Tessellating Stencils

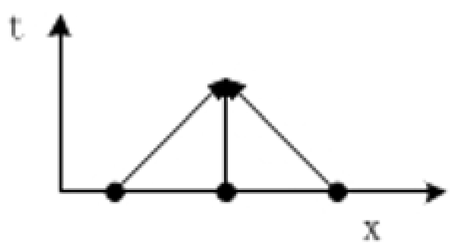
Liang Yuan, Yunquan Zhang, Peng Guo, Shan Huang
SKL of Computer Architecture, ICT, CAS

Outline

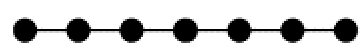
- Introduction
- Related work
- Tessellating Stencils

Stencil Overview

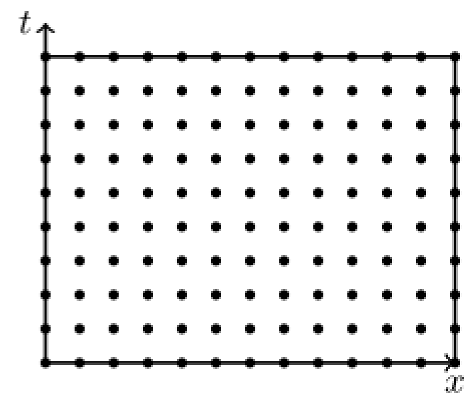
- Stencil
  - update each point in a *d*-dimensional space (*data space*) with a pre-defined pattern of neighbor points
  - time-iterated updates (*iteration space*)



1d3p stencil



Data space



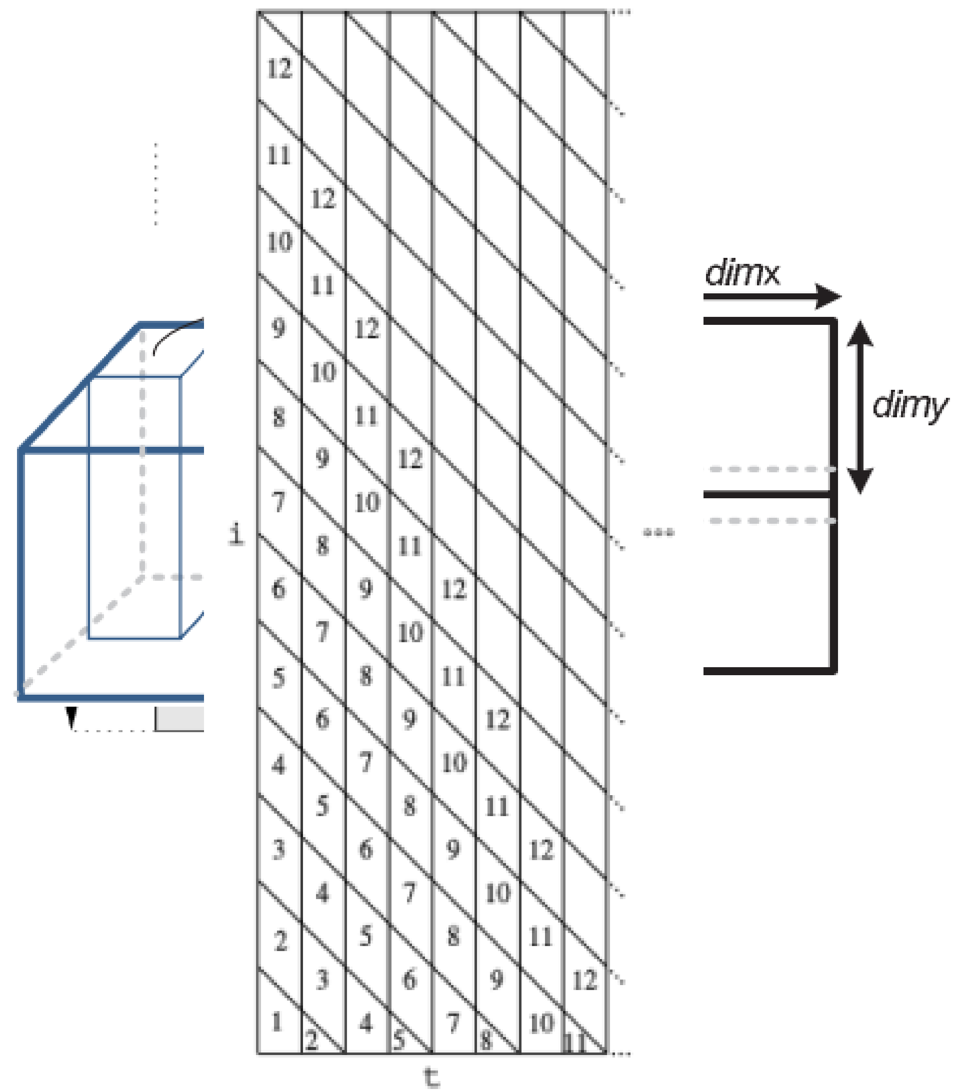
Iteration space

Stencil Overview

- Classification
  - grid dimensions (1D, 2D, ...)
  - number of neighbors (3-point, 5-point, ...)
  - shapes (box, star, ...)
  - dependence types (Gauss-Seidel, Jacobi)
  - boundary conditions (constant, periodic, ...)

