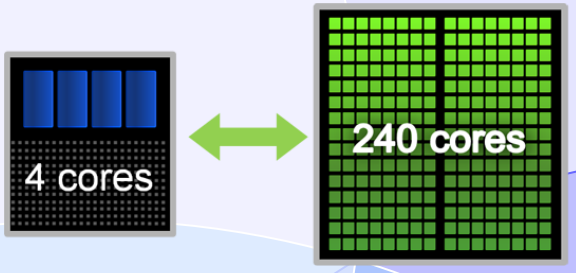
GPUクラスタによる格子QCD計算

広大理　尾崎裕介
石川健一

1.1 Introduction

GeForce GTX 285

- Graphic Processing Units…
  - 1チップに数百個の演算器
  - 多数の演算器による並列計算→～TFLOPS（単精度）
    - CPU…数十GFLOPS
  - バンド幅→～100GB/s
  - コストパフォーマンス→～$400



- GPU の開発環境
  - NVIDIA…CUDA
    - http://www.nvidia.co.jp/object/cuda_home_new_jp.html
  - AMD(ATI)…Stream SDK
    - http://developer.amd.com/gpu/ATIStreamSDK/Pages/default.aspx
  - OpenCL, etc.

1.2 GPU for Lattice QCD simulation

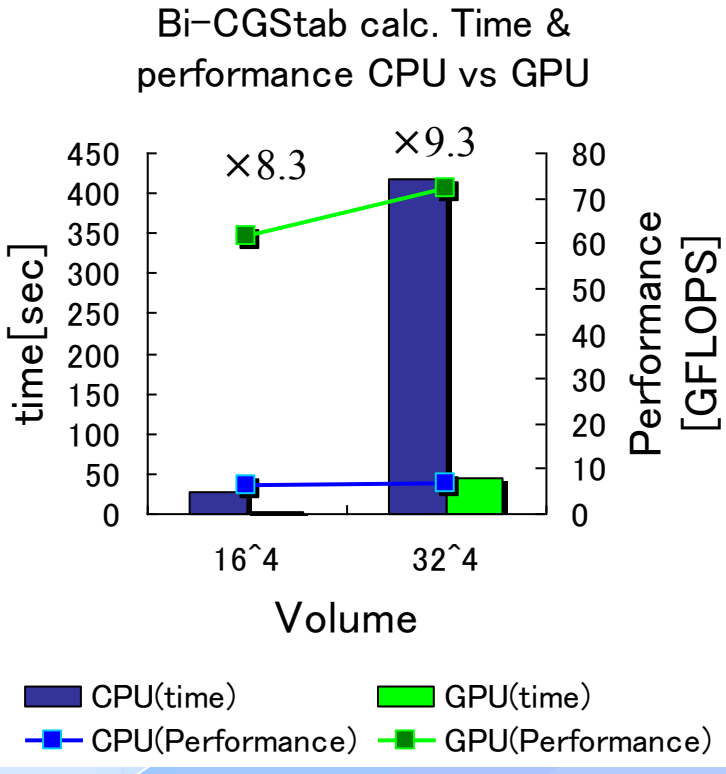
- Lattice QCDの計算時間 $D^{-1}\psi$ がほとんど

$$D^{-1} : \text{quark propagator}$$
$$\psi : \text{quark field}$$

- この計算を行うsolver (mix precision Bi-CGStab) の単精度部分にGPU
  - mix precision solver …

    high precision solver の前処理としてlow precision solverを用いる手法
- 1台のGPUで約10倍の加速効果

- CPU : Core i7 920 @ 2.67GHz
  - coded by Fortran, use SSE, openMP
- GPU : Geforce GTX 285
  - coded by CUDA 2.3
- Use clover fermion
- kappa=0.126, csw=1.0, accuracy=10^{-14}



1.3 motivation of GPU cluster

■ 格子QCDの計算の見積もり

Karl Jansen, arXiv: 0810.5634v2

◆ 計算量 $\approx \left(\frac{20\mathrm{MeV}}{m_q}\right)^{1\sim2} \left(\frac{L}{3\mathrm{fm}}\right)^{4\sim5} \left(\frac{0.1\mathrm{fm}}{a}\right)^{4\sim6}$ [TFLOPS × 年]

