Cache Oblivious Stencil Computations

Matteo Frigo and Volker Strumpen*
IBM Austin Research Laboratory 11501 Burnet Road, Austin, TX 78758
May 25, 2005

Presented by Vishaal Ram, 4/11/24
Stencil Computations
what is a stencil?

- a computation defined on an n dimensional grid along with a time parameter $t$
- each value on the grid at a time $t$ is a function of the neighboring grid cells at time $t-1$, $t-2$, ..., $t-k$
- the input is a set of initial value $a_0$ while the output $T$ time steps later is $a_T$
examples

- If a stencil is a p-point stencil, the value depends on its p neighbors in the previous timestep.
- The n dimensions plus the time dimension together span the (n+1) dimension spacetime.
heat diffusion

- one notable example is heat diffusion which represents a 5-point 2D stencil on a discrete grid:
- the update function is known as the computational kernel

Let $h_t(x, y)$ be the heat at point $(x, y)$ at time $t$.

Heat Equation

$$\frac{\partial h}{\partial t} = \alpha \left( \frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} \right), \quad \alpha = \text{thermal diffusivity}$$

Update Equation (on a discrete grid)

$$h_{t+1}(x, y) = h_t(x, y)$$

$$+ c_x (h_t(x+1, y) - 2h_t(x, y) + h_t(x-1, y))$$

$$+ c_y (h_t(x, y+1) - 2h_t(x, y) + h_t(x, y-1))$$
Stencil Computation Algorithms
standard implementation

- the naive algorithm involves applying the computational kernel to all points at time step \( t \) before timestep \( t+1 \)
- If the number of points in at each time step exceeds the cache size \( Z \), this computation incurs \( O(p) \) cache misses where \( p \) is the number of points computed

```
for t ← 1 to T do
  for i ← 0 to N - 1 do
    compute \( a_t[i] \) from \( a_{t-1} \) using the stencil
```
main result

- The paper presents a novel stencil computation algorithm that when traversing a large rectangular region of (n+1) dimensional spacetime, incurs $O(p/Z^{n+1})$ cache misses.
  - this matches a lower bound proved by Hong and Kong [3] by a constant factor
  - applies to arbitrary stencil and dimension
- this algorithm is also cache oblivious
  - does not contain the cache size as a parameter
One-Dimensional Stencil Algorithm
**walk1**

- We define a procedure `walk1` that traverses a rectangular region \(0 \leq t < T\) and \(0 \leq x < N\).
- For simplicity, we restrict the computation to a 3-point stencil:
  - \((t, x)\) depends on \((t-1, x-1), (t-1, x), (t-1, x+1)\).
- Instead of just considering rectangular regions, we instead consider a more general trapezoidal region with additional parameters \(x_0\) and \(x_1\).

```c
void walk1(int t0, int t1, int x0, int x1, int x_tilde0, int x_tilde1)
{
    int \Delta t = t1 - t0;

    if (\Delta t == 1) {
        /* base case */
        int x;
        for (x = x0; x < x1; ++x)
            kernel(t0, x);
    } else if (\Delta t > 1) {
        if (2 * (x1 - x0) + (\dot{x}_1 - \dot{x}_0) * \Delta t >= 4 * \Delta t) {
            /* space cut */
            int x_m = (2 * (x0 + x1) + (2 + \dot{x}_0 + \dot{x}_1) * \Delta t) / 4;
            walk1(t0, t1, x0, x_m, \dot{x}_0);
            walk1(t0, t1, x_m, x1, \dot{x}_1);
        } else {
            /* time cut */
            int s = \Delta t / 2;
            walk1(t0, t0 + s, x0, \dot{x}_0, x1, \dot{x}_1);
            walk1(t0 + s, t1, x0 + s, \dot{x}_0 + s, x1 + \dot{x}_1 * s, \dot{x}_1);
        }
    }
}
```
trapezoid

For integers $t_0, t_1, x_0, \dot{x}_0, x_1, \dot{x}_1$ we define trapezoid $T(t_0, t_1, x_0, \dot{x}_0, x_1, \dot{x}_1)$ to be the set of points that satisfy $t_0 \leq t \leq t_1$, $x_0 + \dot{x}_0(t - t_0) < x < x_1 + \dot{x}_1(t - t_0)$.

The **height** is computed as $\Delta T = t_1 - t_0$

The **width** is the average lengths of the parallel sides: $w = (x_1 - x_0) + (\dot{x}_1 - \dot{x}_0) \Delta T / 2$

The **center** is the average of the four corners: $x = (x_0 + x_1) / 2 + (\dot{x}_0 + \dot{x}_1) \Delta t / 4$

The **volume** $|T|$ is the number of points in the trapezoid.

