Dynamic Data Cache Resizing: Study of Energy Efficient Cache Architecture

Yongqin Gao  Xiaolong Li

Abstract
With the fast increase of the transistors these years, the power consumption of the IC chip also increased dramatically. High-performance and low-power computing research is trying to find a method to decrease the power consumption without significantly downgrading the performance. Our study is using dynamic data cache resizing techniques to tackle this problem. Dynamic resizing method can use the runtime information to adjust the cache size by shutting down part of the cache or activating part of the cache. By using this method, part of the cache will be shut down when it is not necessary to keep active with regard to performance. Eventually we can save power by this method.

1 Problem Definition and Importance

Ever since 1965, Moore’s Law [2] has accurately predicted that the number of transistors on a single IC would double every 18 months. With the technology advance, a recent microprocessor can house a huge number of transistors on a complex integrated circuit (IC) chip. Increased transistor density has increased operating speeds, but also increased power consumption at even higher rate. This increased power consumption generates undesired heat, which potentially degrades performance, destroys the IC, or injures the user. Therefore current computer architects have to consider both power consumption and performance when designing a complex IC.

Shutting down parts of the microprocessor would be the easiest, most effective mechanism to conserve power. Many general-purpose processors utilize this technique. Because the caching structures on microprocessors use a large percentage of the transistors, shutting down parts of the cache would save a considerable amount of power. However, the size of the cache greatly affects performance, or the time needed to execute programs. The optimal high-performance, low power cache configuration will be a balance between energy consumption and execution time.
Practically, the runtime behavior of any program is changing. Thus there is no global optimal configuration for any running of program. So the best approach is dynamically change the cache size according to runtime program behaviors. In this project, we will simulate the dynamic cache resizing to evaluate this technique.

The rest of the paper will be structured as follows. Section 2 will be the problem definition and importance. Section 3 will be the related works. From Section 4 on, we will discuss our research. Section 4 will be the proposed approach. Section 5 will be the explanation of the experiment method. Section 6 will be the results, alone with some discussion. Section 7 will be the conclusion and future work.

2 Related Works

There are some previous works on the power efficient cache design.

The early researches are focused on static cache resizing. Abella and Gonzalez did a survey on power efficient data cache design [7]. This paper studies different cache organizations that reduce significantly dynamic and static power dissipation with a small performance loss. This study tries to guide processor designers to choose the cache organization with best trade-off between efficiency and power dissipation. This study shows how different L1 multi-banked data cache organizations can obtain similar performance to that where a monolithic cache is used, requiring at the same time less dynamic and static power and enabling to reduce the cache access time. It has been shown that different technology parameters can be combined to obtain high performance caches with small power requirements. But all the cache organizations this paper studied are static resizing cache. Static cache resizing methods are simple and easy to implement – no extra modification needed for hardware and OS, but without runtime information, the preset cache size would already result in wasting some of the power or downgrading the performance.

Dynamic cache resizing is a better method than static method since its usage of runtime information. Zhang, Vahid and Najjar proposed a configurable cache structure [4]. This cache structure is designed for embedded systems. Since embedded systems always have very narrow applications. So the requirement for cache is limited and clear. This cache structure is can be configured by software to save unnecessary power. This kind of passive
cache structure (configurable by software) is good for embedded systems but not good for general purpose IC chip, since the great variety of applications on general purpose IC chip. Also this design need the awareness of hardware for the software developers, which is also a problem for general purpose IC chip.

Active dynamic resizing cache is a good choice for general purpose IC chip. Zhou, Toburen and Conte proposed AMC (Adaptive Mode Control) [3]. AMC works by selectively powering down unused cache lines in the data store, thus saving the leakage power. Unlike other approaches, the tag store information is always maintained so that the impact of powering down data store cache lines on performance can be continually monitored. Two performance indicators are monitored throughout execution. One is the ideal miss rate resulting from tag mismatches. The other is the sleep miss rate, which is a miss due to the cache line in data store has been in sleep mode although the tag information matches. AMC monitors the relationship between these two miss rates during the course of execution and determines how the turn-off interval should be modified. The AMC method has several disadvantages as follows:

1. Simple model: In the AMC, they use miss rate as the only threshold to control the shutoff/activate action of the cache lines. This can cause the thrashing problem, where the system will continue to change the configuration.

2. Active tag: In the AMC, the tag fields of all the cache blocks are all active all the time. This is because the system need to use these tag fields to calculate the miss rate on the fly.

3. Lose of computing power: In the AMC, they need to calculate the miss rate on the fly. Miss rate is not simple, so it may take some of the computing power of the processor to do the work.

Brooks, Tiwari and Martonosi proposed Wattch [1]. Wattch introduced a power analysis model which is interfaced SimpleScalar. The Wattch toolkit generates both performance and power consumption statistics. It uses the same simulators as SimpleScalar, but adds a power analysis module when simulating computer architectures. In our study, we will use software simulation to compare our model with the fixed cache model. We choose Wattch as the simulation toolkits.
Kalla, Hu and Henkel proposed LRU-SEQ [5]. LRU-SEQ is a novel replacement policy for transition energy reduction in instruction caches. Redirecting sequential cache fills to the last bank accessed by regrouping sequential accesses in such a manner, reduces inter-bank transitions and increases the chances that a bank can be shut for a longer period and thus meet the requirements of the break-even time. This is another direction in the low-power and high-performance computing research. We may combine this with our model, which we will talk in the discussion section.

