This space is reserved for the Procedia header, do not use it

Optimal Temporal Blocking for Stencil Computation

Takayuki Muranushi[1] and Junichiro Makino[2]

[1] RIKEN AICS, 7-1-26, Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo, 650-0047, Japan
takayuki.muranushi@riken.jp
[2] jmakino@riken.jp

(This is an online, extended version of the paper submitted to ICCS 2015.)

Abstract
Temporal blocking is a class of algorithms which reduces the required memory bandwidth (B/F ratio) of a given stencil computation, by "blocking" multiple time steps. In this paper, we prove that a lower limit exists for the reduction of the B/F attainable by temporal blocking, under certain conditions. We introduce the PiTCH tiling, an example of temporal blocking method that achieves the optimal B/F ratio. We estimate the performance of PiTCH tiling for various stencil applications on several modern CPUs. We show that PiTCH tiling achieves $1.5 \sim 2$ times better B/F reduction in three-dimensional applications, compared to other temporal blocking schemes. We also show that PiTCH tiling can remove the bandwidth bottleneck from most of the stencil applications considered.

Keywords: Parallel computation,Stencil computation,Optimization

1 Introduction

Many important applications in computational science are based on stencil computation. Its optimization has been an important topic, particularly on machines with deep memory hierarchy, since one of the characteristics of stencil computation is that it requires relatively high memory bandwidth. We measure the requirement for memory bandwidth by B/F (bytes-per-flop ratio, or simply B-over-F), the amount of the data transferred between the main memory and CPU per one floating point operation.

The dependence of the B/F ratios of numerical algorithms on the cache size N can be classified into two categories; *constant* B/F algorithms whose B/F ratios are independent of N, and *cache-dependent* B/F algorithms whose B/F depends on N. Stencil computation has been considered to be a constant B/F algorithm. In traditional implementations, the entire solution at one time step is calculated before moving on to the next time step. For this kind of schemes, we have

$$B/F \quad \geq \quad \beta_{\mathrm{trad}} = 2H_e/C_e \tag{1}$$

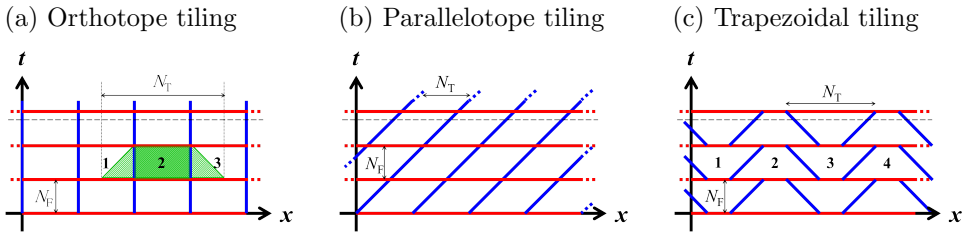

Figure 1: The state-of-the-art temporal blocking methods.

where H_e is the size of the stencil cell and C_e is the computation per cell update. Examples of cache-dependent B/F algorithms include direct-N-body simulations and matrix-matrix multiplications, whose B/F ratios as functions of N are $\mathcal{O}(N^{-1})$ and $\mathcal{O}(N^{-0.5})$, respectively.

The actual B/F ratio possible on high-performance computing (HPC) platforms has been decreasing. Traditional vector processors had $B/F \simeq 4$, while the B/F of the K computer is approximately 0.5. The exascale systems planned in Japan, US and China will have B/F of $0.1 \sim 0.2$. As a result, the efficiency of stencil applications on modern HPC platforms has been decreasing. Thus, if we can reduce the B/F requirement of stencil computation beyond β_{trad}, that would have a very large impact.

In general, B/F requirement can be decreased by exploiting the data locality [11, 5, 10, 8, 7]. *Temporal blocking* is such an optimization technique for stencil computation which can reduce the B/F ratio beyond β_{trad}. The idea of temporal blocking is to decompose the entire computation domain in $(d+1)$-dimensional spacetime, into set of *regions* $\{R\}$ where each $R \subset \mathbb{Z}^{d+1}$ contains grid points from multiple time steps. The shape of R is determined so that the calculation for R fits into the cache of the CPU, and thus the required B/F is reduced. Figure 1 illustrates some of the known blocking algorithms for $d = 1$. (for review see [12].)

The orthotope tiling (Figure 1a) is possibly the simplest temporal blocking scheme. It decomposes the set of space-temporal points $(t, \vec{x}) \in \mathbb{Z}^{d+1}$ into regions $R_{\mathrm{ortho}}(\tau, \vec{\chi})$ labeled by $(\tau, \vec{\chi}) \in \mathbb{Z}^{d+1}$, as follows:

$$(t, \vec{x}) \in R_{\mathrm{ortho}}(\tau, \vec{\chi}) \Leftrightarrow \left\{ \begin{array}{ccccc} \tau N_F & \leq & t & < & (\tau+1)N_F \\ \chi_i N_T & \leq & x_i & < & (\chi_i+1)N_T \end{array} \right. \tag{2}$$

It is called the "overlapped tiling" in [12], because in the orthotope tiling, computation of one region includes calculation of the solutions not only at the points in the region but also at some points in its neighboring regions. In order to obtain the solution for the top segment of region **2**, we need to start computation from the bottom of the green trapezoid, which contains grid points in regions **1** and **3**. On the other hand, by allowing this redundant calculation, all regions in one time slice can be computed in parallel, without the need of inter-region communication.

