An Interpolative Analytical Cache Model with Application to Performance-Power Design Space Exploration

Bin Peng[1], Weng Fai Wong[1], Yong Chiang Tay[2]
[1] Singapore-MIT Alliance, National University of Singapore
[2] Department of Mathematics, National University of Singapore

***Abstract*— Caches are known to consume up to half of all system power in embedded processors. Co-optimizing performance and power consumption of the cache subsystem is therefore an important step in the design of embedded systems, especially those employing application specific instruction processors. One of the main difficulty in such attempts is that cache behaviors are application as well as cache-structure specific. The two general approaches to tackling this problem is to either use exhaustive simulation or analytical modeling. The former is too time-consuming while the latter often lacks the required accuracy. In this paper, we introduce a novel third approach. We propose an analytical cache model that *interpolates* data from runs with direct mapped and fully associative caches so that the entire parameter space involving all set-associativity is spanned. Validation against full trace-driven simulations shows that our model has a very high degree of fidelity, but requires significantly less simulation time than exhaustive simulation. Furthermore, the model works for instruction, data or unified caches. Finally, we show how the model can be coupled with a power model for caches so that one can very quickly decide on pareto-optimal performance-power design points for rapid design space exploration.**

***Index Terms*— Cache, Analytical Model, Performance, Power, Simulation.**

I. Introduction

It is well-known that processor caches significantly improve the performance of applications. However, caches are major consumers of power, taking up to 50% of system power [9]. For embedded systems in which power is a significant constraint, optimizing the cache parameters so as to obtain the best performance-power tradeoff is therefore an important problem. In order to do this, it is necessary first to have an accurate model of what happens in the cache as the application executes. Unfortunately, this is non-trivial because such an interaction is dependent on the application as well as the cache's structure.

There are generally two ways of overcoming the above problem, namely exhaustive simulation or by means of an analytical cache model. Exhaustive simulation requires the application to be simulated over potentially all possible cache structures. This is very time consuming. Analytical models generally use some features of the application to predict their performance on various cache structures. The main problem with this approach is that analytical models proposed thus far are not accurate enough to be used widely.

In this paper, we propose a third approach. First, the application is simulated over a specific number of cache configurations, namely that for direct mapped and fully associative caches. We then introduce an analytical model for *interpolating* these data points so as to span all set-associativity. For the SPEC benchmarks, we found that our model is typically able to achieve a coefficient of determination above 0.9 with respect to full simulation. We believe this degree of accuracy far exceeds that of any analytical model found in the literature.

By coupling our cache performance model with a well-known cache power model, we have a means of very quickly exploring the entire space of cache parameters in order to obtain pareto-optimal design points of interest.

This paper is organized as follows. In Section II we shall first introduce our cache performance model, showing its derivation from first principles. Section III discusses the validation of our model against full simulation. In Section IV, we show how the performance model can be coupled with the CACTI cache power model [18] and how the combined model can be easily used for design space exploration. We will compare our work with that representative published works in Section V. This is followed by a conclusion.

II. Model

In the following, *cache* may refer to instruction, data or unified cache. Let C be the size of a set associative cache.

The cache is divided into *blocks* or lines; let B be the block size. These blocks are grouped into *sets* of equal size. Let A be the size of a set, i.e. the degree of *associativity*; hence, the cache is said to be A-way set associative. Let S be the number of sets. It follows that $C = ABS$.

The cache is *direct-mapped* if $A = 1$; it is *fully-associative* if $S = 1$.

We can number the blocks, and model references to cache by a *trace* that is a sequence of block numbers, like in Fig. 1.

We then adopt Quong's definition of a *gap* [11]: namely, the number of different references between consecutive references

$$\ldots, X, 3, 3, 1, 4, 1, 2, 4, 3, 1, X, \ldots$$

Fig. 1. **In this fragment of a trace, the gap between references to X is 4.**

to the same block. For example, in Fig. 1, the gap between the two references to X is 4.

A. Direct mapped cache ($A = 1$)

For a direct-mapped cache, we start by assuming (like Quong did for the instruction cache) that references are independently and uniformly distributed. Consider a block X in cache. Then, by uniformity,

$$\text{Prob(next reference does not replace } X) = 1 - \frac{1}{S}.$$

By independence,

$$\text{Prob(next } g \text{ references do not replace } X) = \big(1 - \frac{1}{S}\big)^g. \quad (1)$$

To extend Eqn. 1 to all blocks, let g now refer to the *average* gap over all blocks. We then get

$$\text{Prob(a repeated reference is a miss)} = 1 - \big(1 - \frac{1}{S}\big)^g. \quad (2)$$

In reality, references have spatial and temporal locality, so are neither independent nor uniformly distributed. For example, sequential references can increase the miss probability — an effect similar to decreasing S in Eqn. 2; on the other hand, a loop can reduce the miss probability — similar to increasing S in Eqn. 2

We therefore model locality by first introducing a parameter f, thus:

$$\text{Prob(a repeated reference is a miss)} = 1 - \big(1 - \frac{1}{S^f}\big)^g. \quad (3)$$

In other words, the model assumes that locality is equivalent to uniform access plus a change in S to S^f.

