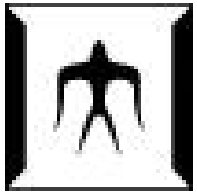
Software Technologies Coping with Memory Hierarchy of GPGPU Clusters for Stencil Computations

Toshio Endo, Guanghao Jin

Tokyo Institute of Technology

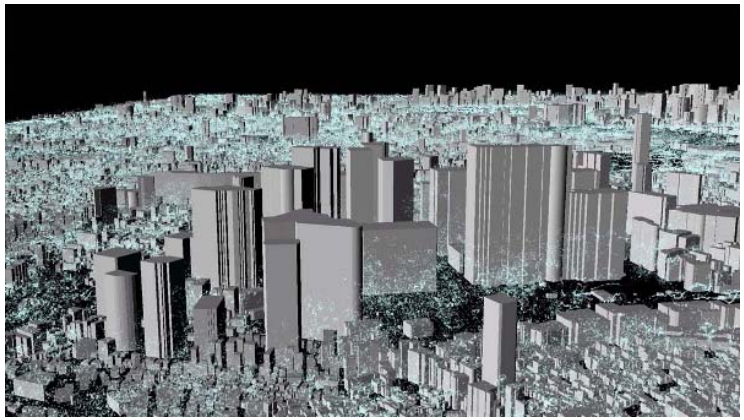


Our Target: Stencil Computations

Important kernels for many simulation applications

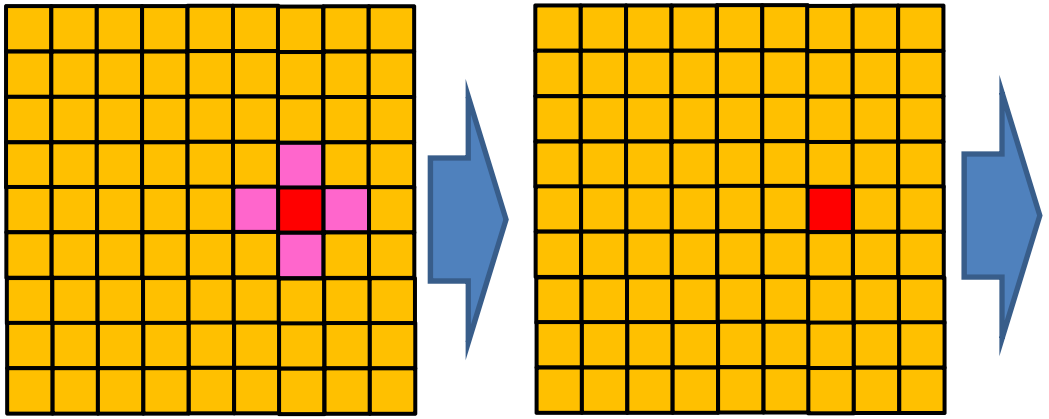
ASUCA weather simulation

Metal crystal simulation (2011 Gordon Bell)



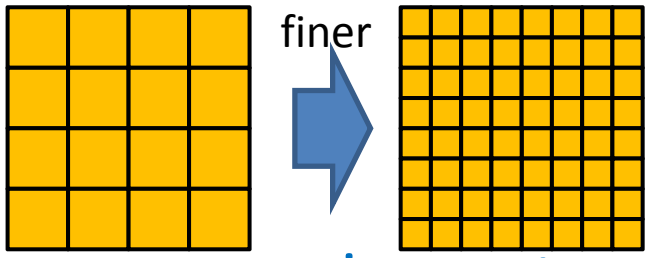
Air flow simulation

=



Requisites for Developing/Improving Stencil Applications

- High Performance
  - Already achieved on GPU clusters
  - High memory BW (GB/s) and Flops are keys
- Large Scale
  - More precise simulations require more Bytes



Currently limited by GPU device memory capacity (=O(1)GB x #devices)

- Low programming cost
  - If applications with O(1K~100K) lines of code exist, it is hard to rewrite entirely

Our Target: Realizing ***extremely Fast&Big simulations*** of O(100PB/s) & O(100PB) around 2020

