Interference Lattice-based Loop Nest Tilings for Stencil Computations

*Rob F. Van der Wijngaart and Michael Frumkin**

1 Introduction

A common method for improving performance of stencil operations on structured multi-dimensional discretization grids is loop tiling. Tile shapes and sizes are usually determined heuristically, based on the size of the primary data cache. We provide a lower bound on the numbers of cache misses that must be incurred by any tiling, and a close achievable bound using a particular tiling based on the grid interference lattice. The latter tiling is used to derive highly efficient loop orderings.

The total number of cache misses of a code is the sum of (necessary) cold misses and misses caused by elements being dropped from the cache between successive loads (replacement misses). Maximizing temporal locality is equivalent to minimizing replacement misses. Temporal locality of loop nests implementing stencil operations is optimized by tilings that avoid data conflicts. We divide the loop nest iteration space into conflict-free tiles, derived from the *cache miss equation* [5]. The tiling involves the definition of the grid interference lattice—an equivalence class of grid points whose images in main memory map to the same location in the cache—and the construction of a special basis for the lattice. Conflicts only occur on the boundaries of the tiles, unless the tiles are too thin. We show that the surface area of the tiles is bounded for grids of any dimensionality, and for caches of any associativity, provided the eccentricity of the fundamental parallelepiped (the tile spanned by the basis) of the lattice is bounded. Eccentricity is determined by two factors, aspect ratio and skewness. The aspect ratio of the parallelepiped can

*Computer Sciences Corporation; M/S T27A-1, NASA Ames Research Center, Moffett Field, CA 94035-1000; e-mail: {wijngaar,frumkin}@nas.nasa.gov, Numerical Aerospace Simulation Systems Division, NASA Ames Research Center

be bounded by appropriate array padding. The skewness can be bounded by the choice of a proper basis. Combining these two strategies ensures that pathologically thin tiles are avoided.

They do not, however, minimize replacement misses per se. The reason is that tile visitation order influences the number of data conflicts on the tile boundaries. If two adjacent tiles are visited successively, there will be no replacement misses on the shared boundary. The iteration space may be covered with pencils larger than the size of the cache while avoiding data conflicts if the pencils are traversed by a scanning-face method. Replacement misses are incurred only on the boundaries of the pencils, and the number of misses is minimized by maximizing the volume of the scanning face, not the volume of the tile.

We present an algorithm for constructing the most efficient scanning face for a given grid and stencil operator. In two dimensions it is based on a continued fraction algorithm. In three dimensions it follows Voronoi's successive minima algorithm [10]. We show experimental results of using the scanning face, and compare with canonical loop orderings.

Related work

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) [2, 5, 6, 12], improvements in data locality (i.e. spatial locality) [12], and reductions in conflicts in data accesses [5, 6, 8, 9]. 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 multiplication algorithms are given in [7]. However, questions concerning bounds on the number of cache misses, and how closely current optimization techniques approach those bounds for stencil operators, remain open.

2 Cache model and definitions

We consider a single-level, virtual-address-mapped, set-associative data cache, organized in a sets of z lines of w words each. Hence, it is 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 *fully associative*, and with parameters $(1, z, w)$ it is *direct-mapped*.

The cache 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 5, has a cache with parameters (2,512,4), which makes S equal to 4K double precision words, or 32KB.

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 misses [5], respectively, and if w equals 1 they completely coincide.

If the computation of a stencil operator containing $|K|$ points features ϕ cache misses and μ cache loads, it can easily be shown [3] that the following interval inequality holds: $|K|^{-1} \leq \frac{\mu}{\phi} \leq w$, which can be used to bound the number of cache misses in terms of the number of cache loads. For codes with good spatial locality we typically have $\mu \approx w\phi$.

We assume throughout that no useful data is in the cache at the beginning of the stencil computation, which is realistic for most scientific programs.

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 provide a lower bound which asserts that, regardless of the order in which the grid points are visited to compute q, at least μ cache loads have to occur.

The following terminology describes 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 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 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 $\mathbf{x}$.

It can be shown that

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

where $|G|$ is the size of the grid and l its smallest dimension. S is the size of the cache in words, and c_d a constant that depends only on the dimensionality of the array: $c_d = 1/(d(2d+1)2^{d+2})$. The leading term of $|G|$ in (1) represent inevitable

cold loads. The proof, given in [3], revolves around covering the iteration space R with a disjoint union of k grid regions 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. B_{ij} is the set of points in R_j which are neighbors of R_i. If we assume that all interior points

