Cache Accurate Time Skewing in Iterative Stencil Computations

Conference Paper · October 2011

DOI: 10.1109/ICPP.2011.47 · Source: IEEE Xplore

4 authors, including:

Mohammed Shaheen
IBM Deutschland GmbH
8 PUBLICATIONS **162** CITATIONS

SEE PROFILE

Cache Accurate Time Skewing in Iterative Stencil Computations

Robert Strzodka*, Mohammed Shaheen*, Dawid Pająk† and Hans-Peter Seidel*
* Max Planck Institut Informatik, Saarbrücken, Germany, {strzodka,mshaheen,hpseidel}@mpi-inf.mpg.de
† West Pomeranian University of Technology, Szczecin, Poland, dpajak@mpi-inf.mpg.de

***Abstract*—We present a time skewing algorithm that breaks the memory wall for certain iterative stencil computations. A stencil computation, even with constant weights, is a completely memory-bound algorithm. For example, for a large 3D domain of 500^3 doubles and 100 iterations on a quad-core Xeon X5482 3.2GHz system, a hand-vectorized and parallelized naive 7-point stencil implementation achieves only 1.4 GFLOPS because the system memory bandwidth limits the performance. Although many efforts have been undertaken to improve the performance of such nested loops, for large data sets they still lag far behind synthetic benchmark performance. The state-of-art automatic locality optimizer PluTo [1] achieves 3.7 GFLOPS for the above stencil, whereas a parallel benchmark executing the inner stencil computation directly on registers performs at 25.1 GFLOPS. In comparison, our algorithm achieves 13.0 GFLOPS (52% of the stencil peak benchmark).**

We present results for 2D and 3D domains in double precision including problems with gigabyte large data sets. The results are compared against hand-optimized naive schemes, PluTo, the stencil peak benchmark and results from literature. For constant stencils of slope one we break the dependence on the low system bandwidth and achieve at least 50% of the stencil peak, thus performing within a factor two of an ideal system with infinite bandwidth (the benchmark runs on registers without memory access). For large stencils and banded matrices the additional data transfers let the limitations of the system bandwidth come into play again, however, our algorithm still gains a large improvement over the other schemes.

***Keywords*-memory wall, memory bound, stencil, banded matrix, time skewing, temporal blocking, wavefront**

I. Introduction

Stencil computations are ubiquitous in scientific computing primarily because the action of discretized local differential or integral operators can be expressed in this form. Solving large partial differential equations (PDEs) in reasonable time requires iterative solvers, thus stencil computations are performed repeatedly for many iterations. Moreover, many PDEs are discretized over time, so there is an additional outer loop for every time iteration.

Most often a stencil computations is just a linear weighting of a small neighborhood. In this case, the stencil computation represents a matrix vector product with the stencil weights forming the rows of the matrix, and the discrete values of the domain forming the vector. The arithmetic intensity of such an operation is very low, with just one multiplication and addition for every vector and stencil component read. Even if the stencil weights are constant and can be stored locally, there are still just two operations for every value read, whereas synthetic benchmarks on our Opteron 2218 and Xeon X5482 suggest that peak bandwidth and peak compute performance would be balanced if 14.9 and 52.6 floating point operations were performed for each access to a double value in the main memory, respectively, see Table I. So only a non-linear stencil computation with many operations could prevent it from being memory bound.

This imbalance between the computation power and system bandwidth is called the *memory wall* problem [2]. The costly introduction of double, triple and quad-channel memory buses has temporarily stopped the further deterioration of this problem, but in the long run we will see a growing discrepancy again: The cost-efficient exponential growth in the number of cores in CPUs cannot be matched by the expensive bus widening or data rate doubling at the same pace. Intensive research into alternative technologies, e.g., stacked memory or optical connection, is underway but an economic solution for the mass-market is not yet in sight [3].

A. Related Work

For small discrete vectors that fit into the processor's caches, the cache bandwidth is the decisive factor of performance, but stencils in scientific computing typically operate on data much bigger than the cache capacity. Substantial work has been performed to optimize the data locality in such cases up to the point where tight lower and upper bounds on the number of data loads can be given [4]. Recent results show large benefits in applying these techniques on multi-core architectures [5]. But no matter how efficiently we load the data into the caches, for data exceeding the cache size, we still read every vector component at least once per timestep from the main memory and for repeated applications of the stencil, this is far too much. To further reduce access to main memory, we need to exploit the outer loops that repeat the stencil computations over the same domain and make use of *temporal locality*. When advancing certain parts of the domain several stencil iterations ahead of the rest, we need to respect data dependencies induced by the form of the stencil. So called *time skewing* techniques have been described by Wolf [6], Song et al, [7] and Wonnacott [8]. Thereby, the time axis corresponds to the number of iterations that the stencil is applied to the entire spatial domain, e.g., this can be the explicit time steps of a PDE solver, or the iterations of an iterative solver for linear equation systems.

With this additional time axis we can form the space-time domain $\Omega \times \{0, \dots, T\}$, where the data at $\Omega \times \{0\}$ is given

and the task is to compute a value for all remaining points in the space-time, see Fig. 1. Now, the general idea of time skewing is to tile the space-time into space-time tiles that can be executed with very few cache misses and ideally also in parallel. These requirements lead to skewed tiles in the space-time, see Fig. 2. The tile dimensions form a large optimization space which can be explored empirically [9]–[11] and systematically [12]–[14], whereby it makes a big difference if the exploration targets mainly data locality, or parallelism, or both equally. A third approach is to use a hierarchical tiling that adapts automatically to the available cache size, which is thus labelled *cache oblivious* [**?**], [9], [15]. A more general approach for optimizing iterative stencil computations is to use a loop transformation and parallelization framework [1], [16]–[19]. We compare our results against one of them in detail, namely PluTo [1], which is an easy-to-use fully automatic tool and a good indicator of the performance that can be achieved immediately on these nested loops without any further user interaction.

B. Motivation and Contribution

Common to all of the above approaches in case of a multi-dimensional domain, is a *multi-dimensional tiling* strategy: the time and multiple (not necessarily all) spatial dimensions are divided in order to form space-time tiles of approximately the same diameter in all divided dimensions. This minimizes the surface area to volume ratio of the space-time tiles and thus reduces cache misses. It is the best general strategy to traverse a space-time of unknown size [20]. However, knowing the typical cache size of 128KiB–4MiB per core and domain sizes $(100\text{–}1000)^d, d = 2, 3$ we contribute an algorithm that does the exact opposite: we tile only one spatial dimension (resulting in enormous space-time tiles) and use the relatively large caches of nowadays cores to reduce the 2D or 3D problem to a 1D problem, where spatial tiling is not necessary and instead a wavefront traversal can be used. Motivation: We obtain large space-time tiles and thus that can still be processed in a SIMD friendly and cache efficient manner.

Another major difference is the treatment of the memory hierarchy. Previous approaches use a *multi-level tiling* strategy: they hierarchically subdivide the space-time tiles either explicitly or automatically with the idea that the basis of the sub-tiles will fit into a deeper cache level (e.g. L1) and thus the sub-tile will be processed faster. We agree in general, however, considering the concrete bandwidth and compute ratios, we explore the opposite direction of ignoring the memory hierarchy and instead maximizing the wavefront size in the last cache level (L2 in our case). Motivation: Large wavefronts maximize the number of space-time points that are processed on-chip in a *highly regular* fashion, while processing data from the L2 cache is not a big limitation. The stencil computation remains memory-bound but only by a small factor: the balanced stencil intensity for L2 on our Opteron 2218 is 2.2 and 3.1 on the Xeon X5482, see Table I, i.e., 2 or 3 floating point operations on every double read from L2 are sufficient to balance peak stencil computation and data transport from L2.

The above numbers are derived from the performance of a vectorized stencil kernel that executes on registers. Because the numbers are small, we also use a vectorized kernel for the actual computation otherwise the data processing could not keep up with the bandwidth of the L2 cache and the memory-bound stencil would become unnecessarily compute-bound. In other words, the vectorization ensures that the kernel remains memory-bound but cannot accelerate the execution beyond that.

