

# Adaptive Optimization for Petascale Heterogeneous CPU/GPU Computing

Canqun Yang, **Feng Wang**, Yunfei Du, Juan Chen, Jie Liu, Huizhan  
Yi and Kai Lu

School of Computer Science,  
National University of Defense Technology, P.R.China

## Agenda

- Introduction
- Issues
- Solutions
- Results

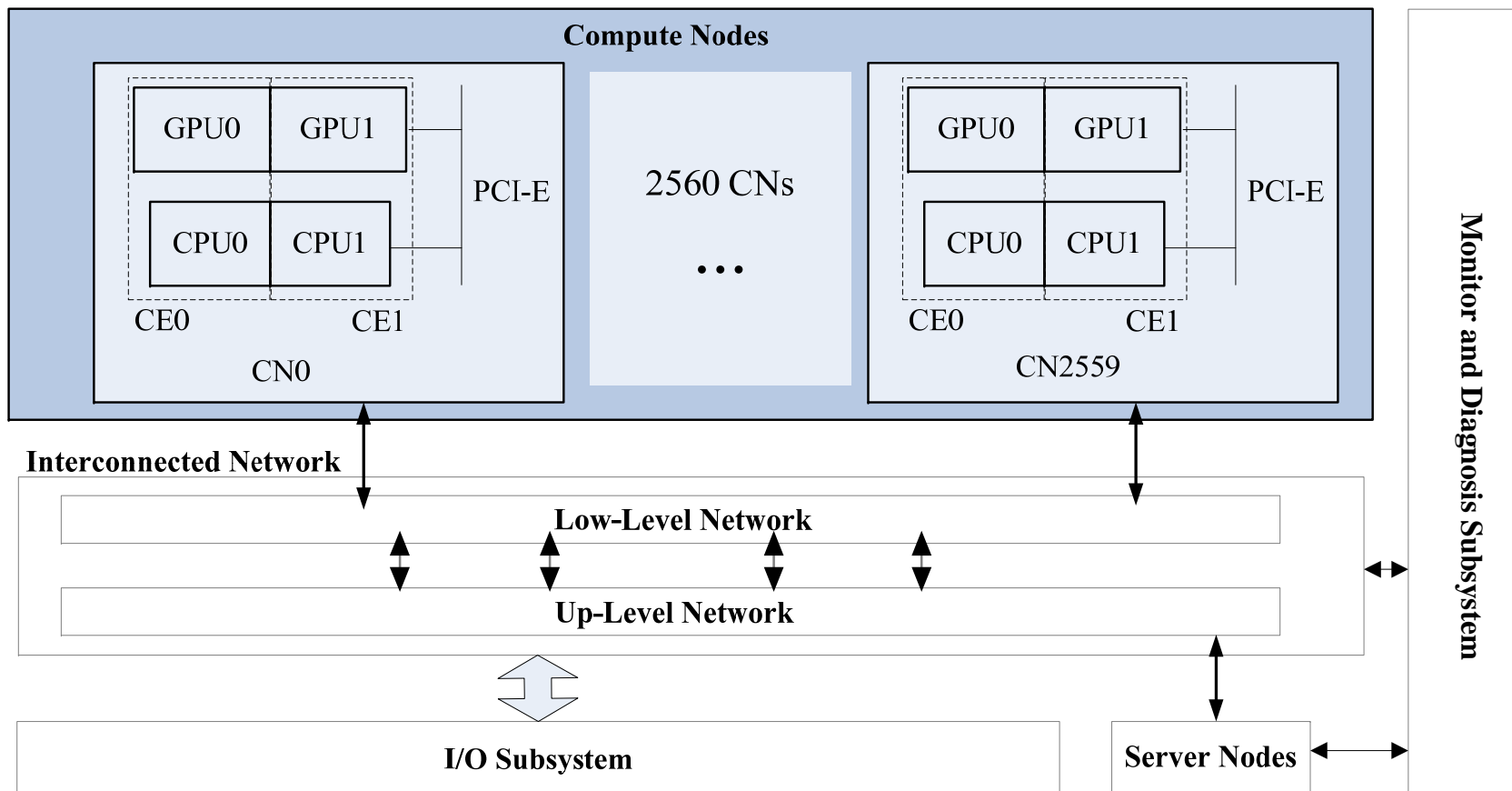


TianHe-1 SuperComputer

# Introduction

- Homogeneous computer systems
  - ◆ Cray Jaguar with 224,000+ CPU cores
- Heterogeneous computer systems
  - ◆ Accelerators: CELL, GPGPU, FPGA, ClearSpeed
  - ◆ IBM Roadrunner ( the first petascale supercomputer)
    - Power + CELL
  - ◆ NUDT Tianhe-1
    - Xeon Quad-core CPUs + AMD 4870 GPUs
    - ranked No.5 in November 2009 on Top500 list
    - ranked No.7 in June 2010

# Overview of TianHe-1 system



# Overview of TianHe-1 system

- One compute element
  - ◆ one quad-core Intel Xeon processors
  - ◆ 32GB shared memory
  - ◆ ATI Radeon HD4870 GPU chips
    - RV770 chip
    - 1GB local memory per chip
  
- Interconnection
  - ◆ two-level QDR Infiniband switches
  - ◆ 40 Gbps aggregate bandwidth
  - ◆ 1.2us latency
  
- The peak performance is 1.206 PFLOPS

## Overview of TianHe-1 system

- Compared with Cell-accelerated system

	GPU-acc System	Cell-acc System
Accelerator Local Memory	1 GB	8 GB
Bandwidth between host and accelerator	Host <-> PCI-E: ~500MB/s PCI-E <-> GPU: 5GB/s	~2 GB/s
Memory Bandwidth of the accelerator	up to 115GB/s	25.6GB/s

