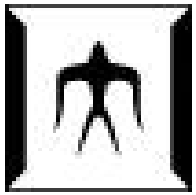
Applying Recursive Temporal Blocking for Stencil Computations to Deeper Memory Hierarchy

Toshio Endo (遠藤敏夫)

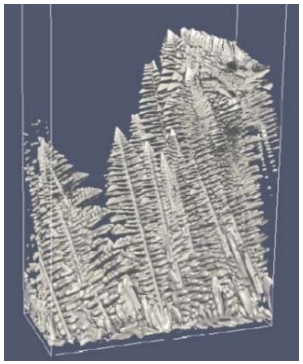
GSIC, Tokyo Institute of Technology (東京工業大学)



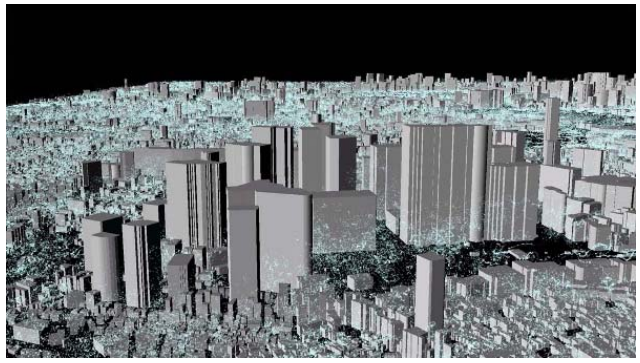
Stencil Computations

Important kernels for various simulations: fluid dynamics, material…

ASUCA weather simulator

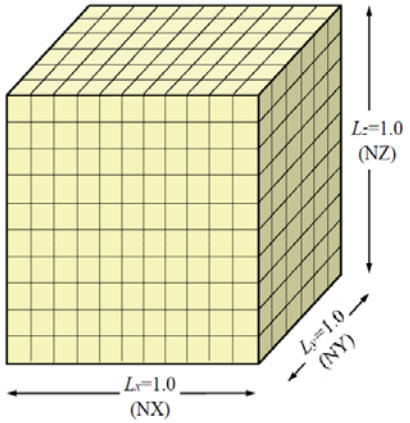


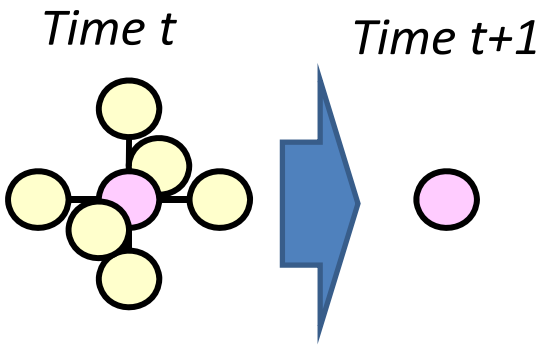
Phase-Field computation (2011 Gordon Bell)



Air flow simulation

=

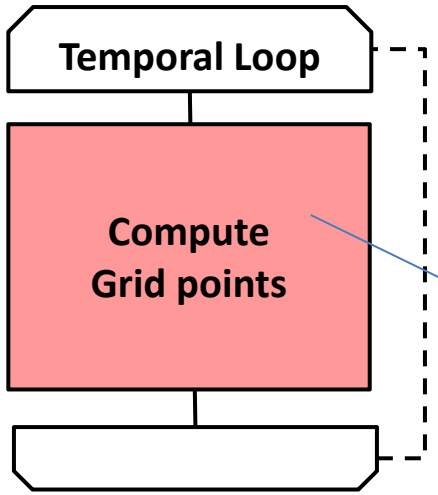


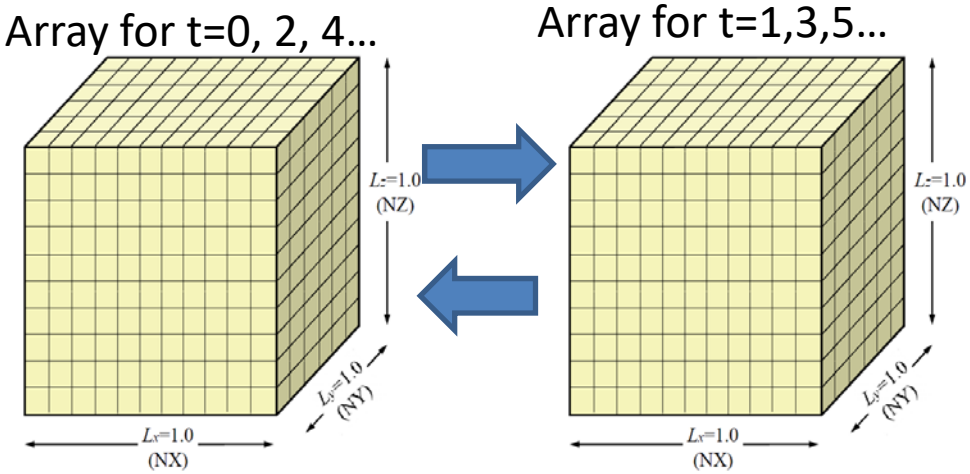


Stencil computations are "memory intensive" →

GPUs computing fits well with 500~1000 GB/s memory BW!