The parallelotope tiling (Figure 1b):

$$(t, \vec{x}) \in R_{\mathrm{para}}(\tau, \vec{\chi}) \Leftrightarrow \left\{ \begin{array}{ccccc} \tau N_F & \leq & t & < & (\tau+1)N_F \\ \chi_i N_T & \leq & x_i - N_s t & < & (\chi_i+1)N_T \end{array} \right. \tag{3}$$

does not require redundant computations. Parallelotope tilings have also been studied in the context of cache-oblivious computations and have shown excellent performance on multithreaded computations [2, 3, 9]. However, in the parallelotope tiling, regions R_{para} located on a temporal plane τ have global dependency. In the case of Figure 1b, the computation must go

from right to left. This global dependency makes parallelotope tiling unsuitable for large-scale parallel computation.

The trapezoidal tiling (Figure 1c) and the diamond tiling [6] are examples of temporal blocking schemes that do not suffer from this global dependency. In Figure 1c, we can first calculate regions $\mathbf{2}, \mathbf{4}, \cdots$ in parallel, and then regions $\mathbf{1}, \mathbf{3}, \cdots$ in parallel. However, these tilings can be applied to only one dimension in their original forms. Therefore, it is difficult to apply them on large-scale parallel computers where we would like to decompose the computational domain in all spatial dimensions.

We identify the four conditions which a temporal blocking scheme should satisfy to be usable on large-scale parallel computers.

1. *Translational Symmetry.* The entire computational domain is filled with a *unit lattice*, which in turn contains one or more $(d+1)$-dimensional polytope *regions*. The spacetime is filled with only one kind of unit lattice, without gap or overlap, first by creating its spatially-translated copies and then by temporally translating them.

2. *Spacelikeness.* For any two adjacent regions A and B, either A depends on B or B depends on A. This means that the slope of the facets of a region R must be equal to or shallower than $1/N_S$, where N_S is the number of grid points in the stencil in one direction.

3. *Independence of the unit lattices.* Any two regions are causally independent, if they match by spatial translation that maps a unit lattice to another.

4. *Finiteness.* The scheme should be able to decompose all of the spatial dimensions, so that the size of the unit lattice is finite in all dimensions of spacetime.

Condition 1 allows us to use a simple, SPMD-like programming model. Condition 2 lets us evaluate each region independently, without overlapped computations. Condition 2 and 3 allow us to evaluate unit lattices located on a temporal plane τ in any order, or in parallel. Condition 4 is needed to achieve scaling up to thousands and millions of nodes.

There already exists several state-of-the-art temporal blocking methods that satisfy all of the four conditions. One way to achieve such a tiling is to apply the trapezoidal tiling recursively to each spatial dimension [4], which is called "nested splitting." In this method (c.f. Figure 6a of [4]), the unit lattice consists of 2^d polytope regions such as hyperfrustums, parallelotopes, and other shapes. Another way to satisfy the four condition is to fill the spacetime with tilted hypercubes [1]. These two approaches are generalizations of trapezoidal tiling and diamond tiling to higher dimensions, respectively.

In this paper, we show that the theoretical limit for the B/F reduction exists. In §2 we show, under certain assumptions, that ${2dN_S/N_{\mathrm{cache}}}^d$ is the lower limit for the B/F reduction that can be achieved by temporal blocking. Here N_S is the halo size of the stencil, and N_{cache} is the number of cells that fit into the cache. In §3 we give a temporal blocking method that achieves this lower limit. In §4 we describe its execution model, and then estimate the performance of the proposed method when applied to several typical stencils on present-day CPUs. Finally in §5, we discuss the implications of the limit, and the possibilities of future software and hardware designs to overcome the limitations of the memory bandwidth.

2 Conditions for the Optimal Temporal Blocking

In this paper, we limit the scope of our analysis to variants of temporal blocking where computation within each region proceed in time direction. We have found that the same lower limit

for the B/F reduction exists for variants of the nested splitting where the computation proceed in space direction, and we will discuss it in more detail in the Appendix.

Tile shapes covered by our analysis include the orthotope, the parallelotope, the trapezoid, and the tilted hypercube. We have found that their asymptotic B/F behavior can be expressed in the following common form:

$$B/F = r\beta_{\mathrm{trad}} \tag{4}$$

$$\text{where} \quad r = \frac{1}{N_F} + \frac{2dN_S}{N_T} + \mathcal{O}\left(\left(\frac{N_S}{N_T}\right)^2\right) \tag{5}$$

Because typical values for three-dimensional applications are $N_S = 3 \sim 4$, and $N_T \equiv {N_{\mathrm{cache}}}^{1/d} \simeq 100$, the second term in Equation (5), $2dN_S/N_T \simeq 0.2$ dominates over the first term if $N_F \geq 5$. As a result, the reduction of B/F is limited by this term even for very large values of N_F.

