Tokyo Tech

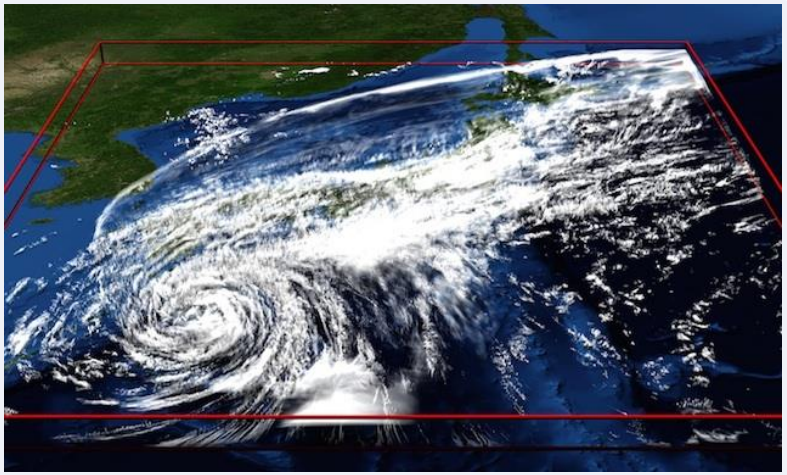
CGO 2020 (25/2/2020)

AN5D: Automated Stencil Framework for High-Degree Temporal Blocking on GPUs

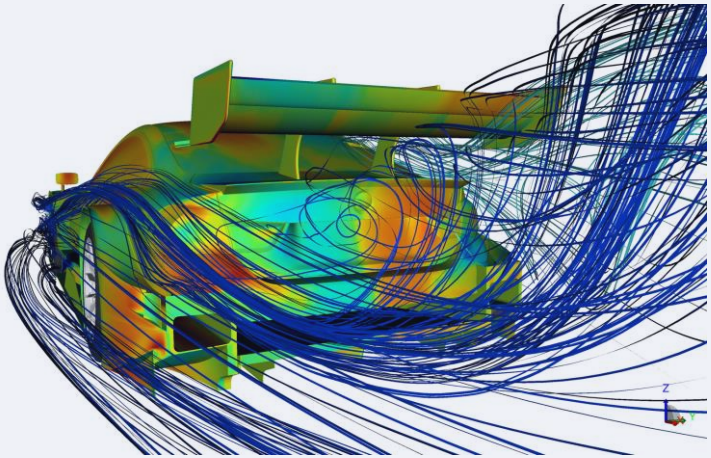
Kazuaki Matsumura [1*], Hamid Reza Zohouri [2*], Mohamed Wahib [3]
Toshio Endo [4], Satoshi Matsuoka [5]

[1] Barcelona Supercomputing Center, Spain [2] Edgecortix Inc., Japan [3] AIST, Japan [4] Tokyo Institute of Technology, Japan
[5] RIKEN CCS, Japan [*] *Work performed while those authors were at Tokyo Tech and K. Matsumura at RWBC-OIL*

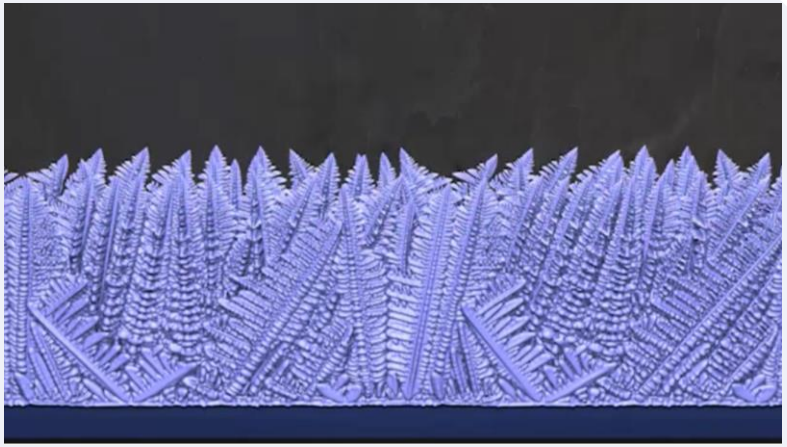
Background – Iterative Stencil Computation



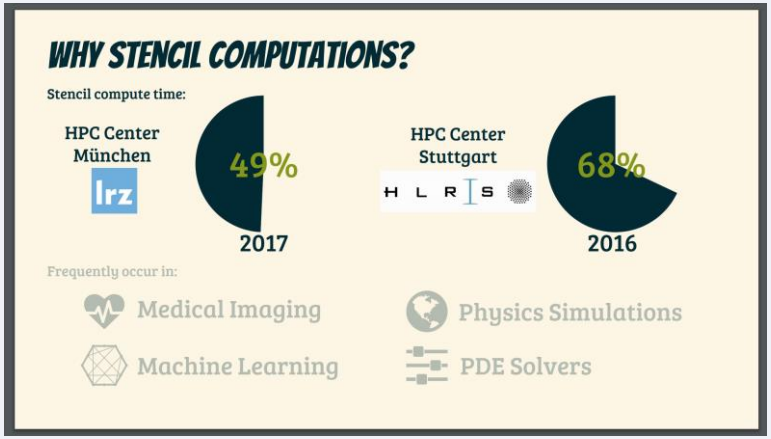
Weather Prediction



Flow Simulation



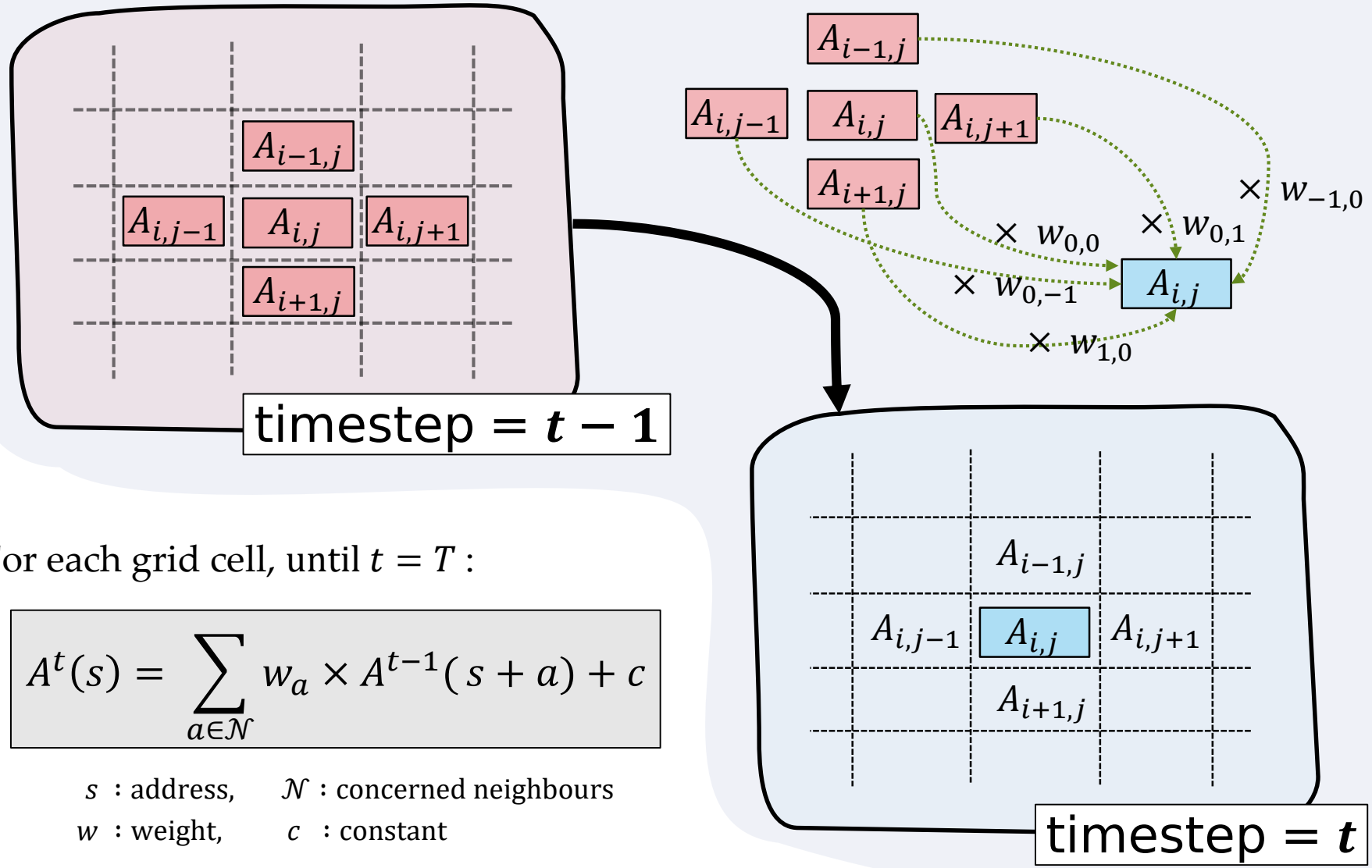
Material Simulation



Stencil Computation in HPC Centers [Hagedron, CGO2018]

http://www.sim.gsic.titech.ac.jp/Japanese/Member/shimokawabe/research.html
http://www.lift-project.org/presentations/2018/CGO-2018.pdf
https://www.press.bmwgroup.com/global/article/detail/T0269421EN/the-role-of-powerful-computers-in-the-development-of-the-bmw-m4-dtm:-a-detailed-look-at-computational-fluid-dynamics