Our Concept towards Fast&Big Stencils

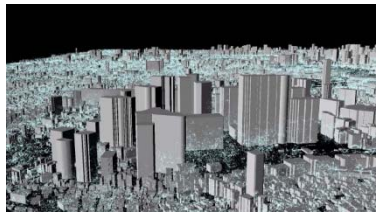
Comm. avoiding algorithms

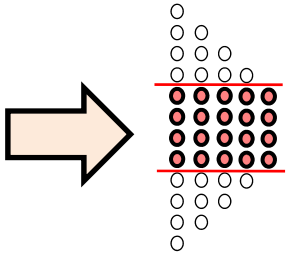
+

System software for memory hierarchy management

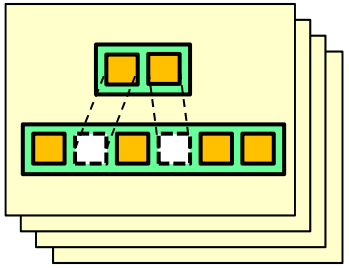
+

HPC Architecture with deeper memory hierarchy

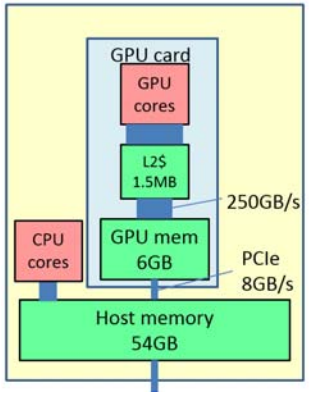




Temporal blocking technique [Wolf 91] improves locality



Our HHRT library supports memory swapping between device memory and host memory



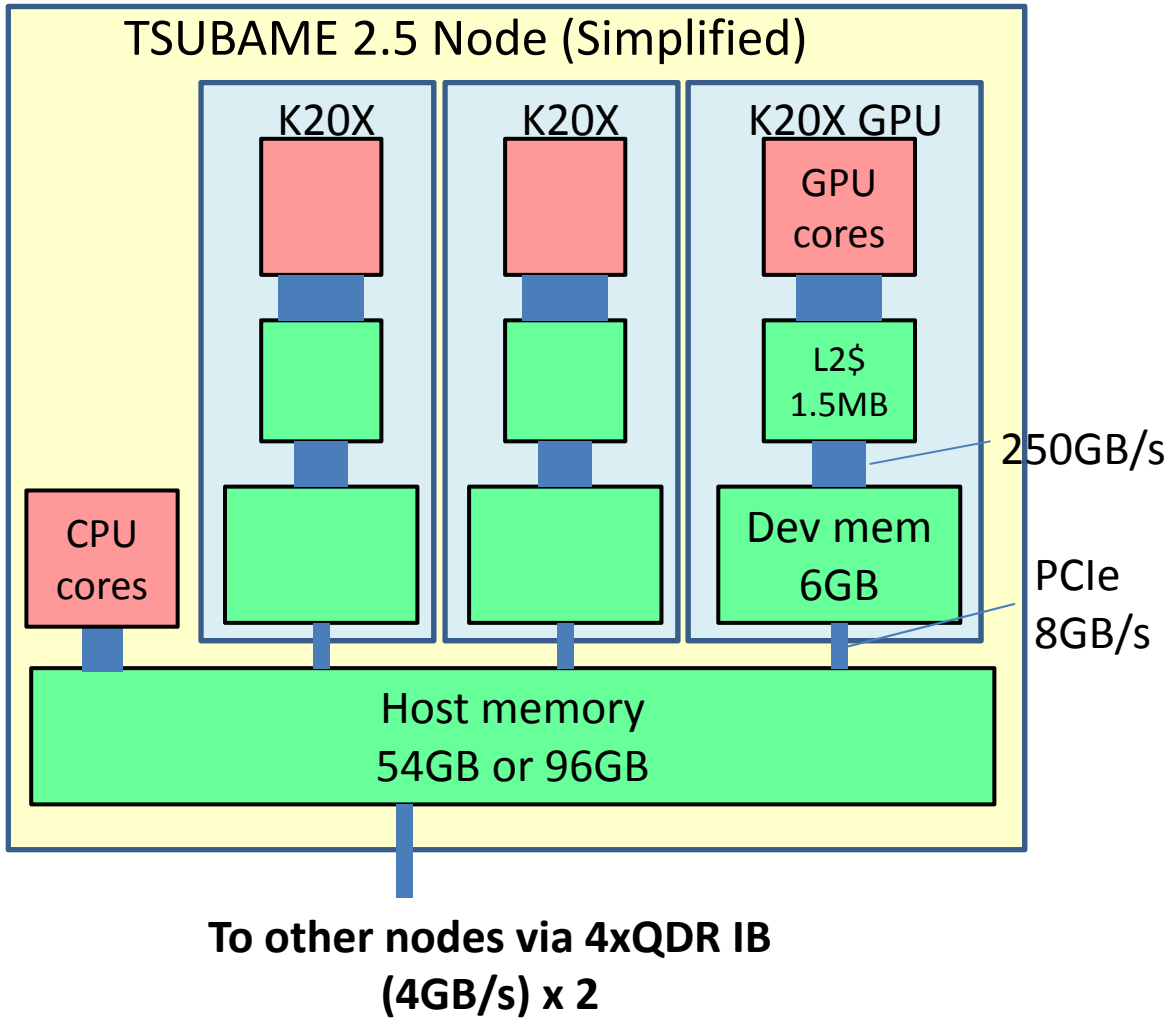
2-Tier memory hierarchy
- GPU device memory:
  Faster (250GB/s) and smaller (6GB)
- Host memory:
  Slower (8GB/s) and larger (>50GB)

via PCIe

TSUBAME2.5 GPU Supercomputer

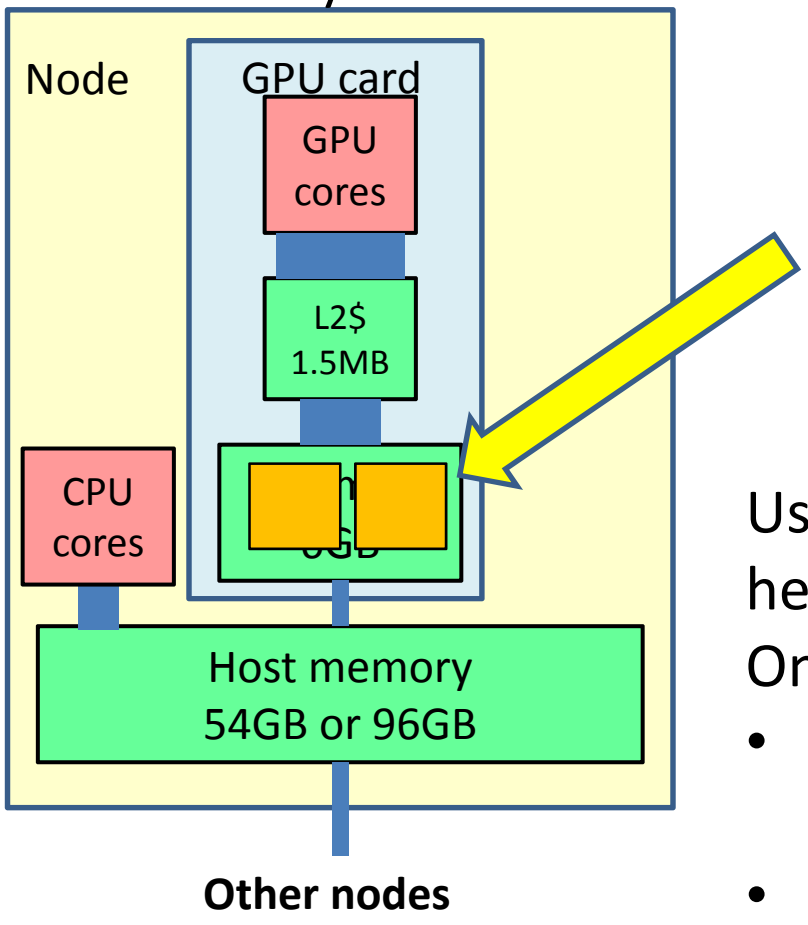
5.7PFlops system has
1408 compute nodes

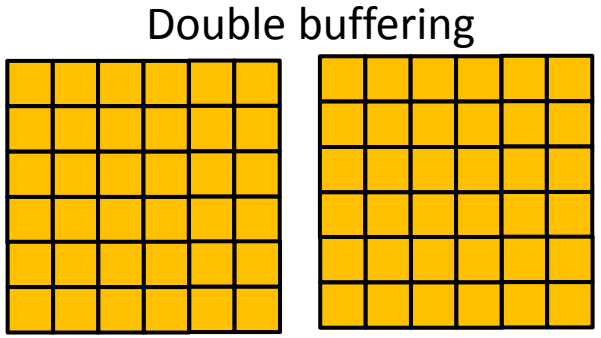
A node has
12 Xeon cores (2.9GHz) &
3 NVIDIA K20X GPUs



Common Ways for Programming Stencil on GPUs

- Usually, code is written with CUDA and MPI (in multi-GPU cases)
- Double buffering (one is for even *t*, and the other for odd *t*)
- Typical programmers let domain sizes smaller than GPU device memory size





Using multi-GPUs & domain decomp help, but host memory size is even larger!
On TSUBAME2.5,