Assume for now that the special case with slopes zero denotes the rectangular region.

Figure 2: Illustration of the trapezoid $T(t_0, t_1, x_0, \dot{x}_0, x_1, \dot{x}_1)$ 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.
walk1 steps

- the algorithm works by recursively decomposing the region into smaller rectangles
- The base case is when the height is one
- Otherwise we perform one of two cuts dividing the trapezoid in half, recursing on each one

Base case: If the height is 1, then the trapezoid consists of the line of spacetime points \((t_0, x)\) with \(x_0 \leq x < x_1\). 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.

```c
int Δt = t_1 - t_0;

if (Δt == 1) {
    /* base case */
    int x;
    for (x = x_0; x < x_1; ++x)
        kernel(t_0, x);
```
Space Cut

- If the width is long enough, perform a diagonal cut from the center splitting the region into another trapezoid and a parallelogram. Then recurse on the trapezoid first.

**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 $T_1 = T(t_0, t_1, x_0, \dot{x}_0, x_m, -1)$, and then trapezoid $T_2 = T(t_0, t_1, x_m, -1, x_1, \dot{x}_1)$. This traversal order is valid because no point in $T_1$ depends upon any point in $T_2$.

From Fig. 3, we obtain

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

- otherwise perform a time cut dividing the region into two trapezoids by cutting horizontally through the center. Then recurse on the bottom region first

**Time cut:** Otherwise, we cut the trapezoid along the horizontal line through the center, cf. Fig. 4. The recursion first traverses trapezoid $T_1 = T(t_0, t_0 + s, x_0, \dot{x}_0, x_1, \dot{x}_1)$, and then trapezoid $T_2 = T(t_0 + s, t_1, x_0 + \dot{x}_0 s, \dot{x}_0, x_1 + \dot{x}_1 s, \dot{x}_1)$, where $s = \Delta t/2$. The order of these traversals is valid because no point in $T_1$ depends on any point in $T_2$. 

```c
} else {
    /* time cut */
    int s = \Delta t / 2;
    walk1(t_0, t_0 + s, x_0, \dot{x}_0, x_1, \dot{x}_1);
    walk1(t_0 + s, t_1, x_0 + \dot{x}_0 * s, \dot{x}_0, x_1 + \dot{x}_1 * s, \dot{x}_1);
}
```
Figure 4: Illustration of a time cut: procedure walk1 cuts the trapezoid along the horizontal line through its center.
summary

- we can guarantee that both cuts always produce valid and well defined regions.
- we can show that this procedure also works on cylindrical where \((t+1, x)\) depends on \((t, (x-1) \mod N), (t, x \mod N), (t, (x+1) \mod N)\).
cylindrical traversal

- traversal order for cylindrical traversal where $N=T=10$

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>79</td>
</tr>
<tr>
<td>8</td>
<td>76</td>
</tr>
<tr>
<td>7</td>
<td>71</td>
</tr>
<tr>
<td>6</td>
<td>62</td>
</tr>
<tr>
<td>5</td>
<td>57</td>
</tr>
<tr>
<td>4</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>34</td>
</tr>
<tr>
<td>1</td>
<td>31</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 5: Cache-oblivious traversal of 1-dimensional spacetime for $N = T = 10$. 
Extension to Multiple Dimensions and Arbitrary Stencils
**arbitrary stencils**

- We first extend walk 1 to a spacetime point \((t+1, x)\) to depend on any \((t, x+k)\) for any
  \[|x - k| \leq \sigma^2\]

- To do this, we simply modify our space cut so that we cut along the center with a line of slope \(\frac{dx}{dt} = -\sigma\). This guarantees that two points in the first region depend on a point in the second region. This cut can be applied whenever \(w \geq 2\sigma \Delta t\), which guarantees the two regions are well defined.
We extend the definition of the 2D trapezoid to an arbitrary number of dimensions. If any of the dimensions permits a space cut, we cut along that dimension and recurse, otherwise we perform a time cut. As the projection onto any dimension matches our 2D case, this algorithm also generates a valid stencil traversal.