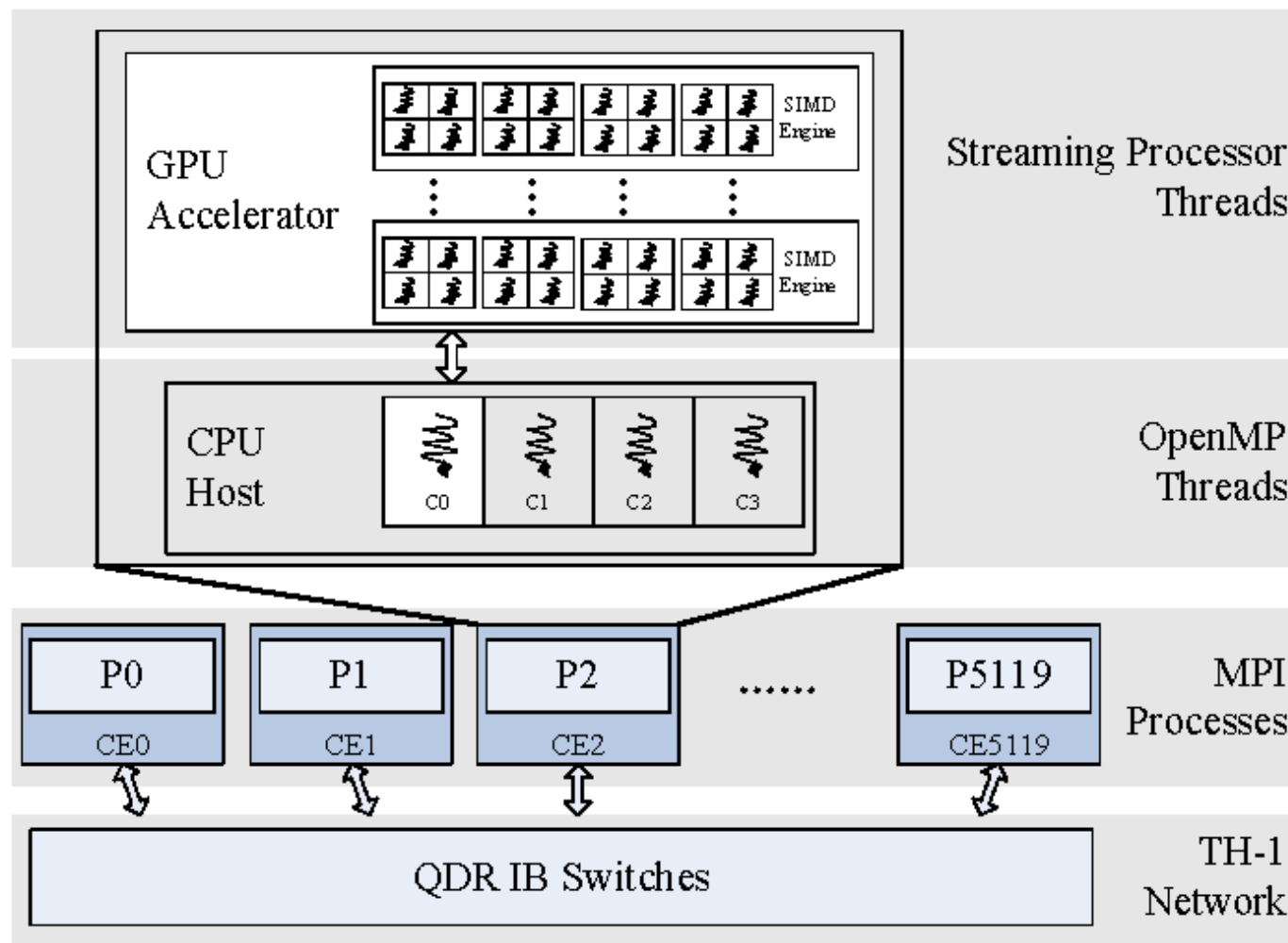
## Issues

- CPUs should not be ignored
  - ◆ CPUs: 214.96 TFLOPS
  - ◆ GPUs: 942.08 TFLOPS
- Load balance across CPUs and GPUs
- Communications between CPUs and GPUs

## Solutions

- We developed a **framework** to combine multiple programming models to make full use of the CPUs and GPUs.
- We present an **adaptive partitioning technique** to distribute the computations across the CPU cores and GPUs to achieve well-balanced workloads with **negligible runtime overhead**.
- We present a **software pipelining technique** for GPU computing to hide effectively the communication overhead between the CPU and GPU memories.
- We employed a **combination method** consisting some traditional and important optimizations to implement a version of Linpack, making TianHe-1 the 5th fastest supercomputer at that time.

# Hybrid programming and executing model



## Linpack Benchmark

- Solves a random dense linear system of equations
- Complexity is  $(2/3)N^3 + 2N^2 + O(N)$   
 $Ax = b; \quad A \in \mathbf{R}^{n \times n}; \quad x, b \in \mathbf{R}^n$
- Ranking supercomputers in the Top500.
- Using LU decomposition method
  - ◆ The matrix update: the matrix-matrix multiply (DGEMM) which is an  $O(N^3)$  operation
  - ◆ Upper (U) matrix factor: a triangular solve with multiple right-hand-sides (DTRSM) kernel which is an  $O(N^2)$  operation