A Simple Stencil Algorithm





In CPU implementation,
spatial (x-, y-, z-) loops are here

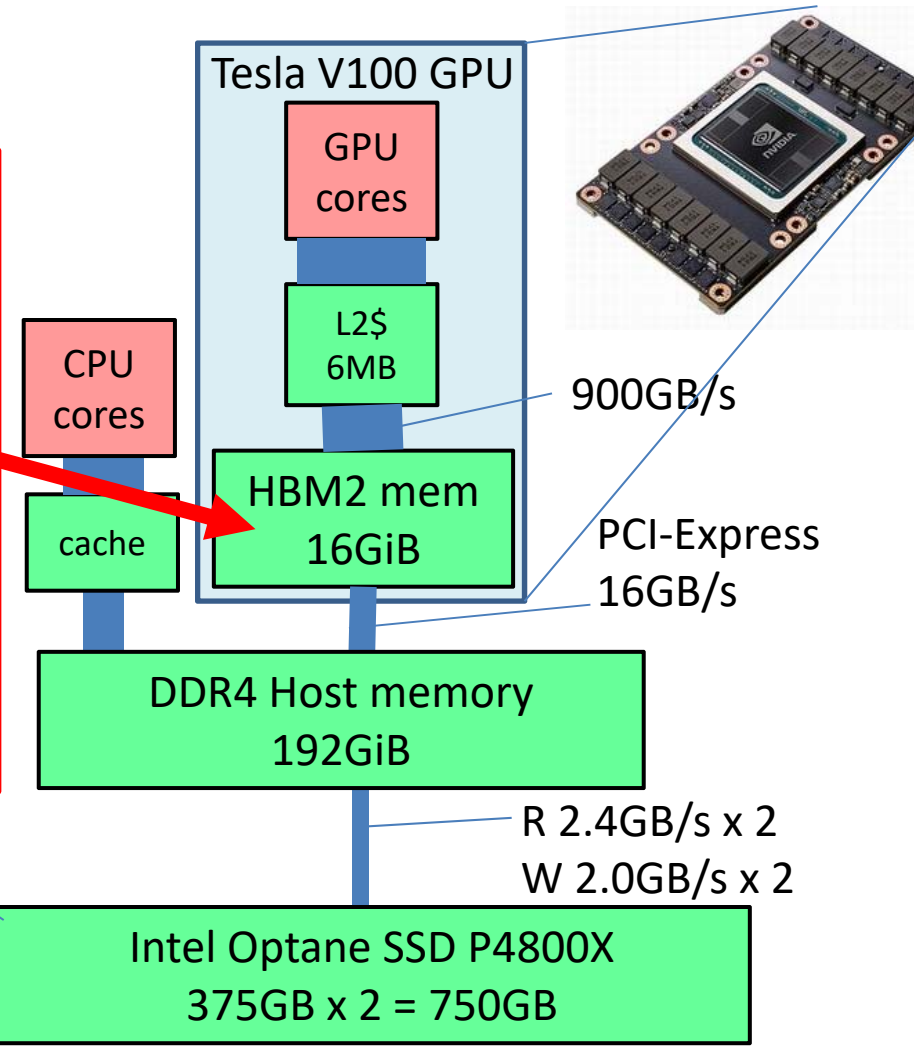
In GPU implementation (our context),
millions threads compute grid points

Memory Hierarchy of GPU Machines and Issues

GPUs are good in speed,
but NOT in SIZE!

In simple implementations on GPUs,
domain sizes are configured as
< (aggregated) GPU memory

➔ Prohibits accurate simulation



Using multiple GPUs is a solution

- But we are still limited by “GPU memory capacity ×#GPUs”
- Larger capacity of lower memory hierarchy is not utilized

How about “Out-of-Core” Execution?

It looks promising to combine

- High-speed of GPUs and
- Large capacity of DDR/SSDs

→Out-of-core execution

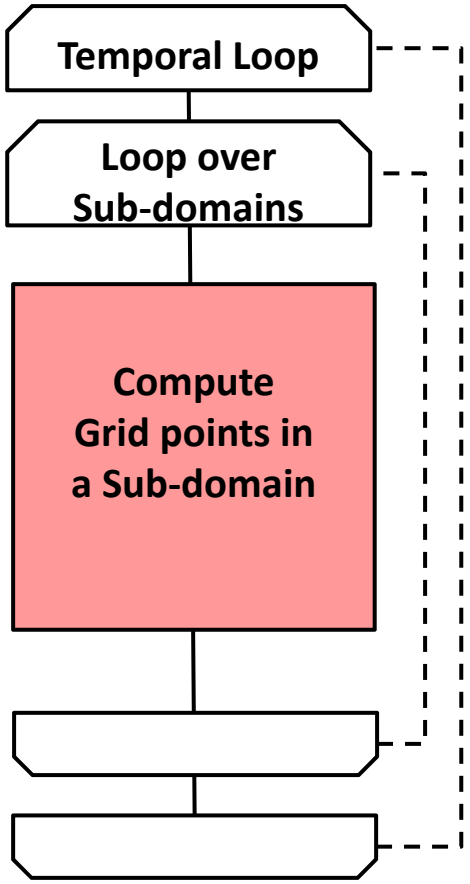
In stencil, we divide the domain into sub-domains

But…

- 8GB domain (< GPU mem) → 31GUP/s
- 64GB domain (< DDR) → 0.36GUP/s
- 256GB domain (< SSD) → 0.24GUP/s

Only 1% speed is TOO SLOW!

