Cache Oblivious Stencil Computations

Matteo Frigo and Volker Strumpen*
IBM Austin Research Laboratory
11501 Burnet Road, Austin, TX 78758

May 25, 2005

Abstract

We present a cache oblivious algorithm for stencil computations, which arise for example in finite-difference methods. Our algorithm applies to arbitrary stencils in n-dimensional spaces. On an "ideal cache" of size Z, our algorithm saves a factor of $\Theta(Z^{1/n})$ cache misses compared to a naive algorithm, and it exploits temporal locality optimally throughout the entire memory hierarchy.

1 Introduction

A ***stencil*** defines the computation of an element in an n-dimensional spatial grid at time t as a function of neighboring grid elements at time $t-1, \ldots, t-k$. This computational pattern arises in many contexts, including explicit finite-difference methods [5]. The n-dimensional grid plus the time dimension span an $(n+1)$-dimensional ***spacetime***.[1] Each spacetime point, except possibly for initial and boundary values, is computed by means of a ***computational kernel***. In practical implementations of stencils, there is often no need to store the entire spacetime; storing a bounded number of time steps per space point is sufficient. For example, consider a 3-point stencil in 1-dimensional space (2-dimensional spacetime): Because the computation of a point at time t depends only upon three points at time $t-1$, it is sufficient to store two time steps only. For this important case of stencil computations with kernels that require a bounded amount of storage per space point, we present a cache-oblivious algorithm that exploits a memory hierarchy optimally.

A ***stencil computation*** is a traversal of spacetime in an order that respects the data dependencies imposed by the stencil. The simplest stencil computation applies the kernel to all spacetime points at time t before computing any point at time $t+1$. On a memory hierarchy, if the size of the storage required for the spacetime points of one time step exceeds the cache size Z, this simple computation incurs a number of cache misses proportional to p, where p is the number of spacetime points computed. In contrast, when traversing a sufficiently large rectangular region of $(n+1)$-dimensional spacetime spanning a sufficiently large time interval, our algorithm incurs at most $O(p/Z^{1/n})$ cache misses on a machine with an ideal cache [2] of size Z. This number of cache misses matches the lower bound of Hong and Kung [3] within a constant factor. Unlike blocked algorithms, our algorithm is ***cache oblivious***: it does not contain the cache size as a parameter [2]. Therefore, the algorithm makes optimal use of each level in a multi-level memory hierarchy. In addition, our algorithm applies to arbitrary stencils and arbitrary space dimension $n > 0$.

Cache oblivious algorithms for special cases of stencil computations have been proposed before. Bilardi and Preparata [1] discuss cache oblivious algorithms for the related problem of simulating large parallel machines on smaller machines in a spacetime-efficient manner. Their algorithms apply to 1-dimensional and 2-dimensional spaces and do not generalize to higher dimensions. In fact, the authors declare the 3-dimensional case, and implicitly higher dimensional spaces, to be an open problem. Prokop [4] gives a cache oblivious stencil algorithm for a 3-point stencil in 1-dimensional space, and proves that the algorithm is optimal. His algorithm is restricted to square spacetime regions, and it does not extend to higher dimensions. We are unaware of any previous solution of the general n-dimensional case.

We introduce a simplified cache-oblivious stencil algorithm for 1-dimensional grids and a 3-point stencil in

*This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under contract No. NBCH30390004.

To appear in the Proceedings of the 19th ACM International Conference on Supercomputing (ICS05).