We keep the rest of the algorithm as simple as possible. We use a single form for all tiles and choose a minimalist parallelization approach: the threads are started once at the beginning and are persistent throughout the computation; furthermore the thread to tile assignment is known at compile-time leading to simple synchronization. Motivation: When striving for benchmark performance in applications, code simplicity is of great benefit to the compiler and hardware. Moreover, dynamic load-balancing is not necessary for tiles of equal size, and replacing barrier synchronization by tile-to-tile synchronization minimizes the idle time otherwise.

In summary, our algorithm (CATS) is based on the reduction of 2D and 3D problems to a one-dimensional, non-hierarchical, all cache consuming wavefront traversal inside the tiles. Our first version CATS1 in Section II-B is related to Wonnacott's wavefront computations [8] and the pipelined temporal blocking [11]. However, both are still multi-dimensional tiling strategies, which CATS is not. Moreover, their parallelization is already different. CATS2 and the general CATS scheme (Sections II-C and II-D) are novel wavefront traversal techniques with an arbitrary wavefront dimensionality and shape: No previous scheme uses a wavefront traversal in such radical fashion that reduces everything to a 1D problem. In view of theoretical results on the lower bound of cache misses in stencil computations [20], it is surprising that the much simpler CATS can compete against the usual strategies of multi-dimensional tiling and multi-level tiling. In fact, we do not claim better asymptotic behavior but rather demonstrate high performance levels on large 2D and 3D domains. We call the algorithm CATS (cache accurate time skewing) because we accurately reuse the *entire* last level cache with each single wavefront computation.

The rest of the paper is organized as follows, in Section II we describe the family of cache accurate time skewing schemes (CATS). Section III describes our empirical setup and extensive comparisons of the execution times of the naive scheme, the PluTo transformed code and the CATS schemes in 2D and 3D. We draw conclusions in Section IV.

II. Cache Accurate Time Skewing (CATS)

This section describes the cache accurate time skewing schemes in comparison to the naive scheme. We first describe some specific variants of CATS and then explain how they combine to give the general CATS scheme.

On a discrete d-dimensional spatial domain $\Omega := \{1, \ldots, W_1\} \times \ldots \times \{1, \ldots, W_d\}$ with $N := \#\Omega$ values we want to apply a stencil $S : \Omega \times \{-s, \ldots, +s\}^d \to \mathbb{R}$ of *slope* s repeatedly to the entire domain T-times. In

Alg. 1 The naive scheme for iterative stencil computations in 2D. The spatial domain is cut along the y-dimension into tiles for parallel execution and ystart(tid), yend(tid) are the tile bounds in dependence on the thread ID tid.

```
naive_2D ()
{
  for(t = 0; t < T; t++) {
    for(y = ystart(tid); y < yend(tid); y++) { // parallelized
    for(x = 0; x < WIDTH; x++) {              // vectorized
      apply 2D stencil at position (x,y,t);
    }}//x,y
    synchronize threads;
  }//t
}
```

Alg. 2 CATS1 for iterative stencil computations in 2D. The loop bounds ystart(tid), yend(tid) represent the extent of the tile (parallelogram) along the traversal dimension y. The loop bounds tstart(ts,y), tend(ts,y) represent the extent of the wavefront along the dimension t within the tile, see Fig.2.

```
CATS1_2D ()
{
  compute height T_Z from cache size (Eq. 1);
  for(ts = 0; ts < T/T_Z; ts++) {
    for(y = ystart(tid); y < yend(tid); y++) { // parallelized
      if(y == ystart(tid+1)) {
        wait for (tid+1) to finish its left tile border;
      }
      for(t = tstart(ts,y); t < tend(ts,y); t++) {
        for(x = 0; x < WIDTH; x++) {      // vectorized
          apply 2D stencil at position (x,y-t,t);
        }//x
      }//t
    }//y
    synchronize threads;
  }//ts
}
```

case of a constant stencil, S does not depend on Ω and has a certain number of non-zero values $N_S := \#S$, otherwise we assume that the stencil is position dependent, $S(x) : \{-s, \ldots, +s\}^d \to \mathbb{R}, x \in \Omega$ and has the same number of non-zero values N_S for every position, and $N \cdot N_S$ values overall.

Our space-time domain is given by $\Omega \times \{0, \ldots, T\}$ with the initial values at $\Omega \times \{0\}$ and boundary values at $\partial\Omega \times \{0, \ldots, T\}$, $\partial\Omega := \{0, W_1{+}1\} \times \ldots \times \{0, W_d{+}1\}$. In the space-time $\Omega \times \{0, \ldots, T\}$ there are TN values to be computed, and each output value requires N_S input values. So in case of a constant stencil we perform TNN_S reads and TN writes; in case of a variable stencil (banded matrix) we perform $2TNN_S$ reads and TN writes.

If we access values from timestep $t-1$ to compute values at timestep t then, irrespective of the scheme, we need to store two copies of Ω during the stencil application. Some stencil computations like Gauss-Seidel, that use values from timestep $t-1$ and t while computing timestep t, can be performed in-place with just one copy of Ω. If these one/two copies of Ω fit into the cache, then all reads and writes will happen in the cache no matter how large T is. The naive scheme performs much better in this case, as can be seen for the 0.5 million elements case in the Figs. 6 and 8.

A. No Skewing - NaiveSSE Scheme

The naive stencil implementation has no data reuse between different iterations. The entire spatial domain advances one timestep after another, see Fig. 1 and Alg. 1. The outermost spatial loop is parallelized with multiple threads, whereby each thread operates on one tile of the domain. The tiles are of the same size so the threads can be synchronized with little overhead after each timestep. The innermost spatial loop (unit stride dimension) is hand-vectorized with SSE2 intrinsics.

B. Skewing One Dimension - CATS1 Scheme

The general idea behind time skewing schemes is to compute multiple timesteps at once in certain parts of the domain thus exploiting the temporal producer-consumer locality. For this purpose we tile one spatial dimension. The plane formed by the chosen spatial dimension and the time dimension is divided into space-time tiles, see Fig. 2. The tiles are skewed to respect the temporal data dependencies induced by the stencil. Processing within the space-time tile has high temporal locality, while data at the tile borders, in general, has to be reloaded from main memory. Skewed tile borders require more data transfer than straight tile borders. The main decision is on the form of the tiles, aiming for maximal temporal locality and parallel processing of tiles. We use parallelogram tiles with split-tiling and wavefront processing (Fig. 2).

These ideas have been described for multiple processors instead of cores already at the onset of time skewing methods by Wonnacott [8], but even in CATS1 we use them differently for multi-dimensional domains. In particular, we show that multi-dimensional tiling of multi-dimensional domains is not necessary. Instead of diagonal wavefronts, we consider axis-aligned wavefronts, and our tile placement is also different. The pipelined temporal blocking by Wittmann et al. [11] and Wellein et al. [21] can also be seen as a variant of space-time wavefront processing. However, they use the term 'wavefront' completely differently, describing the parallelization along the time axis, which benefits from shared caches between multiple threads. This type of 'wavefront' does not exist in our scheme, because we use a different parallelization approach that does not rely on shared caches; instead we construct large space-time wavefronts (using Wonnacott's space-time notion of a wavefront) for the purpose of the data locality maximization.

In wavefront processing we sweep with a skewed space-time surface (the *wavefront*) through the tile along a designated *traversal dimension* (see the arrows in Fig. 2), maintaining a certain number of the most recent wavefronts in the cache. The computation takes place at the wavefront reusing the data from the previous wavefronts. New data must only be fetched from main memory at the tile borders. For a stencil width of $2s+1$ in the traversal dimension, $2s$ wavefronts plus some temporary variables must reside in an ideal cache for perfect data reuse, but because of limited cache associativity and cache line granularity, a certain value $C_S \in (2s, 2s+1]$ is used in practice, e.g., Wonnacott [8] uses the pessimistic $C_S := 3$ for a 3-wide stencil, we conservatively choose $C_S := 2s + 0.8$ after

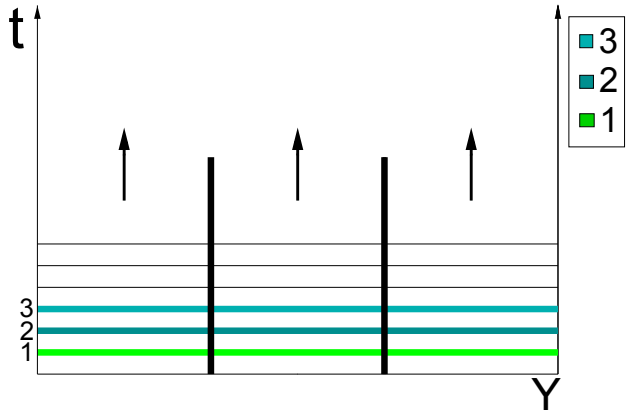

Fig. 1. Naive space-time traversal in parallel with three threads, cf. Alg. 1. Regions of the same color are operated on in parallel, synchronization takes place before starting a different color region. The entire domain progresses one timestep after another in sync in the direction of the arrows. X-dimension goes into the page.

