

Beyond Clusters: Adapting Lattice Optimization for Cloud and GPU

Kai Song

High Performance Computing Services
Lawrence Berkeley National Laboratory
ksong@lbl.gov



Overview

Overview:

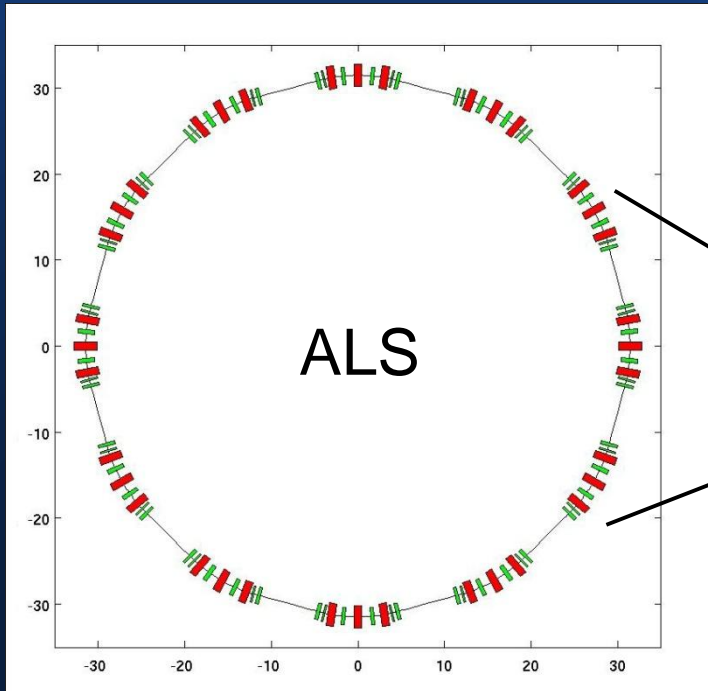
- Case study
- Evaluate computing options from a user's perspective
- Feasibility, cost, and performance

Outline:

- Physics and Application Background
- Cloud Adaptation
- GPU Adaptation
- GPU in the Cloud

Application Background

Multi-variable, multi-objective optimization problem



Genetic Algorithm fits the best.

Adjust the magnet knobs

Multi-Variables:
- quadrupole
- sextupole

Achieve better performance

Multi-objectives:
- emittance
- beta function
- dynamic aperture
- beam lifetime

Application Background

Genetic Algorithm Implementation

- Use MPI to establish a Master/Slave model

Master performs genetic operations and distributes tasks to slaves

Slave evaluate the lattice properties and send the result back to the master

- Pseudo code (Master)

1: Randomly generate the first generation

2: Evaluate the first generation

3: Sort the first generation

4: Repeat

5: select parent to generate child (cross over)

6: mutate child

7: evaluate child

8: merge the parent and child

9: sort the mixed population

10: select the first half of the mixed populations

11: Until reach maximum generation

Distribute to Slaves

(Embarrassingly parallel)

Distribute to Slaves

HPC Cloud Adaptation

Implemented for running on local data center

- Shared resources
 - Number of nodes are limited
 - Queuing time may be long
- Turn around time for research development
- Is it worthwhile to purchase own cluster?



HPC Cloud Adaptation

Cloud Offering

Cluster Compute Instance (CCI)