With Eqn. 3, a repeated reference will miss with probability 1 if $S = 1$ (i.e. the direct mapped cache has just 1 block). This may not be so, since it is possible that consecutive references are to the same block. To model this, we introduce another parameter P_1 to Eqn. 3:

$$\text{Prob(a repeated reference is a miss)} = 1 - \big(1 - \frac{P_1}{S^f}\big)^g; \quad (4)$$

i.e. P_1 is the miss probability if $S = 1$.

When a block is referenced for the first time, it is not a repeated reference, and it causes a *cold* or *compulsory* miss. Let α =Prob(cold miss). From Eqn. 4, we now get miss probability

$$P = \alpha + (1 - \alpha)(1 - \big(1 - \frac{P_1}{S^f}\big)^g). \quad (5)$$

For a trace and a cache configuration, one can measure α, P_1 and, in principle, g. However, it took days for us to measure the average gap in the first 8 million references of the `gzip` trace in the SPEC2000 benchmark (the full trace is about 10 times longer). We therefore determine g through regression, by using the following equivalent form of Eqn. 5:

$$y = fx + d \quad (6)$$

where

$$y = \log\big(1 - \big(\frac{1 - P}{1 - \alpha}\big)^{\frac{1}{g}}\big), x = \log S \text{ and } d = \log P_1.$$

We fit a given set of (S, P) data with the model through an iteration, with g initialized to 0.1 and incremented by 0.001; for each g, we substitute the measured α to get a (x, y) data set, call `MATLAB` to obtain the regression constants f, d and coefficient of determination R^2, then select the g that maximizes R^2.

Fig. 2 shows the result of this fitting process for the `gcc` trace in SPEC2000, when it is simulated with direct mapped data, instruction and unified caches. The fit for data and unified caches is excellent; the fit for instruction cache is good too, considering that no effort is made to model the looping behavior, the function calls, etc.

Note that, in the fitting process, we do not use the measured P_1 to fix d. Rather, we obtain d from the regression and use it to calculate P_1. Fig. 2 shows that the calculated value is close to the measured P_1 in all three cases.

B. Fully associative cache ($S = 1$)

Our model for a fully associative cache is similar to the one above for a direct mapped cache. Now, we assume first that block replacement is random when there is a miss, so (like Eqn. 2)

$$\text{Prob(a repeated reference is a miss)} = 1 - \big(1 - \frac{1}{A}\big)^g.$$

We then introduce another parameter h to model reference locality and replacement policy (least-recently used, etc.). Specifically, we assume that the effect of locality and replacement policy is equivalent to that of random replacement plus a change in A to A^h; i.e.

$$\text{Prob(a repeated reference is a miss)} = 1 - \big(1 - \frac{1}{A^h}\big)^g.$$

After refining this equation with P_1 (for $A = 1$) and factoring in cold misses, we get

$$P = \alpha + (1 - \alpha)(1 - \big(1 - \frac{P_1}{A^h}\big)^g). \quad (7)$$

To test whether this fully associative model gives a good fit for simulation data, we use regression like in the case for the direct map model. Fig. 3 shows that the fit is excellent for `gcc`, but not as good for `gzip`.

C. Set associative cache (general A and S)

For set associative caches, we look for a generalization of the equation for P that reduces to Eqn. 5 when $A = 1$ and to Eqn. 7 when $S = 1$. The simplest possibility is

$$P = \alpha + (1 - \alpha)(1 - \big(1 - \frac{P_1}{S^f A^h}\big)^g). \quad (8)$$

Again, we can rewrite Eqn. 8 as

$$y = fx_1 + hx_2 + d \quad (9)$$

where

$$x_1 = \log S \quad \text{and} \quad x_2 = \log A,$$

and use regression to determine f, g, h and P_1.

We will test the efficacy of Eqn. 8 this way in the next section.