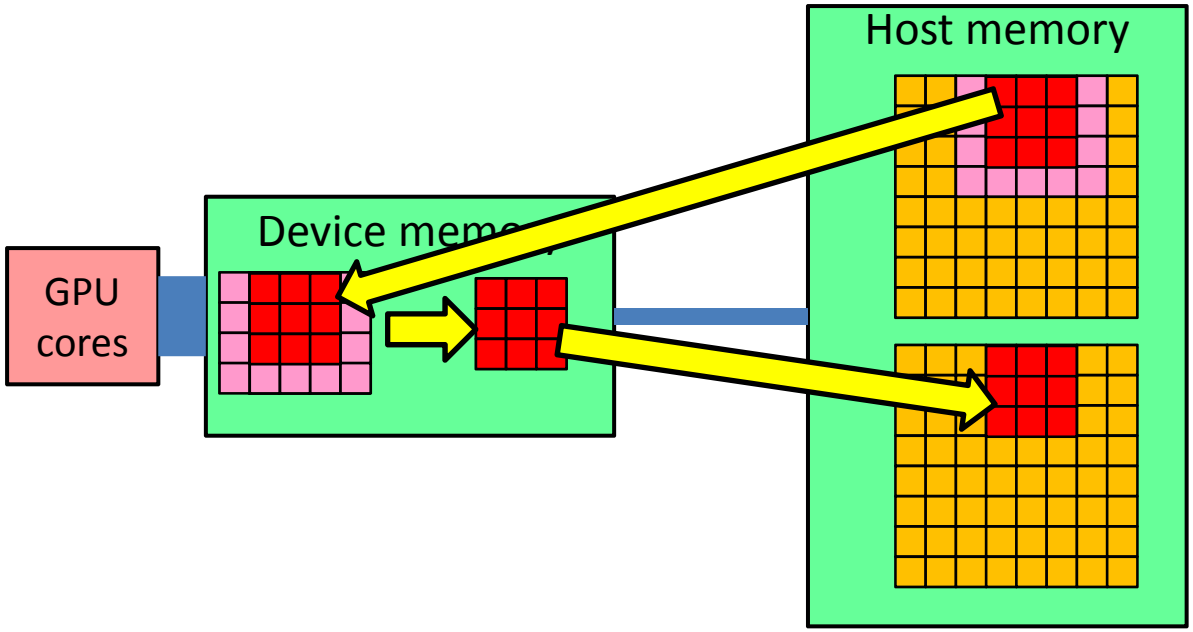
- Total GPU memory: 24TB

∧

- Total host memory: 82TB

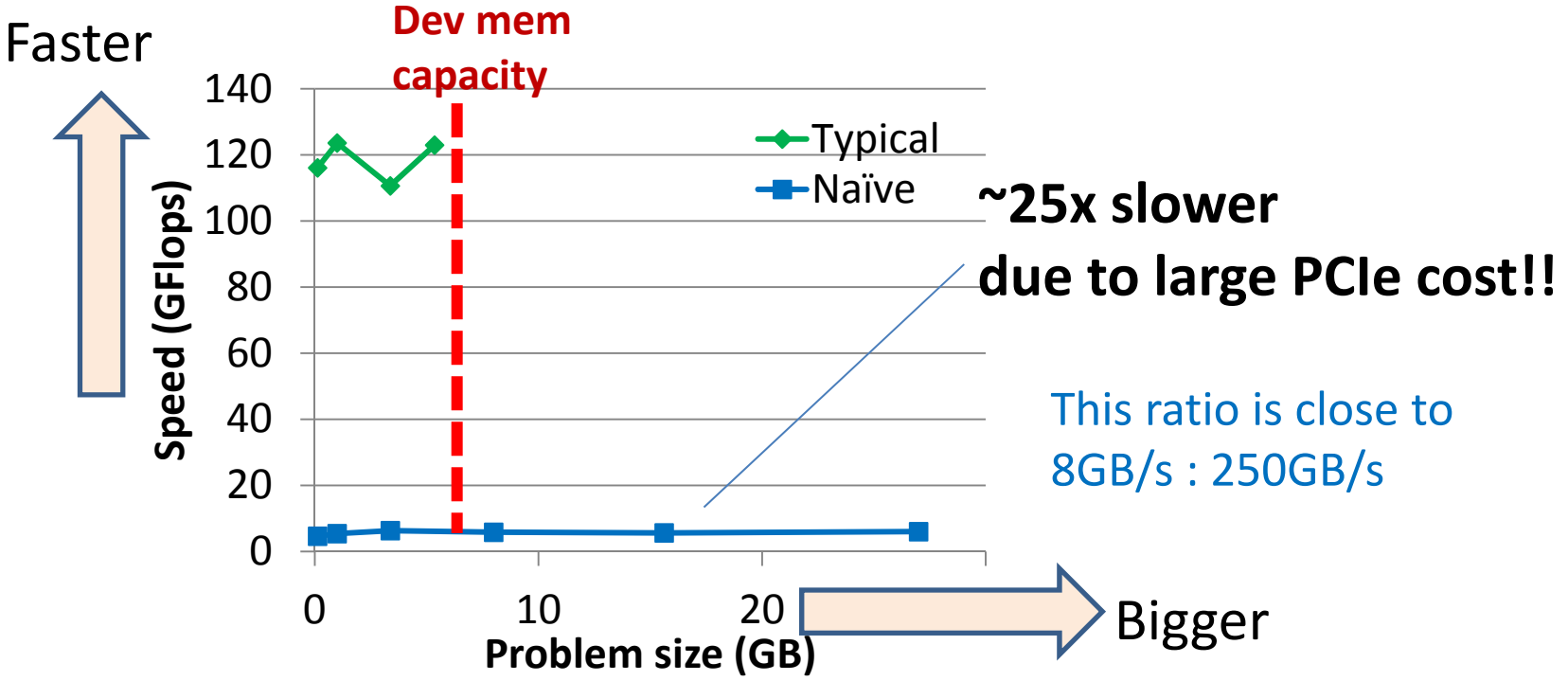
Motivating Example:
What if domain sizes exceed GPU memory? (1)

- A naïve method is:
  - Put domain data on host memory
  - We divide the domain into small sub-domains
  - Repeat
    - Copy a “sub-domain” into GPU
    - Compute
    - Copy back results



Motivating Example:
What if domain sizes exceed GPU memory? (2)

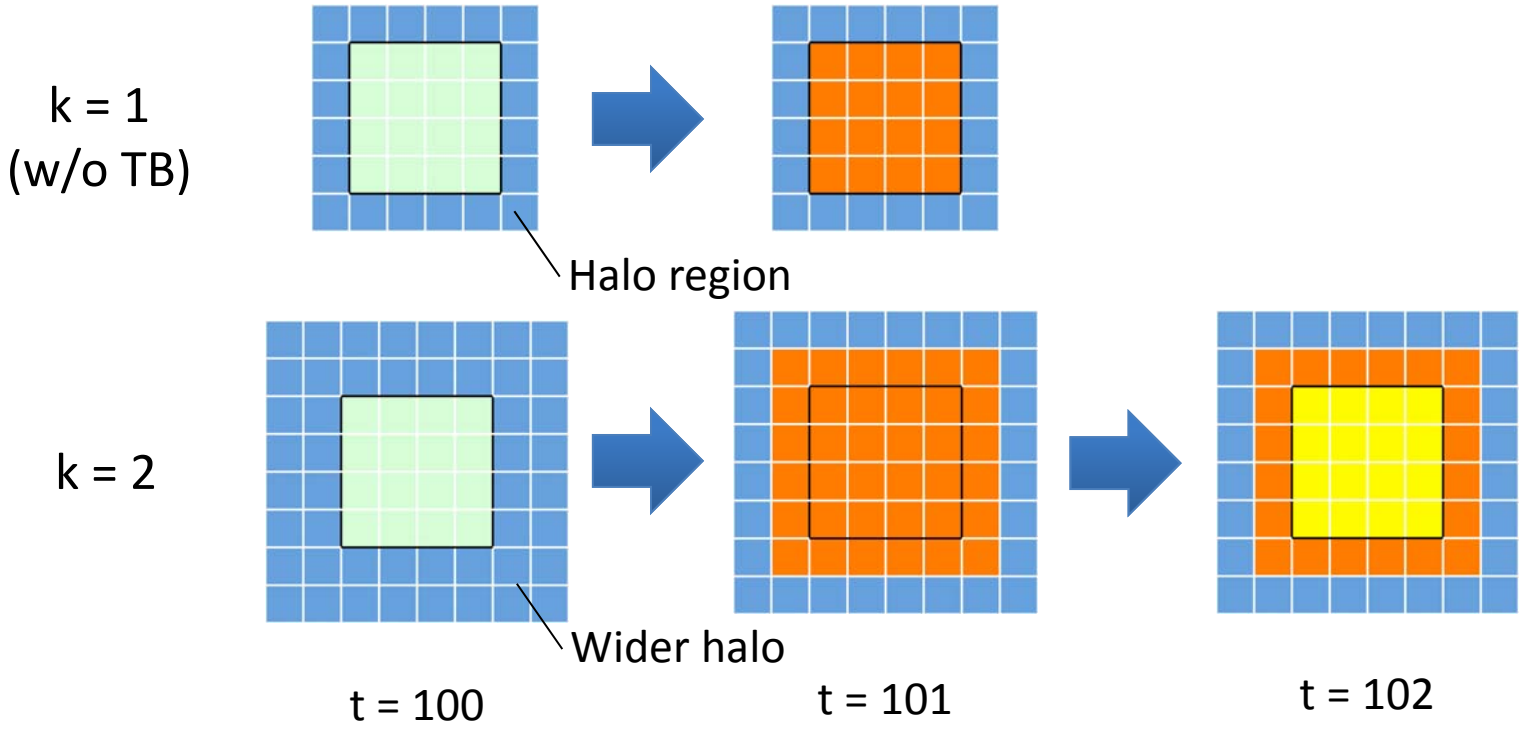
3D 7point stencil on a TSUBAME node
A K20X GPU (6GB GPU mem) is used