Background – Iterative Stencil Computation



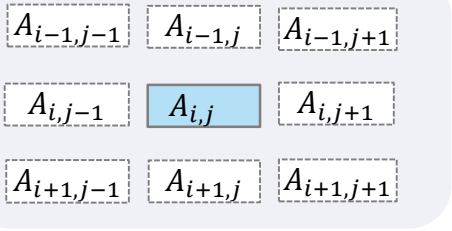
For each grid cell, until $t = T$:

$$A^t(s) = \sum_{a \in \mathcal{N}} w_a \times A^{t-1}(s+a) + c$$

s : address, $\mathcal{N}$: concerned neighbours

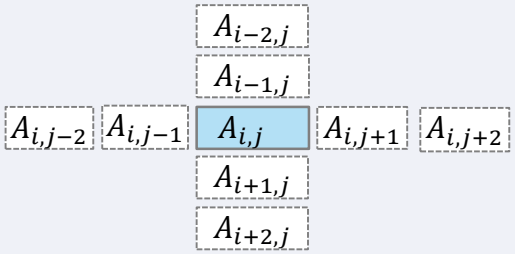
w : weight, c : constant

Background – Iterative Stencil Computation



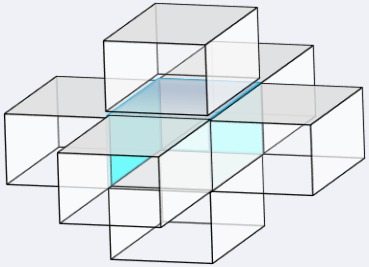
Heat 2D

- $\mathrm{Num}(\mathcal{N}) = 9$
- B/F = 8 / 18 = 0.444



High-order 2D

- $\mathrm{Num}(\mathcal{N}) = 9$
- B/F = 8 / 18 = 0.444



Diffusion 3D

- $\mathrm{Num}(\mathcal{N}) = 7$
- B/F = 8 / 13 = 0.615

* Single precision, with full spatial locality optimization

- Arbitrary pattern & arbitrary dimension: depending on the application
- Typically having high Bandwidth/FLOP (B/F) ratio
- A fundamental computation pattern frequently **offloaded to GPUs**

Background – Bandwidth/FLOP on GPUs

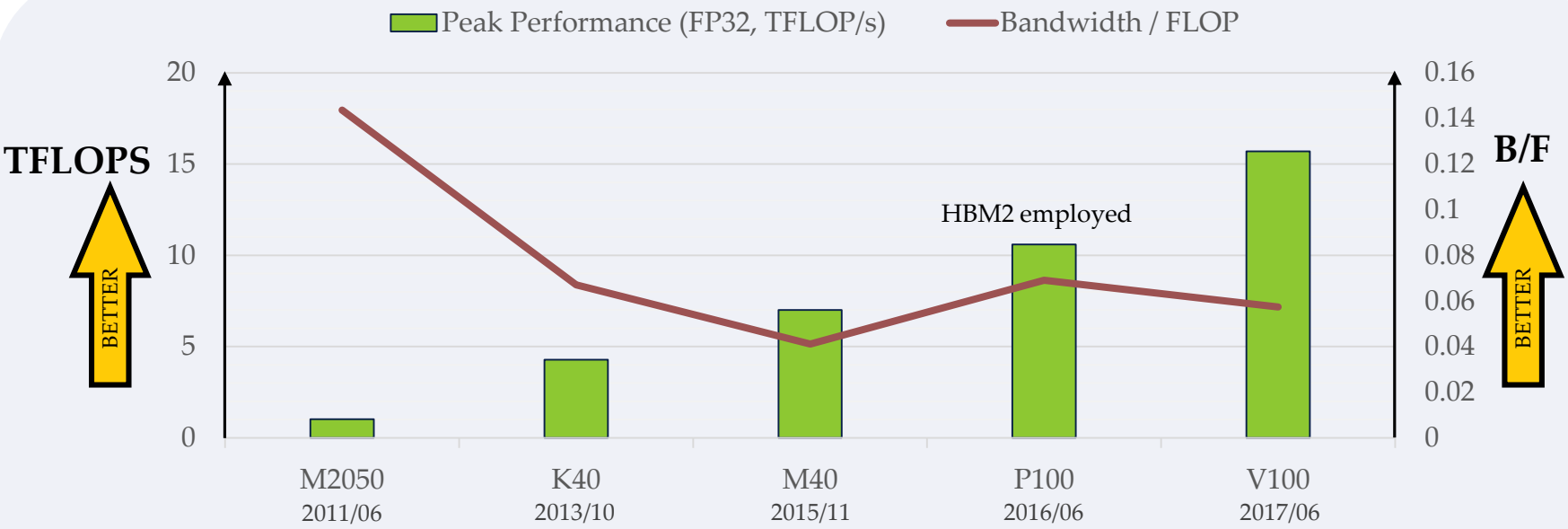
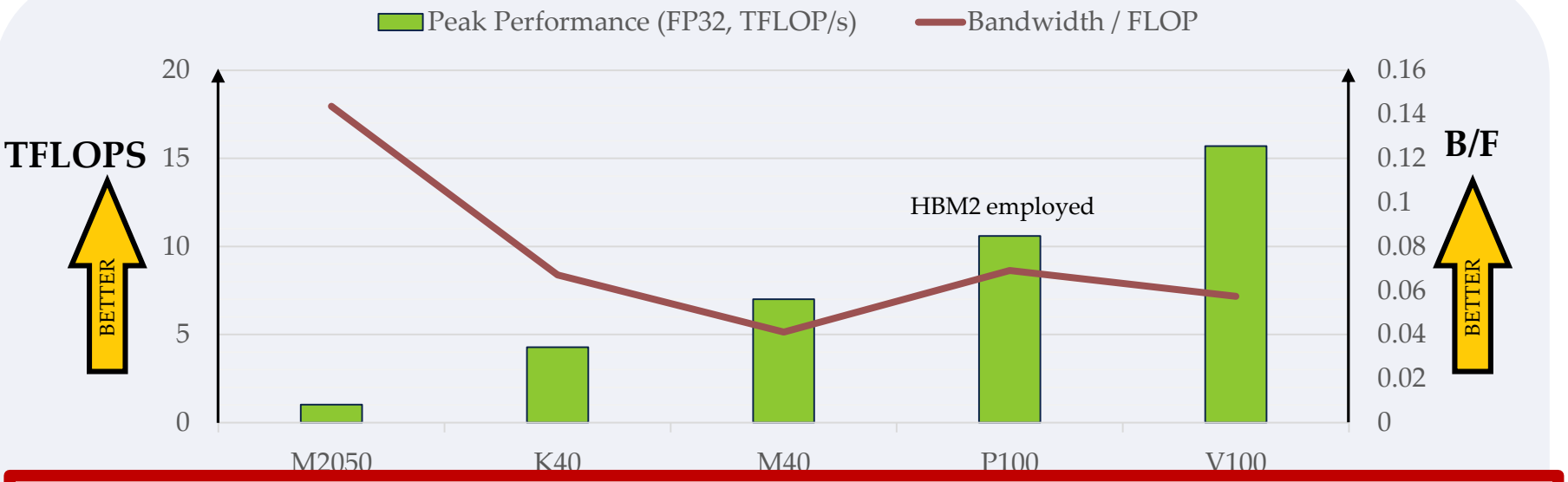


Fig. Performance changes of GPUs

- B/F ratio tends to decrease along with the increase of computation performance
  - Insufficient to maximize computational utilization for stencils
  - Stencils' performance is limited by the bandwidth → **Memory Bound**

https://www.nvidia.com/docs/IO/43395/NV_DS_Tesla_M2050_M2070_Apr10_LowRes.pdf https://www.nvidia.co.jp/content/tesla/pdf/NVIDIA-Tesla-Kepler-Family-Datasheet.pdf
https://devblogs.nvidia.com/accelerating-hyperscale-datacenter-applications-tesla-gpus/ https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf https://www.nvidia.com/content/PDF/Volta-Datasheet.pdf

Background – Bandwidth/FLOP on GPUs



To overcome the bottleneck on memory accesses,

we employ a *cache-friendly* algorithm: **Temporal Blocking**

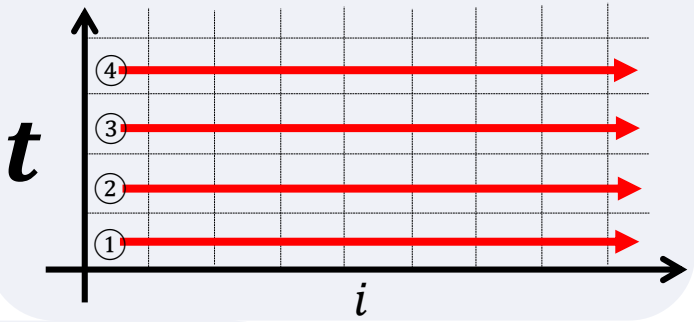
- Insufficient to maximize computational utilization for stencils
- Stencils' performance is limited by the bandwidth → **Memory Bound**

https://www.nvidia.com/docs/IO/43395/NV_DS_Tesla_M2050_M2070_Apr10_LowRes.pdf https://www.nvidia.co.jp/content/tesla/pdf/NVIDIA-Tesla-Kepler-Family-Datasheet.pdf
https://devblogs.nvidia.com/accelerating-hyperscale-datacenter-applications-tesla-gpus/ https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf https://www.nvidia.com/content/PDF/Volta-Datasheet.pdf

Temporal Blocking

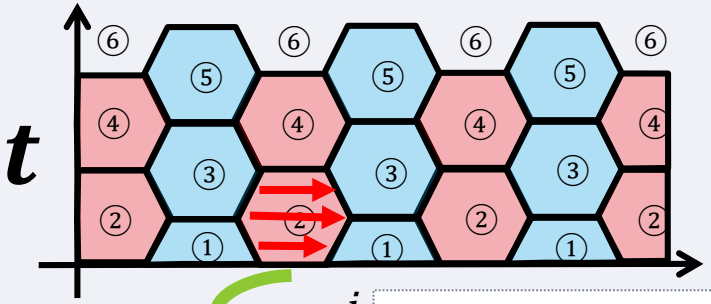
1D : $A_{i-1} \rightarrow A_i \leftarrow A_{i+1}$

Naïve Algorithm (1D)

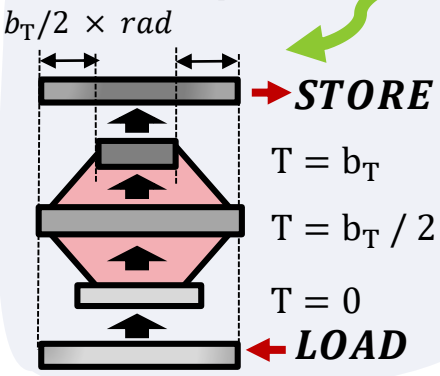


Execution Order

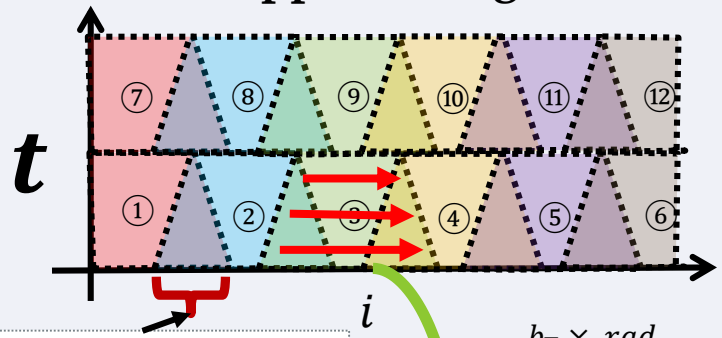
Hexagonal Tiling (1D)



