Efficient cache use for stencil operations on structured discretization grids

Michael Frumkin and Rob F. Van der Wijngaart
NAS Technical Report NAS-00-015 December 2000

{frumkin,wijngaar}@nas.nasa.gov
Computer Sciences Corporation
NASA Ames Research Center
Mail Stop T27A-2
Moffett Field, CA 94035-1000

Abstract

We derive tight bounds on cache misses for evaluation of explicit stencil operators on structured grids. Our lower bound is based on the isoperimetric property of the discrete octahedron. Our upper bound is based on a good surface-to-volume ratio of a parallelepiped spanned by a reduced basis of the interference lattice of a grid. Measurements show that our algorithm typically reduces the number of cache misses by a factor of three, relative to a compiler optimized code. We show that stencil calculations on grids whose interference lattices have a short vector feature abnormally high numbers of cache misses. We call such grids unfavorable and suggest to avoid these in computations by appropriate padding. By direct measurements on a MIPS R10000 processor we show a good correlation between abnormally high numbers of cache misses and unfavorable three-dimensional grids.


NASA

Efficient cache use for stencil operations on structured discretization grids

Michael Frumkin and Rob F. Van der Wijngaart*
Numerical Aerospace Simulation Systems Division
NASA Ames Research Center

Abstract

We derive tight bounds on cache misses for evaluation of explicit stencil operators on structured grids. Our lower bound is based on the isoperimetric property of the discrete octahedron. Our upper bound is based on a good surface-to-volume ratio of a parallelepiped spanned by a reduced basis of the interference lattice of a grid. Measurements show that our algorithm typically reduces the number of cache misses by a factor of three, relative to a compiler optimized code. We show that stencil calculations on grids whose interference lattices have a short vector feature abnormally high numbers of cache misses. We call such grids unfavorable and suggest to avoid these in computations by appropriate padding. By direct measurements on a MIPS R10000 processor we show a good correlation between abnormally high numbers of cache misses and unfavorable three-dimensional grids.

1 Introduction

On modern computers the gap between access times to cache and to global memory amounts to several orders of magnitude, and is growing. As a result, improvement in usage of the memory hierarchy has become a significant source of enhancing application performance. Well-organized data traffic may improve the performance of a program, without changing the actual amount of computation, by reducing the time the processor stalls while waiting for data. Both data location and access pattern affect the amount of data movement in the program, and the effectiveness of the cache.

A number of techniques for improvements in the usage of data caches have been developed in recent years. The techniques include improvements in data reuse (i.e. temporal locality) [3, 4, 5, 13], improvements in data locality (i.e. spatial locality) [13], and reductions in conflicts in data accesses [4, 5, 9, 10]. In practice, these techniques are implemented through code and data transformations such as array padding and loop unrolling, tiling, and fusing. Tight lower and upper bounds on memory hierarchy access complexity for FFT and matrix

*Computer Sciences Corporation; M/S T27A-2, NASA Ames Research Center, Moffett Field, CA 94035-1000; e-mail: {frumkin,wijngaar}@nas.nasa.gov

multiplication algorithms are given in [8]. However, questions concerning bounds on the number of cache misses, and how closely current optimization techniques approach those bounds for stencil operators, remain open.

In this paper we consider improvement of cache usage through maximizing temporal locality in evaluations of explicit stencil operators on structured discretization grids. Our contribution if twofold. First, we prove lower and upper bounds on the number of cache misses for local operators on structured grids. Our lower bound (i.e. the number of unavoidable misses) is based on the discrete isoperimetric theorem. Our upper bound (i.e. the achievable number of misses) is based on a cache fitting algorithm which utilizes a special basis of the grid interference lattice. As shown by example, the lower bound can be achieved in some cases. The second contribution is the identification of grids whose unfavorable dimensions cause significant increases in cache misses. We provide two characterizations of these unfavorable grids. The first one, derived experimentally, states that the product of all relevant grid dimensions is close to a multiple of half the cache size. The second characterization is that the grid interference lattice contains a short vector.

2 Cache model and definitions

We consider a single-level, virtual-address-mapped, set-associative data cache memory, see [7]. The memory is organized in a sets of z lines of w words each. Hence, it can be characterized by the parameter triplet (a, z, w), and its size S equals $a * z * w$ words. A cache with parameters $(a, 1, w)$ is called fully associative, and with parameters $(1, z, w)$ it is called direct-mapped.

The cache memory is used as a temporary fast storage of words used for processing. A word at virtual address A is fetched into cache location $(a(A), z(A), w(A))$, where $w(A) = A \bmod w$, $z(A) = (A/w) \bmod z$, and $a(A)$ is determined according to a replacement policy (usually a variation of *least recently used*). The replacement policy is not important within the scope of this paper.

If a word is fetched, then $w-1$ neighboring words are fetched as well to fill the cache line completely. In practice, a, z, and w are often powers of 2 in order to simplify computation of the location in cache. For example, the MIPS R10000 processor, for which we report some measurements in Section 6, has a cache with parameters (2,512,4), which makes S equal to 4K double precision words, or 32KB.

Our lower bound for the minimum number of cache misses that must be suffered during a stencil computation holds for any cache, including fully associative caches. The upper bound shows that a particular number of cache misses can be achieved by choosing a special sequence of computations. A *cache miss* is defined as a request for a word of data that is not present in the cache at the time of the request. A *cache load* is defined as an explicit request for a word of data for which no explicit request has been made previously (a *cold load*), or whose residence in the cache has expired because of a cache load of another word of data into the exact same location in the cache (a *replacement load*). The definitions of cold and replacement loads are analogous to those of cold and replacement cache misses [4], respectively, and if w equals 1 they completely coincide.

If a piece of code features ϕ cache misses and μ cache loads, it can easily be shown that

$\mu \leq w\phi$. A code with good spatial locality typically has $\mu \approx w\phi$. As can be shown by a simple example, no bound of the form $\phi \leq c\mu$ (c being a constant) can be derived for arbitrary code segments, but if the code implements a nonredundant stencil operation, we have $\phi \leq |K|\mu$, where $|K|$ is the total number of points within the stencil. This is shown as follows. Let the stencil operation be written as $q(\mathbf{x}) = Ku(\mathbf{x})$, with $\mathbf{x} \in \Omega$. Here Ω is the (not necessarily contiguous) point set on which array q is evaluated. Let $\overline{\Omega}$ be the K-extension of Ω, which is the point set on which u must be defined in order to compute q at all points of Ω. The total distinct number of elements of u used is $|\overline{\Omega}|$. The number of cache misses ϕ does not exceed the total number of accesses to array u (may include repeated accesses to the same element), which equals $|K||\Omega|$, so $|\overline{\Omega}| \leq |K||\Omega|$. Consequently, we have the following interval inequality: $|K|^{-1} \leq \frac{\mu}{\phi} \leq w$, which can be used to bound the number of cache misses in terms of the number of cache loads.