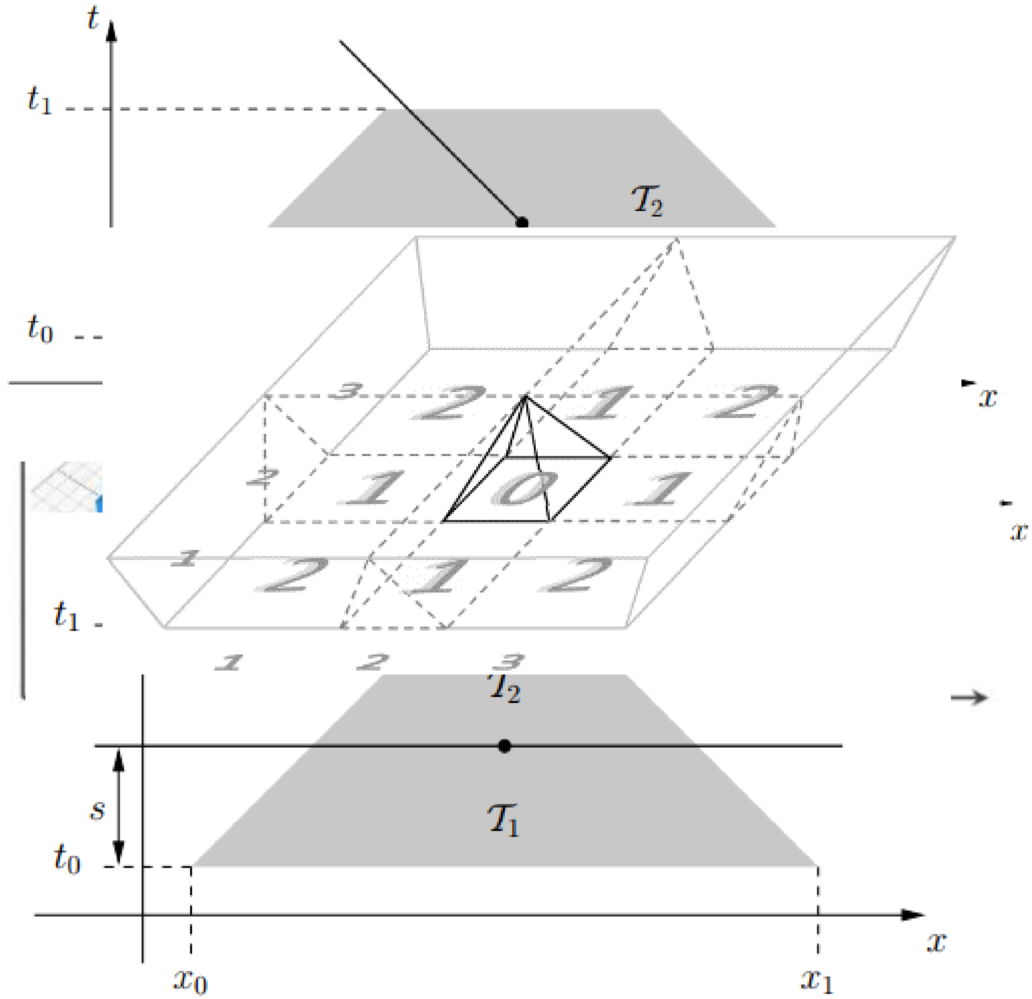
Related Work

- Overlapped tiling
  - Hyper-rectangle [PLDI'07]
  - 3.5d blocking [SC'10]
- Time skewing
  - eliminates the redundant computations [SC'01]
  - pipelined startup and limited concurrency [SPAA'01]



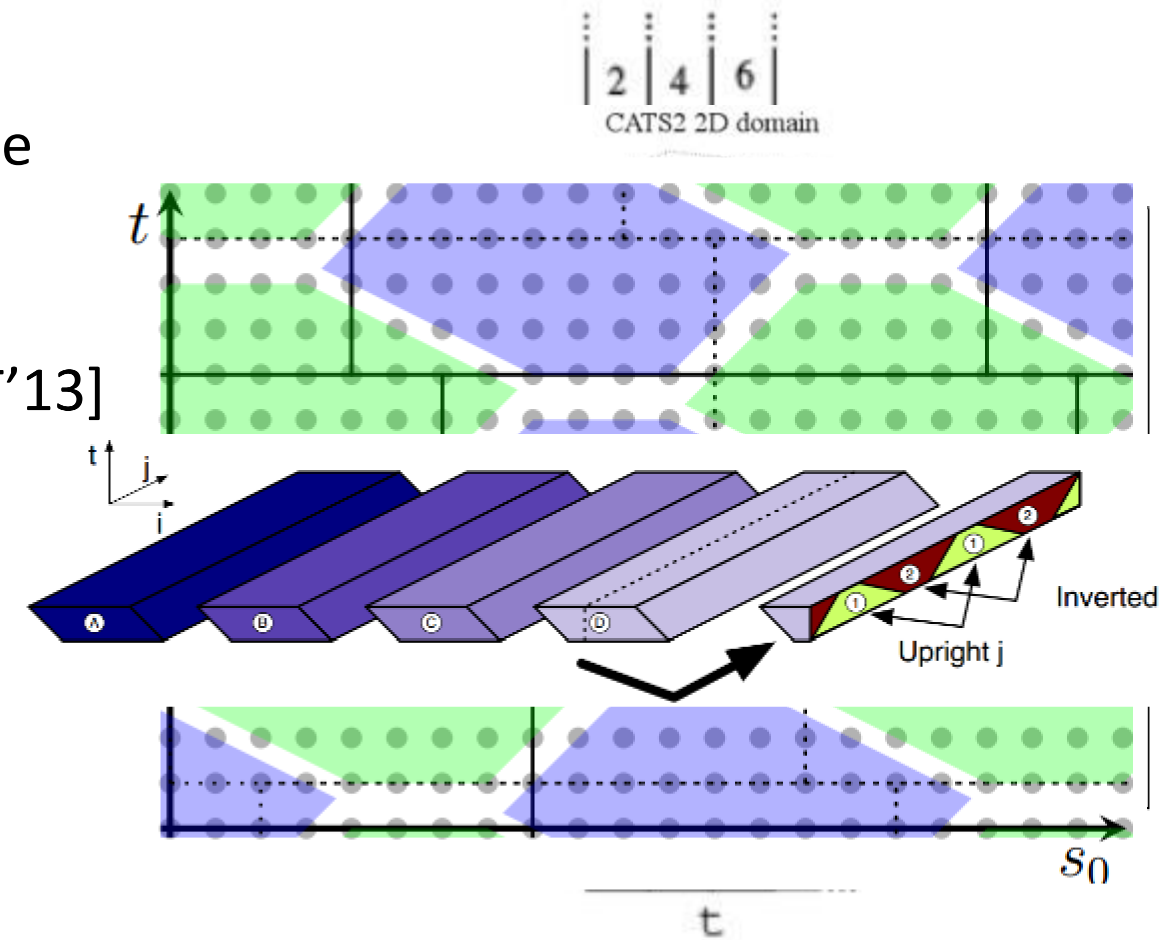
Related Work

- Diamond tiling
  - classic block [PLDI'08]
  - higher dimensional [SC'12]
  - hexagon in 2D and truncated octahedron in 3D [PPL'14]
- Cache oblivious tiling
  - serial cache oblivious stencil algorithms [ICS'05]
  - parallel cache oblivious stencil algorithms [SPAA'06]
  - Pochoir [SPAA'11]