Of the two terms $1/N_F$ and $2dN_S/N_T$ in Equation (5), the former corresponds to the data transfer in time direction through initial and final boundaries, while the latter corresponds to that in space direction through the stencil halo. In Figure 1, they are represented in red and blue line segments, respectively. Using these words our finding can be rephrased as follows. In the large-N_F limit, the halo term dominates the main memory bandwidth. The design of optimal temporal blocking, therefore, should be guided by the optimization of the halo term.

To analyze the data transfer within stencil algorithms, we introduce the notion of *partition* of a region R into *batches*: $\{Q_i \subset \mathbb{Z}^{d+1} \mid Q_i \subset R, i \in [1, \cdots, N_Q]\}$. They are *cache-fitting batches* if and only if:

- $\{Q_i\}$ is a set of collectively exhaustive and mutually exclusive subsets of R.
- $\forall i.\ \#Q_i \leq N_{\mathrm{cache}}$, where $\#A$ denotes the number of elements of the set A.
- all solutions at grid points $\{Q_i\}$ can be computed depending on solutions at points $\{Q_1, \cdots, Q_{i-1}\}$ and points outside R.

Let $h : \mathcal{P}(\mathbb{Z}^{d+1}) \to \mathcal{P}(\mathbb{Z}^{d+1})$ be the function that maps a set of grid points to its halo (the set of points it directly depends on):

$$h(Q) \equiv \{(t', \vec{x'}) \mid (t, \vec{x}) \in Q, t' = t - 1, |x'_i - x_i| \leq N_S\}. \tag{6}$$

Given the cache-filling batches, the stencil algorithm proceeds iteratively, by bringing the values of the solutions at positions $h(Q_i)$ on-cache and by computing the solutions for positions Q_i. This, however, does not mean that all of the grid points in $h(Q_i)$ are transferred from the main memory. The transfer can be omitted for grid points in $h(Q_i) \cap Q_{i-1}$, since they are already on cache from the $(i-1)$-th iteration. Thus, the set of grid points on cache, and the set of points loaded at iteration i, are

$$Q_{\mathrm{cache},i} = Q_{i-1}, \tag{7}$$

$$Q_{\mathrm{ld},i} = h(Q_i) \backslash Q_{\mathrm{cache},i} = h(Q_i) \backslash Q_{i-1}, \tag{8}$$

respectively, and the amount of load at iteration i is

$$H_{\mathrm{ld},i} = \#\left(h(Q_i) \backslash Q_{i-1}\right) \cdot H_e. \tag{9}$$

The total amount of memory transaction, H, is the sum of the initial load, the final store, the halo-loading transactions, and the halo-storing transactions which is the same amount as the loading transactions.

$$H = H_e(\#Q_1 + \#Q_{N_Q}) + 2\sum_{i=1}^{N_Q} H_{\mathrm{ld},i} \tag{10}$$

Here,

$$\begin{aligned}\sum_{i=1}^{N_Q} H_{\mathrm{ld},i} &= 2H_e \sum_{i=1}^{N_Q} \#\left(h(Q_i)\backslash Q_{i-1}\right) \\ &\geq 2H_e \left(\sum_{i=1}^{N_Q} \#h(Q_i) - \sum_{i=1}^{N_Q} \#Q_i \right). \end{aligned} \tag{11}$$

Compared to the set Q_i, the set $h(Q_i)$ has extra N_S-cells-thick layers of grid points in each of the d spatial dimensions.

Now, for any d-dimensional polytope Q, let $V \equiv \mathrm{Vol}_d(Q)$ be its d-dimensional volume, and $S_i \equiv \mathrm{Vol}_{d-1}(P_i(Q))$ be the $(d-1)$-dimensional volume of the projection P_i of Q onto $(d-1)$-dimensional hyperplane that is perpendicular to i-axis. Then the following inequality holds (the sketch of proof is given in Appendix **??**):

$$\sum_{i=1}^{d} S_i \geq d \cdot V^{\frac{d-1}{d}}. \tag{12}$$

From this inequality, and from the fact that $\#Q_i \leq N_{\mathrm{cache}}$ we obtain

$$\frac{\#h(Q_i) - \#Q_i}{\#Q_i} \geq \frac{2dN_S}{N_T} \tag{13}$$

The equality holds if and only if Q_i is a d-dimensional hypercube of size N_T where $N_T = {N_{\mathrm{cache}}}^{1/d}$. By substituting inequality (13) into inequality (11), and by defining $1/N_F \equiv (\#Q_1 + \#Q_{N_Q})/(2\sum_{i=1}^{N_Q} \#Q_i)$, we have

$$\left(\frac{B}{F}\right) = \frac{H}{C_e \sum_{i=1}^{N_Q} \#Q_i} \geq \left(\frac{1}{N_F} + \frac{2dN_S}{N_T}\right) \beta_{\mathrm{trad}}, \tag{14}$$

which gives the lower limit of B/F attainable by temporal blocking.

Now we know that the optimal shape of Q_i is a d-dimensional hypercube of size N_T and one-cell thick in time direction, the next task is to determine the shape of $R = \bigcup_{i=1}^{N_Q} Q_i$. In order to comply the condition 2 (spacelikeness), the only solutions are to let the d-dimensional hypercube "sweep" the spacetime at slope $(\pm 1/N_S, \pm 1/N_S, \cdots, \pm 1/N_S)$. This gives parallelotope regions. Since we have just seen in §1 that parallelotope tiling suffer from global dependency, one might think that the optimal temporal blocking is unattainable.

