Case Study : Scientific Code Acceleration

Kwangmin Yu / Computational Science Lab

BROOKHAVEN NATIONAL LABORATORY

U.S. DEPARTMENT OF ENERGY

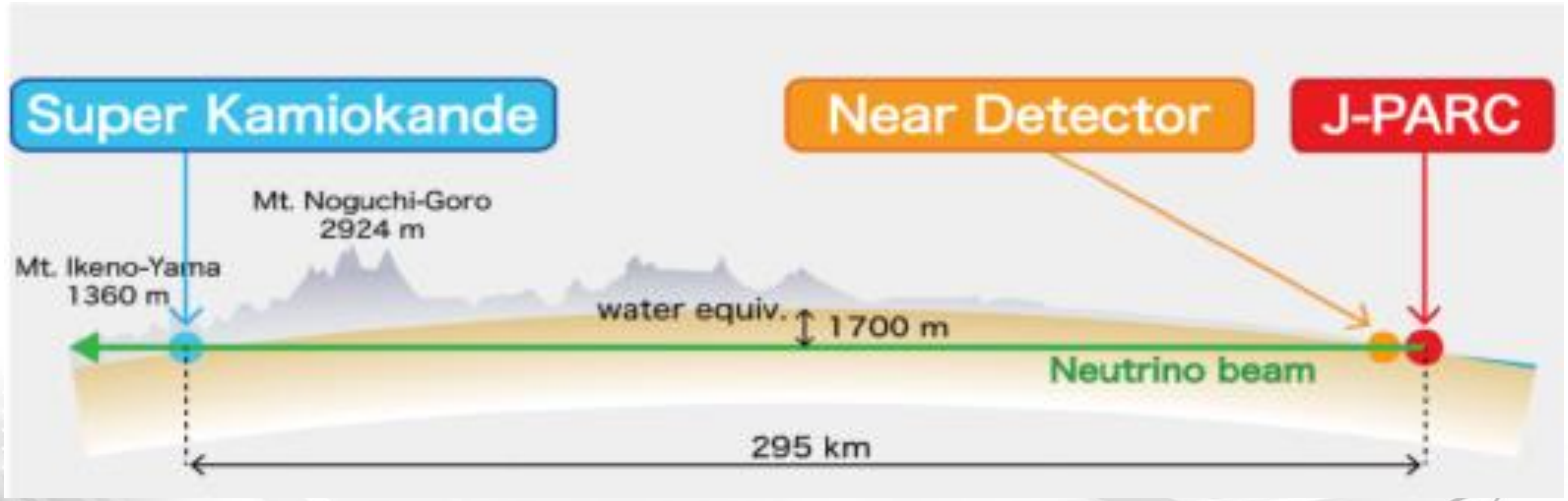
CASE I

Fast Trigger

2017 BNL GPU Hackathon Application

Fast Trigger

- ✓ Part of HyperKamiokande experiment in Japan.
- ✓ Data manipulation code from 40k photosensor collecting light generated by neutrinos.
- ✓ Realtime data processing is required.
- ✓ 99.99% of data is noise. Only 0.01% is collected.
- ✓ **Already gpuized and about 10000 times (four order of magnitude) speed-up over CPU is achieved.**
- ✓ Needs more speed-up because 400 GPUs are needed for realtime processing.
- ✓ NO math library is used (No math & simple data manipulation).



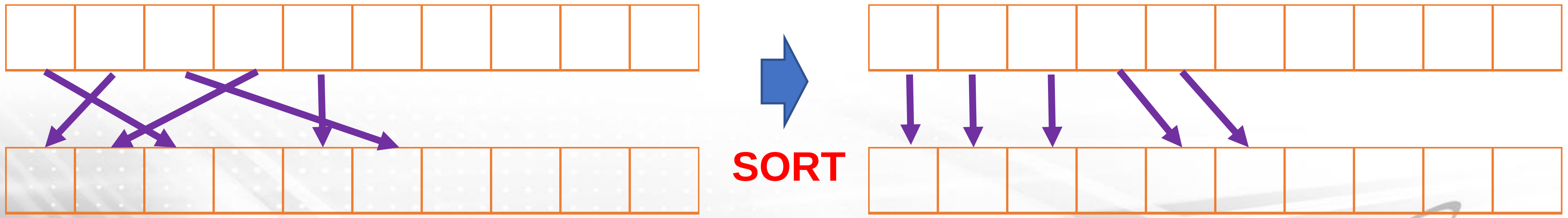
Fast Trigger

- ✓ The sorting is the most significant speedup factor (**3x**).
- ✓ Final speed-up is **5.3x**.

O	Migrating the look-up table to the shared memory (L1 cache)
O	Partial loading of the look-up table (Memory deficiency)
O	Using min max lookup table to shorten memory needs
X	Pair the inputs to coalesce memory
X	L1 cache increase
X	Persistent shared memory
O	Local arrays as temporary stores to reduce atomic adds
O	Sorting data by the look-up table
O	Using __ldg() instrinsic

Strategy	Running Time (ms)
Original code	357
Sorting (Look-up table)	**100**
Sorting + L1 cache (increased)	160
Sorting + Reduced Look-up table	80
Sorting + Reduced Look-up table + L1	100
Sorting + Reduce more	77
Sorting + Reduce more + __ldg()	**68**

GPU Global Memory Coalescing



CASE II

Quantum ESPRESSO

Add-on Module Ver.