3 Our Approach

In our study we will propose an active dynamic data cache resizing model. This model will use the similar idea of AMC, so it will outperform the static model and passive dynamic model in general purpose IC chip. Also we will use different schema to overcome the disadvantages in AMC.

The major parameters of a cache design include block size, number of set and associativity. Dynamically updating these parameters can all change the size of the cache and then lead to balance the cache performance and energy consumption. But in practically, varying block size or number of set are fairly difficult problems. Since block size and set of number will affect the number of necessary tag bits, for example increasing the block size decreases the number of tag bits while decreasing the block size increase the number of tag bits [6]. Varying block size or number of set will increase the complexity of the tag comparison module due to the varying tag length. So we will not consider varying block size or number of set in our experiments.

Our model is adapted from the thesis by Spanberger [8]. We implement the similar model with different statistics on the data cache. Same as the Spanberger study, we just use varying associativity to change the cache size. But our study is different from the Spanberger model in the sense that

1. Our study is in the data cache instead of instruction cache in the Spanberger study. Instruction cache is much more sequential than the data cache due to the continuous property of the instructions. So we want to know if the same idea is good for data cache or not.
2. We also change the statistics to control the behaviors of the model.

The cache resizing model represented as a state model as in Figure 1. There are three states in the model: normal, large and small. In ‘normal’ state, the cache is working like a fixed size cache. In ‘large’ state, the cache will compare the performance of bigger size cache with the current size cache. And in ‘small’ state, the cache will compare the performance of smaller size cache with the current size cache. There are 3 physical structures and 4 thresholds used to control the transitions among modes. The tournament access counter will count the total access to the overall cache. The tournament hit counter will count the total access to the tournament part of the cache (extra way for large mode and LRU way for small mode). The miss saturation counter will count the consecutive miss in the overall cache. Hit-to-win, access length, tournament length and miss saturation are the four thresholds which are preset and used to control the transitions among the states.

![Figure 1: Cache resizing model](image)

The detailed control inequities for the transitions are listed in Table 3.

So in our study, we will simulate the fixed cache model and our model with same benchmarks and simulation configurations to check whether our model will be better than the fixed cache model. Also we will use different threshold values to study the inference of different statistics on the final results.
<table>
<thead>
<tr>
<th>Transition</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Tournament access counter &gt; access length</td>
</tr>
<tr>
<td>B</td>
<td>Tournament hit counter &gt; hit-to-win</td>
</tr>
<tr>
<td>C</td>
<td>Miss saturation counter &gt; miss saturation</td>
</tr>
<tr>
<td>D</td>
<td>Tournament access counter &gt; tournament length</td>
</tr>
<tr>
<td>E</td>
<td>Tournament access counter &gt; tournament length</td>
</tr>
<tr>
<td>F</td>
<td>Tournament hit counter &gt; hit-to-win</td>
</tr>
</tbody>
</table>

Table 1: Transition Conditions

4 Experiment Method

We will use Wattch, which is build upon the SimpleScalar toolkits, as our simulation toolkits. In order to simulate our model, we will change some of the code in the Wattch. One major change is adding more new structure in the code to simulate the extra physical structure we need to monitor the runtime information. The other change is adding a new cache replacement policy in the code as the entry point to use our new dynamically reconfigurable data cache.

According to our cache resizing model, the purpose of the simulations is to see how the model works and how the varying of these parameters affect the delay-power product. The Wattch toolkits are used to build the simulations. The cache configurations of these simulations are same, which is 8 KB with eight-way associative tournament cache and 32 bytes block size. Each simulation executed 20,000,000 instructions. Our method was that we will change only one parameter while setting the other parameters at fixed value. Then we will collect the results to study the effect of these varying parameters to the delay-power product.

The benchmarks we are using in the study is GCC, COMPRESS and GO. We select these three benchmarks because they have different behaviors. COMPRESS is IO-bounded applications, all the data accessed is sequential and it also has minimal loops. GO is AI applications, so it is CPU-intensive applications with large amount of loops. GCC is also CPU-intensive benchmarks, but it is a balance mixture of loop and sequential access, which represent the majority of real applications.
5 Results and Discussion

In this section, we will represent the results with some discussions. For convenience to compare the simulation results with fixed cache size structure, we normalized the simulation data and used the simulation result of fixed size cache as baseline.