3 A lower bound for cache loads for local operators

In this section we consider the following problem: for a given d-dimensional structured grid and a local stencil operator K, how many cache loads must be incurred in order to compute $q = Ku$, where q and u are two arrays defined on the grid. We will provide a lower bound which asserts that, regardless of the order in which the grid points are visited for the computation of q, at least μ cache loads have to occur.

We use the following terminology to describe the operator K. The vectors $\mathbf{k}_1,\ldots,\mathbf{k}_s$, defined such that $q(\mathbf{x})$ (the value of q at the grid point identified by the vector $\mathbf{x}$) is a function of the values $u(\mathbf{x}+\mathbf{k}_1),\ldots,u(\mathbf{x}+\mathbf{k}_s)$, are called *stencil vectors*. Locality of K means that the stencil vectors are contained in a cube $\{\mathbf{k}\,|\,|k_i| \leq r, i = 1,\ldots,d\}$ (r is called the *radius* of K, and $2r+1$ its *diameter*). In this section we assume that K contains only the star stencil (i.e. the $\{\mathbf{0}, \mathbf{e}_1, \ldots, \mathbf{e}_d, -\mathbf{e}_1, \ldots, -\mathbf{e}_d\}$ stencil). A lower bound on the number of cache loads for the star stencil will give us a lower bound for any stencil containing it.

Let q be computed in the K-interior R of a rectangular region (a *grid*) G. We assume that computation of q is performed in a pointwise fashion, that is, at any grid point the value of q is computed completely before computation of the value of q at another point is started. In order to compute the value of q at a grid point $\mathbf{x}$, the values of u at the neighbor points of $\mathbf{x}$ must be loaded into the cache (a point $\mathbf{y}$ is a neighbor of $\mathbf{x}$ if $\mathbf{y}-\mathbf{x}$ is a stencil vector of K). If $\mathbf{x}$ is a neighbor of $\mathbf{y}$ and $u(\mathbf{y})$ has been loaded in cache to compute $q(\mathbf{z})$ but is dropped from the cache before $q(\mathbf{x})$ is computed, then $u(\mathbf{y})$ must be reloaded, resulting in a replacement load associated with $\mathbf{x}$.

To estimate the number of elements, ρ, of array u that must be replaced, we choose a partition of R into a disjoint union of k grid regions R_i, with $R = \cup_{i=1}^{k} R_i$, in such a way that q is computed at all points of R_i before it is computed at any point of R_{i+1}, see Figure 1. Let B_{ij} be the set of points in R_j which are neighbors of R_i. Since the star stencil is symmetrical, the points of B_{ij} are neighbors of B_{ji}. Because any point of B_{ij} can have at most $2d$ neighbors in B_{ji}, we have the following inequalities:

$$\frac{1}{2d}|B_{ij}| \leq |B_{ji}| \leq 2d|B_{ij}|\,. \tag{1}$$

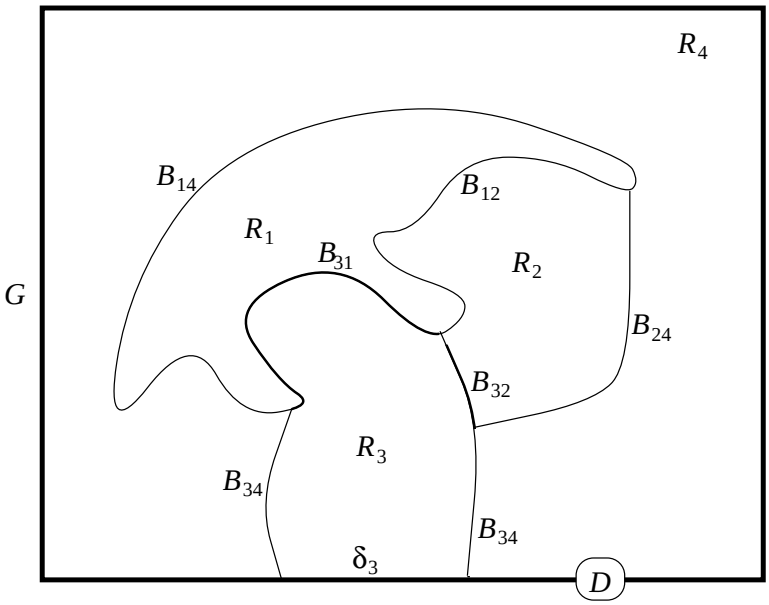


Figure 1: The boundaries B_{ij} of already computed values of q in a sequence of regions R_i. Reloading of some values of u on the boundary of R_3 (heavy lines) results in at least $\max(|B_{31}| + |B_{32}| - S, 0)$ cache loads.

For computation of q in R_i we have to replace at least ρ_i values of u, where ρ_i equals $\max\left(\sum_{j=1}^{i-1} |B_{ij}| - S, 0\right)$. The total number of replaced values in the course of computing q on the entire grid will be at least ρ, where ρ equals $B - kS$, and B equals $\sum_{i=1}^{k}\sum_{j=1}^{i-1} |B_{ij}|$. Summing all the terms $|B_{ij}|$, taking into account Equation (1) and the fact that $|B_{ii}| = 0$, we get:

$$B \geq \frac{1}{4d}\sum_{i=1}^{k}\sum_{j=1}^{k} |B_{ij}| \,. \tag{2}$$

Let δR_i be the exterior boundary of R_i, that is, all points neighbor to R_i not belonging to R_i, and let δ_i be the subset of the grid boundary D having neighbors only in R_i. Here, D is defined as $G \setminus R$. Obviously, $\delta_i \cap \delta_j = \emptyset$ if $i \neq j$, and $\sum_{j=1}^{k} |B_{ij}| \geq |\delta R_i| - |\delta_i|$.