There is tradeoff between speed and size
Keys for improvement are "Communication avoiding" algorithms → Change access patterns

Temporal Blocking (TB) for Comm. Avoiding

- TB was originally proposed for better cache locality [Wolf 91] [Wonnacott 00] → We apply it for GPU computing
- When we pick up a sub-domain, we perform **multiple (*k*-step) updates** on GPU at once, and then proceed to the next one
  - *k*: temporal block size

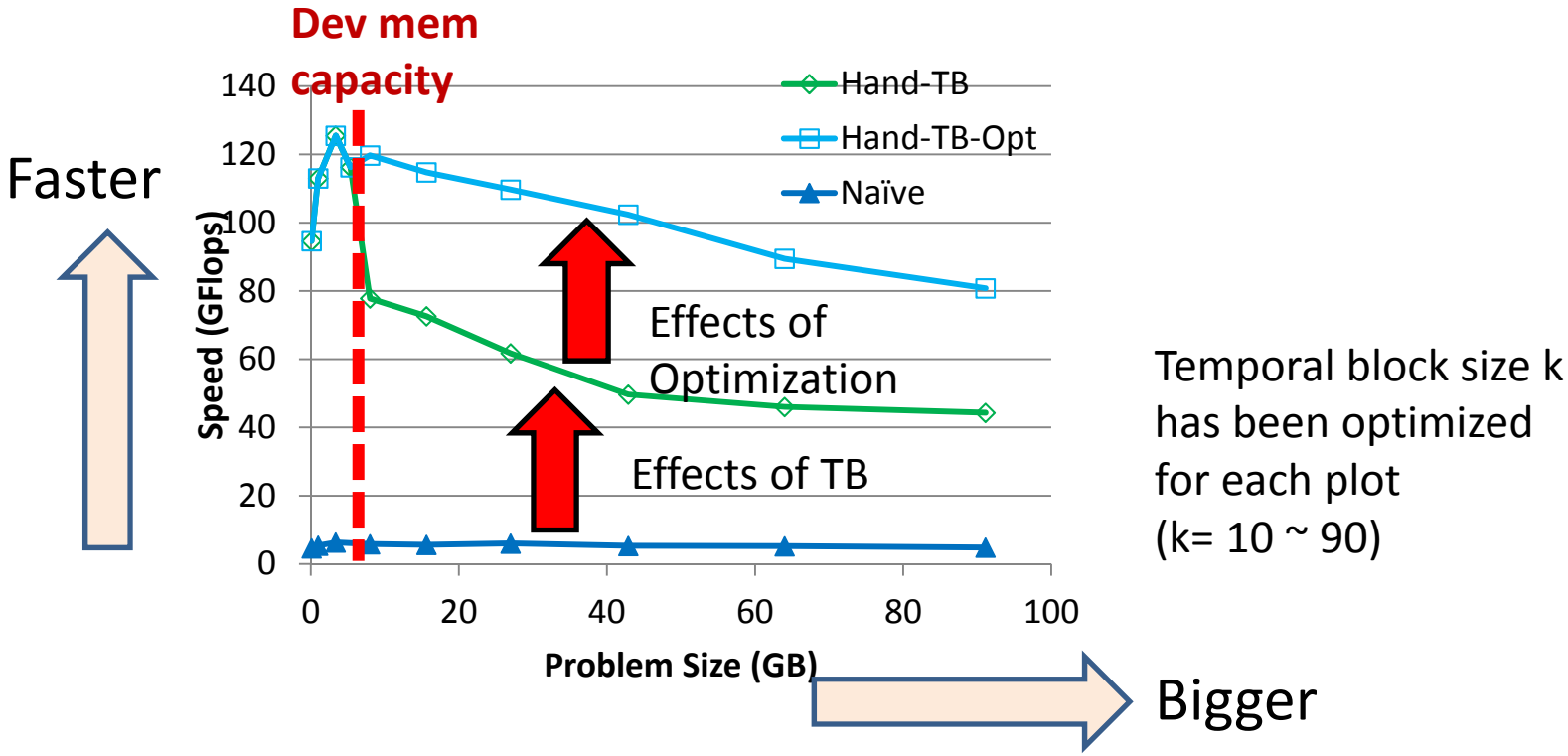


Pros&Cons of TB on GPU

- Pros
  - PCIe communication is reduced to ~1/k !!
- Cons
  - Due to wider halo region, we suffer from redundant computation
  - Code gets much more complex

