モンテカルロ殻模型計算のGPGPUへの適用について

富樫智章[A], 清水則孝[A], 宇都野穣[A,B], 阿部喬[C], 大塚孝治[A,C]

東大CNS[A], JAEA[B], 東大理[C]

HPCI戦略プログラム分野5「物質と宇宙の起源と構造」
全体シンポジウム
秋葉原 2014.3.3

背景と目的

・モンテカルロ殻模型計算の現状: 京による大規模計算で5主殻計算が行われ、
6主殻計算が行われつつある。

エキサスケールに向けて、さらに7-8主殻による大規模計算を目指す。

・エキサスケールコンピューティングでは演算加速器(accelerator)を大々的に用いた計算が有望視されている

→ 現状でもGPU (NVIDIA K20X) を用いたTitan や MIC (Intel Xeon Phi 31S1P) を用いたTianhe-2 がスパコンTop500の1,2位を占めている。

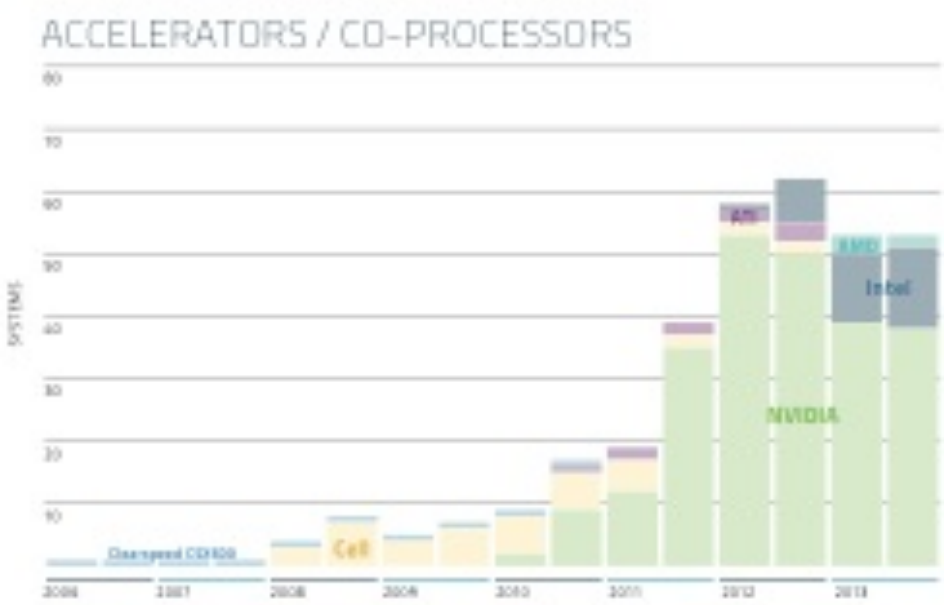
従って、次のエキサスケールに向けて演算加速器(accelerator)により計算効率が飛躍的に向上するようなアルゴリズム開発やチューニングが望まれる。

