

ABSTRACT

The memory hierarchy of high performance and embedded processors has been shown to be one of the major energy consumers. Extrapolating the current trends, this portion is likely to be increased in the near future. In this paper, a technique is proposed which uses an additional mini cache, called the L0-cache, located between the I-cache and the CPU core. This mechanism can provide the instruction stream to the data path, and when managed properly, it can efficiently eliminate the need for high utilization of the more expensive I-cache.

Five techniques are proposed and evaluated which are used to the dynamic analysis of the program instruction access behavior and to proactively guide the L0-cache. The basic idea is that only the most frequently executed portion of the code should be stored in the L0-cache, since this is where the program spends most of its time.

Results of the experiments indicate that more than 60% of the dissipated energy in the I-cache subsystem can be saved.

INTRODUCTION

In recent years, power dissipation has become one of the major design concerns for the microprocessor industry. The shrinking device size and the large number of devices packed in a chip die coupled with large operating frequencies, have led to unacceptably high levels of power dissipation. The problem of wasted power caused by unnecessary activities in various parts of the CPU during code execution has traditionally been ignored in code optimization and architectural design.

Higher frequencies and large transistor counts more than offset the lower voltages and the smaller the devices and they result in large power consumption in a newest version in a processor family.

POWER TRENDS FOR CURRENT MICROPROCESSORS

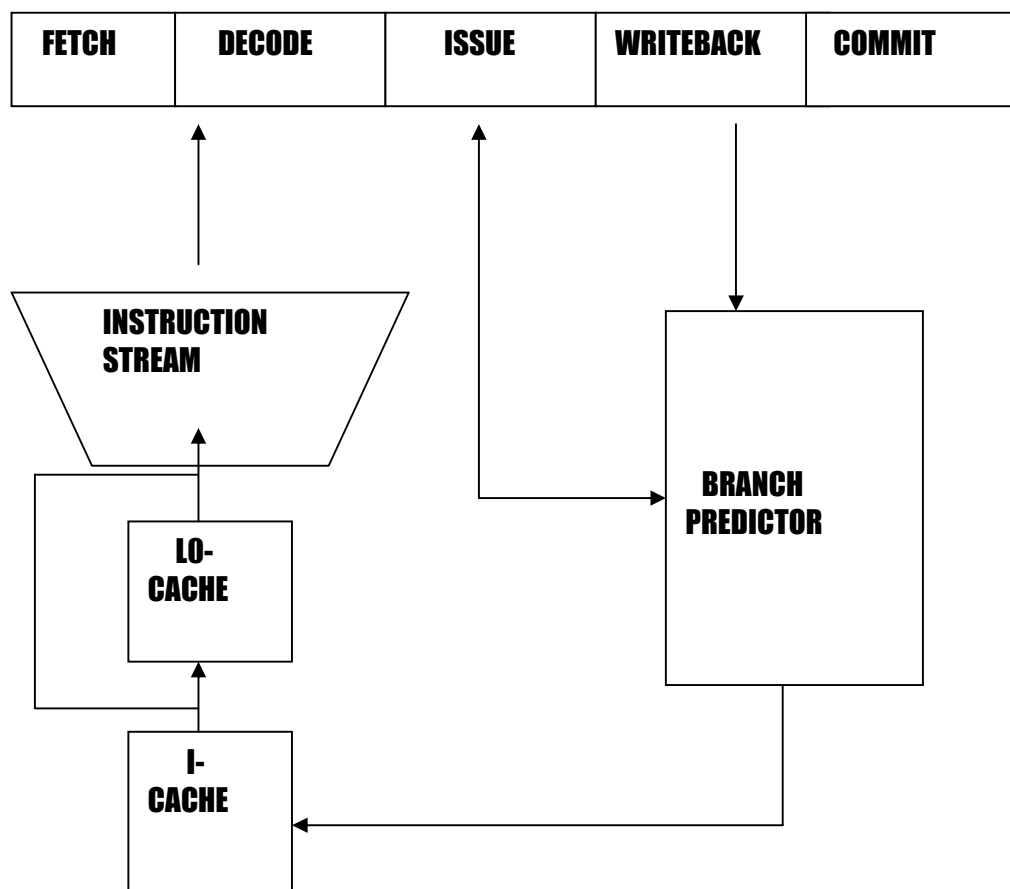
	DEC 21164	DEC 21164 High Freq	Pentium Pro	Pentium II
Freq (Mhz)	433	600	200	300
Power (W)	32.5	45	28.1	41.4