# Adaptive partitioning in the Linpack

## ■ Split DGEMM

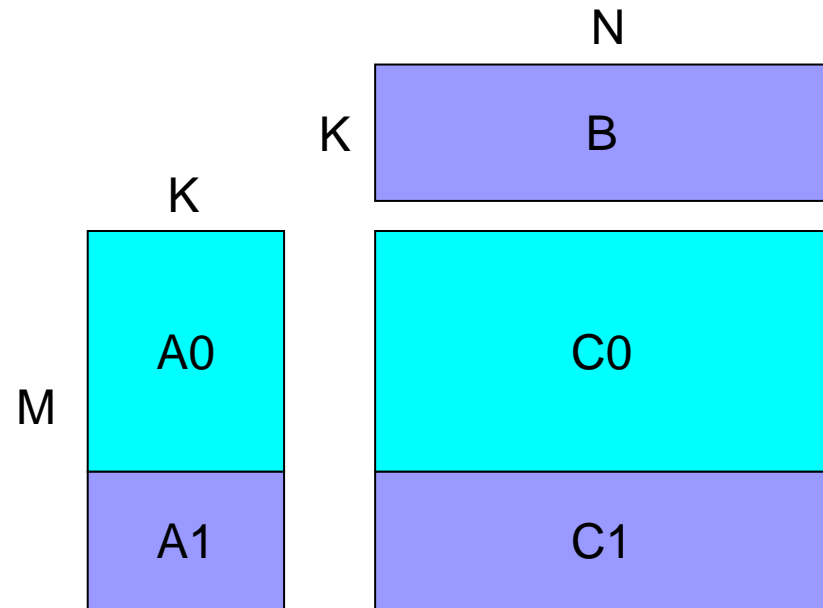
◆  $C = A * B + C$

➤  $C0 = A0 * B + C0$

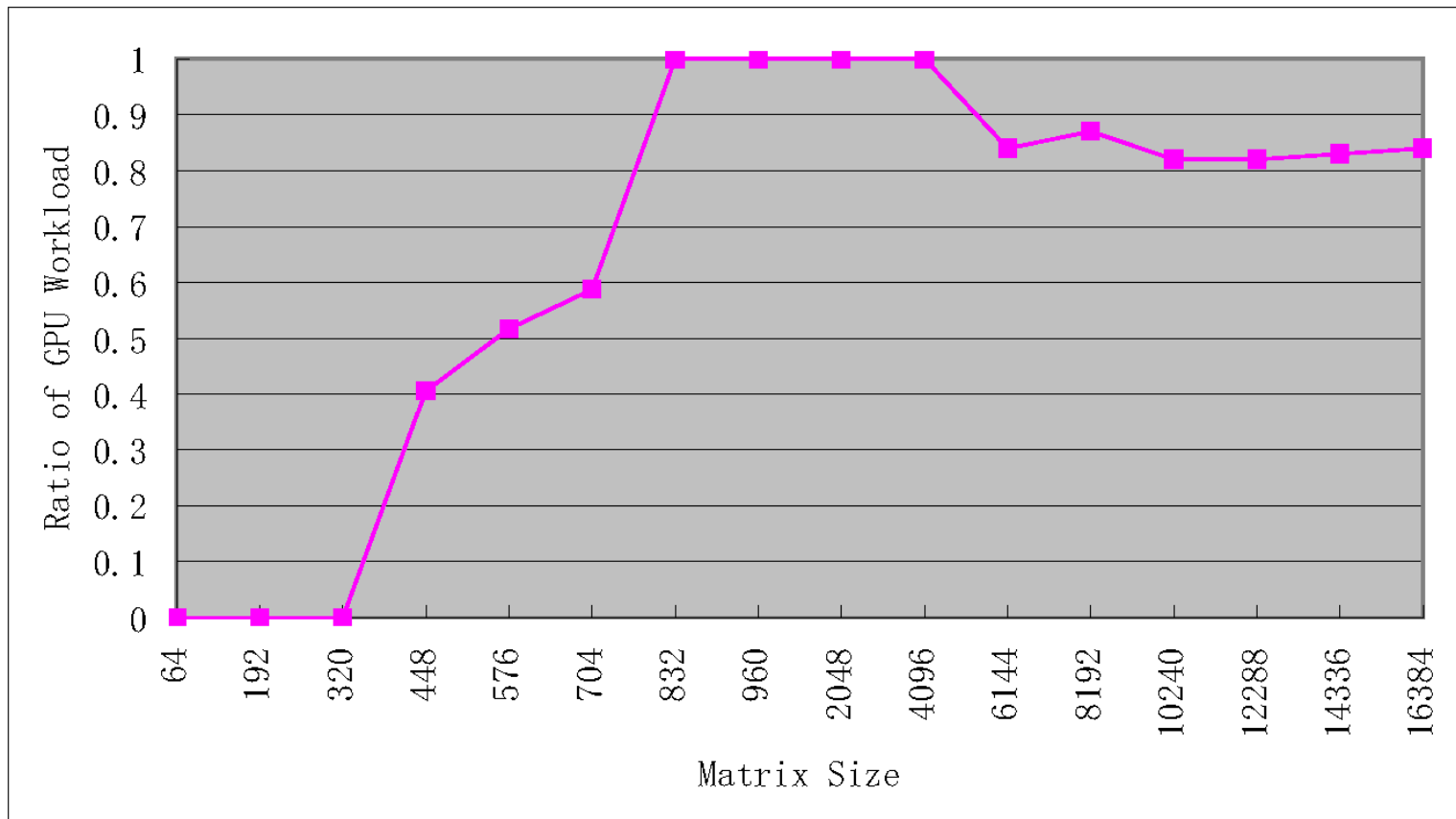
➤  $C1 = A1 * B + C1$

## ■ Determine the split ratio

◆ Statically?

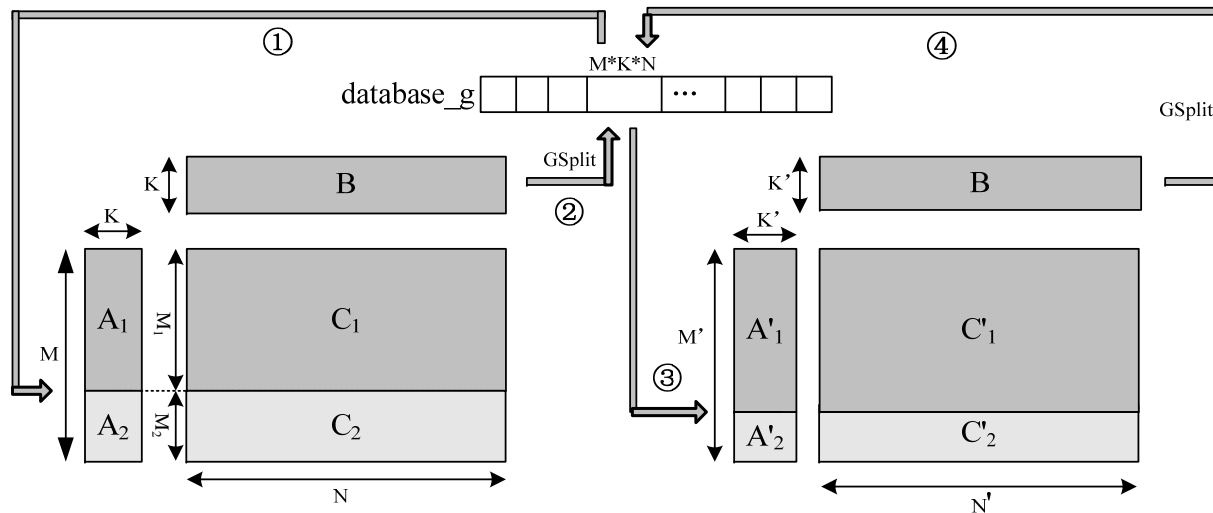


# Adaptive partitioning in the Linpack



# Adaptive partitioning in the Linpack

- Tune the split ratio according to the scale ( $M*N*K$ ) of DGEMM
  - ◆  $W_{[GPU]} = W * GSplit$ ,  $W_{[CPU]} = W * (1 - GSplit)$
  - ◆  $GSplit = P_{[GPU]} / (P_{[GPU]} + P_{[CPU]})$
  - ◆  $M * K * N \approx M' * K' * N'$



