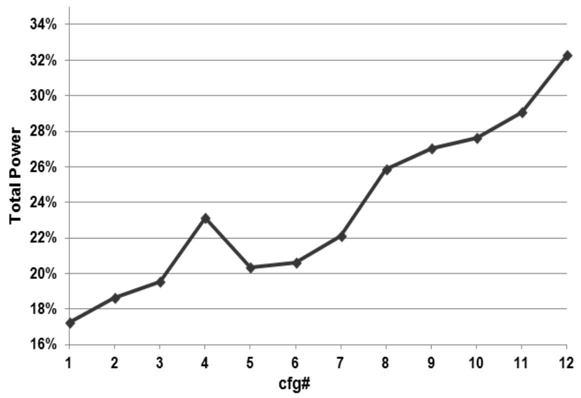


Fig. 4. Analysis of power for different cache hierarchies

Thus the cost functions that consider both power and performance simultaneously can produce better results.

Below, we explore some important parameters based on the power and performance of all 12 configurations which are more effective for embedded applications. To analyze the power of register file, we used MCPAT [30] again. We explored the register file sizes shown in Figure 5. The findings of this part are presented in Figure 6. Based on this figure, the best size among the proposed sizes is 80 that delivers the optimum perfromance per power for register file in the embedded domain. In other sizes, we can see power penalaties. This means that in these sizes power cunsumption is higher that that in the default size (default size=80).

6. Conclusion

In this paper we studied the effect of cache and register file size on the performance of an embedded processor and extracted the best size of these two parameters for embedded applications. The simulation results show that for the selected benchmarks, the best sizes of L1 and L2 caches are 64KB and 128KB, respectively, and the best size of register file is 80. Experiments show that although by increasing the cache size, performance will improve, but in a specific point the performance improvement is saturated and then decreased. Moreover, increasing register file size cannot always improve performance and in a specific size the performance improvement will be saturated. From the area point of view, based on the results of this research, when we select half of the best size of the cache and register file, performance penalty is about 3% on average. In other words, in the sizes smaller than the best size, the acceptable performance can be met. It means we can reach performance requirements in the lower area and also have a better performance/area parameter.

Employing a comprehensive power tool, we explored the power criteria for embedded applications. Considering the power parameters led to reduction in the search space extracted from the performance analysis. Therefore, designers can select the optimum size based on their design criticality from the explored space in this paper.

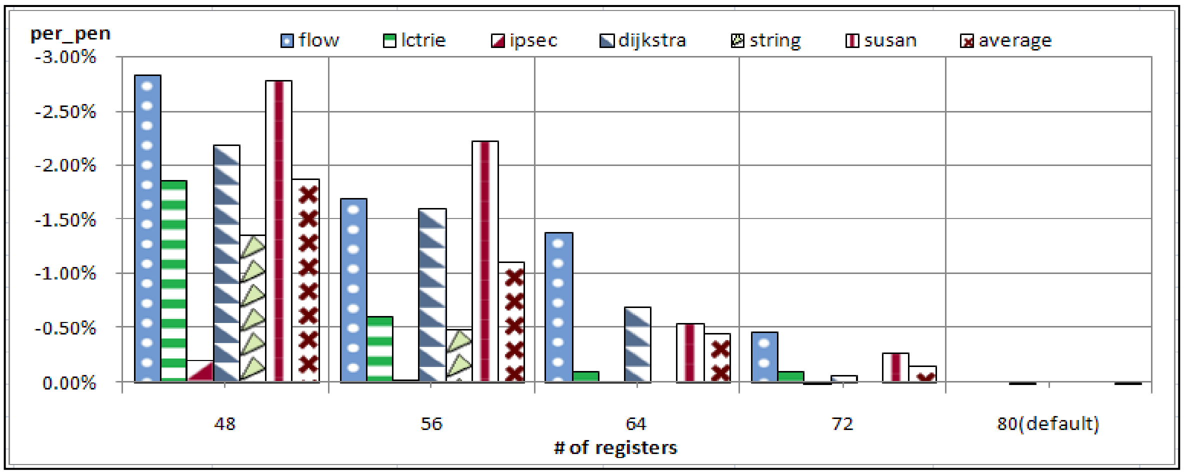
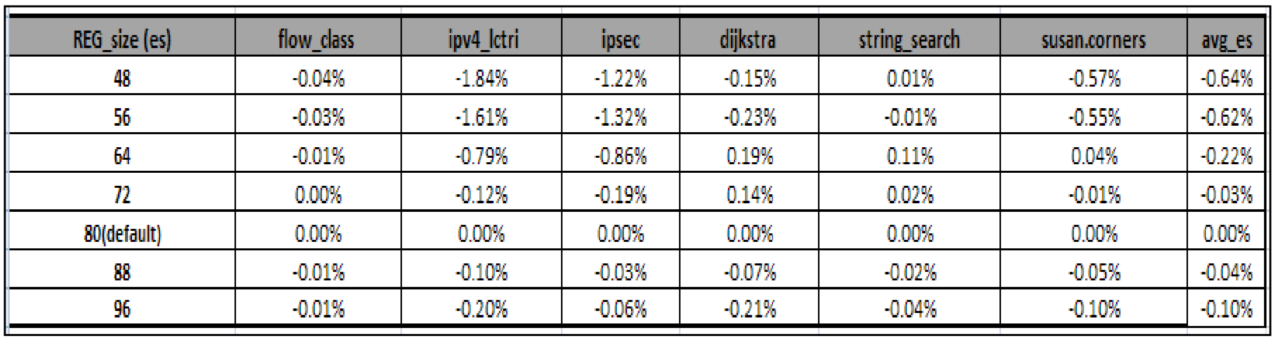