$m_{u,d} = 5\mathrm{MeV}, \quad L = 6\mathrm{fm}, \quad a = 0.1\mathrm{fm}$

⟶ 1TFLOPSのGPUで**60～500年**

◆メモリ

- Bi-CGStab solver for clover fermion with even/odd precondition
- quark field : 96 Byte/site
- gauge field : 288 Byte/site
- clover field : 336 Byte/site
- work vector : 7×96 Byte/site

1392Byte×格子サイズ(60^3×30)
= 9GByte
- Geforce GTX 280: 1GByte
- Tesla C1060 : 4GByte

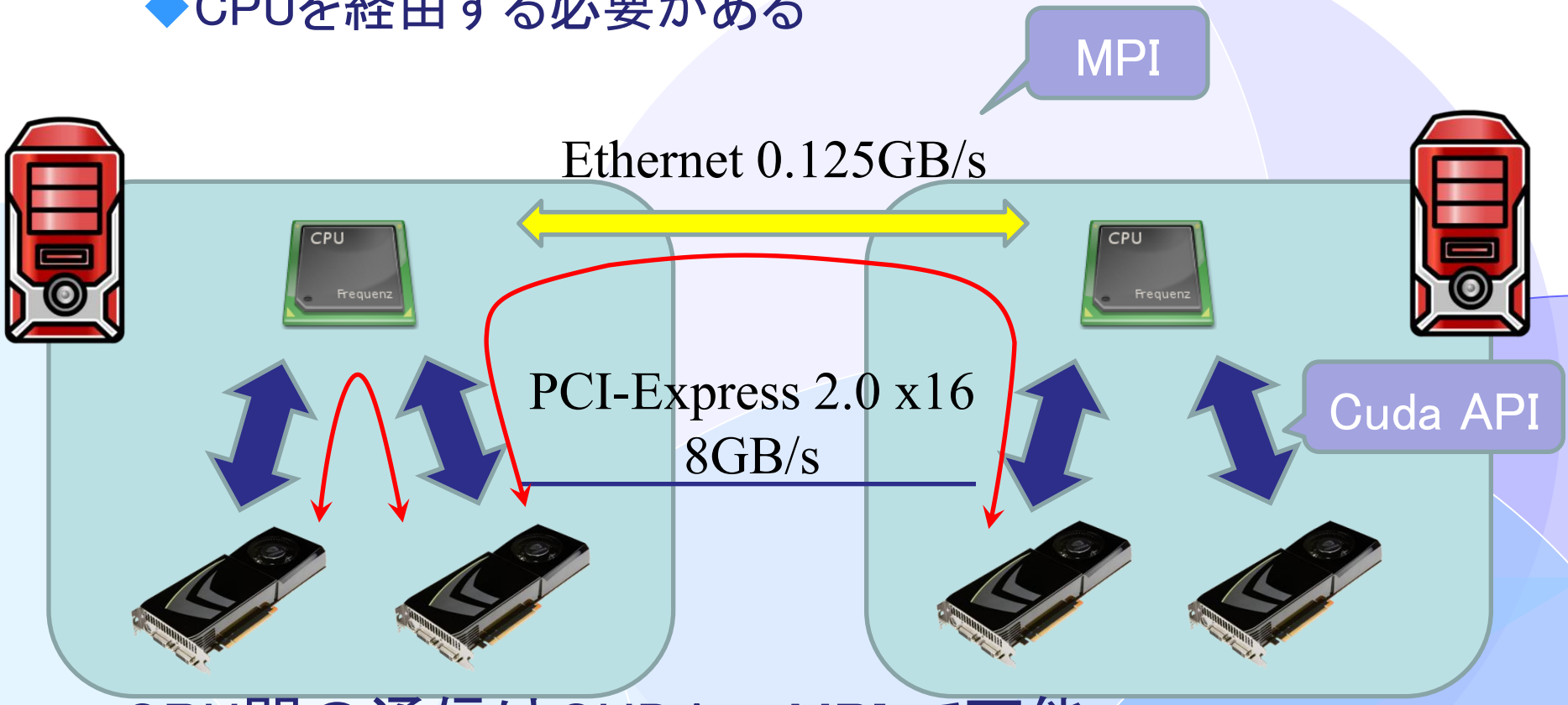
■ 並列計算の必要性

1.4 用意した環境

- GPU : Geforce GTX 285 × 2/CPU
- CPU : intel Core i7 920 × 4
- メモリ : 6GByte × 4
- LAN Adapta : intel Gigabit ET Quad Port Server

