第８回先進スーパーコンピューティング環境研究会（ASE 研究会）開催報告

東京大学情報基盤センター　特任准教授　片桐孝洋

2010 年 12 月 6 日（月）14 時 00 分～17 時 45 分、東京大学情報基盤センター3 階大会議室にて、第 8 回先進スーパーコンピューティング環境研究会（ASE 研究会）が開催されました。

国内の大学･研究機関および企業からの参加者が 20 名あり､活発な議論がなされました。招待講演として、東京女子大学から荻田武史准教授をお呼びし、精度保証計算に関する講演を行いました。また、東京工業大学から下川辺隆史氏をお呼びし、GPU（Graphics Processing Unit）と気象シミュレーションに関する講演を行いました。

荻田武史准教授の講演は、いままで検証なしに行われていた数値計算について、解の存在範囲内の保証をしつつ行う「精度保証計算」の分野の紹介でした。数値線形代数の精度保証計算技術の現状と今後の課題を、わかりやすく紹介されました。スーパーコンピュータを用いたシミュレーションで精度保証しつつ数値計算する事例は、著者の知る限りありません。しかし、数値計算の問題サイズが超大規模化している現状を鑑みると、予期しない精度劣化が生じる可能性があります。今後、重要な研究分野となると思われます。

下川辺隆史氏の講演は､気象シミュレーションプログラムを GPU 向きに改良して実装し、高速化した事例でした。東京工業大学の GPU を搭載したスーパーコンピュータ TSUBAME で性能評価をしています。GPU を搭載した計算機環境での数値計算は、非均質計算（Heterogeneous Computing）と呼ばれ、現在もっとも注目され、かつ、研究されている分野です。次々世代のスーパーコンピューティング環境のエクサスケール・コンピューティング環境では、必須の技術といわれています。

以上の２講演については、本原稿の後に当日の発表資料を掲載します。ご興味のある方は、ご覧ください。

さらに、基盤センターから 1 名の発表と、平成 22 年度前期　若手利用者推薦制度（試行）採択者報告がなされました。参加者間で活発な議論により、交流がなされました。

当日のプログラムを以下に載せます。

● プログラム

【招待講演】 14 時 00 分～15 時 00 分

講演者

荻田武史（東京女子大学）

題目

数値線形代数における精度保証付き数値計算法の現状

要旨

本講演では、数値線形代数、特に連立一次方程式や固有値問題に対する精度保証付き数値計算法の現状について紹介する。

これまでに、密行列系に対しては実用的な精度保証法が開発されてきたが、疎行列系に対しては行列の構造や性質に依存した方法以外では、必ずしも実用的なレベルに到達しているとは言えない。

そこで、本講演では、これまでに提案されてきた精度保証法を概観し、今後の課題について議論する。

【招待講演】 15 時 10 分～16 時 10 分

講演者

下川辺隆史（東京工業大学）

題目

TSUBAME 2.0 の 4000 GPU を用いた次世代気象モデルの大規模高性能計算

要旨

ここ数年、GPU の持つ高速浮動小数点演算能力と高いメモリバンド幅に注目され、GPU を汎用計算に使用する GPGPU 研究が盛んに行われている。

東京工業大学では 2010 年 11 月から TSUBAME 2.0 が稼働を開始した。その計算性能の大部分は 4000 基を超える GPU が担う。

本講演では、気象庁が開発を進める次世代気象シミュレーションコード ASUCA 全体を GPU コード化し、TSUBAME 2.0 で実行させるまでのプロセスと、その実行性能について紹介する。

16 時 30 分～17 時 15 分

講演者

鴨志田良和（東京大学情報基盤センター）

題目

モニタリングツール VGXP を応用したアプリケーション性能のリアルタイム可視化

要旨

並列アプリケーションはプロセスやスレッドなど、動いている対象が多く、複雑だが、直観的な可視化を行うことで、全体を把握した見通し良いデバッグやチューニングが可能になると考えられる。

本発表では並列アプリケーションの動作を、発表者が開発しているモニタリングツールである VGXP を応用して可視化する方法について述べる。可視化の方法については複数の案を検討し、その特徴を述べる。