Now we choose the R_i such that $|\delta R_i| = \sigma \geq 8dS$, where σ is specified below, and let ν equal $\max |R_i|$. Consequently, we have $k \geq |R|/\nu \stackrel{\text{def}}{=} V/\nu$, and

$$\rho \geq \frac{1}{4d}\sum_{i=1}^{k}(|\delta R_i| - |\delta_i|) - kS = k\left(\frac{\sigma}{4d} - S\right) - \frac{1}{4d}\sum_{i=1}^{k}|\delta_i| \geq \frac{V}{\nu}S - \frac{1}{4d}|D| \,. \tag{3}$$

We subsequently choose σ in such a way that

$$\sigma = |\delta O(d,t)| = \sum_{k=1}^{d} 2^k \binom{d}{k}\binom{t}{k-1} \geq 8dS \tag{4}$$

for some t, where $O(d,t)$ is the standard d-dimensional octahedron of radius t (see Appendix A). It follows from Equation 21, Appendix A, that t can be chosen in such a way that σ is

less than $8d(2d+1)S$. Now the value of ν can be estimated using the isoperimetric property of the octahedron (see again Appendix A), namely: $\nu \leq |O(d,t)|$. Hence, we find

$$\frac{S}{\nu} \geq \frac{\sigma}{8d(2d+1)\nu} \geq \frac{|\delta O(d,t)|}{8d(2d+1)|O(d,t)|} \geq c_d S^{-\frac{1}{d-1}} \tag{5}$$

where c_d equals $1/(d(2d+1)2^{d+2})$. This gives the following lower bound:

$$\rho \geq \frac{V}{8d(2d+1)} \frac{|\delta O(d,t)|}{|O(d,t)|} - \frac{1}{4d}|D| \geq V c_d S^{-\frac{1}{d-1}} - \frac{1}{4d}|D| \,. \tag{6}$$

We also have $V + |D| = |G|$ and $|D| \leq 2d|G|/l$, where l is the smallest size of the grid. This gives the final lower bound μ for the total number of elements of u to be loaded into the cache:

$$\mu \geq V + \rho \geq V \left(1 + c_d S^{-\frac{1}{d-1}} - \frac{1}{2l}\right) \geq |G| \left(1 - \frac{2d+1}{l} + (1 - \frac{2d}{l}) c_d S^{-\frac{1}{d-1}}\right) . \tag{7}$$

In general, assuming that the cache associativity a is larger than the diameter of the operator K, the order of this lower bound can not be improved, as shows the following example (remember that our lower bound is valid for a cache with any associativity, including a fully associative cache). Let the spatial extents of a two-dimensional grid be n_1 and n_2, respectively, with n_1 equal to kS and n_2 arbitrary, and perform calculations of the star stencil (i.e. $r = 1$) in the following order:

```
do   i = 0, k*a-1
  do   j = 2, n2-1
    do   i1 = max(2,1+i*(S/a)), min(n1-1,(i+1)*(S/a))
      q(i1,j) = u(i1,j) + ···
    end do
  end do
end do
```

Since n_1 equals kS, all values of q and u having the same value of the second index are mapped into the same cache location within a set. Since a exceeds $2r+1$, none of the values required for the computation of q will be replaced in the cache, except those at a distance r around the lines defined by `i1 = i*S/a`. The total number of elements of u loaded into the cache for execution of this loop nest will therefore be $n_1 n_2 + (n_2 - 2)2r(ka - 1) - 4$, which equals $n_1 n_2 (1 - 2/n_1 + 2a(1 - 2/n_2)/S)$. Similar examples in higher dimensions show that the order of our lower bound (Equation 7) can not be improved.

4 An upper bound for cache loads for local operators. Cache fitting algorithm

In order to obtain an upper bound we present a *cache fitting* algorithm which incurs a small number of replacements. We find a set P of cache conflict-free indices of u and calculate Ku

at the points of P. Then we tile the index space of u with P to minimize the total number of replacements. For the analysis we assume a cache associativity of one, which is the worst case for replacement loads.

Let L be a set in the index space of u having the same image in cache as the index $(0,\ldots,0)$, Figure 2. L is a lattice in the sense that there is a generating set $\{\mathbf{b}_i\}$, $i=1,\ldots,d$, such that L is the set of grid points $\{(0,\ldots,0)+\sum_{i=1}^{d} x_i\mathbf{b}_i \mid x_i \in Z\}$. We call this the *interference lattice* of u. It can be defined as the set of all vectors $(i_1,\ldots,i_d)$, such that

$$(i_1 + n_1 i_2 + n_1 n_2 i_3 + \cdots + n_1 \cdots n_{d-1} i_d) \bmod S = 0\,. \tag{8}$$

In [4] this lattice is defined as the set of solutions to the cache miss equation.

Let P be a fundamental parallelepiped[1] of L. For future reference we note that $\mathrm{vol}(P) = \det L = S$. The second equality follows form the fact that L has a basis $\{\mathbf{v}_i\}$ of the form:

$$\mathbf{v}_1 = S\mathbf{e}_1, \quad \mathbf{v}_i = -m_i\mathbf{e}_1 + \mathbf{e}_i\,, 2 \le i \le d\,, m_{i+1} = \prod_{j=1}^{i} n_j\,. \tag{9}$$

Obviously, the vectors $\mathbf{v}_i$ satisfy Equation 8. Conversely, any vector satisfying Equation 8 can be represented as a linear combination of $\mathbf{v}_1,\ldots,\mathbf{v}_d$, with coefficients $x_k = i_k$ for $k=2,\ldots,d$, and $x_1 = (i_1 + m_2 i_2 + \cdots + m_d i_d)S^{-1}$. x_1 is an integer number, since $i_1 + m_2 i_2 + \cdots + m_d i_d$ is divisible by S according to Equation 8. Since $\mathbf{v}_1,\ldots,\mathbf{v}_d$ are linearly independent vectors, they form a basis of the lattice.

Let F be a face of P (see Figure 2), and let $\mathbf{v}$ be a basis vector of L such that $P = \{\mathbf{f} + x\mathbf{v} \mid \mathbf{f} \in F, 0 \le x < 1\}$. Then shifts $F + (k/g)\mathbf{v}$, $k = \ldots,-1,0,1,\ldots$ contain all integer points of a pencil Q, with $Q = \{\mathbf{f} + x\mathbf{v} \mid \mathbf{f} \in F, x \textit{ is any number}\}$ for an appropriate value of g[2]. The values of q at the points of Q can be computed without replacing reusable values of u, except at a distance of r or less from the boundary of Q. Let $h_1,\ldots,h_s$ be the signed projections along F of the stencil vectors of K onto $\mathbf{v}$, and let h_+ and h_- be the maximum and the minimum of the projections, respectively. We assume also that $|h_+ - h_-|/g < |\mathbf{v}|a$, meaning that the extent of P in the direction of $\mathbf{v}$ is big enough to allow to compute q on F without replacements. It may be impossible to satisfy this condition when the shortest vector in L is shorter than the diameter of K divided by the cache associativity. Lattices with short vectors are discussed in Section 6. The associated grids are called *unfavorable grids*.