3 Our Method

We have found that the parallelotope tiling, Equation (3), can be slightly modified so that it satisfies all of the four conditions. The modified decomposition is as follows:

$$(t, \vec{x}) \in R'(\tau, \vec{\chi}) \Leftrightarrow \left\{ \begin{array}{ccccc} \tau N_F & \leq & t + \frac{1}{N_G}\sum_{i=1}^{d} x_i & < & (\tau+1)N_F \\ \chi_i N_T & \leq & x_i - N_s t & < & (\chi_i + 1)N_T \end{array} \right. \tag{15}$$

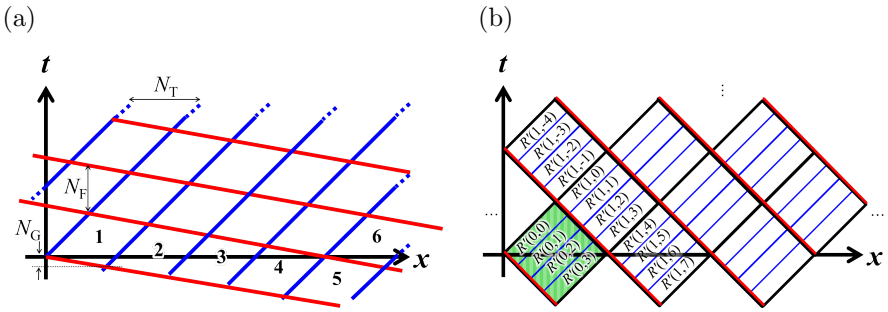

Figure 2: The PiTCH tiling.

Figure 2a shows the decomposition in $d = 1$. We call this PiTCH (Parallelotopes in Tilted Cube Hierarchy) tiling. PiTCH tiling is similar to parallelotope tiling shown in Figure 1b, with the exception that each parallelotope region is deformed and shifted in time direction by multiples of N_G. Because of this modification, PiTCH tiling satisfies independence of the unit lattices, which parallelotope tiling did not. This property is demonstrated in Figure 2a, by the fact that region **1** matches region **6** by spatial translation, not region **5**.

In Figure 2a, regions $\mathbf{1} \cdots \mathbf{4}$ form a unit lattice. This corresponds to regions $R'(0,0), \cdots, R'(0,3)$ in Figure 2b. Generally speaking, a unit lattice of PiTCH tiling consists of $(N_F/N_G)^d$ regions. The unit lattice fills the entire computational domain first by its spatially-translated copies, then by the temporally-translated copies of the space-filling pattern, in such ways that satisfy the four conditions (Figure 2b).

The parameter N_G of PiTCH tiling is constrained by the spacelikeness condition:

$$N_G \leq \frac{N_T}{dN_S}. \tag{16}$$

The inequality (16) is used to model the performance of PiTCH tiling.

4 Execution and Performance Models

Figure 3 shows an execution model of the stencil computation with PiTCH tiling on a parallel computer. The computer consists of multiple computing nodes, connected with inter-node communication network. Each node is equipped with N_{CPU} CPUs, and a main memory of size H_{mem} shared by all CPUs. Each CPU is equipped with its own cache of size H_{cache}.

The computation is distributed over this machine, so that each node is in charge of computing one or several unit lattice(s). The regions within that unit lattice(s) are computed by the CPUs, in the task-parallel manner.

Although the regions within a single unit lattice have dependencies, they can be evaluated in parallel, as shown in Figure 3. In the figure, `NODE 0` is in charge of computing a unit lattice $\{R'(0,0), \cdots, R'(0,3)\}$, so ① `CPU0` is computing the region $R'(0,3)$ ② using its own cache. ③ It writes the values of the leftmost cells of $R'(0,3)$ into a buffer on the main memory ④ from which `CPU1` reads. With this data `CPU1` can ⑤ compute the region $R'(0,2)$. `CPU1` ⑥ uses its own cache to compute the region $R'(0,2)$ and ⑦ writes its leftmost cells to another buffer. ⑧ Initial conditions for regions $\{R'(0,0), \cdots, R'(0,3)\}$ are stored on the main memory of `NODE 0`.

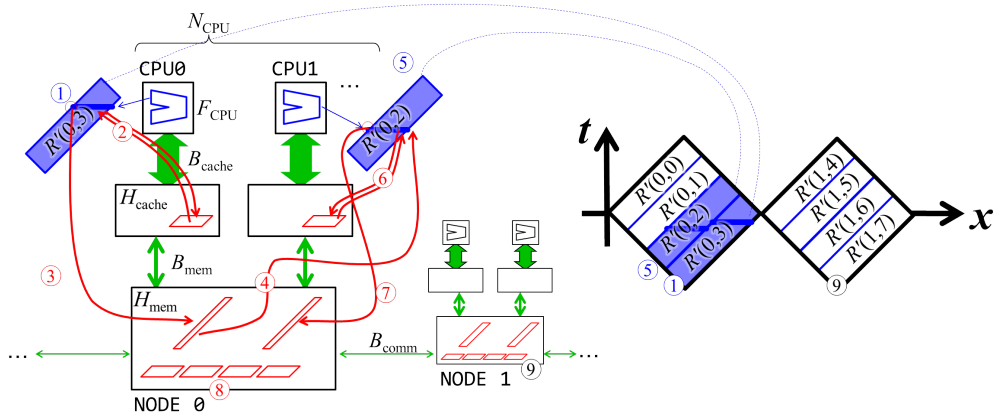

Figure 3: The execution model of PiTCH tiling.