後述するようにモンテカルロ殻模型計算は行列積演算がボトルネックとなっており、単純な演算で高い性能が出るGPGPUに向いていると考えられる。

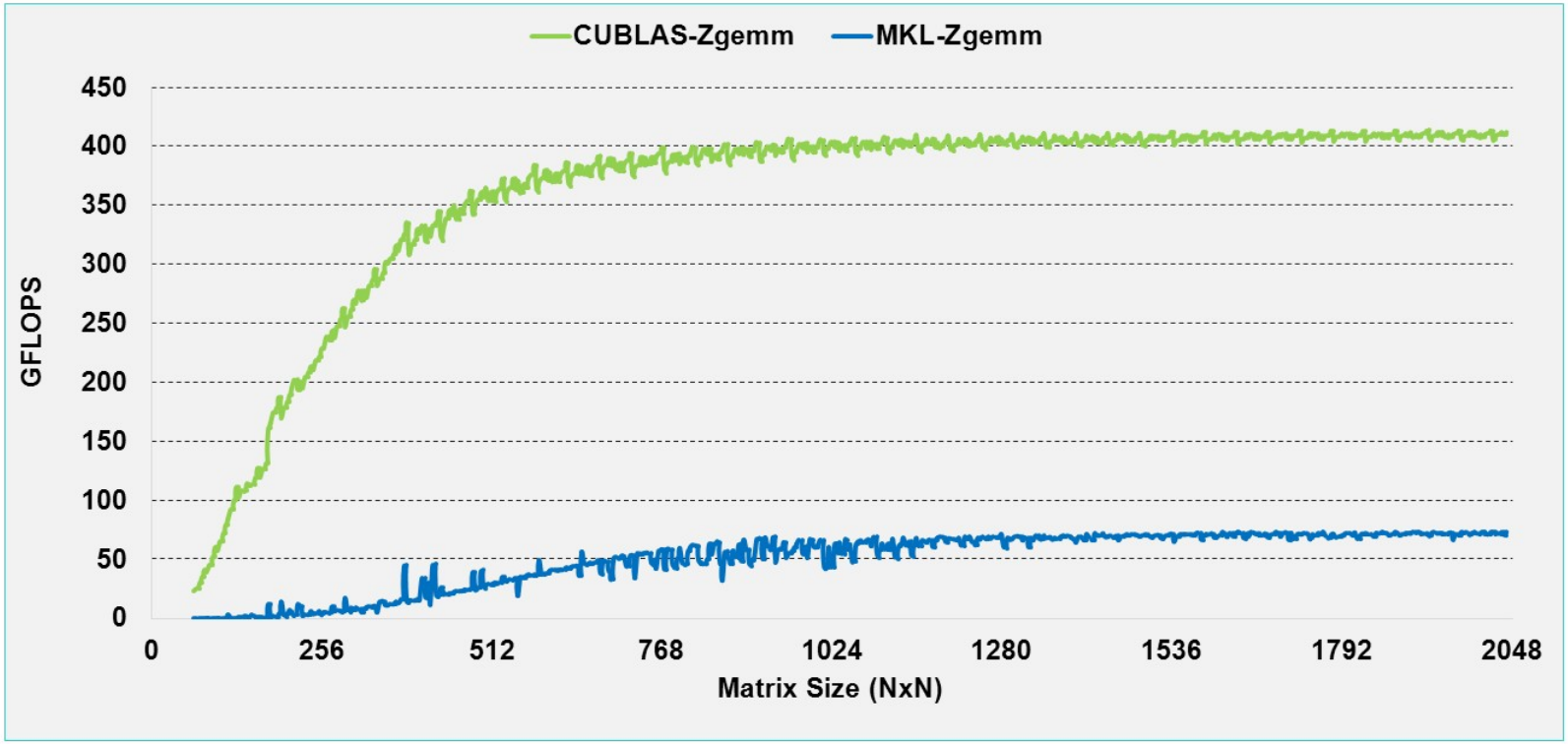
⇒ 今回はGPGPU向けにチューニングを行い性能を確認することが目的

Rank	Computer	Accelerator/Co-Processor Cores
1	TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P	2736000
2	Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x	261632
3	BlueGene/Q, Power BQC 16C 1.60 GHz, Custom	0
4	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect	0
5	BlueGene/Q, Power BQC 16C 1.60GHz, Custom	0
6	Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x	73808
7	PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P	366366
8	BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect	0
9	BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect	0
10	iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR	0
11	Cluster Platform SL390s G7, Xeon X5670 6C 2.930GHz, Infiniband QDR, NVIDIA K20x	57834
12	NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050	100352
13	Atipa Visione IF442 Blade Server, Xeon E5-2670 8C 2.600GHz, Infiniband FDR, Intel Xeon Phi 5110P	171720
14	SGI ICE X, Xeon E5-2670 8C 2.600GHz, Infiniband FDR	0
15	BlueGene/Q, Power BQC 16C 1.60GHz, Custom	0

November 2013 | TOP500 Supercomputer Sites
(http://www.top500.org/lists/2013/11/)
より引用



GPGPUの特徴



• cuBLAS 4.1 on Tesla M2090, ECC on
• MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

•Performance may vary based on OS ver. and motherboard config.

NVIDIA Developer Zone (https://developer.nvidia.com/cuBLAS) より引用

☆GPUは演算量の多い単純な演算を行うのに適している

⇔ 複雑な条件分岐が多いアルゴリズムを処理するのは不得意

※ CPU(ホスト側) ⇆ GPU間のデータ転送速度(メモリバンド幅)はあまり大きくないため、データ転送回数が多くなると性能が落ちてしまう。

モンテカルロ殻模型計算の概要

通常の殻模型計算

$$H = \begin{pmatrix} \star & \star & \star & \star & \star & \cdots \\ \star & \star & \star & \star & & \\ \star & \star & \star & & & \\ \star & \star & & \ddots & & \\ \star & & & & & \\ \vdots & & & & & \end{pmatrix} \Rightarrow \text{対角化} \Rightarrow \begin{pmatrix} E_0 & & & & & 0 \\ & E_1 & & & & \\ & & E_2 & & & \\ & & & \ddots & & \\ & & & & & \\ 0 & & & & & \end{pmatrix}$$

全配位に対する行列要素

次元数 ~ O(10^{10})

モンテカルロ殻模型計算

Review: T. Otsuka , M. Honma, T. Mizusaki, N. Shimizu, Y. Utsuno, Prog. Part. Nucl. Phys. 47, 319 (2001)

$$H \sim \begin{pmatrix} \star & \star & \cdots \\ \star & \ddots & \\ \vdots & & \end{pmatrix} \Rightarrow \text{対角化} \Rightarrow \begin{pmatrix} E'_0 & & 0 \\ & E'_1 & \\ 0 & & \ddots \end{pmatrix}$$

モンテカルロ的+変分計算で基底を選択

次元数 ~ O(100)

エネルギー変分で求める

$$\left|\Phi(q,J^\pi,M,K)\right\rangle = P^{J^\pi}_{MK}\,|\Phi(q)\rangle$$

射影(projection)による
スピンJ・パリティπ状態の回復

$$|\Phi(q)\rangle = \prod_i^{N_f} a_i^\dagger(q)|-\rangle = \prod_i^{N_f}\left(\sum_l^{N_s} D(q)_{li} c_l^\dagger\right)|-\rangle$$

殻模型軌道(0s,0p,...)の生成演算子

※ Hamiltonian行列要素の計算が計算時間の大部分を占める(~95%)