A $n$-dimensional trapezoid $\mathcal{T}(t_0, t_1, x_0^{(i)}, \hat{x}_0^{(i)}, x_1^{(i)}, \hat{x}_1^{(i)})$, 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 $x_0^{(i)} + \hat{x}_0^{(i)} (t - t_0) \leq x^{(i)} < x_1^{(i)} + \hat{x}_1^{(i)} (t - t_0)$ 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.
typedef struct { int x0, x1, x̄1 } C;

void walk(int t₀, int t₁, C c[n])
{
    int Δt = t₁ - t₀;

    if (Δt == 1) {
        basecase(t₀, c);
    } else if (Δt > 1) {
        C *p;

        /* for all dimensions, try to cut space */
        for (p = c; p < c + n; ++p) {
            int x₀ = p->x₀, x₁ = p->x₁, x̄₀ = p->x̄₀, x̄₁ = p->x̄₁;
            if (2 * (x₁ - x₀) + (x̄₁ - x̄₀) + Δt >= 4 * σ * Δt) {
                /* cut space dimension *p */
                C save = *p;  /* save configuration *p */
                int x_m = (2 * (x₀ + x₁) + (2 * σ + x̄₀ + x̄₁) * Δt) / 4;
                *p = (C){ x₀, x₁, x_m, −σ }; walk(t₀, t₁, c);
                *p = (C){ x_m, −σ, x₁, x̄₁ }; walk(t₀, t₁, 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(t₀, t₀ + s, c);

        for (i = 0; i < n; ++i) {
            newc[i] = (C){ c[i].x₀ + c[i].x̄₀ * s, c[i].x₁, c[i].x̄₁ * s, c[i].x̄₁ };}

        walk(t₀ + s, t₁, newc);
    }
}
Cache Analysis
Theorem

We will prove that the walk algorithm incurs $O(\text{Vol}(T)/Z^{1/n})$ caches misses under certain assumptions

- the kernel operates in place meaning $(t, x^{(0)}, x^{(1)}, \ldots, x^{(n-1)})$ is stored in the same memory locations as $(t - k, x^{(0)}, x^{(1)}, \ldots, x^{(n-1)})$.
- the cache is ideal (optimal replacement policy) and fully associative
- the trapezoid is “sufficiently large”
Lemma 1

Lemma 1 Let $T$ be the $n$-dimensional trapezoid $T(t_0, t_1, x_0^{(i)}, \dot{x}_0^{(i)}, x_1^{(i)}, \dot{x}_1^{(i)})$, where $0 \leq i < n$. Let $T$ 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 $O((1 + n)\text{Vol}(T)/m)$ 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:

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

where $\vartheta_i = \dot{x}_1^{(i)} - \dot{x}_0^{(i)}$. Define the auxiliary function
Lemma 1

- Define the auxiliary function which represents the volume of the spacetime region with an additional +/- s. The surface area is then $V(1) - V(0)$

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

- This value is upper bounded by the integral

$$V(s) = \int_{-(\Delta t/2)-s}^{(\Delta t/2)+s} \prod_{0 \leq i < n} (w_i + 2s + \vartheta_i t) \, dt$$
Lemma 1

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

\[
f(s, r) = \prod_{0 \leq i < n} (w_i + (2 + \vartheta_i r)s + \vartheta_i rm)
\]

The derivative \( V'(0) \) is

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

Observe that

\[
m \cdot \left. \frac{df(s, r)}{ds} \right|_{s=0} = f(0, r) \cdot \sum_{0 \leq j < n} \frac{2m + \vartheta_j rm}{w_j + \vartheta_j rm} \leq nf(0, r)
\]

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

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

\[
g'(0) \cdot m \cdot (f(0, g(0)) + f(0, -g(0))) \leq 0 .
\]

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

Q.E.D.
Main Theorem

**Theorem 2** Let \( T \) be the well-defined \( n \)-dimensional trapezoid \( T(t_0, t_1, x_0^{(i)}, x_1^{(i)}, \ldots, x_n^{(i)}) \). Let procedure walk traverse \( T \) 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 \( O(\text{Vol}(T)/Z^{1/n}) \) cache misses.

- We recursively divide the trapezoid into smaller trapezoids until we reach a sub-trapezoid \( S \) with \( O(Z) \) surface points. Due to the in-place memory assumption, we can compute the points in \( S \) with \( O(\partial\text{Vol}(S)) \) misses (replaces happen in cache).
- For \( S \), we know \( \Delta t = \Theta(w_i) \) since otherwise, the corresponding dimension would be cut. Therefore, \( \Delta t = \Omega((\partial\text{Vol}(S))^{1/n}) = \Omega(Z^{1/n}) \).
- From Lemma 1, \( \partial\text{Vol}(S) = O(\text{Vol}(S)/\Delta t) \), from which it follows that the number of cache misses from computing \( S \) is \( O(\text{Vol}(S)/Z^{1/n}) \). Summing over all \( S \), we arrive at the result.
Conclusion

- Future Work
  - Conduct an empirical analysis with real hardware to compare practical cache miss rate
  - Consider cache complexity for multithreaded/parallel versions of walk
- Strengths
  - Algorithm is broadly applicable as its the first of its time to generalize to arbitrary stencils and dimensions
  - Bound reaches theoretical limit
- Weaknesses
  - Needs more empirical testing along with real hardware
  - Makes significant assumptions on the structure of the stencil and types of cache