2.1 Communication GPUs

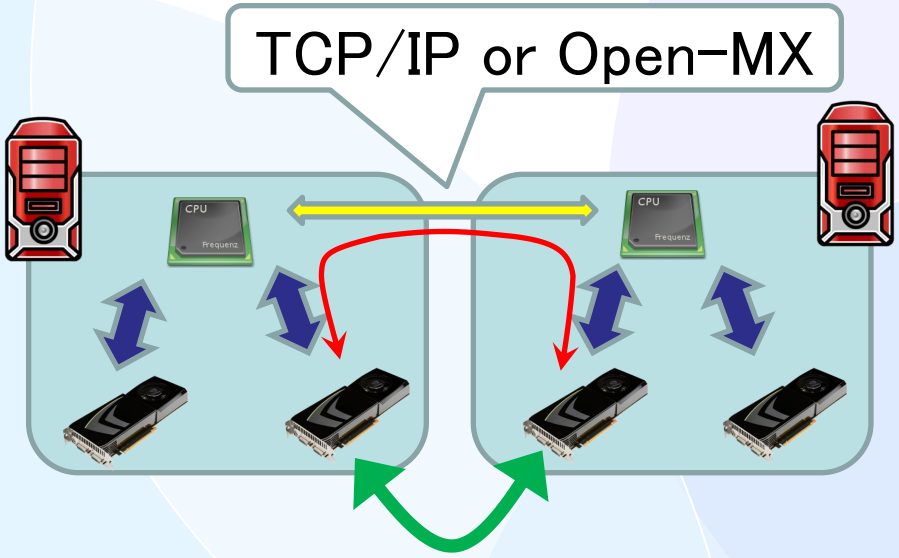
- GPU間で直接データ交換する方法は現在ない
  - CPUを経由する必要がある