Hamiltonian行列要素の計算

Hamiltonian行列要素は密度行列 ρ と相互作用行列との行列演算となる

射影前の基底: $|\Phi(q)\rangle = \prod_i^{N_f} a_i^\dagger(q)|-\rangle = \prod_i^{N_f} \left(\sum_l^{N_s} D(q)_{li} c_l^\dagger \right) |-\rangle$

軌道数N_s × 粒子数N_f の行列

密度行列: $\rho(q',q) = D(q) \cdot \left(D(q')^\dagger \cdot D(q) \right)^{-1} \cdot D(q')^\dagger$

Hamiltonian行列要素:

$$H(q',q) = \frac{N(q',q)}{\det\left(D(q')^\dagger \cdot D(q) \right)} \left(\sum_{l_1 l_2}^{N_s} t_{l_1 l_2} \, \rho_{l_2 l_1}(q',q) + \frac{1}{2} \sum_{l_1 l_2 l_3 l_4}^{N_s} \rho_{l_3 l_1}(q',q) \, \bar{v}_{l_1 l_2, l_3 l_4} \, \rho_{l_4 l_2}(q',q) \right)$$

1体相互作用行列要素

2体相互作用行列要素

射影後の基底: $P^{J^\pi}_{MK} |\Phi(q)\rangle$

有限角度 Ω^r の3次元回転を行った基底の重ね合わせ(角分点N_m個)に対応

$$H(q',q) = \sum_r^{N_m} W^r H(q',q^r)$$

重み関数値

ボトルネック部分への対応

2体相互作用部分の演算量が一番多い

$$\frac{1}{2}\sum_{l_1 l_2 l_3 l_4}^{N_s} \rho_{l_3 l_1}(q',q^r)\ \bar{v}_{l_1 l_2, l_3 l_4}\ \rho_{l_4 l_2}(q',q^r)$$

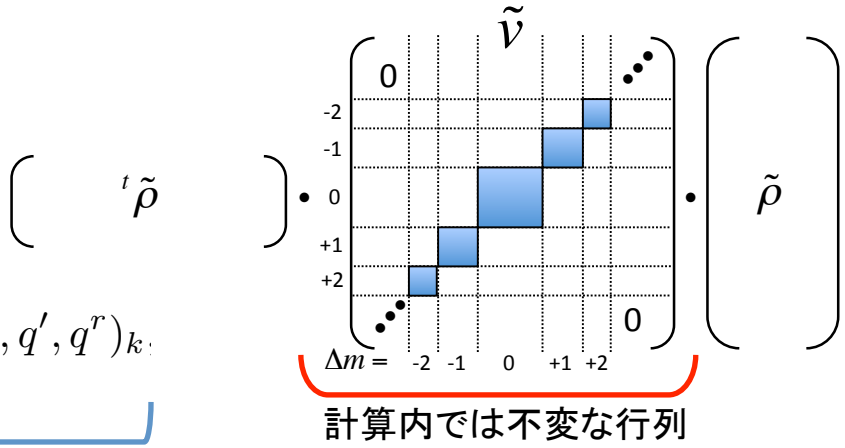
疎行列

行列 × 行列 × 行列

$$= \frac{1}{2}\sum_{\Delta m}\sum_{k'k} \tilde{\rho}\,(-\Delta m, q', q^r)_{k'}\ \tilde{v}\,(-\Delta m, \Delta m)_{k'k}\ \tilde{\rho}\,(\Delta m, q', q^r)_{k}$$

密ブロック行列

ベクトル × 行列 × ベクトル



計算内では不変な行列

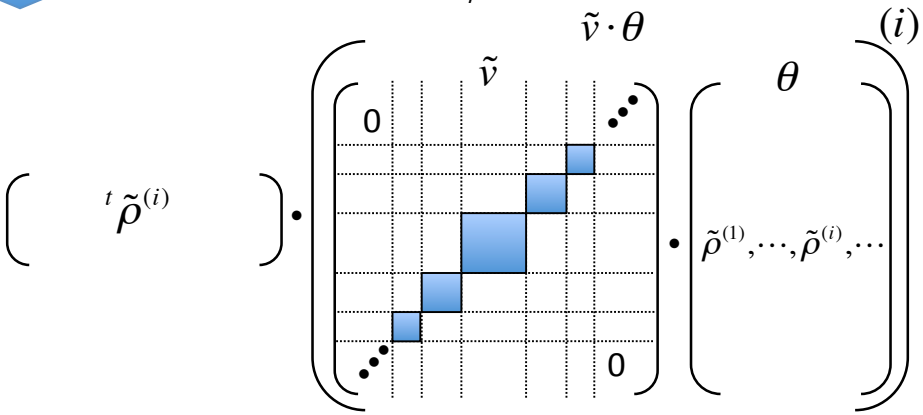
スピンJ・パリティπ射影では角分点ごとに計算を行う

$$H(q',q) = \sum_r^{N_m} W^r H(q',q^r)$$

ある程度の数の角分点の密度行列をまとめて一つの行列 θ として、
2体相互作用行列との行列積を計算

行列積をBLASにより計算(従来法)