[1] We emphasize that we denote the dimensionality of space as n and the dimensionality of spacetime as $n+1$. When using the term *space*, we implicitly exclude the time dimension. When we include the time dimension, we refer to *spacetime*.

```
void walk1(int t0, int t1, int x0, int dx0, int x1, int dx1)
{
    int dt = t1 - t0;

    if (dt == 1) {
        /* base case */
        int x;
        for (x = x0; x < x1; ++x)
            kernel(t0, x);
    } else if (dt > 1) {
        if (2 * (x1 - x0) + (dx1 - dx0) * dt >= 4 * dt) {
            /* space cut */
            int xm = (2 * (x0 + x1) + (2 + dx0 + dx1) * dt) / 4;
            walk1(t0, t1, x0, dx0, xm, -1);
            walk1(t0, t1, xm, -1, x1, dx1);
        } else {
            /* time cut */
            int s = dt / 2;
            walk1(t0, t0 + s, x0, dx0, x1, dx1);
            walk1(t0 + s, t1, x0 + dx0 * s, dx0, x1 + dx1 * s, dx1);
        }
    }
}
```

Figure 1: Procedure `walk1` for traversing a 2-dimensional spacetime spanned by a 1-dimensional grid and time for a 3-point stencil.

Section 2. Then, we present our algorithm for arbitrary stencils and n-dimensional grids in Section 3, and prove bounds on the number of cache misses in Section 4.

2 One-dimensional Stencil Algorithm

Procedure `walk1` in Fig. 1 traverses rectangular spacetime (t, x), where $0 \le t < T$ and $0 \le x < N$. For simpler illustration, we restrict this procedure to observe the dependencies of a 3-point stencil, i.e. the procedure visits point $(t+1, x)$ after visiting points $(t, x-1)$, (t, x), and $(t, x+1)$. Although we are ultimately interested in traversing rectangular spacetime regions, procedure `walk1` operates on more general trapezoidal regions such as the one shown in Fig. 2. For integers t_0, t_1, x_0, $\dot{x}_0$, x_1, and $\dot{x}_1$, we define the ***trapezoid*** $\mathcal{T}(t_0, t_1, x_0, \dot{x}_0, x_1, \dot{x}_1)$ to be the set of integer pairs (t, x) such that $t_0 \le t < t_1$ and $x_0 + \dot{x}_0(t - t_0) \le x < x_1 + \dot{x}_1(t - t_0)$. (We use the Newtonian notation $\dot{x} = dx/dt$.) The ***height*** of the trapezoid is $\Delta t = t_1 - t_0$, and we define the ***width*** to be the average of the lengths of the two parallel sides, i.e. $w = (x_1 - x_0) + (\dot{x}_1 - \dot{x}_0)\Delta t/2$. The ***center*** of the trapezoid is point (t, x), where $t = (t_0 + t_1)/2$ and $x = (x_0 + x_1)/2 + (\dot{x}_0 + \dot{x}_1)\Delta t/4$ (i.e., the average of the four corners). The ***volume*** of the trapezoid is the number of points in the trapezoid: $\mathrm{Vol}(\mathcal{T}) = |\mathcal{T}|$. We only consider ***well-defined trapezoids***, for which

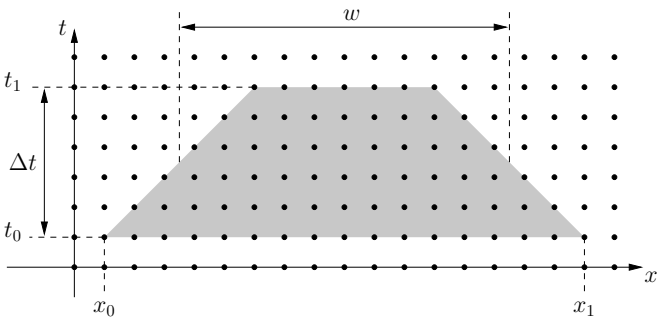

Figure 2: Illustration of the trapezoid $\mathcal{T}(t_0, t_1, x_0, \dot{x}_0, x_1, \dot{x}_1)$ for $\dot{x}_0 = 1$ and $\dot{x}_1 = -1$. The trapezoid includes all points in the shaded region, except for those on the top and right edges.

these three conditions hold: $t_1 \ge t_0$, $x_1 \ge x_0$, and $x_1 + \Delta t \cdot \dot{x}_1 \ge x_0 + \Delta t \cdot \dot{x}_0$.