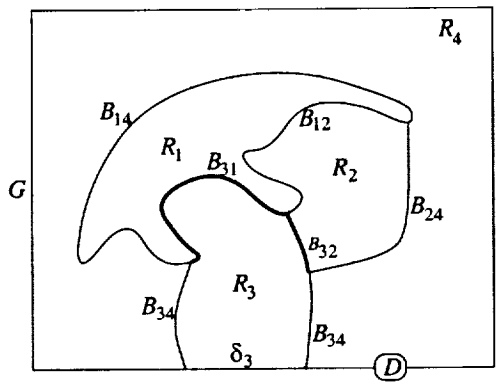

Figure 1. *The boundaries B_{ij} of already computed values of q in a sequence of regions R_i. Reloading of some values of u on the boundary of R_3 (heavy lines) results in at least* $\max(|B_{31}| + |B_{32}| - S, 0)$ *cache loads.*

of R_i can be computed without cache conflicts, the only replacement loads occur for points on the boundary between R_i and R_{i-1}. The number of such loads is minimized by choosing R_i as large as possible for a given surface area δR_i. For the star stencil this is achieved by the diamond-shaped *standard octahedron.*

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 (the 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 can be given in higher dimensions.

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

To obtain an upper (= achievable) 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 $\{(0,\ldots,0)+\sum_{i=1}^{d} x_i\mathbf{b}_i \,|\, x_i \in Z\}$. 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

$$(i_1 + n_1 i_2 + n_1 n_2 i_3 + \cdots + n_1 \cdots n_{d-1} i_d) \bmod S = 0\,. \tag{2}$$

In [5] this lattice is defined as the set of solutions to the cache miss equation.

Let P be a fundamental parallelepiped[1] of L. Let F be a face of P (see Figure 2), and let $\mathbf{b}$ be a basis vector of L such that $P = \{\mathbf{f} + x\mathbf{b} \,|\, \mathbf{f} \in F, 0 \le x < 1\}$. Then shifts $F + (k/g)\mathbf{b}$, $k = \ldots,-1,0,1,\ldots$ contain all integer points of a pencil Q, with $Q = \{\mathbf{f} + x\mathbf{b} \,|\, \mathbf{f} \in F, x \textit{ is any number}\}$ for an appropriate value of g (see [3]). 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. We assume that the extent of P in the direction of $\mathbf{b}$ 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 5. The associated grids are called *unfavorable*.