⑨ `NODE 1` is computing the adjacent unit cell $\{R'(0,4), \cdots, R'(0,7)\}$ in the same manner, and so on.

It is true that at the very beginning of the computation of a unit lattice there is only one region ready. However, because $N_G \ll N_F$, the time the CPUs remain starved of tasks is only a small portion of the total computation time. Furthermore, we can eliminate the stall by overlapping the computation of one unit lattice with that of the next one.

Under this execution model, the main memory of a node must contain at least all the initial conditions for every regions of the unit lattice assigned to the node. Therefore, the maximal number of regions a unit lattice can contain is ${N_{TO}}^d$, where

$$N_{TO} \leq \left(\frac{H_{\mathrm{mem}}}{H_e {N_T}^d} \right)^{1/d}. \tag{17}$$

As shown in Figure 2, $N_F \leq N_G N_{TO}$. Thus, N_F for PiTCH tiling is upper-limited by the main memory size. Since the reduction ratio r is a decreasing function of N_F, we can use inequality (17) to remove N_F from inequality (14):

$$r \geq \frac{dN_S}{N_T N_{TO}} + \frac{2dN_S}{N_T} \geq \left(2 + \left(\frac{N_{\mathrm{cache}}}{N_{\mathrm{mem}}} \right)^{1/d} \right) d \cdot \frac{N_S}{N_T} \tag{18}$$

In contrast, most of the other temporal blocking schemes have their N_F upper-limited by the cache size, and in some schemes r reaches its minimum at certain $N_F = N_{F0}$, instead of decreasing indefinitely. Given such minima, $r = cN_S/N_T$ where c is a constant. Examples are the overlapped tiling and the nested splitting, where $r = 13.0446 N_S/N_T$ at $N_{F0} = 0.1537 N_T/N_S$, and $r = 9.3246 N_S/N_T$ at $N_{F0} = 0.2599 N_T/N_S$, respectively, for $d = 3$.

Compared to these schemes, the upper limit N_{F0} for PiTCH tiling is much larger, since the main memory is 10^3 to 10^4 times larger than the cache in modern computers. From inequality (18), $r = 6.075 \sim 6.6 \cdot N_S/N_T$ and $r = 4.05 \sim 4.4 \cdot N_S/N_T$, respectively, for $d = 2, 3$. The values of N_{F0} are well over 10^4 for $d = 2$, and are $30 \sim 200$ for $d = 3$. The value of r for $d = 3$ is about 0.65 times that of the nested splitting, and 0.47 times the overlapped tiling.

The PiTCH tiling is superior to the other schemes, since it can make the value of r smaller by utilizing the main memory. Only such schemes can achieve the optimal B/F reduction $r = 2dN_S/N_T$, as the main memory size approaches infinity. PiTCH tiling makes a good

(a)	F_{CPU}	N_{CPU}	H_{cache}	B_{mem}	H_{mem}
SU	751GFlop/s	32	2MB	20GB/s	128GB
KC	256GFlop/s	1	6MB	64GB/s	16GB
X2	819GFlop/s	2	20MB	136GB/s	256GB
X1	1178GFlop/s	1	40MB	136GB/s	128GB
HA	410GFlop/s	1	128MB	51GB/s	16GB

(b)	d	H_e	C_e	N_S
D2	2	4	7	1
C2	2	64	300	1
D3	3	4	10	1
S3	3	36	262	3
H3	3	20	3381	4
C3	3	80	500	1

Table 1: (a) The computer models, (b) The application models that represent — D2, D3: diffusion equation solvers, C2, C3: CIP-scheme for Navier-Stokes equations, S3: seismic wave equations solver, H3: Godunov-type Navier-Stokes equations solver.

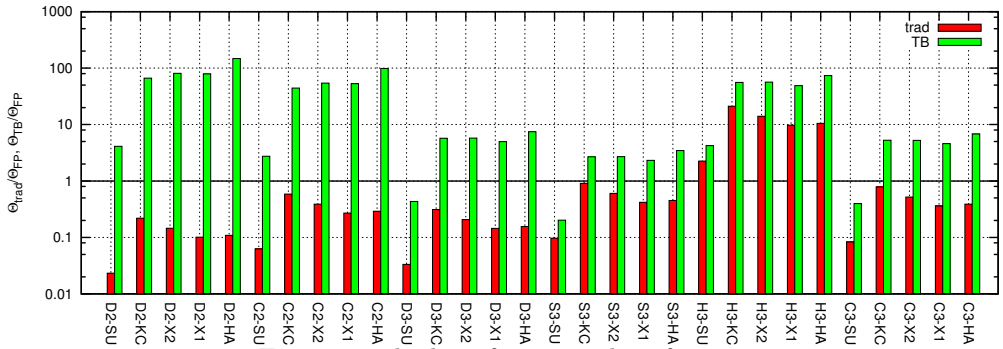

Figure 4: The list of estimated performance.

contrast with diamond tiling [6] on this point; although both methods split the spacetime into diamonds (c.f. Figure 2b), the size of the diamonds is constrained by the main memory in case of the former, and by the cache size in case of the latter.