Quantum ESPRESSO

is an integrated suite of Open-Source computer codes for electronic-structure calculations and materials modeling at the nanoscale.
Based on **Density Functional Theory**, **Plane Waves**, and **Pseudopotentials**.

- Partially GPUized

	K point	**Gamma**	**Non Collinear (NC)**
PW (VLOC_PSI)	O	O	★
PH (H_PSIQ)	★	X	X

O: previous acceleration, ★: our contribution, X: Not accelerated yet

Quantum ESPRESSO

- **Loop over Bands**
  I. **h_psiq_kernel_init_psic_k**
  II. **cufftExecZ2Z(PSIC, Inverse)**
  III. **h_psiq_kernel_vec_prod_k**
  IV. **cufftExecZ2Z(PSIC, Forward)**
  V. **h_psiq_kernel_save_hpsi_k**

FFT Plan Initialization was $\longrightarrow$

1. Memory initialization
2. DFT(Discrete Fourier Transformation) Matrix (Vandermonde Matrix) Initialization

SOLUTION:

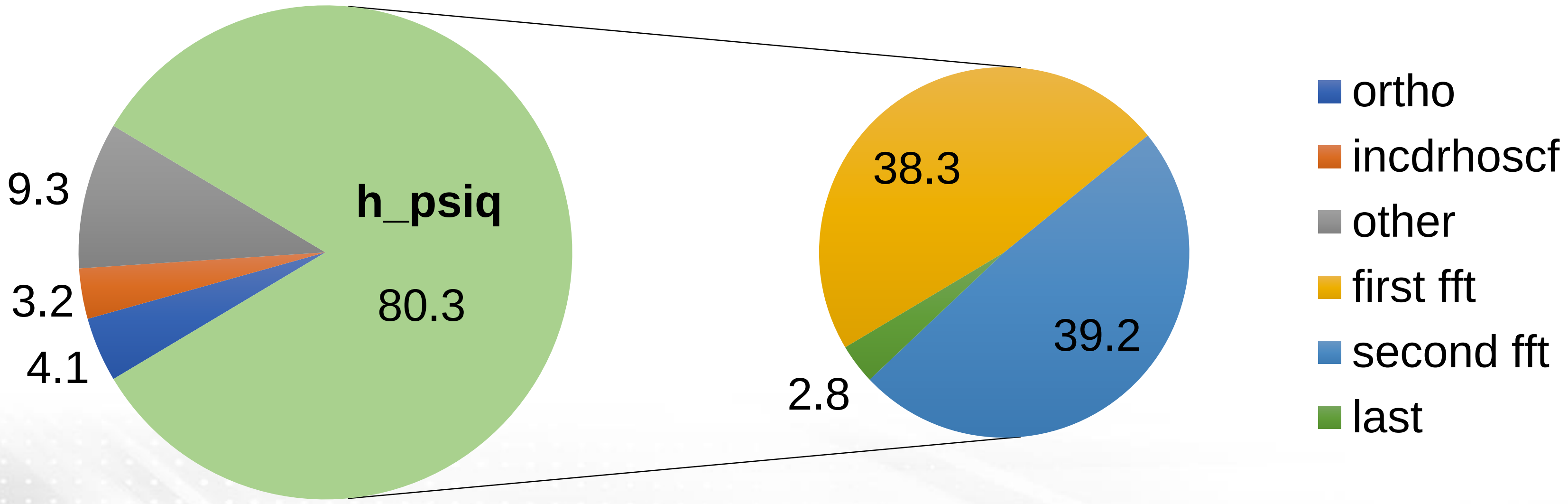
✓ Move the initialization out of the loop

✓ The initialization is called only when the plan size changes.

$$\mathbf{F} = \begin{bmatrix} \omega_N^{0\cdot 0} & \omega_N^{0\cdot 1} & \cdots & \omega_N^{0\cdot(N-1)} \\ \omega_N^{1\cdot 0} & \omega_N^{1\cdot 1} & \cdots & \omega_N^{1\cdot(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_N^{(N-1)\cdot 0} & \omega_N^{(N-1)\cdot 1} & \cdots & \omega_N^{(N-1)\cdot(N-1)} \end{bmatrix}$$