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) b
do   Q = Qmin, Qmax
  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
```

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

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

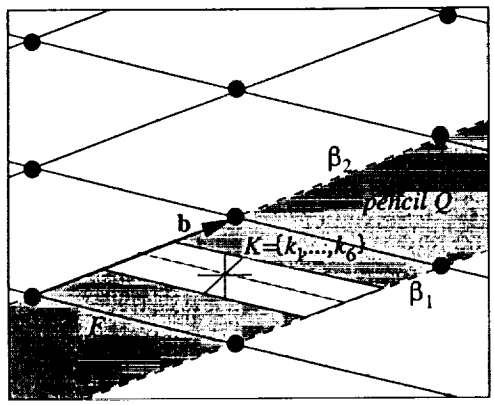

Figure 2. ***The Interference Lattice.*** *Cache fitting set $F + k\mathbf{w}$, $k \in Z$, sweeps across pencil Q in the direction of* $\mathbf{b}$. *Only values of u at points at a distance r or less from the pencil boundaries β_1 and β_2 will be replaced in the cache when K is evaluated inside of Q.*

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 replacements 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. It can be shown that, for the proper choice of P, no more than μ cache loads result when using the cache fitting algorithm:

$$\mu \leq |G| \left(1 + e c'_d S^{-\frac{1}{d}}\right), \tag{3}$$

where $c'_d = 2dr(2r+1)^d 2^{d(d-1)/4}$. The eccentricity e of the basis of the interference lattice is defined by $e = \max(||\mathbf{b}_i||/||\mathbf{b}_j||)$. The proof, given in [3], makes use of the fact that every lattice has a *reduced basis* whose geometrical quality is bounded from below by a factor that only depends on the dimensionality of the grid.

In [3] we show that there exist grids whose interference lattices feature eccentricities that are independent of S (provided that S is a prime power, which is true in most practical cases). Grids whose dimensions are the same modulo S share the same interference lattice. Hence, any grid can be extended (padded) to yield one that has a lattice whose eccentricity is low. For these favorable lattices the relative gap between upper (3) and lower bound (1) 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 (see 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 [5]. See Table 2 in Reference [5], where the sizes of blocks without self interference are approximately 20% smaller than S.

5 Unfavorable array sizes

We compare the measured number of cache misses of our cache fitting algorithm 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 choose the common 13-point star stencil and a test set of three-dimensional grids of sizes $40 \le n_1 < 100$, $n_2 = 91$, and $n_3 = 100$ (the value of the second dimension was chosen to show a typical picture; that of the third dimension is irrelevant). Measured cache misses for both codes are shown in Figure 3. The thin line corresponds to the naturally ordered nest, optimized by the SGI Fortran compiler. The heavy

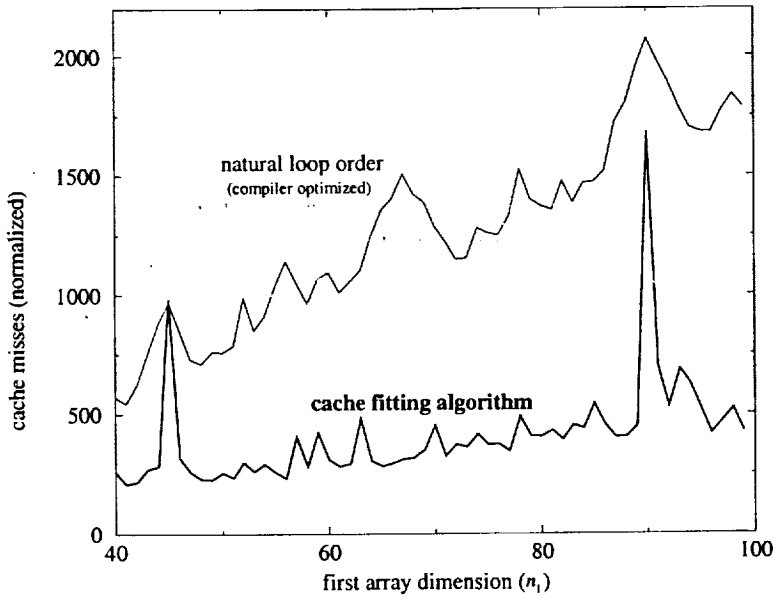

Figure 3. *Measured cache misses on MIPS R10000 processor for* $40 \le n_1 < 100$, $n_2 = 91$ *for 13-point star stencil.*

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. The program was compiled with options "`-O3 -LNO:prefetch=0`," using the MIPSpro f77 compiler, version 7.3.1.1m. We disabled the prefetching compiler

optimization. Otherwise the number of cache misses would increase significantly, because the compiler does aggressive prefetching to try to reduce execution time.

The upper bound for the cache loads from the previous section would suggest that the number of replacement misses will increase in case 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 increases significantly.

To demonstrate these unfavorable grids we again choose the 13-point stencil and force computations in the nest to follow the natural order. Figure 4 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

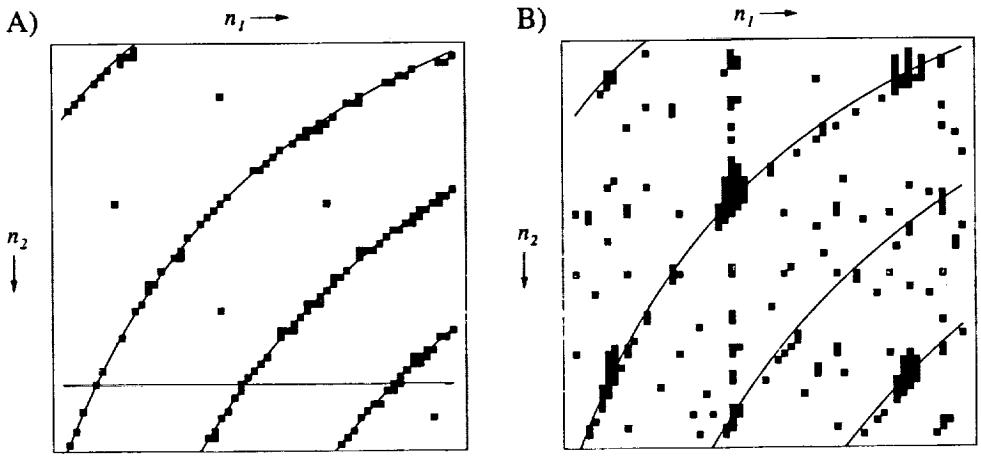

Figure 4. *Plot A shows measured fluctuations of cache misses (above 15% of the upper bound). Plot B shows the interference lattices with short (less than 8 in the L^1 norm) vectors. Array sizes are $40 \leq n_1, n_2 < 100$. The plots can be fitted well by hyperbolae defined by $n_1 n_2 = \frac{1}{2} kS, k = 1, 2, 3, 4$, 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 3.*

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. Note that similar unfavorable cache effects are mentioned in [1].

6 Improvements to cache fitting algorithm

While the cache fitting algorithm described in Section 4 is quite effective in reducing replacement loads, it is not optimal. In deriving it we ignored the order in which

the fundamental parallelepipeds in the grid are visited. Hence, we ignored the replacements that can be avoided by visiting adjacent parallelepipeds, that is, by letting the scanning face traverse pencils unidirectionally. No replacements occur for any point inside the pencil, only on its boundary. Hence, we should minimize the surface area of the pencil, not that of the parallelepiped, by maximizing the number of points in the scanning face. 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 total number of pencils), while avoiding tiles that are thinner than the diameter of the stencil operator divided by the cache associativity.