Effects of Temporal Blocking

3D 7point stencil on a TSUBAME 96GB node
A K20X GPU (6GB GPU mem) is used

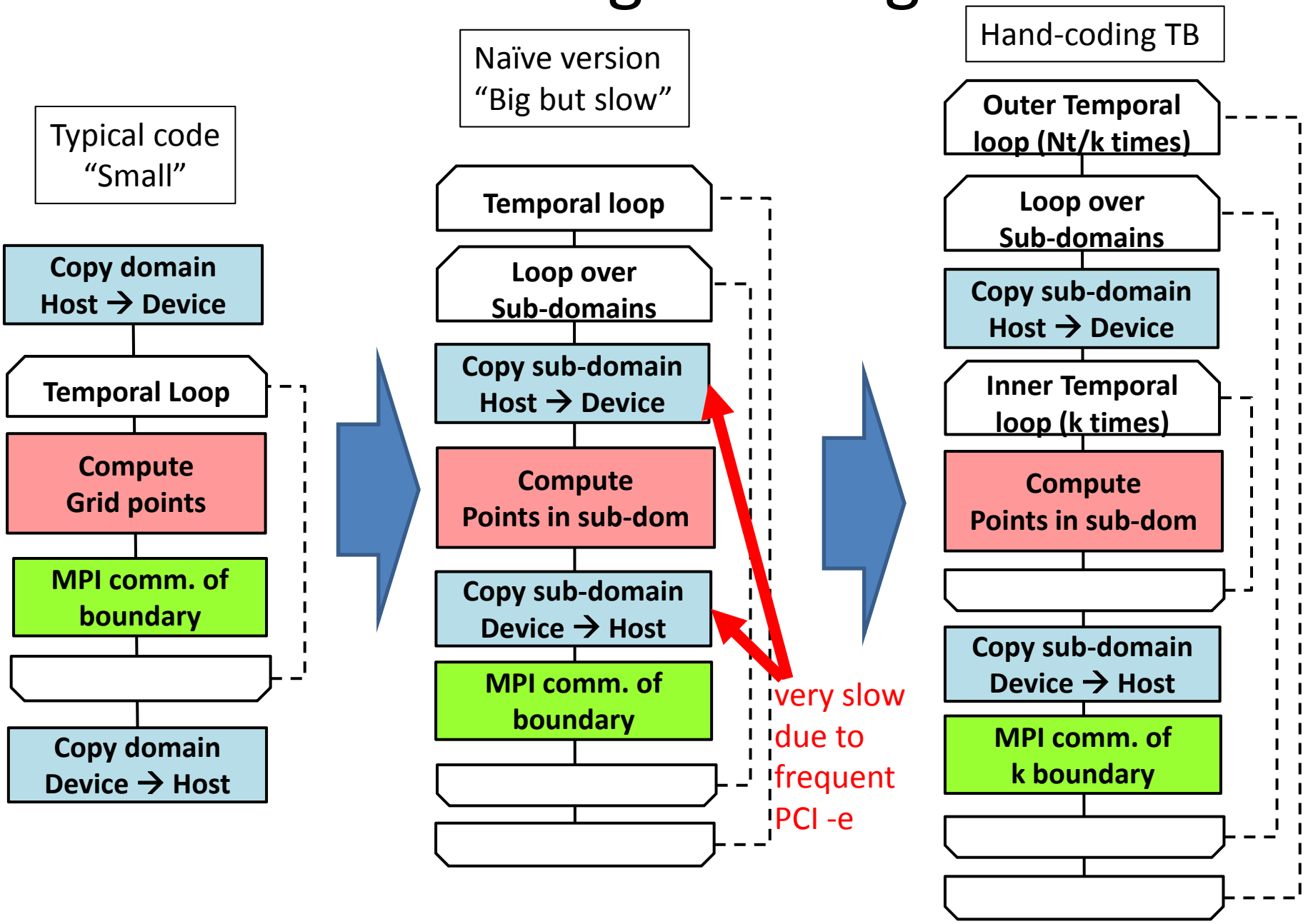


Temporal block size k has been optimized for each plot (k= 10 ~ 90)

For the optimized version, please refer to

G.Jin, T.Endo, S. Matsuoka: A Parallel Optimization Method for Stencil Computation on the Domain that is Bigger than Memory Capacity of GPUs, Cluster 2013

But How about Programming Cost?



Considering Programming Cost

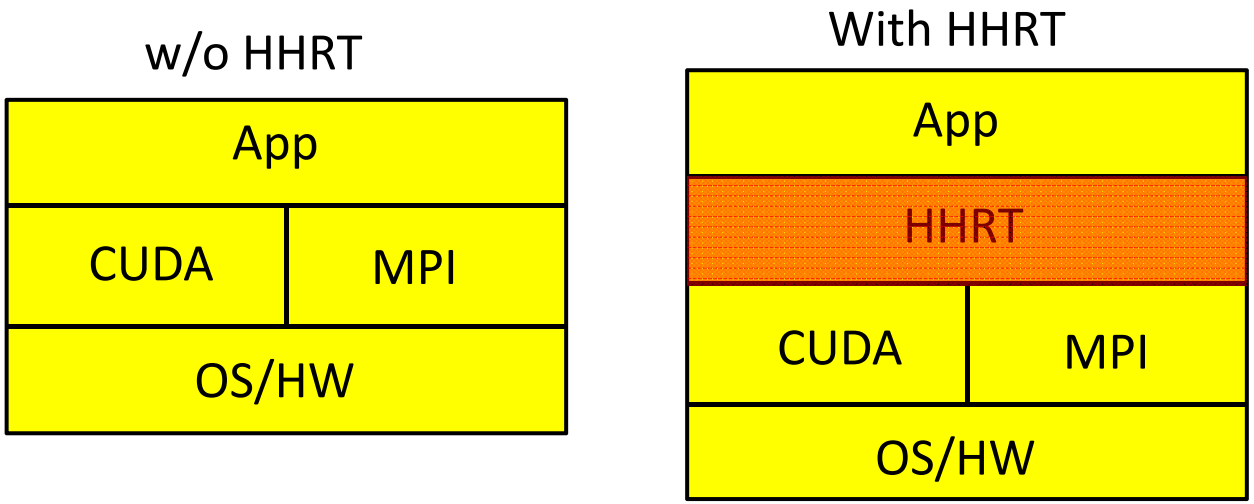
- Differences between “typical” and “hand-coding TB”
  - “Sub-domain” loop is introduced
  - Temporal loop is divided into “inner” and “outer”
  - PCIe and MPI comm is done out of “inner” loop
- We were happy if we could automatically convert “typical” to “TB” code, but it is hard

Instead, our approach is:

- Reducing programming cost by using system software, named HHRT (Hybrid Hierarchical Runtime) , which is aware of memory hierarchy

The HHRT Library

- HHRT supports applications written in CUDA and MPI
  - HHRT works as a wrapper library of CUDA/MPI

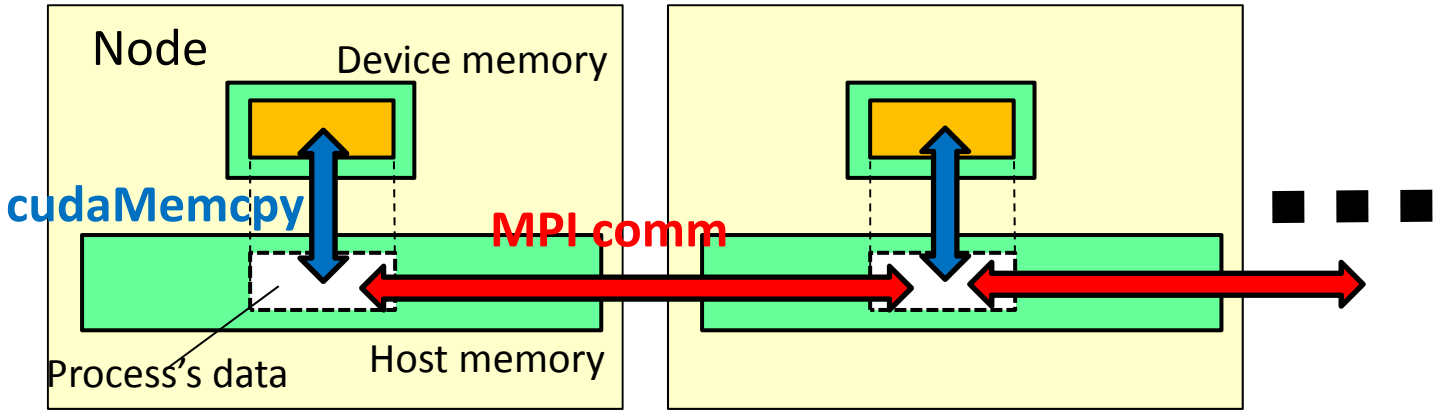


Functions:

