The Inherent Energy Efficiency of Complexity-Adaptive Processors

David H. Albonesi
Dept. of Electrical and Computer Engineering
University of Rochester
Rochester, NY 14627-0231
albonesi@ece.rochester.edu

Abstract

Conventional microprocessor designs that statically set the functionality of the chip at design time may waste considerable energy when running applications whose requirements are poorly matched to the particular hardware organization chosen. This paper describes how Complexity-Adaptive Processors, which employ dynamic structures whose functionality can be modified at runtime, expend less energy as a byproduct of the way in which they optimize performance. Because CAPs attempt to efficiently utilize hardware resources to maximize performance, this improved resource usage results in energy efficiency as well. CAPs exploit repeater methodologies used increasingly in deep submicron designs to achieve these benefits with little or no speed degradation relative to a conventional static design. By tracking hardware activity via performance simulation, we demonstrate that CAPs reduce total expended energy by 23% and 50% for cache hierarchies and instruction queues, respectively, while outperforming conventional designs. The additive effect observed for several applications indicates that a much greater impact can be realized by applying the CAPs approach in concert to a number of hardware structures.

1 Introduction

As power dissipation continues to grow in importance, the hardware resources of high performance microprocessors must be judiciously deployed so as not to needlessly waste energy for little or no performance gain. The major hardware structures of conventional designs, which are fixed at design time, may be inefficiently used at runtime by applications whose requirements are not well-matched to the hardware implementation. For example, an application whose working set is much smaller than the L1 Dcache may waste considerable energy precharging and driving highly capacitive wordlines and bitlines. Similarly, an application whose working set far exceeds the L1 Dcache size may waste energy performing look-ups and fills at multiple levels due to high L1 Dcache miss rates. The most energy-efficient (but not necessarily the best performing) cache organization is that which is well-matched to the application's working set size and access patterns. However, because of the disparity in the cache requirements of various applications, conventional caches often expend much more energy than required for the performance obtained. Other major hardware structures, such as instruction issue queues, similarly waste energy while operating on a diverse workload.

Complexity-Adaptive Processors (CAPs) make more efficient use of chip resources than conventional approaches by tailoring the complexity and clock speed of the chip to the requirements of each individual application. In [1], we show how CAPs can achieve this flexibility without clock speed degradation compared to a conventional approach, and thus achieve significantly greater performance. In this paper, we describe how CAPs can achieve this performance gain while expending considerably less energy than a conventional microprocessor.

The rest of this paper is organized as follows. In the next section, we discuss how the increasing use of repeaters in long interconnects creates the opportunity for new flexible hardware structures. Complexity-Adaptive Processors and then described in Section 3, followed by a discussion in Section 4 of their inherent energy efficiency. In Section 5, our experimental methodology is described. Energy efficiency results are discussed in Section 6, and finally we conclude and discuss future work in Section 7.

2 Dynamic Hardware Structures

As semiconductor feature sizes continue to decrease, to a first order, transistor delays scale linearly with feature size while wire delays remain constant. Thus, wire delays are increasingly dominating overall delay paths. For this reason, repeater methodologies, in which buffers are placed at regular intervals within a long wire to reduce propagation delay, are becoming more commonplace in deep submicron designs. For example, the Sun UltraSPARC-IIi microprocessor, implemented in a 0.25 micron CMOS process, contains over 1,200 buffers to improve wire delay [5]. Note that wire buffers are used not only in busses between major functional blocks, but within self-contained hardware structures as well. The forthcoming HP PA-8500 microprocessor, which is also implemented in 0.25 micron CMOS, uses wire buffers for the global address and data busses of its on-chip caches [3]. As feature sizes decrease to 0.18 micron and below, other smaller structures will require the use of

This research is supported in part by NSF CAREER Award MIP-9701915.