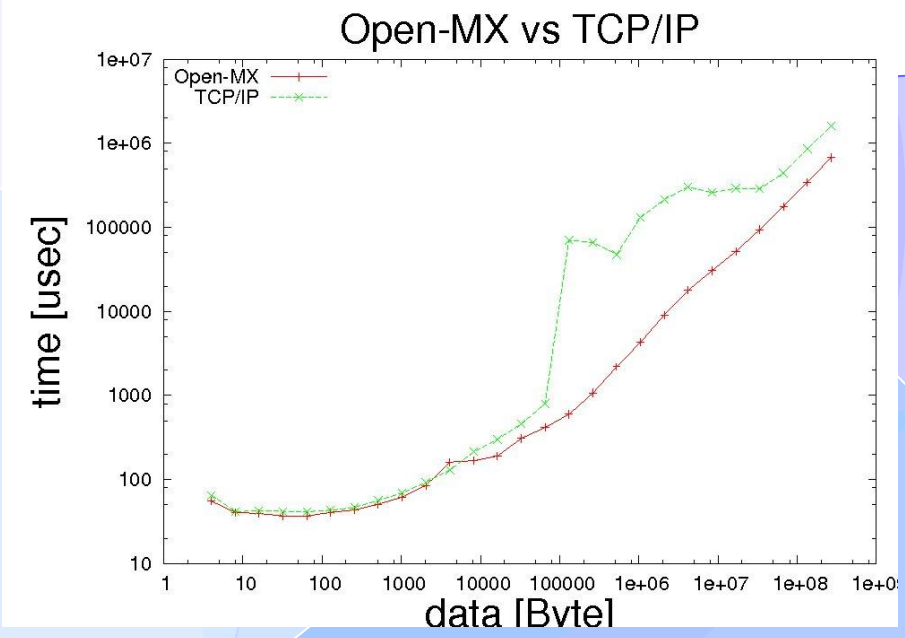


- GPU間の通信はCUDA + MPI で可能

2.2 Open-MX

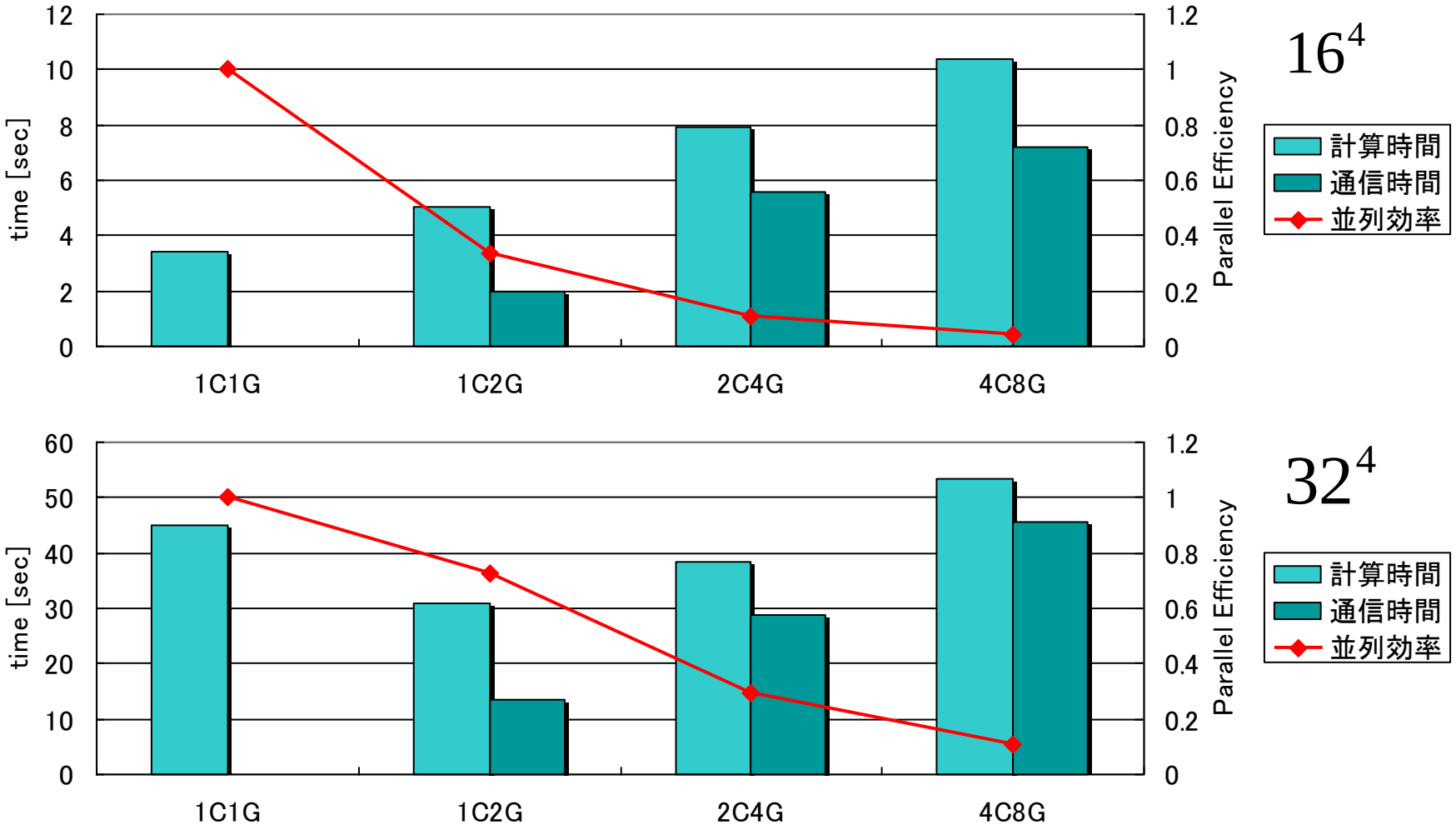
- Myrinetという高速通信で使われているMyrinet Express protocol を TCP/IP protocol の代わりに Ethernet 上で実装する software
  - http://open-mx.gforge.inria.fr/