The *Cache Fitting Algorithm* for computing q is as follows (see Figure 2); here $K(R)$ is the set of points where u must be available in order to compute q at all points of R (i.e. the K-extension of R):

```
set  w = (1/g) v
do   Q = Qmin, Qmax
```

[1] A fundamental parallelepiped of a lattice L is a set of points $\{\sum_{i=1}^{d} x_i\mathbf{b}_i \mid 0 \le x_i < 1\}$ for any basis $\{\mathbf{b}_i\}$ of L.

[2] Let I be a fundamental parallelepiped of the integer lattice in the subspace Y generated by F, and let $\mathbf{e}$ be an integer vector such that $\mathbf{e}$ and the basis of I generate Z^d. Obviously, g must be chosen in such a way that $\mathrm{vol}((1/g)\mathbf{v}, I) = \mathrm{vol}(\mathbf{e}, I) = 1$. Hence, $g = \mathrm{vol}(\mathbf{v}, I) = (1/|F|)\mathrm{vol}(\mathbf{v}, F) = \mathrm{vol}(P)/|F|$, where $|F|$ is the index of the lattice $L \cap Y$ in the integer lattice of Y.

```
  determine face F inside pencil Q
  do   k = kmin, kmax
    load in cache all values of u inside K(F + k * w)
    compute q at F + k * w
  end do
end do
```

In this algorithm the parameters `Qmin`, `Qmax`, `kmin` and `kmax` are determined in such a way that the scanning face F sweeps out the entire grid. Whenever a point is not contained in the grid, it is simply skipped in the nest. Since the scanning face, moving in the direction of $\mathbf{v}$ with step g, passes through all integer points of Q, the algorithm computes the values of q at all points inside Q.

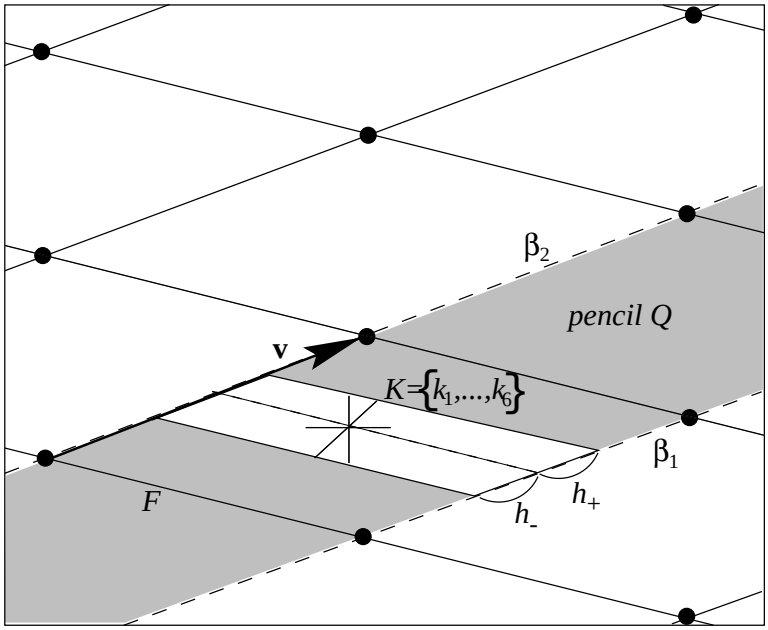


Figure 2: **The Interference Lattice.** Cache fitting set $F+k\mathbf{w}$, $k \in Z$, sweeps across pencil Q in the direction of $\mathbf{v}$. Only values of u at points at a distance r or less from the pencil boundaries β_1 and β_2 will be replaced in the cache when K is evaluated inside of Q.

Replacements misses can occur only at points at a distance of r or less from the boundaries of the pencils. For each of these points at most s replacement need to take place, where s, the size of the stencil, is defined by $s = |K| \leq (2r+1)^d$. So the number of replacements will not exceed $r(2r+1)^d A$, where A is the total surface area of all pencils.

To minimize A we choose P so that Q has a good surface-to-volume ratio. Let P be the fundamental parallelepiped of a reduced basis of L. A basis $\mathbf{b}_1, \ldots, \mathbf{b}_d$ of a d-dimensional lattice L is called reduced if

$$\prod_i ||\mathbf{b}_i|| \leq c_d \det L\,, \tag{10}$$

where c_d is a constant which depends only on d[3]. Let $\mathbf{b}_1$ be the shortest vector of the basis, let the eccentricity e of the basis be defined as $e = \max(||\mathbf{b}_i||/||\mathbf{b}_1||)$, and let ∂P be the surface of P. Now we can derive an estimate for the surface-to-volume ratio of P:

$$\frac{|\partial P|}{\det L} \leq \frac{2\sum_j \prod_{i \neq j} ||\mathbf{b}_i||}{\det L} \leq 2c_d \sum_i \frac{1}{||\mathbf{b}_i||} \leq c'_d \frac{1}{||\mathbf{b}_1||} \leq ec'_d \left(\prod_i ||\mathbf{b}_i|| \right)^{-\frac{1}{d}} \leq ec'_d S^{-\frac{1}{d}}, \quad (11)$$

where we twice used the Hadamard inequality: $\prod_i ||\mathbf{b}_i|| \geq \det L$, the property of reduced bases given by Equation 10, and the abovementioned fact that $\det L = S$. The constant c'_d is defined by $c'_d = 2dc_d$.

A does not exceed the surface area of all fundamental parallelepipeds covering the grid. Since the total number of these parallelepipeds equals $|G|/\det L$, we obtain: $A \leq |\partial P||G|/\det L$, so that the total number of replacements ρ can be bounded by: $\rho \leq r(2r+1)^d |\partial P||G|/\det L$. This, combined with Equation 11, gives an upper bound for the total number of elements to be loaded into the cache in the cache fitting algorithm:

$$\mu \leq |G| + \rho \leq |G| \left(1 + ec''_d S^{-\frac{1}{d}}\right), \quad (12)$$

where c''_d is defined by $c''_d = r(2r+1)^d c'_d$.

Note that if the shortest vector in the interference lattice has length $(S/f)^{1/d}$ for some constant f, it follows that $e < fc_d$. To show this, we sort the basis vectors in Equation 10 in ascending order. Then it follows that $\frac{S}{f}^{\frac{d-1}{d}} ||\mathbf{b}_d|| \leq c_d S$, and hence $e = \frac{||\mathbf{b}_d||}{||\mathbf{b}_1||} \leq fc_d$.