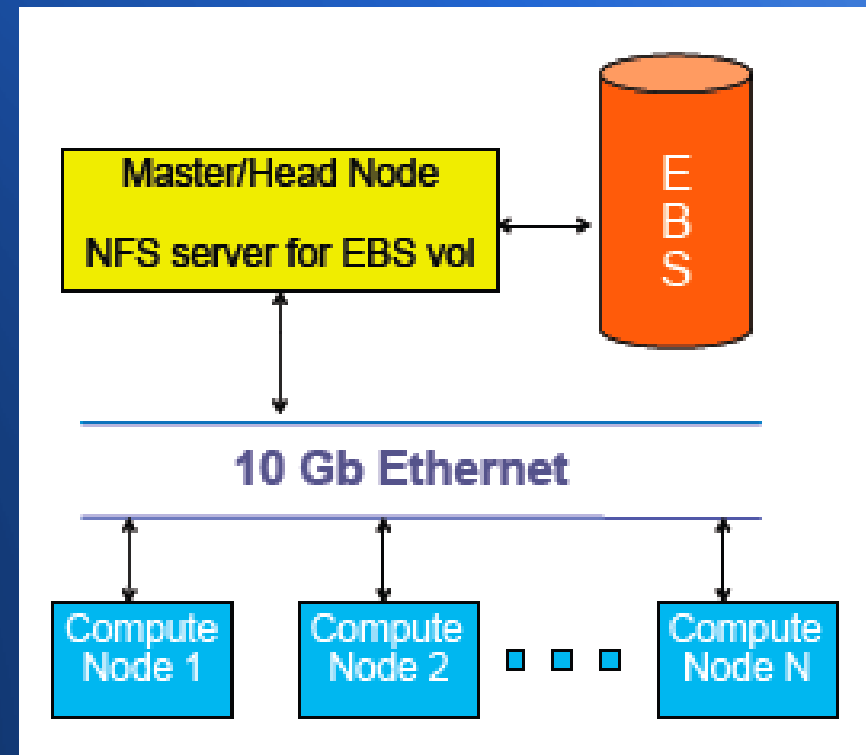
- Recently introduced by Amazon EC2
- Available only from US-EAST region today
- Pre-defined architecture & Hardware specification
- Instance specifications:
 - Dual Quad core Intel Nehalem processors
 - 23 Gb memory
 - 10Gb Ethernet
- HVM (Hardware Virtual Machine) Virtualization
- Hardware not shared with other EC2 instances
- On demand \$0.20/core-hour



HPC Cloud Adaptation

Cluster Configuration

- Managed by AWS Console and Command line API
- NFS mounted EBS volume
- Typical small/medium HPC cluster
- node start/terminate script
- Same placement group will make sure nodes are close by. (same rack)



HPC Cloud Adaptation

Comparison (specification)

	EC2	LRC	Mako	LR2
CPU Arch	X5570	E5430	E5530	X5650
CPU Freq (GHz)	2.93	2.66	2.40	2.66
Cache (MB)	8	12	8	12
HT	On	Off	Off	Off
Interconnect ¹	10	20	40	40
Virtualization	On	Off	Off	Off
Cores/Node	16	8	8	12
Memory/Node	23 GB	16 GB	24 GB	24 GB

¹. Gb/s

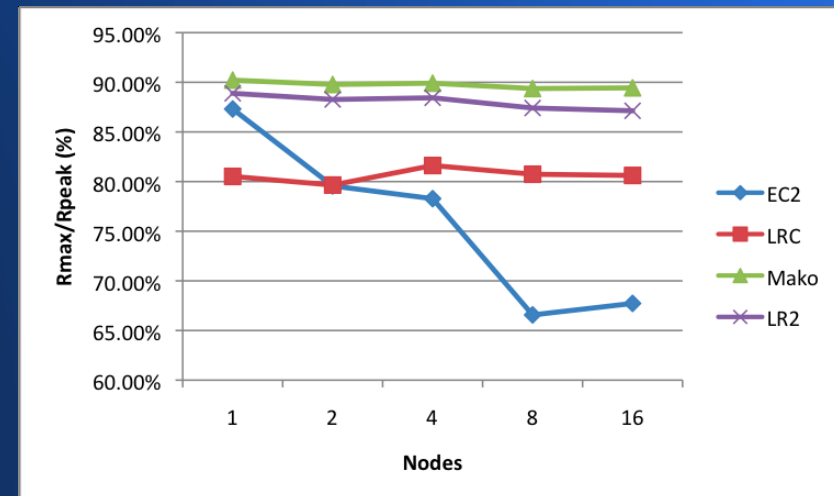
HPC Cloud Adaptation

Comparison (performance)

	EC2	LRC	Mako	LR2
Time (secs)	679	857	724	566

General Benchmark

- HPL benchmark
- node communication causes performance limitation



HPC Cloud Adaptation

Comparison (cost)

- Apple to Apple comparison is hard
- Facility cost varies greatly
- Electricity and Cooling depend on local rates and data center efficiencies
- Effective cost per core/hour depends on local cluster hardware and utilization

