Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2015-033 December 19, 2015

Cache Calculus: Modeling Caches through Differential Equations

Nathan Beckmann and Daniel Sanchez

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Cache Calculus: Modeling Caches through Differential Equations

Nathan Beckmann Daniel Sanchez

Abstract—Caches are critical to performance, yet their behavior is hard to understand and model. In particular, prior work does not provide *closed-form solutions* of cache performance, i.e. simple expressions for the miss rate of a specific access pattern. Existing cache models instead use numerical methods that, unlike closed-form solutions, are computationally expensive and yield limited insight. We present *cache calculus*, a technique that models cache behavior as a system of ordinary differential equations, letting standard calculus techniques find simple and accurate solutions of cache performance for common access patterns.

1 Introduction

Practitioners often rely on simple formulas, like Amdahl's Law and Little's Law, to understand and design computer systems. Simple formulas complement simulation (which may be more accurate, but is restricted to specific inputs) by yielding broad insights about system behavior, and provide first-order answers to common questions—e.g., how large should the reorder buffer be to accommodate a given range of memory access latencies.

Unfortunately, few such formulas exist for caches, and those few are limited to specific access patterns [5, 7]. The standard way to find how a cache will perform is through time-consuming simulation. Alternatively, prior work has proposed cache models that yield faster predictions under arbitrary access patterns [1, 3, 4, 6, 8]. These models avoid the need for long simulations, but they do not provide closed-form solutions:[1] they are numerical in nature, and, like simulation, they produce a miss rate for a particular input, not an equation that lets designers reason about how performance changes with cache size or access pattern.

For some purposes it would be preferable to have simpler models that capture common behaviors. Such models support early design exploration, where simplicity and insight are paramount. They can also be used in algorithmic analysis, e.g. to compare performance on different input sizes. For these uses, cache models need only capture first-order effects, since detailed modeling or simulation would be done later in the design.

This paper presents *cache calculus*, the first general technique that provides closed-form solutions of cache performance. Cache calculus advances the state-of-the-art by modeling caches through differential equations, revealing surprisingly simple expressions for how caches behave. Unlike prior models, which aim to replace simulation, cache calculus complements it by providing simple, closed-form (or transcendental) expressions.[1]

Road map: We begin by motivating cache calculus and presenting the discrete model it builds upon (§2). Cache calculus relaxes this model into a system of ordinary differential equations (ODEs, §3). We then show how to solve the ODEs for common access patterns (§4), including combinations of common patterns. In this paper, we derive formulas for stack, scanning, and random access patterns on caches with random replacement. Finally, we illustrate how cache calculus can be applied to analytically intractable patterns through numerical analysis (§5).

- *The authors are with MIT CSAIL, Cambridge, MA, 02139. E-mail: {beckmann,sanchez}@csail.mit.edu*
- *This work was supported in part by NSF grant CCF-1318384 and a grant from the Qatar Computing Research Institute.*

1. A *closed-form* expression contains a constant number of elementary operations (e.g., Eq. 14), whereas a *transcendental* expression contains a non-elementary function of the variable being solved for (e.g., Eq. 21).

2 Background

Cache models try to predict the cache's hit rate using a probabilistic description of the access pattern and a description of the cache (e.g., its size). Prior models [1, 3, 4, 5, 6, 8] aim to provide an efficient alternative to simulation, to, for example, accelerate design space exploration [1, 5, 8] or perform dynamic cache partitioning [3]. By contrast, our main goal is to augment simulation through simpler, closed-form models. Cache calculus provides simple equations that capture the main tradeoffs and aid in the early stages of design. To that end, we focus on common access patterns and simple approximations.