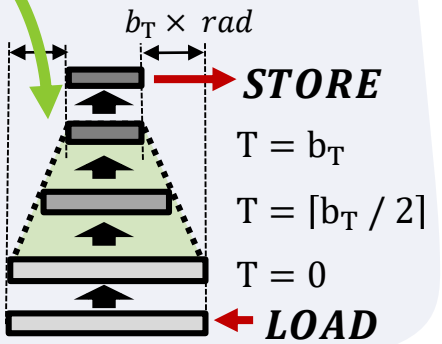
No redundant computation



Overlapped Tiling (1D)



Overlaps that cause **redundant computation**



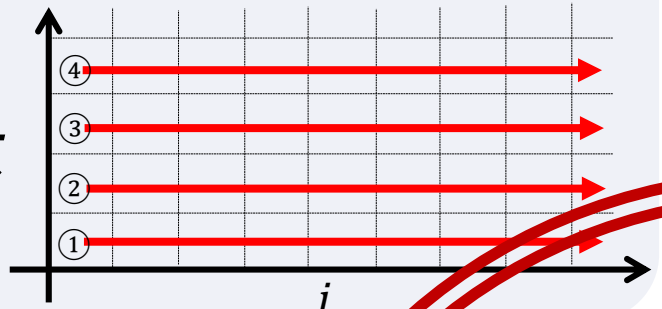
TB reuses previous time-steps' result kept in local memory

Temporal Blocking

1D : $A_{i-1} \rightarrow A_i \leftarrow A_{i+1}$

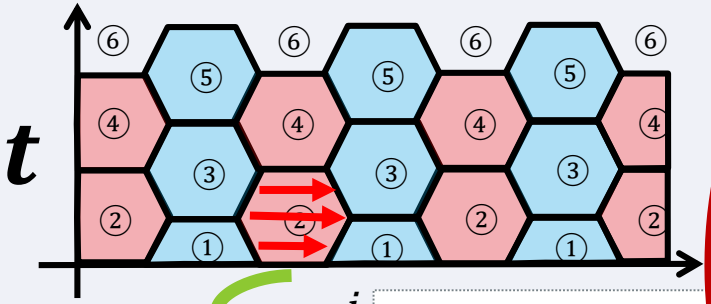
Tokyo Tech

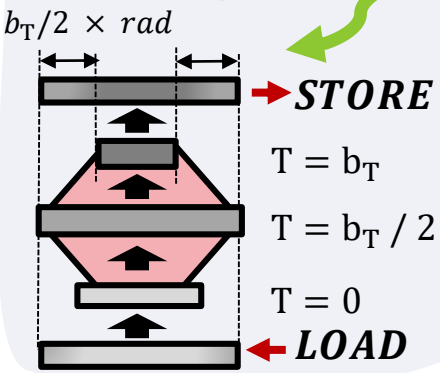
Naïve Algorithm (1D)



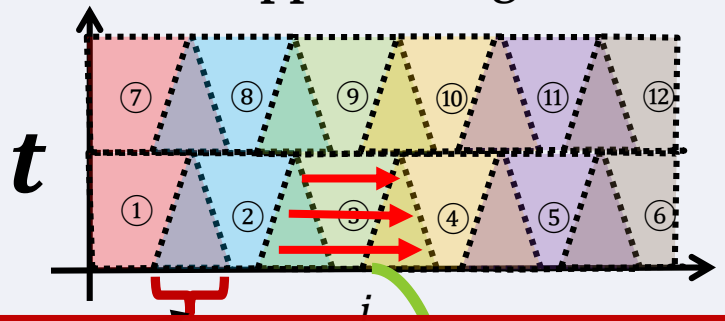
Execution Order

Hexagonal Tiling (1D)





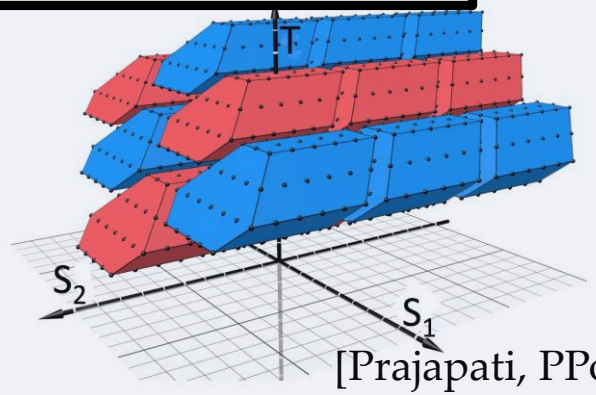
Overlapped Tiling (1D)



Applicative to **streaming execution** for higher dimensions

Temporal Blocking for Higher Dimensions

WITHOUT STREAM

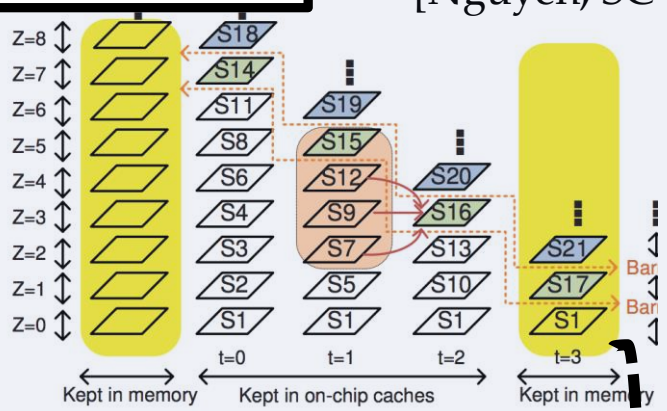


[Prajapati, PPoPP'17]

Hybrid Tiling
1D Hexagonal + (N-1)-D time-skewing

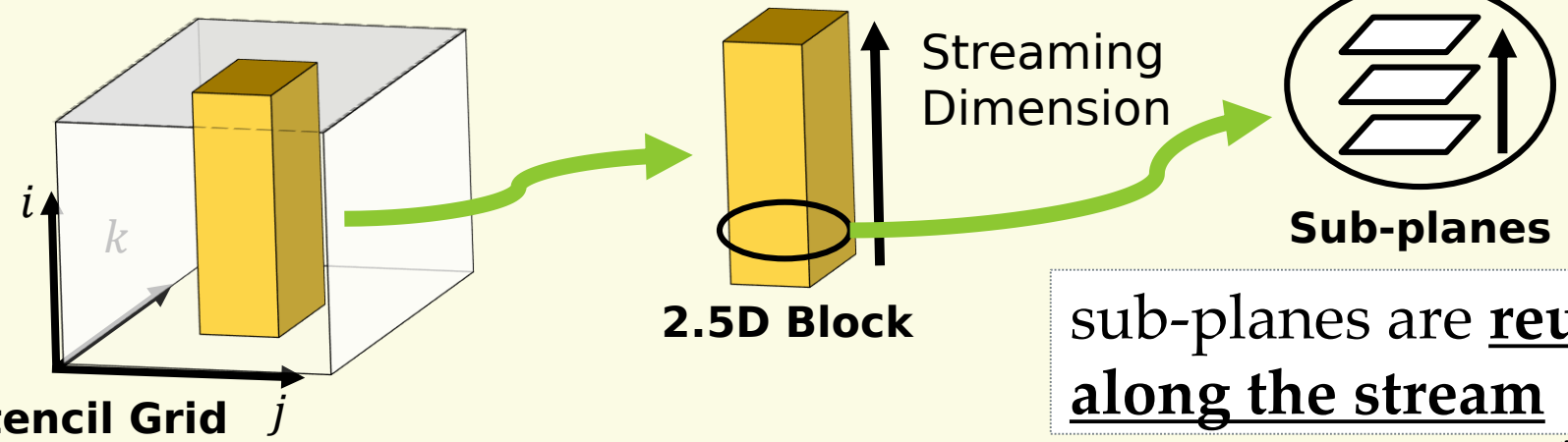
WITH STREAM

[Nguyen, SC'10]



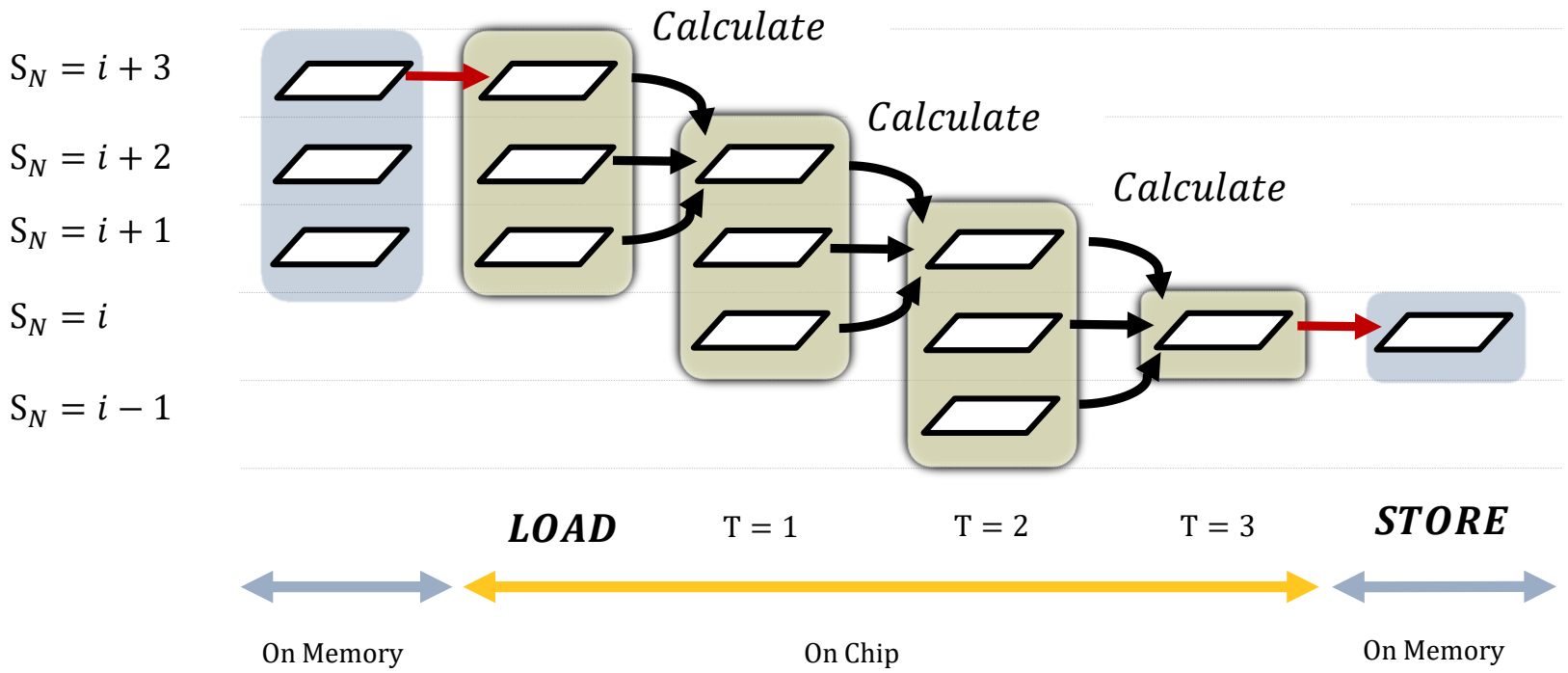
3.5D Blocking
2.5D Spatial & Overlapped

2.5D Blocking 2D Spatial + Streaming (0.5D)



