中国科学院超级计算中心
Supercomputing Center of CAS

Large Scale Heterogeneous Scientific Computing in SCCAS:

Practice and Prospects

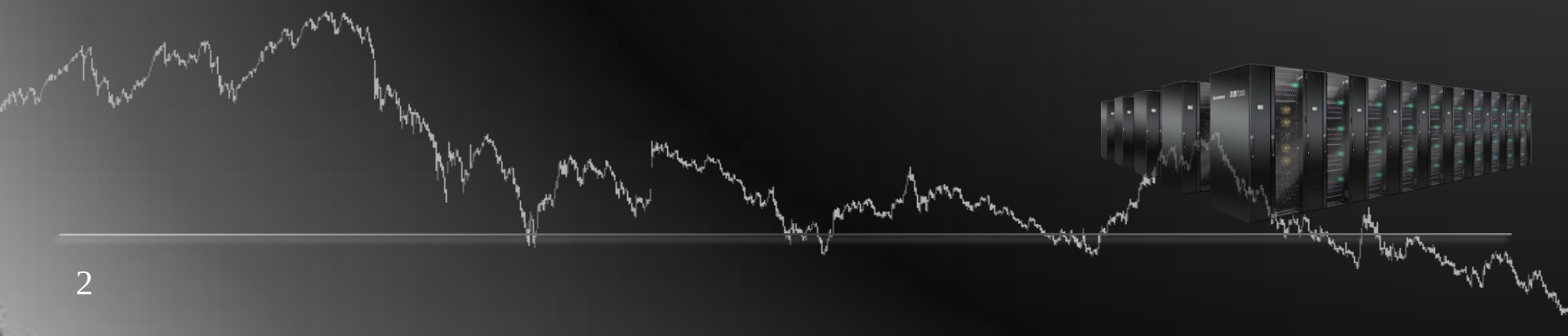
CAO Zongyan

zycao@sccas.cn

2012-9-5

Outline

- Brief introduction
- Two cases with details
- Other practices for scientific applications in SCCAS
- Summary and future considerations



Motivation of using accelerators

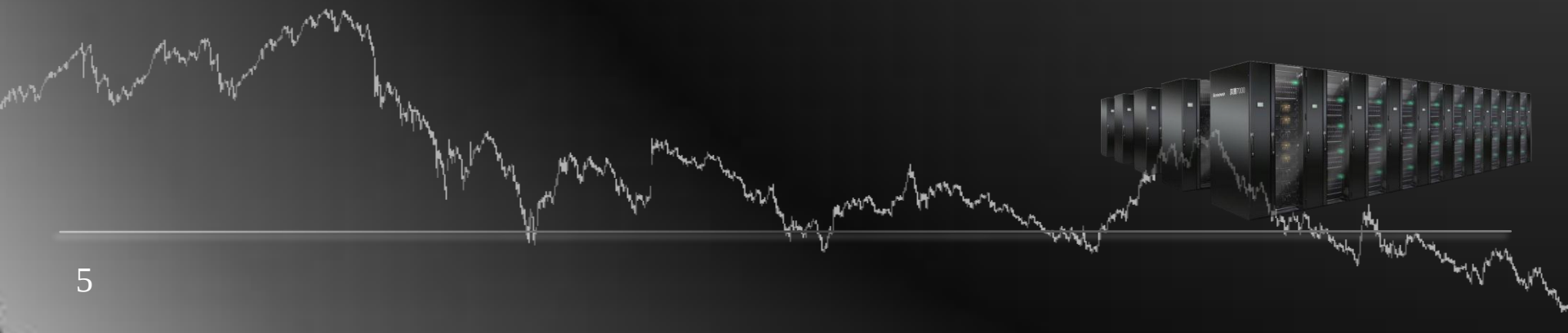
Some basic specifications

	CPU		Accelerator	
	Sandy Bridge-EP Xeon E5-4650	**Interlagos** Opteron 6284SE	**Kepler** Tesla K20	**Knights Corner** Xeon Phi
Vender	Intel	AMD	NVIDIA	Intel
Peak Flops	172.8Gflops	172.8Gflops	>1Tflops	>1Tflops
Cores	8 cores 16 threads w/HT	8 FP modules 16 cores	15 SMX modules 2880 CUDA cores	>50 cores
Connection	QPI	HyperTransport	PCIe Gen3	PCIe
Memory	4-channel DDR3 max 1500GB	4-channel DDR3	384 bit GDDR5	GDDR5 8GB+
Power	130W	140W	250W(K10)	

- 4-way CPU node
  - 700Gflops, ~550W, 1.27Gflops/W
- Single CPU node + 1 accelerator
  - 170+1000Gflops, ~430W, 2.72Gflops/W

Challenges

- What to do
- Performance and scalability
  - High concurrency
  - Memory size
  - Data copy latency
  - Internode communication
- Real co-processing



Heterogeneous practices in SCCAS

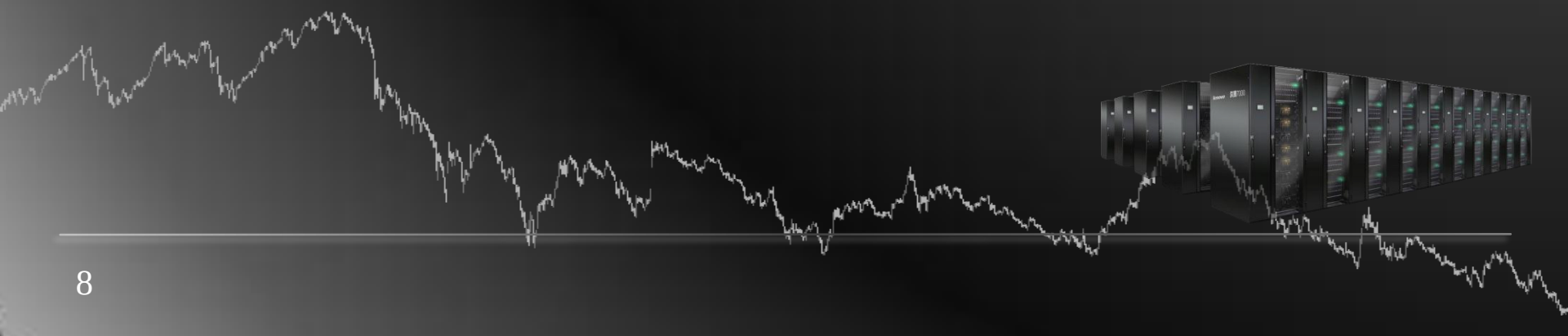
- Materials
  - SC-PEtot
- Cosmology
  - Widgeon
- Geology
  - KTM algorithm
- Bioinformatics
  - Linkage disequilibrium computation
  - InsPecT
- Fluid Mechanics
  - 2D Riemann problem
- Mathematics
  - HPSEPS