Quantum ESPRESSO

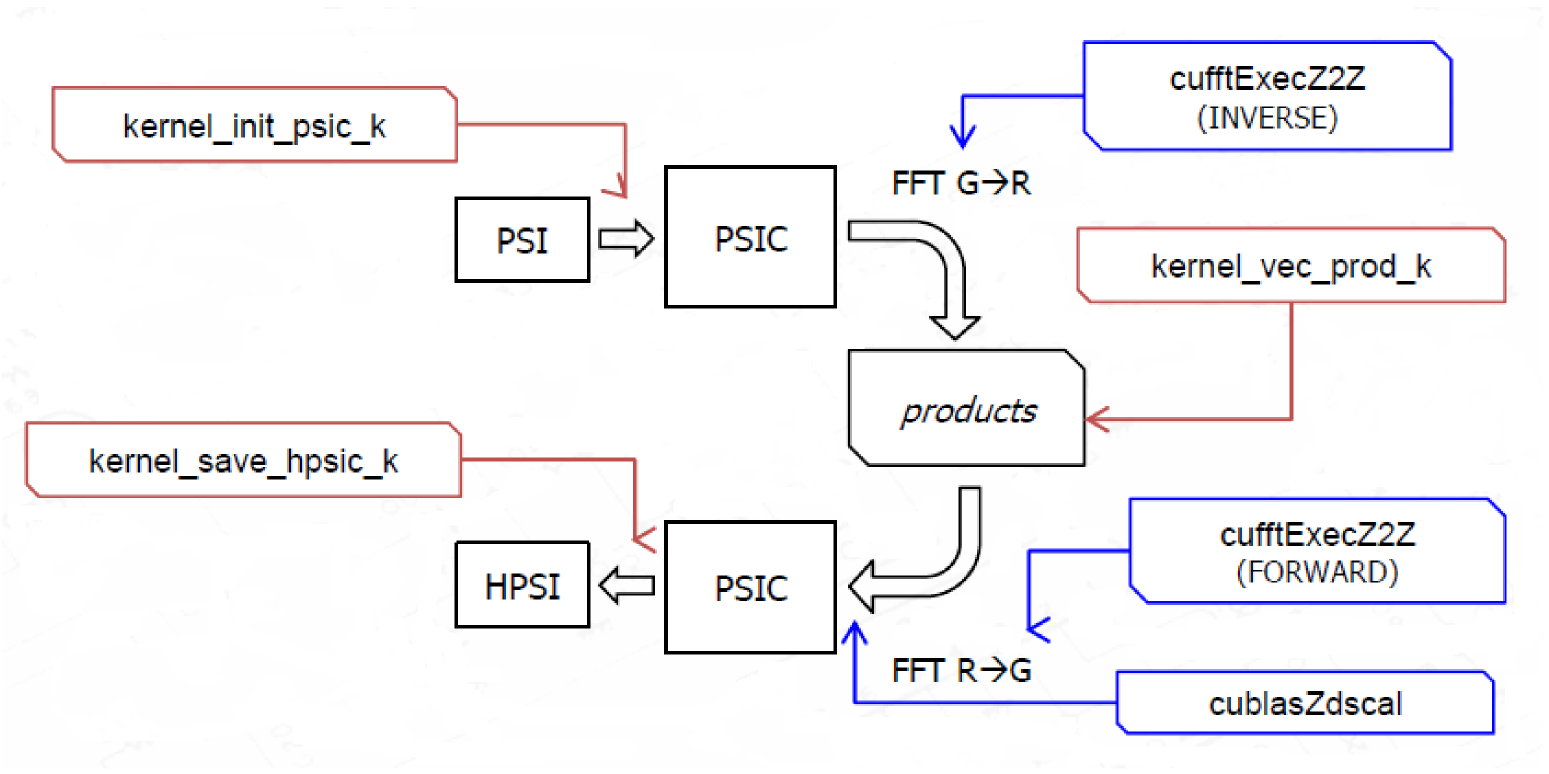
- ✓ Physical system: an Ar atom in a big box
- ✓ Description: RPA calculation for 200 eigenmodes at 12 frequencies
- ✓ Total CPU time: 8 h 51 min with 48 processors



Quantum ESPRESSO
Pseudocode

H_PSIQ ((H - eS) * PSI)	VLOC_PSI (H * PSI)
● Call CALBEC	
● h_psiq_kernel_init_hpsic() ● Loop over Bands I. h_psiq_kernel_init_psic_k II. cufftExecZ2Z(PSIC, Inverse) III. h_psiq_kernel_vec_prod_k IV. cufftExecZ2Z(PSIC, Forward) V. h_psiq_kernel_save_hpsi_k	● Loop over Bands I. h_psiq_kernel_init_psic_k II. cufftExecZ2Z(PSIC, Inverse) III. h_psiq_kernel_vec_prod_k IV. cufftExecZ2Z(PSIC, Forward) V. h_psiq_kernel_save_hpsi_k
● Call ADD_VUSPSI ● Call S_PSI	