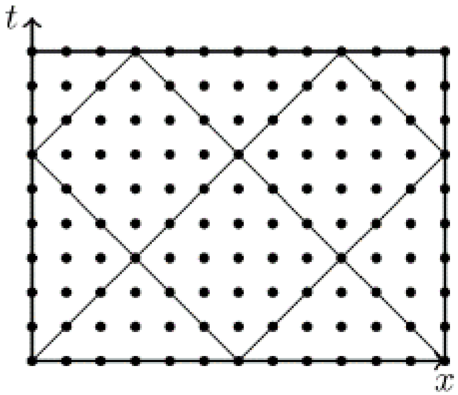


Related Work

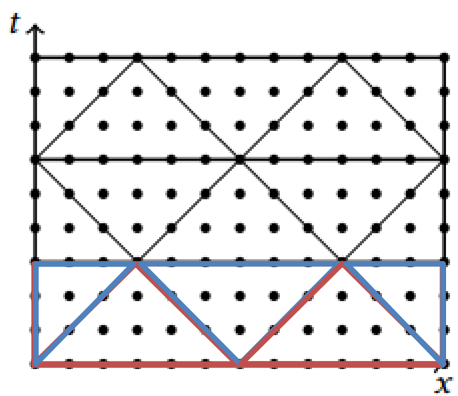
- Split tiling
  - avoids the overhead of the pipelined start-up in wavefront parallelization
  - classic split tiling [IMPACT'13]
  - nested split-tiling [ICS'13]
- Hybrid tiling
  - CATS [ICPP'11]
  - MWD [SIAMJOSC'15]
  - Grosser [CGO'14]
  - Hybrid split-tiling [ICS'13]



Reformulating Classic 1D Diamond Tiling

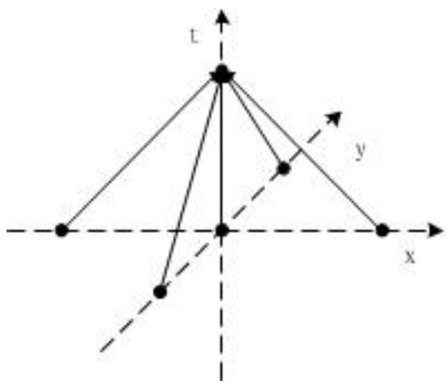


- Iteration space of 1D stencil
- Diamond tiling

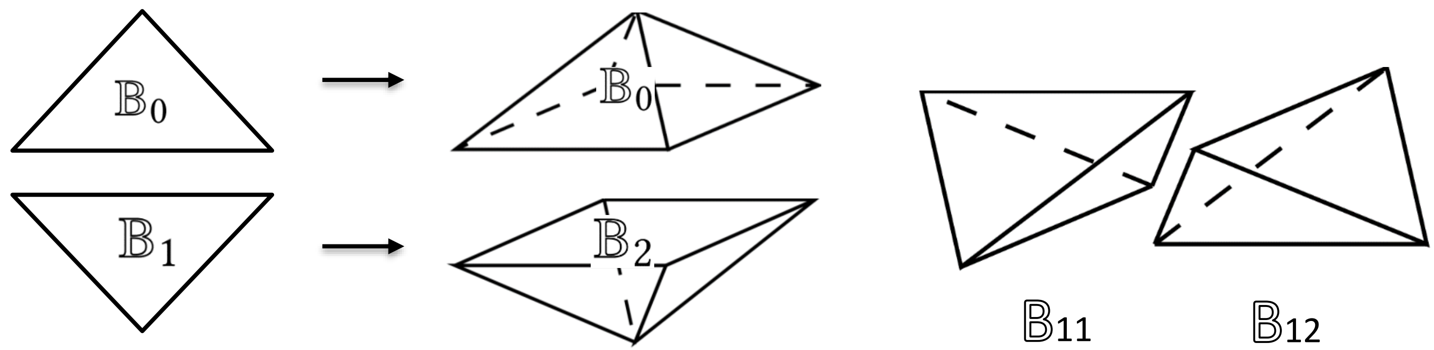


- Each diamond is further split by a horizontal line
- 2D *time tile.*
  - The region between two splitting lines
  - every grid point in a *time tile* starts in a same time dimension and is updated by identical steps.
  - interleaved computations of triangle ($\mathbb{B}_0$) or inverted triangle ($\mathbb{B}_1$)

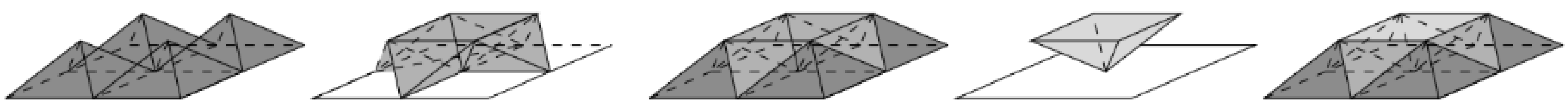
Extend to 2D Stencil



2d5p



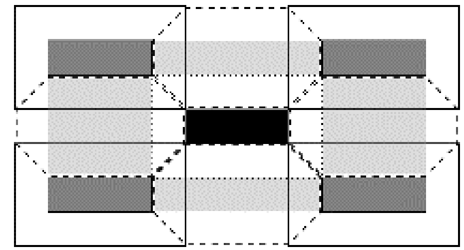
- Triangles in 1d stencil correspond to pyramids $\mathbb{B}_0$
- Inverted triangles correspond to inverted pyramids $\mathbb{B}_2$
- tetrahedrons $\mathbb{B}_1$