Short Conclusion

- Cloud HPC is feasible, and on-demand
- Requires some expertise in System Administration to setup quickly, and costly

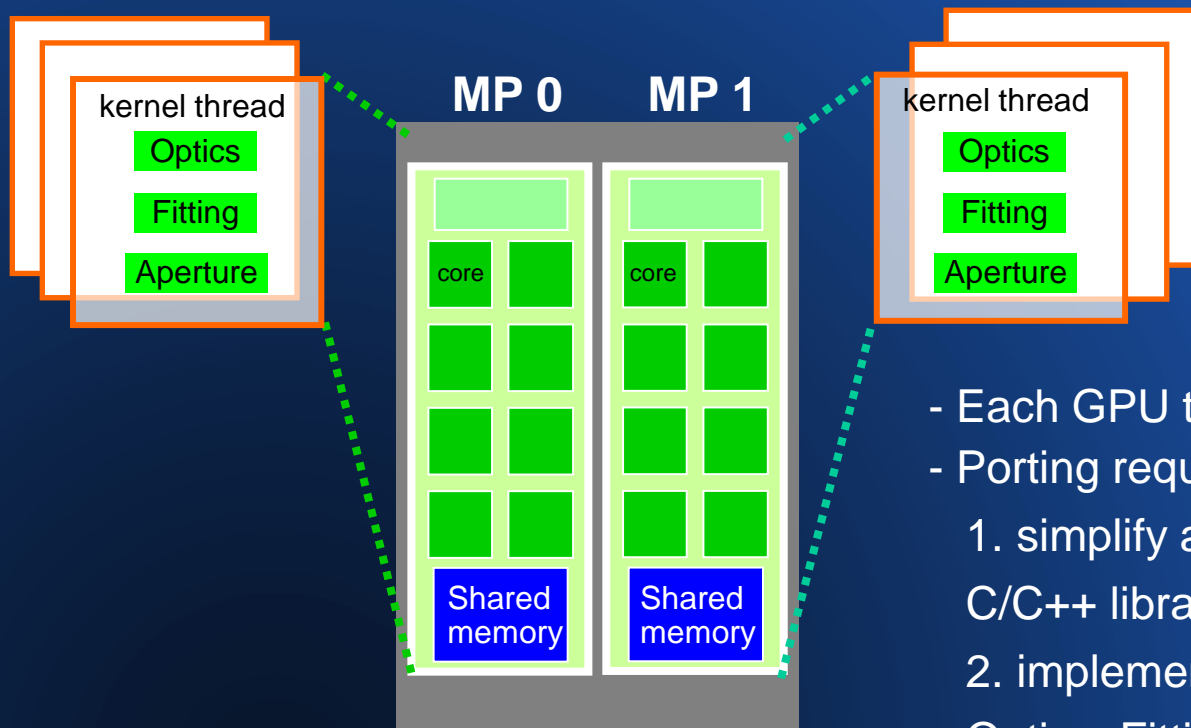
	EC2	Local Cluster
Hardware	X	X
Facilities	X	X
Electricity & Cooling	X	X
HW effort	X	X
SW effort	Not provided	X

\$0.20

\$???

GPU Adaptation

Initial Approach (Optimize Dynamic Aperture Only)



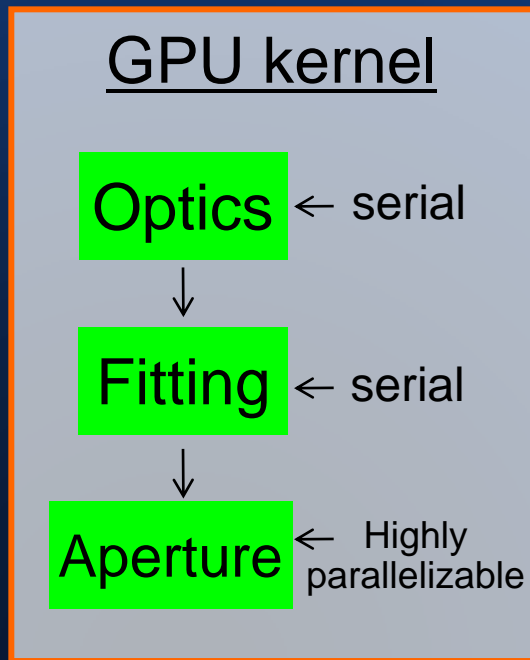
- Each GPU thread takes a MPI process
- Porting requires two major work:
 1. simplify and rewrite data structure of C/C++ library to fit in GPU
 2. implement slave process components: Optics, Fitting, and Aperture
- Kernel size limitation