In Appendix B we show that there are grids whose interference lattices feature f's that are independent of S (provided that S is a prime power, which is true in most practical cases). For these lattices the relative gap between the upper bound (Equation 12) and the lower bound (Equation 7) of the previous section goes to zero as S increases. When the cache associativity exceeds the diameter of K, this gap can be closed. In that case a parallelepiped, built on a reduced basis of the interference lattice of the array indices with $x_d = 0$, can be swept in the d^{th} coordinate direction, similar to the example at the end of Section 3. In general, the cache fitting algorithm gives full cache utilization, in contrast to the algorithm for finding grid-aligned parallelepipeds devoid of interference lattice points, as proposed in [4]. See Table 2 in Reference [4], where the sizes of blocks without self interference are approximately 20% smaller than S.

5 Lower and upper bounds for multiple RHS arrays

In this section we consider the case where multiple arrays are involved in the computation of q. Let p be the number of arrays (we call these the *RHS arrays*), all having the same sizes, and let the stencil of each RHS array include the star stencil. This means, in particular, that for each boundary point of any region R_i (see Figure 1) values of p RHS arrays are necessary[4]

[3]Every lattice has a reduced basis. There is a polynomial algorithm to find a reduced basis with a constant $c_d = 2^{d(d-1)/4}$ [11, Ch. 6.2].

[4]As in Section 3, we assume that computation of q is performed in a pointwise fashion. In this case elements of all RHS arrays have to be loaded into cache simultaneously, reducing the cache size effectively

for computation of q in R_i. Hence, we have to replace at least ρ_i RHS array values, with $\rho_i = \max(p(\sum_{j<i} |B_{ij}|) - S, 0)$. Now we can repeat the arguments of Section 3, with $|V|$ and $|G|$ replaced by $p|V|$ and $p|G|$, respectively, and S replaced by $\lceil S/p \rceil$, to obtain the following lower bound for the number of cache loads for stencil computations with p RHS arrays:

$$\begin{aligned} \mu \geq p|V| + \rho &\geq p|V| \left(1 + c_d \left\lceil \frac{S}{p} \right\rceil^{-\frac{1}{d-1}} - \frac{1}{2l} \right) \\ &\geq p|G| \left(1 - \frac{2d-1}{l} + \left(1 - \frac{2d}{l} \right) c_d \left\lceil \frac{S}{p} \right\rceil^{-\frac{1}{d-1}} \right) . \end{aligned} \tag{13}$$

In order to obtain an upper bound on the number of cache loads for calculations with p RHS arrays, we assume that we are free to choose relative array offsets. Our upper bound is valid under the assumption that the stencil diameter divided by the cache associativity is smaller than the length of the longest lattice basis vector divided by p. Consider a stripwise tiling of the fundamental parallelepiped P of the lattice L, see Figure 3. Each tile P_i has the same size and shape. The size is determined by considering the longest edge vector $\mathbf{v}$ in the fundamental parallelepiped and dividing it into p equal pieces of size $[(S/|F|)/p]||\mathbf{v}||$, so that each tile contains $|F|[(S/|F|)/p]$ integer points, where $|F|$ is the number of integer points in face F. The remainder part of the tiling is indicated by the shaded area. The reason why the longest edge vector is selected for subdivision is as follows. Since we use a reduced basis, the smallest angle between $\mathbf{v}$ and F is bounded from below, so the parallelepiped is always close to orthogonal. Therefore, subdividing the longest edge leads to tiles with the largest inscribed sphere, and thus the largest difference stencil fitting inside the tile.

Let $\{P_i\}$ be the parallelepipeds of the tiling, and let s_i be the address offset of P_i relative to P_1. We assign one parallelepiped to each RHS array and choose starting addresses of the arrays, addr_i, in such a way that the image of P_1 of array i coincides with tile P_i, that is, there is no overlap in the cache between images of P_1 of different arrays: $\mathrm{addr}_i = \mathrm{addr}_1 + m_i S + s_i$, where $m_1 = s_1 = 0$, and $m_i = m_{i-1} + \lceil \frac{|G| - s_i + s_{i-1}}{S} \rceil$, $i = 2, \ldots, p$. Sweeping through the pencil by units of the size of tile P_1 in the direction of $\mathbf{v}$, we can compute Ku without any cache conflicts, except on the boundaries of the pencils. The number of replacement loads of this algorithm can be estimated similarly to the number of replacement loads of a single-array algorithm, taking into account that for calculation of a value u at any point values of all p RHS arrays in the neighbor points may have to be in cache, thus reducing the effective cache size to $[S/p]$:

$$\mu \leq p|G| + \rho \leq p|G| \left(1 + e c_d'' \left\lceil \frac{S}{p} \right\rceil^{-\frac{1}{d}} \right) \tag{14}$$

where c_d'' is a constant which depends only on d, and e is the eccentricity of L.

by a factor of p. Non-pointwise computations may be performed if the operator K is separable, in the sense that it can be written as $K(u_1, u_2, \ldots, u_p) = K_1(u_1, K_2(u_2, \ldots K_p(u_p) \ldots)$. In the case of separability of K, the stencil operation can be split into a succession of independent operations, each involving an intermediate value of q and one RHS array. This would not require to load all p RHS arrays in cache at each point. Instead, it would suffice to write intermediate values of q into main memory, and then load them back into cache for completion of the computations. This results in a larger effective cache size, but more data to be loaded, so splitting the operation need not improve the total number of loads.

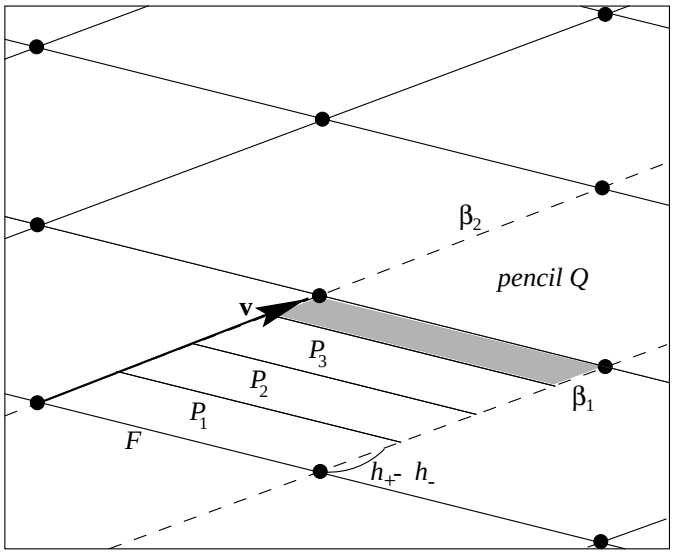


Figure 3: Tiling of the fundamental parallelepiped of a reduced basis of the lattice L. We assume that the projected stencil size $|h_+ - h_-|$ does not exceed $a|\mathbf{v}/p|$ (a is the cache associativity). The tiling effectively reduces the size of the parallelepiped by a factor of at most $2p$ (since $x/p \geq \lfloor x/p \rfloor \geq x/(2p)$), and it increases the cost of a replacement in the cache per point of the boundary of the pencil by at most a factor of p, since elements of all p RHS arrays will be replaced at the same time.