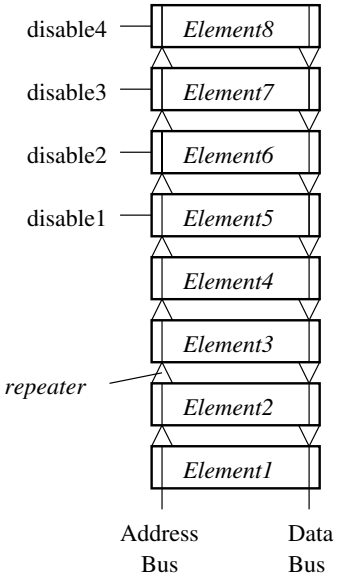

Figure 1: A dynamic hardware structure which can be configured with four to eight elements.

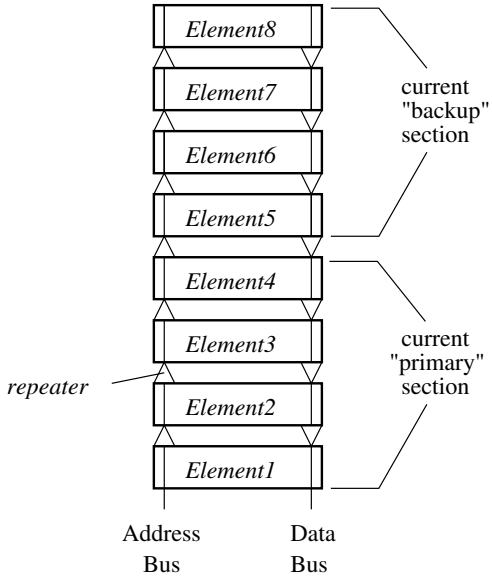

Figure 2: A dynamic hardware structure with configurable "primary" and "backup" sets of elements.

wire buffers in order to meet timing requirements [4].

For these reasons, we expect that many of the major hardware structures of future high performance microprocessors, such as caches, TLBs, branch predictor tables, and instruction queues, will be of the form shown in Figure 1. The hardware structure in this figure consists of replicated elements interconnected by global address/control and data busses driven using repeaters placed at regular intervals to reduce propagation delay. The isolation of individual element capacitances provided by the repeaters creates a distinct hierarchy of element delays, unlike unbuffered structures in which the entire wire capacitance is seen by every element on the bus. By employing address decoder disabling control signals as shown in Figure 1, we can make this structure *dynamic* in the sense that the complexity and delay of the structure can be varied as required by the application. For example, this structure can be configured with between four and eight elements, with the overall delay increasing as a function of the number of elements. Assuming this structure is on the critical timing path with four or more elements, if the clock frequency of the chip is varied according to the number of enabled elements[1], then the IPC/clock rate tradeoff of this structure can be varied at runtime to meet the dynamic requirements of the application. Due to their exploitation of repeater usage, such dynamic hardware structures can be designed with little or no delay penalty relative to a fixed structure.

An alternative to disabling elements is to use them as slower "backups" to the faster "primary" elements as is shown in Figure 2. Here, the difference between the primary and backup elements is the access latency in clock cycles. For example, the four primary elements in Figure 2 may be accessed in fewer cycles than the four backup elements, due to the latter's longer address and data bus delays. Such an approach may be appropriate, for example, for an on-chip Dcache hierarchy, in which the primary and backup elements correlate to L1 and L2 caches, and the division between them is determined as a function of the current working set size and the cycle time and backup element latency of each configuration.

[1] An alternative is to vary the latency in clock cycles of the structure.

3 Complexity-Adaptive Processors

Having discussed how repeater methodologies will lead naturally to the development of dynamic hardware structures, we now describe the overall organization of a Complexity-Adaptive Processor.

The overall elements of a CAP hardware/software system, shown in Figure 3, are as follows:

- Dynamic hardware structures as just described which can vary in complexity and timing (latency and/or cycle time);
- Conventional static hardware structures, used when implementing adaptivity is unwieldly, will strongly impact cycle time, or is ineffective due to a lack of diversity in target application requirements for the particular structure;
- Performance counters which track the performance of each dynamic structure and which are readable via special instructions and accessible to the control hardware;
- Configuration Registers (CRs), loadable by the hardware and by special instructions, which set the configuration of each dynamic structure as well as the clock speed of the chip; different CR values represents different complexity-adaptive configurations, not all of which may be practical;
- A dynamic clocking system whose frequency is controlled via particular CR bits; a change in these bits causes a sequence in which the current clock is disabled and the new one started after an appropriate settling period;