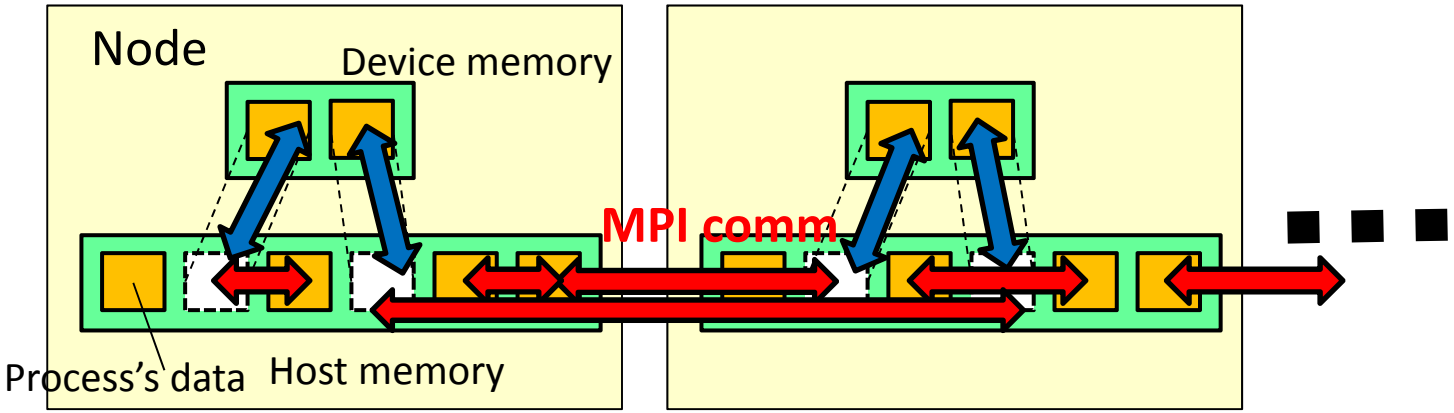
- HHRT supports overprovisioning of MPI processes on each GPU
- HHRT executes memory swapping between device memory and host memory
  - Not “page-wise” swapping, but “process-wise” swapping

Execution model of HHRT

w/o HHRT (typically)



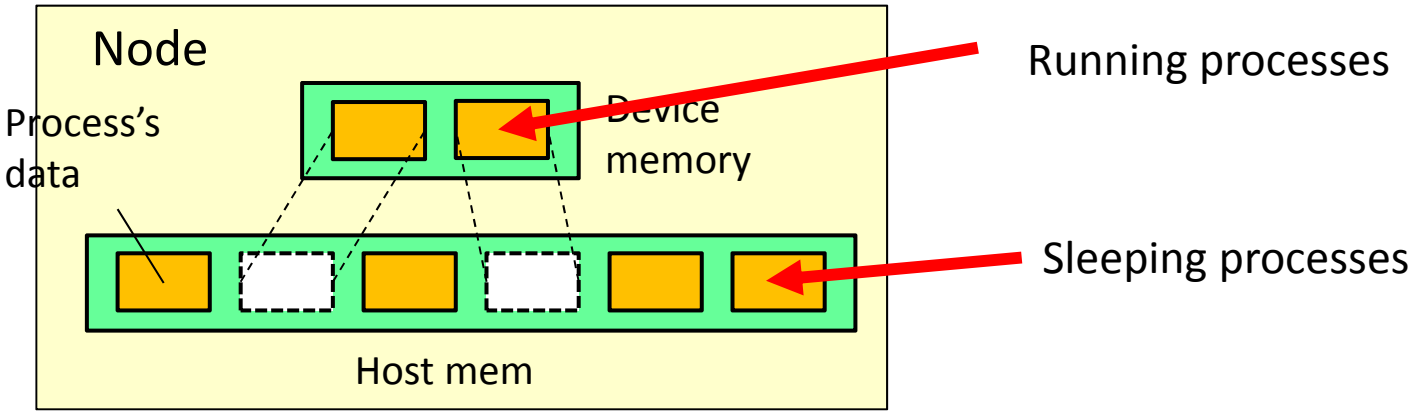
With HHRT



m MPI processes share a single GPU

In this case, m=6

Processes on HHRT



- We suppose

  s < Device-memory-capacity < $m\ s$

  s: Size of data that each process allocates on device memory

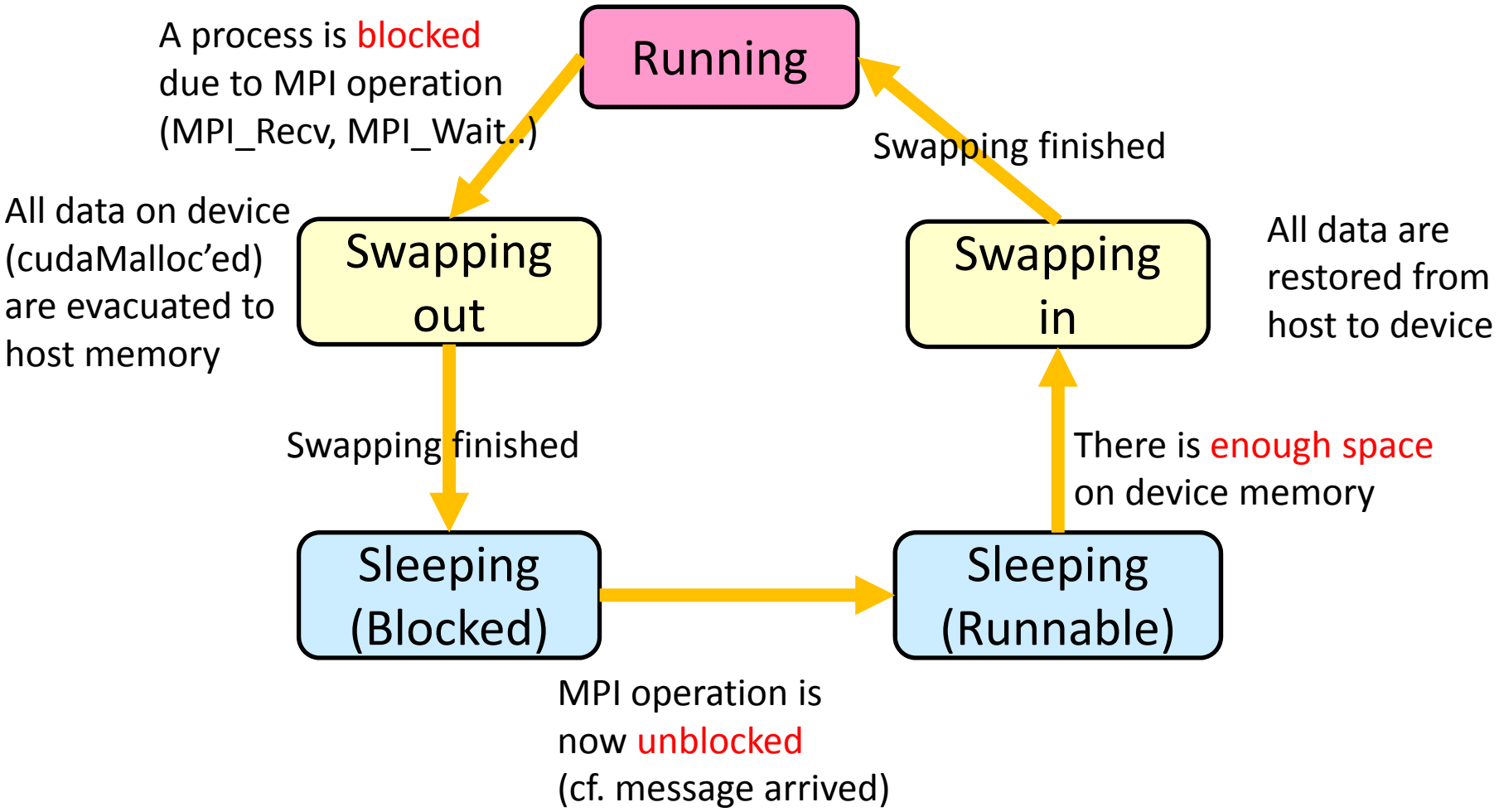
  m: The number of processes sharing a GPU