An second suboptimal property of our original cache fitting algorithm is that it uses non-Cartesian tiles. This results in complicated loop bounds and in reduced reuse of data in the case of non-trivial cache line lengths. More practical tilings are grid-aligned, such as that proposed by Martonosi [5], which attempts to maximize the volume of conflict-free Cartesian cells. This strategy can be improved in a way similar to that described above by minimizing the surface area of the *union* of all tiles that are adjacent in the tile visiting scheme.

Combining the ideas of Cartesian tiles and maximizing scanning face area, we propose the following methods to improve cache reuse in two and three spatial dimensions. In two dimensions choosing conflict-free rectilinear tiles of a given minimum width and maximum height can be done with a continued fraction algorithm, illustrated by Figure 5. Starting point is any pair of lattice vectors v_1, v_2.

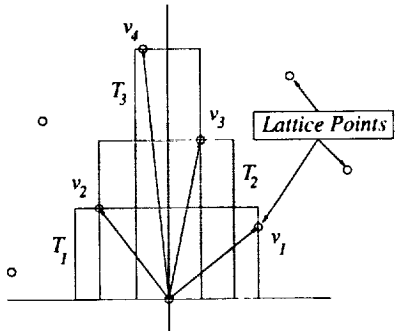

Figure 5. *Continued fraction algorithm for choosing optimal tiles in two dimensions, passing through successive minima* v_1, v_2, v_3, v_4 *and appropriate tiles* T_1, T_2, T_3.

In a rapidly converging sequence of $\mathcal{O}(\log S)$ steps that takes us through successive minima (indicated by v_3, v_4, etc.), a parallelepiped of maximal height, and width at least d/a, is constructed. Multiplying it by a factor of $1/2$ results in a parallelepiped that can be used for conflict-free tiling of the grid, cf. [4], Section 3. It follows from a theorem by Minkovski (see also [4]) that the volume of any tile is at least $S/2$. This is substantially less than the maximal volume of conflict-free tiles, but that

does not matter, since we are using the scanning face method.

In three dimensions the optimal rectilinear tile can be found by application of the Voronoi algorithm [11]. This algorithm, in contrast to the continued fractions algorithm, does involve a search over a space of several alternatives for each step in the scanning face maximization process. However, the exection time of Voronoi algorithm can still be bounded by a polynomial in $\log S$. The volume of tiles generated in this way is at least $S/6$.

7 Conclusions and future work

We have demonstrated tight lower and upper bounds for cache misses for the calculation of explicit operators on structured grids. 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. It can be shown [3] that there are *favorable* grids whose interference lattices have this property, and that any grid can be padded to become favorable. 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.

We also determined that the performance of the cache fitting algorithm can be improved by selecting not the tile with the largest surface-to-volume ratio, but that which has the largest cross section, provided it is not pathologically thin. Adjacent tiles are visited using a scanning face method, which avoids cache loads on tile boundaries in the scan direction. Further practical improvements can be obtained by selecting not the skewed tiles that follow from the cache fitting algorithm, but orthogonal tiles, using a continued fraction (2D) or Voronoi's successive minima (3D) algorithm.

Our results can be extended straightforwardly to stencil computations with multiple rigth hand sides, to non-scalar grid arrays, and to certain implicit stencil computations [3].

In a future study we plan to extend the results of this paper to more general implicit operators, and to tighten the lower bound for cache loads. 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.

Bibliography

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

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

[3] M. Frumkin, R.F. Van der Wijngaart. *Efficient Cache Use for Stencil Operations on Structured Discretization Grids.* NAS Technical Report NAS-00-015, NASA Ames Research Center, Moffett Field, CA, December 2000

[4] M. Frumkin, R.F. Van der Wijngaart. *Using Minimum-Surface Bodies for Iteration Space Partitioning.* See this issue.

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

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

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

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

[9] G. Rivera, C.W. Tseng. *Tiling Optimizations for 3D Scientific Computations.* Proc. Supercomputing 2000, Dallas, TX, November 2000

[10] R. Scheidler, A. Stein. *Voronoi's Algorithm in Purely Cubic Congruence Function Fields of Unit Rank 1.* Mathematics of Computation, **69**, No. 231, 1999, pp. 1245–1266

[11] H.C. Williams, G. Cormak, E. Seah. *Calculation of the Regulator of a Pure Cubic Field.* Mathematics of Computation, **34**, No. 150, 1980, pp. 567–611.

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