(GUP/s: giga updated points per second)



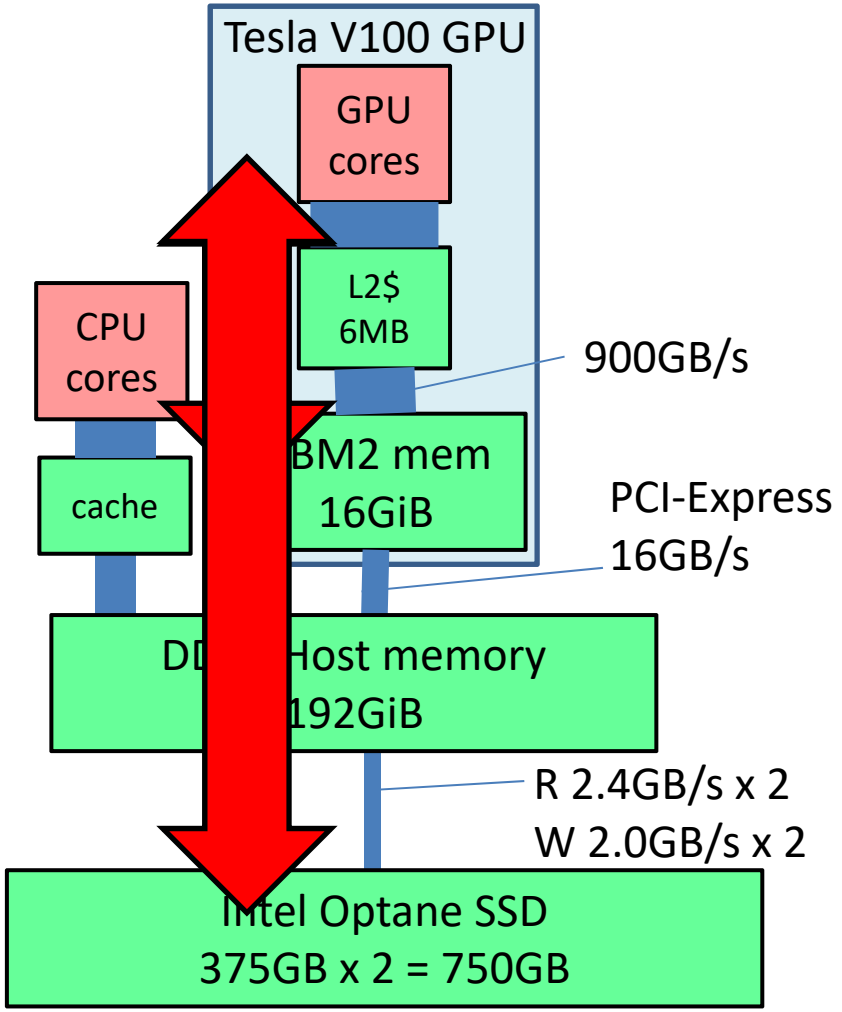
Data movement among memory layers are omitted

Why Out-of-Core Execution is So Slow

- In stencil computations, points the entire domain are scanned every time step

→ Bad access locality

Speed of out-of-core execution is limited by bandwidth of PCI-Express or SSD ☹



Objective

- To achieve high-speed and big stencil computations
  - Hardware: GPU + Optane 3D-XPoint SSD
    - Optane is used to expand memory capacity
    - Non-volatility is not used
  - Middleware: Intel Memory Drive Technology (IMDT)
  - Algorithm: Stencil + Recursive temporal blocking technique

Temporal Blocking

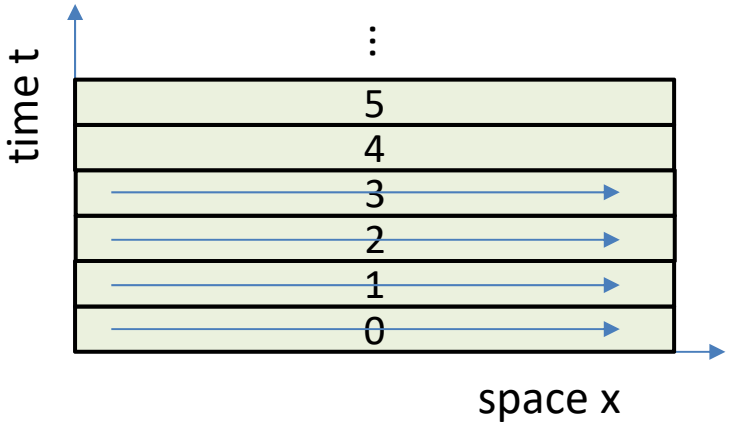
- Simple stencil implementation has bad access locality
  - Spatial loop in temporal loop