Fig. 5. Effects of the register file size on the performance



REG_size (es)	flow_class	ipv4_lctri	ipsec	dijkstra	string_search	susan.corners	avg_es
48	-0.04%	-1.84%	-1.22%	-0.15%	0.01%	-0.57%	-0.64%
56	-0.03%	-1.61%	-1.32%	-0.23%	-0.01%	-0.55%	-0.62%
64	-0.01%	-0.79%	-0.86%	0.19%	0.11%	0.04%	-0.22%
72	0.00%	-0.12%	-0.19%	0.14%	0.02%	-0.01%	-0.03%
80(default)	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
88	-0.01%	-0.10%	-0.03%	-0.07%	-0.02%	-0.05%	-0.04%
96	-0.01%	-0.20%	-0.06%	-0.21%	-0.04%	-0.10%	-0.10%

Fig. 6. Power analysis of the explored sizes of register file.

References

[1] David A. Patterson, John L. Hennessy.: Computer *organization and design*: the hardware/software interface, Morgan Kaufman 2007.

[2] Davanam, N., Byeong Kil Lee., "Towards Smaller-sized Cache for Mobile Processors Using Shared Set-Associatively," international conference on information technology. pp. 1-6. 2010.

[3] Chen, X.E., Aamodt, T.M., "A First-Order Fine-Grained Multithreaded Throughput Model," International Symposium on High Performance Computer Architecture (HPCA), pp.329-340.2009.

[4] Shailender, Chaudhry. Robert, Cypher. Magnus, Ekman. Martin, Karlsson. Anders, Landin. Sherman, Yip. Haakan, Zeffer. Marc,Tremblay," Simultaneous speculative threading: a novel pipeline architecture implemented in sun's rock processor," international symposium on computer architecture(ISCA 2009). pp. 484-495, 2009.

[5] Abhishek, Bhattacharjee. Margaret, Martonosi. "Thread Criticality Predictors for Dynamic Performance, Power, and Resource Management in Chip Multiprocessors," International Symposium on Computer Architecture (ISCA). pp. 168-176, 2009.

[6] Huggahalli, R. Iyer, R. Tetrick, S.,"Direct cache access for high bandwidth network I/O," International Symposium on computer Architecture (ISCA). pp. 50-59, 2005.

[7] Kumar, A. Huggahalli, R.,"Impact of Cache Coherence Protocols on the Processing of Network Traffic," International symposium on Microarchitecture. (MICRO), pp. 161-171, 2007

[8] Kumar, A. Huggahalli, R. Makineni, S.,"Characterization of Direct Cache Access on Multi-core Systems and 10GbE," International Symposium on High Performance Computer Architecture (HPCA). pp. 341-352, 2009.

[9] Kyueun, Yi. Gaudiot, J.-L.,"Network Applications on Simultaneous Multithreading Processors," Journal of IEEE Transaction on Computer (TCOMPUTER). pp. 1200-1209, 2009

[10] Guz, Z. Bolotin, E. Keidar, I. Kolodny, A. Mendelson, A. Weiser, U.C.," Many- Core vs. Many-Thread Machines:Stay Away From the Valley," Journal Computer Architecture Letters (L-CA), pp.25-28, 2009.

[11] Colohan, C.B. Ailamaki, A.C. Steffan, J.G. Mowry, T.C.,"CMP Support for Large and Dependent Speculative Threads," Journal IEEE Transaction on Parallel and Distributed systems (TPDS), pp. 1041-1054, 2007.