SC-PEtot: First large-scale GPU planewave DFT code

- Supported by
  - NSFC (61202054)
  - Knowledge Innovation Program of CAS (CNIC_ZR_201202)
- Collaborated with FDU and LBL
- People involved:
  - SCCAS (computer science)
    - WANG Long, JIA Weile, CAO Zongyan, FU Jiyun
  - FUDAN University (computational mathematics)
    - GAO Weiguo, WU Yue
  - Lawrence Berkeley National Laboratory (physics)
    - WANG Lin-Wang

Sweet spot for planewave DFT calculations

- 100 to 1000 atoms system
  - For >>1000 atoms, using linear scaling method (LS3DF)
- ab initio MD simulation for a few ns
  - State-of-the-art: 1-2 min per MD step



Progress of SC-PEtot

- Extreme redesign and optimization from the PEtot CPU code
  - CUDA + MPI implementation
  - All data in GPU
  - Local FFT
  - Hybrid precision computing
  - compressed data for communication optimization
  - Faster libraries
  - Performance analysis and prediction model

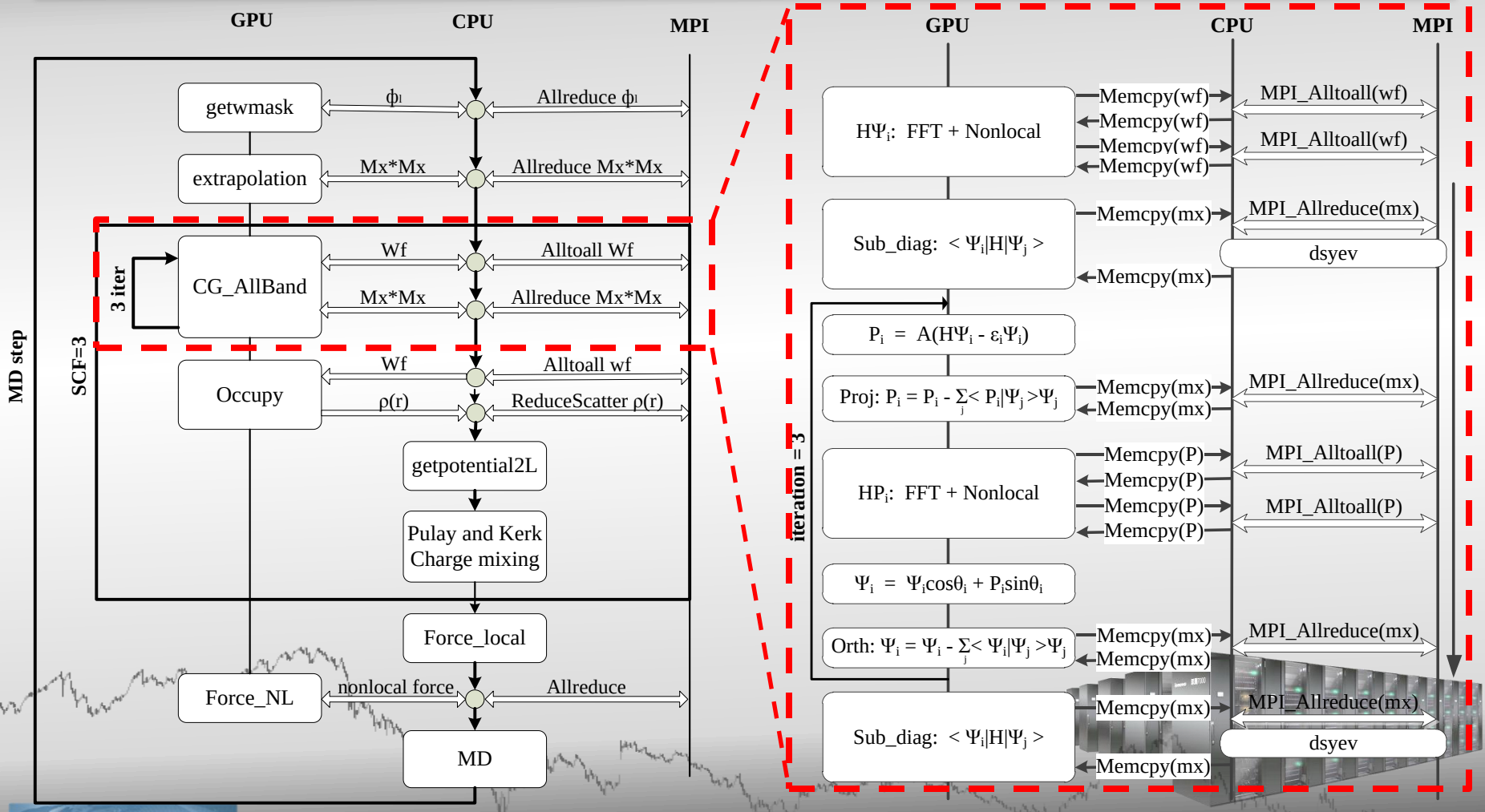
Wang, etc., **Large scale plane wave pseudopotential density functional theory calculations on GPU clusters** (SC'11)

Jia, etc. **The analysis of a plane wave pseudopotential density functional theory code on a GPU machine** (Comp. Phys. Comm., doi: 10.1016/j.cpc.2012.08.002)

Jia, etc. **Fast plane wave density functional theory molecular dynamics calculations on multi-GPU machines** (SC'12 poster)

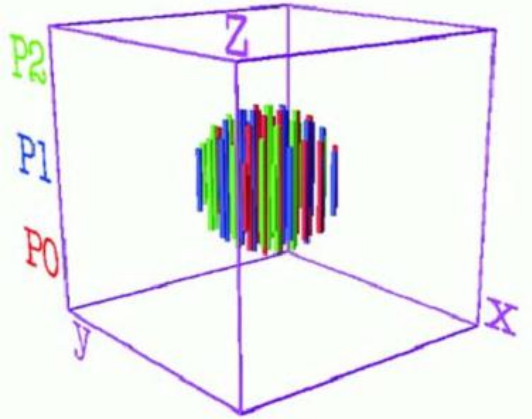
Talks on APS March Meeting 2012, NVIDIA GTC 2012, etc.