We estimated the performance of two stencil solvers, for the model computers and applications listed in Tables 1. One solver is traditional (uses no temporal blocking) and the other is equipped with PiTCH tiling. The estimated performance are shown in Figure 4. The y-axis is either $\Theta_{\mathrm{trad}}/\Theta_{\mathrm{FP}}$ or $\Theta_{\mathrm{TB}}/\Theta_{\mathrm{FP}}$. The y-value lesser than 1 means that the bottleneck is the main-memory bandwidth, while y-value greater than 1 means that the floating-point operation is the bottleneck.

Figure 4 shows that PiTCH tiling can reduce the B/F by $1/2 \sim 1/17$, for three-dimensional applications. It also shows that PiTCH tiling can overcome the main-memory bottleneck for most of the stencil applications.

5 Conclusion and Discussion

Stencil computations have been regarded as constant B/F problems. However, we can transform its B/F requirement to be cache-dependent as $\mathcal{O}({N_{\mathrm{cache}}}^{-1/d})$. We have shown the existence of the upper limit for the B/F reduction ratio that can be attained by temporal blocking. We also have introduced the PiTCH tiling, an example of tiling algorithm that achieves this upper limit. Future work will cover concrete implementation and benchmark results on real machines.

The lower limit we found, $B/F \leq 2dN_S/N_T \cdot \beta_{\mathrm{trad}}$ applies to any temporal blocking algorithms that follows the pattern in §4. There exist temporal blocking schemes which break one

or more of the assumptions we have made, but all known (or newly found) schemes so far also obey the same lower limit (see Appendix B).

The lower limit (14) has strong implications on the hardware designs. The only hardware change we can make to reduce the B/F is to increase the cache size, as long as we stay under the model of Figure 3. Note that the cache refers to what resource we can use to calculate the single region R. The "cache" can be extended over multiple cores, or even over multiple nodes, if lowlatency communication between cores or nodes is available. This is one of the hardware designs that allows us to overcome the limit.

Acknowledgments

The authors thank Drs. Nobu Kishimoto and Masataka Chida of Hakubi Center for Advanced Research, Kyoto University, for the original proof of the inequality (34). The authors thank Kengo Nitadori and Naoto Nakasato for discussions that improved this paper. Part of the research covered in this paper research was funded by MEXT's program for the Development and Improvement for the Next Generation Ultra High-Speed Computer System, under its Subsidies for Operating the Specific Advanced Large Research Facilities. This research used computational resources of the K computer provided by the RIKEN Advanced Institute for Computational Science(AICS).

References

[1] Vinayaka Bandishti, Irshad Pananilath, and Uday Bondhugula. Tiling stencil computations to maximize parallelism. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 40. IEEE Computer Society Press, 2012.

[2] Matteo Frigo and Volker Strumpen. Cache oblivious stencil computations. In *Proceedings of the 19th annual international conference on Supercomputing*, pages 361–366. ACM, 2005.

[3] Matteo Frigo and Volker Strumpen. The memory behavior of cache oblivious stencil computations. *The Journal of Supercomputing*, 39(2):93–112, 2007.

[4] Tom Henretty, Richard Veras, Franz Franchetti, Louis-Noël Pouchet, J Ramanujam, and P Sadayappan. A stencil compiler for short-vector simd architectures. In *Proceedings of the 27th international ACM conference on International conference on supercomputing*, pages 13–24. ACM, 2013.

[5] Monica D Lam, Edward E Rothberg, and Michael E Wolf. The cache performance and optimizations of blocked algorithms. *ACM SIGOPS Operating Systems Review*, 25(Special Issue):63–74, 1991.

[6] Tareq Malas, Georg Hager, Hatem Ltaief, Holger Stengel, Gerhard Wellein, and David Keyes. Multicore-optimized wavefront diamond blocking for optimizing stencil updates. *arXiv preprint arXiv:1410.3060*, 2014.

[7] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *ACM SIGPLAN Notices*, 48(6):519–530, 2013.

[8] J Ramanujam and P Sadayappan. Tiling multidimensional iteration spaces for multicomputers. *Journal of Parallel and Distributed Computing*, 16(2):108–120, 1992.

[9] Robert Strzodka, Mohammed Shaheen, Dawid Pajak, and Hans-Peter Seidel. Cache oblivious parallelograms in iterative stencil computations. In *Proceedings of the 24th ACM International Conference on Supercomputing*, pages 49–59. ACM, 2010.

[10] Michael E Wolf and Monica S Lam. A data locality optimizing algorithm. *ACM Sigplan Notices*, 26(6):30–44, 1991.

[11] Michael Wolfe. More iteration space tiling. In *Proceedings of the 1989 ACM/IEEE conference on Supercomputing*, pages 655–664. ACM, 1989.