17 時 15 分～17 時 45 分

平成 22 年度前期　若手利用者推薦制度（試行）　採択者報告

講演者

　坂下達哉（電気通信大学　大学院情報システム学研究科）

題目

　量子 i.i.d.状態のシミュレーションとその理論的考察

要旨

　量子情報理論において極めて重要な量子 i.i.d.状態のシミュレーションを行う。

　独自開発した計算手法の並列性と演算量の削減効果が高い。並列化には MPI を用いる。問題は固有値解法における精度であるが、多倍長化を施すことで精度を確保する。アルゴリズムは既知の問題に適用するだけでなく、未解決問題の収束式の検証にも適用している。

　したがって、学術的にインパクトがある成果の創出が期待できる。

18 時 00 分～　懇親会（根津周辺）

　ASE 研究会の開催情報はメーリングリストで発信をしております。研究会メーリングリストに参加ご希望の方は、ASE 研究会幹事の片桐（katagiri@cc.u-tokyo.ac.jp）までお知らせください。

以上

数値線形代数における精度保証付き数値計算法の現状

荻田 武史
東京女子大学 現代教養学部 数理科学科

第8回ASE研究会
東京大学 情報基盤センター
2010年12月6日

内容

浮動小数点演算と精度保証付き数値計算の概要

数値計算

数値計算：解析的に解くのが困難な問題を数値的に解く計算やその手法。その計算にはコンピュータによる浮動小数点演算を用いるのが普通。

浮動小数点演算： 簡単に言えば、有限桁（たとえば、10進数で16桁程度）の計算のこと。何かの計算をする度に、決められた桁数以下の部分は打ち切られる。

$\Longrightarrow$ 計算を重ねると次第に計算誤差が蓄積していく

$\Longrightarrow$ 最終的に得られた結果がどれくらい正しいかは問題依存

$\Longrightarrow$ 正しい結果とはまったく違った計算結果が得られることもしばしばある

$\Downarrow$

数値計算による結果を、数学的に厳密な誤差限界と共に与えることを精度保証付き数値計算という。

精度保証付き数値計算

- 微分方程式等を数値計算で厳密に解きたい
- 区間演算／区間解析を基礎とする
- 誤差解析（数値計算で起きる誤差を調べる）
  - モデル化誤差（現象 $\Rightarrow$ 数理モデル（微分方程式））
  - 離散化誤差（微分方程式 $\Rightarrow$ 線形方程式）
  - 打ち切り誤差／丸め誤差（線形方程式 $\Rightarrow$ 近似解）

なぜ線形問題を考えるのか？

- コンピュータは，原理的に無限を扱うことができない
- 問題（微分方程式）を離散化する必要がある
- 多くの問題は，線形問題に帰着できる（もちろん，様々な誤差は発生するが）（たとえば，Googleのページランク等）

$\Longrightarrow$ 線形方程式の効率的な数値解法はたくさん考えられてきた

IEEE 754-1985: 2進浮動小数点演算規格

カリフォルニア大学バークレー校のWilliam Kahanを中心として1985年に制定された2進浮動小数点演算の規格。

単精度（32ビット）や倍精度（64ビット）のフォーマット・演算のルール等を定義している。

たとえば倍精度であれば，符号（1ビット），指数部（11ビット），仮数部（52ビット）。浮動小数点数は正規化（仮数部を1.xxxxxxの形にする）を前提に考えるので、仮数部は53ビット分確保。