$W$  : the workload for a program  
 $W_{[GPU]}$ : the workload to GPU  
 $W_{[CPU]}$ : the workload to CPU  
 $GSplit$ : The fraction of the workload mapped to the GPU  
 $P_{[GPU]}$ : actual GPU performance for workload  
 $P_{[CPU]}$ : actual CPU performance for workload

# Adaptive partitioning in the Linpack

- The print screen of Linpack test

```
SPLIT[0]A (2013.8GFlops), mnk=34048,24320,1216 m_gpu=29240,m_cpu=4808, split=0.858815
DGEMM[0]: tab=N,N mnk=29240,24320,1216 ldabc=34048,1216,35264 GPU=10.66s,162.18G
DGEMM[0]: tab=N,N mnk=4808,24320,1216 ldabc=34048,1216,35264 CPU=9.93s,28.63G

SPLIT[0]A (100.7GFlops), mnk=34048,1216,1216 m_gpu=28136,m_cpu=5912, split=0.826444
DGEMM[0]: tab=N,N mnk=28136,1216,1216 ldabc=34048,1216,35264 GPU=0.64s,130.81G
DGEMM[0]: tab=N,N mnk=5912,1216,1216 ldabc=34048,1216,35264 CPU=0.62s,27.98G

SPLIT[0]A (100.7GFlops), mnk=34048,1216,1216 m_gpu=28136,m_cpu=5912, split=0.826444
DGEMM[0]: tab=N,N mnk=28136,1216,1216 ldabc=34048,1216,35264 GPU=0.63s,132.17G
DGEMM[0]: tab=N,N mnk=5912,1216,1216 ldabc=34048,1216,35264 CPU=0.62s,28.04G

SPLIT[0]A (2013.8GFlops), mnk=34048,24320,1216 m_gpu=28968,m_cpu=5080, split=0.851000
DGEMM[0]: tab=N,N mnk=28968,24320,1216 ldabc=34048,1216,35264 GPU=10.58s,161.93G
DGEMM[0]: tab=N,N mnk=5080,24320,1216 ldabc=34048,1216,35264 CPU=10.51s,28.59G

SPLIT[0]A (100.7GFlops), mnk=34048,1216,1216 m_gpu=28064,m_cpu=5984, split=0.824425
DGEMM[0]: tab=N,N mnk=28064,1216,1216 ldabc=34048,1216,35264 GPU=0.63s,131.96G
DGEMM[0]: tab=N,N mnk=5984,1216,1216 ldabc=34048,1216,35264 CPU=0.65s,27.05G

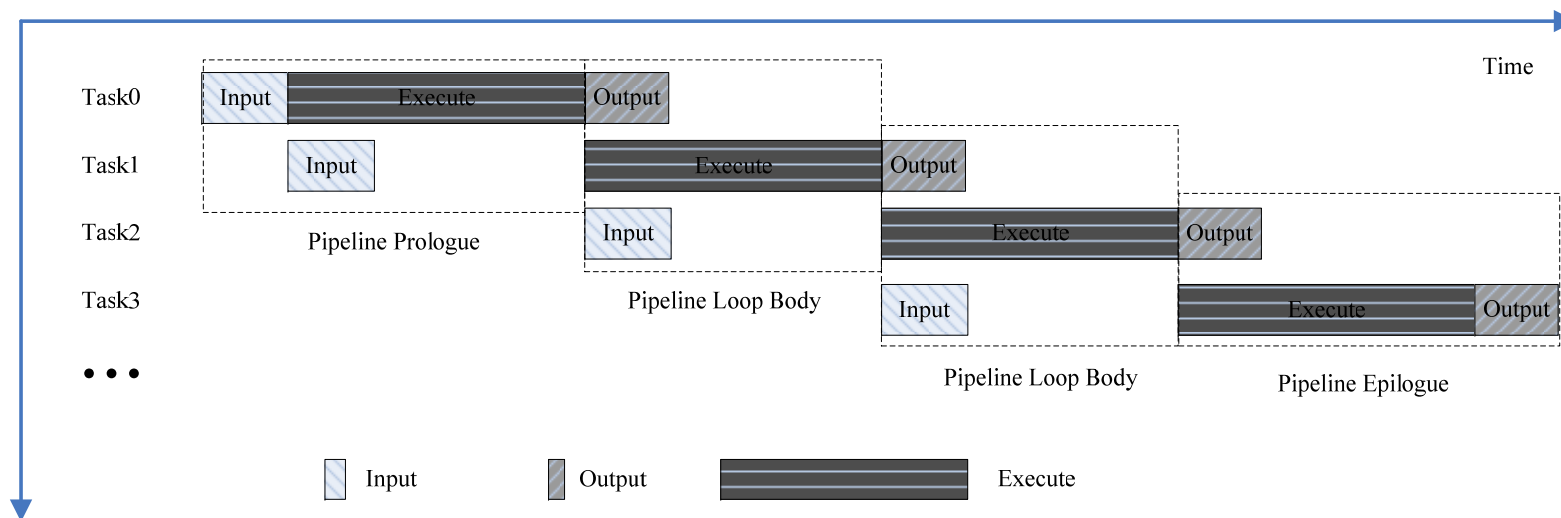
SPLIT[0]A (100.7GFlops), mnk=34048,1216,1216 m_gpu=28288,m_cpu=5760, split=0.830893
DGEMM[0]: tab=N,N mnk=28288,1216,1216 ldabc=34048,1216,35264 GPU=0.63s,132.77G
DGEMM[0]: tab=N,N mnk=5760,1216,1216 ldabc=34048,1216,35264 CPU=0.61s,27.99G
```

## Software Pipelining Method

- The communication is severe
- Our solution
  - ◆ Separate one task into three phases
    - Input data
    - Computation
    - Output the result back to the host
  - ◆ Overlap computation and data transferring