6 Unfavorable array sizes

We have implemented our cache fitting algorithm and compared its measured number of cache misses with that of the compiler-optimized code for the corresponding naturally ordered loop nest on a MIPS R10000 processor (SGI Origin 2000). For comparison we chose a second-order difference operator (the common 13-point star stencil) and a test set including three-dimensional grids of sizes $40 \leq n_1 < 100$, $n_2 = 91$, and $n_3 = 100$ (the value of the second dimension was chosen to show a typical picture; that of the third dimension is irrelevant). A plot of measured cache misses for both codes is shown in Figure 4. The program was compiled with options "`-O3 -LNO:prefetch=0`," using the MIPSpro f77 compiler, version 7.3.1.1m. The prefetch flag disables the prefetching compiler optimization. Without this option the number of cache misses increases significantly, because the compiler does aggressive prefetching to try to reduce execution time.

The upper bounds for the cache misses from the previous sections would suggest that the number of replacement cache misses will increase in the cases where the interference lattice has a very short vector. Very short means that the length is smaller than the diameter of the operator divided by the cache associativity. In this case the self interference would increase significantly. This result suggests how to pad arrays to improve cache performance: the padding should be organized in such a way that the shortest vector in the lattice is not too short, though short enough to minimize the number of pencils (large index of scanning face F). The sweeping is organized such that pencils are as wide as possible (i.e. the smallest

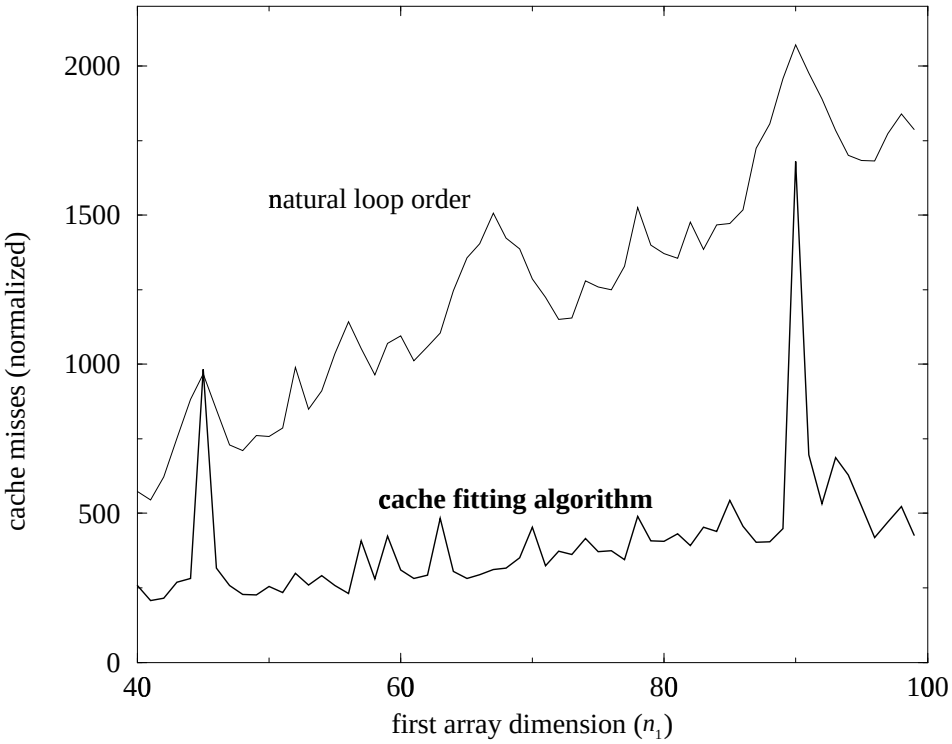


Figure 4: Plot of measured cache misses for $40 \leq n_1 < 100$, $n_2 = 91$ for 13-point star stencil. The thin line corresponds to the naturally ordered nest, optimized by the SGI Fortran compiler. The heavy line corresponds to our cache fitting algorithm. A typical ratio between the two is 3.5. The large fluctuations exhibited by the cache fitting algorithm correspond to grids with short lattice vectors ($n_1 = 45$ and $n_1 = 90$ yield shortest vectors $(1, 0, 1)$ and $(2, 0, 1)$, respectively). For such unfavorable grids the cache misses incurred by the cache fitting algorithm may outnumber those of the compiler-optimized loop nest.

total number of pencils), while avoiding—in the case of multiple RHS arrays—tiles that are thinner than the diameter of the stencil operator divided by the cache associativity.

To demonstrate these unfavorable grids we again choose the second-order stencil and force computations in the nest to follow the natural order[5]. Figure 5 shows the correlation between spikes in the number of cache misses and the presence of a very short vector in the lattice. We call these lattices unfavorable for cache utilization. Arrays having such lattices should be avoided on the target machine. When the shortest vector of the interference lattice is shorter than the diameter of the operator, the number of cache misses sharply increases. The application developer should avoid such unfavorable array sizes, and compilers should avoid these sizes using appropriate padding of array dimensions. Note that similar unfavorable cache effects are mentioned in [1].