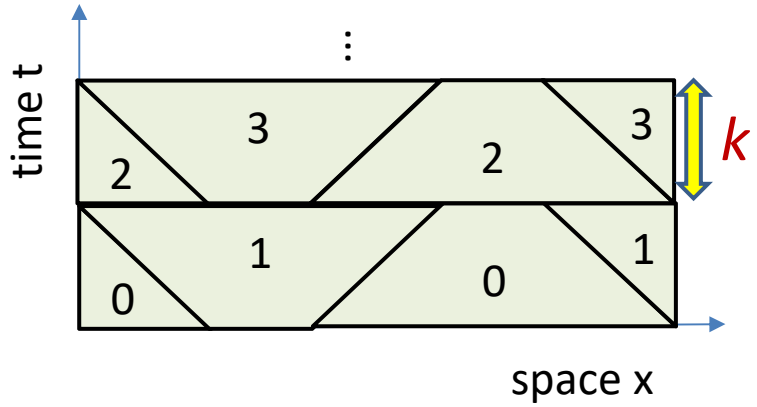
- With temporal blocking, a smaller domain is computed for multiple (*k*) steps at once

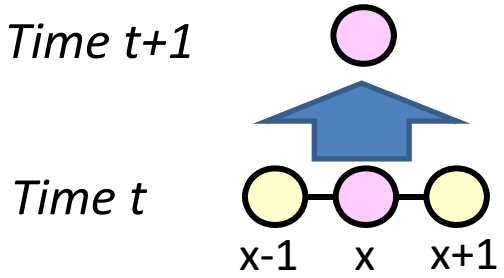
[Wolf 91] [Wonnacott 00] [Datta 08]…
[Endo 14, 16]

→Better access locality!

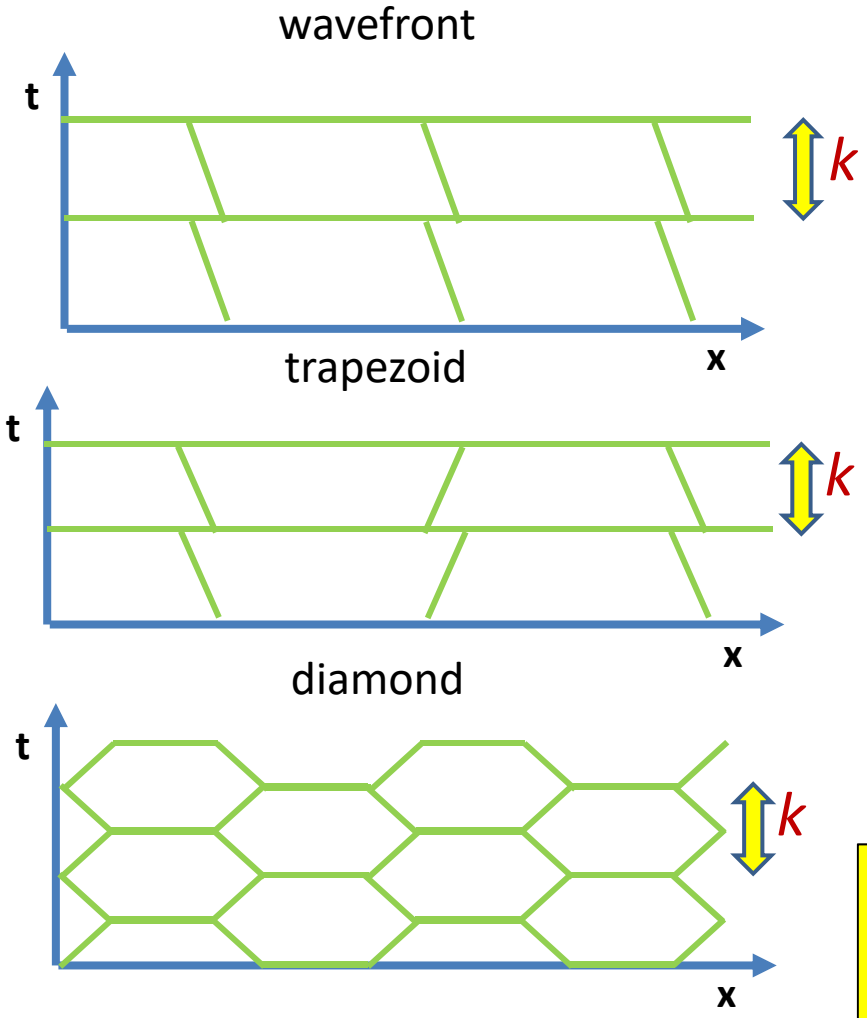
Note: Block shape is not "rectangle" for data dependency

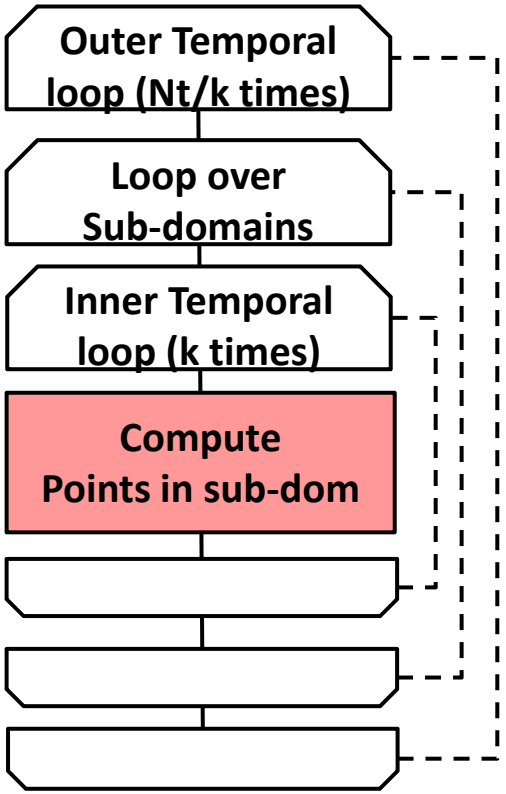






Several Temporal Blocking Methods





All of them are using a single blocking factor *k*

→ Not best for multiple memory layers

→ "Recursive" approach works better

Recursive Temporal Blocking

- Frigo has proposed recursive temporal blocking [Frigo et al. ICS 05]
  - Objective is to harness multiple cache layers
  - Programmers do not have to consider each layer explicitly ☺
    - Only parameter to be configured is a threshold *th* to stop recursion

[Q] Is it effective on multiple memory layers including NVMe SSDs?