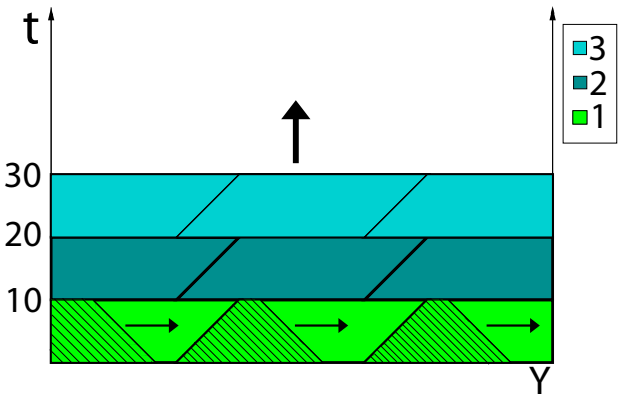

Fig. 2. Cache accurate time skewing with one skewing dimension (CATS1) in parallel with three threads, cf. Alg. 2. Regions of the same color are operated on in parallel, synchronization takes place before starting a different color region. The fine lines show the consecutive wavefront positions and the arrows the traversal direction in each parallelogram. X-dimension goes into the page.

a cache miss analysis.

The main advantage of wavefront processing is that the tiles can be much bigger than the cache, because only C_S wavefronts must reside in the cache for a perfect producer-consumer locality within the tile. One driving idea behind our cache accurate time skewing schemes is to radically maximize the wavefront size at the expense of any other locality optimizations. In case of one dimensional skewing, CATS1 maximizes the wavefront size such that C_S wavefronts barely fit into the private L2 cache of one thread. Let Z be the size of the private L2 cache and $W_{\max}$ the size of the largest domain dimension, the one to be traversed, then the size of our wavefront is $T_Z N/W_{\max}$ and we can compute the maximal temporal extent of our tile T_Z in dependence on Z as

$$T_Z \quad := \quad \lfloor ZW_{\max} / (C_S N) \rfloor . \tag{1}$$

Wonnacott [8] considers diagonal wavefronts $\{(x,y,t) \in \text{Tile} \mid x+y+t = \text{const}\}$ in 2D and concludes that their maximum size in dependence on the domain size makes it impractical for large domains, so both dimensions must be tiled. The validity of this argument depends on what *large* means. For typical cache and domain sizes, we argue in the opposite direction that a wavefront traversal actually makes multi-dimensional tiling unnecessary. The maximum size of our axis-aligned $\{(x,y,t) \in \text{Tile} \mid y+t = \text{const}\}$ wavefronts grows with the domain size in the same fashion, the growth is proportional to $N/W_{\max}$, but in 2D this is not a big problem even for a small cache of 128KiB, e.g., $3 \cdot 10 \cdot 500 \cdot 8\text{B} = 120\text{KB} < 128\text{KiB}$, which means that on a 500^2 domain of doubles we could perform $T_Z = 10$ consecutive timesteps in cache. The next section explains that one-dimensional tiling is sufficient even in case of larger (e.g. 10000^2) domains in 2D and 3D. The reasons for choosing axis-aligned over diagonal wavefronts are the much simpler indexing and more favorable memory access pattern. Axis-aligned refers to the spatial alignment, all wavefronts are always skewed with respect to time.

The time dimension is tiled according to T_Z and Alg. 2 shows the entire CATS1 algorithm in 2D. Fig. 2 ($T_Z = 10$) shows with thin lines the different positions of the wavefronts and how they progress through the space-time tiles in the direction of the arrows. In CATS1, the parallelization takes place along the same dimension (y-loop in Alg. 2) as the wavefront traversal. All threads can start computing concurrently within their parallelograms, there is only a data dependency at the right border of each parallelogram, and thread tid has to wait for thread tid+1 if it reaches its right border faster than tid+1 finished its computation there. For almost all domains the width of the tile is much bigger than its height, so in practice the thread tid does not have to wait. This type of dependence resolution between parallelogram tiles is called *split-tiling* [12]. After completing the wavefront traversal for all tiles in $[0, T_Z)$ the threads are synchronized with little overhead as the tiles are of equal size, and all tiles in $[T_Z, 2T_Z)$ are processed in the same fashion, cf. ts-loop in Alg. 2.

Wonnacott [8] and Krishnamoorthy et al. [12] deal with multi-processor systems, so in order to reduce the communication, they align the base of the higher parallelogram with the top of the lower one in the split-tiling scheme. However, this causes load-balancing problems which we avoid by placing the parallelograms simply axis-aligned on top of each other, see Fig. 2. Because the CPU cores have access to the same main memory, this has no negative effect for us. Irrespective of the parallelogram placement strategy, there is basically no data reuse at the tile borders, because the entire cache is constantly overwritten by the traversing wavefronts.

In 2D and higher dimensions, the innermost loop in CATS1 runs across the entire unit stride dimension (x-loop in CATS1_2D() in Alg. 2) so its vectorized execution ensures that the algorithm remains memory-bound when processing data from the L2 cache. In 3D there are two loops with fixed bounds that span the entire domain. However, these

Alg. 3 CATS2 for iterative stencil computations in 2D. The loop bounds 0, HEIGHT represent the extent of the tile (diamond tube) along the traversal dimension y. The loop bounds tstart(dia,y), tend(dia,y) and xstart(dia,y,t), xend(dia,y,t) represent the extent of the wavefront along the t and x dimension within the tile (diamond tube), see Fig.4.

```
CATS2_2D ()
{
  compute diamond size from cache size (Eq. 2);
  forall( diamond dia∈diamondSet(tid) ){ // parallelized
    wait on the two diamonds below to finish;
    for(y = 0; y < HEIGHT; y++) {
      for(t = tstart(dia,y); t < tend(dia,y); t++) {
        for(x = xstart(dia,y,t); x < xend(dia,y,t); x++) {
          apply 2D stencil at position (x,y-t,t);//↑vectorized
        }//x
      }//t
    }//y
  }//dia
}
```

inner loops also mean that more data resides in the wavefront, e.g., in 3D the wavefront extends in three dimensions (x,y,t)∈[0,WIDTH)×[0,HEIGHT)×[0,T_Z). So if WIDTH and HEIGHT are large, the computed T_Z will be smaller than one and we fall back to the naive scheme. Apparently, multi-dimensional tiling of the domain is required in 3D after all, but we present a different solution in the next section.

C. Skewing Two Dimensions - CATS2 Scheme

CATS1 is a special case because it uses the same spatial dimension for tiling and the wavefront traversal. CATS2 and all higher schemes have a distinct traversal dimension and tiling dimensions. For CATS2 one dimension is tiled, and a second is traversed with the wavefronts. This way we reduce the wavefront size in comparison to CATS1 without the need for multi-dimensional tiling.

CATS2 requires two distinct dimensions so it can be applied only in 2D and higher dimensional spatial domains. Fig. 3 shows the (x,t)-plane with the tiling dimension x in case of CATS2_2D() in Alg. 3. In the (x,t)-plane, the space-time tiles have the shape of diamonds. Together with the traversal dimension (y in 2D), the diamond forms the corresponding space-time tile, a *diamond tube* as depicted in Fig. 4. The diamonds in Fig. 3 are the projections of the diamond tubes onto the (x,t)-plane. The processing of a diamond tube is similar to the traversal in CATS1: a wavefront sweeps through it along the traversal dimension.