# Software Pipelining Method

- prologue/loop body/epilogue
- $Time = T_{input} + T_{output} + N \times T_{execute}$



# Software Pipelining Method

- Work splitting

$$A \times B = C \quad \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \times (B_1 \ B_2) = C$$

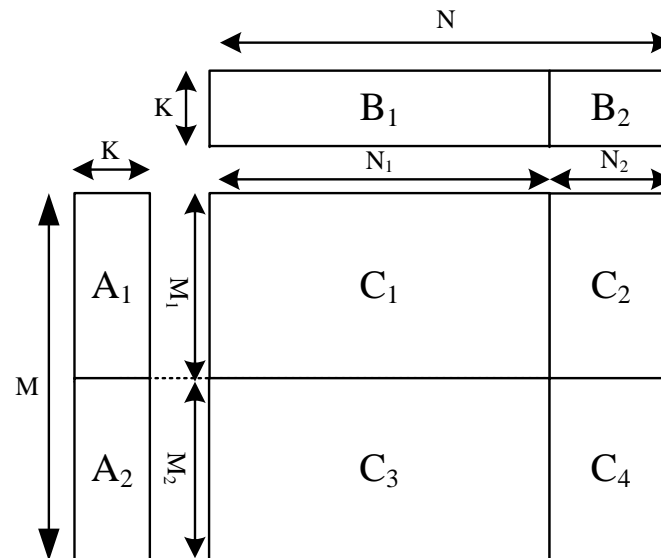
- Four tasks

$$T0 : C_1 = A_1 \times B_1$$

$$T1 : C_2 = A_1 \times B_2$$

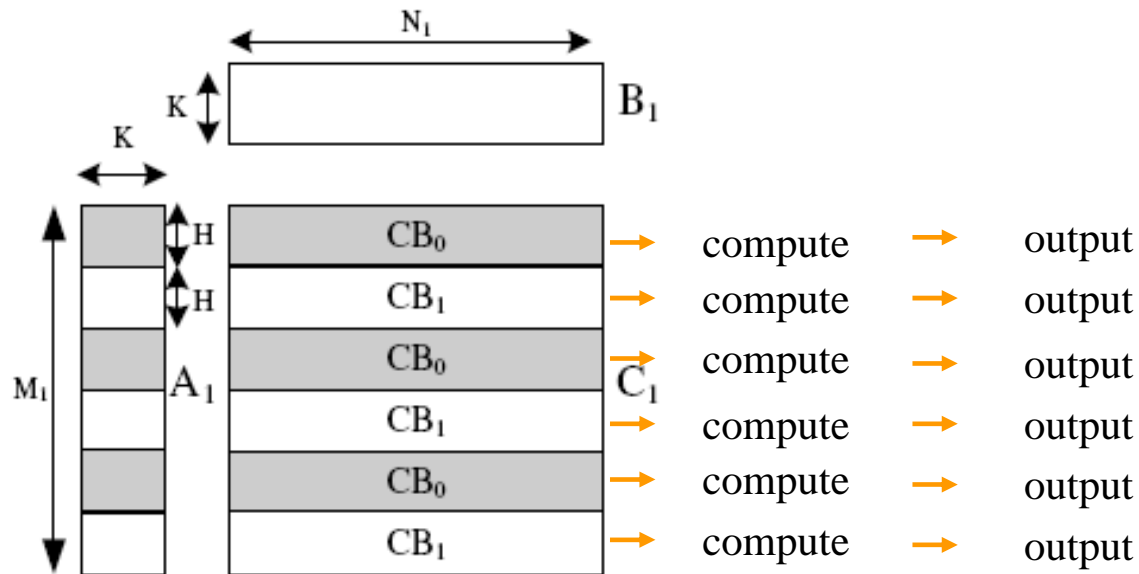
$$T2 : C_3 = A_2 \times B_1$$

$$T3 : C_4 = A_2 \times B_2$$



# Software Pipelining Method

- Optimize 1: Overlap GPU computing with output
  - ◆ the blocking matrix multiplication
  - ◆ Double output buffers:  $CB_0$  and  $CB_1$