2. 3 Calculation on GPU cluster

- kappa=0.126
- csw=1.0
- accuracy=10^{-14}
- Bi-CGStab solver

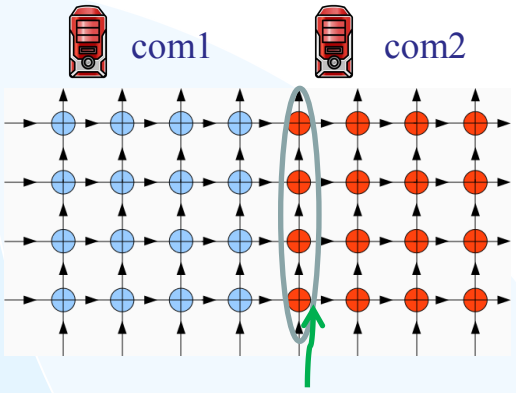


2.4 Detail of hopping part with message passing

■ 通信が必要なのはhopping $D\psi$ の計算部分

$$D = 1 - M_{ab,\alpha\beta}(n,m) \qquad a,b = 1 \sim 3 : \text{color} \qquad \gamma_\mu : 4 \times 4$$
$$\alpha,\beta = 1 \sim 4 : \text{spin} \qquad U_\mu : 3 \times 3$$

$$M_{ab,\alpha\beta}(n,m) = \sum_{\mu=1}^{4} \left[(1-\gamma_{\mu,\alpha\beta}) U_{\mu,ab}(n) \delta_{n+\hat{\mu},m} + (1+\gamma_{\mu,\alpha\beta}) U^{\dagger}_{\mu,ab}(m) \delta_{n-\hat{\mu},m} \right]$$



com1が行う計算に必要なデータ

- com1がデータを持っていないのでcom2からデータを受け取らなければならない
  - 通信
- ○ の計算はcom2からデータが届くのを待っている間に計算できる