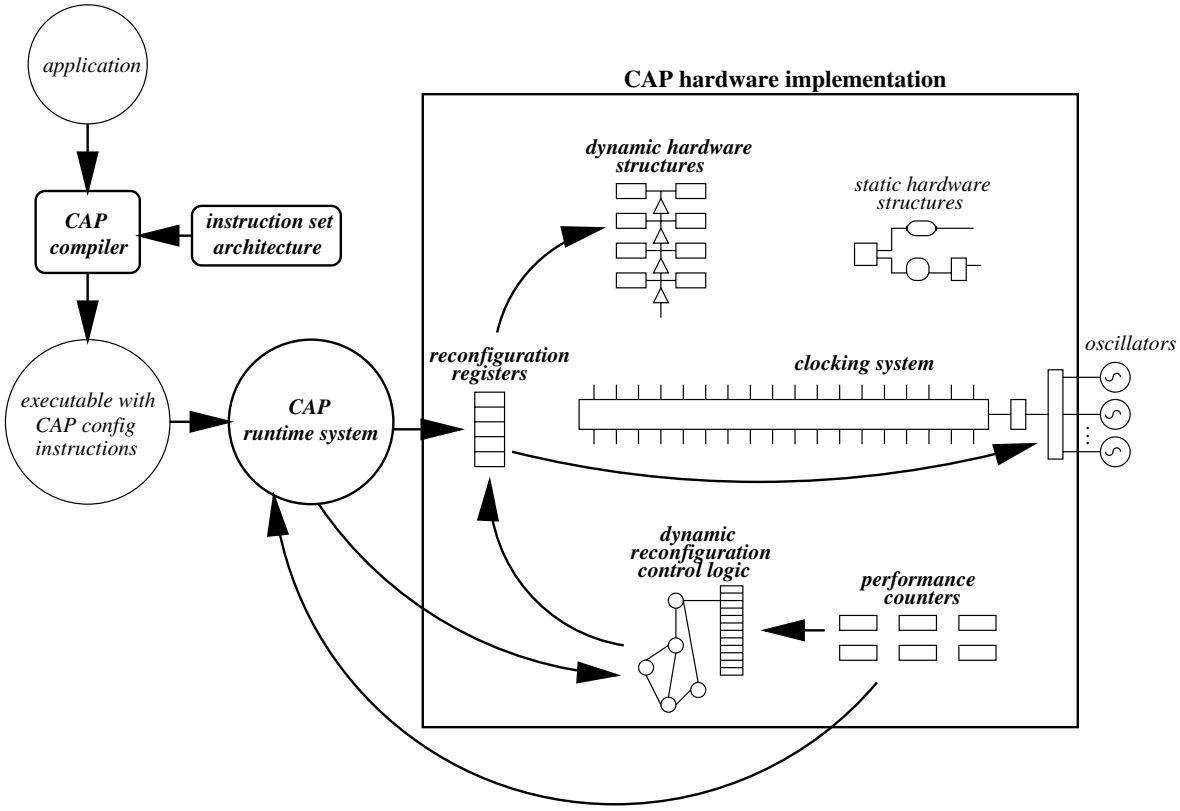

Figure 3: Overall elements of a CAP hardware/software system.

- The instruction set architecture (ISA) consisting of conventional instructions augmented with special instructions for loading CRs and reading the performance counters;
- Configuration control, implemented in the compiler, hardware (dynamic reconfiguration control logic), and runtime system, that acquires information about the application and uses predetermined knowledge about each configuration's IPC and clock speed characteristics to create a *configuration schedule* that matches the hardware implementation to the application dynamically during its execution.