# Software Pipelining Method

## ■ Optimize 2: Data reuse

◆  $T_0, T_1, T_3, T_2$

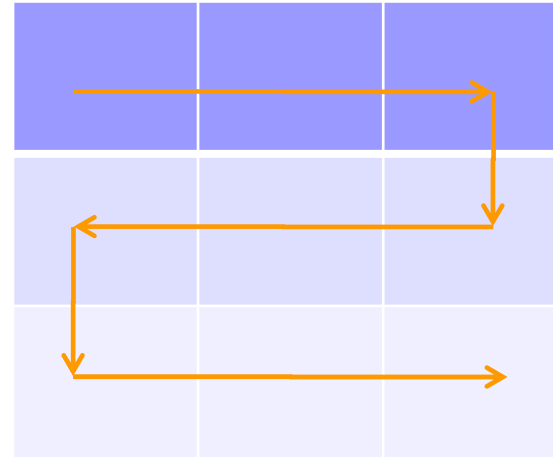
$$T_0 : C_1 = A_1 \times B_1$$

$$T_1 : C_2 = A_1 \times B_2$$

$$T_3 : C_4 = A_2 \times B_2$$

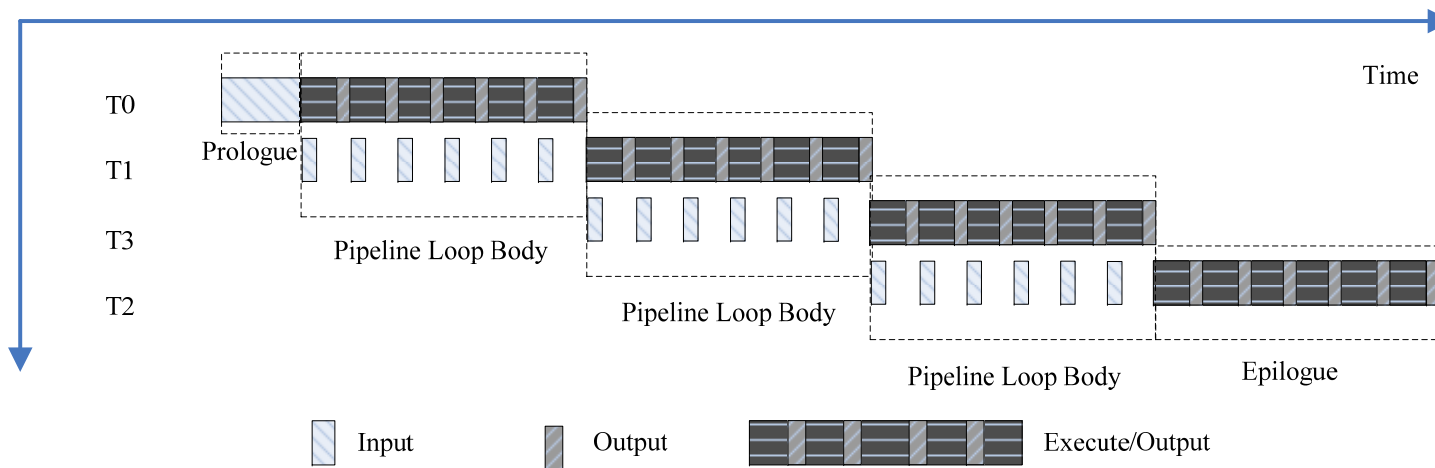
$$T_2 : C_3 = A_2 \times B_1$$

◆  $T_0(A_1 B_1) T_1(B_2) T_3(A_2) T_2(B_1)$



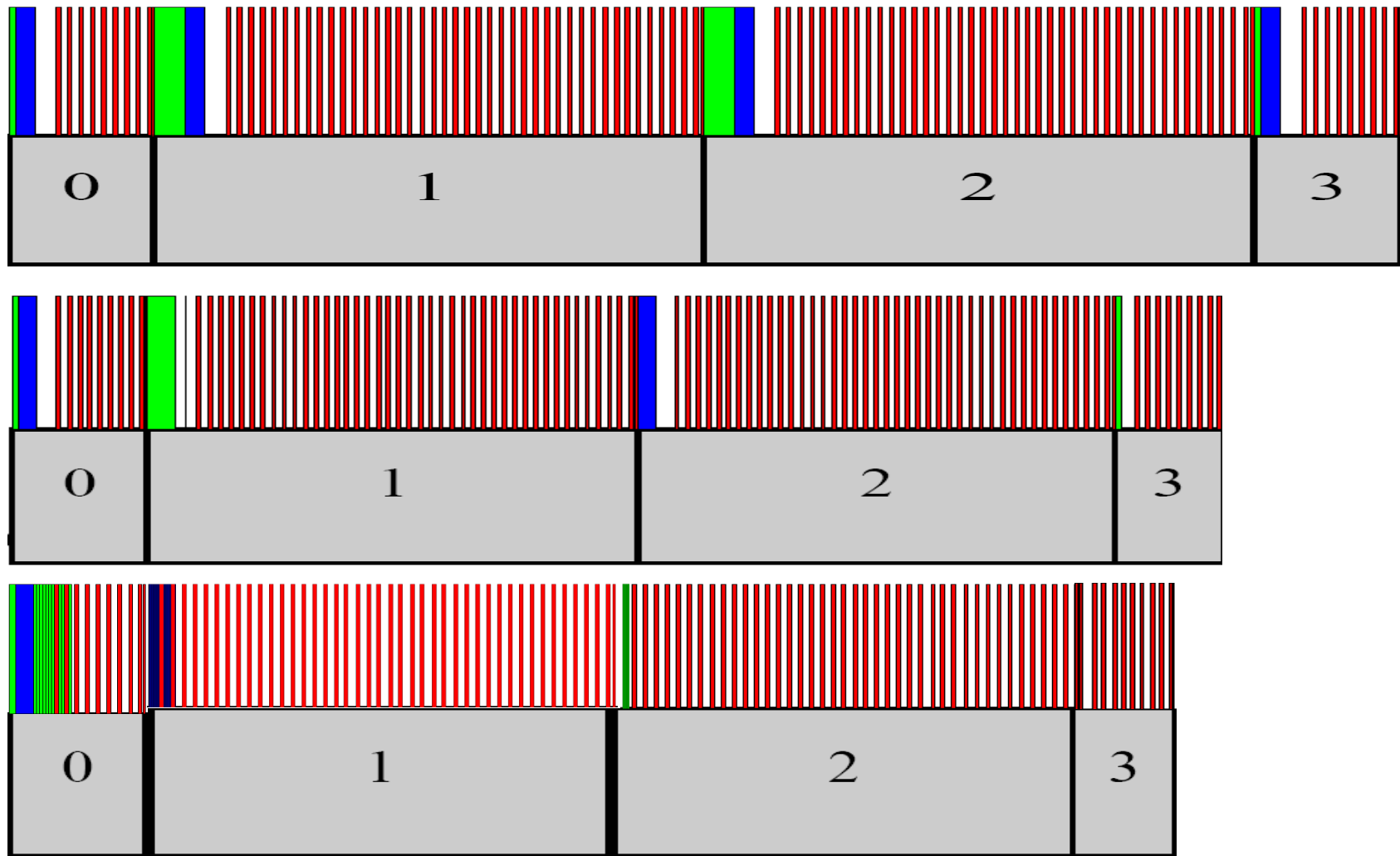
# Software Pipelining Method

- Optimize 3: Overlap GPU computing with the input of the next task



# Adaptive Optimization for Petascale Heterogeneous CPU/GPU Computing

Write A Write B Read C