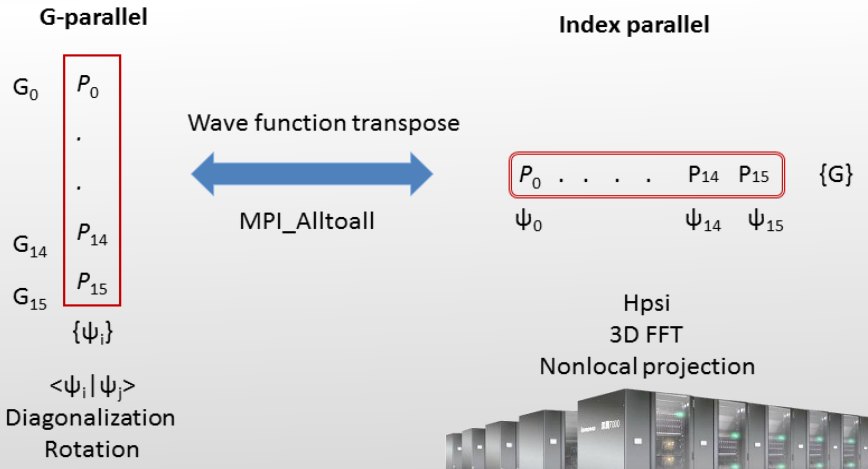
Diagram of SC-PEtot MD steps



Local FFT

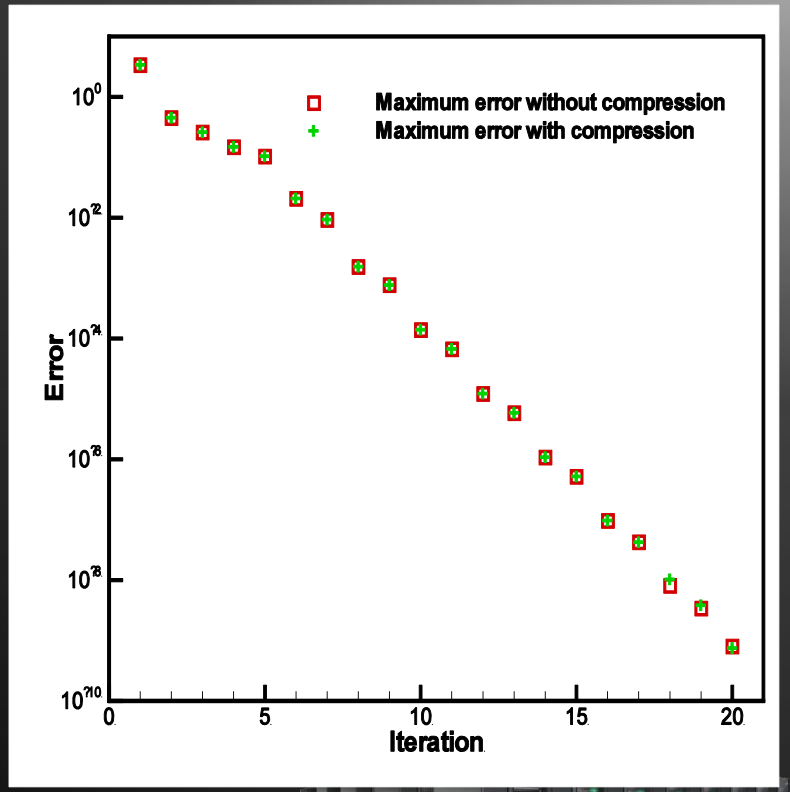
- G-space distribution to i-index distribution
  - Using all-to-all collective communication
- FFT done by CUFFT routine
  - 4x faster





Hybrid precision and data compression

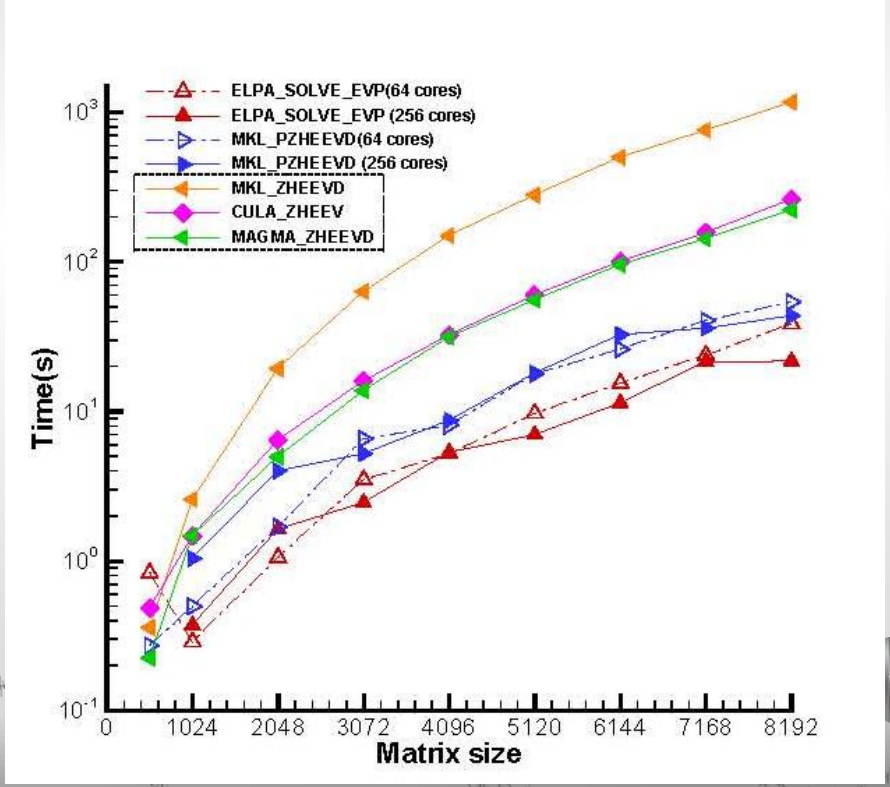
- Residual P
  - 1-6-9 data format
  - ¼ alltoall time
- FFT
  - Single precision
  - Half FFT time
- Subdiagonization
  - Single precision
  - Half allreduce time
  - Half eigenvalue solving time



Convergence of AB_CG

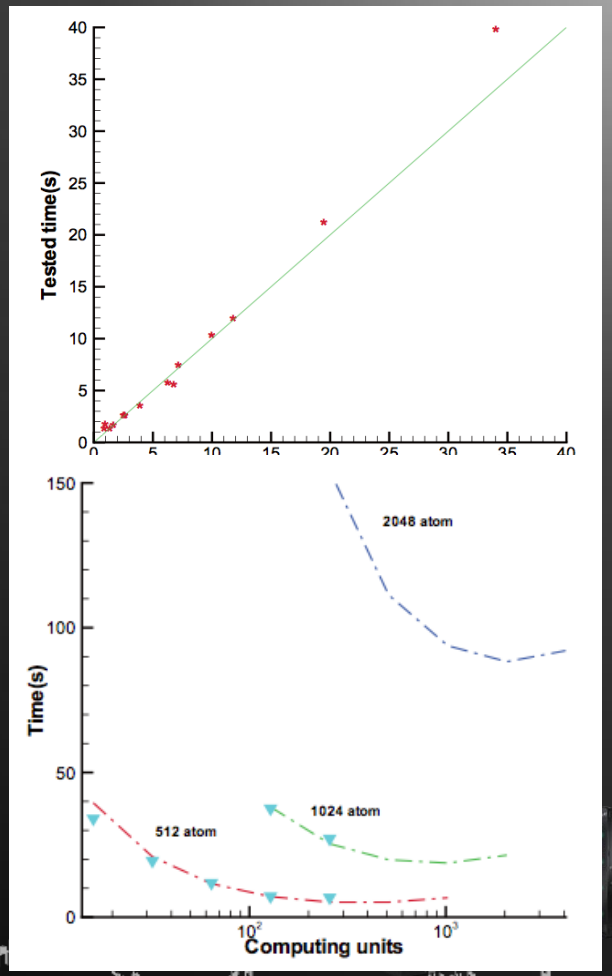
Library choosing

- MAGMA for BLAS routines and Cholesky decomposition
- ELPA (up to 64 CPU cores) for eigenvalue solving