sub-planes are **reused along the stream**

N.5D Blocking



- **Load one sub-plane → Update sub-planes b_T times → Store one sub-plane**
- The number of sub-planes that are kept on each time-step corresponds to the longest distance of neighboring accesses
- The computation flows in wavefronts along time-steps to satisfy dependency

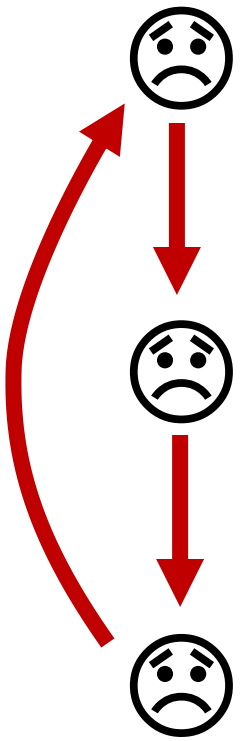
Difficulty on N.5D Blocking

☹ **Hard to implement** (conditions, branches, ..)

☹ **Hard to optimize** (resource restrictions on GPUs)

☹ **Hard to tune** (block/thread/streaming size)

Difficulty on N.5D Blocking



Hard to implement (conditions, branches, ..)

Hard to optimize (resource restrictions on GPUs)

Hard to tune (block/thread/streaming size)

AN5D: Automated N.5D Framework

Tokyo Tech

☺ AN5D **automates N.5D implementation**

→ **Generate CUDA code from stencil patterns in C Code**

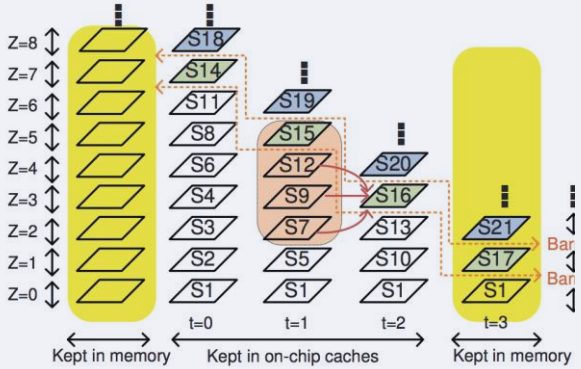
☺ AN5D integrates **on-chip resource optimization**

→ **Unprecedentedly scale with temporal blocking degree up to 10**

☺ AN5D's parameters are **tuned by the roof-line model**

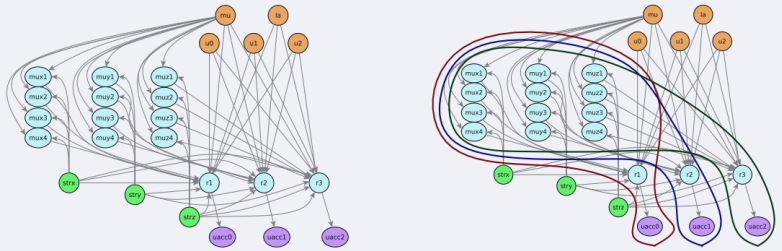
→ **Achieve the highest performance reported so far on V100 GPU**

Previous Work



3.5D Blocking [Nguyen, SC'10]

- (Supposedly manual) implementation that makes 7/27-point 3D stencil and 19-point LBM compute bound on NVIDIA GeForce GTX285
- Temporal-blocking degree up to $b_T \leq 3$



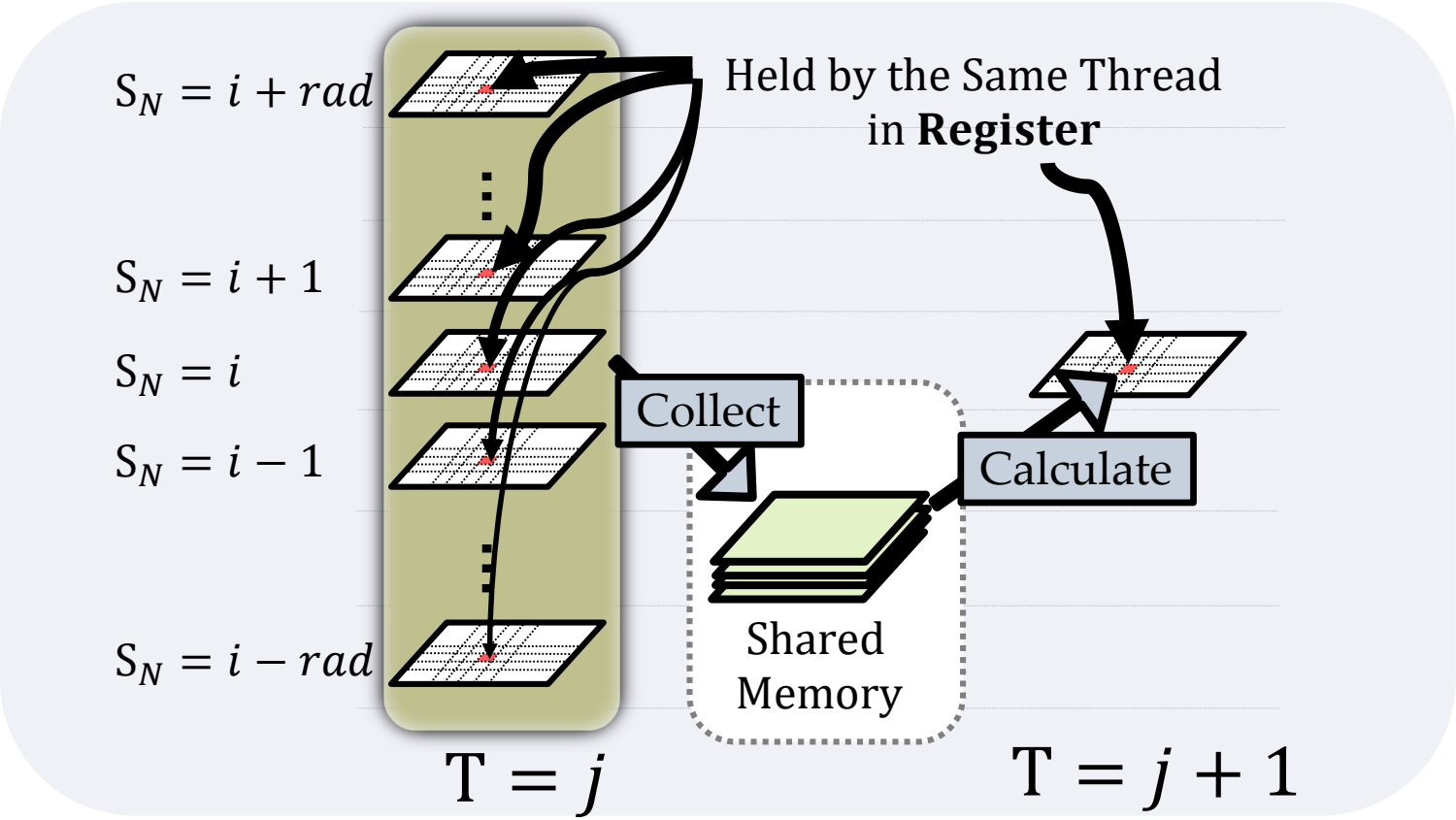
STENCILGEN or ARTEMIS

[Rawat, IEEE 2018 / IPDPS'19]

- Another automated framework for N.5D blocking that re-arranges multi-statements for less register use especially for complex stencils
- Temporal-blocking degree scales up to $b_T = 4$ **(HIGH SM/REG USE)**