Quantum ESPRESSO
Diagram



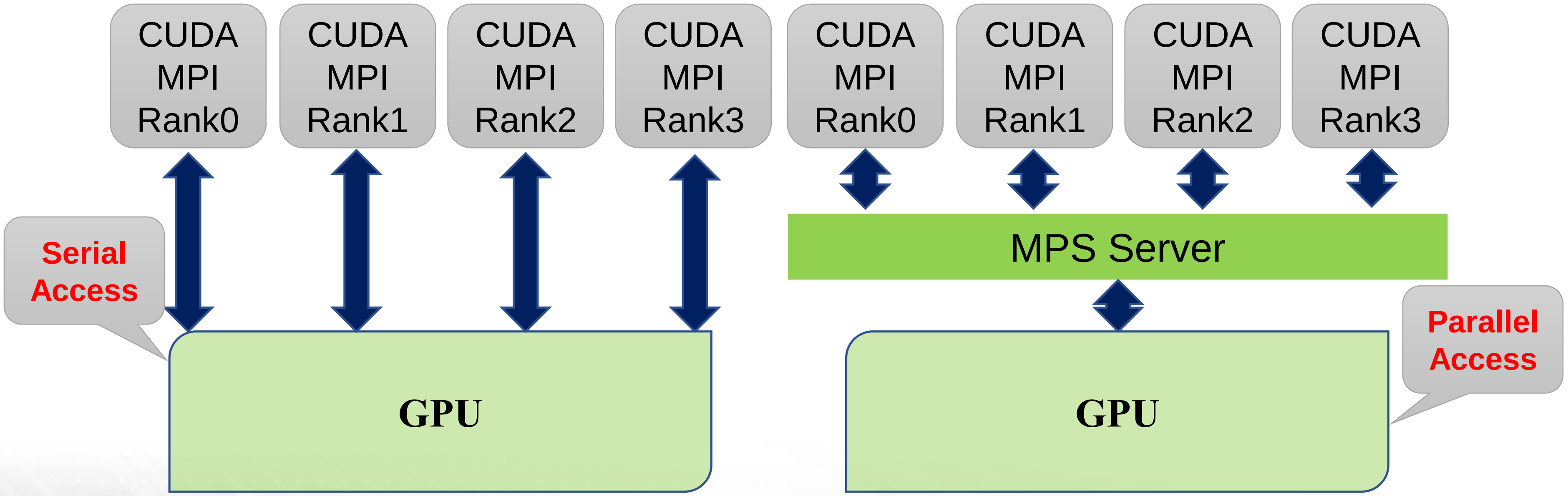
Quantum ESPRESSO
Optimization

Restructuring the CUDA kernel functions to gather scattered GPU memory access.

```
qecheck_cufft_call( cufftExecZ2Z( p_global, (cufftDoubleComplex *) psic_D, … , CUFFT_FORWARD ) );

tscale = 1.0 / (double) ( size_psic );

cublasZdscal(qecudaHandles[ 0 ] , size_psic, &tscale, (cufftDoubleComplex *) psic_D, 1);

kernel_save_hpsi<<< grid2_hpsi, threads2_hpsi, 0, qecudaStreams[ 0 ] >>>(… , hpsi_D, (double *)
psic_D, … );
```

Quantum ESPRESSO
CUDA Multi Process Service

- ✓ Due to the high CPU core count per node, many MPI processes share a limited number GPU resource even though multiple GPU devices are equipped on the node.
- ✓ CUDA MPS is an efficient way to share GPUs on node.



Quantum ESPRESSO
Result

This example illustrates how to use pw.x (PW) and ph.x (PHonon) to calculate phonon frequencies at Gamma and X for Si and C in the diamond structure and for fcc-Ni.

GPU: K40 (Four devices)
# of MPI processes : 4
# of k points : 4
Output file : si.phXsingle.out
FFT size : 96 X 96 X 96

(sec)	H_PISQ()	PHonon
MPI+GPU	27.31	194.15
MPI only	374.79	667.81
Speed-up	**13.72**	**3.44**

Quantum ESPRESSO
Result

The example is divided on two parts, the first one is an example of a molecule (CO2) and the second one is a solid (ZnO-Wurtzite) which are computed in a similar way, but with some small differences. With metals the occupation is determined by smearing and as it is a solid there should be more k-points. For the phonon calculation, the "epsil" should be set to .false. for ZnO, otherwise the code will not be able to compute the dielectric constant and will crash. But it can be set to .true. in the case of CO2.

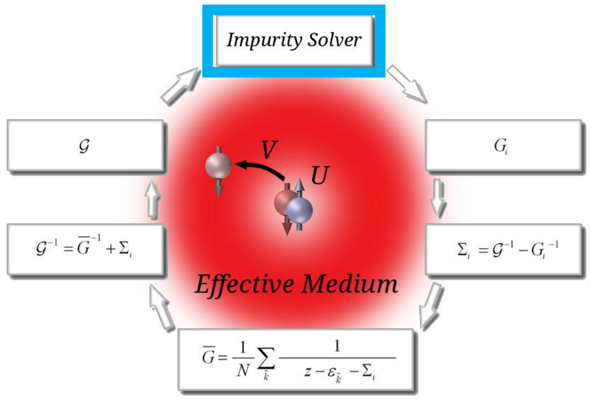
GPU: K40 (Four devices)
# of MPI processes : 8
# of k points : 8
Output file : zno.ph.out
FFT size : 75 X 48 X 48

(sec)	H_PISQ()	PHonon
MPI+GPU	112.60	262.19
MPI only	477.12	633.71
Speed-up	**4.24**	**2.42**

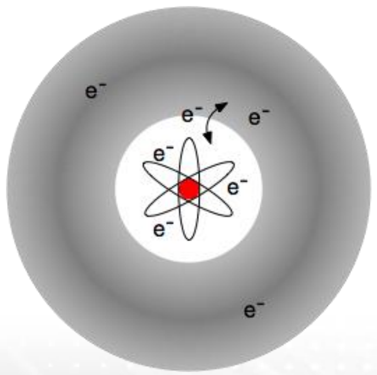
CASE III
Continuous Time – Quantum Monte Carlo (CT-QMC)

CT-QMC
Impurity Model

Dynamical Mean Field Theory (**DMFT**) is a method to determine the electronic structure of strongly correlated materials. In such materials, the approximation of independent electrons, which is used in density functional theory (DFT) and usual band structure calculations, breaks down. Dynamical mean-field theory, a non-perturbative treatment of local interactions between electrons, bridges the gap between the nearly free electron gas limit and the atomic limit of condensed-matter physics.


