Cache Oblivious Stencil Computations

Matteo Frigo and Volker Strumpen

Presented by Luca Musk

The Authors

Matteo Frigo

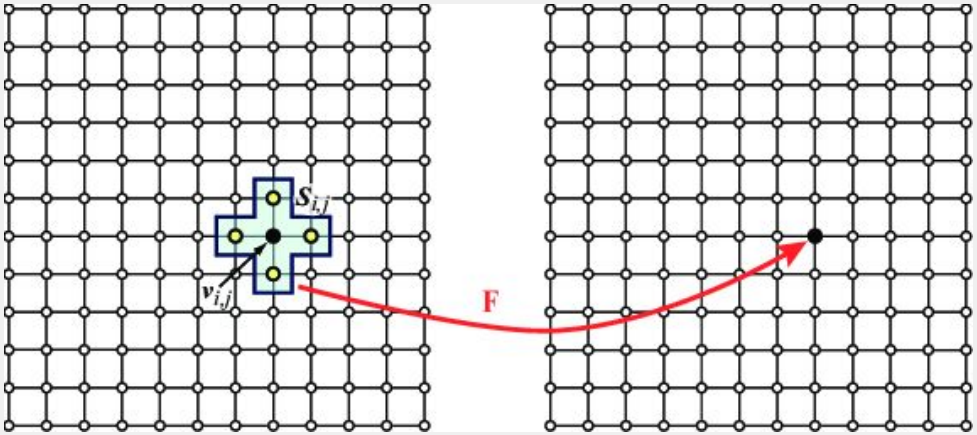
MIT PhD under Charles Leiserson, researching cache-oblivious algorithms and helped to develop Cilk. Now a software engineer at Oracle.

Volker Strumpen

Currently a professor at Rhein-Waal University of Applied Sciences in Germany. Was previously research staff at IBM where he worked with Frigo

Background

Stencils



Prototype implementation of array-processor extensible over multiple FPGAs for scalable stencil computation - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/D-example-of-stencil-computation_fig1_220244592 [accessed 13 Nov 2025]

Stencil

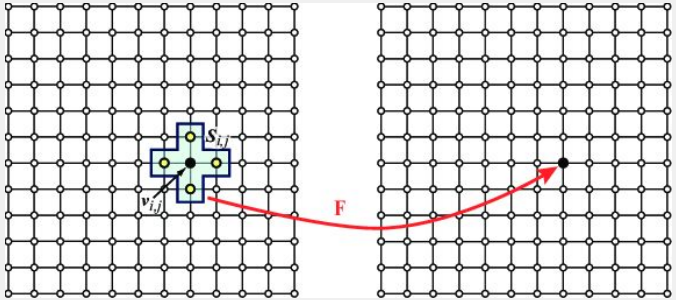
- Assume an n-dimensional set of points
- A stencil computes the value for a given point at time t using elements from its neighbors in prior time points

Stencil Computations

- Computing the result of the stencil over many iterations
- Typically, we want just the final result and not the result at each time step
- This lets us only need k + 1 buffers, where k is the most time steps back our stencil must go

Background

Cache Misses in Stencils



Prototype implementation of array-processor extensible over multiple FPGAs for scalable stencil computation - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/D-example-of-stencil-computation_fig1_220244592 [accessed 13 Nov 2025]

<u>Naive Algorithm</u>

For each point (i,j) in N_t:

Fetch points (i-1, j), (i+1, j), (i, j-1), (i, j+1), and (i,j) from N_(t-1)

Set (i,j) = Avg((i-1, j), (i+1, j), (i, j-1), (i, j+1), (i,j))

<u>Cache Misses</u>

Assume N_i is in row major order and our cache is of size Z with a line of length B.

For each point fetch, we will either have a hit or have to fill a line of size B. Assuming N_i does not fit in Z, we'll need to start fresh for each iteration. This means we will take N/B misses per iteration, and so our misses are O(NT/B)