The special case $\mathcal{T}(t_0, t_1, x_0, 0, x_1, 0)$ denotes a rectangular region. In this section, we restrict the ***slopes*** $\dot{x}_0$ and $\dot{x}_1$ of the edges to assume values 1, -1, or 0, delaying the general case until Section 3.

Fig. 1 shows procedure `walk1` as a recursive C function whose parameters denote the trapezoid $\mathcal{T}(t_0, t_1, x_0, \dot{x}_0, x_1, \dot{x}_1)$. The procedure visits all points of the trapezoid in an order that respects the stencil dependencies. Procedure `walk1` decomposes the trapezoid

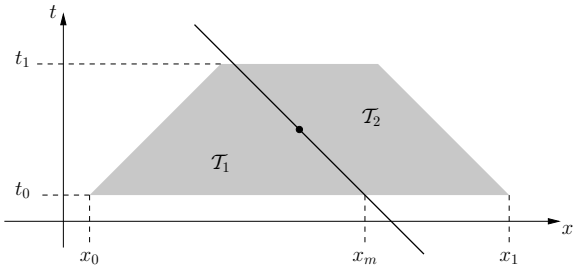

Figure 3: Illustration of a *space cut*. When the space dimension is "large enough" (see text), procedure `walk1` cuts the trapezoid along the line of slope -1 through its center.

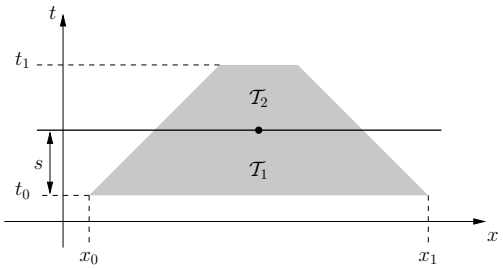

Figure 4: Illustration of a *time cut*: procedure `walk1` cuts the trapezoid along the horizontal line through its center.

recursively into smaller trapezoids, according to the following rules.

Base case: If the height is 1, then the trapezoid consists of the line of spacetime points (t_0, x) with $x_0 \le x < x_1$. The procedure visits all these points, calling the application-specific procedure `kernel`. The traversal order is not important because these points do not depend on each other.

Space cut: If the width is at least twice the height, then we cut the trapezoid along the line with slope -1 through the center of the trapezoid, cf. Fig. 3. The recursion first traverses trapezoid $\mathcal{T}_1 = \mathcal{T}(t_0, t_1, x_0, \dot{x}_0, x_m, -1)$, and then trapezoid $\mathcal{T}_2 = \mathcal{T}(t_0, t_1, x_m, -1, x_1, \dot{x}_1)$. This traversal order is valid because no point in $\mathcal{T}_1$ depends upon any point in $\mathcal{T}_2$.

From Fig. 3, we obtain

$$x_m = \frac{1}{2}(x_0 + x_1) + \frac{1}{4}(\dot{x}_0 + \dot{x}_1)\Delta t + \frac{1}{2}\Delta t \ .$$

Time cut: Otherwise, we cut the trapezoid along the horizontal line through the center, cf. Fig. 4. The recursion first traverses trapezoid $\mathcal{T}_1 = \mathcal{T}(t_0, t_0 + s, x_0, \dot{x}_0, x_1, \dot{x}_1)$, and then trapezoid $\mathcal{T}_2 = \mathcal{T}(t_0 + s, t_1, x_0 + \dot{x}_0 s, \dot{x}_0, x_1 + \dot{x}_1 s, \dot{x}_1)$, where $s = \Delta t/2$. The order of these traversals is valid because no point in $\mathcal{T}_1$ depends on any point in $\mathcal{T}_2$.