Impurity Solver
$\mathcal{G}$
G_i
V
U
$\mathcal{G}^{-1} = \bar{G}^{-1} + \Sigma_i$
$\Sigma_i = \mathcal{G}^{-1} - G_i^{-1}$
Effective Medium
$\bar{G} = \frac{1}{N}\sum_{\bar{k}} \frac{1}{z - \varepsilon_{\bar{k}} - \Sigma_i}$

- Computing $\mathbf{O} = \sum_c O(c)\,\omega(c)/\,Z$ where $O(c) = \langle c|\hat{A}e^{-\beta\hat{H}}|c\rangle/\omega(c)$, $\mathrm{Z} = \sum_c \omega(c)$, and $\boldsymbol{\omega(c)} = \langle \boldsymbol{c}|\boldsymbol{e^{-\beta\hat{H}}}|\boldsymbol{c}\rangle$
- $\mathbf{O} = \sum_c \boldsymbol{O(c)}\,(\frac{\boldsymbol{\omega(c)}}{\boldsymbol{Z}}) = \sum_c \boldsymbol{O(c)}\,\boldsymbol{\rho(c)}$ where $\rho(c) = \frac{\boldsymbol{\omega(c)}}{\boldsymbol{Z}}$
- O is the expectation of O(X) with the p.d.f. $\rho(x) \Longrightarrow$ **O = E[O(X)]**
- **O = E[O(X)]** $= \lim_{n\to\infty} \frac{1}{N}\sum_c O(c) \simeq \frac{\mathbf{1}}{\boldsymbol{N}} \sum_c \boldsymbol{O(c)}$



Impurity Model

$$\langle O\rangle = \sum_c O(c)p(c) \qquad c := (\alpha_1, \alpha_2, \ldots, \alpha_{2k})$$

$$p(c) = \mathrm{Tr}[\mathbf{F}_{\alpha_1}\mathbf{F}_{\alpha_2}\cdots\mathbf{F}_{\alpha_{2k}}] \times \mathrm{Det}\mathbf{M}(c) \quad \mathbf{F}_{\alpha_i} \in \mathbb{R}^{M\times N}$$

- **Random sampling from $\boldsymbol{\rho}$ is the main goal of the code.**
- **Metropolis-Hasting alg. is used for the sampling.**

CT-QMC
Metropolis-Hasting Algorithm

- The Metropolis-Hasting Algorithm for sampling $\boldsymbol{\omega}(\boldsymbol{c})$
- Markov Chain: $c_1 \rightarrow \cdots \rightarrow c_n \rightarrow c_{n+1} \rightarrow \cdots$
- One iteration at step n

I. Proposal trial configuration c'

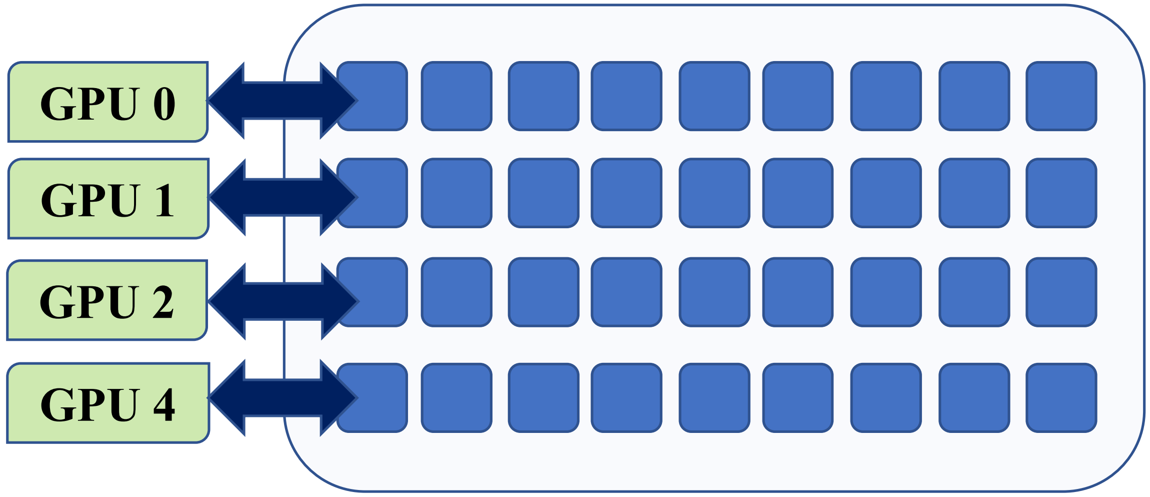
II. Compute $p = \frac{\boldsymbol{\omega}(\boldsymbol{c'})}{\boldsymbol{\omega}(\boldsymbol{c_n})}$