III. Using the model to predict cache misses

We now demonstrate one application of the model, namely to predict cache misses. We do this in three steps:

(Step 1) Measure misses (including cold miss α) for direct map and fully associative caches.
(Step 2) Combine the measurements from Step 1 into one data set and fit it with Eqn. 9, to obtain f, g, h and P_1.
(Step 3) Substitute α from Step 1 and f, g, h and P_1 from Step 2 into Eqn. 8 and evaluate P for any A and S values.

Fig. 4 shows that, applying these three steps to the `gcc` workload, we obtain from the direct map and fully associative data a good prediction of the cache misses for set associative caches of size $C = 1$Kbytes and $C = 16$Kbytes..

However, the graphs also show that the `gcc` workload has miss probabilities that are roughly constant for a fixed cache size, regardless of associativity. (This is similarly true of `gzip`.)

To test the model's predictive ability against a workload that has greater variation in miss probability, we use some data published by Gee et al. [4]. Their data did not give α and fully associative misses, so we replace Step 1 by estimating α and using their measurements for direct map and $(C = 2K, A = 2), (C = 4K, A = 4), (C = 8K, A = 8)$.

Fig. 5 shows good agreement between measurement and the model's prediction for data and unified caches of size $C = 16K$ and $C = 256K$, using Gee et al.'s floating point benchmark data.

IV. Application to Performance-Power Design Space Exploration

Since cache consumes more than half power of all system power in embedded systems, reducing power consumption of cache is equally important as improving cache performance. In order to compute the power consumption of the cache design and analyses the pareto-optimal performance-power cache design, we combine a cache power model (CACTI) into our analytical model. The power consumption of cache consists of the static consumption and the dynamic power consumption. The static power consumption is due to current leakage and the dynamic power consumption is due to logic switching and the charging and discharging of the load capacitance. The CACTI power model, however, just concerns one part of the dynamic power consumption, namely that due to the charging and discharging of the load capacitance. In order to account for all power consumption of the cache, we adopt Zhang's equations [19] which considers both static and dynamic power consumption. Thus, the equation we use for computing the total power due to memory access is as follows:

$$power\_mem = power\_dynamic + power\_static \quad (10)$$

where:

$$power\_dynamic = \underline{cache\_hits} * \underline{power\_hit} + \underline{cache\_misses} * power\_miss$$
$$power\_miss = \underline{power\_offchip\_access} + \underline{power\_uP\_stall} + \underline{power\_cache\_block\_fill}$$
$$power\_static = \underline{cycles} * \underline{power\_static\_per\_cycle}$$

The underline terms are those we obtain through measurements or simulations. We compute *cache_hits* and *cache_misses* by running SimpleScalar simulations for each cache configuration. We compute *power_hit* of each cache configuration by using CACTI.

Determing the *power_miss* term is challenging. The *power_offchip_access* value is the power of accessing off-chip memory and the *power_uP_stall* is the power consumed when the microprocessor is stalled while waiting for the memory system to provide an instruction or data. *power_cache_block_fill* is the power for writing a block into the cache. The challenge stems from the fact that the first two terms are highly dependent on the particular memory and microprocessor being used. To be "accurate," it is possible to evaluate a "real" microprocessor system to determine the values for those terms. While accurate, those results may not apply to other systems, which may use different processors, memories, and caches. Therefore, Zhang's method is to choose instead to create a "realistic" system, and then to vary that system to see the impact across a range of different systems. They examined the three terms of *power_offchip_access*, *power_uP_stall*, and *power_cache_block_fill* for typical commercial memories and microprocessors, and found that *power_miss* ranged from 50 to 200 times bigger than *power_hit*. Thus, we redefined *power_miss* as:

$$power\_miss = k\_miss\_power * power\_hit \quad (11)$$

and we considered the situation of *k_miss_power* equal to 50 to 200.

Finally, *cycles* is the total number of cycles for the benchmark to execute, as computed by SimpleScalar, using a cache with single cycle access on a hit and using 20 cycles on a miss. *power_static_per_cycle* is the total static power consumed per cycle. This value is also highly system dependent, so we again consider a variety of possibilities, by defining this value as a percentage of total power including both dynamic and static power:

$$power\_static\_per\_cycle = k\_static * power\_total\_per\_cycle \quad (12)$$

k_static is a percentage that we can set. We define the *k_static* as 30% to 50% of the total power.

According to Eqn. 10, we compute the power consumption by obtaining *cache_hits* and *cache_misses* from SimpleScalar simulation and *power_hit* from CACTI simulation. Fig. 6 and Fig. 7 show the power consumed by SPEC2000 benchmarks, where *k_miss_power* is set to 50% and *k_static* is set to 30%. The results indicate that cache consumes more power as the set associativity increases when the cache size is fixed.