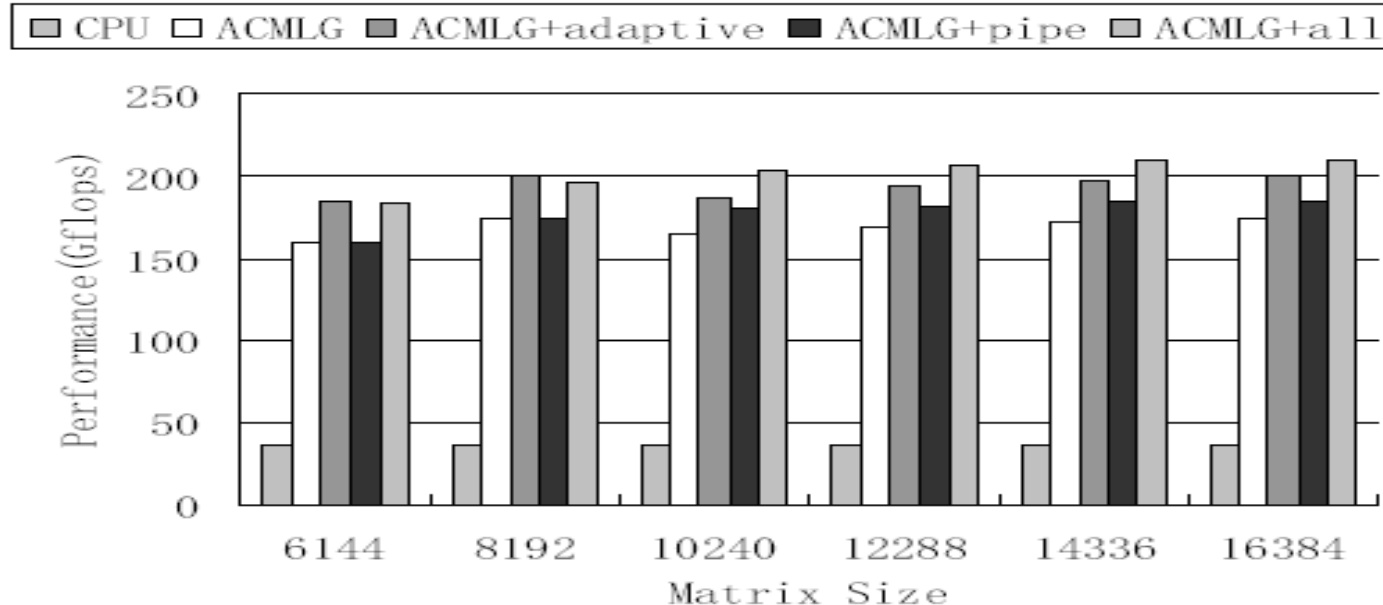


## Experiment and Evaluation

- Single compute element
  - ◆ 1CPU + 1GPU chip
  - ◆ One thread per cpu core
  - ◆ Intel Math Kernel Library 10.2.1.017 (MKL) for CPU
  - ◆ Vendor's library: ACML-GPU 1.0 (AMD Core Math Library for Graphic Processors)
  - ◆ Our BLAS library

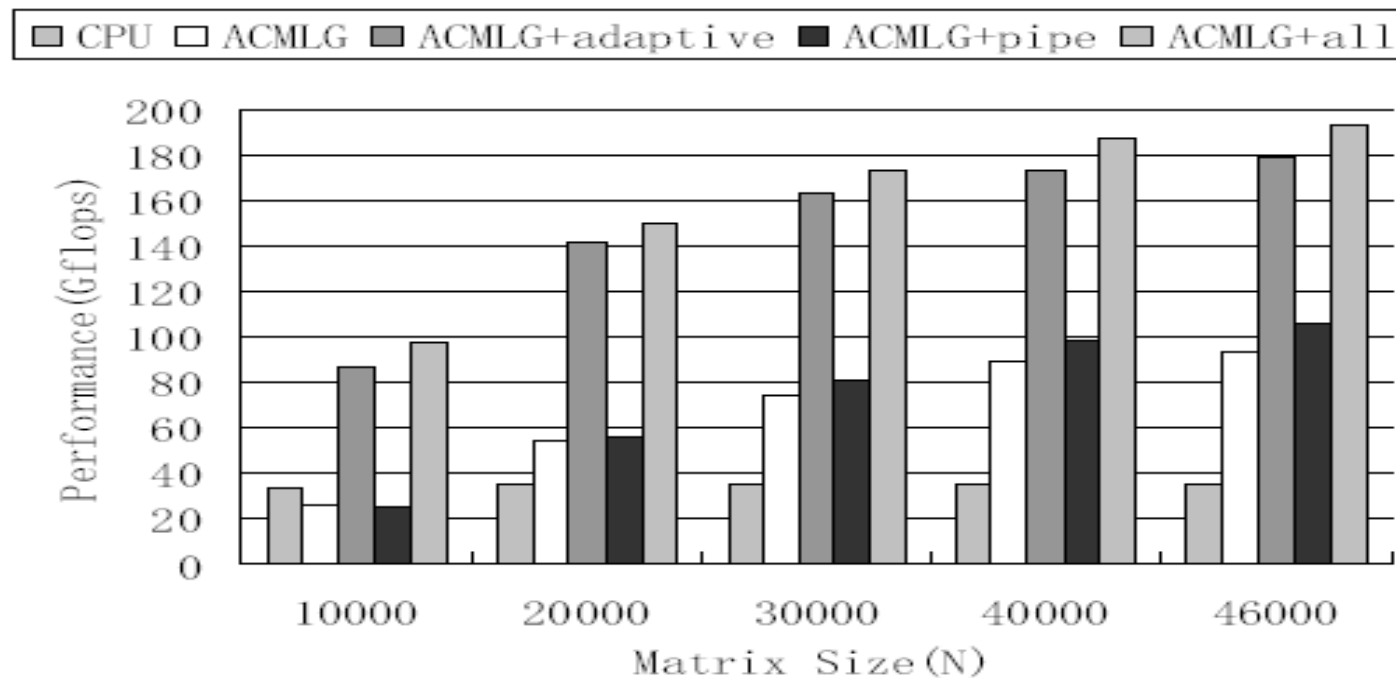
## Results of DGEMM

- The adaptive mapping improved 14.64%
- The pipeline method got 7.61%
- Overall achieved 22.19% improvement