The process of compiling and running an application on a CAP machine is as follows. The CAP compiler analyzes the application's hardware requirements for different phases of its execution. For example, it may analyze data cache requirements based on working set analysis, or determine the ILP based on the control flow graph. With this information, and knowledge about the hardware's available configurations, the compiler determines whether it can with good confidence create an effective configuration schedule, specifying at what points within the application the hardware should be reconfigured, and to which organizations. The schedule is created by inserting special instructions at particular points within the application that load the CRs with the desired configuration. In cases where dynamic runtime information is necessary to determine the schedule, this task is performed by the runtime system or the dynamic reconfiguration control logic. For example, TLB configuration scheduling may be best handled in conjunction with the TLB miss handler, based on runtime TLB performance monitoring, while the optimal branch predictor size may in some cases be best determined by runtime hardware. A major CAP design challenge is determining the optimal configuration schedule for several dynamic structures that interact with each other as well as with static structures, and which may be controlled by up to three different sources (compiler, runtime, and hardware).

Through these various mechanisms, the CRs are loaded at various points in an application's execution, resulting in reconfiguration of dynamic structures and changes in clock frequency. For runtime control, the performance counters are queried at regular intervals of operation, and using history information about past decisions, a prediction is made about the configuration that will perform best over the next interval. Assuming tens of cycles are required for each reconfiguration operation (based on the time to reliably switch clock frequencies), then an interval on the order of thousands of instructions is necessary to maintain a reasonable level of reconfiguration overhead.

4 Improving Energy Efficiency Via CAPs

There are two main aspects of CAPs that allow for reduced power consumption: the ability to disable or optimally allocate (between primary and backup sections) hardware elements, and the ability to control the frequency of the clock. One option is to explicitly manage these features in order to reduce power consumption.

For example, in a portable environment, when battery life falls below a certain threshold, a low power mode in which the processor is still fully functional can be enabled by setting all dynamic hardware structures to their minimum size, and selecting the slowest clock.

In addition, CAP designs have an inherent energy efficiency that is a byproduct of the way in which they optimize performance. Because the CAP hardware is configured to match the characteristics of each application, hardware structures are generally used more efficiently, and thus expend less energy, for a given task. It is this inherent energy efficiency of CAPs that follows naturally from optimizing performance that we explore in the rest of this paper.

5 Experimental Methodology

We examined the results of our preliminary performance analysis [1] of two-level on-chip Dcache hierarchies and instruction issue queues to estimate the relative energy expended by CAP and conventional approaches for these structures. For the Dcache hierarchy, we assumed a total of 128KB of cache, and for the CAP design, allowed the division between the L1 and L2 caches to be varied in steps of 8KB up to a total L1 Dcache size of 64KB on an application-by-application basis. We compared this approach with the best overall-performing conventional design: one with a 16KB 4-way set-associative L1 Dcache with the rest of the 128KB allocated to the L2 cache. We ran each benchmark on our cache simulator for 100 million memory references. The CAP instruction queue varied in size from 16 to 128 entries in steps of 16 entries. Unused entries were disabled. The best-performing conventional design contained 64 entries. We used the SimpleScalar simulator [2] and ran each benchmark for 100 million instructions. More details on the evaluation methodology and benchmarks used can be found in [1].

To estimate relative expended energy, we calculate the *activity ratio* for each approach by tracking the number and types of operations, and estimating the activity generated by each. For the two-level cache design, we track the number of L1 and L2 operations, and calculate the total number of cache banks activated considering the number activated for each operation and the L1/L2 configuration. We then take the ratio of the total activity for the CAP approach and for the conventional design. Because we use an exclusive caching policy, misses in the L1 Dcache to a valid location that hit in the L2 Dcache result in a swap between the two caches. In addition, the global miss ratio of the hierarchy remains constant due to the use of a random replacement policy. Thus, a CAP design that is optimized for performance in general also promotes energy efficiency by optimizing the amount of L1 Dcache allocated for a given application, and reducing L2 Dcache activity.

For the instruction queue, we did not have the ability to track the number of instruction queue accesses. However, because the rest of the design was almost ideal (large caches and queues, perfect branch prediction, plentiful functional units), we used the total number of cycles executed to approximate the relative number of instruction queue accesses for each benchmark and each approach (CAP and conventional). We then multiplied this number by the number of queue entries to get the total activity factor.

