Dynamic Register File Partitioning in Superscalar Microprocessors for Energy Efficiency

Meltem Özsoy[1], Y. Onur Koçberber[2], Mehmet Kayaalp[1], Oğuz Ergin[3]
[1]State University of New York - Binghamton
[2]EPFL - Parallel Systems Architecture Lab (PARSA)
[3]TOBB University of Economics and Technology
{mozsoy, yokocberber, mkayaalp, oergin}@etu.edu.tr

***Abstract*— Register file is one of the vital and energy consuming parts inside microprocessor. Many studies show that it is one of the hot spots on the chip. It is also observed by many researchers that many of the produced values in a processor are narrow. By using the narrow values, register files can store fewer bits and may be designed to need less static and dynamic energy. In this paper we propose a register file design that stores data in narrow value groups and values are written to those groups according to their widths. Size of narrow value groups can be set dynamically according to the behavior of the program while having the same performance. We show that the register file which has dynamically changing narrow value groups offers static and dynamic energy savings in the register file up to 65% with negligible performance loss.**

I. Introduction

Superscalar microprocessors use aggressive techniques like out-of-order execution and dynamic scheduling in order to achieve higher performance. The number of instructions that can be inside the processor at a given time is also increased to exploit instruction level parallelism as much as possible. As the processors try to execute large number of instructions in a small time period, larger hardware resources are required to satisfy storage needs.

Register file is one of the major sources of power dissipation in superscalar pipelines [1]. The amount of the consumed energy of the register file inside the CPU ranges from 10% to 16% [31][32] which makes it a spotlight for many researchers. In processors that employ register renaming, a physical register file is used to remove false data dependencies. The size of this physical register file is kept as large as possible to allow more instructions to go through pipeline without being blocked because of the lack of an available physical register. Also recently the datapath widths are increased to 64 bits which also increases the width of the registers causing them to dissipate more energy during reading and writing operations. Although the register file is built to satisfy the resource needs of the instructions when the activity is at its peak level, usually the program has some phases during which the resources are not used to their full extent. For the register file, this underutilization can be in terms of the number of registers [2] or the number of bits inside a register [3].

In this paper, we exploit two observations to reduce the energy dissipation of the register file: (1) many of the registers are idle during some phases of the programs (2) many of the stored values are narrow in the sense that they can be represented with fewer bits than that is provided by the storage space. In order to achieve energy efficiency, we propose static and dynamic partitioning of the register file according to different value-width groups. In our baseline scheme we statically divide the register file into three different width groups (16, 34 and 64 bits); in fact this is similar to having a banked register file structure [4][5][6][7][8] with some narrow banks [18]. As a second scheme, we propose a single register file structure which can be logically divided into three groups to support different value widths. In this second scheme, we power-gate the upper order bits of the region that holds narrow register values where the decision of switching the registers on/off is based on the register usage statistics dynamically. Our results show that, even with static partitioning of the register file it is possible to achieve more than 50% dynamic energy reduction. Our schemes also reduce the static energy dissipation of the register file.

II. Motivation and Related Work

It is previously observed by many researchers that many of produced and consumed values inside a processor contains a lot of unneeded consecutive sign bits on their upper portion [9][10][11][12]. The narrowness of the values were exploited to improve performance [9][11][13], for energy efficiency [10][14] and to improve the reliability of the processors [15][16]. The encoding of bytes that contain all zeros is also another form of using narrow values for energy efficiency [17]. Furthermore, register file partitioning as banked register files was designed in previous works [18][19]. Banked register file design is based on fixed narrowness of the values and fixed amount of registers. In our design, while the amount of registers remain the same, it is possible to achieve higher energy efficiency because of the fact that our dynamic decision scheme does not keep registers powered more than they are needed. Other previous register file designs other than using narrow values, aim to gain from area, power and performance, are hierarchical [20][21][22], clustered [23][24] and virtual [25][26] register files. Again based on the content of the register, narrow values are used to pack multiple values into one register or reusing the registers [3][27][28][29][30].

Although the dynamic resizing [2] and the multi banking [18][33] of the register file are previously proposed an on-demand register width determination approach was never

implemented. This paper aims at reducing the performance degradation caused by a static partitioning scheme while trying to achieve the energy efficiency offered by a perfect static partitioning scheme through dynamic decisions and power-gating. To the best of our knowledge, this is the first work that employs in-register dynamic partitioning to avoid performance degradation while achieving energy efficiency.