Fig. 4 visualizes the processing of a 2D spatial domain. Therein the diamond tube is a 3D space-time tile, and the wavefront a skewed 2D diamond. For a 3D spatial domain, the diamond tube is 4D and the wavefront is 3D, therefore, the problem is still reduced to a 1D traversal. The key insight is that a wavefront traversal can be performed with a wavefront of arbitrary dimensionality and arbitrary shape. Thus multi-dimensional tiling is not necessary for generating temporal locality and we can process much larger space-time tiles than usual in a cache efficient manner. This is a new idea in wavefront processing of multi-dimensional domains.

We use diamonds in the tiling dimension because of their favorable surface area to volume ratio (cache miss reduction), they are independent of each other when arranged side-by-side (parallel execution), and require only one tile form to cover the plane (simplicity). Orozco and Gao [22] give a quantitative analysis for the first property, however, they use the diamond shape only in 1D with a traditional bottom-up processing of the tile in cache. The second property avoids the problem of dependent tiles encountered by Liu and Li [23], where they have to relax the numerical properties of the scheme in order to gain better parallelization.

As in CATS1, we pursue the goal of maximizing the wavefront size without reverting to multi-dimensional tiling. Let Z be the private L2 cache size of each thread, $W_{\max}$ be the size of the largest domain dimension which is traversed, and $W_{\max 2}$ be the second largest which is tiled. Let B_Z be the width of the single-form diamond, then $B_Z^2/(2s)$ is its area, and B_Z can be computed as

$$B_Z \quad := \quad \left\lfloor \left(2sZW_{\max}W_{\max 2}/\left(C_S N\right)\right)^{\frac{1}{2}} \right\rfloor . \qquad (2)$$

This value determines how many diamonds will fit side by side along the tiling dimension. As we consider large domains, we have sufficiently many independent diamonds to occupy multiple threads. Should this not be the case because of a small tiling dimension, then we can swap the traversal and tiling dimensions or switch to CATS1 which will tile and traverse the same dimension.

Orozco and Gao [22] process their diamonds in rows with a global synchronization between rows, but this is not necessary as Fig. 3 shows. Because the computation in each diamond depends only on the two diamonds below it, the processing can be easily parallelized irrespective of how many diamonds reside in a row. Moreover, we do not need a global synchronization among threads, instead every diamond simply waits on the two diamonds below it before it starts processing, see the dia-loop in Alg. 3. The a-priori thread to tile assignment may still lead to some idle time, but this is much smaller than Fig. 3 suggests at first, e.g., the thread that computes the tiny triangle at the right border continues *immediately* with the third diamond in the second row because the two green diamonds below have already finished.

In the previous section, we have seen that CATS1 runs into problems on large 3D domains. CATS2 has no problems in 3D because the size of the wavefront inside the diamond tube that needs to reside in the cache is now further restricted by B_Z. Only on enormous 3D or higher dimensional domains, that do not fit into a typical main memory size of 8 GiB, we would need to switch to higher order CATS schemes that are discussed next.

D. Multiple Skewing - General CATS Scheme

By adding more tiling dimensions we can define CATS3, CATS4, etc. In these schemes we still have one traversal dimension but multiple tiling dimensions. The additional complexity in comparison to CATS2 is the more complicated form of space-time tiles, which corresponds to more loops with variable bounds in the algorithm. But even if enormous domain

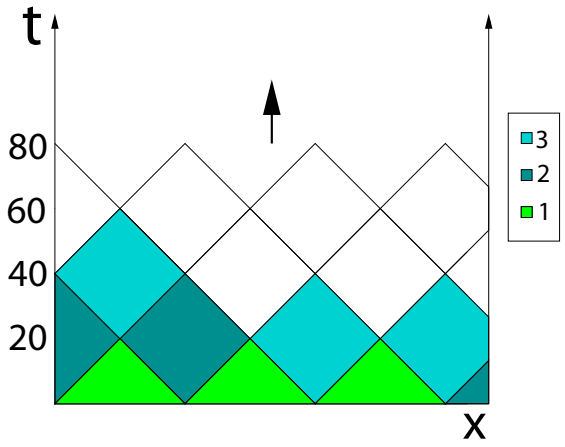

Fig. 3. Cache accurate time skewing with two skewing dimensions (CATS2) in parallel with three threads, cf. Alg. 3. The colors show the a-priori thread to tile assignment, but there is no global synchronization, each diamond waits on the two below. This figure shows the (x,t)-plane for CATS2_2D(), each of the diamonds extends also in the y-dimension (which goes into the page) forming a diamond tube, see Fig. 4.

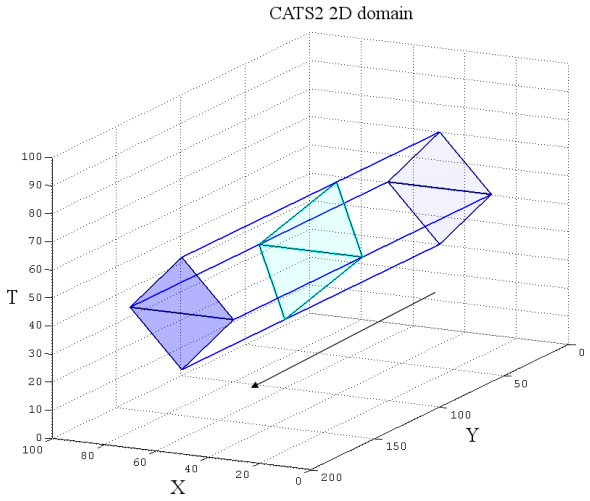

Fig. 4. In CATS2_2D() (Alg. 3) each thread sweeps a diamond-shaped wavefront through a diamond tube region of the space-time. First all values within the current wavefront are computed then the wavefront moves by 1 along the y-dimension. No unnecessary cache misses occur inside the diamond tube although it is much bigger than the cache.

sizes force us to tile multiple dimensions in CATS3 and higher, in contrast to classical multi-dimensional tiling approaches, we tile two dimensions less, one is reserved for the wavefront traversal, the other for vectorization.

When tiling multiple dimensions, we can freely choose which of them should also be parallelized. The tiled and parallelized dimensions use the diamond shape, whereas the tiled-only dimensions may also use space dependent tiles like the parallelograms. On multi-core processors it is sufficient to parallelize just one of the tiling dimensions. Only when extracting hundredfold parallelism on many-core processors, we would also parallelize more tiling dimensions.

In general, a d-dimensional domain admits the use of the CATSk scheme with $k = 1, \ldots, d$. The difference $d - k$ specifies how many dimensions have not been skewed and thus how many inner loops with fixed bounds that scheme has. All values traversed in these loops must reside in the cache, and therefore this difference is usually 0, 1 or 2. If $d - k = 0$ then the cache size poses no problem at all, but the execution of the innermost loop is less efficient because of the variable loop bounds. For common cache sizes of 128KiB–4MiB per core and domain sizes $(100\text{–}1000)^d$, choosing CATS$(d-1)$ for a d-dimensional spatial domain is a safe choice that gives fixed loop bounds for the unit stride dimension. We define the general CATS scheme to be this combination of the CATSk schemes. We only deviate in two cases: for 1D problems CATS0 is equivalent to the naive scheme so CATS1 is the better choice; for very large dimension sizes, e.g., 10000^2 CATS1 would hold the values from the inner loop only for very few timesteps simultaneously and then switching to CATS2 despite the variable loop bounds is better. As a rule of thumb, we switch from CATS$(k-1)$ to CATSk when the wavefront in CATS$(k-1)$ would extend over less than 10 timesteps.

III. Results

We compare the performance of the following schemes on iterative stencil computations:

- NaiveSSE: Our own hand-parallelized (pthreads) and vectorized (SSE2) naive stencil scheme as described in Section II-A.
- PluTo [1]: code transformed by the automatic parallelizer and locality optimizer for multicores PluTo, version 0.4.2.
- Stencil peak: the stencil benchmark that executes directly on registers without data transfers, see Table I.
- CATS: Our general cache accurate time skewing scheme with the selection of individual schemes described in Section II-D. The innermost loop uses a vectorized (SSE2) kernel and parallelization uses pthreads.