Courtesy to Joshua A. Anderson, Virtual School of Computational Science and Engineering

GPU Adaptation

Further Development

Original approach:



Serial, push back to CPU

Parallel in GPU

New approach:

Optics(CPU)

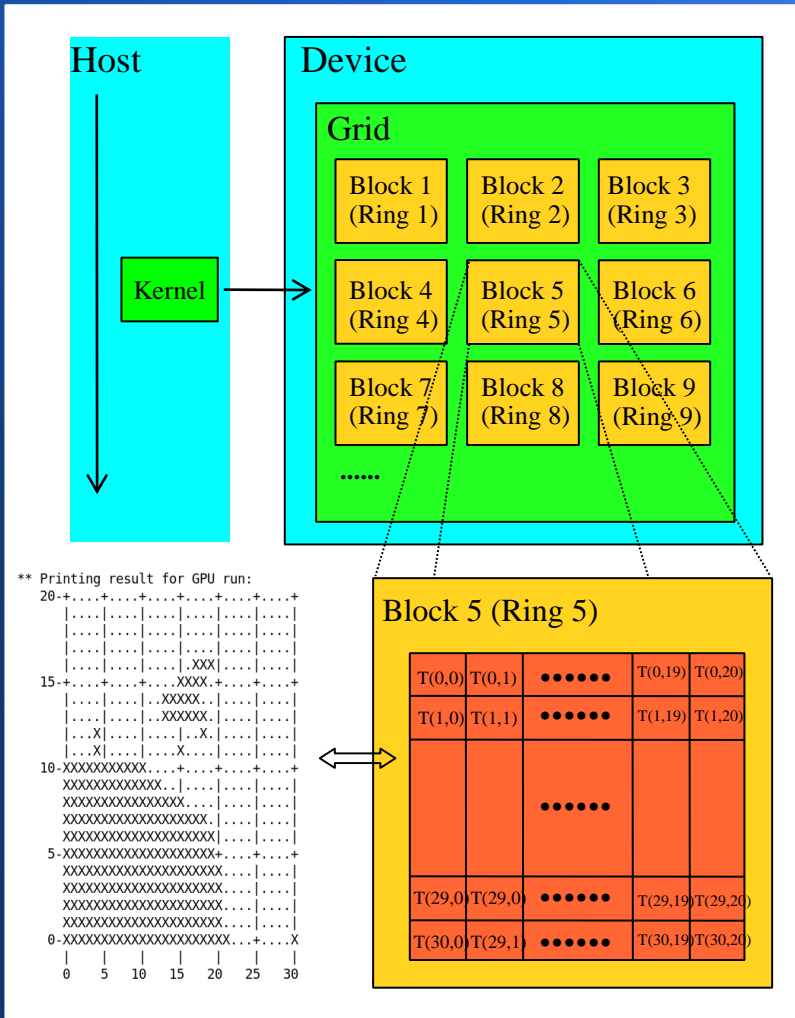
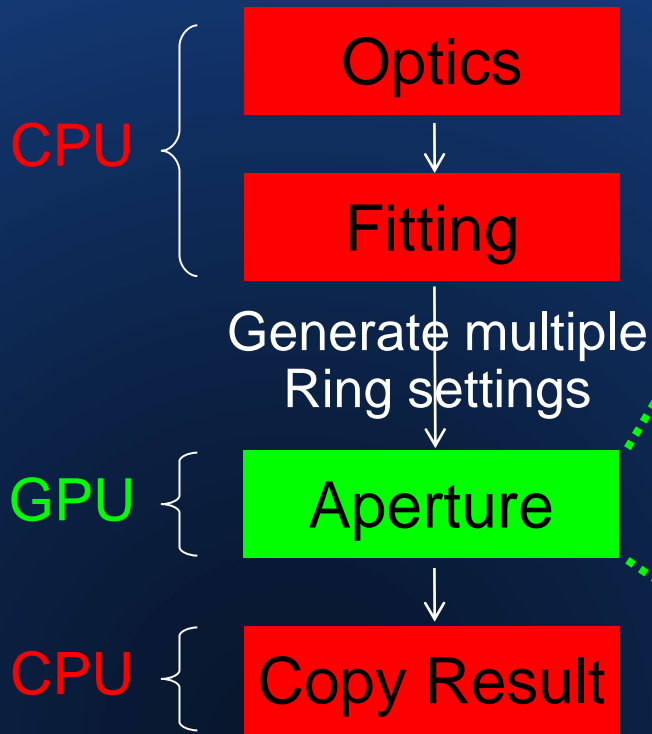
Fitting(CPU)