Specifically, we extend a discrete cache model from our prior work [3], hereafter called "the discrete model". The discrete model was designed to predict the performance of arbitrary age-based replacement policies. It relies on two critical simplifying assumptions. First, that the cache provides sufficient associativity so that conflict misses are a second-order concern. Hashing, common in commercial LLCs, satisfies this assumption. Second, that *reuse distances*[2] are independent and identically distributed (iid) following the probability distribution denoted $\mathrm{P}_D(d)$.[3] This assumption is motivated by observing that private caches filter successive references to the same address, so the LLC sees an access stream stripped of short-term temporal correlations. Although approximate, the iid assumption is surprisingly robust across many access patterns and policies (§4), and in turn makes the model tractable.

From these assumptions, the discrete model produces three interdependent equations that describe the cache's behavior on the access pattern. Specifically, it describes the probability of different events as a function of their *age*.[4] The discrete model describes the distribution of ages A, hits H, and evictions E. For example, $\mathrm{P}_A(10)$ is the probability that a randomly selected line has age 10, and $\mathrm{P}_E(16)$ is the probability that a randomly selected line will be evicted at age 16. The cache's hit rate is the total hit probability $\mathrm{P}[\text{hit}] = \sum_{x=1}^{\infty} \mathrm{P}_H(x)$.

We now briefly describe the equations in the discrete model, since they are the basis for the ODEs used in cache calculus. Refer to [3] for detailed explanations.

Ages: In order to survive to age a, a line must first survive to age $a-1$. Thus the probability a randomly selected cache line has age a decreases with larger a, and the probability a random cache line has age a is just the sum of all hits or evictions at later ages. Finally, since every access produces exactly one line of age 1 (whether by hit or eviction), with cache size S

$$\mathrm{P}_A(a) = \frac{1}{S} \sum_{x=a}^{\infty} \mathrm{P}_H(x) + \mathrm{P}_E(x) \tag{1}$$

Hits: To compute the probability of hits at age a, the discrete model observes that a hit of age a must have reuse distance a. Thus the hit probability is related to the reuse distance probability. Moreover, evictions at earlier ages make later hits less likely. Specifically, the probability a random cache line will hit at age a is

2. In this paper, *reuse distance* is the number of accesses between references to the same address; contrast with *stack distance*, the number of unique addresses. Some prior work treats these terms synonymously.

3. In our notation, $\mathrm{P}_X(x)$ is the probability that random variable X equals x, and $\mathrm{P}[X > x]$ is the probability that X is larger than x.

4. The *age* of a cache line is the number of accesses that have elapsed since it was last referenced (i.e., since last hit or insertion).

$$\mathrm{P}_H(a) = \mathrm{P}_D(a) \times \left(1 - \sum_{x=1}^{a-1} \frac{\mathrm{P}_E(x)}{\mathrm{P}[D > x]}\right) \tag{2}$$

The latter factor is the fraction of hits at age a lost due to evictions at earlier ages.

Evictions: Evictions are the most complicated part of the discrete model. Due to space constraints, in this paper we specialize the eviction equation for random replacement.

With random replacement, the probability that eviction occurs at age a is just the probability that the cache misses and that the (randomly chosen) victim has age a

$$\mathrm{P}_E(a) = \mathrm{P}[\text{miss}] \times \mathrm{P}_A(a) \tag{3}$$

The discrete model was thoroughly validated and shown to be accurate: on SPEC CPU2006 benchmarks it yields negligible median error and mean error of under 4%. But despite this accuracy, the discrete model equations do not readily yield insight into the relationship between the cache size S or access pattern D. Moreover, it requires iteration that is orders of magnitude more expensive than cache calculus's models. These problems afflict all prior, general-purpose models [1, 4, 6, 8], and other models [5, 7] are limited to specific access patterns.

3 Cache Calculus

Cache calculus transforms the discrete model into a system of ordinary differential equations that admit standard, well-studied calculus techniques. We first introduce variables representing the cumulative probability function (CDF) of each random variable in the discrete model, shown in Table 1.

TABLE 1
ODE variables and corresponding random variables in discrete model.