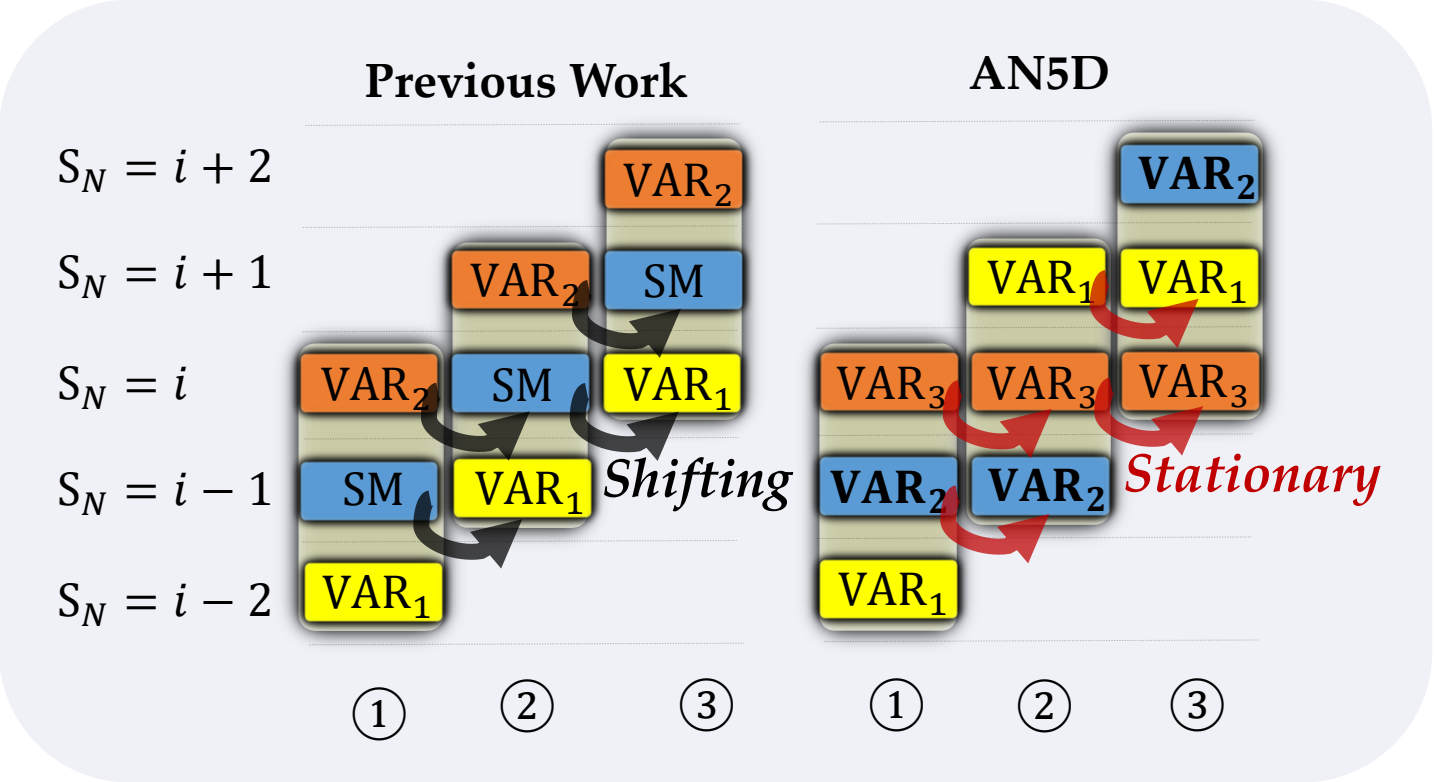
→ **AN5D does not re-build the sequence of compute statements but works around the computational flow**

Shared Memory Optimization



- **Don't use shared memory along the stream**; Use variables (registers)
  - On the computation, AN5D uses SM as a temporal double buffer
  - SM use decreased to $2 / b_T$; but ***more register use***

Register Optimization



- Static variable assignment to each sub-plane computation
  - Reduces data stores from $1 + 2 \times rad$ to 1
  - ***Allows fully loop unrolling***

STENCILGEN CODE

```
1  Initial_Computation;
2  for (/* Streaming */) {
3    if (Inside_Block_t0) {                                  } LOAD
4      /* Data Shifting */
5      b0 = SM0[...]; SM0[...] = t0; t0 = input[...];
6    }
7    __syncthreads();                                        }
8    if (Inside_Block_t1) {                                  }
9      /* Compute */                                         }
10     float __temp_2__ = b0;                                }
11     float __temp_5__ = SM0[...];                          } CALC1
12     // ...                                                }
13     float __temp_17__ = t0;                               }
14     float __temp_18__ = (__temp_14__ + 5.2f * __temp_17__);}
15     float __temp_19__ = (__temp_18__ / 118);              }
16     /* Data Shifting */                                   }
17     b1 = SM1[...]; SM1[...] = t1;  t1 = __temp_19__;      }
18   }
19   __syncthreads();                                        } CALC2
20   if (Inside_Block_t2) { /* Compute, Data Shifting */ }   }
21   __syncthreads ();                                       } CALC3
22   if (Inside_Block_t3) { /* Compute, Data Shifting */ }   }
23   __syncthreads ();                                       }
24   if (Inside_Block_t4) {                                  }
25     /* Compute */                                         }
26     float __temp_80__ = b3;                               }
27     float __temp_81__ = SM3[...];                         }
28     // ...                                                } STORE
29     float __temp_87__ = t3;                               }
30     float __temp_88__ = (__temp_86__ + 5.2f * __temp_87__);}
31     float __temp_89__ = (__temp_88__ / 118);              }
32     /* Output */                                          }
33     output[...] = __temp_89__;                            }
34   }                                                       }
35 }
```

AN5D CODE

```
1  if (Lowermost_Block) {                                                                  }
2    LOAD(reg_3_0, 0); LOAD(reg_0_1, 1); LOAD(reg_0_2, 2);                                 }
3    CALC1(reg_1_1, reg_3_0, reg_0_1, reg_0_2);                                            }
4    LOAD(reg_0_0, 3); CALC1(...); CALC2(...);                                             }
5    LOAD(reg_0_1, 4); CALC1(...); CALC2(...); CALC3(...);                                 }
6    LOAD(reg_0_2, 5); CALC1(...); CALC2(...); CALC3(...);                                 }
7    STORE(1, reg_3_0, reg_3_1, reg_3_2);                                                  }
8    LOAD(reg_0_0, 6); /* ... */                                                         }
9    STORE(4, reg_3_0, reg_3_1, reg_3_2);                                                  }
10 }                                                                                       }
11 else {                                                                                  }
12   LOAD(reg_0_0, 0); LOAD(reg_0_1, 1); LOAD(reg_0_2, 2);                                 }
13   CALC1(reg_1_1, reg_0_0, reg_0_1, reg_0_2);                                            }
14   LOAD(reg_0_0, 3); CALC1(...);                                                         }
15   LOAD(reg_0_1, 4); CALC1(...); CALC2(...);                                             }
16   LOAD(reg_0_2, 5); CALC1(...); CALC2(...);                                             }
17   LOAD(reg_0_0, 6); CALC1(...); CALC2(...); CALC3(...);                                 }
18   LOAD(reg_0_1, 7); CALC1(...); CALC2(...); CALC3(...);                                 }
19   LOAD(reg_0_2, 8); CALC1(...); CALC2(...); CALC3(...);                                 }
20   STORE(4, reg_3_0, reg_3_1, reg_3_2);                                                  }
21 }                                                                                       }
22 if (!Uppermost_Block) {                                                                }
23   for (i = 9; i <= Stream_End - 3; i += 3) {                                            }
24     LOAD(reg_0_0, i);      CALC1(...); CALC2(...); CALC3(...);                         }
25     STORE(i - 4, reg_3_1, reg_3_2, reg_3_0);                                            }
26     LOAD(reg_0_1, i + 1); CALC1(...); CALC2(...); CALC3(...);                           }
27     STORE(i - 3, reg_3_2, reg_3_0, reg_3_1);                                            }
28     LOAD(reg_0_2, i + 2); CALC1(...); CALC2(...); CALC3(...);                           }
29     STORE(i - 2, reg_3_0, reg_3_1, reg_3_2);                                            }
30   }                                                                                     }
31   if (i == Stream_End) return;                                                          }
32   LOAD(reg_0_0, i); CALC1(...); CALC2(...); CALC3(...);                                 }
33   STORE(i - 4, reg_3_1, reg_3_2, reg_3_0); i++;                                         }
34   if (i == Stream_End) return; /* ... */                                              }
35 } else { /* Inner phase, Tail phase */ }
```

Head phase

Inner phase

Tail phase

- STENCILGEN code consists of calculation of each time-step in one whole loop
- **AN5D unrolls such loops to macro sequences accompanied by the static register allocation**

STENCILGEN CODE

```
Initial_Computation;
for (/* Streaming */) {
  if (Inside_Block_t0) {
    /* Data Shifting */
    b0 = SM0[...]; SM0[...] = t0; t0 = input[...];
  }
  __syncthreads();
  if (Inside_Block_t1) {
    /* Compute */
    float __temp_2__ = b0;
    float __temp_5__ = SM0[...];
    // ...
    float __temp_17__ = t0;
    float __temp_18__ = (__temp_14__ + 5.2f * __temp_17__);
    float __temp_19__ = (__temp_18__ / 118);
    /* Data Shifting */
    b1 = SM1[...]; SM1[...] = t1;  t1 = __temp_19__;
  }
  __syncthreads();
  if (Inside_Block_t2) { /* Compute, Data Shifting */ }
  __syncthreads ();
  if (Inside_Block_t3) { /* Compute, Data Shifting */ }
  __syncthreads ();
  if (Inside_Block_t4) {
    /* Compute */
    float __temp_80__ = b3;
    float __temp_81__ = SM3[...];
    // ...
    float __temp_87__ = t3;
    float __temp_88__ = (__temp_86__ + 5.2f * __temp_87__);
    float __temp_89__ = (__temp_88__ / 118);
    /* Output */
    output[...] = __temp_89__;
  }
}
```

LOAD

CALC1

CALC2

CALC3

STORE

AN5D CODE

```
if (Lowermost_Block) {
  LOAD(reg_3_0, 0); LOAD(reg_0_1, 1); LOAD(reg_0_2, 2);
  CALC1(reg_1_1, reg_3_0, reg_0_1, reg_0_2);
  LOAD(reg_0_0, 3); CALC1(...); CALC2(...);
  LOAD(reg_0_1, 4); CALC1(...); CALC2(...); CALC3(...);
  LOAD(reg_0_2, 5); CALC1(...); CALC2(...); CALC3(...);
  STORE(3, reg_3_0, reg_3_1, reg_3_2);
  LOAD(reg_0_0, 6); /* ... */
  STORE(4, reg_3_0, reg_3_1, reg_3_2);
}
else {
  LOAD(reg_0_0, 0); LOAD(reg_0_1, 1); LOAD(reg_0_2, 2);
  CALC1(reg_1_1, reg_0_0, reg_0_1, reg_0_2);
  LOAD(reg_0_0, 3); CALC1(...);
  LOAD(reg_0_1, 4); CALC1(...); CALC2(...);
  LOAD(reg_0_2, 5); CALC1(...); CALC2(...);
  LOAD(reg_0_0, 6); CALC1(...); CALC2(...); CALC3(...);
  LOAD(reg_0_1, 7); CALC1(...); CALC2(...); CALC3(...);
  LOAD(reg_0_2, 8); CALC1(...); CALC2(...); CALC3(...);
  STORE(4, reg_3_0, reg_3_1, reg_3_2);
}
if (!Uppermost_Block) {
  for (i = 9; i <= Stream_End - 3; i += 3) {
    LOAD(reg_0_0, i);     CALC1(...); CALC2(...); CALC3(...);
    STORE(i - 4, reg_3_1, reg_3_2, reg_3_0);
    LOAD(reg_0_1, i + 1); CALC1(...); CALC2(...); CALC3(...);
    STORE(i - 3, reg_3_2, reg_3_0, reg_3_1);
    LOAD(reg_0_2, i + 2); CALC1(...); CALC2(...); CALC3(...);
    STORE(i - 2, reg_3_0, reg_3_1, reg_3_2);
  }
  if (i == Stream_End) return;
  LOAD(reg_0_0, i); CALC1(...); CALC2(...); CALC3(...);
  STORE(i - 4, reg_3_1, reg_3_2, reg_3_0); i++;
  if (i == Stream_End) return; /* ... */
} else { /* Inner phase, Tail phase */ }
```