Our hardware configuration is listed in Table I. As general compiler options we use -O3 -funroll-loops and for the icpc compiler also -xHOST -no-prec-div. The NaiveSSE scheme does not require any parameters, it only needs a scalar and a vectorized kernel that are called from the nested loops.

For PluTo-0.4.2 we use -tile -l2tile to tile the code for the L1 and L2 cache, -multipipe to extract multiple degrees of parallelism, -parallel to parallelize the code using OpenMP, -unroll to automatically unroll up to two loops, and -nofuse to separate all strongly-connected components in the dependence graphs. The options -unroll -nonuse do not make a difference in performance in our tests. In 3D we decided to omit the option -l2tile as the transformation process was taking hours and did not provide performance gains. We use the original examples provided with PluTo and modify them from constant

Table I Hardware configurations of our test machines. The machines have been chosen such that one (Opteron) has a modest ratio between measured system and cache bandwidth, while the other (Xeon) has a high ratio. This ratio is the main source of acceleration of time skewing against naive schemes.
The measured bandwidth numbers have been obtained with the RAMspeed benchmarking tool and the double precision (DP) FLOPS numbers come from our own SSE benchmarks. For the peak DP number we perform independent multiply-add operations on registers, for the stencil DP number we run the inner stencil computation (products and accumulation) on registers. This value is lower because of the read-after-write dependencies in the computation. All benchmarks show results for the entire machine achieved with 4 threads.

Brand	AMD	Intel
Processor	Opteron 2218	Xeon X5482
Code-named	Santa Rosa	Harpertown
Frequency	2.6 GHz	3.2 GHz
Number of sockets	2	1
Cores per socket	2	4
L1 Cache per core	64 KiB	32 KiB
L2 Cache per core	1 MiB	3 MiB
Operating system	Linux 64 bit	Linux 64 bit
Parallelization	4 pthreads	4 pthreads
Vectorization	SSE2	SSE2
Compiler	g++ 4.3.2	icpc 11.1
Measured L1 Bandwidth	79.3 GB/s	194.6 GB/s
Measured L2 Bandwidth	40.6 GB/s	64.2 GB/s
Measured Sys. Bandwidth	11.2 GB/s	6.20 GB/s
Measured Peak DP FLOPS	20.8 G	40.8 G
Measured Stencil DP FLOPS	11.5 G	25.1 G
L2 Band./Sys. Bandwidth	3.6	10.4
Peak DP/(Sys. Band./8B) Balanced arith. intensity for Sys.	14.9	52.6
Stencil DP/(Sys. Band./8B) Balanced stencil intensity for Sys.	8.2	32.4
Stencil DP/(L2 Band./8B) Balanced stencil intensity for L2	2.2	3.1

to variable stencil where necessary. It is not feasible to hand-vectorize the transformed code because of the high number of generated loops, e.g., 142 loops for the constant 7-point stencil in 3D. However, we ensure the best possible performance by retransforming and recompiling the examples every time with compile-time known domain sizes and aggressive icpc auto-vectorization, the compilation process alone takes about 15 minutes.

CATS takes as parameters the size of the last cache level (L2 for us), the slope of the stencil s, the memory size of a data type and optionally additional cache requirements, e.g., the matrix coefficients. CATS is implemented as a library not a code generation framework. The kernel may perform arbitrary index calculations and non-linear operations on the data within the stencil region $\{-s,\ldots,+s\}^d$ and on the specified amount of additional values like matrix coefficients. Beside the parameters, the user only provides a scalar and a vectorized version of the kernel, the same kernels used by the optimized naive scheme.

Our test applications comprise constant and variable stencils in 2D and 3D with 0.5 to 128 million double precision elements. In 2D, we have squares ranging from 706^2 to 11282^2 elements and in 3D, cubes from 80^3 to 500^3. In case of constant stencils, this amounts to a memory consumption of up to 2GiB for the two vectors, and in case of variable stencils we use at most 32 million elements consuming 0.5GiB plus 1.75GiB for the matrix in 3D. We use a general 5-point stencil in 2D (5 muls plus 4 adds equal 9 flops) and a 7-point in 3D (7 muls plus 6 adds equal 13 flops). The number of iterations is either $T = 100$ (solid graphs in the figures), or $T = 10$ (dashed graphs in the figures). The last stencil application is the FDTD 2D example (11 flops) that comes with PluTo.

All figures show the execution time in seconds against the number of elements in millions with both axes being logarithmic. The number of elements doubles between two consecutive graph points, but the doubling is not totally exact because of the square or cubic root operations involved in computing a square or cube with a predefined number of elements.

A. Constant Stencil

In this section, we present results for constant stencils of slope $s = 1$. Figs. 5, 6 show the execution times for 2D spatial domains and Figs. 7, 8 for 3D. The graphs have many features in common.

a) Large slowdown of the naive scheme on the Xeon when transitioning from 0.5 to 1.0 million elements: The Xeon has 12MiB of L2 cache (cf. Table I), so that two vectors of 0.5 million elements ($2 \cdot 8\text{B} \cdot 0.5\text{M} = 8\text{MB}$) fit into the cache. The one million elements case already requires 16MB, which exceed the cache size, so the performance of the naive scheme suffers a large slowdown and from thereon becomes completely limited by the available system bandwidth. The CATS scheme, on the other hand, has a more consistent scaling and simply ignores the fact that the data does not fit into the cache any more. This causes the CATS graph for $T = 100$ iterations on the Xeon in 2D (Fig. 6) and 3D (Fig. 8) to come close to the naive graph for $T = 10$ iterations on large problems. The PluTo scheme also scales consistently but at a much lower level. The Opteron does not show the jump on the naive scheme because its 4MiB of L2 cache cannot accommodate two copies of the 0.5 million elements, so it is already in the slow mode determined by the system bandwidth.

b) The Opteron is faster than the Xeon on the naive scheme but slower on PluTo and CATS: The faster execution on the naive schemes is directly related to the higher system bandwith on this machine as it is the limiting performance factor, see Table I. For the time skewing PluTo and CATS schemes, on the other hand, the system bandwidth is less relevant even when the data size exceeds the cache size more than hundredfold, as in the case of the 128 million element examples with 1GiB of data for each vector. The cache bandwidth is the decisive factor, hence the Xeon is better and consequently shows better results despite its low system bandwidth.

For the achievable acceleration factor the ratio of cache to system bandwidth (3.6 Opteron, 10.4 Xeon, see Table I) and

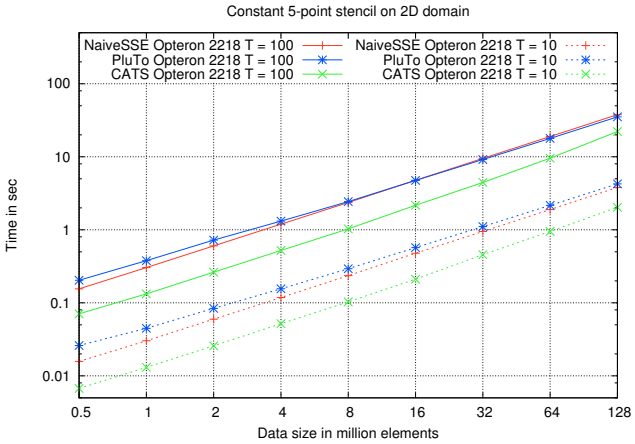

Fig. 5. Timings of the Opteron 2218 with constant stencils in 2D. GFLOPS for 128 million elements with $T = 100$: NaiveSSE Opteron 3.4, PluTo Opteron 3.6, CATS Opteron 5.8 (50% of stencil peak).