First, we will discuss the result with varying tournament length. The result is shown in Figure 2. The definition of $X$ coordinate is the the benchmarks we are running and the $Y$ coordinate is the delay-power product value. In each benchmark, we have four series of data, represented by different color. According to the legend, '0' represents the fixed size cache and the others represent different tournament lengths. In our simulation, when tournament length was 8192, it produced the lowest normalized delay-power (DP) product for GCC. But for GO, the lowest DP product existed when it was 4096. This means the performance is not uniformed in different benchmarks. And we can find in the figure that all of DP products are less than one which means that they still outperformed conventional caches in regards to delay-power. One exception is the COMPRESS benchmark. We can see varying tournament length has almost no effect on compress benchmark, this is because compress benchmark is I/O bounded, tournament cache won’t do any good to it. we will see the same result for COMPRESS in the other simulations.

Second, we will discuss the result with varying access length. The result is shown in Figure 3. The definition of $X$ and $Y$ coordinates are similar to previous figure and '0' series represent the fixed size cache also. Access length is corresponds to the number of cache accesses that must occur before the cache will change to small state. Because of this, this number controls how aggressively the cache will get smaller. With a small accesses length, the tournament cache will shrink too fast so that it may cause more misses and thus it will have more miss penalty and waste cycles and energy. On the other hand, a large access length causes the cache shrink too slow. In this case, parts of the cache are merely consumed power without improving performance and waste energy. In our simulation, we can see that access length of 65536 will generate least delay-power product than others.

Then we will discuss the result with varying miss saturation. The result is shown in Figure 4. The definition of $X$ and $Y$ coordinates are same to previous figure and '0' series represent the fixed size cache also. Miss saturation is corresponds to the number of consec-
utive misses. Unlike the accesses length, this threshold controls how aggressively the cache gets larger. In our simulation, this threshold has different effect on different benchmarks. In the figure, we can see that varying miss saturation has no effect on COMPRESS. For GCC, when it was 2, the model got the best delay-power product. When increased it to 4, we got a even worse delay-power product than baseline. But for GO, 4 is the best option. That also means the improvement of tournament data cache depends on benchmark’s behaviors also.

Now we will discuss the result with varying hit-to-win. The result is shown in Figure 5. The definition of $X$ and $Y$ coordinates are similar to previous figure and ‘0’ series represent the fixed size cache also. Hit-to-win is corresponds to the number of the hits to the cache block that will be shutdown in small state or activated in large state. In small state, even if some cache access hit to the block waiting to be shutdown, but as long as the number of hit less than hit-to-win, that block will not be shutdown. And in large mode, even if some cache access hit to the block waiting to be activated, but as long as the number of hit less than hit-to-win, that block still will not be activated. So In some sense, it is a miss tolerance before reconfiguring the cache. In our simulation, when hit-to-win was 4, the model got the best delay-power product. But it does not mean the higher the hit-to-win, the better the
performance in terms of delay-power product. We need to run more simulation to find the best configuration.

We may notice that, there is not much difference between the normalized delay-power product produced in simulations. That does not mean we only can get limited benefit from tournament cache model, because we use total power and total delay to calculate delay-power product and we only run 20,000,000 instructions. It is obvious that the longer time it runs, the more benefits we can get.

Finally we will show the tournament history of one of the run. The result is shown in Figure 6. It is one of the tournament history of the simulations. In some sense, this reveals the benchmark’s behavior. In this figure, we can see our cache model is effective, it dynamically changed data cache size to follow the benchmark’s behavior. We think we may be able to use this information as prediction of application’s behavior to optimize the performance of tournament cache. This interesting work will be discussed in future work later.

Figure 3: Comparison with Varying Access Length
6 Conclusion and Further work

From our simulation, we can conclude that dynamic data cache resizing reduced power consumption by shutting down parts of the cache without degrading performance. Our model is effective although its performance depends on the application’s behavior.

Future work: Actually, our work concentrates on two part: one is when to reconfigure and another is how to reconfigure. We believe that there still are lots of interesting work to do about tournament cache. In fact we had prepared to implement some of following future work, however, due to the tight schedule, we only did a small part of our original plan. The future work will probably focus on the following topics.

- In this project, due to the tight schedule, we only ran short simulations and tested how the parameter respectively affect the delay-power product. In future, we should run longer simulation and use the combination of each single optimal value of those parameters to see how they affect delay-power product interactively.

- A predictor of application’s behavior can be implemented based on the tournament cache. We have collected the tournament history, we believe it can tell me more
interesting things about application’s behavior. If we could use these information to predict the application’s future behavior.

- It will be better to adjust these threshold dynamically during the runtime. By dynamically adjust the value of each threshold according to the predictor of application’s behavior, we can improve delay-power product. If the predictor is not available, we still could work on this based on tournament history.

- In our model, the change of cache size is linear. In future, we may use heuristic strategy to decide how many cache we should change. we may use quicker decrease, slower increase or slower decrease and quicker increase or dynamically change. we believe that it should an interesting work and could improve performance more.

- In our project, we only consider dynamically change associativity. In future, we may change the block size or the sets of number simultaneously. Once we have a predictor of demand to cache, we may do more crazy things. When the cache can not be larger, we may according to the prediction of demand to cache, dynamically change block size or the number of cache and associativity or even all of them to get higher performance.
We believe this is most interesting and exciting part but it may cause much more complexity to hardware design.

References