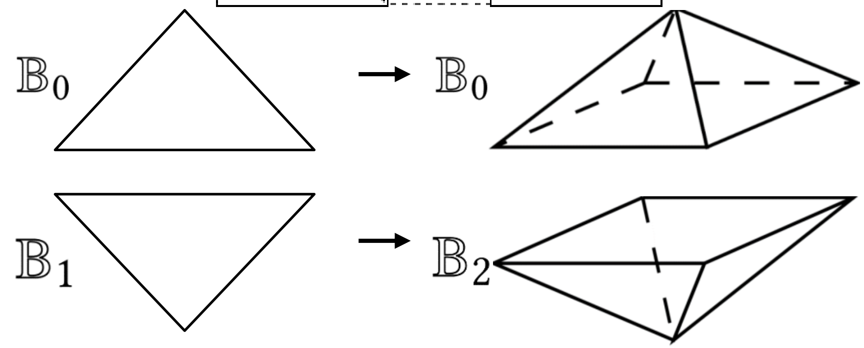
(a) $\mathbb{B}_0$ **(b)** $\mathbb{B}_1$ **(c)** $\mathbb{B}_0+\mathbb{B}_1$ **(d)** $\mathbb{B}_2$ **(e)** $\mathbb{B}_0+\mathbb{B}_1+\mathbb{B}_2$

Implementation

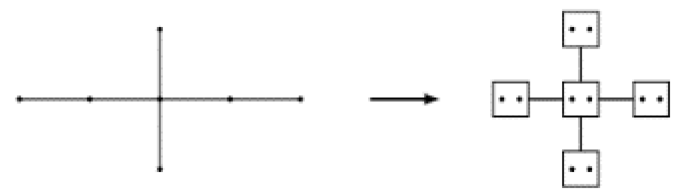
- Coarsening
  - cut in different sizes



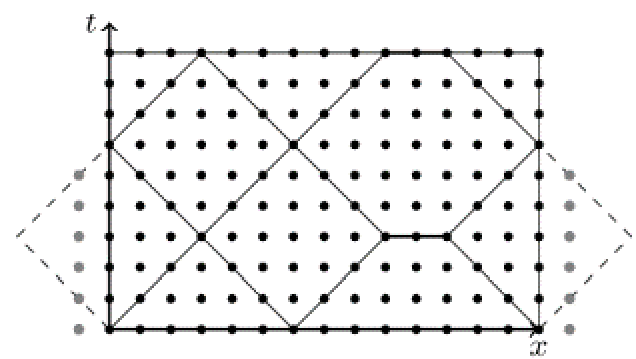
- Merge
  - $\mathbb{B}_0$ and $\mathbb{B}_1$ in 1D stencil.
  - $\mathbb{B}_0$ and $\mathbb{B}_2$ in 2D stencil.



- m-order stencil
  - combine every m points to a supernode.
  - 1-order one between supernodes.



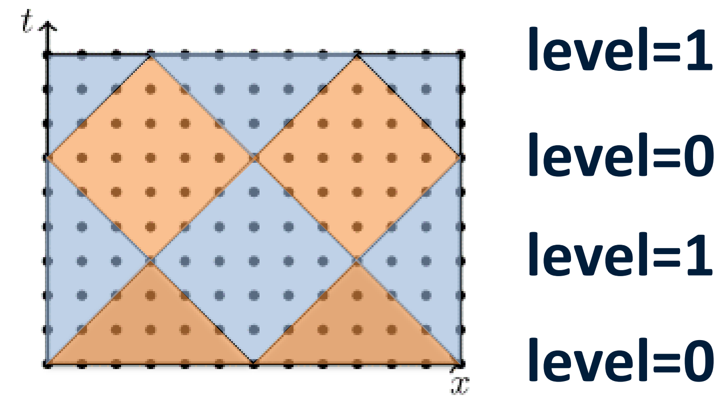
- Periodic boundary
  - stretch and transform one block



1D code

```
for(tt = -bt; tt < T ; tt += bt ){
    for(n = 0; n < #B0B1[level]; n++,level = 1 - level) {
        for(t = max(tt, 0) ; t < min( tt + 2 * bt, T); t++){
            xmin = max( XSLOPE, xright[level] - bx + n * ix
                        + abs(t+1, tt+bt) * XSLOPE);
            xmax = min( N + XSLOPE, xright[level] + n * ix
                        - abs(t+1, tt+bt) * XSLOPE);
            for(x = xmin; x < xmax; x++){
                update(t, x);}}}}
```

bt: block size in time dimension
bx: block size in data space
bx ≠ bt: support of coarsening
#B0B1 is different in different level
Determine the scope of 1D data space



2D code

```
for(tt = -bt; tt < T; tt += bt){
    for(n = 0; n < #B0B2[level]; n++){
        for(t = max(tt,0); t < min( tt + 2*bt, T); t++){
            calculate xmin, xmax, ymin, ymax of B0 and B2
            for(x = xmin; x < xmax; x++) {
                for(y = ymin; y < ymax; y++){
                    update(t, x, y); }}}}
    for(n = 0; n < #B1[0] + #B1[1]; n++){
        for(t = tt+bt ; t < min( tt + 2*bt, T); t++) {
            if(n<#B1[level]){
                calculate xmin, xmax, ymin, ymax of B11}
            else{
                calculate xmin, xmax, ymin, ymax of B12}
            for(x = xmin; x < xmax; x++) {
                for(y = ymin; y < ymax; y++){
                    update(t, x, y); }}}}
    level = 1 - level;}
```