Very often the memory hierarchy access latencies dominate the execution time of the program, the very high utilization of the instruction memory hierarchy entails high energy demands on the on chip I-cache subsystem. In order to reduce the effective energy dissipation per instruction access, the addition of an extra cache is proposed, which serves as the primary cache of the processor, and is used to store the most frequently executed portion of the code.

WORKING WITH THE L0 CACHE

Some dynamic techniques are used to manage the L0-cache. The problem that the dynamic techniques seek to solve is how to select the basic blocks to be stored in the L0-cache while the program is being executed. If a block is selected, the CPU will access the L0-cache first; otherwise, it will go directly to the I-cache and bypass the L0-cache. In the case of an L0-cache miss, the CPU is directed to fetch instructions from the I-cache and to transfer the instructions from the I-cache to the L0-cache.

PIPELINE MICRO ARCHITECTURE



BRANCH PREDICTION & CONFIDENCE ESTIMATION – A BRIEF OVERVIEW

Branch Prediction

Branch prediction is an important technique to increase parallelism in the CPU, by predicting the outcome of a conditional branch instruction as soon as it is decoded. Successful branch prediction mechanisms take advantage of the non-random nature of branch behavior. Most branches are either mostly taken in the course of program execution.

The commonly used branch predictors are:

1. **Bimodal branch predictor.**

Bimodal branch predictor uses a counter for determining the prediction. Each time a branch is taken, the counter is incremented by one, and each time it falls through, it is decremented by one. Looking onto the value of the counter does the prediction. If it is less than a threshold value, the branch is predicted as not taken; otherwise, it is predicted as taken.

2. **Global branch predictor**

Global branch predictor considers the past behavior of the current branch as well as the other branches to predict the behavior of the current branch.

3. **Confidence Estimation**

The relatively new concept confidence estimation has been introduced to keep track of the quality of branch prediction. Confidence estimators are hardware mechanisms that are accessed in parallel with the branch predictors when a branch is decoded, and they are modified when the branch is resolved. They characterize a branch

as ‘high confidence’ or ‘low confidence’ depending upon the branch predictor for the particular branch. If the branch predictor predicted a branch correctly most of the time, the confidence estimator would designate the prediction as ‘high confidence’ otherwise as ‘low confidence’.

BASIC IDEA OF THE DYNAMIC MANAGEMENT

SCHEME

The dynamic scheme for the L0-cache should be able to select the most frequently executed basic blocks for placement in the L0-cache. It should also rely on existing mechanisms without much hardware investment if it to be attractive for energy reduction.

The branch prediction in conjunction with confidence estimation is a reliable solution to this problem.

Unusual Behavior of the Branches

A branch that was predicted ‘taken’ with ‘high confidence’ will be expected to be taken during program execution. If it not taken, it will be assumed to be behaving ‘unusually’.

The basic idea is that, if a branch behaves ‘unusually’, the dynamic scheme disables the L0-cache access for the subsequent basic blocks. Under this scheme, only basic blocks that are to be executed frequently tend to make it to the L0-cache, hence avoiding cache pollution problems in the L0-cache.

DYNAMIC TECHNIQUES FOR L0-CACHE

MANAGEMENT

The dynamic techniques discussed in the subsequent portions select the basic blocks to be placed in the L0-cache. There are five techniques for the management of L0-cache.