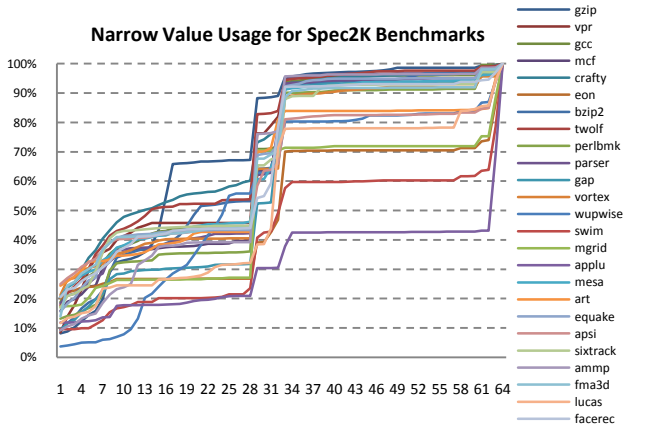

Fig. 1. Value width distribution for SPEC CPU2000 benchmarks.

Fig. 1 shows the value widths across all SPEC CPU2000 benchmarks. As the graph reveals, it is possible to cover more than 90% of the values by using a storage space of 34 bits for most of the benchmarks. According to the narrowness of the values in Fig. 1, if we divide the values into three groups of 16-bit, 34-bit and 64-bits, a large percentage of the values can be covered by the narrower partitions. The goal here is to cover as many of the values as possible inside the narrowest partition, since the narrowest values offer the most energy saving opportunities. After dividing the values into three groups we observed how the produced values fall into these categories in different program phases. For this purpose we divided each benchmark into 10 million cycle chunks and obtained the statistics separately for each chunk. Fig. 2 shows the changes in the rates of occurrence for each value width group. The results show that the narrowness of the values inside the program changes as the program progresses. At some points in the program 16-bit percentage is higher than the other simulated chunks. This observation shows that it may be possible to save more energy or reduce the performance impact of any scheme that makes use of narrow values by dynamically arranging the size of the narrow partitions.

Since the value width that will be created by an instruction cannot be known at the time of register allocation, value widths have to be predicted. Luckily, value widths are highly predictable and complex predictors are not required. This was also observed previously by other researchers who showed that even a last value predictor can achieve more than 80% accuracy [3][34][13]. Since accurate prediction is not the main objective, for simulation purposes, we introduced some bits for the instruction cache to hold only the last value generated by the instruction. These bits are resulted very small 2KB cache holding the narrowness of the last produced value of the instruction. The cache is placed near register file, I-cache and register file accesses also goes to this cache to get/set the narrow value group of the instruction. As we simulated this cache with Cacti HP [35] its power usage is negligible.

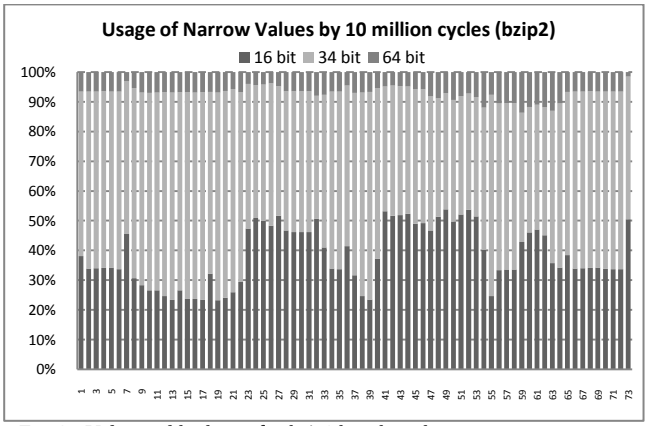

Fig. 2. Value width phases for *bzip2* benchmark.