#B0B2 represents #B0 in level 0 or #B2 in level 1

two kinds of B1: B11 and B12

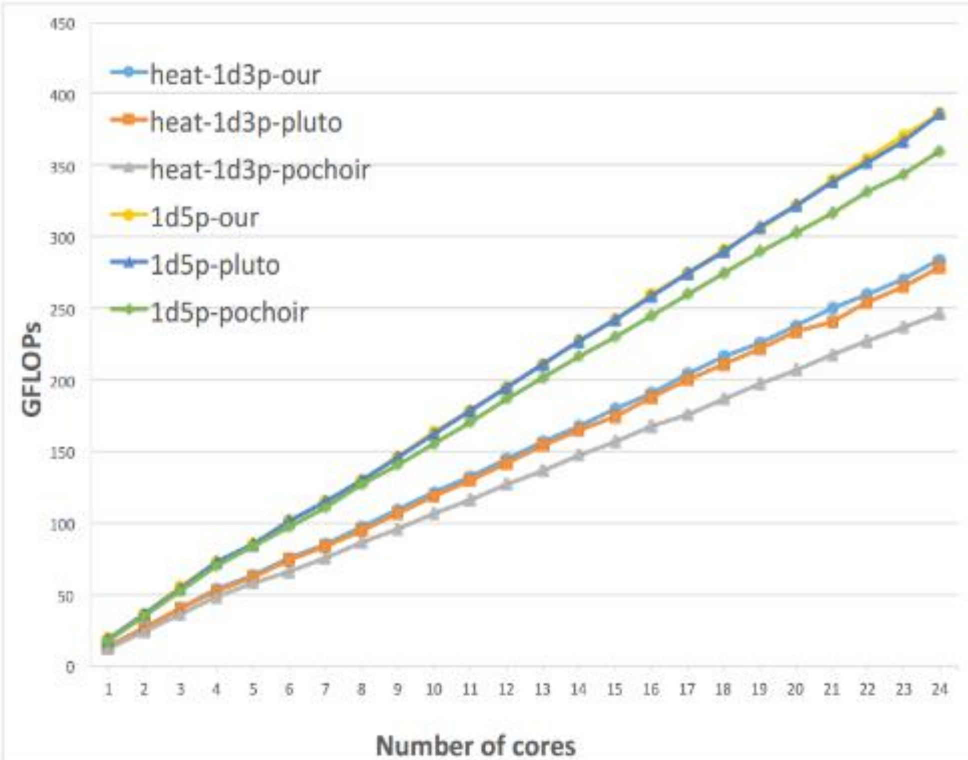
for 3D stencil we leave the unit-stride dimension uncut.

Evaluation

- two Intel Xeon E5-2670 processors with 2.70 GHz clock speed
- 32KB private L1 cache
- 256KB private L2 cache
- a unified 30MB L3 cache shared by 12 cores
- ICC complier version 16.0.1, flag ‘-O3 -openmp’.

Benchmark	Problem Size	our blocking	Pluto blocking
Heat-1D	12000000×4000	2000×1000	2000×2000
1d5p	12000000×4000	2000×500	2000×2000
Heat-2D	$6000^2 \times 2000$	$128 \times 256 \times 64$	$64 \times 64 \times 64$
2d9p	$6000^2 \times 2000$	$128 \times 256 \times 64$	$64 \times 64 \times 64$
Game of life	$6000^2 \times 2000$	$128 \times 256 \times 64$	$128 \times 128 \times 128$
Heat-3D	$256^3 \times 1000$	$24 \times 24 \times 12$	$12 \times 12 \times 12$
3d27p	$256^3 \times 1000$	$24 \times 24 \times 12$	$12 \times 12 \times 12$

