The placement of matrices when using XOR-based hashing

Bavo Nootaert*,
Hans Vandierendonck*,
Koen De Bosschere*

* *ELIS, Ghent University, Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium*

ABSTRACT

To reduce the number of conflicts in caches, hash function are used. One family of hash function that has been reported to perform well in the presence of stride patterns are based on XOR-ing address bits together. We show that, in contrast to traditional modulo-2^m-based hashing, for XOR-based hashing the placement of stride patterns and matrices has a significant impact on conflicts.

KEYWORDS: cache, optimization, stride pattern, XOR-based hash function

1 Introduction

Hash functions have been studied with the aim of avoiding conflict misses in caches [Schl93, Gonz97, Toph99]. In scientific programs, stride patterns are common. It is important to choose a hash function so as to map as many strides as possible with few conflicts. Conventional bit selection has the disadvantage of mapping even strides onto only one halve of the sets. A well studied alternative are the XOR-based hash functions [Frai85, Gonz97, Vand05], and a special subclass thereof, the polynomial hash functions [Rau91]. These functions are computed using arithmetic over GF(2) and can be evaluated using solely XOR-gates.

In [Toph99] benchmarks are used to measure the performance of these hash functions. Analytical studies of XOR-based hash functions are usually limited to vector space patterns [Frai85, Vand05]. Analyzing these patterns is rather restrictive, since this class contains only certain strides, aligned to certain base addresses. In [Rau91], some properties are proven about the effect of polynomial hash functions on stride patterns starting at base address 0. Although some of the theorems can be extended to include a different base address [Noot05], they are aimed at determining equivalence among patterns, and generally cannot say whether a particular base address is actually a good choice. Indeed, the number of conflicts varies greatly depending on the alignment of the stride pattern.

In Section 2 we model self interference in a matrix using stride patterns. In Section 3 we use this model to show the potential impact of the base address, independent of any particular benchmark.

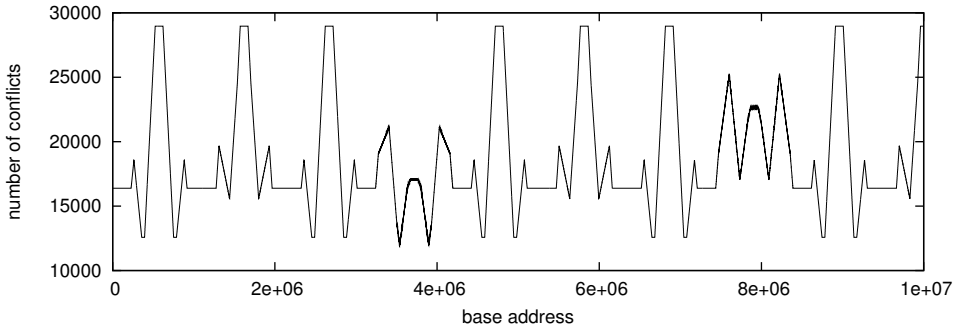

Figure 1: The number of conflicts for a 976×976 matrix.

2 A model for self interference in a matrix

We focus on self interference [Lam91] in a matrix: if the cache is large enough, cross interference can be ignored. Assume interference is mainly restricted to within separate columns or rows of a matrix. This assumption is valid if each column or row is reused repeatedly, without intermediate accesses to other columns of the same matrix.

A matrix can now be considered as a collection of stride patterns. An $S \times S$ matrix stored in row major order is made of S patterns with stride S (the columns), and S with stride 1 (the rows). Let a be the starting address of a matrix A. The patterns with stride S start at addresses a, $a+1$, ..., $a+S-1$, and the others start at addresses a, $a+S$, ..., $a+(S-1)S$.

A *conflict* occurs when an element of a pattern is mapped to a set that already contains an element. The number of conflicts is independent of the order in which the elements are mapped. We define the number of *conflicts of a matrix* as the sum of the conflicts of the column access patterns and the row access patterns.

3 An example

Consider a cache with 2^{13} sets and a mapping that XORs two slices of 13 bits together, and discards any bits above the 26 least significant. Assume for simplicity that a cache line can hold exactly one element of the matrix.

Figure 1 shows the number of conflicts for $S = 976$ and base addresses ranging from 0 to $10,000,000$. Here, optimal placement reduces the number of conflicts with about 59% compared to the worst case.

As can be observed, there is some symmetry, which is described next. Proofs can be found in [Noot05].

For a pattern with stride S of size W, the number of conflicts is symmetric around the points b_j, given by:

$$b_j = 2^{j+t} + \left\lceil \frac{S}{2} \right\rceil - \left\lceil \frac{WS}{2} \right\rceil - 1,$$

where t is the smallest integer that makes b_0 positive.

The range $[0, 2b_j]$ does not reach b_{j+1}. So if one examines the base addresses starting from 0, the range $]2b_j, b_{j+1}]$ contains new information. In this range, there may be an optimal base address that has not yet been discovered.

After the range $[0, 2b_0]$, which is symmetric since b_0 is its center, has been reflected around b_1, both the original and its image are reflected again around b_2, resulting in four copies of the range. After the next symmetry point, we have eight copies, and so on.

It can be proven that for the model described in Section 2, the number of conflicts for a matrix shows a similar symmetric behavior. The symmetry points for an $S \times S$-matrix are

$$b'_j = 2^{j+t'} - \left\lceil \frac{S^2}{2} \right\rceil,$$

where t' is the smallest integer that makes b'_0 positive.

4 Conclusion

The base address of a stride pattern or a matrix pattern has a significant impact on the number of conflicts, when mapped using a XOR-based hash function. This is the opposite of conventional modulo-based hashing, where the base address has no effect at all on self interference. These results offer a new possibility for optimizing the number of conflicts.

References

[Frai85] J. FRAILONG, W. JALBY, AND J. LENFANT. XOR-schemes: a flexible data organization in parallel memories. In *Proceedings of the 1985 International Conference on Parallel Processing*, pages 276–283, August 1985.

[Gonz97] A. GONZÁLEZ, M. VALERO, N. TOPHAM, AND J. PARCERISA. Eliminating cache conflict misses through XOR-based placement functions. In *ICS '97: Proceedings of the 11th international conference on Supercomputing*, pages 76–83. ACM Press, 1997.

[Lam91] M. LAM, E. ROTHBERG, AND M. E. WOLF. The cache performance and optimizations of blocked algorithms. In *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 63–74, April 1991.

[Noot05] B. NOOTAERT, H. VANDIERENDONCK, AND K. DE BOSSCHERE. How patterns in memory references affect the performance of hash functions in cache memories. Technical Report R105.002, Ghent University, March 2005.

[Rau91] B. RAU. Pseudo-randomly interleaved memory. In *Proceedings of the 18th Annual International Symposium on Computer Architecture*, pages 74–83, May 1991.

[Schl93] M. SCHLANSKER, R. SHAW, AND S. SIVARAMAKRISHNAN. Randomization and associativity in the design of placement-insensitive caches. Technical report, HP Computer Systems Laboratory, June 1993.

[Toph99] N. TOPHAM AND A. GONZÁLEZ. Randomized Cache Placement for Eliminating Conflicts. *IEEE Transactions on Computers*, 48(2):185–192, 1999.

[Vand05] H. VANDIERENDONCK AND K. DE BOSSCHERE. XOR-based Hash Functions. *IEEE Transactions on Computers*, 54(7):800–812, 2005.