V. Related Works

One of the earliest analytical models was a power-law model proposed by Chow [2]. Rao [12] proposed the Independent Reference Model based on probabilistic principles. Voldman [17] and Thiébaut [15] applied *fractal geometry* to the prediction of the cache miss ratio. With this background, the seminal paper of Agarwal, Hennessy, and Horowitz [1] introduced a probabilistic cache model. The technical report of Fricker and Robert [3] also proposed a probabilistic cache model which is claimed to extend Agarwal's model and provide a more systematic study of the parameters that influence the behavior of a cache. While both Agarwal's model and Fricker's model need to take some parameter values from the trace-driven simulation, LaMarca and Ladner's analytical model [8] uses an algorithm instead of trace data as input. Ghosh, Martonosi and Malik [5] provided methods for generating and solving cache miss equations that give a detailed representation of the cache misses in loop-oriented scientific code. They also used the equations to guide code optimizations for improving cache performance.

Since inter-process conflicts cause noticeable effect on the cache performance, Agarwal's model also includes cache miss equations for inter-process conflicts. Thiébaut and Stone [16] developed an analytical model for cache-reload transients when a process is invoked periodically. Suh, Devadas, and Rudolph [14] presented an analytical model for the behavior of a cache in the multiprocessing system. Their model is claimed to work for any time quanta and needs only the isolated miss-rate curves for cache process,compared to Agarwal's model which works only for long enough time quanta and requires some parameters which is hard to be collected.

Unlike the analytical models where formulas are deduced mathematically, Higbie [6] proposed a computational model where performance equations are derived from extensive empirical data. It is well known that cache simulation can provide more accurate performance results than the analytical models do, but it is time intensive. In order to reduce the simulation time, some cache simulation algorithms [7] [13] take advantage of two properties, *inclustion* (that larger caches contain a superset of the blocks in smaller caches [10]) and *set-refinement*(that blocks mapping to the same set in larger caches map to the same set in smaller sets), to generate cache miss ratios of more than one cache configurations in single pass simulation.

Different from above analytical models, our analytical model predicts the cache miss ratio and interpolates data from runs with direct mapped and fully associative caches so that the entire parameters space involving all set-associativity is spanned. Moreover, our model can be coupled with a power model, for example the CACTI model, to get pareto-optimal performance-power design points for rapid cache design space exploration which can be used to guide the design of programmable cache. Malik, Moyer and Cermak [9] described one type of such programmable caches which provides certain features for power and performance tuning.

VI. Conclusion

In the paper, we present an analytical model that interpolates data from runs with direct-mapped and fully associative caches so that entire parameter space involving all set-associativity is spanned. Our analytical results highly agree with the results from full trace-driven simulations on cache miss-ratio. Furthermore, our analytical model can be coupled with a power model for programmable caches such that one can very quickly decide on pareto-optimal performance-power design points for rapid design space exploration.

Acknowledgment

This work is funded by Singapore-MIT Alliance. And the authors wish to thank Dr Yongxin Zhu for his helpful work on cache power model.

References

[1] A. Agarwal, J. Hennessy, and M. Horowitz. An analytical cache model. *ACM Trans. Comput. Syst.*, 7(2):184–215, 1989.

[2] C. K. Chow. Determining the optimum capacity of a cache memory. *IBM Technical Disclosure Bulletin*, 17(10):3163–3166, 1975.

[3] C. Fricker and P. Robert. An analytical cache model. Technical Report RR-1496, INRIA Rocquencourt, France, 1991.

[4] J. Gee, M. Hill, D. Pnevmatikatos, and A. Smith. Cache performance of the SPEC92 benchmark suite. *IEEE Micro*, 13(4):17–27, 1993.

[5] S. Ghosh, M. Martonosi, and S. Malik. Cache miss equations: An analytical representation of cache misses. In *Proc. of the 11th International Conference on Supercomputing*, pages 317–324. ACM Press, 1997.

[6] L. Higbee. Quick and easy cache performance analysis. *SIGARCH Comput. Archit. News*, 18(2):33–44, 1990.

[7] M. D. Hill and A. J. Smith. Evaluating associativity in cpu caches. *IEEE Trans. Comput.*, 38(12):1612–1630, 1989.

[8] A. LaMarca and R. E. Ladner. The influence of caches on the performance of heaps. *ACM Journal of Experimental Algorithms*, 1:4, 1996.

[9] A. Malik, B. Moyer, and D. Cermak. A low power unified cache architecture providing power and performance flexibility. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, pages 241–243. ACM Press, 2000.