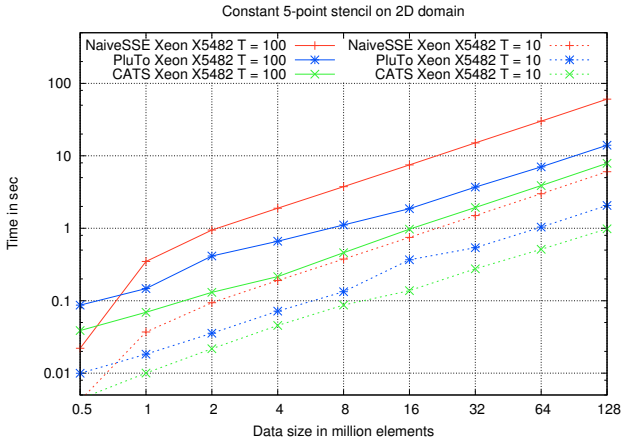

Fig. 6. Timings of the Xeon X5482 with constant stencils in 2D. GFLOPS for 128 million elements with $T = 100$: NaiveSSE Xeon 1.9, PluTo Xeon 8.2, CATS Xeon 16.2 (65% of stencil peak).

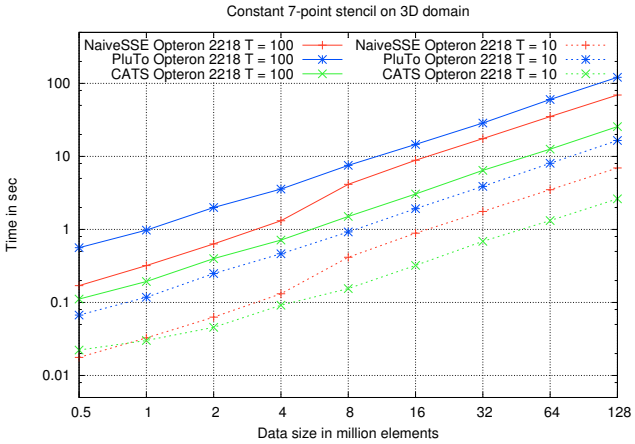

Fig. 7. Timings of the Opteron 2218 with constant stencils in 3D. GFLOPS for 128 million elements with $T = 100$: NaiveSSE Opteron 2.4, PluTo Opteron 1.5, CATS Opteron 6.4 (55% of stencil peak).

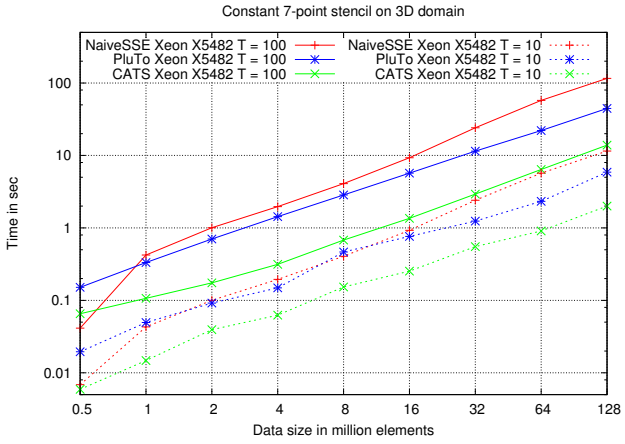

Fig. 8. Timings of the Xeon X5482 with constant stencils in 3D. GFLOPS for 128 million elements with $T = 100$: NaiveSSE Xeon 1.4, PluTo Xeon 3.7, CATS Xeon 13 (52% of stencil peak).

the scheme's ability to exploit this ratio are important. CATS exploits this ratio well outperforming the naive scheme on the Opteron by a factor 2 on average, and on the Xeon by at least 7.5x. PluTo does also benefit from the ratio but to a smaller extent. It performs on average slower than the naive scheme on the Opteron, but faster on the Xeon due to the bigger ratio on the Xeon.

c) Performance in 2D is generally better than in 3D: This is not suprising as the surface area to volume ratio is worse in 3D but the effect on the schemes varies substantially. The naive scheme in 3D maintains the same performance as in 2D on smaller domains, which makes sense because the same amount of data is transported and system bandwidth is the limiting factor. Beyond a certain size in 3D, four 2D slices (3 input plus 1 output) of the domain do not fit into the cache anymore so that stencil neighbors have to be brought into cache multiple times and performance degrades. PluTo works best in 2D where it is on par with the naive scheme on the Opteron and much faster on the Xeon. In 3D the performance degrades by more than 2x in both cases. CATS also slows down in 3D but only by around 20%, so the speedup over PluTo grows to more than 3.5x.

d) Comparison with stencil peak: The stencil peak benchmark (Table I) measures the performance of the stencil computation in case of infinite bandwidth. The CATS scheme achieves more than 50% of this performance even when operating on gigabyte large domains connected with low system bandwidth (only 6.2 GB/s on the Xeon).

B. Banded Matrix

If the stencil is not constant but rather varies across the domain, then its application corresponds to a banded matrix vector product. In Section II we assumed N_S as the number of non-empty stencil elements, this corresponds to the number of bands in the matrix. For the space-time traversal this means that not only the vector components (domain values) must reside in the cache but also the corresponding matrix entries. We need the matrix entries only for the current wavefront

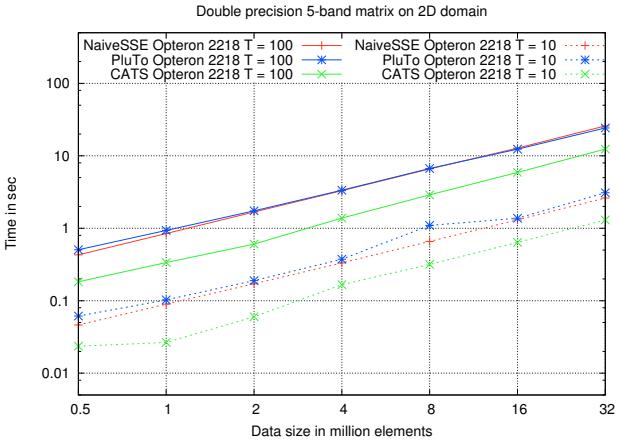

Fig. 9. Timings of the Opteron 2218 with a banded matrix in 2D. GFLOPS for 32 million elements with $T = 100$: NaiveSSE Opteron 1.1, PluTo Opteron 1.2, CATS Opteron 2.8 (24% of stencil peak).

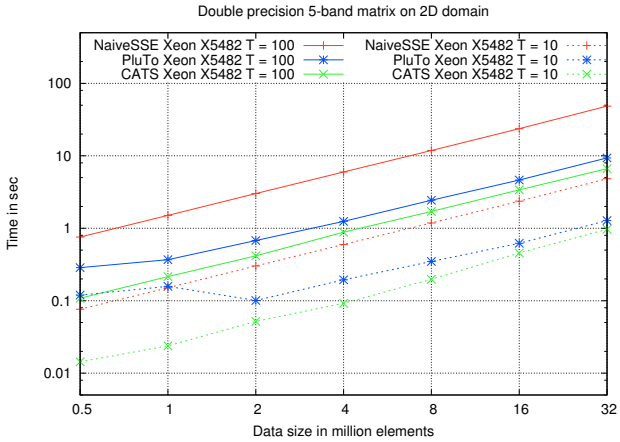

Fig. 10. Timings of the Xeon X5482 with a banded matrix in 2D. GFLOPS for 32 million elements with $T = 100$: NaiveSSE Xeon 0.6, PluTo Xeon 3.1, CATS Xeon 4.9 (20% of stencil peak).

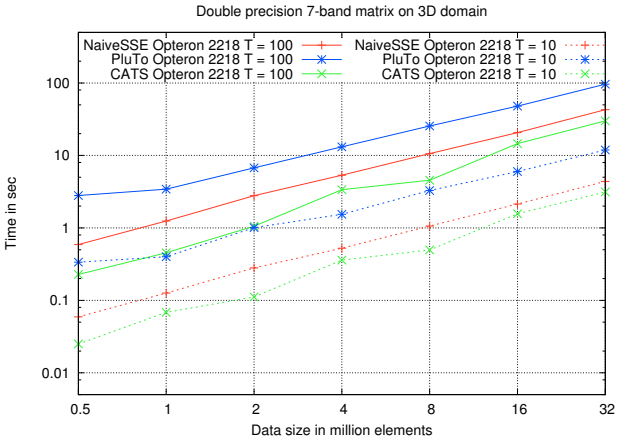

Fig. 11. Timings of the Opteron 2218 with a banded matrix in 3D. GFLOPS for 32 million elements with $T = 100$: NaiveSSE Opteron 1.0, PluTo Opteron 0.4, CATS Opteron 1.5 (13% of stencil peak).