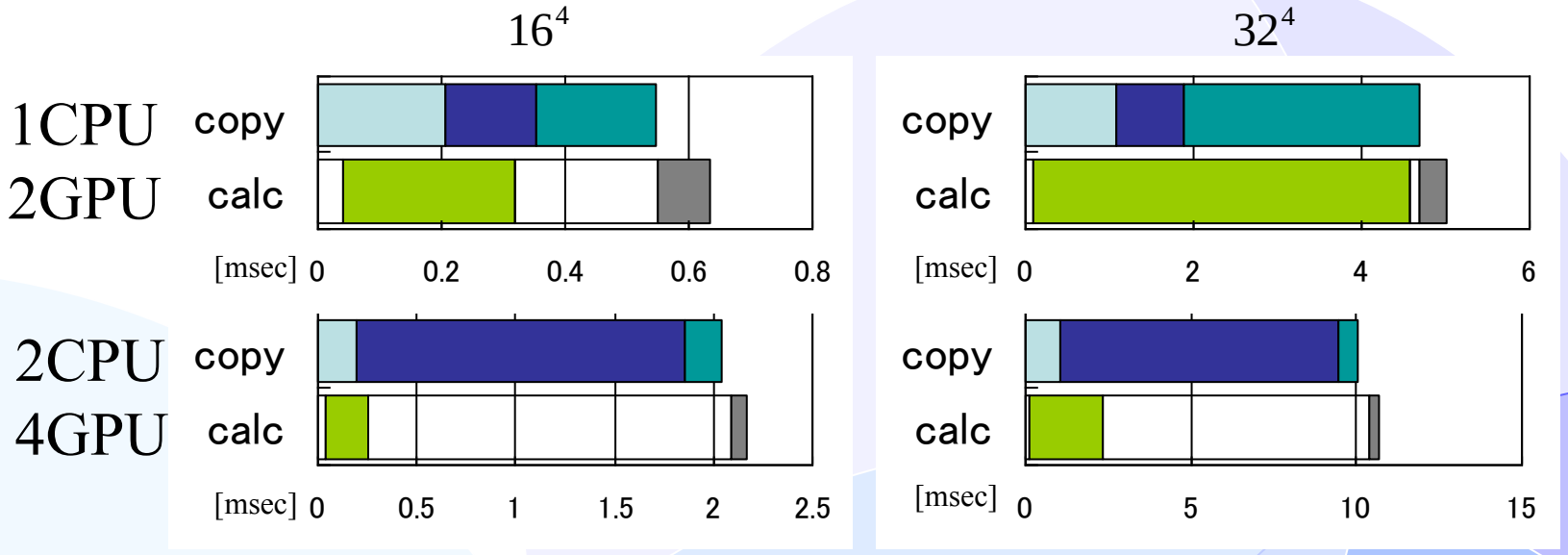
○ の計算時間 ≥ 通信時間 ⇒ 効率的

GPUの計算でも計算と通信の同時実行可能　　GPU+MPIでもOK

前頁の結果は計算と通信を同時実行していたのだが…

2. 5 Parallel execution comm. & calc. at hopping part

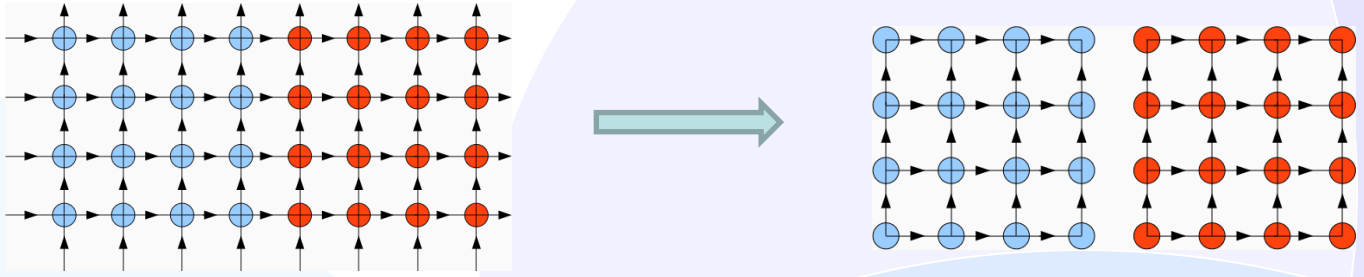
- 計算と通信を同時実行している様子



- 体積32^4, 1CPU, 2GPUでは通信をうまく隠せた
- MPI通信が必要な2CPU, 4GPUではGPUがほとんど遊んでいる
  - MPI通信が遅いため

3. 1 Domain-Decomposition Method

- block Jacobi method
  - Lüscher : Comput.Phys.Commun., Vol.156, pp.209–220, 2004
- 通信が必要な原因は領域をまたがるlink



左図≈右図とおもうことができたなら、左の計算結果≈右の計算結果

$$\begin{pmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{pmatrix} \sim \begin{pmatrix} D_{11} & 0 \\ 0 & D_{22} \end{pmatrix} \Rightarrow \begin{pmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{pmatrix}^{-1} \sim \begin{pmatrix} D_{11}^{-1} & 0 \\ 0 & D_{22}^{-1} \end{pmatrix}$$

通信なしで計算可能

- このまま解として選ぶことはできないが、前処理として利用することで反復回数を減らせる
- D_{ii}^{-1}の領域を隣の領域と重ねることもできる(RAS)

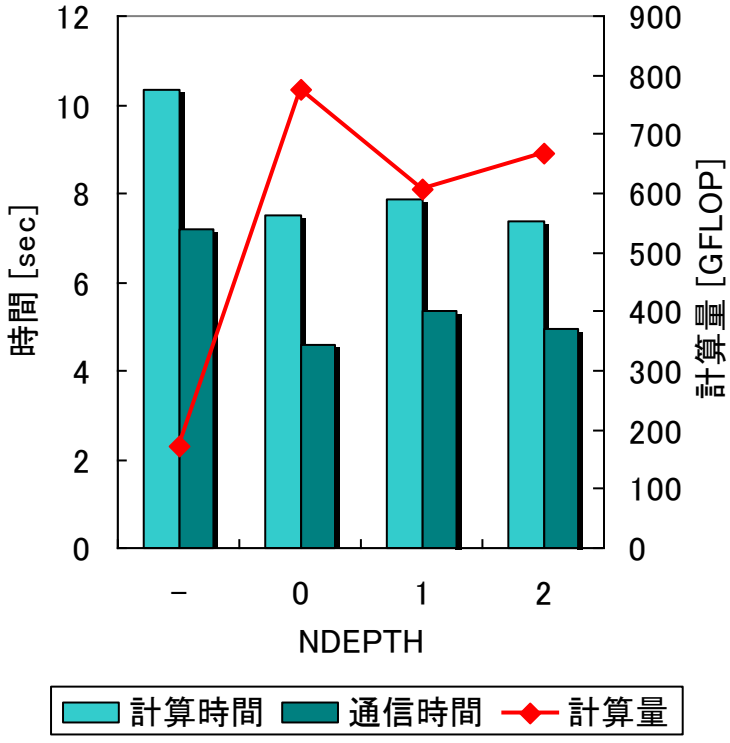
3. 2 Calculation on GPU cluster with DD-Method