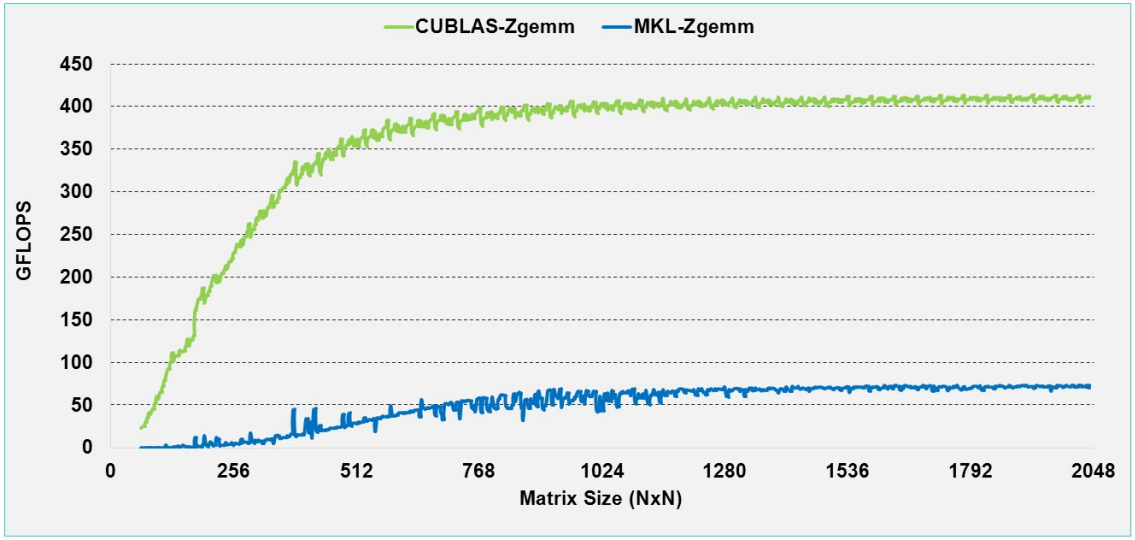
行列積をcuBLASでGPUにより計算



Y. Utsuno, N. Shimizu, T. Otsuka, T. Abe, Comp. Phys. Comm. 184, 102 (2013)

GPUによる行列積演算

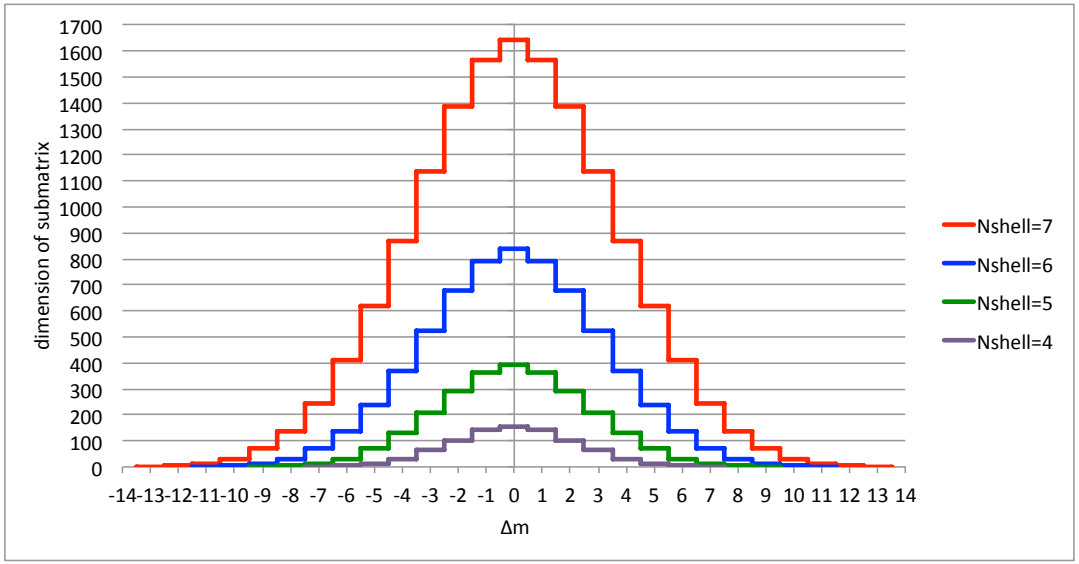
NVIDIA Developer Zone (https://developer.nvidia.com/cuBLAS) より引用



• cuBLAS 4.1 on Tesla M2090, ECC on
• MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

•Performance may vary based on OS ver. and motherboard config.

cuBLASでは大きな次元での行列積の演算性能が飛躍的に向上



主殻(Nshell)ごとの2体相互作用密ブロック行列の次元数

cuBLASを用いることで
より大きな主殻(Nshell)での
性能が向上することが期待
できる

フローチャート

$D(q'), D(q)$ が与えられる

※相互作用のデータ $\tilde{t}, \tilde{v}$ は予めGPUに転送

角分点のメッシュ:

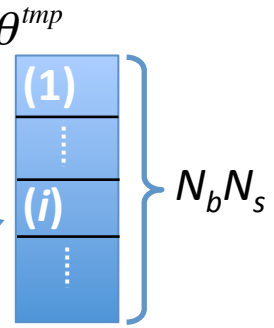
1 | 2 | 3 | … | *ib* | … | *jb* | … | N_m