In the two recursive cases, even though the computation of x_m or s is based on integer divisions with truncation or rounding, one can prove that both $\mathcal{T}_1$ and $\mathcal{T}_2$ are well-defined and nonempty no matter how the quotient is truncated or rounded. Thus, procedure `walk1` is guaranteed to terminate because it reduces the original problem to strictly smaller subproblems.

Procedure `walk1` traverses the rectangular region $\mathcal{T}(0, T, 0, 0, N, 0)$ as a special case. Perhaps surprisingly, the same procedure also works for *cylindrical* regions in which point $(t+1, x)$ depends on points $(t, (x-1) \bmod N)$, (t, x), and $(t, (x+1) \bmod N)$. To use `walk1` in this fashion, invoke it on $\mathcal{T}(0, T, 0, 1, N, 1)$ and interpret all indices $(\mathrm{mod} N)$ in the kernel. Fig. 5 illustrates how this scheme works for $N = T = 10$. In the left part of the figure, we mark each spacetime point with consecutive integers in the order in which the point is visited. Thus, point $(t, x) = (0, 0)$ is visited first, point $(0, 1)$ second, etc. The right part of the figure shows the recursively nested trapezoids produced by `walk1`. Procedure `walk1` traverses the spacetime region in the black trapezoid rather than the grey spacetime rectangle, but the traversal order is consistent with a cylindrical stencil problem if all indices are interpreted $(\mathrm{mod} N)$ in the kernel.

3 Multi-dimensional Algorithm

In this section, we generalize procedure `walk1` from Section 2 in two ways. First, we relax the restriction to the 3-point stencil and allow arbitrary stencils. In particular, we allow spacetime point $(t+1, x)$ to depend on all points $(t, x+k)$, where $|k| \le \sigma$.[2] Second, we generalize our procedure for arbitrary-dimensional spacetime. Fig. 6 shows a C implementation of the multi-dimensional `walk` procedure.

We first extend procedure `walk1` to work for $|\dot{x}_0| \le \sigma$ and $|\dot{x}_1| \le \sigma$, for an arbitrary slope σ. In the "space cut" case, we cut along a line of slope $dx/dt = -\sigma$ through the center. This cut guarantees that no point in the left trapezoid $\mathcal{T}_1$ depends upon any point in the right trapezoid $\mathcal{T}_2$. Therefore, the modified algorithm traverses spacetime in an order consistent with the stencil dependencies. The expression for x_m (see Fig. 3) for arbitrary slope σ becomes

$$x_m = \frac{1}{2}(x_0 + x_1) + \frac{1}{4}(\dot{x}_0 + \dot{x}_1)\Delta t + \frac{1}{2}\sigma\Delta t \ .$$

[2] The generalization of the stencil with respect to dependencies of time steps $t, t-1, \ldots, t-j$ for $j > 1$ follows by induction, and by choosing slope $\sigma = \max_j(\sigma_j)$, where σ_j is the slope between time steps $t+1-j$ and $t-j$.

$t\backslash x$	0	1	2	3	4	5	6	7	8	9
9	79	88	89	90	94	95	97	98	99	78
8	76	77	85	86	87	92	93	96	74	75
7	71	72	73	82	83	84	91	68	69	70
6	62	63	66	67	80	81	54	55	58	59
5	57	60	61	64	65	50	51	52	53	56
4	45	47	48	49	28	29	38	39	40	44
3	42	43	46	24	25	26	27	35	36	37
2	34	41	18	19	20	21	22	23	32	33
1	31	4	5	8	9	12	13	16	17	30
0	0	1	2	3	6	7	10	11	14	15

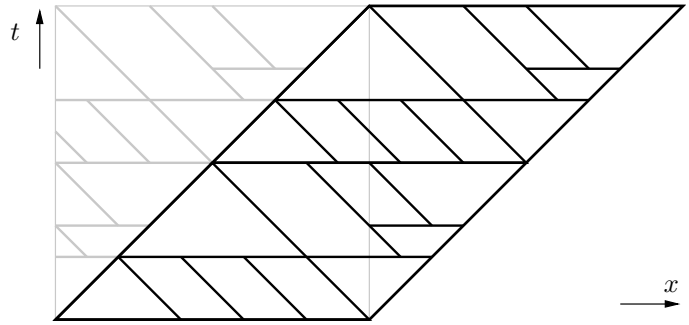

Figure 5: Cache-oblivious traversal of 1-dimensional spacetime for $N = T = 10$.