[12] Bienia, C. Kumar, S. Kai Li.," PARSEC vs. SPLASH-2: A quantitative comparison of two multithreaded benchmark suites on Chip- Multiprocessors," International Symposium on Workload Characterization (IISWC), pp.47-56, 2008.

[13] Guanjun, Jiang. Du, Chen. Binbin, Wu. Yi, Zhao. Tianzhou, Chen. Jingwei, Liu.," CMP Thread Assignment Based on Group sharing L2 Cache," International Conference on Embedded Computing, pp. 298-303, 2009.

[14] McNairy, C. Bhatia, R.," Montecito: a dual core dual thread Itanium processor," IEEE Journal MICRO, pp.10-20, 2005.

[15] Alastruey, J. Monreal, T. Cazorla, F. Vinals, V. Valero, M.,"Selection of the Register File Size and the Resource Allocation Policy on SMT Processors Policy on SMT Processors," International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), pp.63-70, 2008.

[16] A, Yamamoto. Y, Tanaka. H, Ando. T, Shimada.," Data prefetching and address pre-calculation through instruction pre-execution with two-step physical register deallocation," in MEDEA-8, pp. 41–48, 2007.

[17] Yamamoto. Y, Tanaka. H, Ando. T, Shimada.,"Two-step physical register deallocation for data prefetching and address precalculation," IPSJ Trans. on Advanced Computing Systems. vol. 1, no. 2, pp. 34–46, 2008.

[18] Tanaka, Y. Ando, H.,"Reducing register file size through instruction pre execution enhanced by value prediction,"IEEE International Conference on Computer Design, pp. 238 – 245. 2009.

[19] R, Ubal. J, Sahuquillo. S, Petit. P, L'opez.," Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors," journal Proc. of the 19th Int'l Symposium on Computer Architecture and High Performance Computing. Oct 2007.

[20] Ramaswamy, Ramaswamy. Tilman, Wolf.," PacketBench: A tool for workload characterization of network processing," in Proc. of IEEE 6th Annual Workshop on Workload Characterization (WWC-6), Austin, TX. pp. 42-50, Oct. 2003.

[21] Shyamkumar Thoziyoor, Naveen Muralimanohar, and Norman P. Jouppi, "CACTI 5.0 technical report", form Advanced Architecture Laboratory, HP Laboratories HPL-2007. [Online]. Available:www.hpl.hp.com/research/cacti/

[22] Hyunjin, Lee. Sangyeun, Cho. Childers, B.R.," StimulusCache: Boosting Performance of Chip Multi-processors with Excess Cache," IEEE 16th International Symposium on High Performance Computer Architecture (HPCA), pp, 1 – 12, 2010.

[23] Chung, E.S. Hoe, J.C.," High-Level Design and Validation of the BlueSPARC Multithreaded Processor," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD). vol. 29, no. 10, pp. 1459– 1470, 2010.

[24] Davanam, N. Byeong, Kil. Lee.," Towards Smaller-sized Cache for Mobile Processors Using Shared Set-Associativity," international conference on information technology, pp. 1-6, 2010.

[25] Kyueun Yi, and Gaudiot J. L, "Network Applications on Simultaneous Multithreading Processors," *IEEE Transaction on Computer (TCOMPUTER).*vol. 59, no. 9, pp. 1200-1209, SEPTEMBER 2010.

[26]Wang, H. Koren, I. Krishna, C.," Utilization-Based Resource Partitioning for Power-Performance Efficiency in SMT Processors," IEEE Transactions on Parallel and Distributed Systems, (TPDS) vol. 22, no. 99, pp. 191-216, 2010.

[27]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 InternationalWorkshop onWorkload Characterization,* pp. 3-14,2001.

[28] S.K. Dash, T. Srikanthan, "Instruction Cache Tuning for Embedded Multitasking Applications," *IEEE/IFIP International Symposium on Rapid System Prototyping,* pp. 152-158, 2009.

[29] Mehdi Alipour and Mostafa E. Salehi "Design Space Exploration to Find the Optimum Cache and Register File Size for Embedded Applications*", 9th Int'l Conf. Embedded Systems and Applications, Pp. 214-219,* ESA'11, las vegas, USA, July 18-21, 2011.

[30] Sheng Li, Jung Ho Ahn, Jay B. Brockman,and Norman P. Jouppi, "McPAT 1.0: An integrated power, area, and timing modeling framework for multicore architectures," available online at: http://www.hpl.hp.com/research/mcpat/McPATAlpha_TechRep.pdf.