Discrete model	$\mathrm{P}[D < a]$	$\mathrm{P}[A < a]$	$\mathrm{P}[H < a]$	$\mathrm{P}[E < a]$
Cache calculus	$D(a)$	$A(a)$	$H(a)$	$E(a)$

Using these variables, we can rewrite the discrete model equations. First, following ODE conventions, we often drop the parameter a; i.e., H denotes $H(a)$ unless otherwise specified. Second, observe from the Fundamental Theorem of Calculus that $H' = \mathrm{P}_H(a)$, or $f_H(a)$ using continuous probability notation. The same holds for other variables.

Ages: The continuous approximation of the age distribution is

$$A' = \frac{1 - (H + E)}{S} \tag{4}$$

In detail, the transformation is (starting from Eq. 1)

$$\mathrm{P}_A(a) = \frac{1}{S} \sum_{x=a}^{\infty} \mathrm{P}_H(x) + \mathrm{P}_E(x)$$

Relaxing to continuous probability

$$\begin{aligned} f_A(a) &= \frac{1}{S} \int_a^{\infty} \left[f_H(x) + f_E(x)\right] \mathrm{d}x \\ &= \frac{1}{S} \left(1 - \int_0^{a} \left[f_H(x) + f_E(x)\right] \mathrm{d}x\right) \end{aligned}$$

Substituting $A' = f_A(a)$ and $H + E = \int_0^a \left[f_H(x) + f_E(x)\right] \mathrm{d}x$ yields Eq. 4. The remaining transformations are similar, but due to space constraints we omit their details.

Hits: To write the hit distribution as an ODE, it is helpful to introduce a new variable that represents the second factor in Eq. 2; i.e., the fraction of hits lost to earlier evictions

$$B(a) = \int_0^a \frac{E'(x)}{1 - D(x)} \mathrm{d}x \qquad \Rightarrow \qquad B' = \frac{E'}{1 - D} \tag{5}$$

With B, the hit distribution is simply

$$H' = D'(1 - B) \tag{6}$$

Evictions: Let $m = \mathrm{P}[\text{miss}]$ be the cache's miss rate. Then for random replacement (Eq. 3), the eviction distribution is

$$E' = mA' \tag{7}$$

3.1 Summary

The above equations form a non-autonomous system of differential equations in four variables (A, B, H, E—recall that D is a model input). The system yields the cache's miss rate as $m = \lim_{a\to\infty} E$, which always has one solution in $[0, 1]$ as well as the extraneous solution $m = 0$. Moreover, the system is linear for random replacement, making it easy to find closed-form solutions. Unfortunately, the system is non-linear for LRU and other policies, but can be solved using numerical analysis (§5) or a fully associative formulation (omitted for space; see [2, §D.5]).

4 Analytical Solution

We now consider cache performance under three access patterns: the *scan* pattern iterates repeatedly over an array, the *stack* pattern scans back and forth over an array, and the *random* pattern accesses random array elements. These are illustrated over time on an N-element array in the left side of Fig. 1.

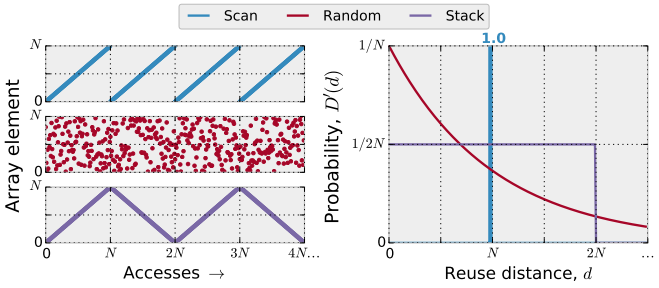


Fig. 1: Example access patterns. (left) Accesses over time to different array elements. (right) Resulting reuse distance distribution.