The space cut can be applied when width $w \geq 2\sigma\Delta t$, which guarantees that the two trapezoids that result from the cut are well-defined and nonempty.

Next, we consider n-dimensional stencils, where $n > 0$ is the number of space dimensions (i.e., excluding time). A n-dimensional trapezoid $\mathcal{T}(t_0, t_1, x_0^{(i)}, \dot{x}_0^{(i)}, x_1^{(i)}, \dot{x}_1^{(i)})$, where $0 \leq i < n$, is the set of integer tuples $(t, x^{(0)}, x^{(1)}, \ldots, x^{(n-1)})$ such that $t_0 \leq t < t_1$ and $x_0^{(i)} + \dot{x}_0^{(i)}(t - t_0) \leq x^{(i)} < x_1^{(i)} + \dot{x}_1^{(i)}(t - t_0)$ for all $0 \leq i < n$. Informally, for each dimension i, the projection of the multi-dimensional trapezoid onto the $(t, x^{(i)})$ plane looks like the 1-dimensional trapezoid in Fig. 2. Consequently, we can apply the same recursive decomposition that we used in the 1-dimensional case: if any dimension i permits a space cut in the $(t, x^{(i)})$ plane, then cut space in dimension i. Otherwise, if none of the space dimensions can be split, cut time in the same fashion as in the 1-dimensional case.

Procedure `walk` in Fig. 6 implements the multi-dimensional trapezoid by means of an array of tuples of type `C`, the ***configuration tuple*** for one space dimension. Fig. 6 hides the traversal of the n-dimensional base case in procedure `basecase`. We leave it as a programming exercise to develop this procedure, which visits all points of the rectangular parallelepiped at time step t_0 in all space dimensions by calling application specific procedure `kernel`.

4 Analysis of Cache Misses

In this section, we prove Theorem 2, which states that procedure `walk` incurs $O(\text{Vol}(\mathcal{T})/Z^{1/n})$ cache misses on a machine with an ideal cache of size Z, provided that the kernel operates "in-place," that the cache is "ideal," and that the trapezoid is "sufficiently large."

We say that the kernel of a stencil computation is ***in-place*** if for some k, the kernel stores spacetime point $(t, x^{(0)}, x^{(1)}, \ldots, x^{(n-1)})$ in the same memory locations where spacetime point $(t - k, x^{(0)}, x^{(1)}, \ldots, x^{(n-1)})$ was stored, destroying the old value. Our analysis only applies to in-place kernels, but this condition is true in most practical situations[3].

We use the ***ideal cache*** model from [2]. The ideal cache is fully associative and implements an optimal replacement policy. While [2] allows the cache to be partitioned into cache lines of size L, we restrict ourselves to the case $L = 1$ in this paper.

We start with a lemma that relates the volume and the surface of an n-dimensional trapezoid.

Lemma 1 Let $\mathcal{T}$ be the n-dimensional trapezoid $\mathcal{T}(t_0, t_1, x_0^{(i)}, \dot{x}_0^{(i)}, x_1^{(i)}, \dot{x}_1^{(i)})$, where $0 \leq i < n$. Let $\mathcal{T}$ be well-defined, w_i be the width of the trapezoid in dimension i, and let $m = \min(\Delta t, w_0, w_1, \ldots, w_{n-1})/2$. Then, there are $O((1 + n)\text{Vol}(\mathcal{T})/m)$ points on the surface of the trapezoid.