- 4CPU
- 8GPU

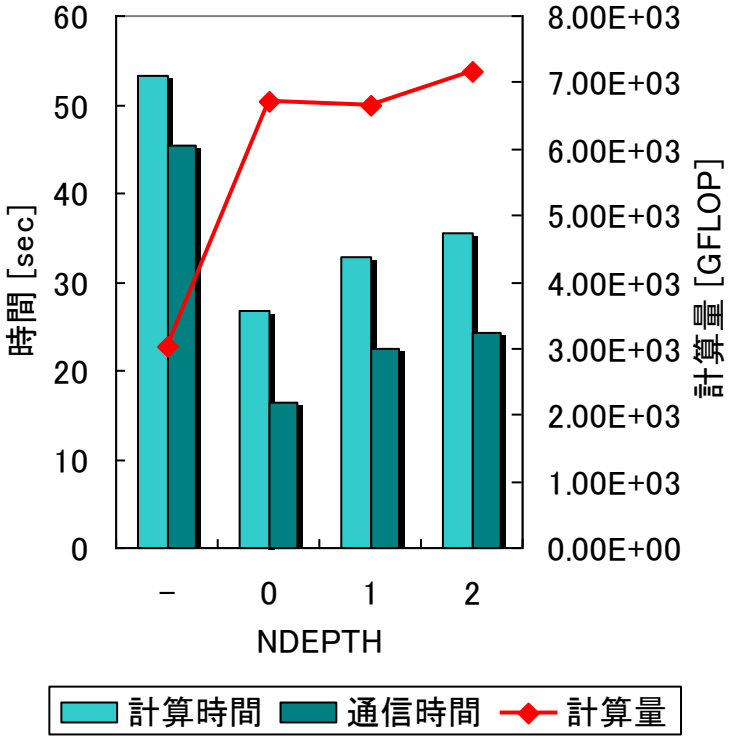
- kappa = 0.126
- csw = 1.0
- accuracy = 10^{-14}

- out solver : Bi-CGStab
- in solver : richardson method, with iteration 5
- RAS iteration : 3

16^4



32^4



4.1 Summury

- GPUは高速でリーズナブルな演算アクセラレータ
  - ゲーム用のデバイス
  - CUDA等の開発環境を用いて科学技術計算
- 今回はGPUを複数台用いてlatticeQCDのsolver計算を行った
  - 直接GPU間の通信を行う方法は現在のところない
  - GPU間の通信は cuda(StreamSDK,OpenCL)+MPI
- 通信(特にEthernet)が遅いため台数効果は得られなかった
  - MPIを使った場合はむしろ遅くなる
- 通信を改善するため領域分割法を試した

4.2 future

- Infiniband 等の高速通信ではどうか？
- チューニングの余地はないか？
  - 通信コードは最適か？
  - harf spin にして通信量を半分にしたらどうか？
- もっと通信量を減らせるアルゴリズムはないか？
  - Multi-Grid？
- Fermi アーキテクチャではどうか？
  - 次世代GPUアーキテクチャ「Fermi」
    - 倍精度演算の強化、ECCサポート、L2キャッシュ等
    - G80/GT200 とはかなり異なるアーキテクチャ
    - Fermiを採用したGeforce GTX 480が3/26（北米）でリリース
    - Fermiを採用したTeslaも第二四半期中にリリース

BACKUP

Overrap comm. & calc in CUDA+MPI

```
thread_fork(); // for Multi-GPU on 1 node

cudaStream_t strm[2]; // 0:calc,1:copy
for(i=0;i<2;i++) cudaStreamCreate(&strm[i]);

cudaMemcpyAsync(…,DeviceToHost,strm[1]);

run_in_kernel<<<BK,TH,d_shared,strm[0]>>>(…);

cudaStreamSynchronize(stream[1]);
thread_barrier();
if(hostThreadId==0) MPI_Sendrecv(…);
thread_barrier();

cudaMemcpyAsync(…,HostToDevice,strm[1]);

cudaThreadSynchronize();

run_out_kernel<<<BK,TH,d_shared,strm[0]>>>(…);

for(i=0;i<2;i++) cudaStreamDestroy(strm[i]);

thread_join();
```

G→C

in calc.