→ We can support larger data size than device memory in total

- But we cannot keep all of *m* processes running

→ HHRT makes some processes "sleep" forcibly and implicitly

State Transition of Each Process



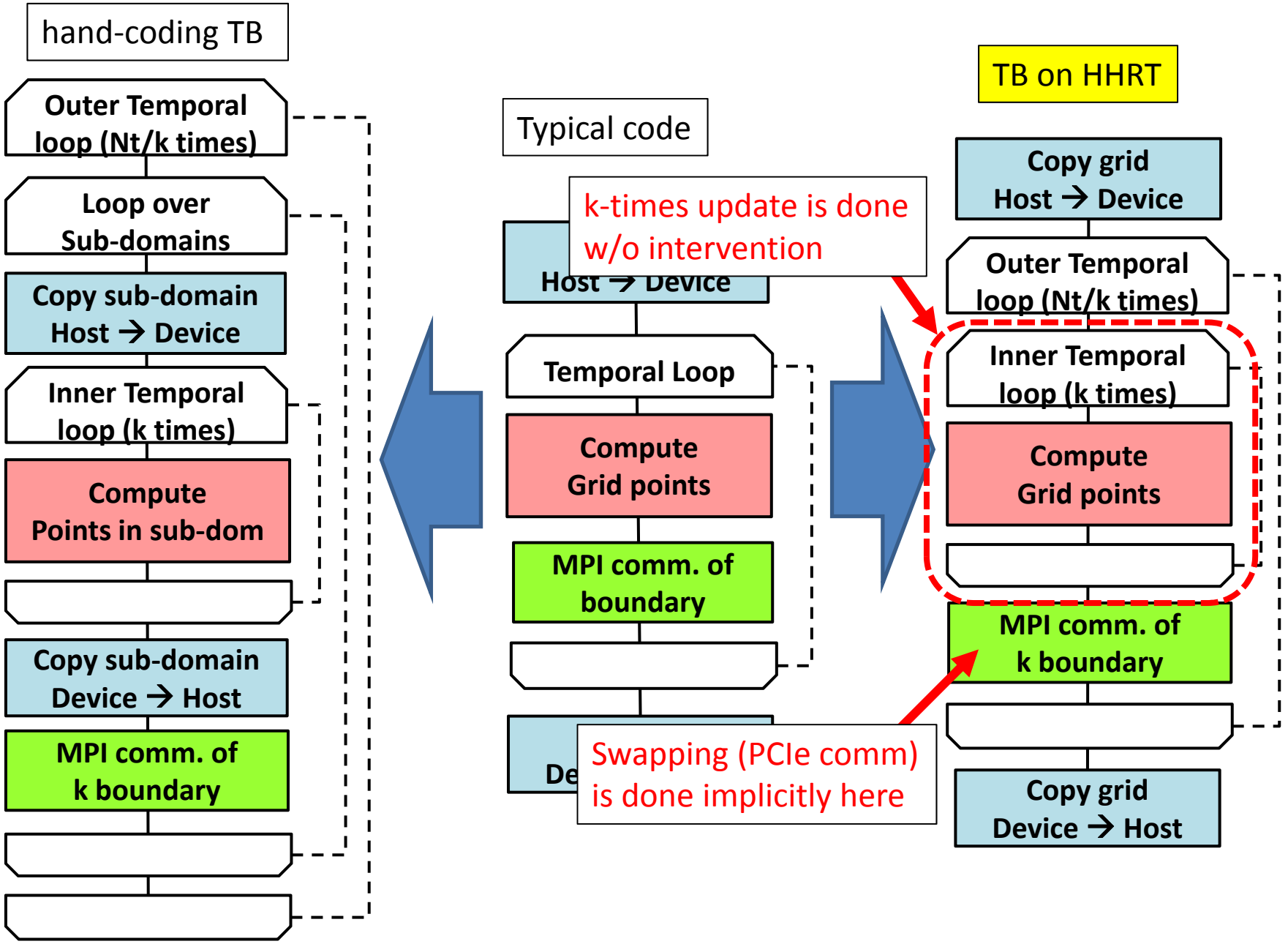
Implementing Temporal Blocking on HHRT (1)

- How do we divide larger domain into smaller sub-domains?
  - w/o HHRT: 1GPU ← 1 process ← m sub-domains
  - With HHRT: 1GPU ← m processes ← m domains

    Each process maintains only one domain

    We don't need additional sub-domain loop
- How is domain data moved?
  - w/o HHRT: PCIe comm is done explicitly
  - With HHRT: Implicitly within MPI comm
- On the other hand, doubly nested temporal loops should be written in hand

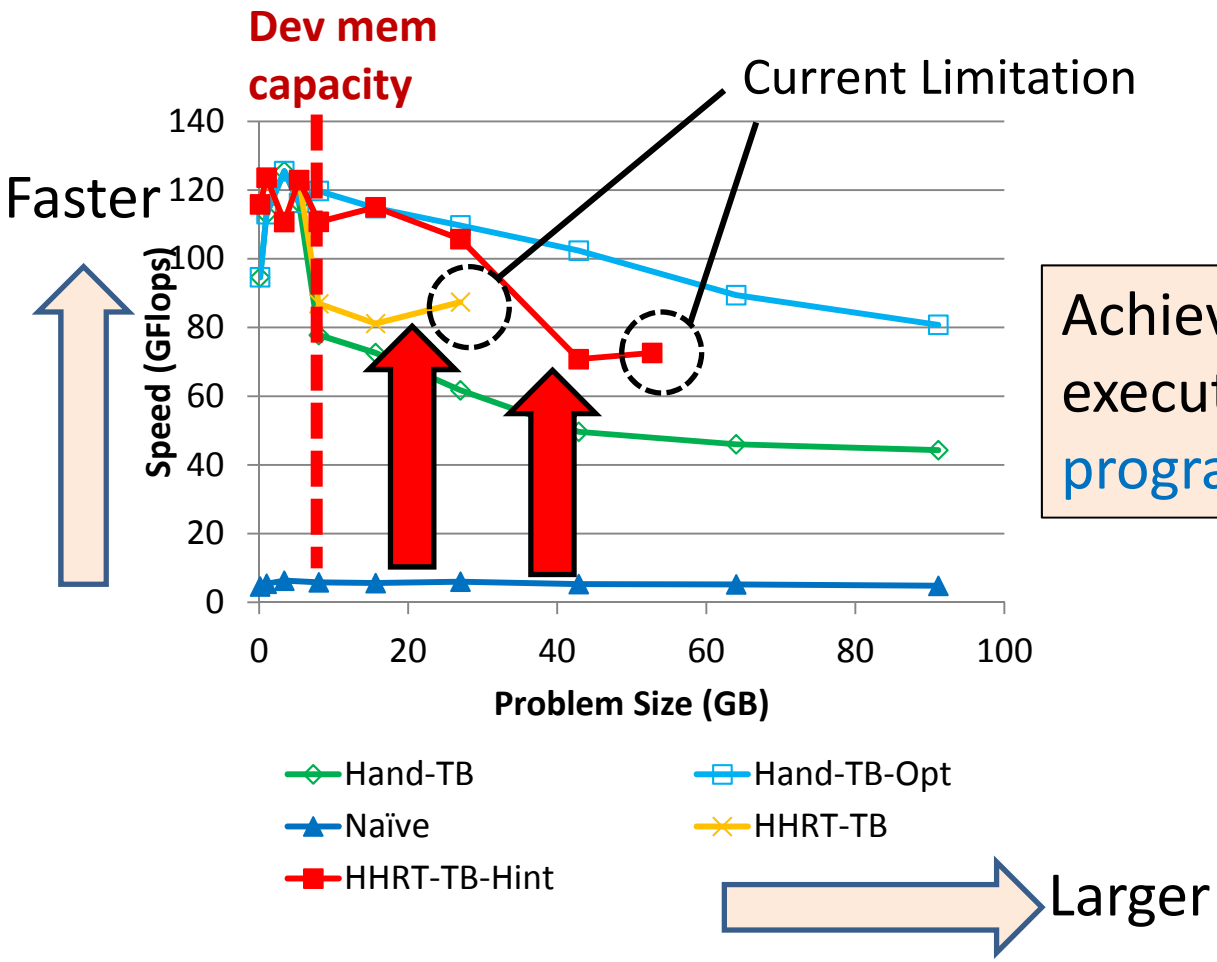
Implementing Temporal Blocking on HHRT (2)