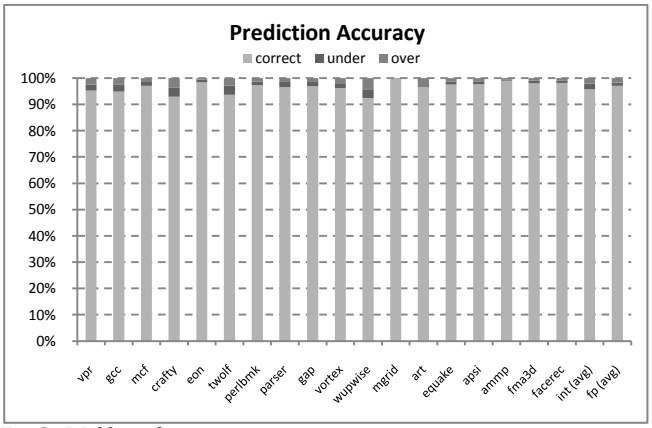

Fig. 3. Width prediction accuracy.

In width prediction there are three prediction cases possible: correct, underprediction and overprediction. Overprediction usually does not cause a problem other than a lost opportunity for energy saving while underprediction is a problem that needs addressing. When a value is underpredicted, it means that the width of the value produced at the writeback stage is more than the space that is allocated to it. In this case a reallocation is unavoidable. Therefore in width prediction, underprediction cases need to be so small that these cases can be solved through uncomplicated procedures such as a flushing operation that is done on a branch misprediction. Fig. 3 shows the prediction accuracy statistics for SPEC CPU2000 benchmarks. As the results reveal, correct and overprediction cases amount to more than 95% of the cases which is vital for a scheme that will be heavily dependent on the success of the width predictor.

III. Partitioned Register File

In a superscalar microprocessor that employs register renaming, a new physical register is assigned to each result-producing instruction at the rename stage. If there are no available physical registers, the pipeline stalls until a register

is free. In this work we propose a register file structure that is capable of holding values that belong to different value width groups. As it is in the baseline case, a register has to be allocated to an instruction but the result produced by the instruction has to be predicted beforehand in order to allocate a suitable register.

A. Static Register File Partitioning

In order to use the narrowness of the values for energy efficiency the register file can be divided into different value width groups. We decided to have three groups as a design choice to show the effectiveness of the proposed scheme. Although it is possible to have more value width groups, it should be noted that the complexity of the proposed schemes increase with the increasing number of value width groups.

Static register file partitioning was also previously investigated by Nalluri et al. in [33] which aim to reduce energy dissipation by using small banks depending on the register file access profile of application. This approach was extended by employing the narrow values by Wang et al. in [18]. Similar to Wangs's scheme, in our design the register file is divided into three groups that are 16-bit, 34-bit and 64-bit wide. Rather than having different banks, we implemented the register file where some of the upper order bits are removed from the layout and the register is only given a limited number of bitcells. After the instructions are brought from the instruction cache, they are decoded and they arrive at the rename stage. Once an instruction arrives at the rename stage a physical register is assigned from a suitable register group according to the result that is produced by the width predictor. It is possible that there are no physical registers available inside the register group indicated by the predictor. When such a case occurs, the register allocator can allocate a register from the wider width group at the expense of some energy efficiency. However, as allocating a wider register is against the purpose of saving energy, the designer may also stall the pipeline and trade some performance loss for energy efficiency.

After an instruction allocates a register, it proceeds to the issue queue of the processor and is issued to an available function unit. After execution the result's width is calculated and is checked against the prediction. If there is a misprediction (in terms of an underprediction) there are some design choices on how to proceed: the processor may flush after fixing the width predictor and brings in the same instruction for a second time or the processor may try to reallocate the register and checks for an available register. For the latter case, it should be noted that reallocating a new register is a costly process as all of the consumers need to be updated with the new register id. In our design the first step for underprediction recovery is to find a new register at the same narrow value group, if the group is out of registers than the bigger group is searched for a new register. If a register is found, the new register tag is broadcasted to the issue queue for dependent instructions. Broadcast is done by using the unused ports of the issue queue by that time. Finally if there is no available register then processor flushes the instructions for the misprediction which is the case that is seen only for the 0.003% of the recovery situations on average. The state machine that shows the new control structure is summarized in Fig. 4.