Aperture(GPU)

kernel thread
calculate one grid point

GPU Adaptation

Implementation



GPU Adaptation

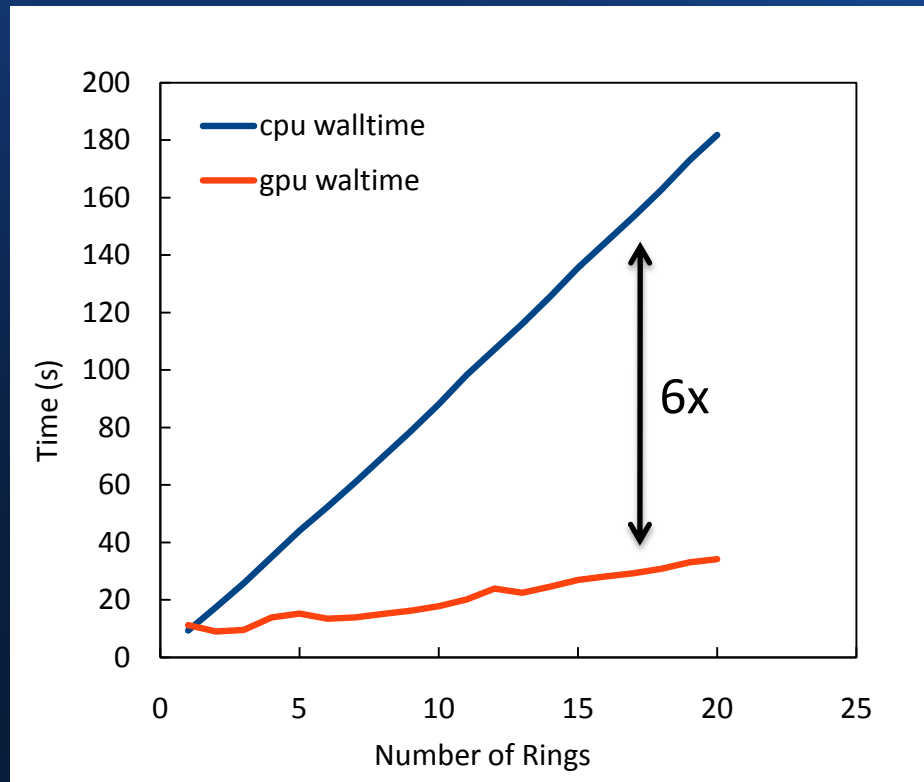
Results (GPU architecture)

- Tesla C2050, Fermi
- 14 multiprocessor
- 32 cores/MP
- total of 448 cores
- threads need to be small and compact
- Cache and Shared Memory Size: 64KB