- 1. Simple Method.**
- 2. Static Method.**
- 3. Dynamic Confidence Estimation Method.**
- 4. Restrictive Dynamic Confidence Estimation Method.**
- 5. Dynamic Distance Estimation Method.**

Different dynamic techniques trade off energy reduction with performance degradation.

SIMPLE METHOD

The confidence estimation mechanism is not used in simple method. The branch predictor can be used as a stand-alone mechanism to provide insight on which portions of the code are frequently executed and which is not. A mispredicted branch is assumed to drive the thread of execution to an infrequently executed part of the program.

The strategy used for selecting the basic blocks is as follows:

If a branch predictor is mispredicted, the machine will access the I-cache to fetch the instructions. If a branch is predicted correctly, the machine will access the L0-cache.

In a misprediction, the pipeline will flush and the machine will start fetching the instructions from the correct address by accessing the I-cache.

DELAY RESULTS

	256B		512B	
Bench mark	8B	16B	8B	16B
Tomcatv	0.185	0.177	0.100	0.121
Go	0.609	0.509	0.572	0.488
Li	0.453	0.363	0.403	0.322

ENERGY RESULTS

	256B		512B	
Bench mark	8B	16B	8B	16B
Tomcatv	1.050	1.032	1.011	1.006
Go	1.159	1.091	1.138	1.079
Li	1.189	1.118	1.159	1.093

The energy dissipation and the execution time of the original configuration that uses no L0-cache is taken as unity, and normalize everything with respect to that.

STATIC METHOD

The selection criteria adopted for the basic blocks is:

- ❑ If a 'high confidence' branch was predicted incorrectly, the I-cache is accessed for the subsequent basic blocks.
- ❑ If more than n low confidence branches have been decoded in a row, the I-cache is accessed.

Therefore the L0-cache will be bypassed when either of the two conditions are satisfied. In any other case the machine will access the L0-cache.

The first rule for accessing the I-cache is due to the fact that a mispredicted 'high confidence' branch behaves 'unusually' and drives the program to an infrequently executed portion of the code. The second rule is due to the fact that a series of 'low confidence' branches will also suffer from the same problem since the probability that they all are predicted correctly is low.

DELAY RESULTS

	256B		512B	
Bench mark	8B	16B	8B	16B
Tomcatv	1.049	1.025	1.011	1.006
Go	1.090	1.052	1.078	1.045
Li	1.198	1.124	1.169	1.099

ENERGY RESULTS

	256B		512B	
	256B		512B	
Bench mark	8B	16B	8B	16B
Tomcatv	0.185	0.163	0.100	0.120
Go	0.757	0.702	0.736	0.690
Li	0.431	0.337	0.382	0.296

DYNAMIC CONFIDENCE ESTIMATION METHOD

Dynamic confidence estimation method is a dynamic version of the static method. The confidence of each branch is determined dynamically. The management of the cache subsystem is identical to the static method.

The I-cache is accessed if

- A high confidence branch is mispredicted.
- More than n successive ‘low confidence’ branches are encountered.

The dynamic confidence estimation mechanism is slightly better in terms of energy reduction than in the simple or static method. Since the confidence estimator can adapt dynamically to the temporalities of the code, it is more accurate in characterizing a branch and, then, regulating the access of the L0-cache.

DELAY RESULTS

	256B		512B	
Bench mark	8B	16B	8B	16B
Tomcatv	1.046	1.029	1.008	1.006
Go	1.149	1.085	1.130	1.074
Li	1.194	1.122	1.164	1.097

ENERGY RESULTS

	256B		512B	
Bench mark	8B	16B	8B	16B
Tomcatv	0.181	0.174	0.096	0.119
Go	0.642	0.548	0.609	0.529
Li	0.435	0.344	0.386	0.303

RESTRICTIVE DYNAMIC CONFIDENCE ESTIMATION METHOD

The methods described in previous sections tend to place a large number of basic blocks in the L0-cache, thus degrading performance. Restrictive dynamic scheme is a more selective scheme in which only the really important basic blocks would be selected for the L0-cache.