Background

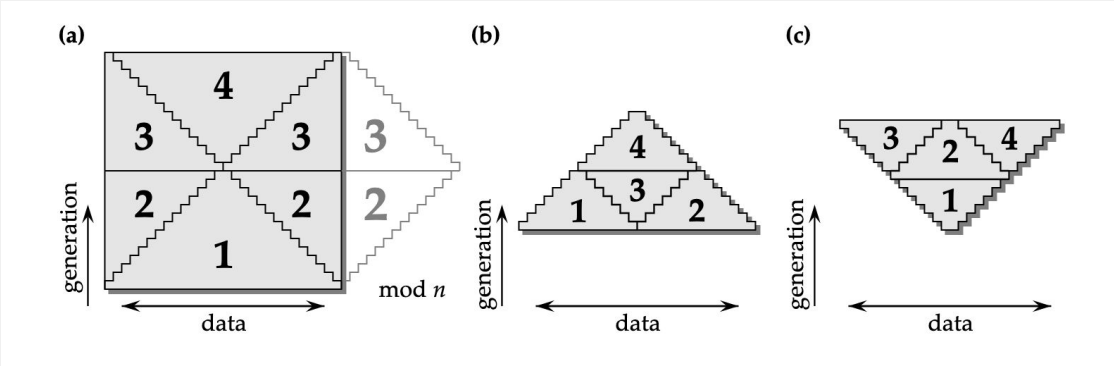
Prior Work

Bilardi and Preparata

- Focused on simulating parallel machines
- Some similar properties, but does not go beyond dimension 2 for stencil applicability

Prokop

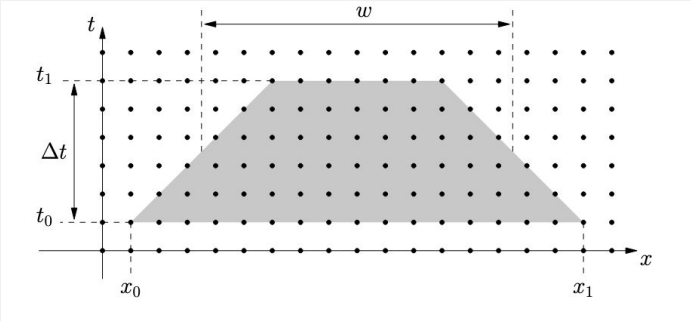
- Cache oblivious, optimal stencil for dimension 1
- Heavily restrained based on dimension and requires operating on a square spacetime region



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

The Algorithm

1-Dimensional Stenciling



Core Idea: Break the spacetime region into trapezoids

- Example – a 3x1 1-Dimensional stencil
- The trapezoid represents the data dependencies over time

Given a trapezoid, we have three cases:

1. Base case: t1 - t0 == 1
    a. Here, we can simply perform the stencil kernel on the elements in row t0 as there's no data dependencies and output into t1

The Algorithm

1-D Stenciling: Space Cuts

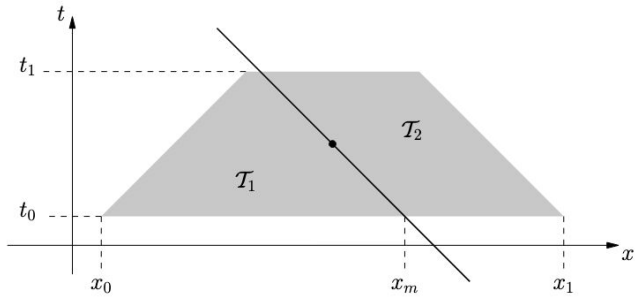


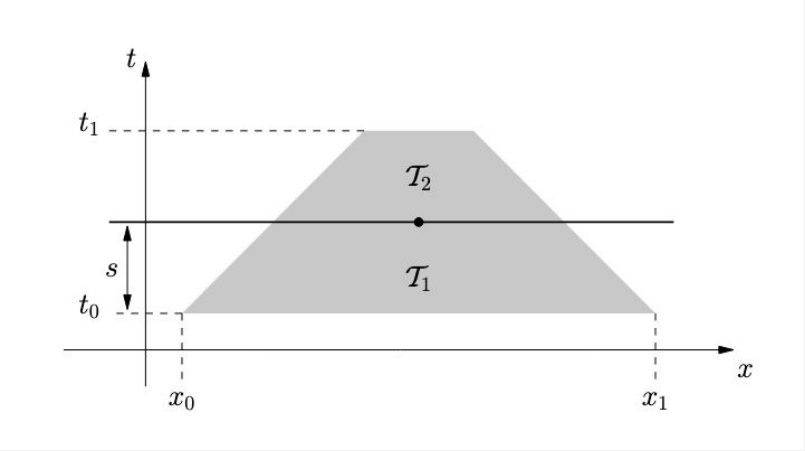
Figure 3: Illustration of a *space cut*. When the space dimension is "large enough" (see text), procedure `walk1` cuts the trapezoid along the line of slope -1 through its center.