Head phase

Inner phase

Tail phase

- STENCILGEN code consists of calculation of each time-step in one whole loop
- **AN5D unrolls such loops to macro sequences accompanied by the static register allocation**

STENCILGEN CODE

```
Initial_Computation;
for (/* Streaming */) {
  if (Inside_Block_t0) {
    /* Data Shifting */
    b0 = SM0[...]; SM0[...] = t0; t0 = input[...];
  }
  __syncthreads();
  if (Inside_Block_t1) {
    /* Compute */
    float __temp_2__ = b0;
    float __temp_5__ = SM0[...];
    // ...
  }
  __syncthreads ();
  if (Inside_Block_t3) { /* Compute, Data Shifting */ }
  __syncthreads ();
  if (Inside_Block_t4) {
    /* Compute */
    float __temp_80__ = b3;
    float __temp_81__ = SM3[...];
    // ...
    float __temp_87__ = t3;
    float __temp_88__ = (__temp_86__ + 5.2f * __temp_87__);
    float __temp_89__ = (__temp_88__ / 118);
    /* Output */
    output[...] = __temp_89__;
  }
}
```

LOAD

CALC1

CALC3

STORE

Only loop that should be kept for instruction fetch latency

AN5D CODE

```
if (Lowermost_Block) {
  LOAD(reg_3_0, 0); LOAD(reg_0_1, 1); LOAD(reg_0_2, 2);
  CALC1(reg_1_1, reg_3_0, reg_0_1, reg_0_2);
  LOAD(reg_0_0, 3); CALC1(...); CALC2(...);
  LOAD(reg_0_1, 4); CALC1(...); CALC2(...); CALC3(...);
  LOAD(reg_0_2, 5); CALC1(...); CALC2(...); CALC3(...);
  STORE(1, reg_3_0, reg_3_1, reg_3_2);
  LOAD(reg_0_0, 6); /* ... */
  STORE(4, reg_3_0, reg_3_1, reg_3_2);
}
else {
  LOAD(reg_0_0, 0); LOAD(reg_0_1, 1); LOAD(reg_0_2, 2);
  _1_1, reg_0_0, reg_0_1, reg_0_2);
  0_0, 3); CALC1(...);
  0_1, 4); CALC1(...); CALC2(...);
  0_2, 5); CALC1(...); CALC2(...);
  0_0, 6); CALC1(...); CALC2(...); CALC3(...);
  0_1, 7); CALC1(...); CALC2(...); CALC3(...);
  0_2, 8); CALC1(...); CALC2(...); CALC3(...);
  reg_3_0, reg_3_1, reg_3_2);
}
if (!Uppermost_Block) {
  for (i = 9; i <= Stream_End - 3; i += 3) {
    LOAD(reg_0_0, i);     CALC1(...); CALC2(...); CALC3(...);
    STORE(i - 4, reg_3_1, reg_3_2, reg_3_0);
    LOAD(reg_0_1, i + 1); CALC1(...); CALC2(...); CALC3(...);
    STORE(i - 3, reg_3_2, reg_3_0, reg_3_1);
    LOAD(reg_0_2, i + 2); CALC1(...); CALC2(...); CALC3(...);
    STORE(i - 2, reg_3_0, reg_3_1, reg_3_2);
  }
  if (i == Stream_End) return;
  LOAD(reg_0_0, i); CALC1(...); CALC2(...); CALC3(...);
  STORE(i - 4, reg_3_1, reg_3_2, reg_3_0); i++;
  if (i == Stream_End) return; /* ... */
} else { /* Inner phase, Tail phase */ }
```

Head phase

Inner phase

Tail phase

- STENCILGEN code consists of calculation of each time-step in one whole loop
- **AN5D unrolls such loops to macro sequences accompanied by the static register allocation**

Other On-Chip Resource Optimizations

- Disabling vectorized shared memory accesses
  - Access through a device function that wraps SM accesses
  - Performance increase on high-order stencils w/ less register use

```
__device__ float __smref_wrap(float *sm, size_t index) { return sm[index]; }
```

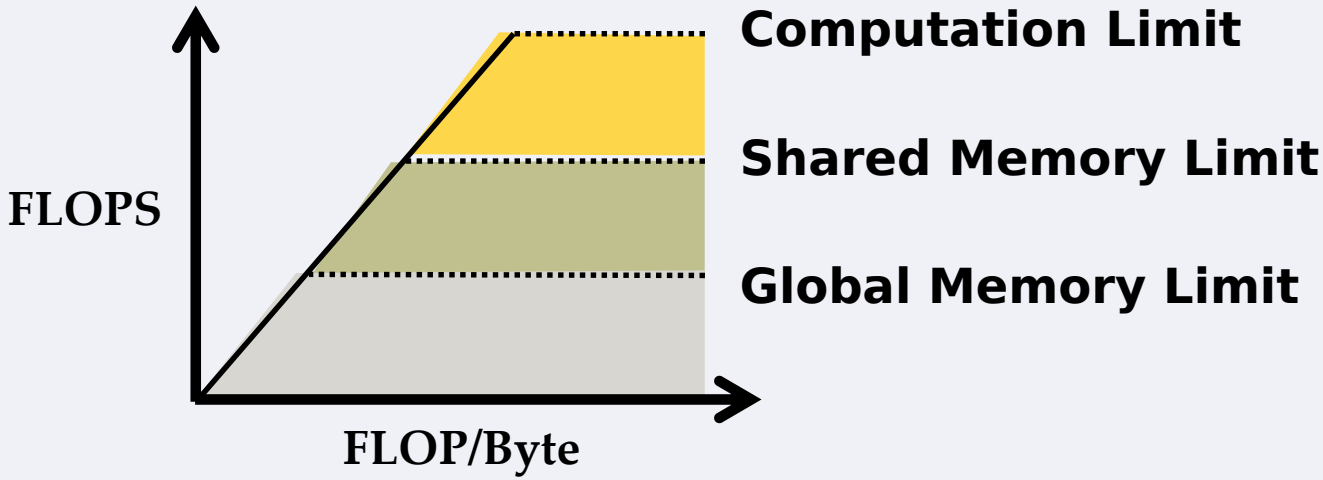
- Constant substitution to double buffered SM
  - Inserting the explicit address for compiler optimization

```
__a_sm = __a_sm_base + blockSize * N;
```

- Integrated with techniques proposed in previous work
  - Diagonal-access free optimization
  - Associative stencil optimization (Partial summation)
  - Streaming division

Performance Tuning

ROOFLINE MODEL

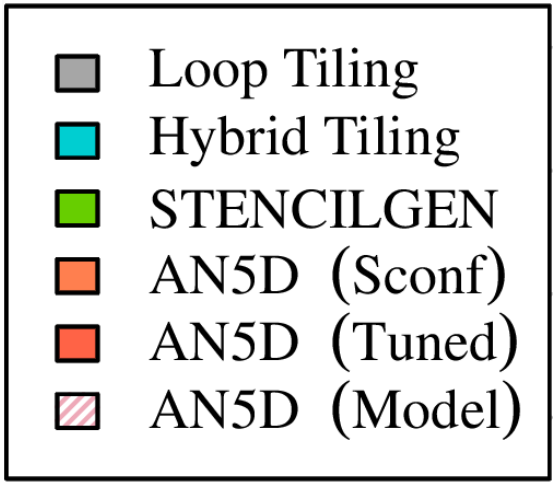


$$time_{\text{model}} = \max\left(time_{\text{comp}}, time_{\text{sm}}, time_{\text{gm}}\right) / \ eff$$

- Select the top-5 parameters under the performance prediction by the roofline model
  - Parameter: streaming length, temporal/spatial block size
  - Tuning the compiler option --maxrregcount=(none/32/64/96)

Evaluation – Methodology

- AN5D is implemented on PPCG *(Polyhedral Parallel Code Generator)*
- Evaluation on NVIDIA Tesla **V100 SMX2** / P100 SMX2 (**float** / double)
  - A wide range of synthetic/general 2D/3D stencils (size $16{,}384^2/512^3$)
- Performance Comparison to Loop Tiling / Hybrid / STENCILGEN



Hybrid Tiling: Parameter tuned by brute-force from 10/5K configs (2D/3D)

STENCILGEN: Code available on the repository (fixed parameters)