[5]This forcing is accomplished by introducing a dependence through a Fortran subroutine that performs a circular shift of its arguments

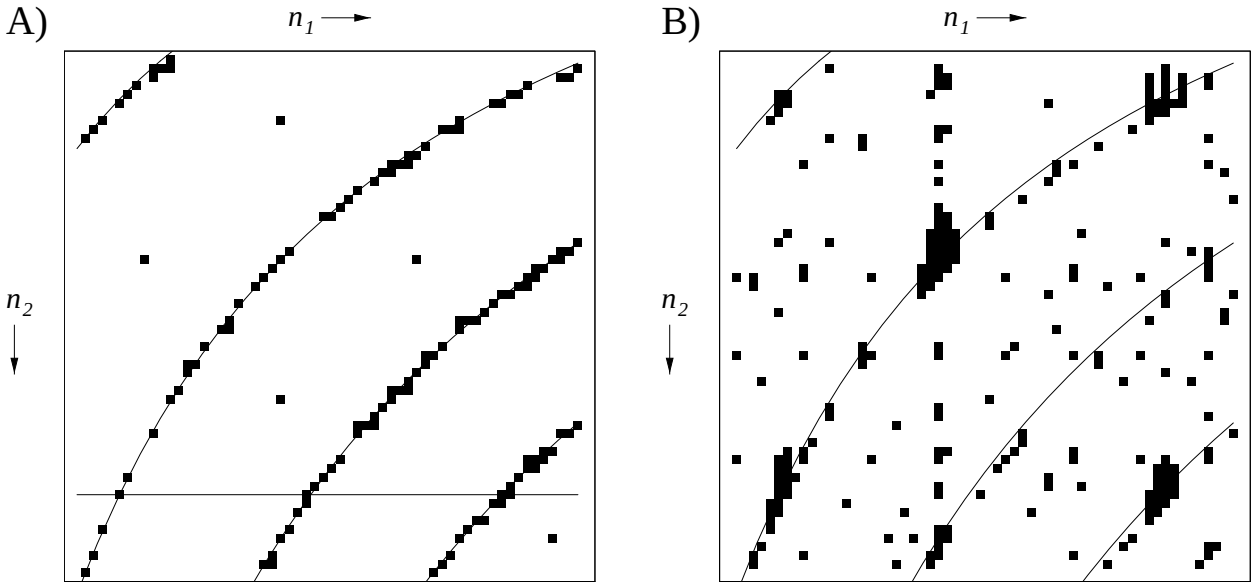


Figure 5: Plot A shows measured fluctuations of cache misses (above 15% of the upper bound). Plot B shows the interference lattices with short (less than 8 in the L^1 norm) vectors. Array sizes are $40 \leq n_1, n_2 < 100$. The plots can be fitted well by hyperbolae defined by $n_1 n_2 = \frac{1}{2} kS, k = 1, 2, 3, 4$, meaning that arrays with unfavorable size are those whose z-slices are (close to) multiples of half the cache size. The horizontal line in Plot A shows the position of the graph from Figure 4.

7 Conclusions and future work

We have demonstrated tight lower and upper bounds for cache misses for calculations of an explicit operator K on a structured grid. Our lower bound is valid in the general case of fully associative caches, and is based on a discrete isoperimetric theorem. Our upper bound is based on a cache fitting algorithm which uses the fundamental parallelepiped of a special basis of the interference lattice to fit the data in the cache. The upper bound assumes that the shortest vector in the interference lattice is not too short. We have shown that there are grids whose interference lattices have this property. We have also shown that the presence of a very short vector in the lattice correlates with fluctuations of actual cache misses for calculation of a second-order explicit operator on three-dimensional grids. The fluctuations occur on grids with unfavorable sizes, i.e. on those whose product of the first two dimensions is (close to) a multiple of half the cache size.

Our results can be extended straightforwardly to implicit stencil computations (i.e. those of the form $q \leftarrow K(q)$) when the problem has a one-dimensional data dependence. Such a data dependence exists if computations of q at grid points can take place in an arbitrary order, except that there is a single index i for which $q(x_1, \ldots, i, \ldots, x_d)$ must be evaluated before $q(x_1, \ldots, i + \alpha, \ldots, x_d)$ can be calculated (the constant α is either +1 or -1). Clearly, the lower bound is not affected by the implicitness of K. The previously derived upper bound can still be achieved by prescribing the proper visiting order of points within each

parallelepiped, the scanning face direction within each pencil (positive or negative sweep direction), and the visiting order of subsequent pencils. This is always possible for a one-dimensional data dependency.

Our results can also be extended to arrays that store more than one word per grid point (tensor arrays). The lower bound of Section 3 for operations with multiple right hand sides immediately applies to tensor arrays. The upper bound of that section also applies, provided the tensor components can be stored as independent subarrays.

In a future study we plan to extend the results of this paper to more general implicit operators, to operators on unstructured grids, and to tensor arrays with restricted storage models. We intend to study more closely the dependence of cache misses on the size of the operator's stencil. We also plan to enhance the presented results by taking into account a secondary cache and TLB, and to formulate bounds for cache misses more directly than through the determination of cache loads.

Appendix A: The simplex and the octahedron

In this section we list some basic facts on the number of integer points in the octahedron and simplex. The standard octahedron is defined as:

$$O(d,t) = \left\{ \mathbf{x} \in Z^d \mid \sum_{i=1}^{d} |x_i| \leq t \right\} \tag{15}$$

and the standard simplex as:

$$S(d,t) = \left\{ \mathbf{x} \in Z^d \mid 0 \leq x_1, \ldots, x_d \,, \sum_{i=1}^{d} |x_i| \leq t \right\} . \tag{16}$$

If we consider sections of the octahedron by planes $x_1 = k$, $k = -t, \ldots, t$, then for the number of integer points in the octahedron we get the following recurrence relation:

$$|O(d,t)| = |O(d-1,t)| + 2 \sum_{k=0}^{t-1} |O(d-1,k)| \,. \tag{17}$$

This relation can be used to prove that

$$|O(d,t)| = \sum_{k=0}^{d} 2^k \binom{d}{k} \binom{t}{k} \tag{18}$$

and that

$$|\delta O(d,t-1)| = |O(d,t) - O(d,t-1)| = \sum_{k=1}^{d} 2^k \binom{d}{k} \binom{t-1}{k-1} . \tag{19}$$

Also, the relation

$$|\delta O(d,t)| = |\delta O(d,t-1)| + |\delta O(d-1,t)| + |\delta O(d-1,t-1)| \tag{20}$$

shows that

$$|\delta O(d,t)| \leq (2d+1)|\delta O(d,t-1)|\,. \tag{21}$$

For the number of integer points in the simplex we have the following recurrence relation:

$$|S(d,t)| = |S(d-1,t)| + |S(d,t-1)|\,. \tag{22}$$

This can be used to prove that (cf. [6], Table 169, see also [12], Section 5):

$$|S(d,t)| = \sum_{k=1}^{d} \binom{d}{k}\binom{t}{k} = \binom{d+t}{d}\,. \tag{23}$$

From Equations 18 and 21 it follows that $|O(d,t)| \leq 2^d|S(d,t)|$. Also, since $\delta O(d,t-1)$ contains at least two nonoverlapping simplices $S(d-1,t)$, and can be covered by 2^d such simplices, we see that

$$2|S(d-1,t)| \leq |\delta O(d,t-1)| \leq 2^d|S(d-1,t)|\,,\ d \geq 2\,. \tag{24}$$

Hence, if $|S(d-1,t)|$ equals S, we have for $d \geq 2$:

$$\frac{|\delta O(d,t)|}{|O(d,t)|} \geq \frac{|\delta O(d-1,t)|}{|O(d,t)|} \geq \frac{2|S(d-1,t)|}{2^d|S(d,t)|} = 2^{-d+1}\left(1+\frac{t}{d}\right)^{-1} \geq 2^{-d+1}S^{-\frac{1}{d-1}} \tag{25}$$

since from Equation 23 it follows that if $|S(d-1,t)|$ does not exceed S, then $1+t/d$ does not exceed $S^{1/(d-1)}$.