OpenMPで
スレッド並列化

(1), (2), … (*i*), … (N_b)のループ

$D(q)$ から(*i*) 番目の回転角に対応した $D(q^{(i)})$ を生成し

$D(q^{(i)}) \cdot (D(q')^{\dagger} \cdot D(q^{(i)}))^{-1}$ まで計算を行い θ^{tmp} に保存



配列 $\theta^{tmp}, D(q')^{\dagger}$ をGPUに転送

※配列をまとめて一度に転送

(GPU computing)

1. $\theta^{tmp} \cdot D(q')^{\dagger}$ を計算し、密行列積に対応した形に置換: $\tilde{\theta} = (\tilde{\rho}^{(1)}, \ldots, \tilde{\rho}^{(i)}, \ldots, \tilde{\rho}^{(N_b)})$

2. Hamiltonian行列要素 $H^0(q', q^{(i)})$ を計算: $\underbrace{\left(\tilde{t} \cdot \tilde{\theta}\right)^{(i)}}_{\text{1体演算子}} + \underbrace{{}^{t}\tilde{\rho}^{(i)} \cdot \left(\tilde{v} \cdot \tilde{\theta}\right)^{(i)}}_{\text{2体演算子}}$

計算結果 $H^0(q', q^{(1)}), \ldots, H^0(q', q^{(i)}), \ldots, H^0(q', q^{(N_b)})$ をホスト側に転送

ベンチマークテスト

計算環境

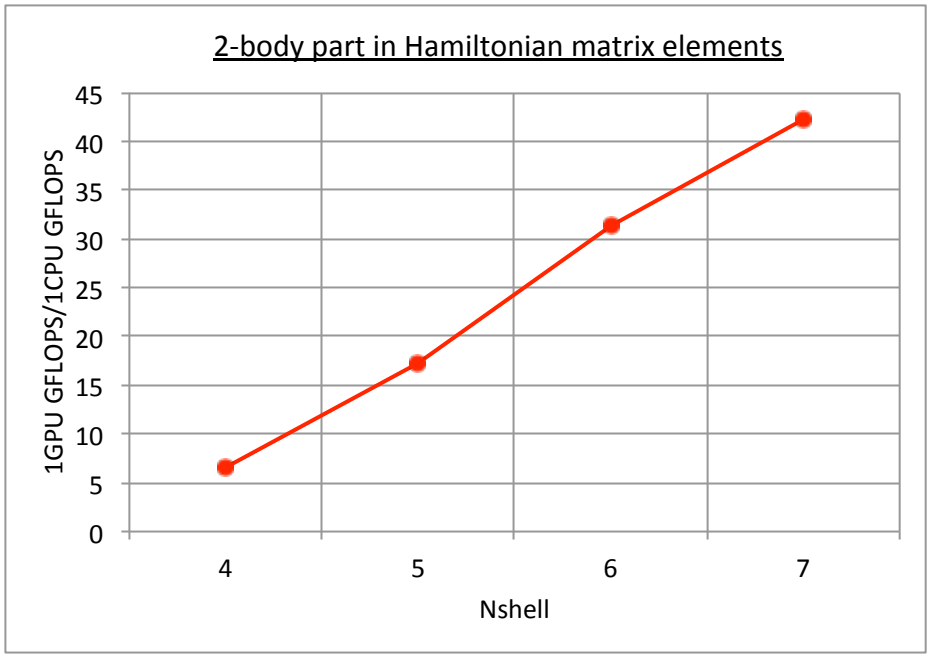
- CPU: AMD Opteron 6274 16 core (理論ピーク性能:17.6 GFLOPS/thread) x 1
- GPU: NVIDIA Tesla K20X (理論ピーク性能(倍精度): 1.31 TFLOPS) x 1
- コンパイラ: PGI Accelerator Fortran 13.10 (NVIDIA CUDA 5.0 使用オプション)
- 1ノード