The right side of Fig. 1 shows the resulting reuse distance distributions. All references in the scanning pattern have the same reuse distance N; random accesses have exponentially distributed reuse distances; and the stack pattern has uniform reuse distances from 1 to $2N$. Hence the cumulative reuse distance distributions for each are

$$D_{\text{scan}}(x) = \begin{cases} 0 & \text{If } x < N \\ 1 & \text{Otherwise} \end{cases} \tag{8}$$

$$D_{\text{random}}(x) = 1 - \left(\frac{N-1}{N}\right)^x \tag{9}$$

$$D_{\text{stack}}(x) = \begin{cases} x/2N & \text{If } x < 2N \\ 1 & \text{Otherwise} \end{cases} \tag{10}$$

Note that reuse distances in the scan and random patterns are iid, whereas those in the stack pattern are not. Yet we will see that the model is accurate on each pattern.

One intuitive result from the following analysis is that the cache's miss rate is *scale-invariant*. That is, only the ratio of the array size and cache size matter, not their absolute size. We denote this ratio $\omega = S/N$.

4.1 Simple patterns

To arrive at closed-form solutions for random replacement, we solve the following initial value problem derived from the ODEs:

$$H'' = \frac{D''}{D'} H' - \frac{D'}{1 - D} E' \quad \text{and} \quad E'' = -\frac{m}{S}(H' + E') \tag{11}$$

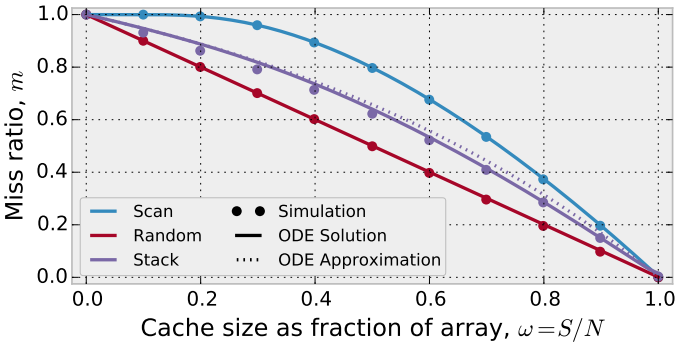


Fig. 2: Comparison of ODE solution (—) vs. simulation (●) on random replacement for several access patterns.

These equations eliminate variables A and B, simplifying the analysis. The initial values for all variables are zero, so $H(0) = E(0) = 0$ and (by substitution) $H'(0) = D'(0)$ and $E'(0) = m/S$. (Age zero simply denotes infinitesimally small ages.) These initial values mean that there are no hits or evictions initially, so short reuse distances hit, and, since small ages are more common, evictions are also more likely.

Scanning pattern: To solve for the cache's miss rate on a scanning pattern with random replacement, we first observe that *(i)* all lines hit at age N if not evicted, *(ii)* no hits occur at ages less than N, and *(iii)* all evictions occur at ages less than N. Thus for ages $0 < a < N$, $H' = 0$ and

$$E'' = -\frac{m}{S}E' \quad \Rightarrow \quad E = 1 - \mathrm{e}^{-ma/S} \tag{12}$$

The right-hand side follows from initial conditions. We can now compute the miss rate m. Since no evictions occur beyond age N

$$m = \int_0^N E' \,\mathrm{d}a = E(N) - E(0) = 1 - \mathrm{e}^{-m/\omega} \tag{13}$$

We solve for m via the product logarithm $W_0(x) = z \Rightarrow z\mathrm{e}^z = x$

$$\boxed{m = 1 + \omega\, W_0\left(-\mathrm{e}^{-1/\omega}/\omega\right)} \tag{14}$$

Fig. 2 compares Eq. 14 with simulation of microbenchmarks on a small cache using random replacement. (We have confirmed that model results hold for different array sizes N.) Simulation results are given by circles (●), model solutions are solid lines (—), and colors show different access patterns. As the figure shows, Eq. 14 accurately predicts the scanning miss rate over all cache sizes.