[10] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Taiger. Evaluation techniques for storage hierarchies. *IBM Syst.*, 1970.

[11] R. Quong. Expected i-cache miss rates via the gap model. In *21st Annual International Symposium of Computer Architecture*, pages 372–383, April 1994.

[12] G. Rao. Performance analysis of cache memories. *Journal of ACM*, 25(3):378–395, 1978.

[13] R. Sugumar and S. Abraham. Efficient simulation of multiple cache configurations using binomial trees. Technical Report CSE-TR-111-91, 1991.

[14] G. Suh, S. Devadas, and L. Rudolph. Analytical cache models with applications to cache partitioning. In *Proc. of the 15th International Conference on Supercomputing*, pages 1–12, 2001.

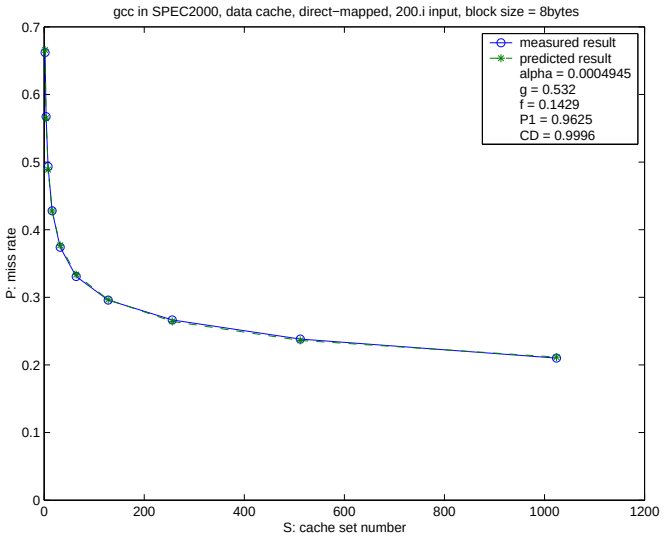
[15] D. Thiébaut. On the fractal dimension of computer programs and its application to the prediction of the cache miss ratio. *IEEE Trans. Comput.*, 38(7):1012–1026, 1989.

[16] D. Thiébaut and H. Stone. Footprints in the cache. *ACM Trans. Comput. Syst.*, 5(4):305–329, 1987.

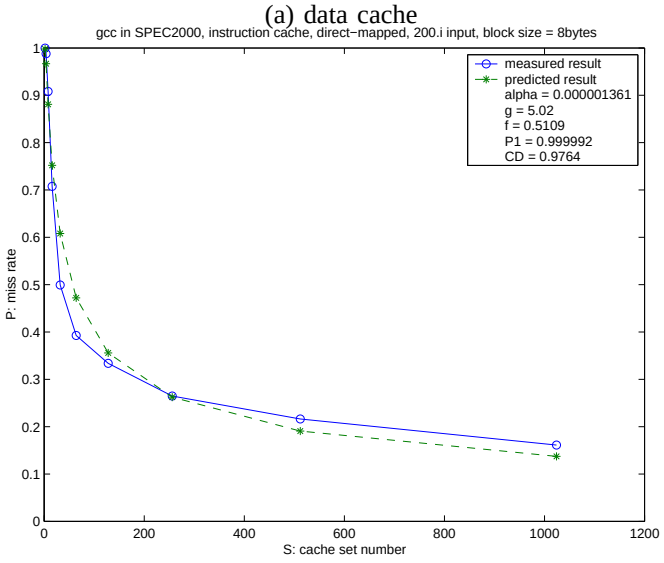
[17] J. Voldman, S. Mandelbrot, L. Hoevel, J. Knight, and P. Rosenfeld. Fractal nature of software-cache interaction. *IBM J. Res. Dev.*, 27(2):164–170, 1983.

[18] S. Wilton and N. Jouppi. Cacti: An enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits*, 31(5):677–688, 1996.