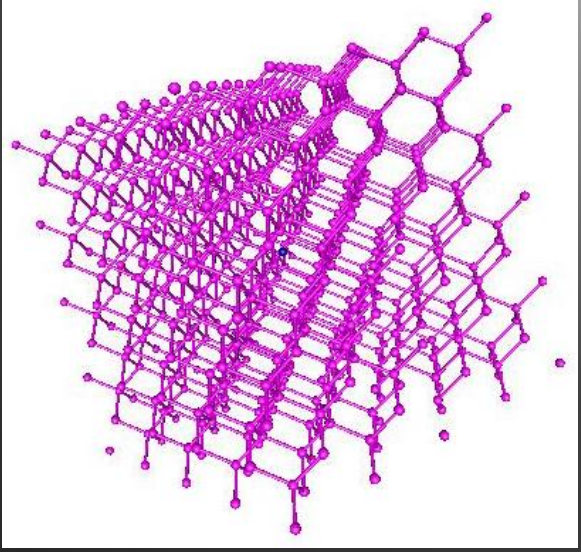
Performance analyzing equations

Time	Formula	Parameter
T_{FFT}	$\alpha_1 \cdot \frac{N \cdot logN \cdot M_x}{N_p}$	α_1=2.99e-9
$T_{Nonlocal}$	$\alpha_2 \cdot \frac{N_{atom} \cdot N_r \cdot M_x}{N_p}$	α_2=3.77e-8
$T_{Alltoall}$	$\alpha_3 \cdot (N_p - 1) \cdot \frac{N_G \cdot M_x}{{N_p}^2} + \beta_3 \cdot (N_p - 1)$	α_3=1.73e-7 β_3=2.92e-3
$T_{Allreduce}$	$\alpha_4 \cdot (2N_p - 1) \cdot \frac{{M_x}^2}{N_p} + \beta_4 \cdot (2N_p - 1)$	α_4=2.78e-7 β_4=6.73e-4
T_{Wf_copy}	$\alpha_5 \cdot \frac{N_G \cdot M_x}{N_p}$	α_5=1.50e-7
T_{Mx_copy}	$\alpha_6 \cdot {M_x}^2$	α_6=3.25e-7
T_{CUBLAS}	$\alpha_7 \cdot \frac{N_G \cdot {M_x}^2}{N_p}$	α_7=3.81e-10
$T_{Precond\&Line}$	$\alpha_8 \cdot \frac{N_G \cdot M_x}{N_p}$	α_8=7.33e-9
$T_{Compress}$	$\alpha_9 \cdot \frac{N_G \cdot M_x}{N_p}$	α_9=5.85e-8
$T_{Malloc\&Free}$	$\alpha_{10} \cdot \frac{N_G \cdot M_x}{N_p} + \beta_{10}$	α_{10}=2.65e-8 β_{10}=9.23e-2
T_{Zheev}	$\alpha_{11} \cdot {M_x}^3$	α_{11}=7.61e-10



Test results

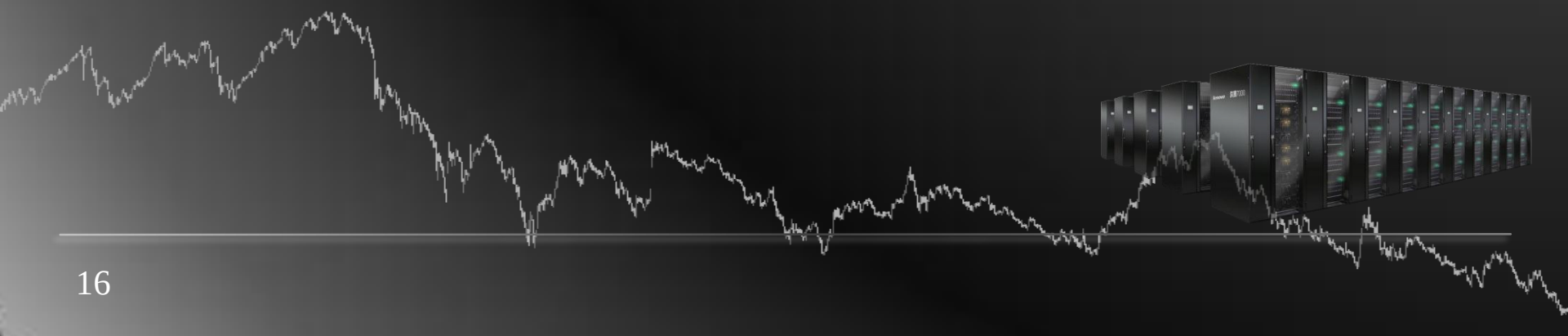
- Testing system:
  - 512 atom GaAsN bulk system
- Platform
  - OLCF Titan first phase
  - IPE Mole-8.5
- Results
  - 18x absolute speedup for MD steps



No. of CPU core	32×16	64×16	128×16	256×16
PEtot_CPU(Titan)	277s	223s	203s	216s
No. of GPU	32	64	128	256
PEtot_GPU(Titan)	31.6s	20.8s	13.2s	11.4s
PEtot_GPU(Mole-8.5)	33.1s	21.6s	13.7s	14.7s

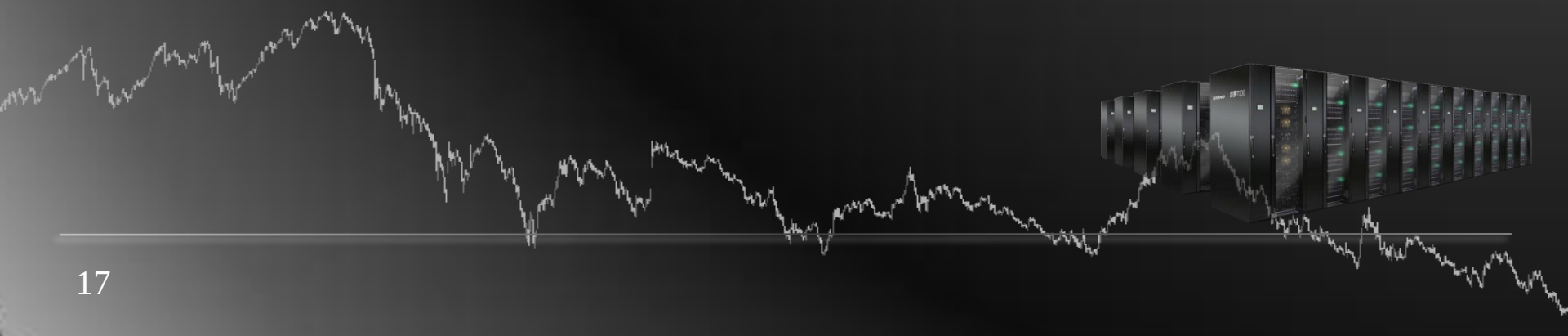
Conclusion of SC-PEtot work

- It is possible to use GPU to implement planewave DFT calculation with >10 times absolutely speedup.
- Rethink the whole parallelization scheme, and introduce new algorithm.
- Move most data and the calculation into the GPU.
- Hybrid precision computing may possibly benefit the performance, but it must be proved not influencing the correctness of the computation.
- Try new numerical libraries for algebra routines.