III. $c_{n+1} = \begin{cases} c' : Accept\ with\ probability\ p \\ c_n : Reject\ \ otherwise \end{cases}$

IV. Set n = n+1

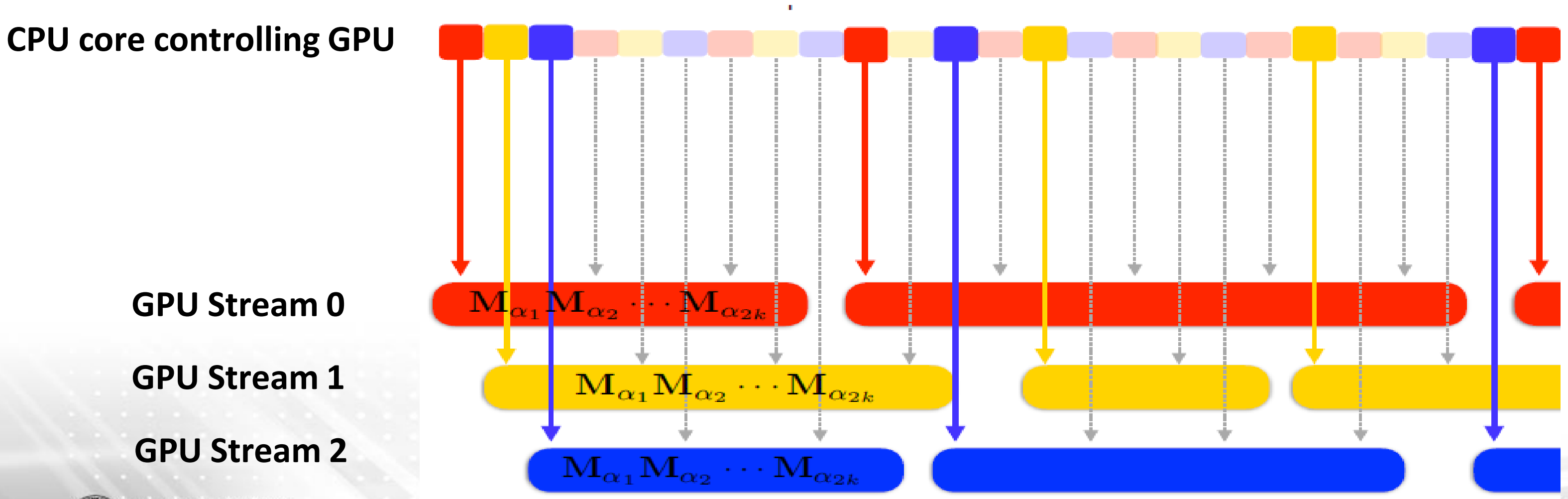
$$p(c) = \mathrm{Tr}[\mathbf{F}_{\alpha_1}\mathbf{F}_{\alpha_2}\cdots\mathbf{F}_{\alpha_{2k}}] \times \mathrm{Det}\mathbf{M}(c)$$

CT-QMC
CPU-GPU Concurrent Run



Speedup Test

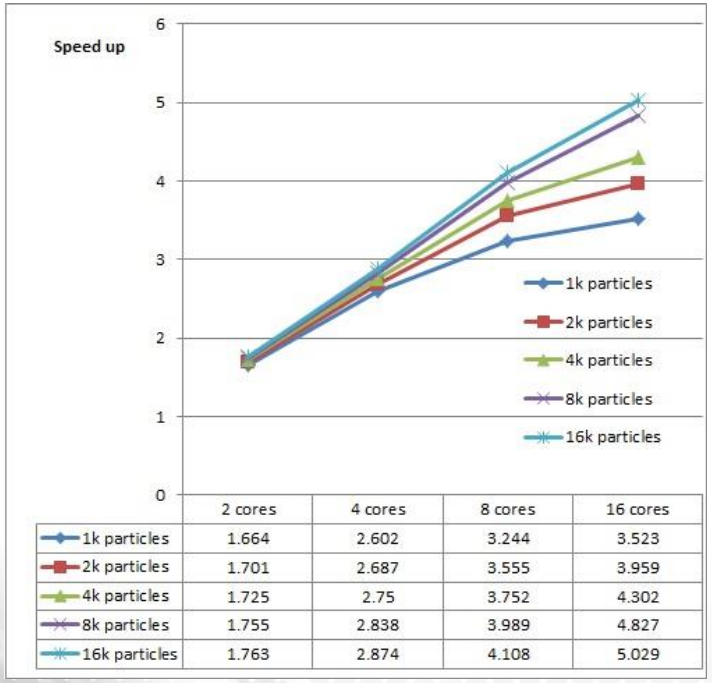
Computing Env.	Speedup
IC with K80	**3.5x**
IC with P100	**5x**
Titan (ORNL)	**5x**
SummitDev (ORNL)	**12.5x**


CPU core controlling GPU
GPU Stream 0
GPU Stream 1
GPU Stream 2
$M_{\alpha_1} M_{\alpha_2} \cdots M_{\alpha_{2k}}$
$M_{\alpha_1} M_{\alpha_2} \cdots M_{\alpha_{2k}}$
$M_{\alpha_1} M_{\alpha_2} \cdots M_{\alpha_{2k}}$