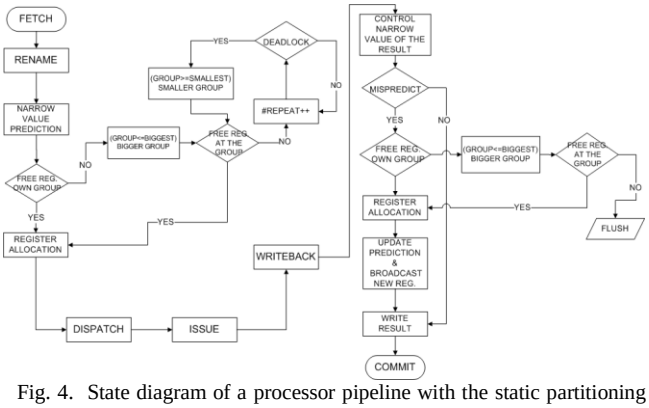

Fig. 4. State diagram of a processor pipeline with the static partitioning scheme.

In static register file partitioning, dynamic and static energy dissipation is reduced since the upper order bits of the narrow registers are completely removed. As the number of registers is fixed in this case and there is no way to fit a wide value into a narrow one, it is unavoidable to have a performance impact. As shown in Fig. 2, the program phases show different rates of value widths at different time intervals. Consequently, there will be a time period when there are more 64-bit registers required than it can be provided by the processor. On the other hand, if the number of 16-bit registers required by the program exceeds that is available on the processor, energy saving opportunities are lost. In order to reduce the performance degradation and energy inefficiency caused by this phenomenon, we propose to adjust the number of registers in each value width group dynamically.

B. Dynamic Register File Partitioning

Dynamic register file partitioning is accomplished by allowing different parts of the register file to be shut down when it is necessary. The register file as a whole is designed in three separate columns where it is possible to power-gate some of the lines if more narrow registers are needed. Also as wide registers are needed by the program phase; it is possible to power some of the registers to increase the number of wide registers.

A register file is an array of SRAM bitcells where the rows of the array share the same word select, power and ground lines. Likewise all the bitcells in the same column share the same bit lines. In order to save area and avoid running unnecessary lines through the register file, two rows of these SRAM arrays share the same power line. For this purpose one row is drawn upside-down on the layout so that two rows can be connected to the same power line. Therefore when the upper order registers are powered up or down, the operation has to be done on both registers at once.

In dynamic register file partitioning, it is important to decide on the interval where the resizing of the width blocks

will be done. As turning the SRAM blocks on/off can be costly in performance, keeping the period too small may degrade performance. Also, if the period is too large, it is difficult to sample meaningful information from the interval and adapt to the program phase. We experimented with the time intervals and chose 1024 cycle time intervals to make a resizing decision.

It is important to decide on what kind of heuristics to apply when the decision time comes. If in a time interval 16-bit blocks are more frequent it makes sense to increase the number of 16-bit blocks to achieve more energy reduction. If the 64-bit blocks were more frequent in the previous block and there was performance loss, it makes sense to turn the power of some registers to increase the number of full-width registers so that the performance degradation is avoided.

At the time of register allocation, if an instruction fails to find an available register that is suitable to its value width prediction, it means that the partitioning in the register file is not correct. In this case, while the instruction proceeds to allocate a register from a wider portion, it increases the corresponding counter of the requested block. Once the time interval is finished the counters are normalized against the size of the corresponding block. This is done because of the difference between a partition that has only one entry and misses 200 instructions and a partition that has 100 entries and misses 200 instructions. In this example the former needs registers more than the latter. Our dynamic partitioning heuristics is shown in Fig. 5. When comparing the ratios of blocks against each other it should be kept in mind that the two rates can be equal or one of them can be zero which may result in a divide by zero error. This state diagram is also updated (by adding a small value that won't alter the result to the counters) in order to solve this problem.

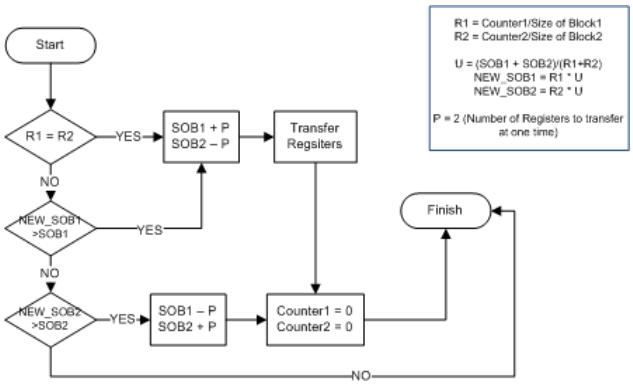

Fig. 5. Dynamic register file partitioning heuristics.