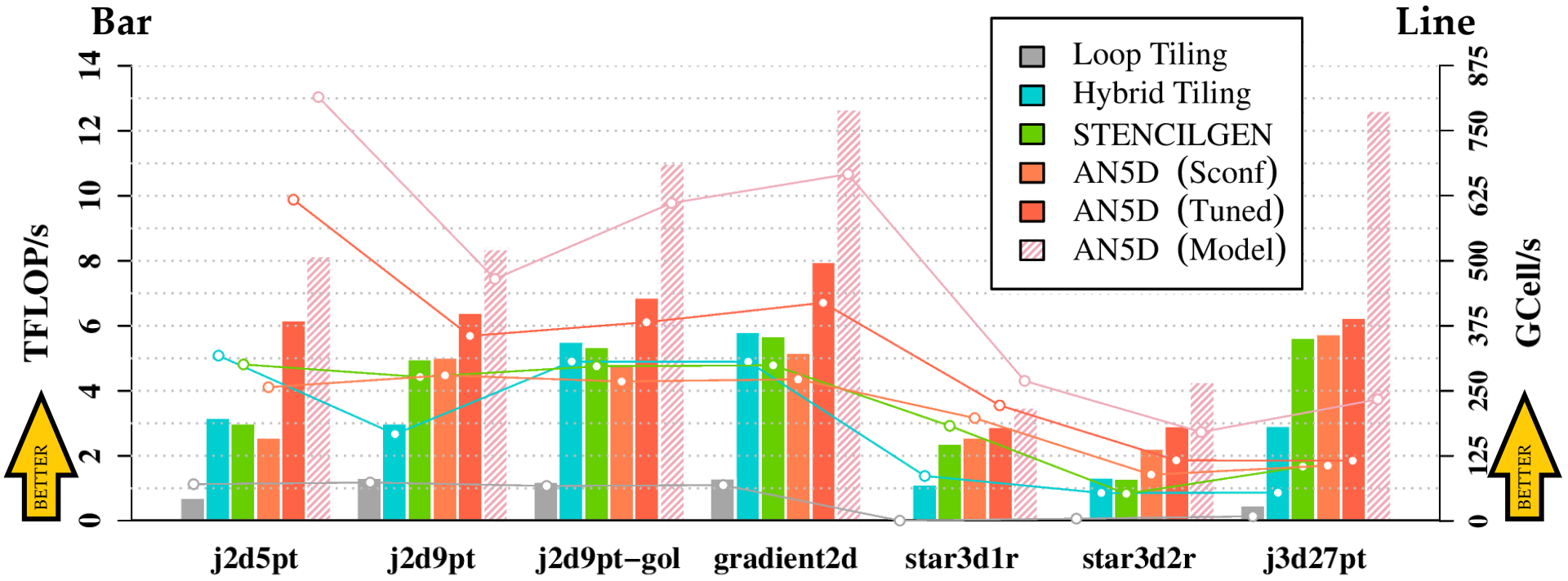
AN5D (**Sconf**): Configured with the same parameters as STENCILGEN

AN5D (**Tuned**): Parameter tuned with the roofline model

AN5D (**Model**): Predicted performance by the roofline model

∗ **Tuned** parameters (including high-degree temporal blocking) are only applicable to AN5D because of on-chip resource restriction

Evaluation – Performance Comparison

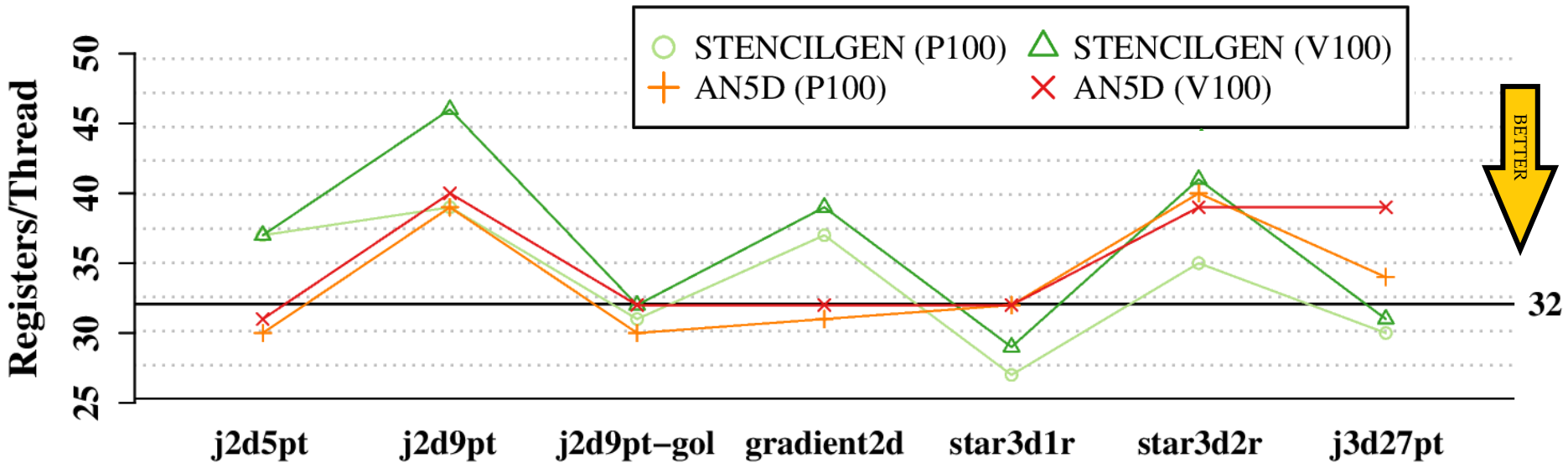


Speed up on average :

- STENCILGEN → AN5D (**Sconf**): 1.07x
- Hybrid Tiling → AN5D (**Tuned**): 1.94x
- AN5D (**Sconf**) → AN5D (**Tuned**): 1.45x

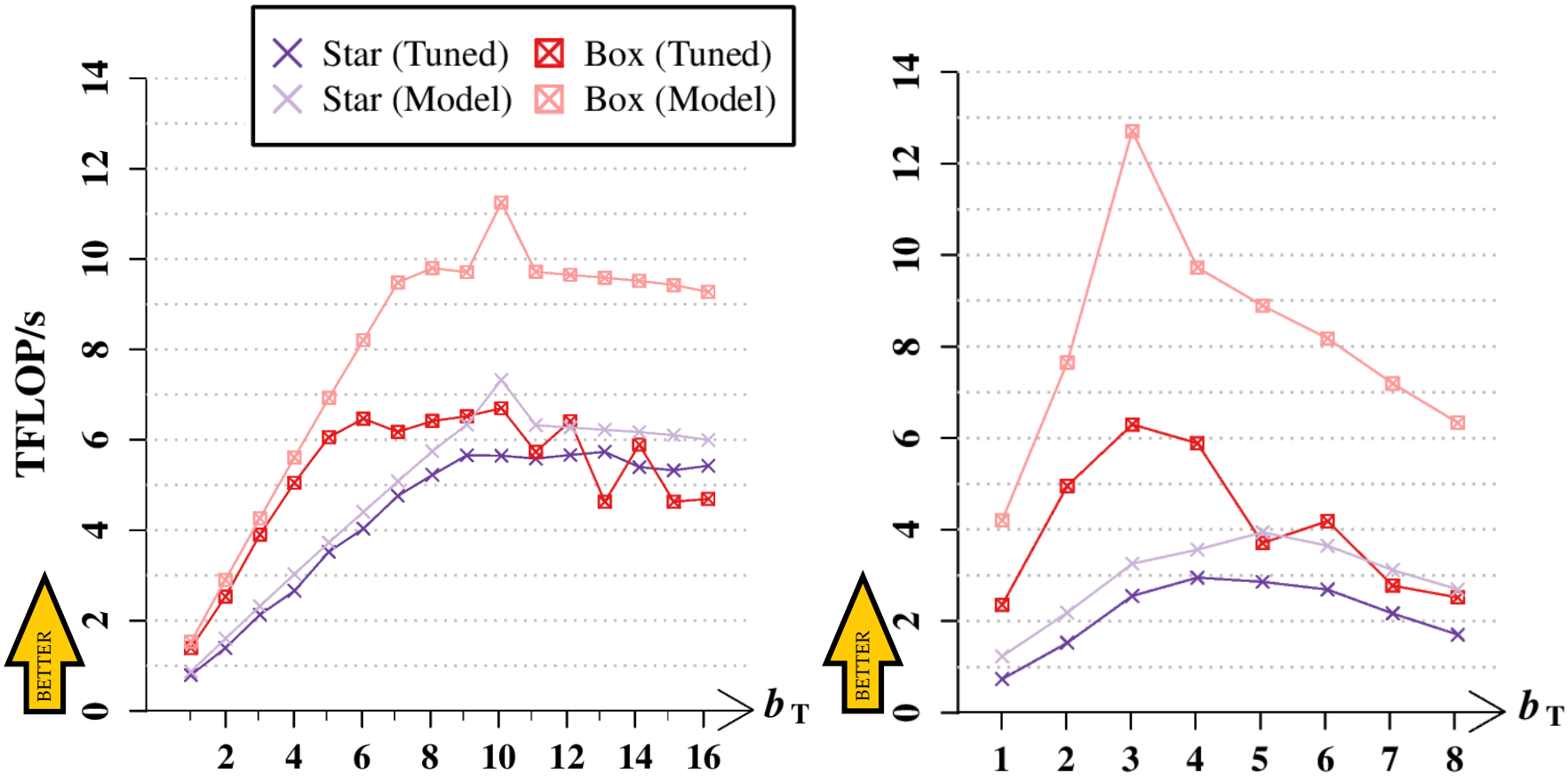
- Our performance model predicted all cases as shared memory bound
- Performance degradation on Model → Tuned caused by low efficiency of shared memory

Evaluation – Register Use



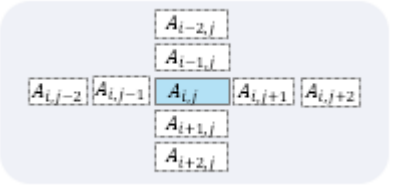
- AN5D originally necessitates b_T extra registers per thread for sub-plane management
  - In practice, AN5D uses fewer registers compared STENCILGEN on average
  - When we limit register usage to 32 (max to achieve 100% processor occupancy), no register spilling caused despite STENCILGEN does on j2d9pt and star3d2r