MIC enabling project: Widgeon

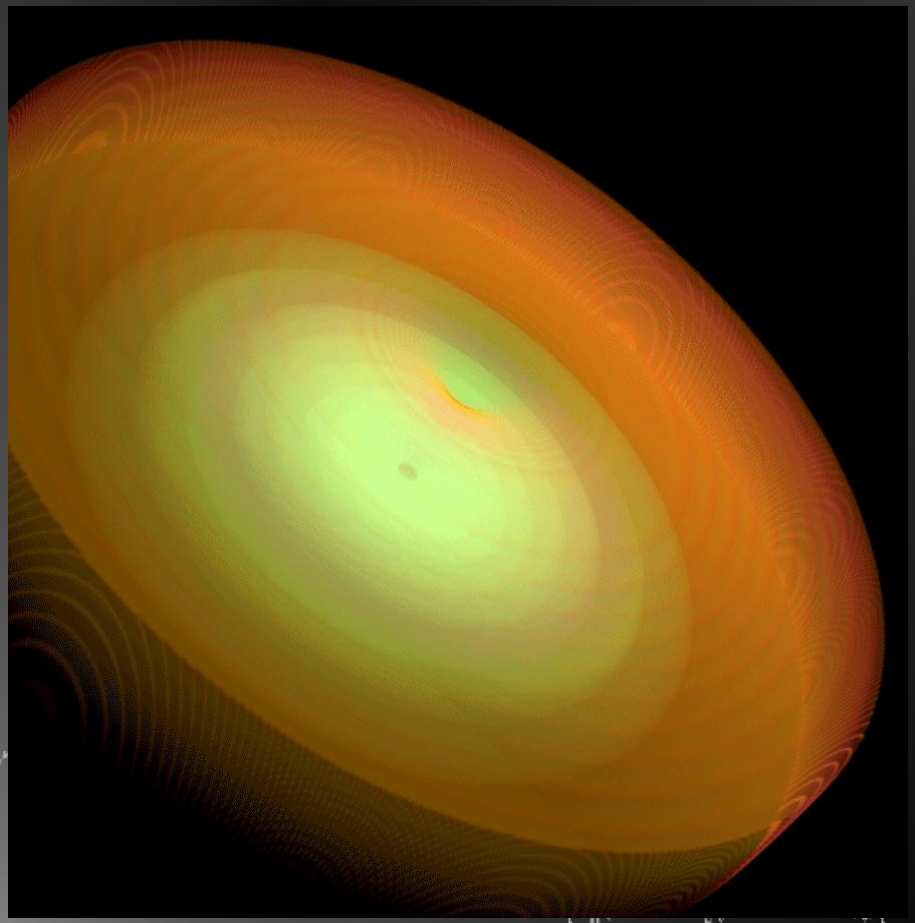
- Supported by and collaborated with Intel
- People involved:
  - SCCAS
    - CAO Zongyan, MENG Chen, WANG Long
  - Intel
    - ZHOU Shan