Although this approach is not exact, we believe that by tracking activity in this manner that we obtain a reasonable first-order approximation of relative expended energy.

6 Results

Table 1 shows event and activity counts as well as the activity ratio (CAP/conventional) for the ten benchmarks in which the CAPs configuration differs from the best conventional approach. The eleven benchmarks which use identical CAPs and conventional configurations are not shown as the expended energy is the same. The "L1 size (CAPs)" column denotes the CAPs configuration which performed best for each benchmark. The last column indicates the ratio of CAPs total activity to conventional total activity.

For five of the benchmarks, a CAPs configuration with an 8KB L1 Dcache outperforms the conventional approach using a 16KB L1 Dcache due to the former's faster cycle time, despite the increase in L1 misses and L1-L2 swaps incurred, as indicated in Table 1. Interestingly, the total activity count for the CAPs configuration is lower than the conventional approach. This is because for the conventional approach, twice as much L1 Dcache must be precharged and probed for each load operation (we make the simplifying assumption that only the selected way is activated on a store). This offsets the additional L2 probe and L1-L2 swap activity incurred with the CAPs configuration, and the fact that more L2 cache must be probed in the CAPs case on an L1 miss. This more efficient cache allocation reduces expended energy by 33% in the case of mgrid.

The tradeoff is different for the five remaining benchmarks where the CAPs L1 Dcache is larger than the conventional 16KB cache. Here, the reduction in L1 miss and L1-L2 swap activities must offset the additional L1 Dcache load activity for the CAPs configuration to expend less energy. This is true in all cases except for wave5, whose conventional cache experiences fewer total L1 misses than the other four benchmarks in this category. For benchmarks such as stereo and appcg whose requirements are not well-matched to the conventional organization, the energy savings with the CAP approach are significant: 44% for stereo and 62% for appcg. Because these benchmarks run significantly faster on the CAPs configuration as well [1], the reduction in the energy-delay product, an indicator of the efficiency with which a particular performance level is obtained, is even more striking: 70% for both benchmarks. Overall, 23% less energy is expended by the CAPs configuration for these benchmarks as a byproduct of performance optimization.

Table 2 shows the relevant data for the best-performing CAPs and conventional instruction queues for those benchmarks in which the CAPs and conventional configurations differ. Here, unused entries are disabled for the CAPs approach. In the cases in which the CAPs configuration performs best with fewer entries than the 64-entry conventional approach (due to the fact that the cycle time improvement overrides the IPC penalty incurred), this means that fewer elements are activated on each instruction queue access. However, more issue operations are required as the smaller window results in fewer instructions issued on average per issue operation. In all cases, the energy savings from acti-

benchmark	L1 size (CAP)	loads	stores	L1 misses		L2 misses	L1-L2 swaps		total activity		activity ratio
				conv	CAP		conv	CAP	conv	CAP	(CAP/conv)
m88ksim	8KB	64.6	35.4	2.48	2.97	2.41	2.43	2.48	370.5	261.6	0.71
compress	24KB	80.0	20.0	12.3	6.32	0.23	0.22	0.22	697.2	671.1	0.96
airshed	8KB	81.2	18.8	32.0	32.5	0.03	3.24	3.73	1275.0	1193.6	0.94
stereo	48KB	74.1	25.9	54.1	7.39	6.06	11.6	1.91	1910.2	1078.0	0.56
radar	8KB	61.2	38.8	1.51	4.20	0.50	0.87	3.54	328.8	295.5	0.90
appcg	64KB	4.84	95.2	12.0	0.49	0.47	11.9	0.49	474.6	181.9	0.38
swim	24KB	75.9	24.1	13.8	5.27	5.27	3.25	2.66	737.1	629.9	0.85
mgrid	8KB	95.4	4.60	4.55	4.64	4.12	1.45	1.45	511.7	344.9	0.67
applu	8KB	72.2	27.8	8.53	9.20	8.22	5.03	5.29	577.2	470.9	0.82
wave5	24KB	72.9	27.1	7.05	3.83	0.64	5.17	2.27	529.1	571.0	1.08
total									7411.3	5698.4	0.77