When the normalized access rates of two partitions are the same, the partitions will be divided into two parts. In this case the widest partition may lose a lot of registers which may create a performance drop in the future. In order to avoid this we introduced a limit of 10 registers and do not allow the widest partition to drop under this limit.

In our implementation, we did not change the size of all blocks at each interval. Instead we decided the change between 1st and 2nd blocks in one interval and 2nd and 3rd blocks in the other interval. In each relocation, one couple of rows is transferred to the other block and the bits are powered on and off as needed. Once the decision is made, there can be some valid values just in between the register blocks. If there is a valid value in a block that is to be turned off, this value has to be transferred to another location. This transfer process starts at the register renaming stage by reserving a register for the data that will be moved. Once the data is moved the register is transferred to the narrower block.

In our design two register file lines can be power gated together since it is the most suitable way of power gating due to area and timing constraints. Power gating is applied by placing sleep transistors between V_{DD} and pMOS network or between V_{SS} and nMOS network. There are two major bottlenecks for power gating. One of them is the area overhead, which a coarse grain implementation can overcome by having small number of relatively big sized sleep transistors to drive very long power lines. Second bottleneck of power gating is the wake up delay, which a fine-grain implementation can solve by having sleep transistors to drive small blocks of circuits.

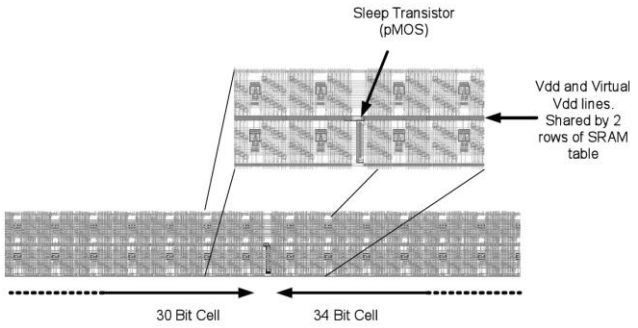

Fig. 6. Sleep transistors used for power gating.

In our implementation, every double lines of register file can be completely put to sleep state. As shown in Fig. 6 sleep transistor cuts off the power of the 30 bit cells in one row. However one V_{DD} line is shared by two rows according to the silicon area constraint. Hence one sleep transistor drives 60 bit cells. Area overhead of the dynamic partitioned register file is negligible when compared to the original register file since a coarse grain approach is achieved by using small sized sleep transistors. The wake up delay for shut off rows is one cycle. Sleep transistor which drives the 60 bit cells are signaled to wake up at 300ps, bit cell power line reaches the steady state nearly at 650ps. When a 2 GHz clock frequency is considered, this 350ps delay corresponds to a one cycle wake up penalty.

IV. Simulation Methodology

In order to observe the effects of proposed schemes on the processor performance and energy dissipation we used the PTLsim simulator [36] which is capable of executing 64-bit x86 instructions. All of the proposed hardware schemes are implemented inside PTLsim and 23 benchmarks from the SPEC CPU2000 benchmark suite are simulated to get an

idea on realistic workloads. All of the benchmarks were run for 1 billion committed instructions with the configuration parameters for our simulations are given in Table 1.

Table 1. Simulation Parameters.

Parameter	Configuration
Machine width	4-wide fetch, 4-wide issue, 4-wide commit
Window size	32 entry issue queue, 80 entry load/store queue, 128 entry ROB, 256 integer physical registers
Function Units and Latency	Integer ALU (6/1), load/store unit (2/2), integer multiply (2/4), integer division (1/32), floating-point addition (2/6), floating-point multiplication (2/6), floating-point division (2/6).
L1 I-Cache	32 KB, 4-way set-associative, 64 byte line, 1 cycle hit time
L1 D-Cache	16 KB, 4–way set–associative, 64 byte line, 2 cycle hit time
L2 Unified Cache	256 KB , 16–way set–associative, 64 byte line, 6 cycles hit time
L3 Unified Cache	4 MB , 32–way set–associative, 64 byte line, 14 cycles hit time
BTB	1024 entry, 4–way set–associative
Branch Predictor	64K entry bimodal and two level Combined
Memory	140 cycle delay

Simulation results were combined with the data obtained from full custom layouts of the highly optimized register file designs which were implemented using the Cadence design tools [37]. Energy dissipations were measured from circuit level simulations of actual full custom CMOS layouts of the processor components and were combined with the statistics gleaned from microarchitectural simulations. A 90nm CMOS (UMC) technology with a V_{DD} of 1 Volt was used in order to measure the energy dissipations at 80°C.