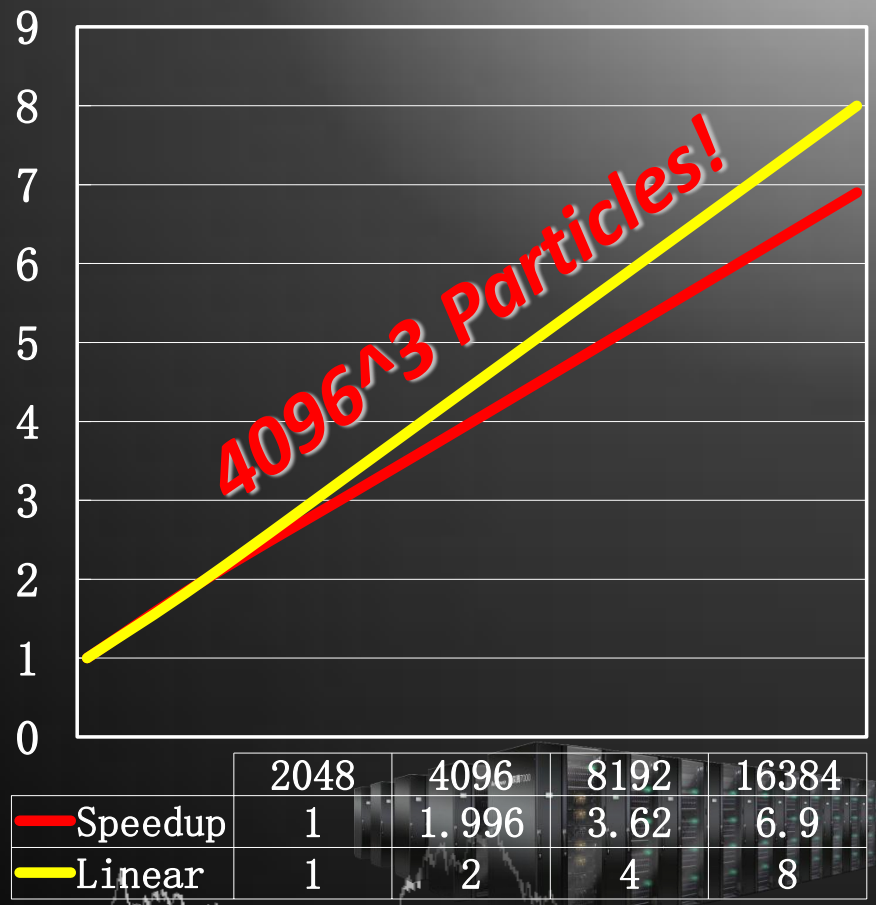


中国科学院超级计算中心
Supercomputing Center of CAS

CHANGES 2012

Widgeon: a Galactic wind simulation code





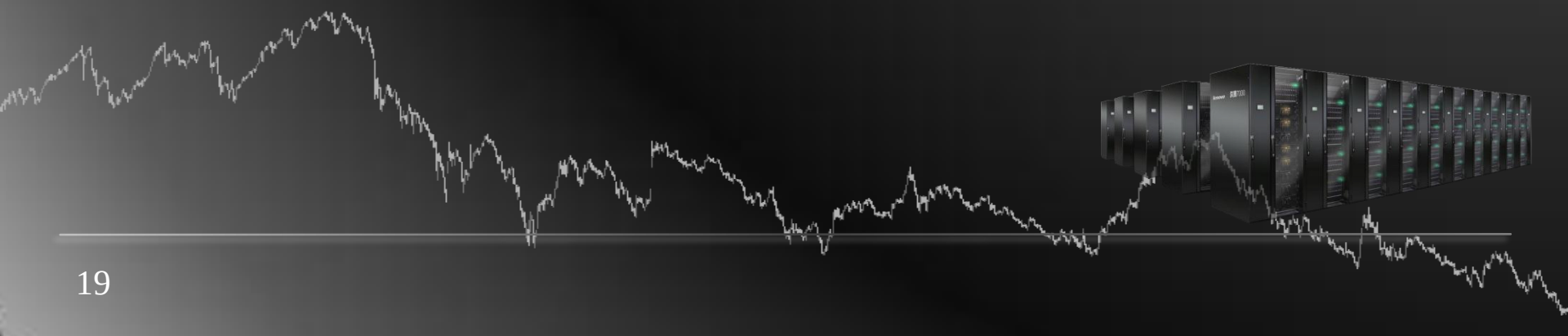
	2048	4096	8192	16384
Speedup	1	1.996	3.62	6.9
Linear	1	2	4	8

Potential for using accelerators

- >95% time to intensively calculate the equation:

$$u_t + f(u)_x + g(u)_y + h(u)_z = 0$$

- Exchange data only with neighbors, no collective communication
- High MPI scalability on clusters



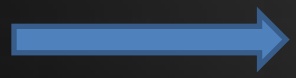
Some progress

- Running with stand alone MPI mode on KNC
  - Very bad performance at the beginning
  - Offload mode should get worse performance
- Vectorization ratio of the code is essential for the computing performance
  - Hardly optimizing the code, for better vectorization
  - Many optimization also made the code running faster on CPU
- Writing out a basic GPU code
  - Slower than one CPU core

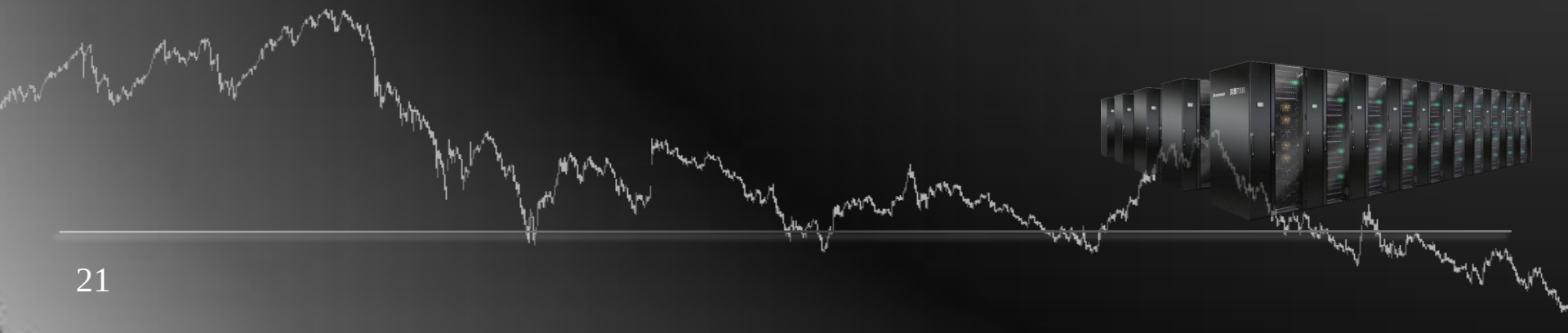
Code optimization details (1)

- 1. Eliminating the data dependencies in the loops to make the loops vectorizable for the compiler.

```
DO i=1, N
   ……
   calc out a temp num, store in t
   f(i) = f(i) + t
   ……
   calc out another temp num, store in t
   f(i) = f(i) + t
ENDDO
```



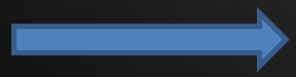
```
DO i=1, N
   ……
   calc out a temp num, store in t1
……
   calc out another temp num, store in t2
   f(i) = f(i) + t1 + t2
ENDDO
```



Code optimization details (2)

- 2. Rearranging and combining the loops to remove some temporary arrays.

```
DO i=1, N+1
   ……
   calculate f(i)
   calculate v(i) from f(i)
   ……
ENDDO

DO i=1, N
   ……
   g(i) = (f(i) + f(i+1)) * v(i+1)
   ……
ENDDO
```

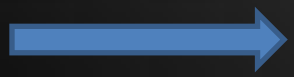


```
calculate f(1)
calculate v(1) from f(1)

DO i=2, N+1
   ……
   calculate f(i)
   calculate v from f(i)
   g(i-1) = (f(i-1) + f(i)) * v
   ……
ENDDO
```

Code optimization details (3)

- 3. Manually unrolling the small inner loops to make more calculation vectorized.

```
DO i=1, N
   ……
   lots of calculation
   DO j=1,5
      f(j,i) = u(j,i) * a
   ENDDO
   lots of calculation
   ……

ENDDO
```

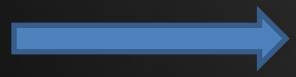


```
DO i=1, N
   ……
   lots of calculation
   f(1,i) = u(1,i) * a
   f(2,i) = u(2,i) * a
   f(3,i) = u(3,i) * a
   f(4,i) = u(4,i) * a
   f(5,i) = u(5,i) * a
   lots of calculation
   ……

ENDDO
```

Code optimization details (4)

- 4. Rearrange the rows and the columns of array in order to make memory access contiguous in nearby loop steps.

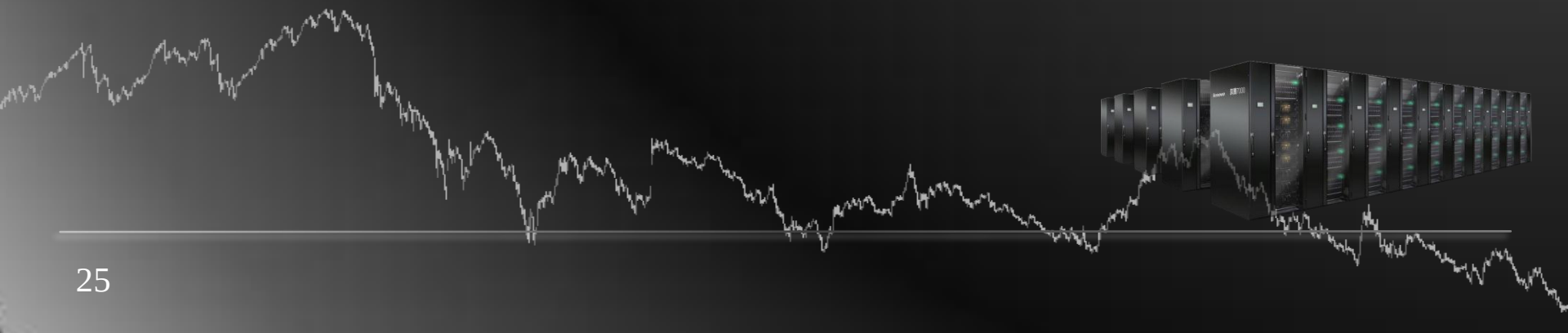
```
REAL,DIMENSION(3,N)::f,u
DO i=1, N
   ……
   f(1,i) = u(1,i) * a
   f(2,i) = u(2,i) * b
   f(3,i) = u(3,i) * c
……
ENDDO
```