The *isoperimetric inequality* [12], Theorem 2, asserts that the size of the boundary of a subset R in Z^d is at least as big as the size of the standard sphere that contains $|R|$ points[6]. It is easy to see that any standard sphere is sandwiched between two standard octahedrons whose radii differ by one. Since the octahedron has the largest volume for a given fixed-size boundary, Inequality 25 is true for any lattice body with a boundary of size S.

Appendix B: The existence of grids with favorable lattices

In order to prove that for every cache of size $S = p^n$, where p is a prime number, there are grids with interference lattices whose shortest vector length exceeds $(S/f)^{1/d}$, with f independent of S, we show:

a. For every dimensionality d there exists a lattice L of the same dimension whose basis has the form given in Equation 9 (Section 4), and whose shortest vector is sufficiently long, and

b. a grid can be constructed that has L as its interference lattice.

[6] The standard sphere, defined in [12], is the integer point set of minimal surface area for any given number of interior points.

Corollary: Since grids with dimensions $n_i + k_i S$, $i = 1, \ldots, d$ have the same interference lattice for any non-negative integers k_i, any grid can be embedded in a favorable larger grid.

Proof:

Step a: Let a lattice L have a basis of the form of Equation 9. Any lattice vector $\mathbf{x}$ with L^∞ norm at most l must be a solution of the following system of inequalities:

$$\left.\begin{array}{c} |x_i| \le l\,, i = 2, \ldots, d \\ |Sx_1 + m_2 x_2 + \cdots + m_d x_d| \le l \end{array}\right\}. \tag{26}$$

Existence of a solution to this system is equivalent to that of the system

$$\left.\begin{array}{c} |x_i| \le l\,, i = 2, \ldots, d \\ \left\| \frac{m_2}{S} x_2 + \cdots + \frac{m_d}{S} x_d \right\| \le \frac{l}{S} \end{array}\right\} \tag{27}$$

where $\|.\|$ denotes the distance to the nearest integer number. Theorem VIII, Ch. 1 [2], states that there are real numbers $\mu_2, \ldots, \mu_d$, and a constant c_d''' depending only on d, such that

$$\|\mu_2 x_2 + \cdots + \mu_d x_d\| \ge \frac{c_d'''}{l^{d-1}} \tag{28}$$

for all nonzero $\mathbf{x}$ satisfying $|x_i| \le l\,, i = 2, \ldots, d$.

If we choose the nonzero integers m_i in such a way that $|m_i - S\mu_i| \le 2$ for $i = 2, \ldots, d$, then we get

$$\left\| \frac{m_2}{S} x_2 + \cdots + \frac{m_d}{S} x_d \right\| \ge \frac{c_d'''}{l^{d-1}} - (d-1)\frac{l}{S} = \left(\frac{S}{l^d} c_d''' - (d-1) \right) \frac{l}{S} \tag{29}$$

which shows that Equation 26 has no integer solutions if $l < (c_d'''/(Sd))^{1/d}$. Hence, f in Section 4 can be chosen as: $f = d/c_d'''$, and the lattice with the basis given by Equation 9 has a reduced basis with eccentricity depending only on d.

Step b: In order to find a grid whose interference lattice is L, we first sort the m_i in order of increasing $\gcd(m_i, S)$. Since we assume that $S = p^n$, where p is prime, we know that $\gcd(m_i, S)$ divides $\gcd(m_{i+1}, S)$, and the appropriate grid dimensions n_i can be found directly by solving the congruencies $(n_i m_i - m_{i+1}) \bmod S = 0$.

Acknowledgements

We are grateful to professor Leonid Khachijan for some help with the theoretical part of the paper, and to Jerry Yan and Henry Jin for discussions on the practical aspects of the paper. This work is partially supported by the HPCC/CAS NASA program, and was executed under NAS task order A61812D.

References

[1] D.H. Bailey. *Unfavorable Strides in Cache Memory Systems.* Scientific Programming, Vol. 4, pp. 53–58, 1995

[2] J.W.S. Cassels. *Introduction to Diophantine Approximations.* Cambridge Univ. Press, 1957 (Ch. I, Theorem VIII)

[3] S. Coleman, K.S. McKinley. *Tile Size Selection Using Cache Organization and Data Layout.* In Proc. SIGPLAN '95, Conf. on Programming Language Design and Implementation, June 1995, pp. 279–289

[4] S. Gosh, M. Martonosi, S. Malik. *Cache Miss Equations: An Analytical Representation of Cache Misses.* ACM ICS 1997, pp. 317–324

[5] S. Gosh, M. Martonosi, S. Malik. *Automated Cache Optimization using CME Driven Diagnostics.* ACM ICS 2000, 11 p.

[6] R. Graham, D. Knuth, O. Patashnik. *Concrete Mathematics.* Addison-Wesley, 1989

[7] J.L. Hennessy, D.A. Patterson. *Computer Organization and Design.* Morgan Kaufmann Publishers, San Mateo, CA, 1994

[8] J.W. Hong, H.T. Kung. *I/O Complexity: The Red-Blue Pebble Game.* IEEE Symposium on Theoretical Computer Science, 1981, pp. 326-333

[9] G. Rivera, C.W. Tseng. *Data Transformations for Eliminating Conflict Misses.* PLDI 1998, pp. 38–49

[10] G. Rivera, C.W. Tseng. *Eliminating Conflict Misses for High Performance Architectures.* ACM ICS 1998, pp. 353–360

[11] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, 1986

[12] D.L. Wang, P. Wang. *Discrete Isoperimetric Problems.* SIAM J. Appl. Math., Vol. 32, pp. 860–870, 1977

[13] M.E. Wolf, M. Lam. *A Data Locality Optimizing Algorithm.* In Proc. SIGPLAN '91, Conf. on Programming Language Design and Implementation, June 1991, pp. 30–44