V. Results and Discussions

A. Static Register File Partitioning

In order to compare the dynamic partitioning against static partitioning we set up some test cases for static partitioning. The five combinations we experimented with are shown in Table 2.

Table 2. Static Partitioning Combinations.

	Block 1		Block 2		Block 3	
Comb.	# Rows	# Bits	# Rows	# Bits	# Rows	# Bits
1	85	56	85	60	86	64
2	85	16	85	34	86	64
3	85	10	85	32	86	64
4	50	16	65	32	141	64
5	56	16	76	32	124	64

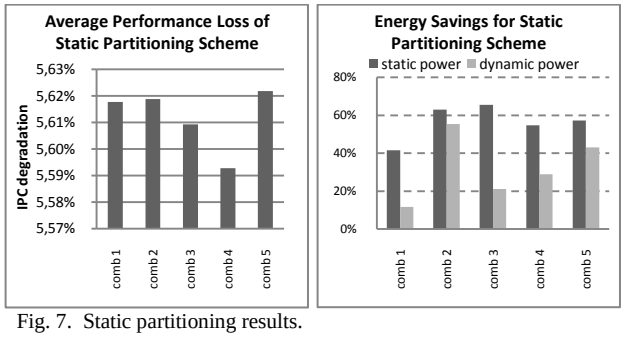

Fig. 7. Static partitioning results.

Fig. 7 shows the performance loss and energy dissipation savings when the static partitioning is employed on average across all SPEC CPU2000 benchmarks. As shown in the figure, static partitioning results in a performance degradation of more than 5.5% on average for all combinations while for combination 2 the best static and dynamic energy dissipation reduction is achieved with almost 60% reduction for both.

B. Dynamic Register File Partitioning

Fig. 8 shows the energy savings achieved by our dynamic partitioning scheme for SPEC CPU2000 benchmarks. As the figure reveals, it is possible to achieve more than 50% reduction in static and dynamic energy dissipation. The performance change caused by the static and dynamic partitioning schemes is shown in Fig. 9. The light colored bar uses the y-axis on the right and shows the performance degradation for the dynamic partitioning scheme while the dark colored by shows the performance degradation for the static partitioning scheme and uses the y-axis on the left. Note that the two y-axes have different scales and the performance degradation of static partitioning is higher than the static partitioning scheme by orders of magnitude for some benchmarks. While some benchmarks show some small performance gain, on the average dynamic partitioning results in very small performance degradation. The graph is in line with the intuition that dynamic will outperform the static partitioning scheme for all benchmarks except for *eon*, *twolf*, *lucas* and *equake* that show some small benefit for the static partitioning scheme.

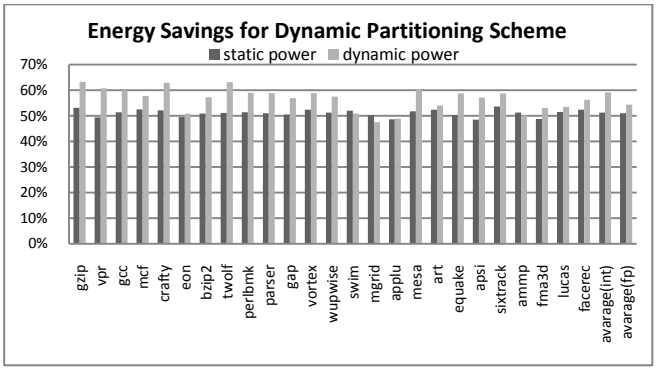

Fig. 8. Dynamic partitioning energy savings.

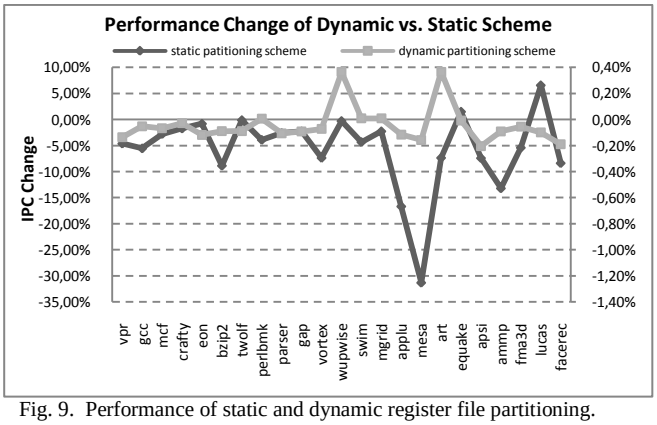

Fig. 9. Performance of static and dynamic register file partitioning.