倍精度の演算精度は、$2^{-53} \approx 1.11 \times 10^{-16}$ となるため、10進数でおよそ16桁程度。

IEEE 754では、浮動小数点数の四則演算をsemimorphismと呼ばれる性質によって定義している。

まず実数から浮動小数点数への丸め（写像）を定義する。

IEEE 754では丸めモードとして、以下の4種類がある：

- 最近点への丸め
- $+\infty$方向への丸め
- $-\infty$方向への丸め
- 原点方向への丸め

$\mathbb{R}$： 実数の集合
$\mathbb{F}$： 浮動小数点数の集合

$r \in \mathbb{R}$に対し，それぞれの丸めモードは以下のように定義される：

最近点への丸め $\bigcirc : \mathbb{R} \to \mathbb{F}$は，$\bigcirc(r)$を$r$に最も近い$f \in \mathbb{F}$とする．
$+\infty$方向への丸め $\triangle : \mathbb{R} \to \mathbb{F}$は，$\triangle(r)$を$f \geqq r$を満たす$f \in \mathbb{F}$で最も小さいものとする．
$-\infty$方向への丸め $\triangledown : \mathbb{R} \to \mathbb{F}$は，$\triangledown(r)$を$f \leqq r$を満たす$f \in \mathbb{F}$で最も大きいものとする．
原点方向への丸め $\square : \mathbb{R} \to \mathbb{F}$は，$\square(r)$を$|f| \leqq |r|$を満たす$r$に最も近い$f \in \mathbb{F}$とする．

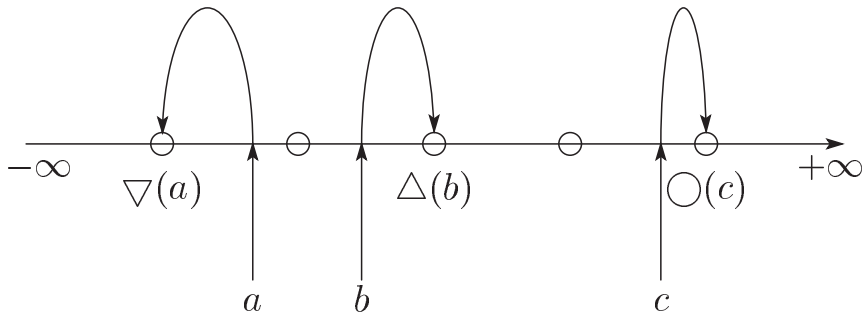


Figure 1: 丸めモードのイメージ．数直線上の$\circ$は離散的に分布している浮動小数点数を表している．

次に、各丸めモードごとに浮動小数点数の四則演算を定義する。

丸めモードを指定したとき、その下での浮動小数点数の四則演算は以下のように定義される。たとえば、$a, b \in \mathbb{F}$に対して、最近点への丸めモードでの加算$\oplus$は実数上の加算$+$を使って

$$a \oplus b = \bigcirc(a + b)$$

が満たされるように定義される。減算や乗除算についても同様。

この他に定められているのは、平方根のみ。
IEEE 754-2008では、FMA命令（積和命令）も規格化。
初等関数は、推奨はするが、要請はしない。

IEEE 754規格があると何がうれしいか？

誤差解析の観点から言えば、近似式ではなく不等式が使えるようになる。

任意の$a, b \in \mathbb{F}$に対して、最近点への丸めだけでは

$$\bigcirc(a+b) \approx a+b$$

丸めモードをうまく利用すると

$$\nabla(a+b) \leqq a+b \leqq \triangle(a+b)$$

$\Longrightarrow$ 丸め誤差に対して、厳密な議論が可能となる。

IEEE 754規格があると何がうれしいか？

- 現在、ほとんどのコンピュータは、IEEE 754規格に準拠している。
- コンピュータによる計算のルールを数学的（理論的かつ厳密的）に捉えられるようになる。
- コンピュータによる計算によって発生する誤差を定量的に計ることができるようになる。

$\Longrightarrow$ 信頼性の高いソフトウェアを作成可能となる。

区間解析

(1950年代) Sunaga： 実数の区間を基礎データ型とする区間演算という概念を導入した．

(1960年代) Moore： 区間解析を学位論文のテーマとし，専門書を出版した．

(2000年頃) Oishi-Rump： 区間演算の単位をベクトルの内積や行列の乗算に拡張した．高速精度保証の始まり．

区間演算

$\mathbb{IR}$： 実数区間の集合
$A = [\underline{a}, \overline{a}] = \{x \in \mathbb{R} : \underline{a} \leq x \leq \overline{a}\} \in \mathbb{IR}$

$A, B \in \mathbb{IR}$ に対して、区間演算 $\circ \in \{+, -, *, /\}$ は

$$A \circ B := \bigcap \{C \in \mathbb{IR} : \text{ すべての } a \in A,\ b \in B \text{ に対して } a \circ b \in C\}$$