2. Space case: x1 - x0 >= 2 * (t1 - t0)
    a. Cut the trapezoid through its center parallel to the data dependency
    b. Recurse on T1, then T2
    c. The order is necessary since points in T2 depend on results from T1, but no point in T1 needs a result from T2

Small Caveat: we actually need x1 - x0 >= 2 * o * (t1 - t0), where o is the upper bound of the magnitude of the data dependency

The Algorithm

1-D Stenciling: Time Cuts



3. Time case: x1 - x0 < 2 * (t1 - t0)
    a. Cut the trapezoid through its center, parallel to the x axis
    b. Recurse on T1 first and then T2, also due to a data dependence between the two

Small Caveat: we actually need x1 - x0 < 2 * o * (t1 - t0), where o is the upper bound of the magnitude of the data dependency

The Algorithm

1-D Stenciling: Example

$t\backslash x$	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

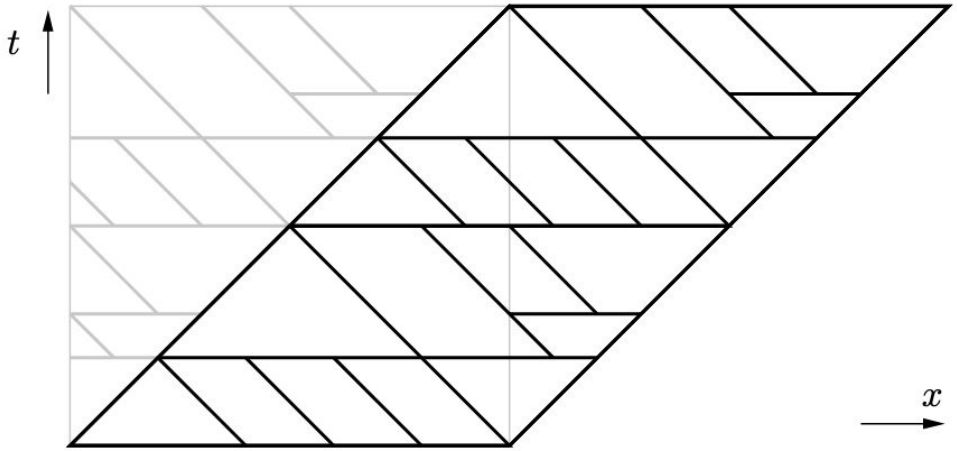
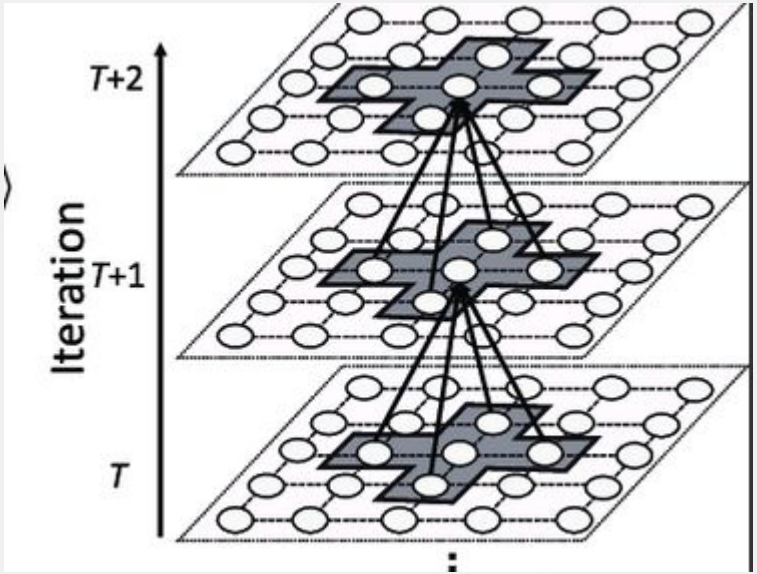


Figure 5: Cache-oblivious traversal of 1-dimensional spacetime for $N = T = 10$.

The Algorithm

Multi-D Stenciling



Multi-FPGA Accelerator Architecture for Stencil Computation Exploiting Spacial and Temporal Scalability - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Stencil-computation-using-4-point-2-D-stencil_fig1_332809964 [accessed 13 Nov 2025]

Very simple generalization
N-dimensional trapezoids

- Each slice in the (t, x_i) plane forms a 1-d trapezoid

The same formula as before applies

- Space cut along (t, x_i) if the ith dimension's trapezoid is viable
- Otherwise if no space cuts are possible, time cut

Analysis

Cache Complexity: The Base Case

Consider the base case where the trapezoid fits in cache

- Here, we should expect to incur a miss along the boundaries of the rectangle

How do we bound the surface of our trapezoids?

$$Vol(\mathcal{T}) = \sum_{-\Delta t/2}^{\Delta t/2} \prod_{0}^{n} (w_i + v_i t)$$

- v_i(t) = the difference in the data dependency slopes in the i dimension

$$V(s) = \sum_{-\Delta t/2-s}^{\Delta t/2+s} \prod_{0}^{n} (w_i + 2s + v_i t)$$

- V(0) = Vol(T), V(1) - V(0) >= Sur(T)
- We can therefore approximate Sur(T) with V'(0)

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 $0 \le i < n$. 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 $O((1+n)\mathrm{Vol}(\mathcal{T})/m)$ points on the surface of the trapezoid.