[12] David G Wonnacott and Michelle Mills Strout. On the scalability of loop tiling techniques. *IMPACT 2013*, pages 3–11, 2013.

A Projection Inequality

Let V be bounded, Lebesgue measurable set in $\mathbb{R}^d$. Let P_i denote the projection that removes the i-th element of a vector:

$$P_i\left((x_1, \cdots, x_d)\right) \quad \equiv \quad (x_1, \cdots, \widehat{x_i}, \cdots, x_d), \tag{19}$$

where $\widehat{x_i}$ denotes the removal of the element x_i. Let $S_i = P_i(V)$. We further assume that S_i are Lebesgue measurable sets in $\mathbb{R}^{d-1}$. Let Vol_d denote the d-dimensional volume in $\mathbb{R}^d$. Then

Theorem A.1.

$$\mathrm{Vol}_d(V) \quad \leq \quad \prod_{i=1}^{d} \mathrm{Vol}_{d-1}(S_i)^{\frac{1}{d-1}}. \tag{20}$$

In order to prove this theorem, we first prove the following lemma:

Lemma A.2. *For $d \geq 2$,*

$$\begin{aligned} & \int_{\mathbb{R}^d} \prod_{i=1}^{d} \chi_{S_i}(x_1, \cdots, \widehat{x_i}, \cdots, x_d) dx_1 \cdots dx_d \\ \leq \quad & \prod_{i=1}^{d} \left(\int_{\mathbb{R}^{d-1}} \chi_{S_i}(x_1, \cdots, \widehat{x_i}, \cdots, x_d) dx_1 \cdots \widehat{dx_i} \cdots dx_d \right)^{\frac{1}{d-1}}. \end{aligned} \tag{21}$$

Proof. By induction.

Base Case: for $d = 2$, Inequation (21) reads

$$\int_{\mathbb{R}^2} \chi_{S_1}(x_2)\chi_{S_2}(x_1) dx_1 dx_2 \quad \leq \quad \int_{\mathbb{R}} \chi_{S_1}(x_2) dx_2 \int_{\mathbb{R}} \chi_{S_2}(x_1) dx_1. \tag{22}$$

The equality always holds.

Inductive Step: To prove the lemma for some $d > 2$, we first rewrite the left hand side of Inequation (21) by separating the $i = d$ term, as follows:

$$\text{l.h.s.} \quad = \quad \int_{\mathbb{R}} \int_{\mathbb{R}^{d-1}} \left[\prod_{i=1}^{d-1} \chi_{S_i}(x_1, \cdots, \widehat{x_i}, \cdots, x_d) \right] \cdot \chi_{S_d}(x_1, \cdots, x_{d-1}) dx_1 \cdots dx_{d-1} \cdot dx_d. \tag{23}$$

Now, from Hölder's inequality, $\forall p, q.\ \frac{1}{p} + \frac{1}{q} = 1,\ p, q > 0$:

$$\int_S |f(x)g(x)|\, dx \quad \leq \quad \left(\int_S |f(x)|^p\, dx \right)^{\frac{1}{p}} \left(\int_S |f(x)|^q\, dx \right)^{\frac{1}{q}} \tag{24}$$

By applying Inequation (24) to integral over $x_1, \cdots, x_{d-1}$ in Equation 23,

$$\begin{aligned} \text{l.h.s. of (21)} \quad \leq \quad & \int_{\mathbb{R}} \left(\int_{\mathbb{R}^{d-1}} \prod_{i=1}^{d-1} \chi_{S_i}(x_1, \cdots, \widehat{x_i}, \cdots, x_d) dx_1 \cdots dx_{d-1} \right)^{\frac{1}{p}} \\ & \left(\int_{\mathbb{R}^{d-1}} \chi_{S_d}(x_1, \cdots, x_{d-1}) dx_1 \cdots dx_{d-1} \right)^{\frac{1}{q}} dx_d \end{aligned} \tag{25}$$

We have used the fact that $|\chi_{S_i}|^p = \chi_{S_i}$ since the value of χ_{S_i} is either 0 or 1 by definition. Now using the Lemma A.2 for $d-1$ case, and by setting $p = \frac{d-1}{d-2}, q = d-1$,

$$\begin{aligned} \text{l.h.s. of (21)} \quad \leq \quad & \int_{\mathbb{R}} \prod_{i=1}^{d-1} \left(\int_{\mathbb{R}^{d-2}} \chi_{S_i}(x_1, \cdots, \widehat{x_i}, \cdots, x_d) dx_1 \cdots \widehat{dx_i} \cdots dx_{d-1} \right)^{\frac{1}{d-1}} dx_d \\ & \cdot \left(\int_{\mathbb{R}^{d-1}} \chi_{S_d}(x_1, \cdots, x_{d-1}) dx_1 \cdots dx_{d-1} \right)^{\frac{1}{d-1}} \end{aligned} \tag{26}$$

Now, following inequality can be proved using Hölger's inequality:

$$\left| \int_S f_1(x) \cdot f_2(x) \cdots f_n(x) dx \right| \quad \leq \quad \left(\int_S |f_1(x)|^n \right)^{\frac{1}{n}} \cdots \left(\int_S |f_n(x)|^n \right)^{\frac{1}{n}}. \tag{27}$$