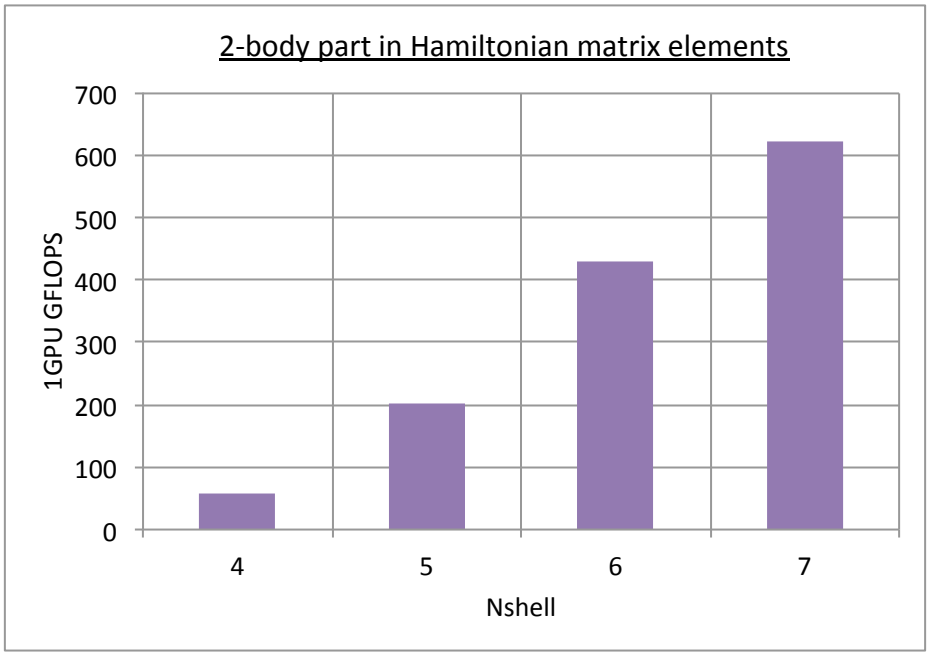
対象とする原子核

^{16}O (陽子数:8, 中性子数:8), スピン・パリティ: 0^+
主殻(Nshell) : 4, 5, 6, 7
相互作用: JIPS16 (A.M.Shirokov *et al.*, PLB644, 33 (2007))
→ 5基底のHamiltonian行列要素の計算(15要素)を行い、
CPUのみの場合とGPUを用いた場合とで性能を比較

$N_{\rm shell}$	N_s	Harmonic-Oscillator Single-Particle Orbits
4	40	$(spdsfp) \equiv (0s_{1/2})_2(0p_{3/2})_4(0p_{1/2})_2(0d_{5/2})_6(0d_{3/2})_4(1s_{1/2})_2$ $(0f_{7/2})_8(0f_{5/2})_6(1p_{3/2})_4(1p_{1/2})_2$
5	70	$(spdsfp) + (gds)$, $(gds) \equiv (0g_{9/2})_{10}(0g_{7/2})_8(1d_{5/2})_6(1d_{3/2})_4(2s_{1/2})_2$
6	112	$(spdsfp) + (gds) + (hfp)$, $(hfp) \equiv (0h_{11/2})_{12}(0h_{9/2})_{10}(1f_{7/2})_8(1f_{5/2})_6(2p_{3/2})_4(2p_{1/2})_2$
7	168	$(spdsfp) + (gds) + (hfp) + (igds)$, $(igds) \equiv (0i_{13/2})_{14}(0i_{11/2})_{12}(1g_{9/2})_{10}(1g_{7/2})_8(2d_{5/2})_6(2d_{3/2})_4(3s_{1/2})_2$

2体相互作用部分の計測





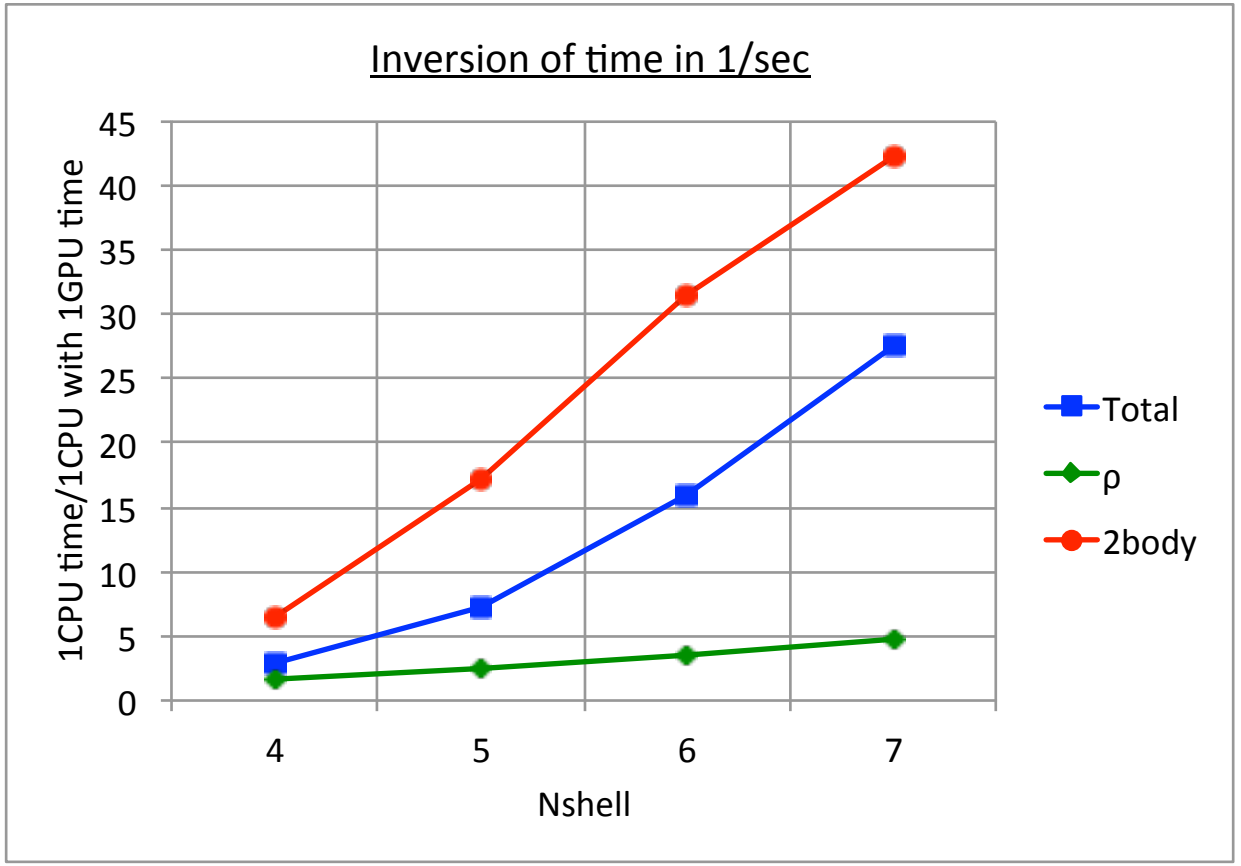
→ 2体相互作用部分についてFlops数を
GPUとCPU 1 threadで比較(GPU/CPU 1thread)