Analysis

Cache Complexity: The Base Case

We can also approximate V(s)

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

After substituting t=(m+s)r, we get

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

To introduce some shorthand, let's let

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

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

And so

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

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 $0 \le i < n$. 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 $O((1+n)\mathrm{Vol}(\mathcal{T})/m)$ points on the surface of the trapezoid.

Analysis

Cache Complexity: The Base Case

Returning to Sur(T), recognize that since Sur(T) < V(1) - V(0), we can approximate it with V'(0), which is

$$V'(0) = g'(0) \cdot m \cdot (f(0, g(0)) + f(0, -g(0))) + \int_{-g(0)}^{g(0)} (f(0, r) + m \cdot \frac{df(s, r)}{ds}|_{s=0}) dt$$

Notice that:

$$m \cdot \frac{df(s, r)}{ds}|_{s=0} = f(0, r) \cdot \sum_{0 \leq j < n} \frac{2m + v_j rm}{w_j + v_j rm}$$

Since m is half the minimum of w_j, this is upper bounded by n * f(0,r)

Analysis

Cache Complexity: The Base Case

Finally, note that

$$g'(s) = (m - \Delta t/2)/(m+s)^2$$

Since m < Dt/2, this must be negative. Thus

$$g'(0) \cdot m \cdot (f(0, g(0)) + f(0, -g(0)))$$

Must be negative as well! Thus:

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 $O((1+n)\mathrm{Vol}(\mathcal{T})/m)$ points on the surface of the trapezoid.

$$V'(0) \leq \int_{-\Delta t/(2m)}^{\Delta t/(2m)} ((1+n) f(0, r)) dr = (1+n) V(0)/m$$

MATH IS ALMOST OVER!!!

https://static.vecteezy.com/system/resources/previews/054/709/020/non_2x/festive-emoji-character-celebrating-with-colorful-confetti-wearing-a-party-hat-and-expressing-joy-and-excitement-for-special-occasions-free-vector.jpg

Analysis

Cache Complexity: Bringing it Together

The algorithm cuts trapezoids till it finally finds one that fits in the cache of size Z

- We incur misses linear to the amount of points on the surface, O(Sur(T))

Since we cut space if our width is ever too big, we have that Dt is Theta(wi), and so is asymptotically lower bounded by Sur(T)^(1/n)

Since Dt is O(wi), m is Theta(Dt), and so Sur(T) is O(Vol(T)/Dt). Thus, our misses per subtrapezoid are O(Vol(T)/Sur(T)^(1/n))=O(Vol(T)/Z^(1/n)).

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.

Conclusion

Paper Review

Strengths

- Simple algorithm
- Well explained, with a simpler version and a more general case
- Great theoretical bounds
- Rigorous analysis

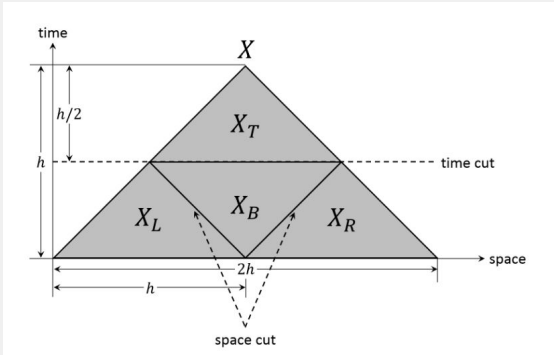
Weaknesses

- No implementation
- Quite a few assumptions (i.e. ideal cache) which are difficult to weigh without a practical implementation

Next Steps

- Implementation
- Consider parallelization

Yuan Tang, Ronghui You, Haibin Kan, Jesmin Jahan Tithi, Pramod Ganapathi, and Rezaul A. Chowdhury. 2014. Improving Parallelism of Recursive Stencil Computations without Sacrificing Cache Performance. In Proceedings of the Second Workshop on Optimizing Stencil Computations (WOSC '14). Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/2686745.2686752



Conclusion

Discussion Questions

1. Do you see non-ideal cache scenarios (i.e. LRU replacement) to be a potential concern for implementations of this algorithm?
2. Does this algorithm seem reasonable to GPU settings where caching may be done explicitly but where much of stenciling workloads appear?
3. Can you think of any ways to parallelize this workload?

Thank you!