VI. CONCLUSION

In this paper, we describe a new energy efficient partitioning scheme for physical register file of out of order processors. Our scheme is designed for efficient energy and efficient area usage by exploiting narrow width values. Narrow values are predicted for partitioning of the register file and the values are written after the prediction to the right or a free place according to the status of register file. We simulated 2 different register file partitioning techniques as static and dynamic partitioning. With static partitioning scheme it is possible to achieve 55% reduction in energy dissipation and 50% reduction in register file area with an average 5% IPC loss. We also showed that with a dynamic partitioning scheme it is possible to reduce the energy dissipation by 60% with negligible IPC reduction and virtually no change in the register file area.

VII. ACKNOWLEDGMENTS

This work was supported by the Scientific and Technological Research Council of Turkey (TUBITAK) through the research grants 107E043 and 109E043.

REFERENCES

[1] Brooks, D., Tiwari, V., Martonosi, M., Wattch: a framework for architectural-level power analysis and optimizations, Computer Architecture, 2000. Proceedings of the 27th International Symposium, 83-94, 2000

[2] Ponomarev, D., Kucuk, G., Ghose, K., Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources, Microarchitecture, 2001. MICRO-34. Proceedings. 34th ACM/IEEE International Symposium, 90-101, December 2001

[3] Ergin, O.; Balkan, D.; Ghose, K.; Ponomarev, D., Register Packing: Exploiting Narrow-Width Operands for Reducing Register File Pressure, Microarchitecture, 2004. MICRO-37 2004. 37th International Symposium, 304-315, December 2004

[4] Zyuban, V., Kogge, P., Split Register File Architectures for Inherently Lower Power Microprocessors, Power-Driven Microarchitecture Workshop, in conjunction with ISCA'98, June 1998

[5] Park, I., Powell, M., Vijaykumar, T., Reducing register ports for higher speed and lower energy, Proceedings of the International Symposium on Microarchitecture, 2002.

[6] Cruz, J.L., Gonzalez, A., Valero, M., N.P. Topham, Multiple-banked register file architectures, Proceedings of the 27th Annual International Symposium on Computer Architecture, 2000

[7] Tseng, J., Asanovic, K., Banked multiported register files for high frequency superscalar microprocessors, 30th International Symposium on Computer Architecture (ISCA-30), San Diego, CA, 2003

[8] Saito, T., Maeda, M., Hironaka, T., Tanigawa, K., Sueyoshi, T., Aoyama, K., Koide, T., Mattausch, H.J., Design of superscalar processor with multi-bank register file, Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium, 3507-3510, May 2005

[9] Brooks, D., Martonosi, M., Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance, Proceedings of the 5th International Symposium on High Performance Computer Architecture, 13, January 1999

[10] Kondo, M., Nakamura, H., A small, fast and low-power register file by bit-partitioning, Proceedings of the 11th International Symposium on High-Performance Computer Architecture, 2005

[11] Lipasti, M.H., Mestan, B.R., Gunadi, E., Physical register inlining, Proceedings of the 31st Annual International Symposium on Computer Architecture, 2004

[12] Ergin, O., Exploiting narrow values for energy efficiency in the register files of superscalar microprocessors, 16th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2006

[13] Loh, G.H., Exploiting data-width locality to increase superscalar execution bandwidth, Proceedings of the 35th International Symposium on Microarchitecture (MICRO), 2002

[14] Aggarwal, A., Franklin, M., Energy efficient asymmetrically ported register files, Proceedings of ICCD'03, 2003

[15] Hu, J., Wang, S., Ziavras, S.G., In-register duplication: Exploiting narrow-width value for improving register file reliability, Proceedings of the International Conference on Dependable Systems and Networks (DSN), Philadelphia, PA, 2006

[16] Ergin, O., Unsal, O.S., Vera, X., Gonzalez, A., "Reducing Soft Errors through Operand Width Aware Policies", *IEEE Transactions on Dependable and Secure Computing (TDSC)*, Vol. 6, No.3, July/September 2009, pp.217-230.