→ 2体相互作用部分についてのGPUのFlops数

より大きな主殻(Nshell)でGPUは性能を発揮
→ Nshell = 6: ~30倍，Nshell = 7: ~42倍

Nshell = 7 で理論ピーク性能の~50%
(~600 GFlops/1.31 TFlops)

Hamiltonian行列要素全体での計測

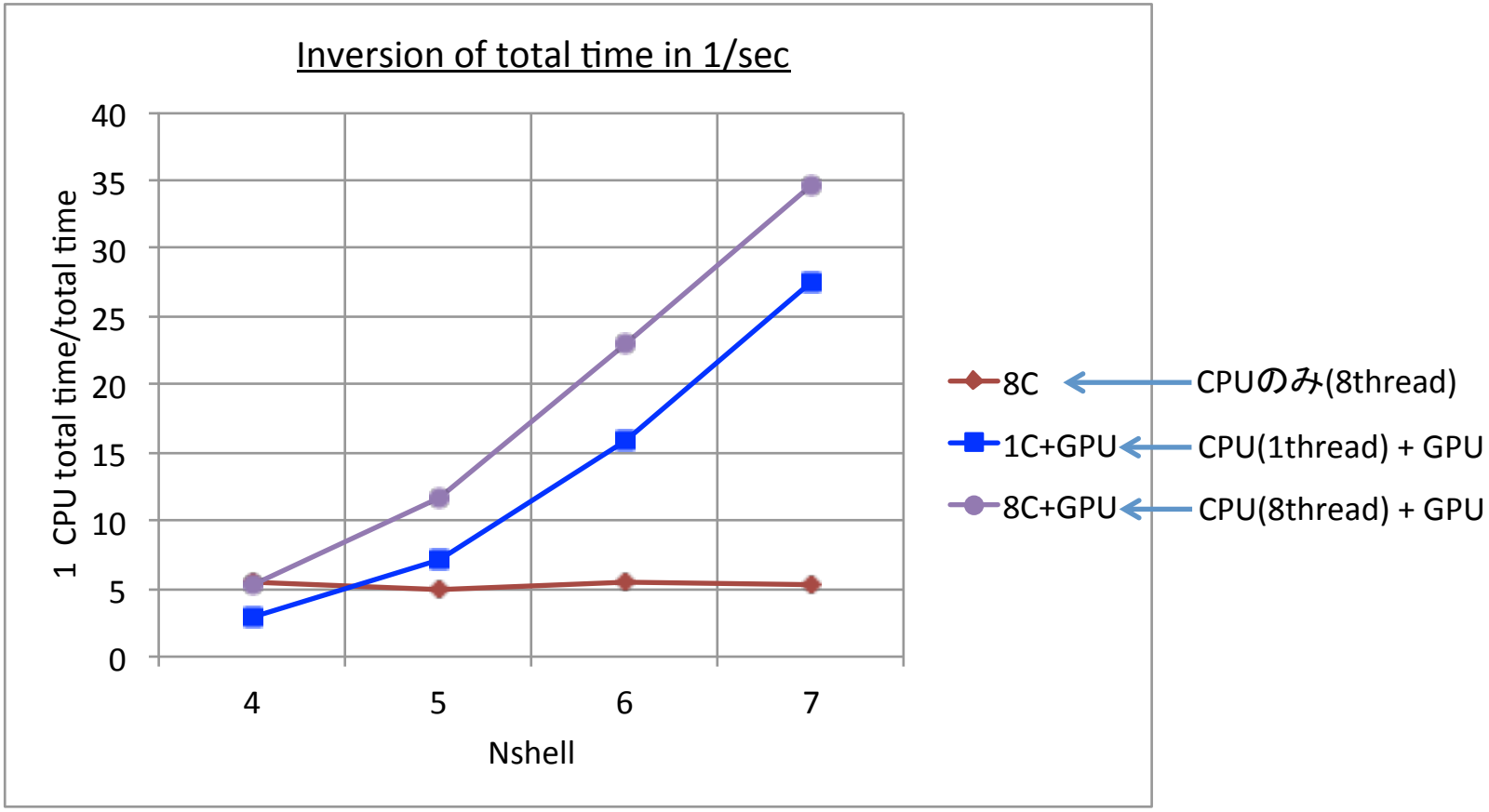


→ Hamiltonian行列要素計算にかかるelapsed timeの逆数を
GPUとCPU 1 threadで比較(GPU+CPU 1thread/CPU 1thread)