Current Results with HHRT

3D 7point stencil on a TSUBAME 96GB node
A K20X GPU (6GB GPU mem) is used

- SLES Linux 11SP1
- CUDA5.5
- OpenMPI 1.6.3
- gcc 4.3.4



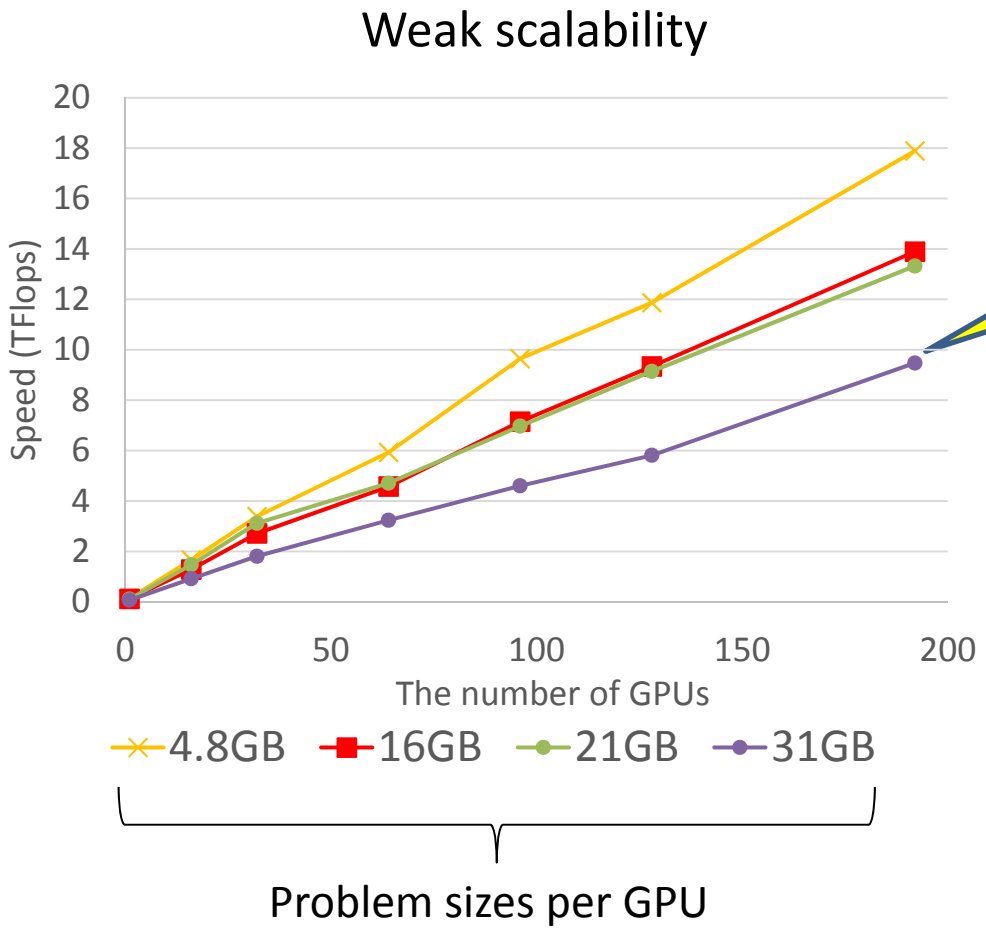
Achieving **fast&(fairly) big** execution with moderate programming cost

Current Limitations

- Memory pressure
  - HHRT itself consumes host memory for swap buffers → Applications cannot use the entire host memory
    - Programmers can provide "hints" on livingness of each buffer → memory pressure is reduced and performance is improved
    - Refer to Section IV-B of our paper
- Performance
  - Optimizations done in Hand-coding version have not integrated
  - Communication of halo region is heavy
    - cudaMemcpy(D2H) – MPI_Send/Recv – cudaMemcpy(H2D) are required even within a single GPU
  - Scheduling algorithm of HHRT is to be improved

Scalability with HHRT

- 3D 7point stencil on multiple TSUBAME2.5 54GB nodes
- 1GPU per node is used

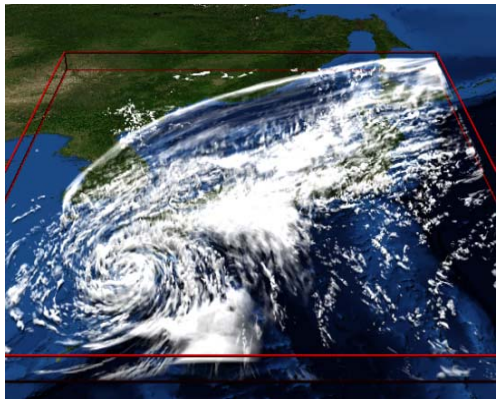
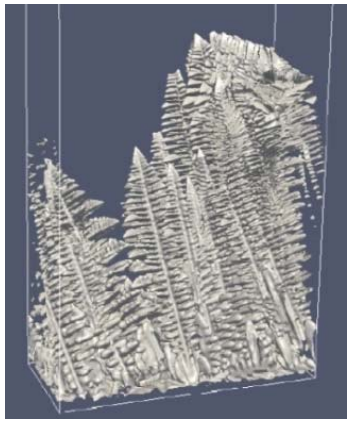
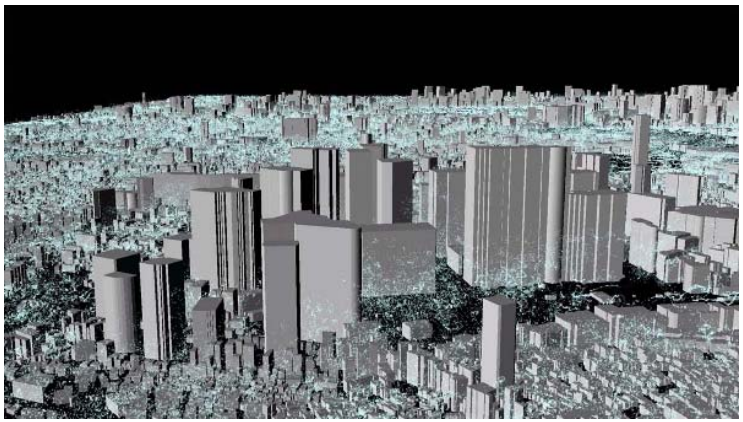


9.5TFlops with 6TB Problem

Showing good scalability

Summary

Fast&Big&Easy (Stencil) **Simulations** are becoming ready

>>

Comm. Avoiding Algorithms + System Software + Deeper Memory Hierarchy

Future Work

- Harnessing memory hierarchy with 3-tier or more
  - Device memory + Host memory + Flash/burst buffer

  → Towards O(1TB)/node scale
- Improving performance
  - Removing redundant computation
  - Supporting GPU-direct by HHRT
  - Improving scheduling methods of HHRT
- HHRT for Xeon Phi
- Defining next-gen memory architecture
  - HMC/HBM, ReRAM, PCM, STT-RAM…
  - Which should be included? Capacity balance?