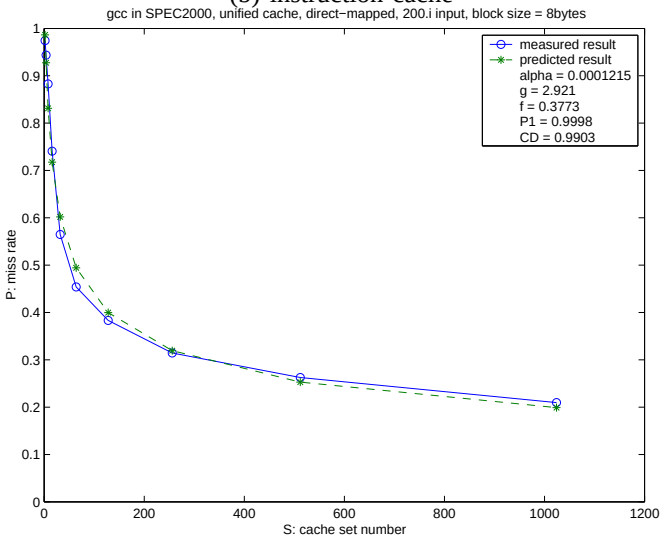
[19] C. Zhang, F. Vahid, and W. Najjar. A highly configurable cache architecture for embedded system. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 136–146. ACM Press, 2003.



(a) data cache

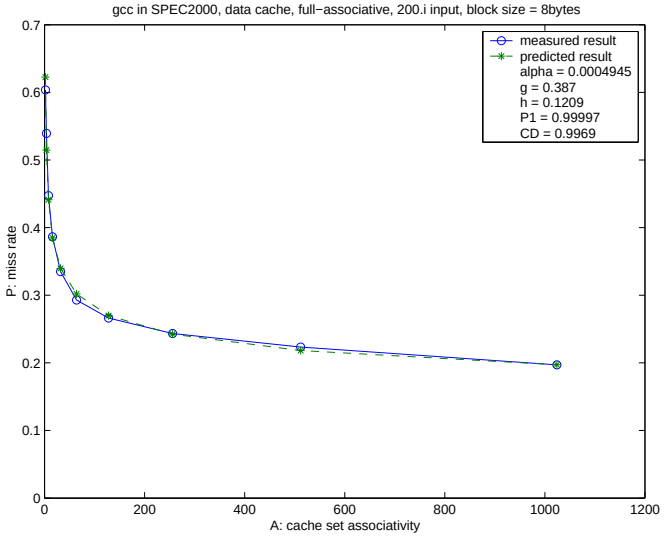


(b) instruction cache

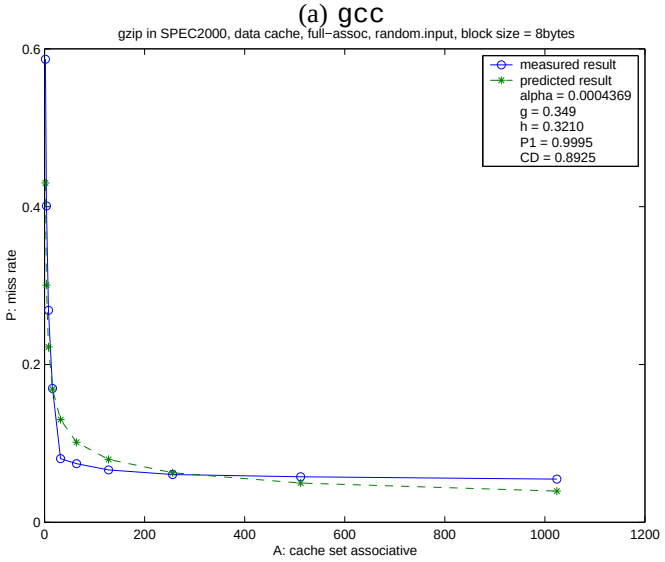


(c) unified cache

Fig. 2. **Eqn. 5 gives a good fit for measured miss probability for a direct map cache; the workload is `gcc`.**