Experimental Results

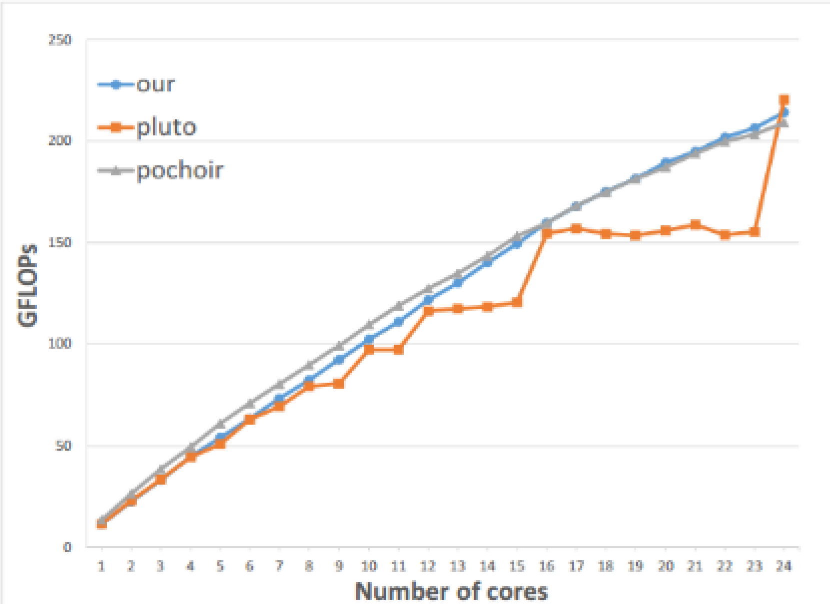


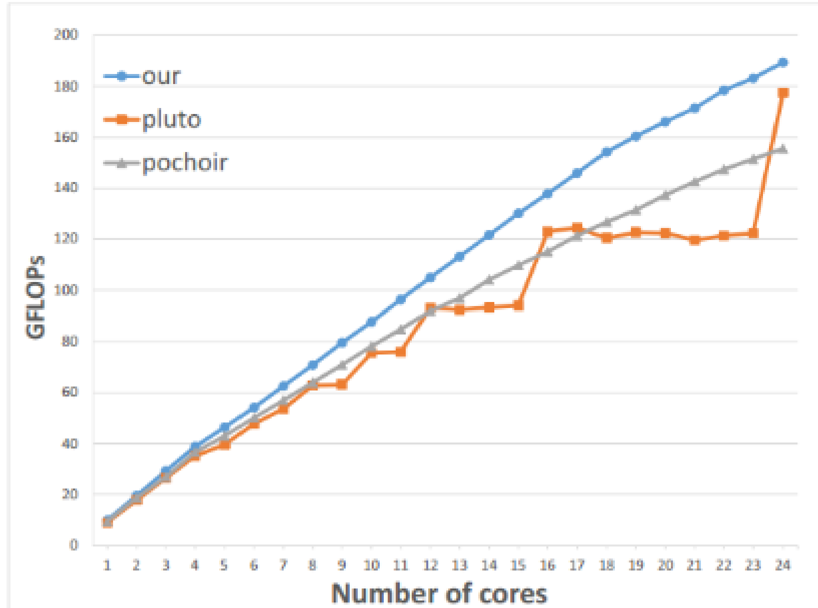
same to Pluto

better than Pochoir

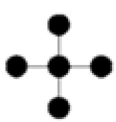
- Our scheme and PluTo produce the same diamond tiling codes
- Pochoir generates trapezoidal blocks of different sizes.

Experimental Results



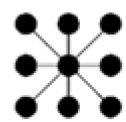


2d5p



Pluto outperforms less than 5% with 24 cores

2d9p

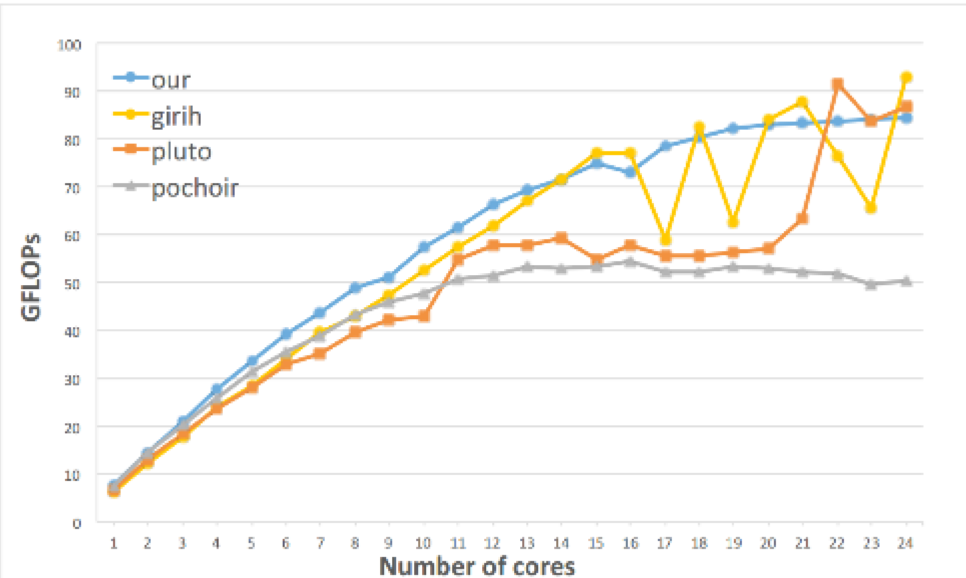


Our code outperforms Pluto and Pochoir by 14% and 20% on average.

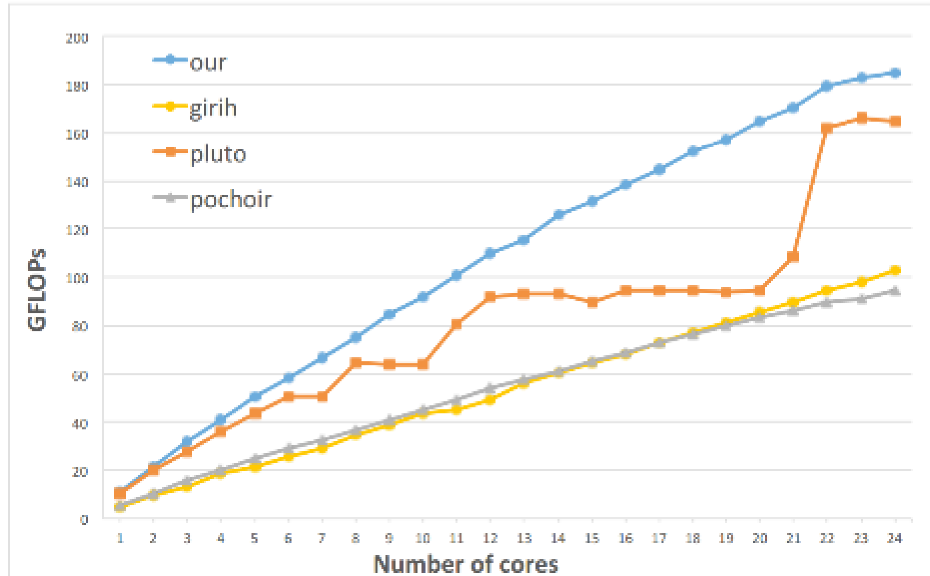
- more suitable for box stencil.