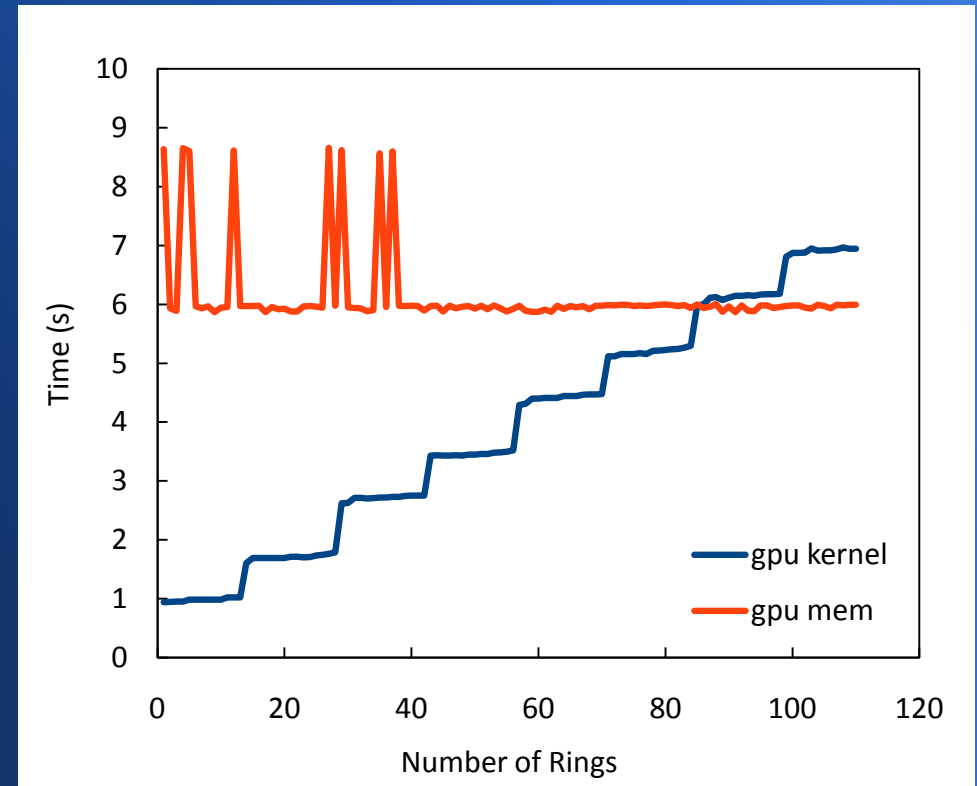
GPU Adaptation

Results

Walltime Comparison



GPU Kernel Execution



GPU Adaptation

Lessons Learned

Porting existing C/C++ library to GPU is not trivial

- dealing with data structures with many pointers/classes
- difficult without having GPU architecture in consideration
- limitation of GPU kernel size

Re-evaluate the original data structure and algorithm

- due to architectures difference between GPU and CPU
- Somebody with a comprehensive insight of the library is extremely helpful
- optimize specifically for GPU



Cloud GPU

EC2 Cluster GPU Instance

- CPU architecture same as Cluster Compute Instance
- 2 x Nvidia Tesla “Fermi” M2050 GPUs
- 22GB memory
- API name: cg1.4xlarge
- On demand \$2.10/instance-hour
- Not available for Windows OS
- Very close to local GPU used