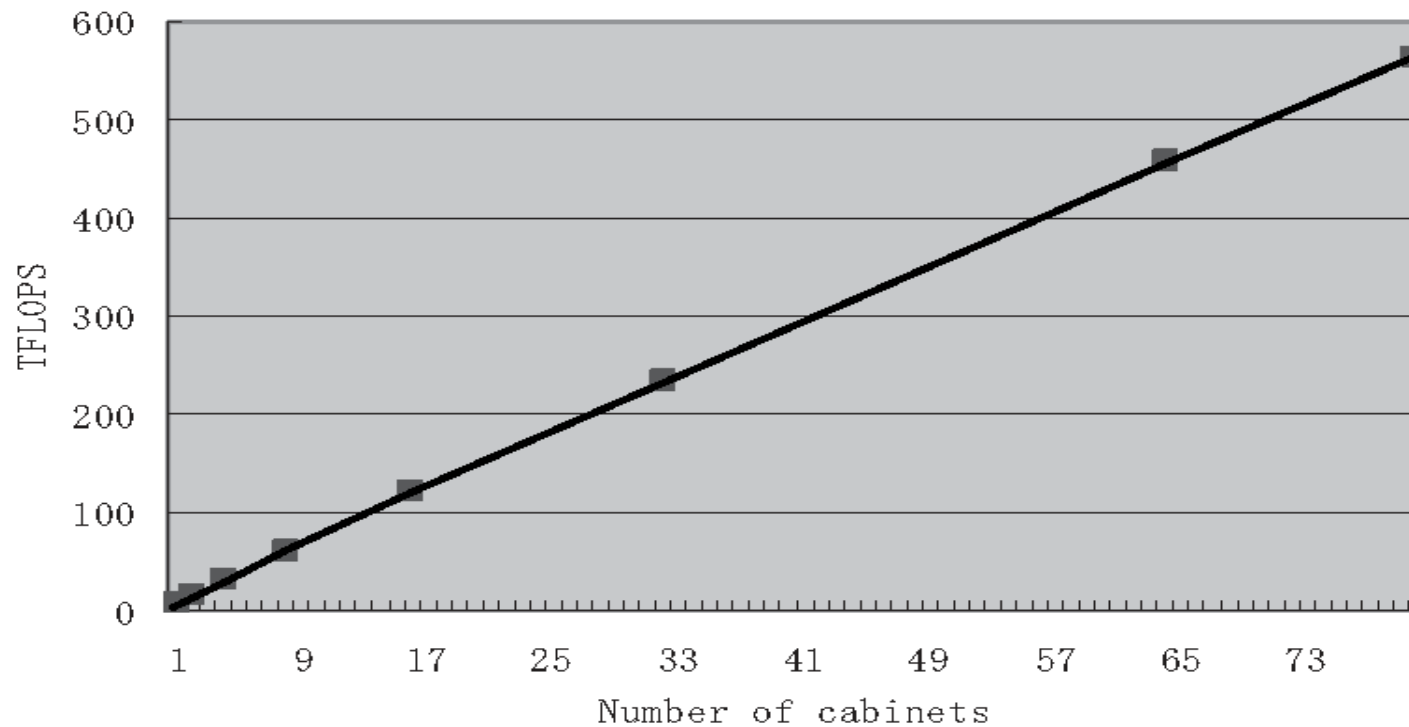
## Results of Linpack

- 196.7 GFLOPS for a matrix of size  $N = 46000$
- 70.1% of the peak on one compute element



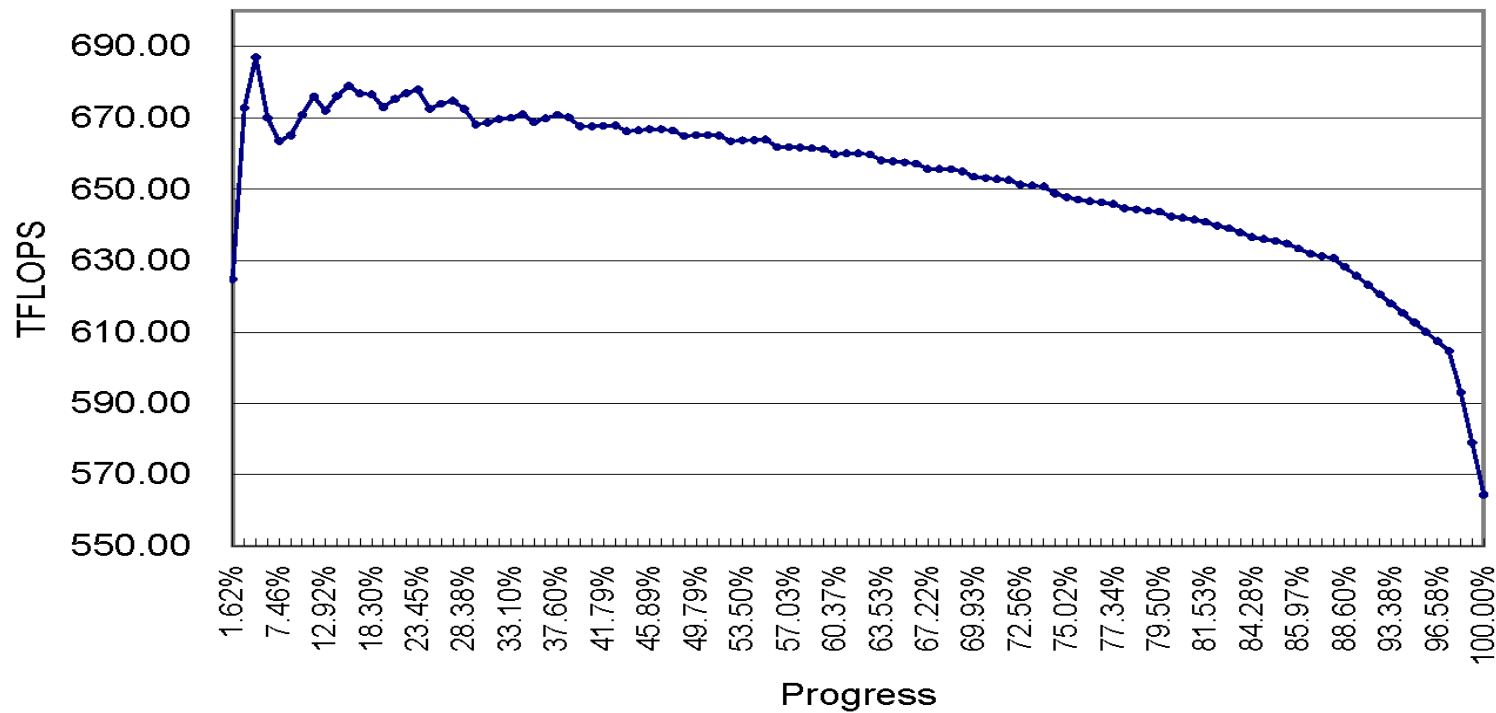
## Results of Multi-Cabinets

- Scaling efficiency is 87.76% from 1 to 80 cabinets. TFLOPS



# Results of full configuration

- Performance of Linpack running on TianHe-1
  - ◆ 563.1 TFLOPS



Thanks