Random accesses: The reuse distance distribution for a random access pattern has exponential decay, since every address has $1/N$ chance of being referenced at each access. Substituting Eq. 9 into Eq. 11 yields a simpler equation for H''

$$H'' = (H' + E')\ln\frac{N-1}{N} \tag{15}$$

This equation has a similar form to evictions (Eq. 11). To simplify, let $\ell = \ln(N/(N-1))$. Given the initial conditions, these equations yield a solution of the form

$$H' = \ell\, \mathrm{e}^{-(\ell + m/S)a} \quad \text{and} \quad E' = \frac{m}{S}\,\mathrm{e}^{-(\ell+m/S)a} \tag{16}$$

We can now compute the miss rate

$$m = \int_0^\infty E' \,\mathrm{d}a = \frac{m/S}{\ell + m/S} \tag{17}$$

which means either m is zero or, if $S < N$,

$$\boxed{m = 1 - S\ell \approx 1 - \omega} \tag{18}$$

The final approximation holds for large N, where $\ell \approx 1/N$. Fig. 2 shows that the model is, once again, an excellent match with simulation.

Stack pattern: Similar to the scanning pattern, the stack pattern only has hits or evictions at ages $0 < a \le 2N$. We simplify the hit distribution using the derivatives of Eq. 10, $D' = 1/2N$ and $D'' = 0$

$$H'' = \frac{E'}{a - 2N} \tag{19}$$

Evictions do not simplify and are unchanged from Eq. 11.

Given the initial conditions, these equations are solved by

$$H' = \frac{1}{2N}\mathrm{e}^{-ma/S} \quad \text{and} \quad E' = \left(1 - \frac{a}{2N}\right)\frac{m}{S}\,\mathrm{e}^{-ma/S} \tag{20}$$

This yields the miss rate

$$m = \int_0^{2N} E' \,\mathrm{d}a = 1 + \frac{\mathrm{e}^{-2m/\omega} - 1}{2m/\omega} \tag{21}$$

Unlike prior patterns, Eq. 21 does not yield an explicit solution for m (it is a transcendental equation). But it can be used to quickly iterate to a fixed-point on m by repeatedly evaluating the right-hand side of Eq. 21. This iteration is far more efficient than iterating on a full discrete model like [1, 3, 4, 6, 8].

Although an exact, closed-form solution is not forthcoming for the stack distribution, we can get a closed-form approximation. First, consider that when $N \gg S$, the miss rate is high $m \approx 1$, the cache is small $\omega \approx 0$, and thus $\mathrm{e}^{-2m/\omega} \approx 0$. Hence

$$m \approx 1 - \frac{\omega}{2m} \quad \Rightarrow \quad m \approx \frac{1}{2}\left(1 + \sqrt{1 - 2\omega}\right) \tag{22}$$

This approximation works well at high miss rates, and we can extend it across the full range by taking its Taylor series around $\omega = 0$

$$\boxed{m \approx 1 - \frac{\omega}{2} - \frac{\omega^2}{4} - \frac{\omega^3}{4} - O\left(\omega^4\right)} \tag{23}$$

Fig. 2 compares this approximation $(\cdots)$, 100 iterations of Eq. 21 (—), and simulation (●). It shows that the approximation is fairly accurate throughout the range. The small error at low miss rates comes from reuse distances not being iid.

4.2 Hybrid patterns

Cache calculus is easily extended to model more complicated patterns, for example by combining the patterns just seen. We give one simple example of a program scanning over arrays of different sizes.

Hybrid scan: Suppose that a program accesses two data structures, a fraction p of accesses scanning an array of size N_1, and a fraction $1 - p$ scanning an array of size N_2. This access pattern produces a more complicated reuse distance distribution, shown in Fig. 3. Note that this pattern also does *not* satisfy the iid assumption, yet we will see that the model remains accurate.