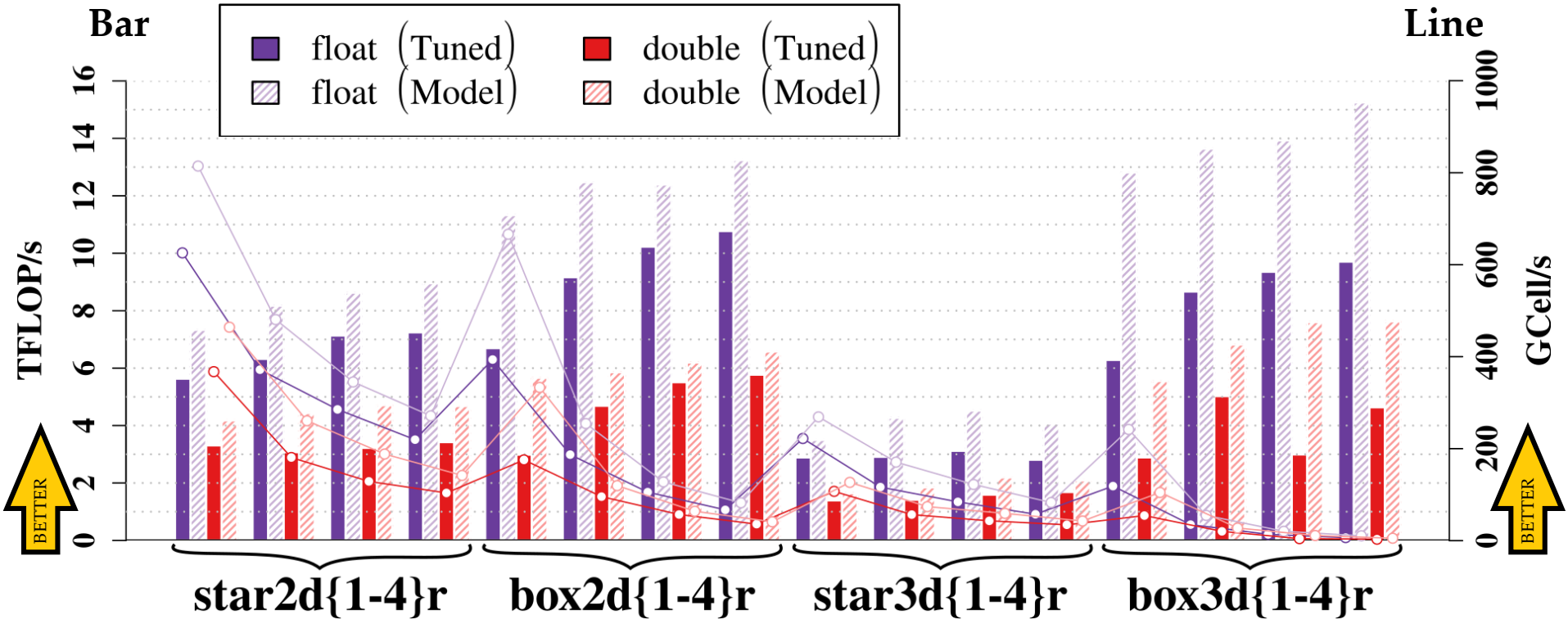
Evaluation – Scaling with TB Degree



- 2D stencils scale up to $b_T = 10$
- 3D stencils scale up to $b_T = 5$ for star-shaped stencils and $b_T = 3$ for box-shaped

Evaluation – High-Order Stencils





- Best perf of 1st order gained with high-degree temporal blocking (2D: 8~15, 3D: 3~5)
- Most cases including fourth-order stencils had the best performance with $b_T \geq 2$
- High-order 3d boxes had $b_T = 1$, but still achieved around 60% of peak perf on V100

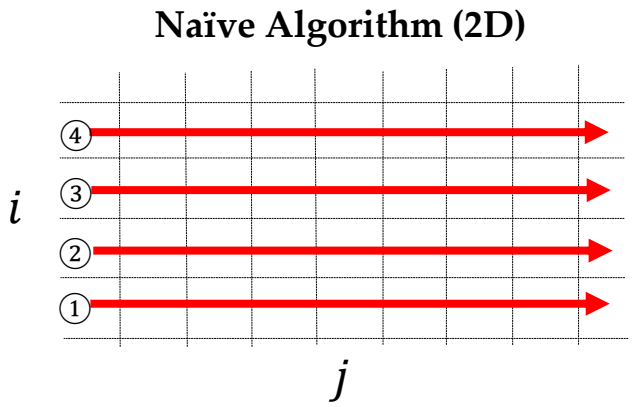
Conclusion

- We built an automated N.5D framework with integration of multiple on-chip resource optimization
  - For the first time, N.5D blocking scaled up to $b_T = 10$ on 2D stencils
  - High computational efficiency even on 3D & high-order stencils
- Quick tuning by the performance model
- Achieved the highest performance for all evaluated stencil benchmarks on the state-of-the-art Tesla V100

Tokyo Tech

Back up

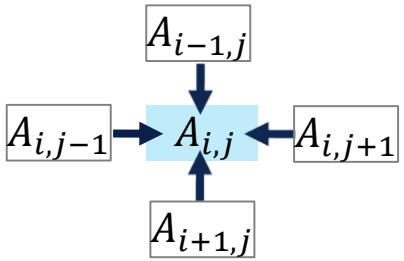
Naïve Execution of Stencil Computation



- Sequential execution in indexical order
- Abandoning the parallelism between cells
- Not utilizing the parallelism on GPU

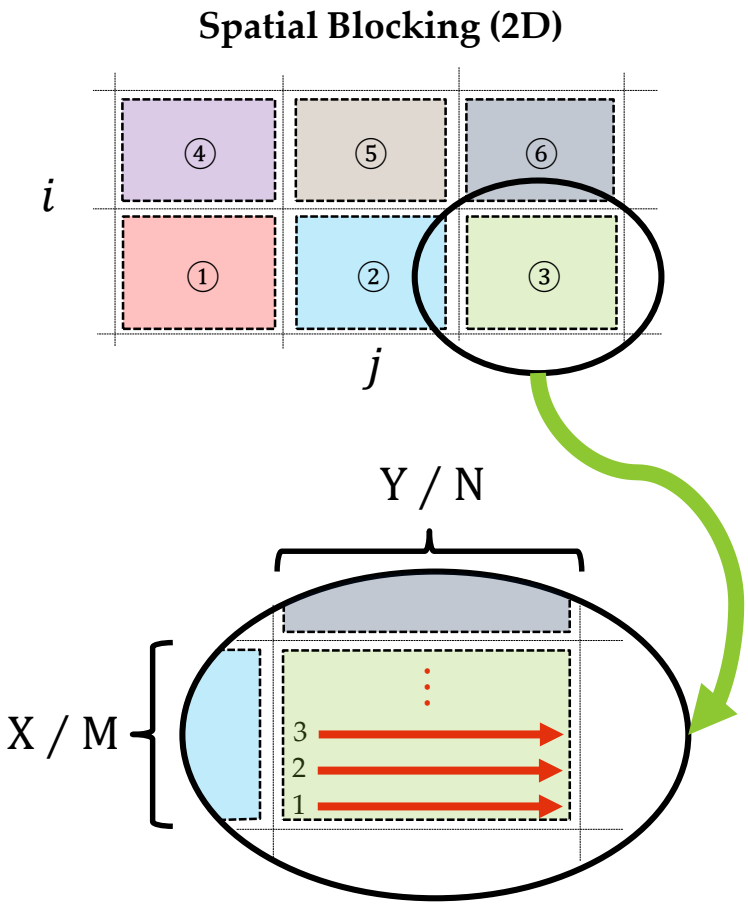
Optimization →

- Spatial Blocking
- Temporal Blocking
  - Overlapped Tiling
  - Hexagonal Tiling



Update of Jacobi2D

Spatial Blocking



- Divided blocks are executed in parallel on GPU
- The number of memory accesses is reduced as spatial locality is increased
- Local memory is available for explicit data reuse (Shared Memory, Register)

(X, Y) : Grid Size
(M, N): Number of Division

V100 SXM2	Single-Precision Performance:	15.7 TFLOP/s
	Streaming Multiprocessors Per Chip:	80
	Global-Memory Size:	16 GiB
	Shared-Memory Size Per SM:	96 KiB
	L2 Cache Size:	6144 KiB
	Register File Size Per SM:	256 KiB
	Measured Bandwidth (Global Memory):	750 GiB/s
	Measured Bandwidth (Shared Memory):	12,080 GiB/s
P100 SXM2	Single-Precision Performance:	10.6 TFLOP/s
	Streaming Multiprocessors Per Chip:	56
	Global-Memory Size:	16 GiB
	Shared-Memory Size Per SM:	64 KiB
	L2 Cache Size:	4096 KiB
	Register File Size Per SM:	256 KiB
	Measured Bandwidth (Global Memory):	510 GiB/s
	Measured Bandwidth (Shared Memory):	7,763 GiB/s
CPU	V100: Intel Xeon Gold 6148 @ 2.4GHz	× 2
	P100: Intel Xeon E5-2630 v4 @ 2.20GHz	× 1

Table 3. Benchmarks

Stencil	Computation	FLOP/Cell
star2d{x}r x ∈ [1, 4]	$c_{(x,y)}f_{(x,y)} + \sum_{i=-x,\, i\neq 0}^{x} (c_{(x+i,y)}f_{(x+i,y)} + c_{(x,y+i)}f_{(x,y+i)})$	8x + 1
box2d{x}r x ∈ [1, 4]	$\sum_{i=-x}^{x}\sum_{j=-x}^{x} c_{(x+i,y+j)}f_{(x+i,y+j)}$	$2\times$ $(2x+1)^2 - 1$
j2d5pt	$(c_{(x,y)}f_{(x,y)} + \sum_{i=-1,\, i\neq 0}^{1} (c_{(x+i,y)}f_{(x+i,y)} + c_{(x,y+i)}f_{(x,y+i)}))/c_0$	10
j2d9pt	$(c_{(x,y)}f_{(x,y)} + \sum_{i=-2,\, i\neq 0}^{2} (c_{(x+i,y)}f_{(x+i,y)} + c_{(x,y+i)}f_{(x,y+i)}))/c_0$	18
j2d9pt-gol	$(\sum_{i=-1}^{1}\sum_{j=-1}^{1} c_{(x+i,y+j)}f_{(x+i,y+j)})/c_0$	18
gradient2d	$c_{(x,y)}f_{(x,y)} + 1.0/\mathrm{sqrt}(c_0 + \sum_{i=-1,\, i\neq 0}^{1} ((f_{(x,y)} - f_{(x+i,y)}) \times (f_{(x,y)} - f_{(x+i,y)}) + (f_{(x,y)} - f_{(x,y+i)}) \times (f_{(x,y)} - f_{(x,y+i)})))$	19
star3d{x}r x ∈ [1, 4]	$c_{(x,y)}f_{(x,y)} + \sum_{i=-x,\, i\neq 0}^{x} (c_{(x+i,y,z)}f_{(x+i,y,z)} + c_{(x,y+i,z)}f_{(x,y+i,z)} + c_{(x,y,z+i)}f_{(x,y,z+i)})$	12x + 1
box3d{x}r x ∈ [1, 4]	$\sum_{i=-x}^{x}\sum_{j=-x}^{x}\sum_{k=-x}^{x} c_{(x+i,y+j,z+k)}\, f_{(x+i,y+j,z+k)}$	$2\times$ $(2x+1)^3 - 1$
j3d27pt	$(\sum_{i=-1}^{1}\sum_{j=-1}^{1}\sum_{k=-1}^{1} c_{(x+i,y+j,z+k)} f_{(x+i,y+j,z+k)})/c_0$	54