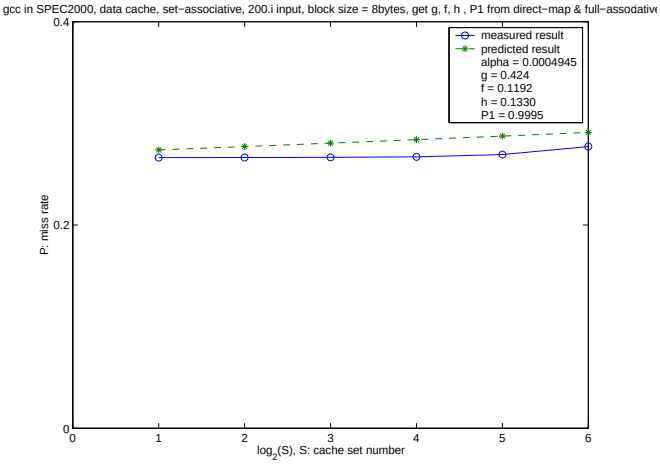


(a) `gcc`

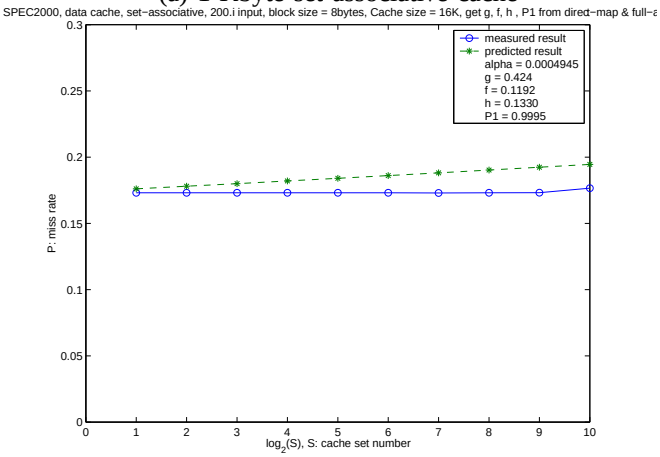


(b) `gzip`

Fig. 3. **Using Eqn. 7 to fit the measured miss probability for a fully associative data cache. The fit is good for `gcc`, but not as good for `gzip`.**

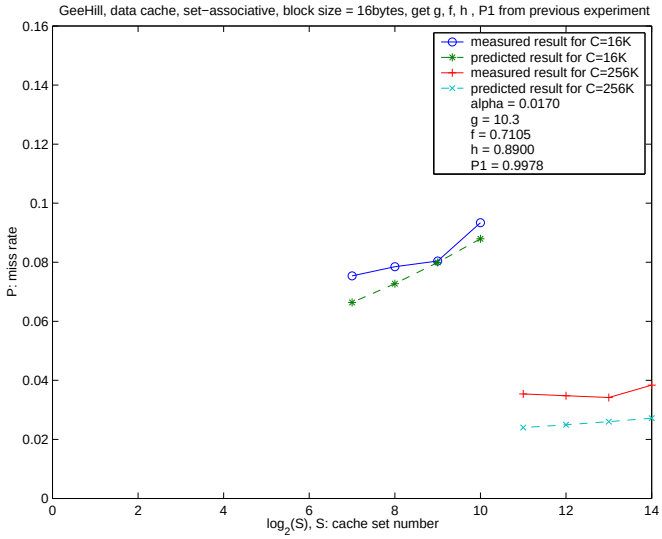


(a) 1-Kbyte set associative cache

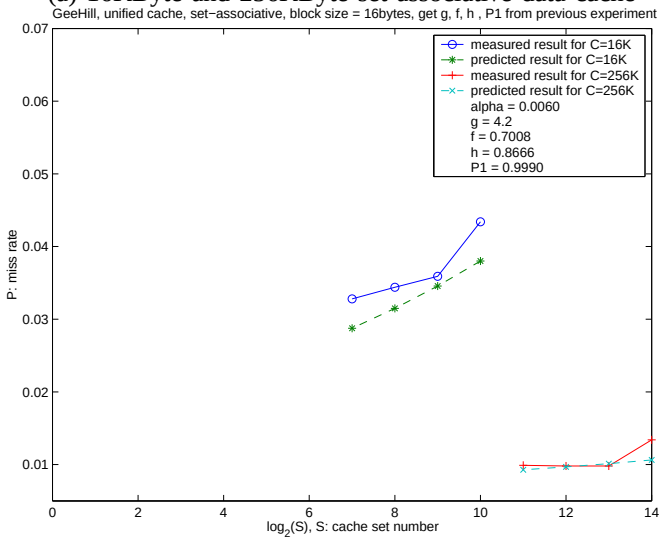


(b) 16-Kbyte set associative cache

Fig. 4. **Comparison of miss measurements and predictions using the 3-step process for set associative data cache misses. The errors are less than 10%.**