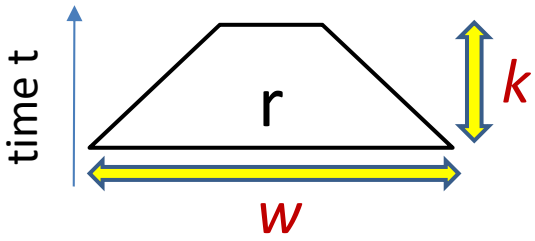
Recursive Temporal Blocking Algorithm

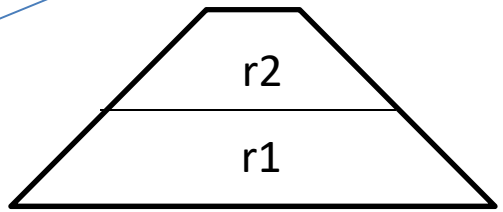
(Slightly modified from Frigo's)

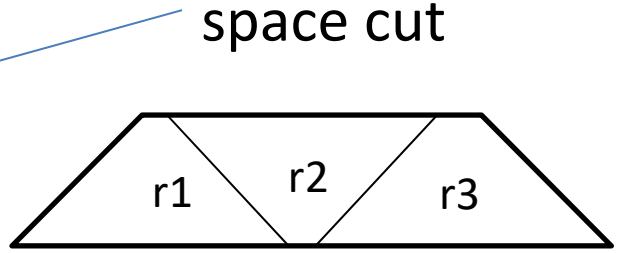
r: Region to be computed (space x time)

th: Threshold

```
comp(r) {
  k = height of r;
  w = width of r;
  if (w < th) {
    Compute r on GPU;
  }
  else if (w < 4*k) {
    (r1, r2) = timecut(r);
    comp(r1); comp(r2);
  }
  else {
    (r1,r2,r3) = spacecut(r);
    comp(r1); comp(r3);
    comp(r2);
  }}
```