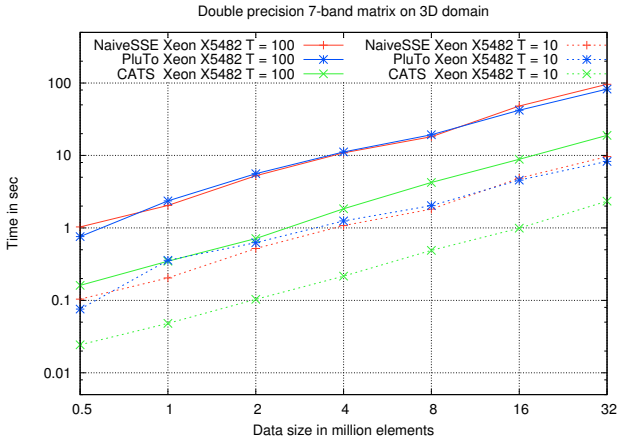

Fig. 12. Timings of the Xeon X5482 with a banded matrix in 3D. GFLOPS for 32 million elements with $T = 100$: NaiveSSE Xeon 0.4, PluTo Xeon 0.5, CATS Xeon 2.5 (10% of stencil peak).

during the computation, so C_S must be replaced by $C_S + N_S$ in our formulas Eq. 1 and Eq. 2 that compute the maximum extent of the wavefront. We run performance tests with $T = 10$ and $T = 100$ iterations shown in Figs. 9, 10 for 2D and Figs. 11, 12 for 3D. We make similar observations to the constant stencil case.

e) The Opteron is faster than the Xeon on the naive scheme but slower on PluTo and CATS: The main reason is the same as for the constant stencil: for the naive scheme the system bandwidth matters most while for the time skewing schemes the cache bandwidth is more important. However, the performance ratios between the Opteron and the Xeon for the naive scheme are now larger and for PluTo and CATS smaller than before, because the additional matrix transfers increases the influence of the system bandwidth speed on all schemes.

f) Performance in 2D is generally better than in 3D: This effect is further enforced by the fact that the 2D matrix has $N_S = 5$ bands while the 3D matrix has $N_S = 7$. This time the naive scheme is the least affected by the transition from 2D to 3D. Therefore, CATS's advantage over the naive scheme drops from 2.5x to 1.5x on the Opteron and from 8.2x to 6.2x on the Xeon. For PluTo it means that equal performance with the naive scheme drops to worse on the Opteron and much better performance drops to equal on the Xeon.

g) Comparison with stencil peak: The application of a constant stencil is already memory-bound so in the matrix case the memory wall is truly a devastating performance killer. The naive scheme and PluTo reach less than 2% of the available stencil peak performance on the Xeon in 3D. The comparison against the stencil peak benchmark reveals the true extent of the memory wall problem.

In general the naive scheme with the banded matrix runs around 2.5x–3x times slower than with constant stencils. The performance of PluTo and CATS is reduced by similar factors in 2D, but in 3D most reductions fall in the range 4x–5.5x. The additional matrix transfers make the performance depend on the low system bandwidth again. Although CATS clearly outperforms the other schemes, the comparison against the

stencil peak benchmark is less favorable. CATS stays above 20% in 2D and 10% in 3D. This is good in comparison to the 2% from above, but in case of infinite bandwidth we could still run up to 10 times faster.

C. FDTD Solver

The previous sections analyzed basic stencil computations on a scalar domain with constant or variable weights in detail. In practice, these basic stencil computations appear in different variations. In this section we examine one such variation that is often used to demonstrate the efficiency of time skewing schemes, namely a 2D Finite Difference Time Domain (FDTD) electromagnetic kernel.

For PluTo we use the code given in the paper [1], which is also included as a software example. For CATS we fuse the three loops in 2D FDTD manually to obtain a single kernel. Then we write a vectorized version of this kernel and pass its pointer to the naive scheme and CATS. Figs. 13, 14 show the results. Because this is a vector valued problem with 3 doubles for each point in the space-time, more data must be kept in cache which forces the wavefronts to become smaller. Not surprisingly the results are a slowed down version of the 2D constant stencil tests in Figs. 5, 6. PluTo has a small advantage over the naive implementation of around 1.2x on the Opteron and a clearer advantage of 1.7x on the Xeon. CATS beats the naive scheme by 1.7x (1.4x vs. PluTo) on the Opteron and 5.3x (3.2x vs. PluTo) on the Xeon.

D. Scalability

GFLOPS of ...	1 thread	2 threads	4 threads
CATS Opteron	1.7	3.3	6.4
CATS Xeon	5	9.6	13

The table above shows how CATS scales from one to four threads on the constant 7-point stencil for the 128 million elements problem in 3D with $T = 100$ iterations. Although this is a memory-bound problem, both the Opteron and the Xeon scale almost perfectly from one to two threads. Supported by higher system bandwidth (11.2 GB/s) the Opteron also scales well to four threads, while the lower system bandwidth (6.20 GB/s) of the Xeon limits the gains from additional cores.

E. Larger Stencils

GFLOPS of ...	$s = 1$	$s = 2$	$s = 3$
NaiveSSE Opteron	2.4	3.1	3.1
PluTo Opteron	1.5	0.9	0.9
CATS Opteron	6.4	7.5	4.7
GFLOPS of ...	$s = 1$	$s = 2$	$s = 3$
NaiveSSE Xeon	1.4	1.9	1.7
PluTo Xeon	3.7	4.3	1.9
CATS Xeon	13.0	8.5	4.6

Up to now we have shown results for the most common stencils of slope $s = 1$. Larger slopes worsen the surface area to volume ratio of the space-time tiles. Above we compare the performance of the constant 7-point stencil of slope 1, the 13-point stencil of slope 2, and the 19-point stencil of slope 3 for the 128 million elements problem in 3D with $T = 100$ iterations. We see that CATS maintains a clear advantage in all cases despite the different performance dependence of the schemes on the slope s.

F. Result Comparison

`A` ([5]) 3D Laplace (8 flops) $256^3 \times 100$ on Xeon X5550; `B` ([21]) 3D Jacobi (8 flops) $512^3 \times 100$ on Xeon X5550; `C` ([11]) 3D Jacobi (6 flops) $600^3 \times 100$ on Xeon X5550; `D` ([19]) 2D FDTD (11 flops) $2000^2 \times 2000$ on Xeon E5462; CATS on Xeon X5482.

giga updates/sec	A	B	C	D
paper	0.49	1.2	1.75	0.70
CATS	1.31	0.85	0.62	0.61

In the above table we compare CATS against most recent results from literature. `A` shows the maximum performance that can be achieved without time skewing, `B` and `C` serve as representatives of Wonnacott's wavefront processsing [8] adapted to multi-cores and enriched by shared cache optimization, and `D` shows strongest results of an automatic loop transformation framework, PTile [19]. In all cases we run exactly the same kernel as described in the corresponding paper. `A` uses an autotuner for parameter settings, `B` and `C` use manually selected parameters, `D` and CATS use internal formulas and heuristics. `A` to `D` run on Nehalem architecture with integrated memory controllers whereas `D` and CATS run on previous generation CPUs with FSB.

A fair comparison of *absolute* performance across different machines is difficult, e.g., our measured system bandwidth of 6.20 GB/s is three times smaller than the 18.5 GB/s measured by Wittmann et al. [11] on their Xeon X5550, or PluTo achieves 0.55 giga updates/sec on the Xeon E5462 [19] whereas PluTo on our machine runs at only 0.23 giga updates/sec. Therefore, in our detailed results we have chosen PluTo as an automatic, easy-to-use tool which can be quickly executed on any machine to provide a good baseline for a *relative* performance comparison. Most of our speedups against PluTo lie in the range 2x-4x and we have not seen this level of improvement previously in literature.

Note, that PluTo is a much more general tool and offers more functionality than the optimization of iterative stencil computations. Both in software and hardware there is always a trade-off between the specialization of a solution and its performance. We demonstrate in this paper that for the class of iterative stencil computations there is still significant room for improvement and hope that eventually these techniques will find their way into automatic transformation frameworks in order to reduce this gap.