全体の計算ではNshell = 7で ~27倍の性能

スレッド並列との併用

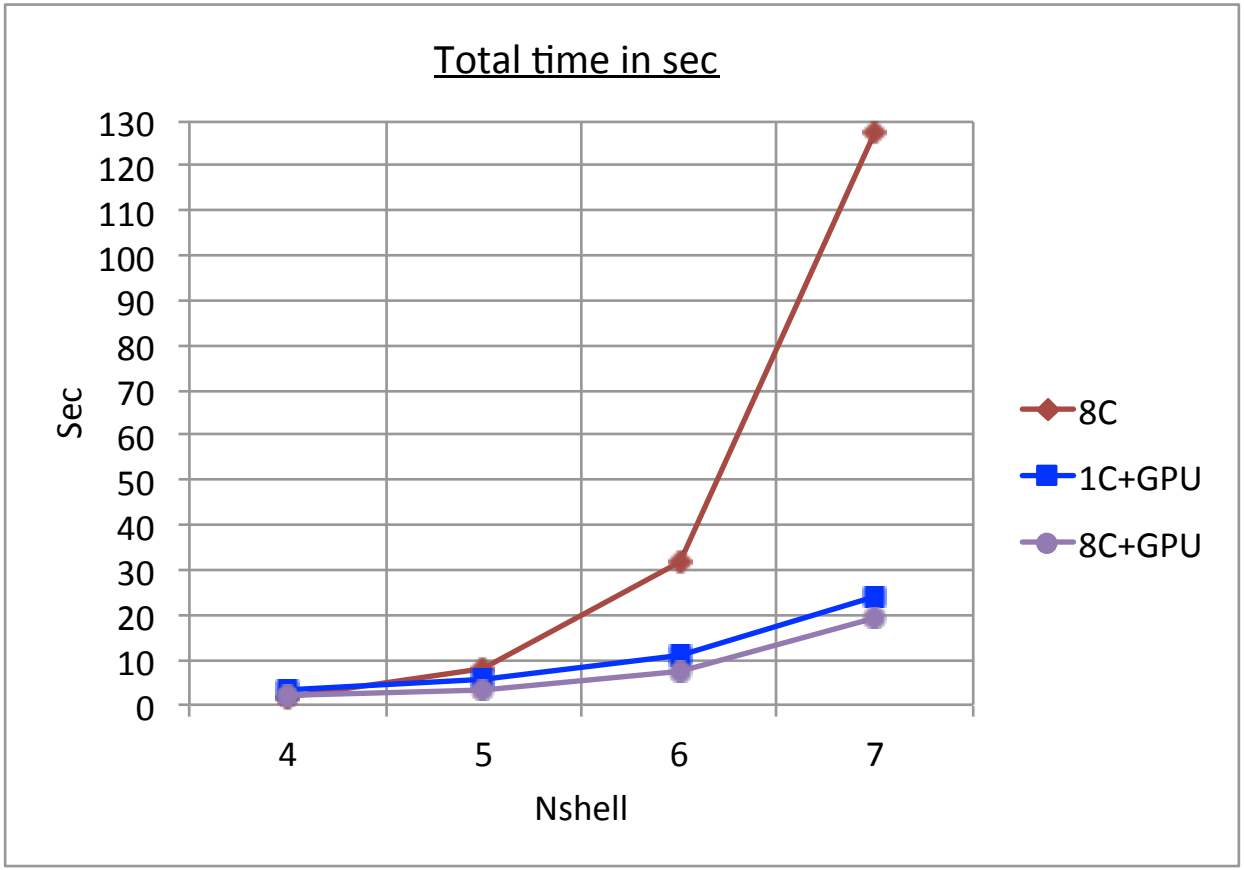
OpenMP 8 threadと併用して、全体の計算にかかるelapsed timeの逆数を取り、CPU 1 threadの場合との性能を比較



Nshell = 4の場合はCPU 8threadとほぼ性能が変わらない
Nshellが大きくなるにつれて大きく性能が向上(OpenMP+GPU, Nshell = 7で~35倍)

実時間計測について

1行列要素当たりの計算にかかった実時間を表示



CPUだけではNshellが増えるにつれて指数関数的に計算時間が増加
GPUを用いると計算時間の増加をある程度抑えることができる

CPUとGPUの計算時間の内訳について

OpenMP 8 thread + GPUでの計算時間の内訳 (1行列要素当たり)

Nshell	Total (sec)	CPU (sec)	GPU (sec)
4	1.80	0.45	1.35
5	3.51	0.70	2.81
6	7.47	0.77	6.70
7	19.21	1.16	18.05

CPUでの計算時間はGPUでの計算時間と比べて小さくオーバーヘッドの問題は(今のところ)起こらない

※一部をCPUで計算している密度行列 ρ は粒子数が多くなるにつれて
計算時間が増えるため、質量数が多くなると考慮する必要があるかもしれない

まとめと展望

- モンテカルロ殻模型計算では、行列積演算部分がボトルネックとなるためGPGPUにより性能が向上させることができる。

- 特にNshellが大きくなり、扱う行列の次元が大きくなるとより性能を発揮しやすくなる。→ 今後の大規模計算に有利になると考えられる

今後の現実的な計算に向けて…

- 多GPU, 多ノード用の計算コードを実装していく。

- 現状では理論ピーク性能の50%程度なので、さらにGPUの性能を引き出すチューニングを試みる。
  例) Dynamic Parallelism