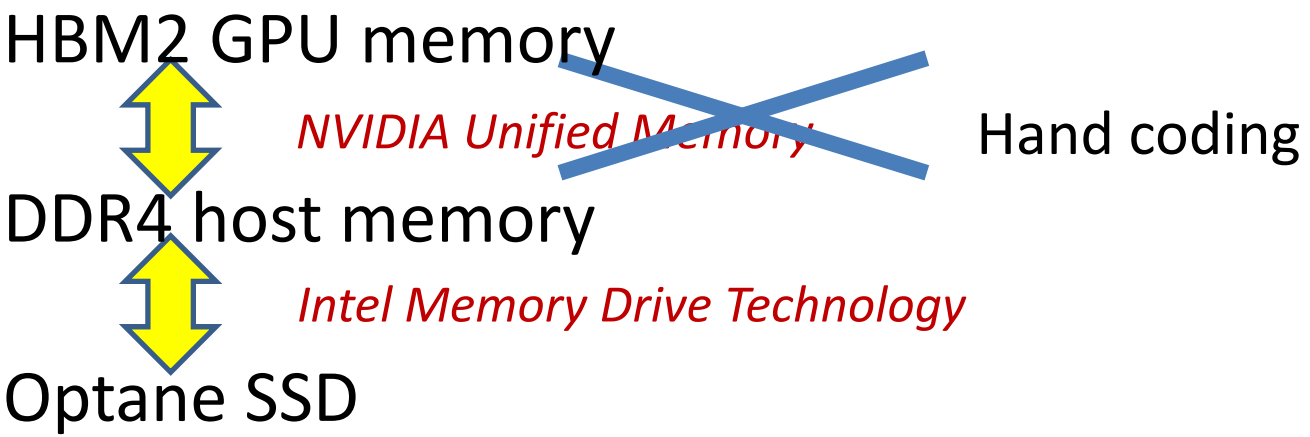


Implementation

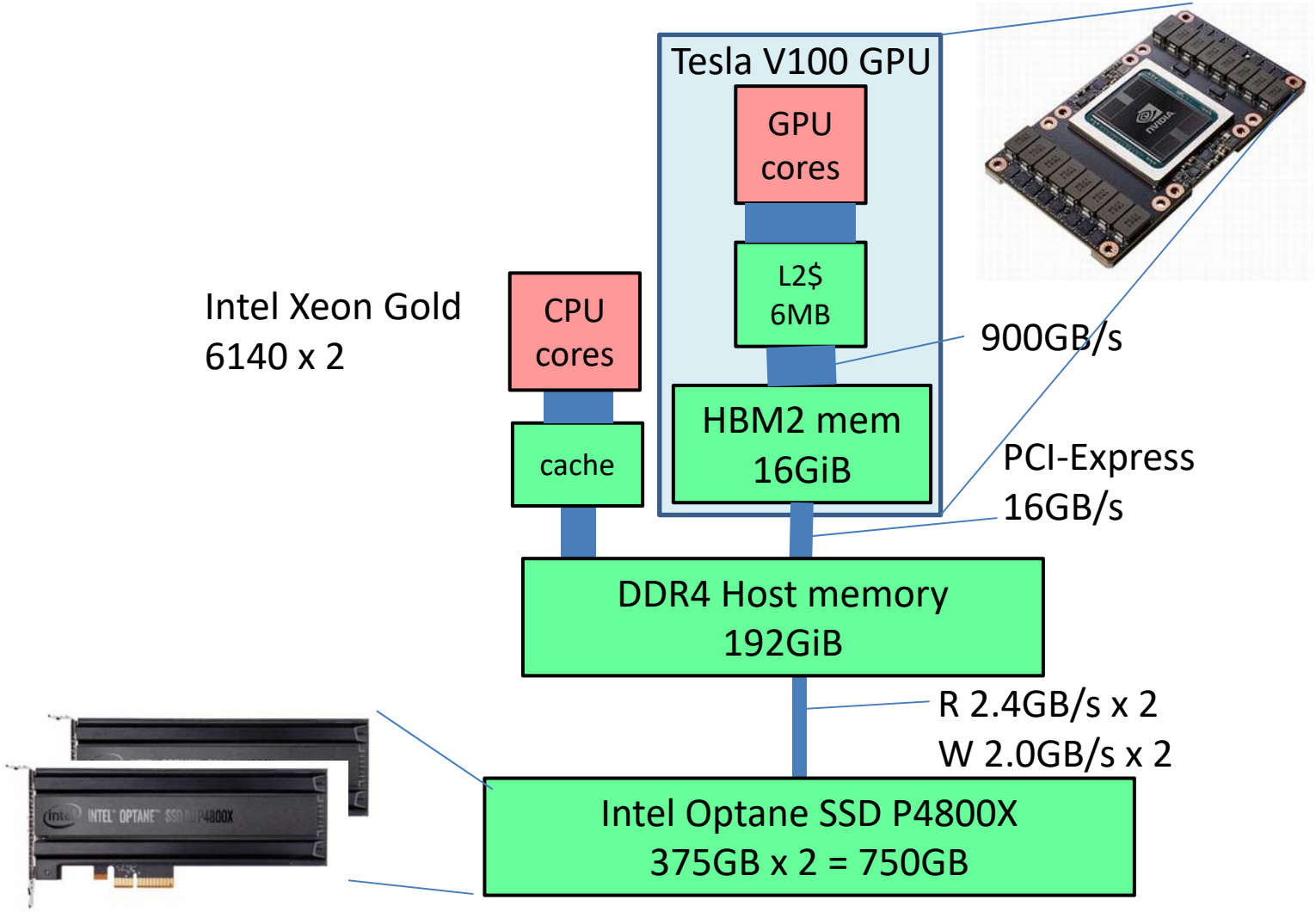
A simple 3D-7point stencil has been implemented

Domain region is divided only in z-dimension

- Leaf computation on a GPU: NVIDIA CUDA is used
- Memory movement among memory layers
  - Automatic movement is better for programmability