IV. Conclusions

We have presented CATS, a cache accurate time skewing scheme for iterative stencil computations on multi-core processors. It is based on a novel usage of a wavefront traversal in multi-dimensional time skewing, an unconventional departure

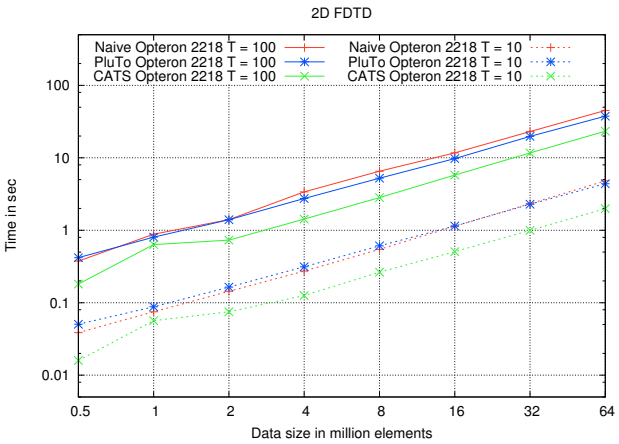

Fig. 13. Timings of the Opteron 2218 for FDTD in 2D. GFLOPS for 64 million elements with $T = 100$: NaiveSSE Opteron 1.6, PluTo Opteron 1.9, CATS Opteron 2.7 .

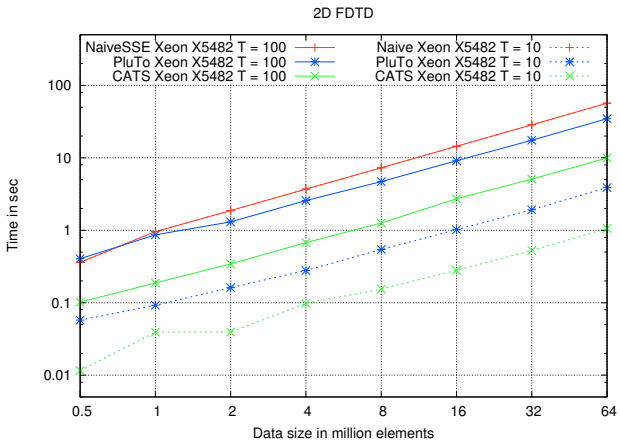

Fig. 14. Timings of the Xeon X5482 for FDTD in 2D. GFLOPS for 64 million elements with $T = 100$: NaiveSSE Xeon 1.2, PluTo Xeon 2.0, CATS Xeon 6.4 .

from the commonly used techniques of multi-dimensional tiling and multi-level tiling. The strategy is particularly successful on stencils of slope one, where the algorithm breaks the dependence on the low system bandwidth and achieves at least 50% of the stencil peak benchmark performance in 2D and 3D even when operating on gigabyte large domains. This is a significant improvement over the optimized naive scheme and the state-of-art in automatic optimization. For large stencils and banded matrices the system bandwidth limits the performance again but in comparison CATS maintains a clear advantage.

In future, we want to analyze and model the performance of CATS and include support for NUMA memory handling. In this way we will be able to study scalability on many-core shared memory machines.

REFERENCES

[1] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan, "A practical automatic polyhedral parallelizer and locality optimizer," *SIGPLAN Not.*, vol. 43, no. 6, pp. 101–113, 2008.

[2] M. Wilkes, "The memory gap," in *Proc. of Workshop on Solving the Memory Wall Problem*, Vancouver, BC, Canada, 2000, http://-www.ece.neu.edu/conf/wall2k/wilkes1.pdf.

[3] SEMATECH, "International technology roadmap for semiconductors (ITRS)," http://www.sematech.org/corporate/annual, 2009.

[4] M. A. Frumkin and R. F. Van der Wijngaart, "Tight bounds on cache use for stencil operations on rectangular grids," *Journal of ACM*, vol. 49, no. 3, pp. 434–453, 2002.

[5] S. Kamil, C. Chan, L. Oliker, J. Shalf, and S. Williams, "An auto-tuning framework for parallel multicore stencil computations," in *International Parallel & Distributed Processing Symposium (IPDPS)*, 2010.

[6] M. Wolf, "More iteration space tiling," in *Proceedings of Supercomputing '89*, 1989.

[7] Y. Song and Z. Li, "New tiling techniques to improve cache temporal locality," in *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, 1999.

[8] D. Wonnacott, "Using time skewing to eliminate idle time due to memory bandwidth and network limitations," in *Proceedings of International Parallel and Distributed Processing Symposium*, 2000.

[9] S. Kamil, K. Datta, S. Williams, L. Oliker, J. Shalf, and K. Yelick, "Implicit and explicit optimizations for stencil computations," in *MSPC '06: Proceedings of the 2006 workshop on Memory system performance and correctness*. ACM, 2006, pp. 51–60.

[10] K. Datta, S. Kamil, S. Williams, L. Oliker, J. Shalf, and K. Yelick, "Optimization and performance modeling of stencil computations on modern microprocessors," *SIAM Review*, vol. 51, no. 1, pp. 129–159.

[11] M. Wittmann, G. Hager, and G. Wellein, "Multicore-aware parallel temporal blocking of stencil codes for shared and distributed memory," in *Proc. Workshop on Large-Scale Parallel Processing (LSPP'10) at IPDPS'10*, 2010.

[12] S. Krishnamoorthy, M. Baskaran, U. Bondhugula, J. Ramanujam, A. Rountev, and P. Sadayappan, "Effective automatic parallelization of stencil computations," *SIGPLAN Not.*, vol. 42, no. 6, pp. 235–244, 2007.

[13] L. Renganarayana, M. Harthikote-Matha, R. Dewri, and S. Rajopadhye, "Towards optimal multi-level tiling for stencil computations," in *Proceedings of International Parallel and Distributed Processing Symposium*. IEEE Computer Society, 2007.

[14] A. Nguyen, N. Satish, J. Chhugani, C. Kim, and P. Dubey, "3.5-D blocking optimization for stencil computations on modern CPUs and GPUs," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10, 2010, pp. 1–13.

[15] R. Strzodka, M. Shaheen, D. Pajak, and H.-P. Seidel, "Cache oblivious parallelograms in iterative stencil computations," in *ICS '10: Proceedings of the 24th ACM International Conference on Supercomputing*. ACM, 2010, pp. 49–59.

[16] M. Griebl, "Automatic parallelization of loop programs for distributed memory architectures," University of Passau, Jun. 2004, habilitation thesis.

[17] D. Kim, L. Renganarayanan, D. Rostron, S. V. Rajopadhye, and M. M. Strout, "Multi-level tiling: M for the price of one," in *Proceedings of the ACM/IEEE Conference on Supercomputing*, 2007, p. 51.

[18] A. Hartono, M. M. Baskaran, C. Bastoul, A. Cohen, S. Krishnamoorthy, B. Norris, J. Ramanujam, and P. Sadayappan, "Parametric multi-level tiling of imperfectly nested loops," in *Proceedings of the 23rd International Conference on Supercomputing*, 2009, pp. 147–157.

[19] M. M. Baskaran, A. Hartono, S. Tavarageri, T. Henretty, J. Ramanujam, and P. Sadayappan, "Parametrized tiling revisited," in *Proc. of the International Symposium on Code Generation and Optimization (CGO'10)*, 2010.

[20] H. Jia-Wei and H. T. Kung, "I/o complexity: The red-blue pebble game," in *STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*. ACM, 1981, pp. 326–333.

[21] G. Wellein, G. Hager, T. Zeiser, M. Wittmann, and H. Fehske, "Efficient temporal blocking for stencil computations by multicore-aware wavefront parallelization," in *Proc. IEEE International Computer Software and Applications Conference (COMPSAC'09)*, 2009.

[22] D. Orozco and G. Gao, "Mapping the FDTD application to many-core chip architectures," University of Delaware, Tech. Rep., Mar. 2009.

[23] L. Liu and Z. Li, "Improving parallelism and locality with asynchronous algorithms," in *Proceedings ACM symposium on Principles and practice of parallel programming*, ser. PPoPP '10, 2010, pp. 213–222.