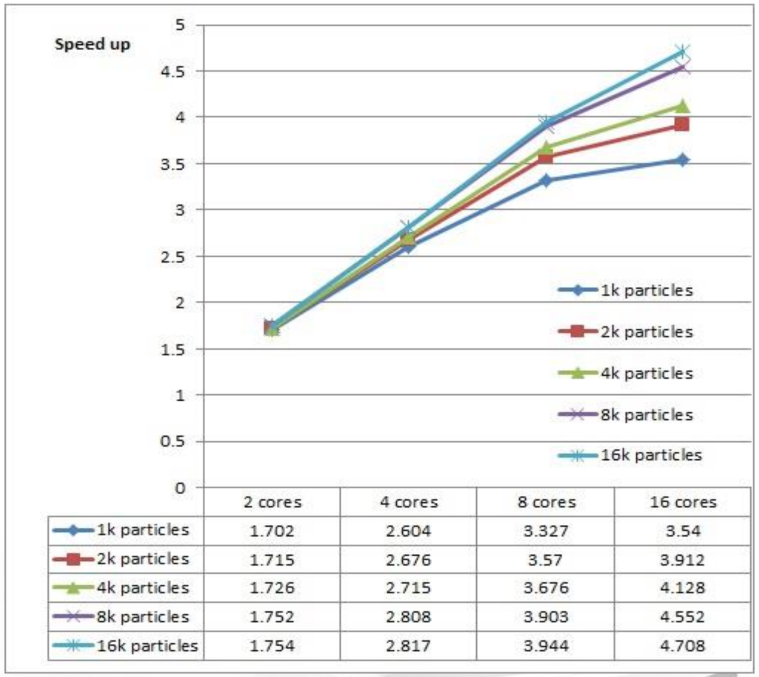
OTHERS

MAD-X

- Methodical Accelerator Design (version 10)
- Designing accelerators and testing beam behavior
- Widely used for lattice design in accelerators throughout the world.
- Currently studying space charge effects for future RHIC Upgrades.
- OpenMP parallelization is applied.
- Maximum speed-up is **5x**.



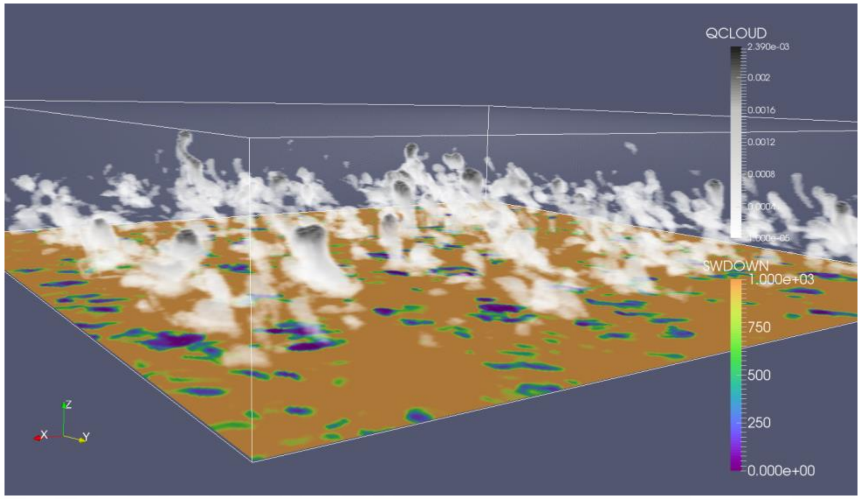
	2 cores	4 cores	8 cores	16 cores
1k particles	1.664	2.602	3.244	3.523
2k particles	1.701	2.687	3.555	3.959
4k particles	1.725	2.75	3.752	4.302
8k particles	1.755	2.838	3.989	4.827
16k particles	1.763	2.874	4.108	5.029

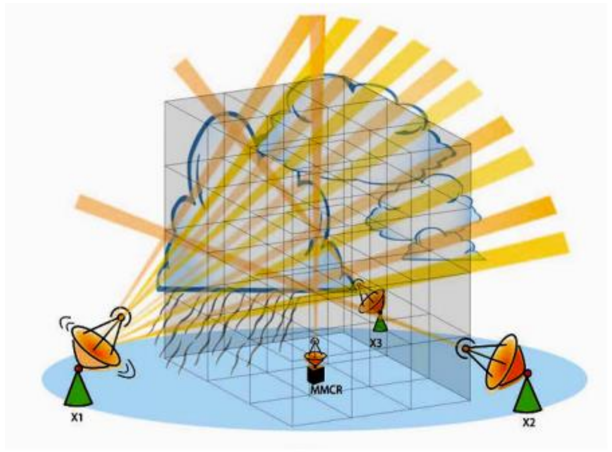


	2 cores	4 cores	8 cores	16 cores
1k particles	1.702	2.604	3.327	3.54
2k particles	1.715	2.676	3.57	3.912
4k particles	1.726	2.715	3.676	4.128
8k particles	1.752	2.808	3.903	4.552
16k particles	1.754	2.817	3.944	4.708

CR-SIM

- Acronym of Cloud Resolving Model Radar SIMulator.
- Generates a virtual (synthetic) view of what a radar would see if incorporated into an atmospheric model that resolves clouds.
- Model validation tool (creates virtual observation by radars.)
- Has world wide user community and the community is growing.
- **168x** speedup by restructuring I/O parts and applying OpenMP.