Experimental Environment

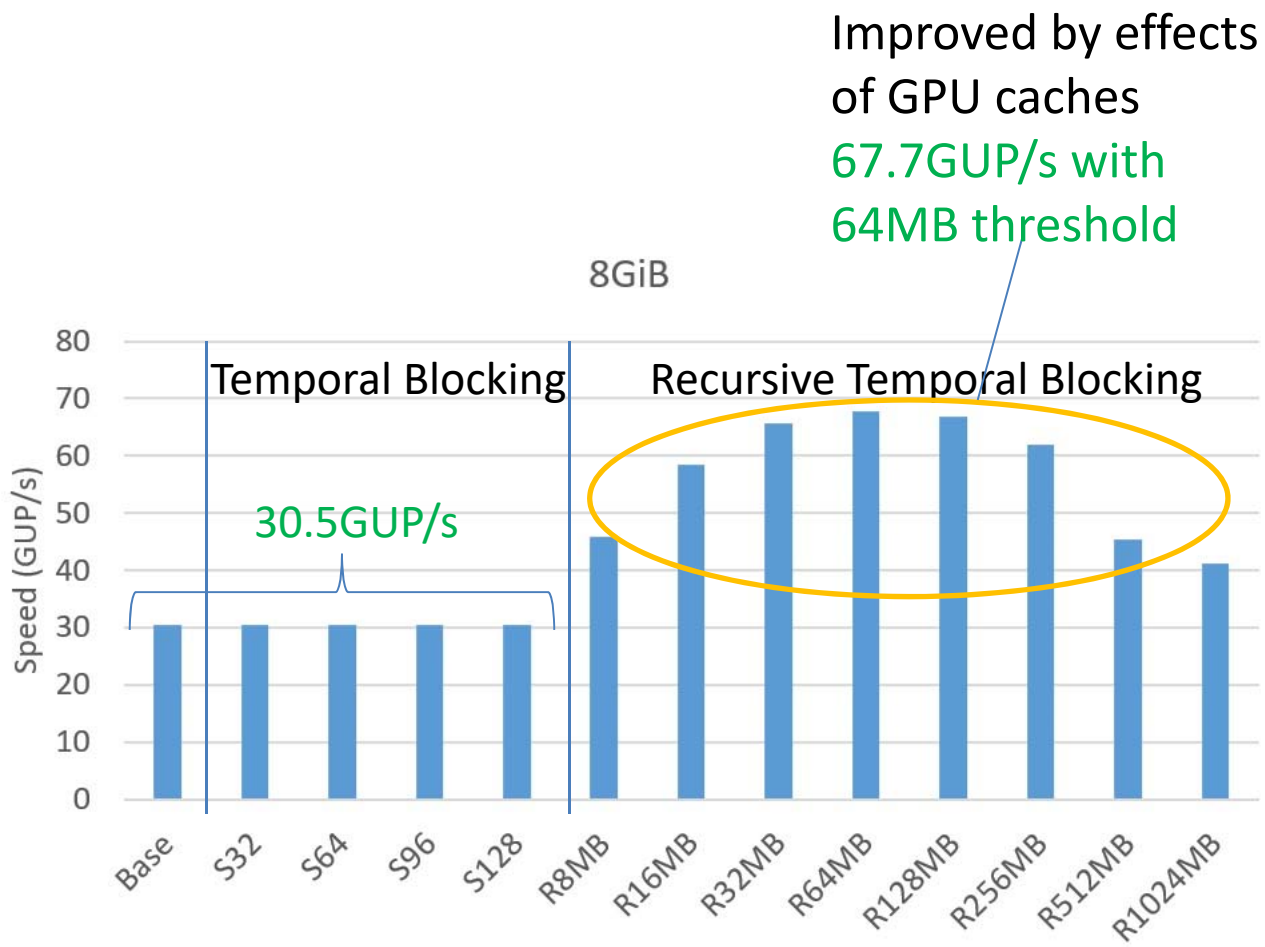


Experimental Conditions

- Domain sizes
  - 8GB, 64GB, 256GB
- The followings are compared
  - Base: Base implementation
  - Sxxx: With temporal blocking with single k
    - xxx is temporal block size k
    - S32, S64, S96, S128
  - Rxxx: With recursive temporal blocking
    - xxx is a threshold to stop recursive calls
    - R8MB to R1024MB