Experimental Results

- 3D stencil leaves unit-stride uncut



3d7p



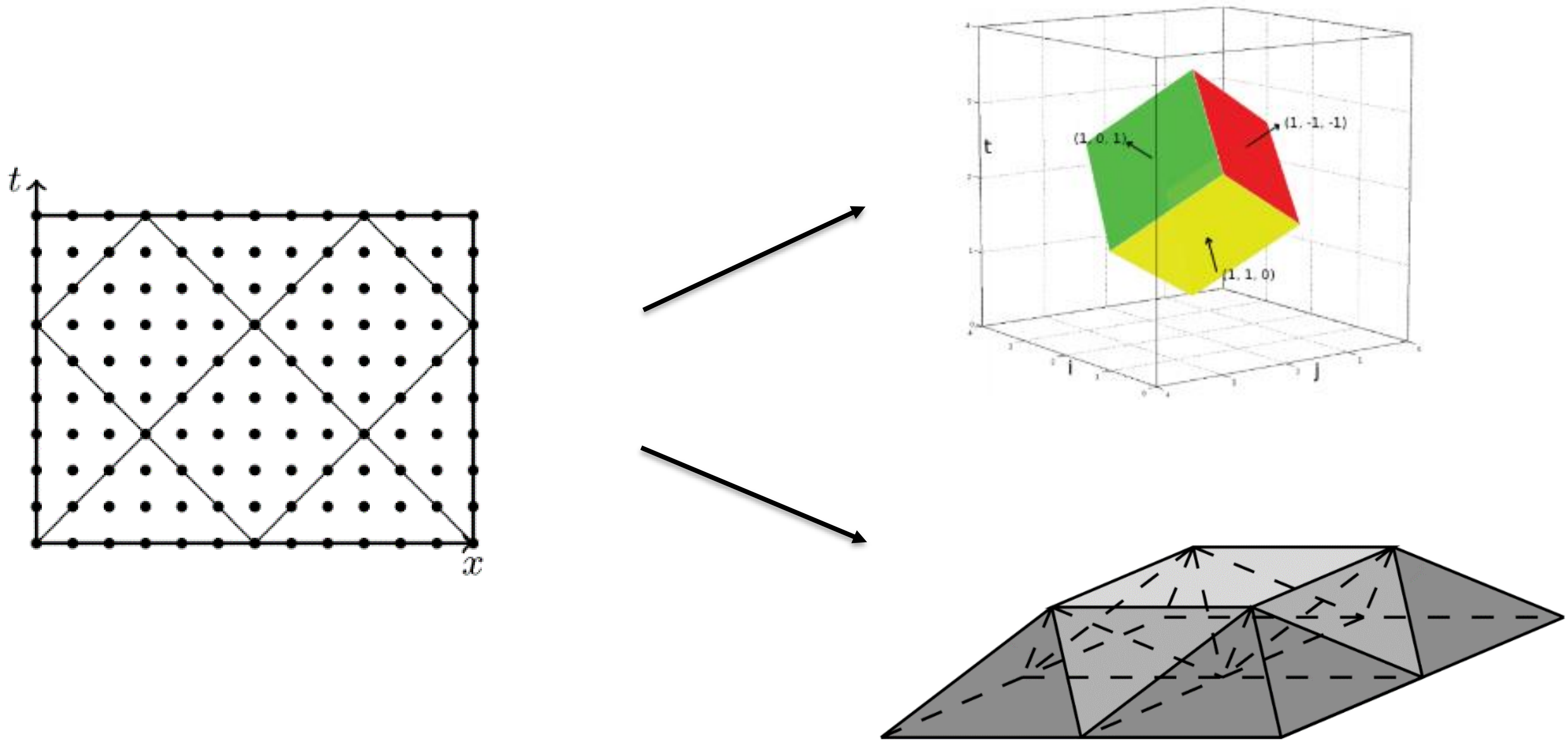
3d27p

Our code and Pochoir exhibit better scalability than Pluto.
Our code outperforms Pluto and Pochoir by a maximum of 33% and 68%, and by 22% and 31% on average.

Our code outperforms Pluto and Pochoir by a maximum of 74% and 100%, and by 30% and 99% on average.

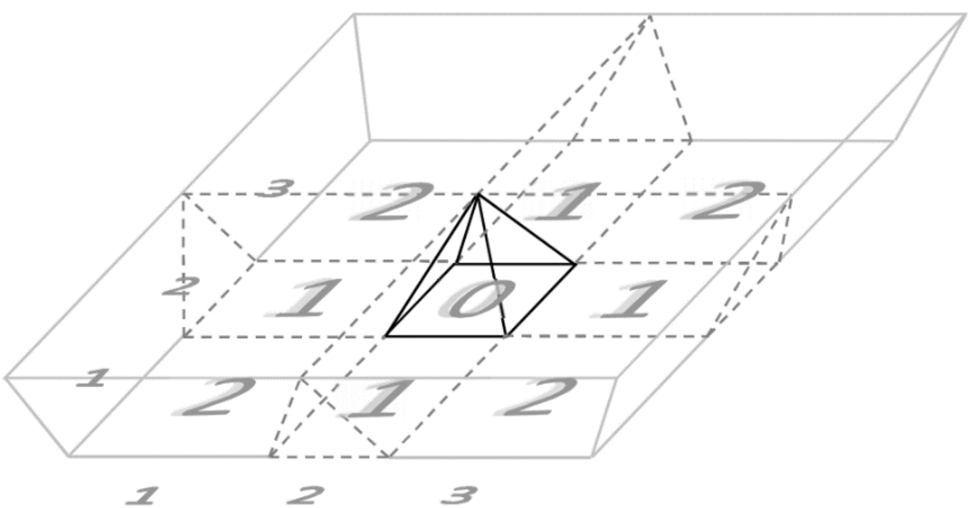
new?

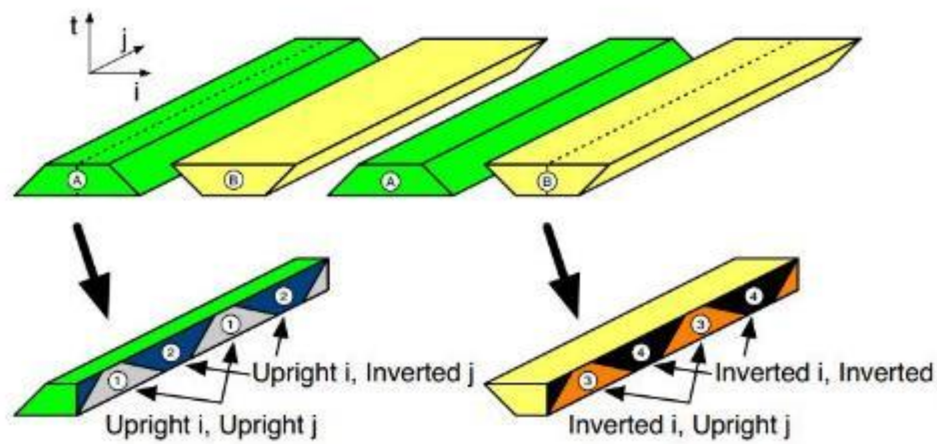
- extension of 1D diamond



Not ~~that~~ new!