This is a generalization of Cauchy-Schwarz inequality; the $n = 2$ case corresponds to the original. By applying Inequality (27) to the integral over x_d in (26),

$$\begin{aligned} \text{l.h.s. of (21)} \quad \leq \quad & \prod_{i=1}^{d-1} \left(\int_{\mathbb{R}} \int_{\mathbb{R}^{d-2}} \chi_{S_i}(x_1, \cdots, \widehat{x_i}, \cdots, x_d) dx_1 \cdots \widehat{dx_i} \cdots dx_{d-1} \cdot dx_d \right)^{\frac{1}{d-1}} \\ & \cdot \left(\int_{\mathbb{R}^{d-1}} \chi_{S_d}(x_1, \cdots, x_{d-1}) dx_1 \cdots dx_{d-1} \right)^{\frac{1}{d-1}}. \end{aligned} \tag{28}$$

□

Now we prove Theorem A.1.

Proof of Theorem A.1. Let χ_V and χ_{S_i} be characteristic functions for V and S_i, i.e.

$$\chi_V(x) \quad = \quad \begin{cases} 1 & \cdots \quad x \in V \\ 0 & \cdots \quad x \notin V \end{cases} \qquad \text{where} \quad x = (x_1, \cdots, x_d), \tag{29}$$

$$\chi_{S_i}(x) \quad = \quad \begin{cases} 1 & \cdots \quad x \in S_i \\ 0 & \cdots \quad x \notin S_i \end{cases} \qquad \text{where} \quad x = (x_1, \cdots, \widehat{x_i}, \cdots, x_d). \tag{30}$$

Then $\forall x = (x_1, \cdots, x_d) \in \mathbb{R}^d$, χ_V and χ_{S_i} satisfy the following inequality:

$$\chi_V(x_1, \cdots, x_d) \quad \leq \quad \prod_{i=1}^{d} \chi_{S_i}(x_1, \cdots, \widehat{x_i}, \cdots, x_d), \tag{31}$$

since $(x_1, \cdots, x_d) \in V \Rightarrow \forall i.\ (x_1, \cdots, \widehat{x_i}, \cdots, x_d) \in S_i$. By integrating (31) over $\mathbb{R}^d$,

$$\mathrm{Vol}_d(V) \quad \leq \quad \int_{\mathbb{R}^d} \prod_{i=1}^{d} \chi_{S_i}(x_1, \cdots, \widehat{x_i}, \cdots, x_d) dx_1 \cdots dx_d. \tag{32}$$

By applying Lemma A.2 to the right hand side,

$$\begin{aligned} \mathrm{Vol}_d(V) \quad & \leq \quad \prod_{i=1}^{d} \left(\int_{\mathbb{R}^{d-1}} \chi_{S_i}(x_1, \cdots, \widehat{x_i}, \cdots, x_d) dx_1 \cdots \widehat{dx_i} \cdots dx_d \right)^{\frac{1}{d-1}} \\ & = \quad \prod_{i=1}^{d} \mathrm{Vol}_{d-1}(S_i)^{\frac{1}{d-1}}. \end{aligned} \tag{33}$$

□

Corollary A.3 (Projection Inequality)**.**

$$d \cdot \mathrm{Vol}_d(V)^{\frac{d-1}{d}} \leq \sum_{i=1}^{d} \mathrm{Vol}_{d-1}(S_i). \tag{34}$$

Proof. By applying the inequality of arithmetic and geometric means

$$\left(\prod_{i=1}^{n} a_i\right)^{\frac{1}{n}} \quad \leq \quad \frac{1}{n}\sum_{i=1}^{n} a_i \tag{35}$$

to the right hand side of Inequation (20). □

The authors speculate the relation between the projection inequality and the isoperimetric inequality, but the margin is too narrow to contain the proof.

B Extension to Assumptions Made in This Paper

There are other ways of using the cache than that assumed in §2. At iteration $i-1$, we can choose to keep data in $h(Q_{i-1})$ on cache, in addition to those in Q_{i-1}. Also, if we break the conditions for cache-filling batches and allow $h(Q_i) \cap Q_i \neq \phi$, we can further reduce the amount of loads by using computational results for some of the points in Q_i to compute other points in Q_i. Under these modifications, the set of grid points that are kept on cache and that are loaded are

$$Q'_{\mathrm{cache},i} \quad = \quad h(Q_i) \cap (Q_{i-1} \cup h(Q_{i-1})), \tag{36}$$

$$Q'_{\mathrm{ld},i} \quad = \quad h(Q_i) \backslash (Q_i \cup Q_{i-1} \cup h(Q_{i-1})), \tag{37}$$

respectively (c.f. Equations (7),(8)).

We have found many kinds of temporal blocking schemes that under Equations (36) and (37) show the same asymptotic B/F behavior (14) as PiTCH decomposition. Although they share drawback of having dependent computation within each Q_i, they might show advantage to PiTCH decomposition in other points.

So far we have not found any algorithm that achieves B/F lower than (14) even under Equations (36) and (37). We believe that the lower limit (14) holds even under the extensions, but we have not proven it yet.