Scanning over the first array takes N_1/p accesses on average, with some variance depending on how often it is accessed, and likewise for the second array. With large reuse distances, we

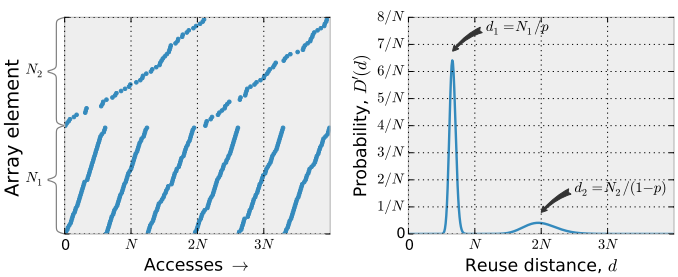


Fig. 3: A more complex pattern that scans over two arrays. The N array elements are split evenly into two arrays N_1 and N_2, and $p = 3/4$ accesses go to the first array.

can model the reuse distance distribution using the normal distribution (details omitted).

However, the normal distribution is complicated to analyze, so we instead model that all accesses to the first array have reuse distance $d_1 = N_1/p$, and all to the second have reuse distance $d_2 = N_2/(1-p)$. With this crude approximation, there are no hits except at ages d_1 and d_2.

Without loss of generality, suppose that $d_1 < d_2$. Then until age d_1, evictions are unchanged from the prior scanning example (Eq. 12), and thus at age d_1 there have been

$$E(d_1) = 1 - \mathrm{e}^{-md_1/S} \tag{24}$$

evictions. The hits at age d_1 are just the fraction of accesses to the first array that have not been evicted (Eq. 6)

$$H(d_1) = D'_{\text{mixed-scan}}(d_1)\left(1 - E(d_1)\right) = p\,\mathrm{e}^{-md_1/S} \tag{25}$$

Since accesses to the first array hit at age d_1, afterwards only accesses to the second array remain. This makes later ages less common. Specifically, just after age d_1, only a fraction $(1-p)\ \mathrm{e}^{-md_1/S}/S$ of lines remain (computed from the age distribution, Eq. 4). Since evictions depend on the age probability (Eq. 7), the eviction distribution changes after age d_1 as well. Specifically, we must choose new constants when solving Eq. 12. After doing so, we find that the evictions from ages d_1 to d_2 are

$$E(d_2) - E(d_1) = (1-p)(\mathrm{e}^{-md_1/s} - \mathrm{e}^{-md_2/s}) \tag{26}$$

And thus the overall miss rate is (adding Equations 24 and 26)

$$m = 1 - p\,\mathrm{e}^{-mN_1/pS} - (1-p)\,\mathrm{e}^{-mN_2/(1-p)S} \tag{27}$$

Eq. 27 has a similar form to the miss rate when scanning a single array (Eq. 13), but with two terms, one for each array. Indeed, Eq. 27 generalizes to many arrays with sizes N_i and access probabilities p_i

$$\boxed{1 - m = \sum_i p_i\,\mathrm{e}^{-mN_i/p_iS}} \tag{28}$$

The left-hand side is the cache's hit rate, and each term in the right-hand sum gives the hits to the i^{th} array.

Unlike when scanning a single array, Eq. 28 does not yield an explicit, closed-form solution. However, it does provide a transcendental expression that can be repeatedly evaluated to rapidly converge to a fixed point on m (see Fig. 5b in the next section). Fig. 4 shows that the ODE solution is accurate for a variety of different array sizes with 100 iterations of Eq. 27.