ただし、除算では $0 \notin B$ とする。

$A = [\underline{a}, \overline{a}]$, $B = [\underline{b}, \overline{b}]$ に対し

$$\begin{aligned} A + B &= [\underline{a} + \underline{b}, \overline{a} + \overline{b}] \\ A - B &= [\underline{a} - \overline{b}, \overline{a} - \underline{b}] \end{aligned}$$

区間演算を用いた内積計算

$x, y \in \mathbb{F}^n$ に対する古典的な区間演算による内積$x^T y$の計算（$x^T y \in [s]$）

$$
\begin{array}{l}
\textit{function}\ [s] = \texttt{IncDot}(x, y) \\
\quad \underline{s} = 0; \quad \overline{s} = 0; \\
\quad \textit{for}\ i = 1 : n \\
\quad\quad \texttt{setround}(-1); \\
\quad\quad \underline{s} = \mathrm{fl}_{\bigtriangledown}\left(\underline{s} + x_i * y_i\right); \quad \% \text{ lower bound of } x^T y \\
\quad\quad \texttt{setround}(+1); \\
\quad\quad \overline{s} = \mathrm{fl}_{\bigtriangleup}\left(\overline{s} + x_i * y_i\right); \quad \% \text{ upper bound of } x^T y \\
\quad \textit{end for} \qquad\qquad\qquad\qquad\quad \% \ [s] = [\underline{s}, \overline{s}]
\end{array}
$$

丸めモードの変更：$2n$回

区間演算は万能？

浮動小数点演算をすべて区間演算に置き換えれば良いか？

⇓

区間ガウスの消去法を代表として、多くの反例がある。

精度保証付き数値計算の基本原理

1. 通常の浮動小数点演算による結果をできるだけ利用する
   - 通常の近似計算による結果は大抵良い
   - 高速な数値計算ライブラリ（BLAS/LAPACKなど）の利用
2. 区間演算は、できるだけ避ける
   - 計算速度の大幅な低下を防ぐ
3. 区間演算は、できるだけ後で使う
   - 得られた計算結果の区間幅の増大を防ぐ

ベクトル化区間演算(1)

$x, y \in \mathbb{F}^n$ に対するベクトル化区間演算による内積$x^T y$の計算（$x^T y \in [s]$）

$$
\begin{array}{l}
\textit{function}\ [s] = \texttt{FastIncDot}(x, y) \\
\quad \texttt{setround}(-1); \\
\quad \underline{s} = \mathrm{fl}_{\bigtriangledown}\left(x^T y\right); \quad \% \text{ lower bound of } x^T y \\
\quad \texttt{setround}(+1); \\
\quad \overline{s} = \mathrm{fl}_{\bigtriangleup}\left(x^T y\right); \quad \% \text{ upper bound of } x^T y
\end{array}
$$

丸めモードの変更：2回

ベクトル化区間演算(2)

$A \in \mathbb{F}^{m\times p}$, $B \in \mathbb{F}^{p\times n}$ に対するベクトル化区間演算による行列積 $A \cdot B$ の計算（$\underline{C} \leqq AB \leqq \overline{C}$）

$$
\begin{array}{l}
\textit{function}\ [C] = \texttt{FastIncMM}(A, B) \\
\quad \texttt{setround}(-1); \\
\quad \underline{C} = \mathrm{fl}_{\triangledown}(A * B); \quad \% \text{ lower bound of } A \cdot B \\
\quad \texttt{setround}(+1); \\
\quad \overline{C} = \mathrm{fl}_{\triangle}(A * B); \quad \% \text{ upper bound of } A \cdot B
\end{array}
$$