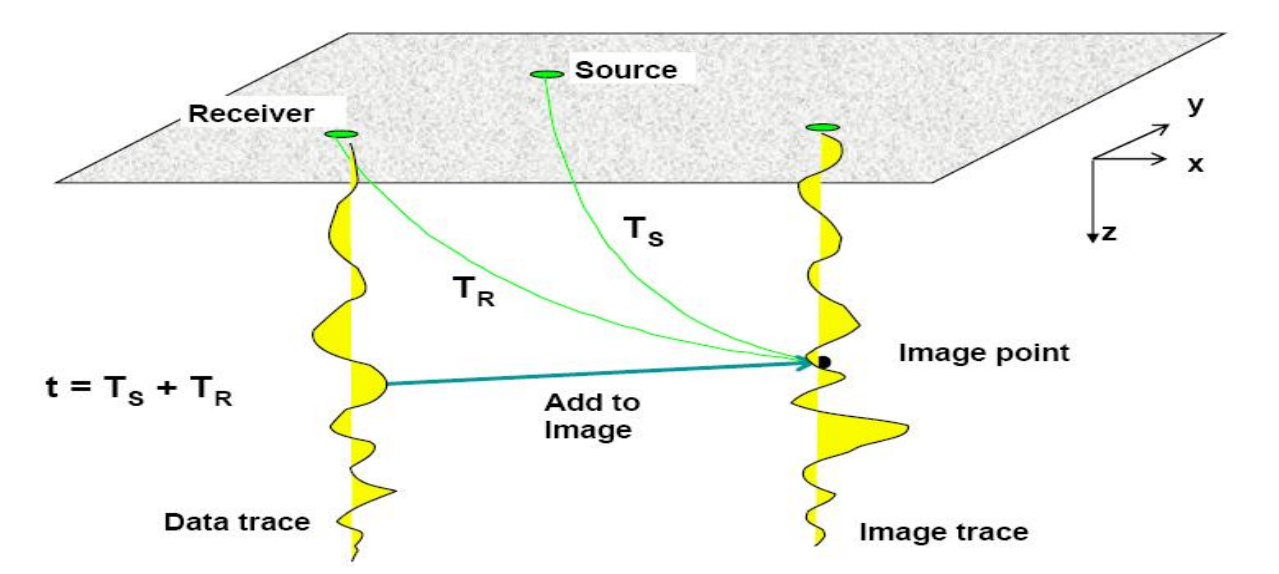
```
REAL,DIMENSION(N,3)::f,u
DO i=1, N
   ……
   f(i,1) = u(i,3) * a
   f(i,2) = u(i,3) * b
   f(i,3) = u(i,3) * c
……
ENDDO
```

On-going work

- Deep analysis with special tools
  - Find a way to higher performance
- GPU implementation for comparing
  - Better understanding of the generalities and differences between GPU and MIC
- Using some other modes for running
  - Especially as coprocessor



GPU implementation of Kirchhoff time migration



- For a measured time sample on a given input trace, the travel time from source and receiver can be calculated using the velocity model and summed as t.
- If the computed travel time equals the actual recorded time at that point, the amplitude will be filtered and accumulated to contribute to the output sample.

Held by Dr. LIU Fang, supported by Beijing Petrosound Geoservices Ltd.

Implementation details

- A hybrid system on multi-GPU clusters
  - Implemented by MPI/CUDA using CPU/GPU cooperation
  - Based on a typical master-slave mode
  - Mapped to three levels: MPI process - pthread - CUDA thread
- Test pata
  - 2D data with 1423 CDPs and 132036 traces (1.3GB)
  - 3D data with 3 inlines and 300 crosslines and 108368 traces (694MB)
- Test platform
  - 8 nodes composed of 4 CPU cores and 2 GPUs per node

Mode	2D			3D		
	CPU	GPU	speedup	CPU	GPU	speedup
Migration stack	256s	78s	3.3	140s	58s	2.4
Velan Movie	1068s	264s	4.0	264	151s	1.8
crp	517s	276s	1.9	549s	102s	5.4
angle	1281s	160s	8.0	269	57s	4.7

Accelerating Linkage Disequilibrium Computation on GPU

- Linkage disequilibrium
  - Consider the haplotypes for two loci A and B with two alleles each—a two-locus, two-allele model
  - The following table defines the frequencies of each combination:

Haplotype	Frequency
A_1B_1	x_{11}
A_1B_2	x_{12}
A_2B_1	x_{21}
A_2B_2	x_{22}

Allele	Frequency
A_1	$p_1 = x_{11} + x_{12}$
A_2	$p_2 = x_{21} + x_{22}$
B_1	$q_1 = x_{11} + x_{21}$
B_2	$q_2 = x_{12} + x_{22}$

$$D = x_{11} - p_1 q_1$$

$$r = \frac{D}{\sqrt{p_1 q_1 p_2 q_2}}$$

  - The absent alleles are recorded by ‘N’, while the rest are by ‘A/G/C/T’

Held by Dr. LIU Fang, supported by China Agriculture University

Optimization details

- Pre-process the input data and convert the data type from ‘char’ to two bits
  - Data bit：whether the primary allele
  - Flag bit：whether absent
- Count N,x11,p1 and q1 using a special designed instruction ‘__popc’ on GPU
  - Returns the number of bits that are set to 1 in the binary representation **in a clock cycle on Fermi devices**
  - Load input 32 chars by 2 reads of unsigned integers
  - Can gain a speedup of more than 32 times theoretically
- Divide data into blocks
  - Less memory consumption on GPU
  - Better for parallel work on GPU clusters
- Re-arranage input data for coalesce read on GPU
- Using ‘atomicInc’ instruction to only store the related pairs
- Map thread organization from 2D matrix to 3D grid
  - Less threads in first two dimensions
  - Same input data for the same block, better for cache hit

Some test results

- Test Platform：
  - GPU version: NVIDIA C2075(448 cuda cores)
  - CPU version: Intel Xeon 5410 (4 CPU cores)
- Timing:
  - N=1000，M=296 (No I/O)
    - CPU：0.62s
    - GPU：0.00091s
    - speedup：more than 600 times
  - N= 45099，M=515 (No I/O)
    - CPU：2126.25 s
    - GPU：1.39 s
    - speedup：more than 1500 times
  - Larger input data:
    - Divide into files with moderate size
    - Gains the similar speedup each