[17] Villa, L., Zhang, M., and Asanović, K., Dynamic zero compression for cache energy reduction. In Proceedings of the 33rd Annual ACM/IEEE international Symposium on Microarchitecture (Monterey, California, United States). MICRO 33. ACM, 214-220, New York, NY, 2000

[18] Wang, S., Yang, H., Hu J., Ziavras S.G., Asymmetrically Banked Value-Aware Register Files, VLSI, 2007. ISVLSI '07. IEEE Computer Society Annual Symposium, 363-368, March 2007

[19] S. Wang, H. Yang, J. Hu, and S. G. Ziavras, "Asymmetrically banked value-aware register files for low energy and high performance," Microprocessors and Microsystems, vol. 32, no. 3, pp. 171–182, May 2008.

[20] J. A. Butts and G. S. Sohi, "Use-based register caching with decoupled indexing," in Proceedings of the 31st annual international symposium on Computer architecture, 2004, pp. 302–313.

[21] J. A. Swensen and Y. N. Patt, "Hierarchical registers for scientific computers," in Proc. of ICS'88, 1988, pp. 346–354.

[22] R. Balasubramonian, S. Dwarkadas, and D. H. Albonesi, "Reducing the complexity of the register file in dynamic superscalar processors," in Proceedings of Micro-34, December 2001, pp. 237–248.

[23] K. I. Farkas, P. Chow, N. P. Jouppi, and Z. Vranesic, "The multicluster architecture: reducing cycle time through partitioning," in Proc. Micro 30, 1997, pp. 149–159.

[24] R. Canal, J.-M. Parcerisa, and A. Gonzalez, "Dynamic cluster assignment mechanisms," in Proc. of HPCA-06, 2000, pp. 132–142.

[25] A. Gonzalez, J. Gonzalez, and M. Valero, "Virtual-physical registers," in Proc. of HPCA-4, 1998, pp. 175–184.

[26] T. Monreal, A. Gonzalez, M. Valero, J. Gonzalez, and V. Vinals, "Delaying physical register allocation through virtual-physical registers," in Proc. of Micro-32, 1999, pp. 186–192.

[27] S. Jourdan, R. Ronen, M. Bekerman, B. Shomar, and A. Yoaz, "A novel renaming scheme to exploit value temporal locality through physical register reuse and unification," in Proc. Micro-31, 1998, pp. 216–225.

[28] J. F. Martinez, J. Renau, M. C. Huang, M. Prvulovic, and J. Torrellas, "Cherry: checkpointed early resource recycling in out-of-order microprocessors," in Proc. fo Micro-35, 2002, pp. 3–14.

[29] S. Balakrishnan and G. Sohi, "Exploiting value locality in physical register files," in Proc. of Micro-36, 2003, pp. 265–276.

[30] R. Gonzalez, A. Cristal, D. Ortega, A. Veidenbaum, and M. Valero, "A content aware integer register file organization," in Proceedings of the 31st Annual International Symposium on Computer Architecture, June 2004.

[31] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In International Symposium on Computer Architecture, 2009

[32] Gonzales, D.R.; , "Micro-RISC architecture for the wireless market," *Micro, IEEE* , vol.19, no.4, pp.30-37, Jul-Aug 1999 doi: 10.1109/40.782565

[33] Nalluri, R., Garg, R., Panda, P.R., "Customization of Register File Banking Architecture for Low Power" in Proceedings of the 20th international Conference on VLSI Design Held Jointly with 6th international Conference: Embedded Systems. VLSID. IEEE Computer Society, 239-244, Washington, DC, January 2007

[34] Loh, G., "Width Prediction for Reducing Value Predictor Size and Power", in First Value Prediction Workshop, ISCA, 2003.

[35] S. Thoziyoor, N. Muralimanohar, and N. Jouppi. CACTI 5.0. Technical Report HPL-2007-167, HP Laboratories, 2007

[36] Yourst, M.T., "PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator," Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium, 23-34, April 2007

[37] "Virtuoso Spectre Circuit Simulator" http://www.cadence.com/products/cic/spectre_circuit/pages/default.aspx access date: 07.13.2009