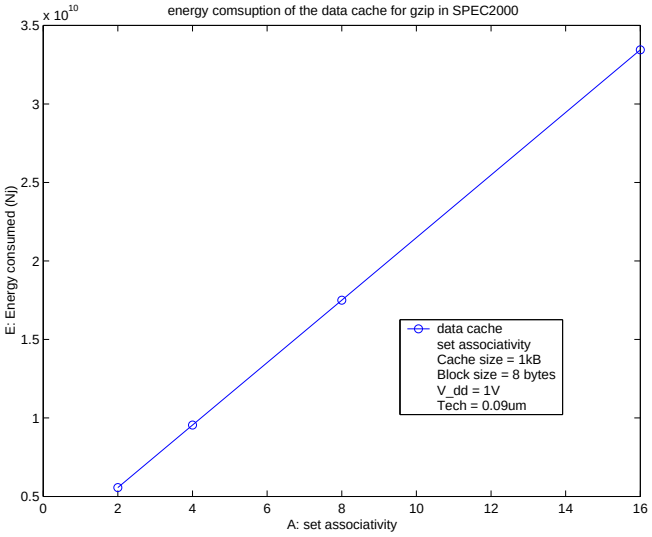


(a) 16KByte and 256KByte set associative data cache

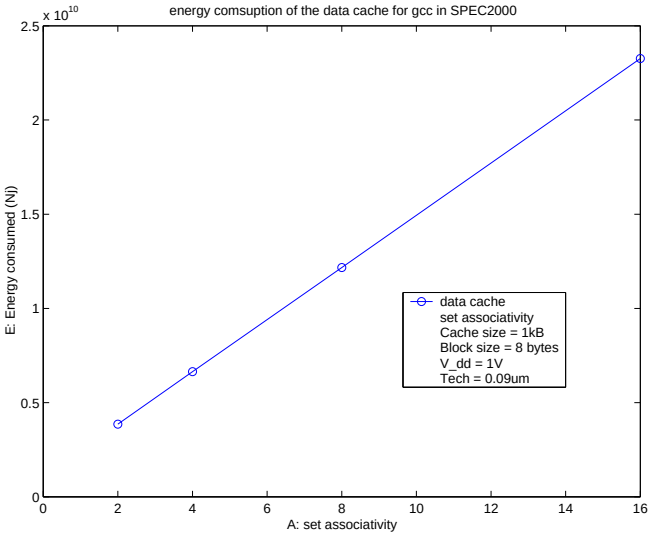


(b) 16KByte and 256KByte set associative unified cache

Fig. 5. **Comparison of miss measurements and predictions for Gee et al.'s data. (Their data for instruction cache were rounded off to $P = 0$ for the larger cache sizes, making it impossible to estimate the cold miss α.)**

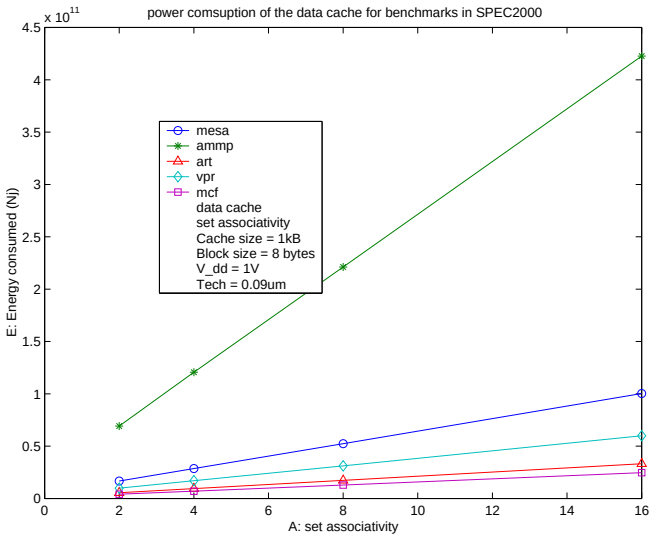


(a)gzip

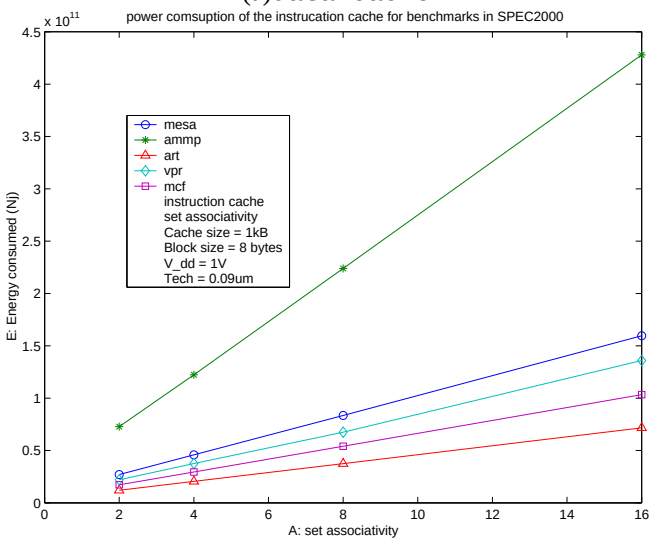


(b)gcc

Fig. 6. **Power consumed by the set associative data cache for gzip and gcc**



(a)data cache



(b)instruction cache

Fig. 7. **Power consumed by the set associative cache for SPEC benchmarks**