Results: 8GB Domain

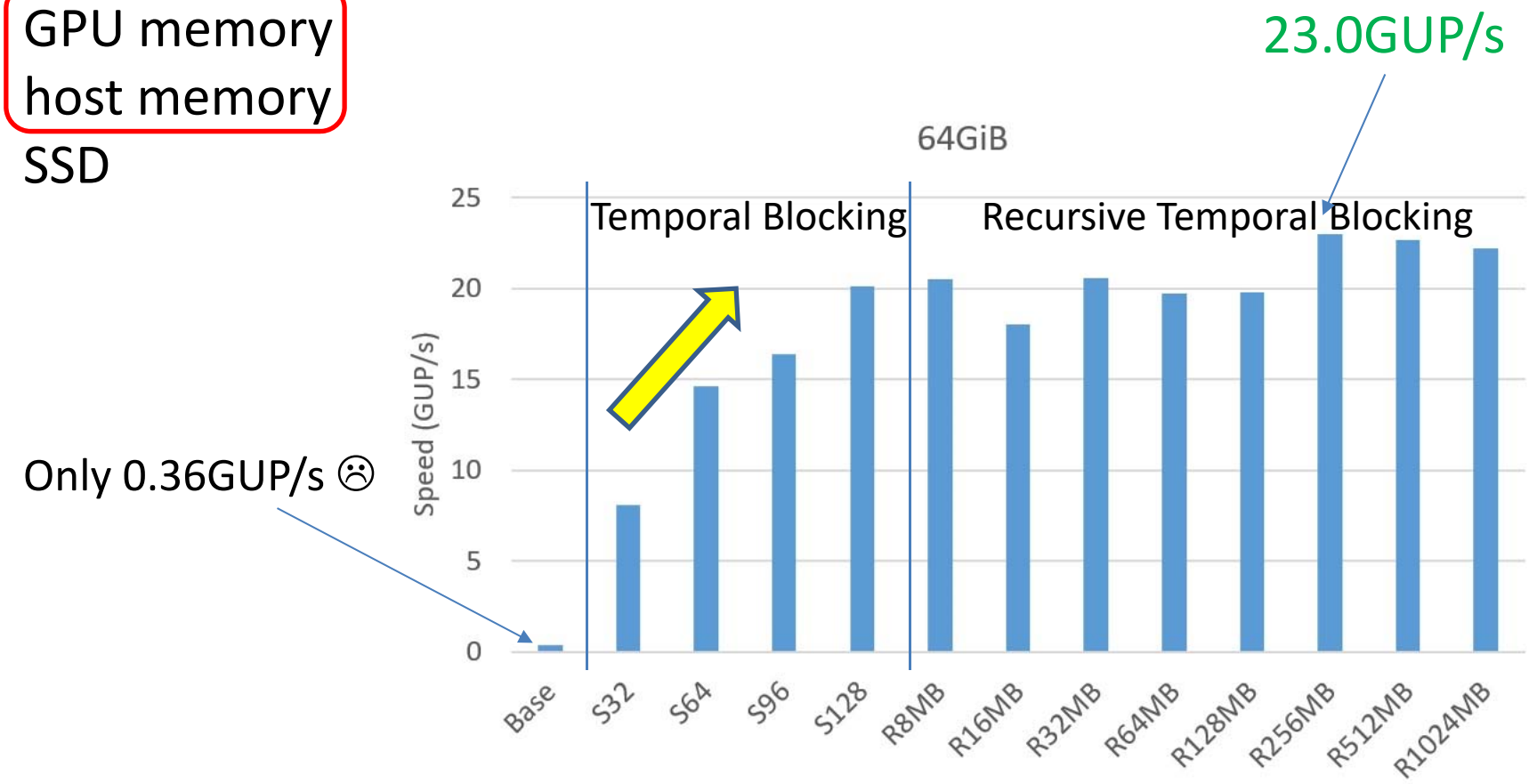
GPU memory
host memory
SSD



GUP/s: giga updated points per second

Results: 64GB Domain

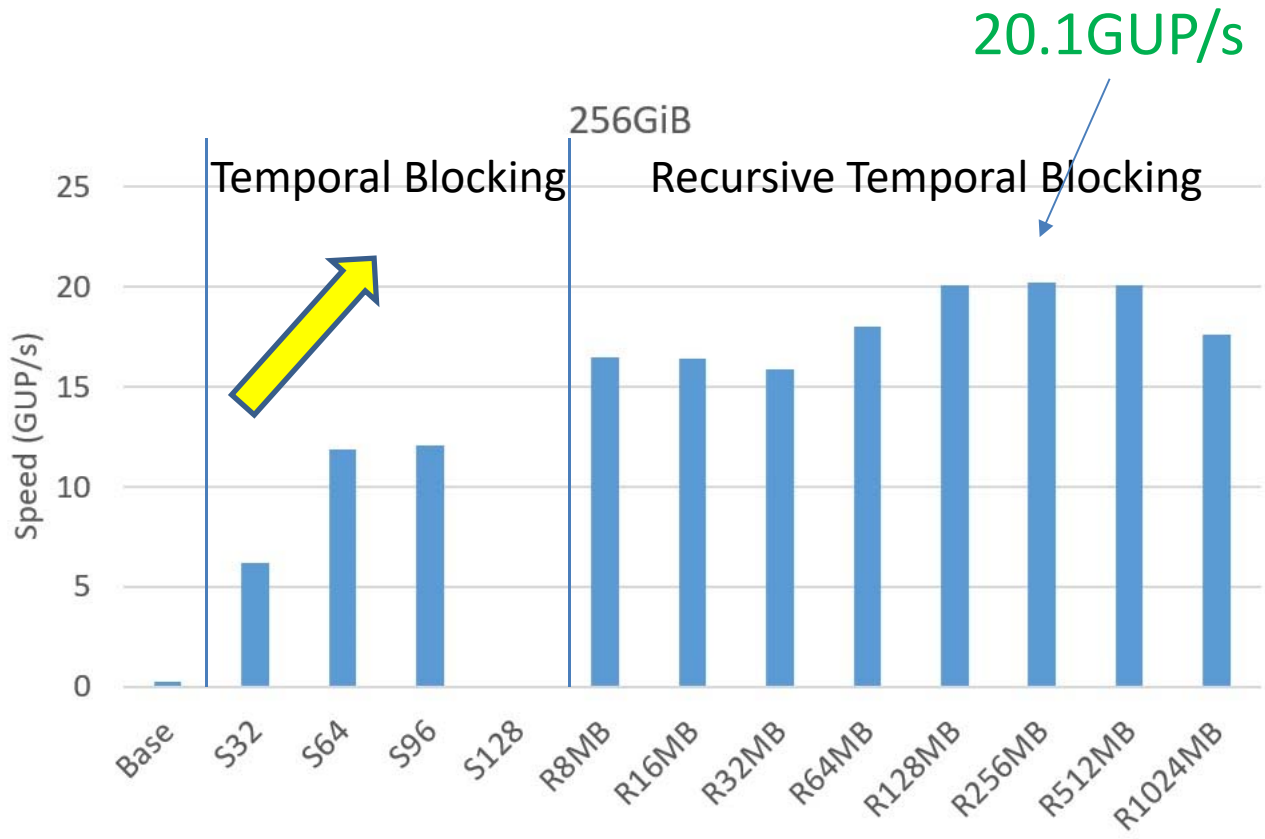
GPU memory
host memory
SSD



GUP/s: giga updated points per second

Results: 256GB Domain

GPU memory
host memory
SSD



GUP/s: giga updated points per second

Summary

- Toward high-speed & big stencil computations, a recursive algorithm efficiently harness memory hierarchy
  - HBM2 GPU memory + DDR4 host memoy + Optane SSDs
  - Also it works well with GPU cache !

Issues & Future Work

- Out-of-core performance (20.1GUP/s) is still 30% of In-core performance (67.7GUP/s)

  The implementation is still in the early stage.

  We need to improve it by

  – Overlapping computation and data movement
  – Comparing automatic movement and manual movement
  – Considering memory access alignment on GPUs
  – Combining existing optimizations such as 3.5D blocking

- Using multiple GPUs, multiple nodes…
- Using newer NVM technologies, including 3D-XPoint based DIMMs