Proof: The volume of the trapezoid is the sum for all time slices of the number of points in the (rectangular) slice:

$$\text{Vol}(\mathcal{T}) = \sum_{-\Delta t/2 \leq t < \Delta t/2} \prod_{0 \leq i < n} (w_i + \vartheta_i t) \ ,$$

where $\vartheta_i = \dot{x}_1^{(i)} - \dot{x}_0^{(i)}$. Define the auxiliary function $V(s)$ as:

$$V(s) = \sum_{-(\Delta t/2) - s \leq t < (\Delta t/2) + s} \prod_{0 \leq i < n} (w_i + 2s + \vartheta_i t) \ . \quad (1)$$

Then, we have $\text{Vol}(\mathcal{T}) = V(0)$, and the number of points on the surface $\partial\text{Vol}(\mathcal{T})$ is at most $V(1) - V(0)$. We

[3] If the kernel stores the whole spacetime into distinct memory locations, then each point in the trapezoid must obviously incur a cache miss and no savings are possible.

```
typedef struct { int x_0, ẋ_0, x_1, ẋ_1 } C;

void walk(int t_0, int t_1, C c[n])
{
    int Δt = t_1 - t_0;

    if (Δt == 1) {
        basecase(t_0, c);
    } else if (Δt > 1) {
        C *p;

        /* for all dimensions, try to cut space */
        for (p = c; p < c + n; ++p) {
            int x_0 = p->x_0, x_1 = p->x_1, ẋ_0 = p->ẋ_0, ẋ_1 = p->ẋ_1;
            if (2 * (x_1 - x_0) + (ẋ_1 - ẋ_0) * Δt >= 4 * σ * Δt) {
                /* cut space dimension *p */
                C save = *p;   /* save configuration *p */
                int x_m = (2 * (x_0 + x_1) + (2 * σ + ẋ_0 + ẋ_1) * Δt) / 4;
                *p = (C){ x_0, ẋ_0, x_m, -σ };   walk(t_0, t_1, c);
                *p = (C){ x_m, -σ, x_1, ẋ_1 };   walk(t_0, t_1, c);
                *p = save;     /* restore configuration *p */
                return;
            }
        }

        {
            /* because no space cut is possible, cut time */
            int s = Δt / 2;
            C newc[n];
            int i;

            walk(t_0, t_0 + s, c);

            for (i = 0; i < n; ++i) {
                newc[i] = (C){ c[i].x_0 + c[i].ẋ_0 * s, c[i].ẋ_0,
                               c[i].x_1 + c[i].ẋ_1 * s, c[i].ẋ_1 };
            }

            walk(t_0 + s, t_1, newc);
        }
    }
}
```

Figure 6: A C99 implementation of the multi-dimensional `walk` procedure. The code assumes that n is a compile-time constant. The base case and the definition of the slope σ are not shown.

approximate the sum in Eq. (1) with the integral

$$V(s) = \int_{-(\Delta t/2)-s}^{(\Delta t/2)+s} \prod_{0 \le i < n} (w_i + 2s + \vartheta_i t)\, dt$$

and the surface $\partial\mathrm{Vol}(\mathcal{T})$ with the derivative $V'(0)$. After the substitution $t = (m+s)r$, we obtain

$$V(s) = \int_{-g(s)}^{g(s)} (m+s) f(s,r)\, dr \ ,$$

where $g(s) = ((\Delta t/2) + s)/(m+s)$ and

$$f(s,r) = \prod_{0 \le i < n} (w_i + (2 + \vartheta_i r)s + \vartheta_i r m) \ .$$

The derivative $V'(0)$ is

$$V'(0) = g'(0) \cdot m \cdot \big(f(0, g(0)) + f(0, -g(0))\big) + \int_{-g(0)}^{g(0)} \left(f(0,r) + m \cdot \left.\frac{df(s,r)}{ds}\right|_{s=0} \right) dr \ . \tag{2}$$

Observe that

$$m \cdot \left.\frac{df(s,r)}{ds}\right|_{s=0} = f(0,r) \cdot \sum_{0 \le j < n} \frac{2m + \vartheta_j r m}{w_j + \vartheta_j r m} \le n f(0,r) \ , \tag{3}$$

where the inequality holds because $(2m + \vartheta_j r m)/(w_j + \vartheta_j r m) \le 1$, which holds because we have $2m \le w_j$ by definition of m, and because we have $w_j + \vartheta_j r m \ge 0$ since the trapezoid is well-defined.