Table 1: Cache hierarchy event counts, total activity counts, and CAP/conventional activity ratio for each benchmark. All counts are in millions.

benchmark	IQ entries (CAP)	executed cycles		total activity		activity ratio
		conv	CAP	conv	CAP	(CAP/conv)
m88ksim	16	29.1	40.6	1862.6	649.1	0.35
compress	128	32.6	22.2	2088.9	2847.2	1.36
ijpeg	32	22.8	23.1	1461.9	738.0	0.50
airshed	32	23.7	25.0	1517.2	799.8	0.53
radar	16	110.1	141.8	7046.1	2268.9	0.32
appcg	16	48.4	49.8	3099.7	796.9	0.26
applu	128	27.6	19.8	1765.7	2535.0	1.44
fpppp	16	90.1	101.5	5766.3	1624.4	0.28
total				24608.5	12259.1	0.50

Table 2: Instruction queue executed cycles, total activity counts, and CAP/conventional activity ratio for each benchmark. All counts are in millions.

vating fewer elements overrides the energy cost of more queue accesses. This translates into as much as a 74% reduction in queue activity (in the case of appcg). Again, this benefit is achieved not through explicit power management, but simply as a byproduct of optimizing performance.

The opposite effect is observed for compress and applu which perform best with a larger 128-entry queue. The result is a significant increase in expended energy with the CAPs approach, despite the reduction in queue accesses. However, as these are the only two benchmarks using more entries than the conventional approach, the overall result is a 50% reduction in expended energy with the CAPs approach. Clearly, if more benchmarks performed best with more entries than the best average-performing configuration, then the energy savings would be less, perhaps significantly so. In addition, if configurations with more than 128 entries were available and some applications performed best with these configurations, then even a greater increase in expended energy would be incurred for these applications with the CAPs approach. However, it is likely that the inclusion of these applications into the benchmark suite would change the best-performing conventional approach to one with more entries. Thus, we expect that even with a wide range of benchmark behavior, that the CAPs approach will expend less energy overall due to its better use of hardware resources. However, the benefit is less clear with a structure in which elements are disabled than one in which the resources are allocated between primary and backup elements.

Examining Tables 1 and 2, we see that some applications, such as m88ksim, experience significant reductions in expended energy for both the CAP cache hierarchy and instruction queue. Thus, by applying the CAPs approach to other structures such as TLBs and branch predictors, we expect that a one to two order of magnitude reduction in expended energy is possible for applications whose hardware requirements are particularly poorly-matched to those in a conventional microprocessor design. A key point is that CAPs can achieve this improvement without adversely impacting the performance or expended energy of well-matched applications.

7 Conclusions and Future Work

The energy efficiency of a microprocessor is highly dependent on how well the hardware design matches the requirements of a particular application. In this paper, we have described how Complexity-Adaptive Processors inherently achieve energy efficiency as a byproduct of performance optimization by dynamically configuring their hardware organization to the problem at hand. By exploiting repeater methodologies used increasingly in deep submicron designs, CAPs achieve this benefit with little or no cycle time degradation over a conventional approach. By examining our performance results for a number of benchmarks, we found that the CAPs design reduced overall expended energy by 23% for cache hierarchies and 50% for instruction queues. We also discovered an additive effect for some benchmarks indicating that a much greater impact can be realized by applying the CAPs approach in concert to a number of hardware structures. In the future, we plan on obtaining results using more precise energy models for these and other hardware structures.

References

[1] D.H. Albonesi. Dynamic IPC/clock rate optimization. *Proceedings of the 25th International Symposium on Computer Architecture*, June 1998.

[2] D. Burger and T.M. Austin. The simplescalar toolset, version 2.0. Technical Report TR-97-1342, University of Wisconsin-Madison, June 1997.

[3] J. Fleischman. Private communication. November 1997.

[4] D. Matzke. Will physical scalability sabotage performance gains? *IEEE Computer*, 30(9):37–39, September 1997.

[5] K.B. Normoyle et al. UltraSPARC-IIi: Expanding the boundaries of a system on a chip. *IEEE Micro*, 18(2):14–24, March 1998.