- Existing techniques produce similar blocks
  - Cache oblivious [SPAA'11]
  - Nested split-tiling [ICS'13]
  - divide multiple dimensions simultaneously

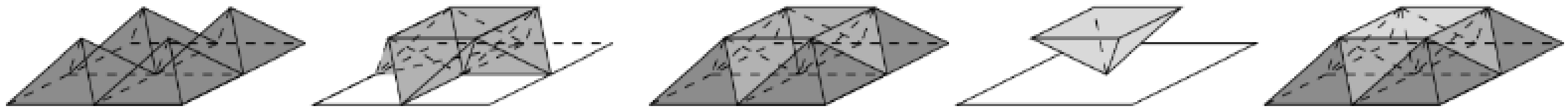




Advantage!

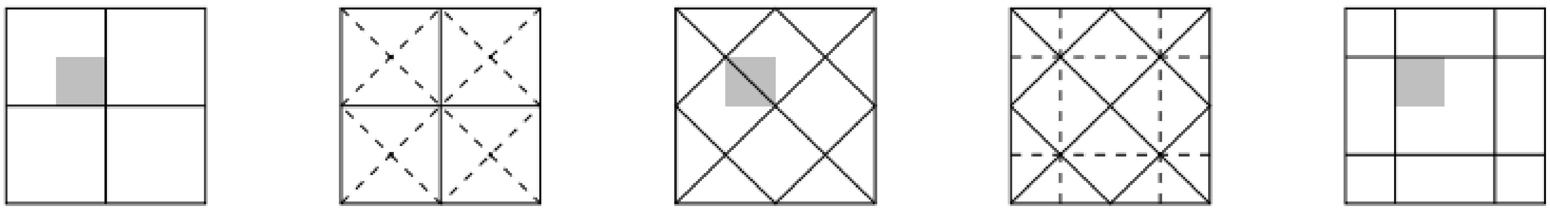
- Diamond tiling
  - fixed tile size at compile time
  - small size at the apex of a diamond
  - hard to choose the proper tile sizes to ensure the concurrent start
- Cache oblivious tiling
  - overhead of recursion
  - artificial dependencies
- Split tiling
  - synchronization overhead 2^d

Formulating the two-level tessellation



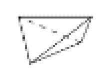
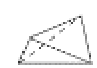
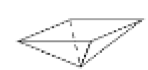
(a) $\mathbb{B}_0$ (b) $\mathbb{B}_1$ (c) $\mathbb{B}_0$+$\mathbb{B}_1$ (d) $\mathbb{B}_2$ (e) $\mathbb{B}_0$+$\mathbb{B}_1$+$\mathbb{B}_2$

Project $\mathbb{B}_i$ in iteration space on data space, denoted as B_i



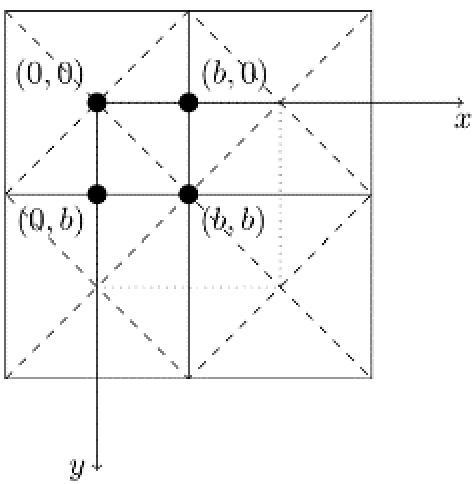
(a) B_0 (b) $\mathrm{B}_0 \rightarrow \mathrm{B}_1$ (c) B_1 (d) $\mathrm{B}_1 \rightarrow \mathrm{B}_2$ (e) B_2

Determine the number of updating of each point in B_i.

	B_i'	step1	step2	step3	$\mathbb{B}_i$
i = 0	0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 1 2 2 2 1 0 0 1 2 3 2 1 0 0 1 2 2 2 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1	1	
i = 1	0 0 1 0 0 1 2 1 0 0 1 2 3 2 1 0 0 1 2 1 0 0 1 0 0	1 1 1 1 1	1 1 1 1 1 1 1 1 1	1 1 1 1 1	
i = 1	0 0 1 0 0 1 2 1 0 0 1 2 3 2 1 0 0 1 2 1 0 0 1 0 0	1 1 1 1 1	1 1 1 1 1 1 1 1 1	1 1 1 1 1	
i = 2	0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 1 2 2 2 1 0 0 1 2 3 2 1 0 0 1 2 2 2 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0	1	1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	

Formulating the two-level tessellation

- Consider a point *a* (*a0, . . . ,ad−1*) in B0(0, . . . , 0)
- updated time in B*i*, denoted as *Ti*(*a0, . . . ,ad−1*).



$$T_i(a_0, \ldots, a_{d-1}) = \min(b, a_i, \ldots, a_{d-1}) - \max(0, a_0, \ldots, a_{i-1})$$

$$\sum_{i=0}^{d} T_i(a_0, \ldots, a_{d-1}) = (b - a_0) + (a_0 - a_1) + \cdots + (a_{d-2} - a_{d-1}) + a_{d-1} = b$$

$a_i >= a_{i+1}$

Summary

- two-level tessellation scheme
  - with highly concurrent execution
  - executes without redundant computation
  - achieves maximize parallelism
  - without false dependencies.
  - Calculate blocks without relying on the compiler
  - Extend to n-dimensional stencil

- mathematical structure of tessellation
  - Associate coordinates of elements with number of updating steps for each block.