Accelaration of InsPecT

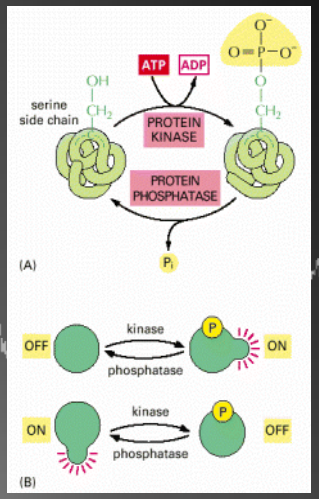
- A commonly used software in SIBS
  - CUDA+MPI
  - Master/Slave mode

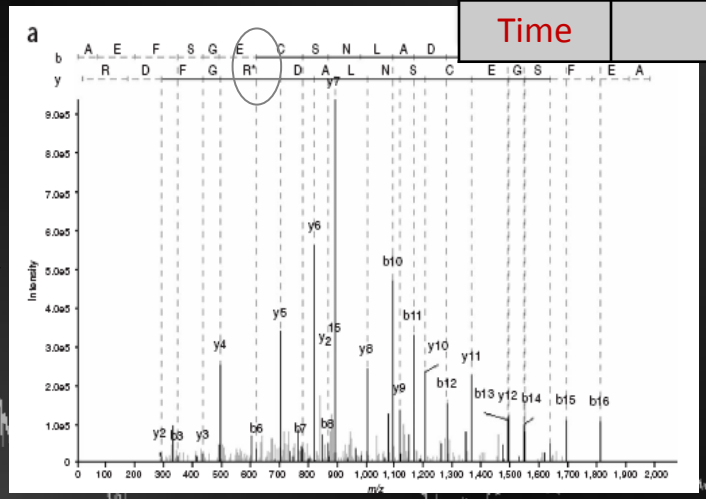
Held by Dr. LANG Xianyu.

cuda-InsPecT
- via MPI+CUDA

Software: cuda-InsPecT（two modifications）
Database: 62346 mass spectrometric，107962 protein sequences
Platform: Dawning 6000A

	Single CPU core(estimate)	677 Tesla C2050 GPUs
Time	6 year	2.034 h





HPSEPS

- Parallel eigenvalue solver
  - Multi-GPU solution for symmetric tridiagonal matrices
  - Based on MRRR algorithm
  - MPI+CUDA

Held by Prof. ZHAO Yonghua

20k*20k matrix performance					
GPUs	1	2	4	8	16
Time(s)	824.2	480.4	290.9	210.4	135.8

1 GPU compared to 1 CPU core				
Size	2000	5000	10000	15000
CPU	23.8	548.1	2320.18	8942.4
GPU	2.06	29.49	102.32	365.56
Speedup	11.67	18.93	22.68	24.5

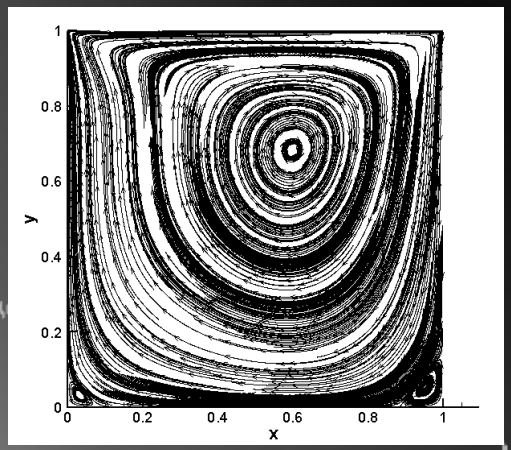
16 GPUs with huge matrices			
Size	20000	30000	50000
Time(s)	135.8	433.44	1388.33

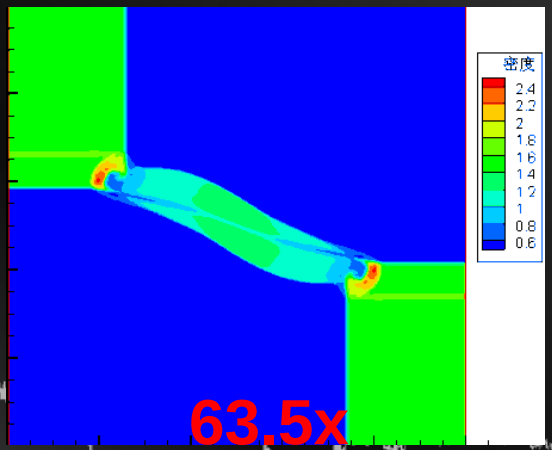
2D Riemann problem in CFD

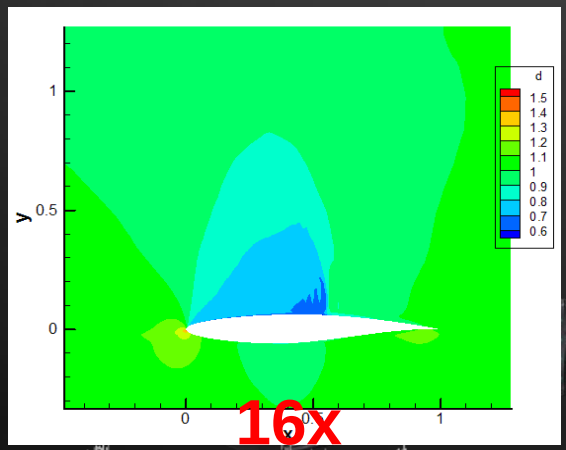
- Be a kernel of the solvers in CCFD software
  - OpenCL/CUDA (2 versions)

Held by Prof. LU Zhonghua

Size	OpenCL	CUDA
512 * 512	4.7x	10.43x
1024 * 1024	7.69x	14.72x
2048 * 2048	8.6x	16x
4096 * 4096	30.7x	57.14x
5120 * 5120	34.8x	63.17x







Some considerations

- How to decide whether an application can be speedup on the accelerators, Especially for large scale computations?
- Are there more efficient models to implement the heterogeneous codes with out performance loss?
- How to build a simple but available performance model for applications and using it to find out the bottlenecks?
- If accelerators do most computation, what will CPUs be?
- How to build future machines for a supercomputing center, CPUs, GPUs, MICs?

Summary

- Heterogeneous computing with accelerators is one of the trend in scientific High-performance computing. It will play an important role on the way to Exascale.
- We are keeping the pace with the development of heterogeneous computing. Guys in SCCAS actively participate and practice heterogeneous computing and got some achievement, especially on GPU computing.
- Before another revolutionary technique comes out, using MPI with heterogeneous computing kernel is still a efficient model of implementing large scale applications.
- Working on scientific applications, we need the cooperations for scientists of specialized subject, mathematics and computer science.
- Scalability may become a problem, but the essential thing is to extremely explore the absolutely real performance of the accelerators.
- There should be more practice or research work to find out an easy way to make good use of both CPUs and accelerators.

中国科学院超级计算中心
Supercomputing Center of CAS

THANK YOU!