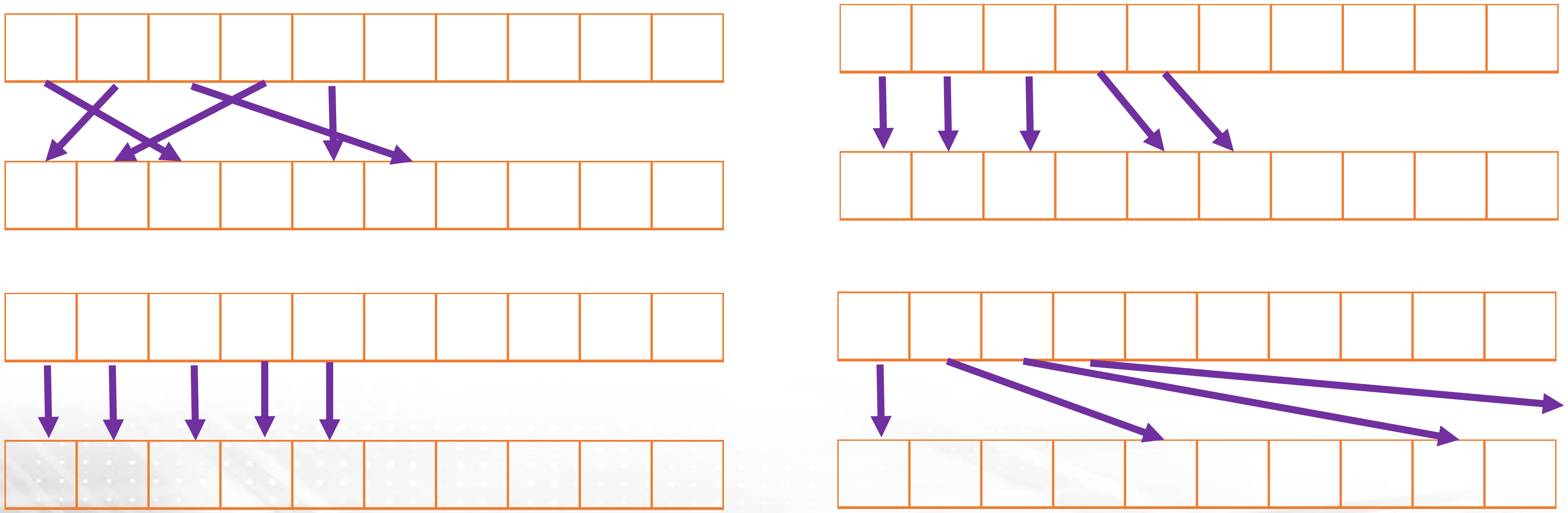




SUMMARY

SUMMARY
GPU Memory

- Minimize data transfer between GPU & CPU
- Minimize GPU global memory access
- Maximize GPU shared memory usage
- GPU Global Memory Coalescing
  - Every 128 byte (32 * 4 byte) successive memory can be accessed by a warp (32 threads) in a single transaction.

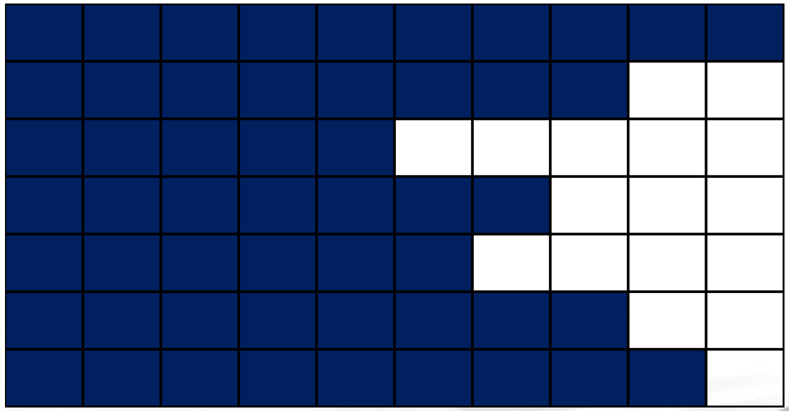


SUMMARY
Branching

```
For loop
  ⋮

  If then
    ⋮
  Else then
    ⋮
  End if

  ⋮
End for
```

```
If then
  ⋮
  For loop
    ⋮
  End for
Else then
  ⋮
  For loop
    ⋮
  End for
End if
```

```
__global__ void kerPropInit(…) {
  ⋮
  if (threadIdx.x == SOME_VALUE) {
    ⋮
  }
  ⋮
}
```



Ex. Boundary

SUMMARY
Restructuring

- ✓ No general rule. Totally depends on problem.
- ✓ Restructuring data structure and algorithms for massively parallel computing environment.
- ✓ This requires deep understanding of the domain science (at least the main algorithms and workflow of the code).

  Examples:

  From tiny many matrices To Huge one matrix

  Reordering temporal process

  Removing branches

- ✓ New algorithm for GPU or Heterogeneous System.