The selection mechanism is slightly modified as:

- The L0-cache is accessed only if a ‘high confidence’ branch is predicted correctly. The I-cache is accessed in any other case.

This method selects some of the most frequently executed basic blocks, yet it misses some others. It has much lower performance degradation, at the expense of lower energy reduction. It is probably preferable in a system where performance is more important than energy.

DELAY RESULTS

	256B		512B	
Bench mark	8B	16B	8B	16B
Tomcatv	1.046	1.024	1.009	1.006
Go	1.073	1.044	1.063	1.038
Li	1.149	1.092	1.123	1.073

ENERGY RESULTS

	256B		512B	
Bench mark	8B	16B	8B	16B
Tomcatv	0.202	0.183	0.119	0.141
Go	0.800	0.758	0.783	0.748
Li	0.601	0.529	0.560	0.498

DYNAMIC DISTANCE ESTIMATION METHOD

The dynamic distance estimation method is based on the fact that, a mispredicted branch triggers a series of successive mispredicted branches. The method works as follows:

All n branches after a mispredicted branch are tagged as 'low confidence' otherwise as 'high confidence'. The basic blocks after a 'low confidence' branch are fetched from the L0-cache. The net effect is that a branch misprediction causes a series of fetches from the I-cache.

A counter is used to measure the distance of a branch from the previous mispredicted branch. This scheme is also very selective in storing instructions from the L0-cache, even more than the restrictive dynamic estimation method.

DELAY RESULTS

	256B		512B	
Bench mark	8B	16B	8B	16B
Tomcatv	1.045	1.029	1.008	1.005
Go	1.064	1.037	1.056	1.033
Li	1.154	1.095	1.132	1.077

E NERGY RESULTS

	256B		512B	
Bench mark	8B	16B	8B	16B
Tomcatv	0.197	0.192	0.115	0.137
Go	0.820	0.781	0.806	0.773
Li	0.577	0.502	0.540	0.472

COMPARISON OF DYNAMIC TECHNIQUES

The energy reduction and delay increase is a function of the algorithm used for the regulation of the L0-cache access, the size of the L0-cache, its block size and its associability. For example, a larger block size causes a larger hit ratio in the L0-cache. This results into smaller performance overhead, and bigger efficiency since the I-cache does not need to be accessed so often.

On the other hand if the block size increase does not have a large impact on the hit ratio, the energy dissipation may go up, since a cache with a larger block size is less energy efficient than a cache with the same size but smaller block size.

The static method and dynamic confidence method make the assumption:

The less frequently executed basic blocks usually follow less predictable branches that are mispredicted.

The simple method and restrictive dynamic confidence estimation method address the problem from another angle. They make the assumption:

Most frequently executed basic blocks usually follow high predictable branches. The dynamic distance estimation method is the most successful in reducing the performance overhead, but the least successful in energy reduction.

This method possesses stricter requirement for a basic block to be selected for the L0-cache than the original dynamic confidence estimation method.

Larger block size and associability will have a beneficial effect on both energy and performance. The hit rate of a small cache is more sensitive to the variation of the block size and the associability.

CONCLUSION

This paper presents the method for dynamic selection of basic blocks for placement in the L0-cache. It explains the functionality of the branch prediction and the confidence estimation mechanisms in high performance processors.

Finally five different dynamic techniques were discussed for the selection of the basic blocks. These techniques try to capture the execution profile of the basic blocks by using the branch statistics that are gathered by the branch predictor.

The experiment evaluation demonstrates the applicability of the dynamic techniques for the management of the L0-cache. Different techniques can trade off energy with delay by regulating the way the L0-cache is accessed.