Cloud GPU

GPU Instance Configuration

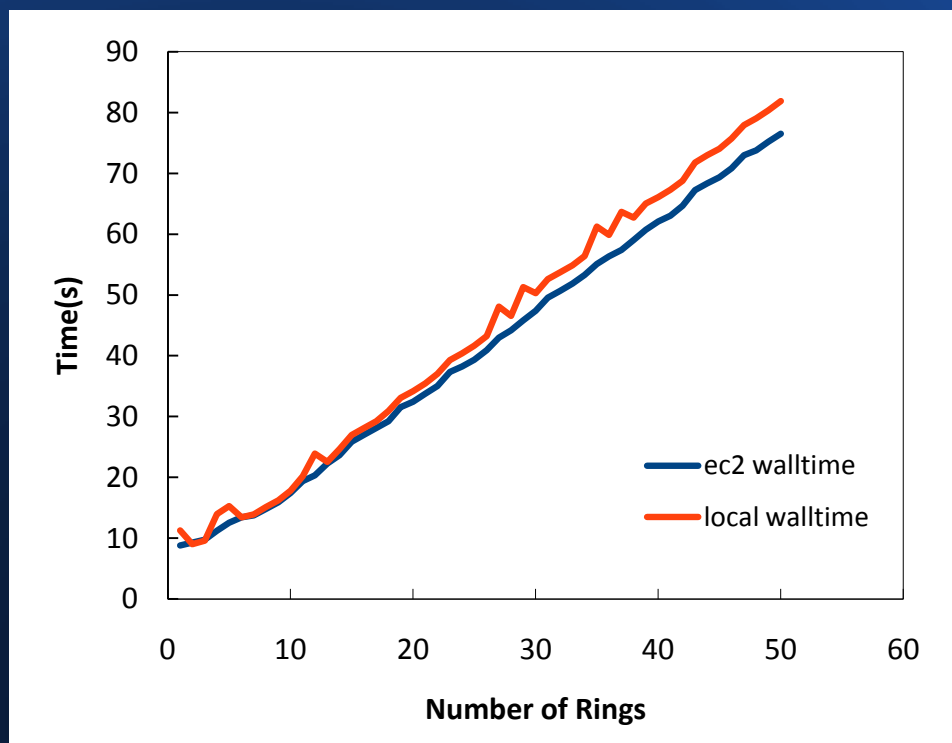
- Very straight forward to set up
- CUDA is already available
- yum install gcc-c++
- compile the code
- ready to go



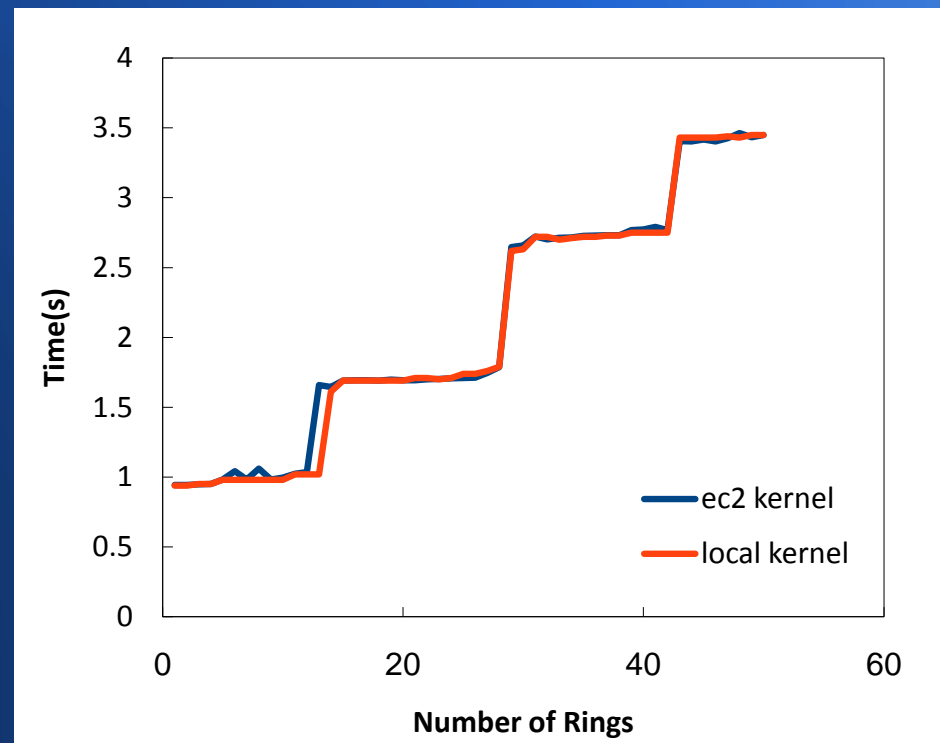
Cloud GPU

Comparison

Walltime Comparison



Kernel Comparison



Conclusion

A case study using Lattice Optimization Application

- CPU intense, low data processing and low communication

EC2 Cloud HPC

- Feasible and adequate performance
- On-demand feature is very attractive
- data and network intensive applications may suffer

Porting to GPU

- Porting existing C/C++ code is not trivial
- Understand data structure and algorithm
- Identify the suitable part to be parallelized

EC2 Cloud GPU

- Performance is almost identical to local GPU
- Good alternative for local GPU



Thank You

Questions?