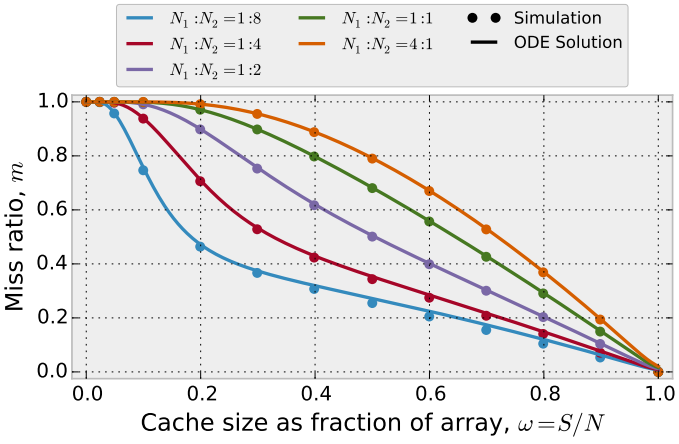
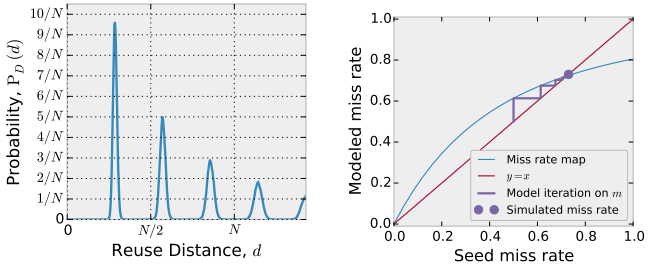


Fig. 4: Comparison of ODE solution (—) vs. simulation (●) on random replacement for several array ratios on the mixed scanning pattern. Total array size $N = N_1 + N_2$ and $p = 3/4$ of accesses go the first array.

5 Numerical Solution

Some access patterns are difficult to express analytically, or come from online monitors that cannot be pre-characterized. Even in these cases, cache calculus provides efficient miss rate predictions via numerical analysis. Specifically, we solve the system of ordinary different equations on A, B, H, and E from §3 using the Runge-Kutta method. Adaptive methods could be used to improve efficiency, while providing rigorous error bounds unlike prior models. Otherwise, this application of cache calculus is similar to prior reuse-distance-based models [3, 4, 8].

As in analytical solutions, numerical solutions require the miss rate m to set up initial conditions, but m itself is a product of the solution. Numerical solutions therefore converge on m through iteration (similar to Fig. 4.2). Fig. 5 shows an example.



(a) Reuse distance distribution. (b) Iteration on m.

Fig. 5: Example numerical solution on a complex access pattern.

The left-hand side shows the reuse distance distribution of an access pattern that scans over an array while randomly skipping some elements at each step. This distribution is gathered empirically from a pseudo-randomly generated trace. The right-hand side shows how numerical solution converges to the actual miss rate. We solve the initial value problem, starting from $m = 0.5$, to produce a new miss rate (plotted on the line "Miss rate map"). The solution then repeats using the new miss rate as a seed until it converges to a fixed point (where it intersects the line $y = x$). As shown, this process takes just a few iterations to converge ("Model iteration on m").

6 Conclusion

Architects frequently use simple mathematical models to reason about their designs, but such closed-form expressions are lacking for caches. We have presented *cache calculus*, the first general technique that yields simple mathematical formulas for cache performance across many access patterns. Cache calculus leverages the well-studied theory of ODEs, yielding closed-form or transcendental models of cache performance.

References

[1] A. Agarwal, J. Hennessy, and M. Horowitz, "An analytical cache model," *ACM Trans. Comp. Sys. (TOCS)*, vol. 7, no. 2, 1989.

[2] N. Beckmann, "Design and analysis of spatially-partitioned shared caches," Ph.D. dissertation, MIT, 2015.

[3] N. Beckmann and D. Sanchez, "Modeling cache performance beyond lru," in *HPCA-22*, 2016.

[4] E. Berg and E. Hagersten, "Statcache: A probabilistic approach to efficient and accurate data locality analysis," in *ISPASS*, 2004.

[5] H. Che, Y. Tung, and Z. Wang, "Hierarchical web caching systems: Modeling, design and experimental results," in *IEEE Journal on Selected Areas in Communications*, 2002.

[6] C. Ding and Y. Zhong, "Predicting whole-program locality through reuse distance analysis," in *PLDI*, 2003.

[7] M. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," in *ISCA-34*, 2007.

[8] R. Sen and D. A. Wood, "Reuse-based online models for caches," in *Proc. SIGMETRICS*, 2013.