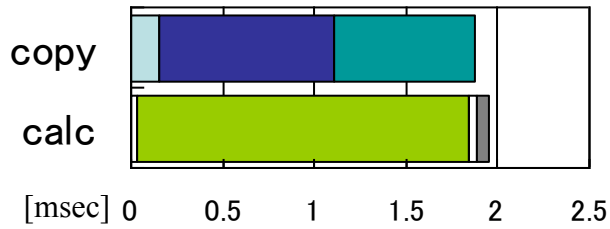
C→C

C→G

out calc.

- cudaMemcpyAsync
- cudaSreamを作る
- copyとkernelの実行にstreamを指定する
- barrierを忘れずに

512×16×16×8
2 CPU, 4GPU
hopping part
use MPI commnication



没

Detail of hopping part with message passing

■ 通信が必要なのはhopping $D\psi$ の計算部分

$$D = 1 - M_{ab,\alpha\beta}(n,m)$$

$a, b = 1 \sim 3$: color　　γ_μ : 4×4

$\alpha, \beta = 1 \sim 4$: spin　　U_μ : 3×3

$$M_{ab,\alpha\beta}(n,m) = \sum_{\mu=1}^{4} \left[(1-\gamma_{\mu,\alpha\beta}) U_{\mu,ab}(n) \delta_{n+\hat{\mu},m} + (1+\gamma_{\mu,\alpha\beta}) U^{\dagger}_{\mu,ab}(m) \delta_{n-\hat{\mu},m} \right]$$

$$\begin{pmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{pmatrix} \begin{pmatrix} \psi_1 \\ \psi_2 \end{pmatrix}$$

com1の領域が担当するquark field

com2の領域が担当するquark field

com1が行う計算

$D_{11}\psi_1 + D_{12}\psi_2$

com1がデータを持っていないので
com2からデータを受け取らなければならない

通信

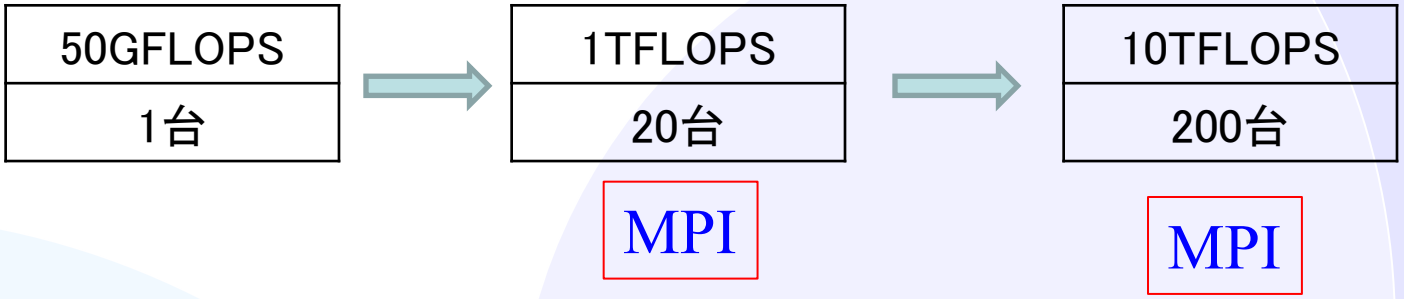
この計算はcom2からデータが届くのを待っている間に計算できる

$D_{11}\psi_1$ の計算時間 $\geq$ 通信時間 $\Rightarrow$ 効率的

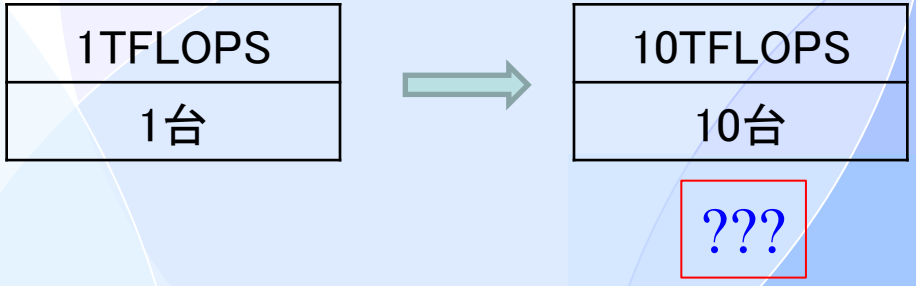
GPU+MPIでもOK

Cluster of GPU

- CPUベースの構成で高性能な計算機



- GPUベースでも同様に…



- GPU間の通信が必要