Further observe that, because $m \le \Delta t/2$ holds by definition of m, we have that $g'(s) = (m - \Delta t/2)/(m+s)^2 \le 0$. Because the trapezoid is well-defined, we have $f(s,r) \ge 0$ and $m \ge 0$. Therefore, we obtain

$$g'(0) \cdot m \cdot \big(f(0, g(0)) + f(0, -g(0))\big) \le 0 \ . \tag{4}$$

By substituting Eqs. (3) and (4) into Eq. (2), we obtain the result $V'(0) \le (1+n)V(0)/m$, and the lemma follows. Q.E.D.

Theorem 2 Let $\mathcal{T}$ be the well-defined n-dimensional trapezoid $\mathcal{T}(t_0, t_1, x_0^{(i)}, \dot{x}_0^{(i)}, x_1^{(i)}, \dot{x}_1^{(i)})$. Let procedure `walk` traverse $\mathcal{T}$ and execute a kernel in-place on a machine with an ideal cache of size Z. Assume that $\Delta t = \Omega(Z^{1/n})$ and that $w_i = \Omega(Z^{1/n})$ for all i, where w_i is the width of the trapezoid in dimension i. Then, procedure `walk` incurs at most $O(\mathrm{Vol}(\mathcal{T})/Z^{1/n})$ cache misses.

Proof: (Sketch) Procedure `walk` recursively cuts a large trapezoid into smaller trapezoids. During this recursion, a sub-trapezoid $\mathcal{S}$ eventually becomes so small that it has $\Theta(Z)$ points on its surface. Because the problem is in-place, all spacetime points in $\mathcal{S}$ can be computed with $O(\partial\mathrm{Vol}(\mathcal{S}))$ cache misses, since the cache is ideal by assumption.

If a space dimension i exists for which $w_i \ge 2\sigma\Delta t$, then `walk` cuts space dimension i, and otherwise it cuts time. Consequently, for sub-trapezoid $\mathcal{S}$ we have $\Delta t = \Theta(w_i)$ for all i. Therefore, we have $\Delta t = \Omega((\partial\mathrm{Vol}(\mathcal{S}))^{1/n}) = \Omega(Z^{1/n})$. From Lemma 1, we obtain $\partial\mathrm{Vol}(\mathcal{S}) = O(\mathrm{Vol}(\mathcal{S})/\Delta t)$. Thus, the number of cache misses for executing $\mathcal{S}$ is $O(\mathrm{Vol}(\mathcal{S})/Z^{1/n})$. The theorem follows by adding the cache misses incurred by all such sub-trapezoids. Q.E.D.

References

[1] Gianfranco Bilardi and Franco P. Preparata. Upper bounds to processor-time tradeoffs under bounded-speed message propagation. In *SPAA '95: Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, pages 185–194. ACM Press, 1995.

[2] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *Proc. 40th Ann. Symp. Foundations of Computer Science (FOCS '99)*, New York, USA, October 1999.

[3] Jia-Wei Hong and H. T. Kung. I/O complexity: the red-blue pebbling game. In *Proc. Thirteenth Ann. ACM Symp. Theory of Computing*, pages 326–333, Milwaukee, 1981.

[4] Harald Prokop. Cache-oblivious algorithms. Master's thesis, Massachusetts Inst. of Technology, June 1999.

[5] G. D. Smith. *Numerical Solution of Partial Differential Equations: Finite Difference Methods*. Oxford University Press, 3rd edition, 1985.