丸めモードの変更：2回
高速なBLAS（行列計算ライブラリ）が利用可

どれくらい計算時間が違うか

Table 1: 行列乗算の計算時間(sec)とMFlops: Intel Core 2 Extreme X9650 (3.0 GHz, quad-core), gcc 4.12, GotoBLAS v1.25 [JJIAM, 26:2–3 (2009)]

n	BLAS (DGEMM)	ベクトル化区間演算	古典区間演算
500	$1.32 \cdot 10^{-2}$ $(18.9 \cdot 10^3)$	$2.66 \cdot 10^{-2}$ $(18.8 \cdot 10^3)$	7.71 (65)
1000	$5.09 \cdot 10^{-2}$ $(39.3 \cdot 10^3)$	0.10 $(39.1 \cdot 10^3)$	62.55 (64)
2000	0.37 $(42.9 \cdot 10^3)$	0.75 $(42.7 \cdot 10^3)$	495.2 (65)
5000	5.66 $(44.2 \cdot 10^3)$	11.38 $(43.9 \cdot 10^3)$	7892 (63)
10000	44.73 $(44.7 \cdot 10^3)$	89.45 $(44.7 \cdot 10^3)$	$>$ half a day $(-)$

連立一次方程式に対する精度保証付き数値計算法

準備

- $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n$

$$
|\boldsymbol{x}| = (|x_1|, |x_2|, \ldots, |x_n|)^T \in \mathbb{R}^n
$$
$$
\|\boldsymbol{x}\|_\infty = \max_{1\leq i\leq n} |x_i|
$$

- $A = (a_{ij}) \in \mathbb{R}^{m\times n}$

$$
|A| = (|a_{ij}|) \in \mathbb{R}^{m\times n}
$$
$$
\|A\|_2 = \sqrt{\lambda_{\max}(A^T A)}
$$
$$
\|A\|_\infty = \max_{1\leq i\leq m} \sum_{1\leq j\leq n} |a_{ij}|
$$

- $A=(a_{ij}),\ B=(b_{ij})\in\mathbb{R}^{m\times n}$

$$A\le B \iff \text{すべての } (i,j) \text{ について } a_{ij}\le b_{ij}$$

- $A\in\mathbb{C}^{n\times n}$, Aの固有値を$\lambda_i(A)$, $i=1,2,\dots,n$とする。

$$\lambda_{\min}=|\lambda_1|\le|\lambda_2|\le\cdots\le|\lambda_n|=\lambda_{\max}$$

連立一次方程式の数値解の精度保証

$A\in\mathbb{R}^{n\times n}$, $\boldsymbol{b}\in\mathbb{R}^n$ に対して $A\boldsymbol{x}=\boldsymbol{b}$ を解く（Aは正則）

$$\Longrightarrow\begin{cases}\boldsymbol{x}^* & :\text{真の解（厳密解 } x^*=A^{-1}b)\\ \widetilde{\boldsymbol{x}} & :\text{近似解（数値解）}\end{cases}$$

$A\boldsymbol{x}=\boldsymbol{b}$ の近似解 $\widetilde{\boldsymbol{x}}$ の精度保証

$$\|A^{-1}\|\le\alpha \implies \|\widetilde{\boldsymbol{x}}-\boldsymbol{x}^*\|_\infty\le\varepsilon\in\mathbb{R}\ \text{（ノルム評価）}$$
$$\implies |\widetilde{\boldsymbol{x}}-\boldsymbol{x}^*|\le\boldsymbol{\epsilon}\in\mathbb{R}^n\ \text{（要素毎評価）}$$

精度保証法（一般密行列・ノルム評価）

定理 (cf. Oishi-Rump [Numer. Math., 90 (2002)])

$A\in\mathbb{R}^{n\times n}$, $\boldsymbol{b}\in\mathbb{R}^n$, $R:A^{-1}$ の近似，$I:n\times n$ 単位行列.
$\|RA-I\|_\infty<1$ のとき A は正則で,

$$\|A^{-1}\|_\infty\le\frac{\|R\|_\infty}{1-\|RA-I\|_\infty},$$

$$\|\widetilde{\boldsymbol{x}}-A^{-1}b\|_\infty\le\frac{\|R(A\widetilde{\boldsymbol{x}}-\boldsymbol{b})\|_\infty}{1-\|RA-I\|_\infty}\le\frac{\|R\|_\infty\|A\widetilde{\boldsymbol{x}}-\boldsymbol{b}\|_\infty}{1-\|RA-I\|_\infty}.$$

Oishi-Rumpによって，精度保証が $\frac{2}{3}n^3$ flops（近似解を得るのと同じ計算量）でできることが示された.

精度保証法（一般密行列・成分毎評価）

定理 (Yamamoto [JJAM, 1 (1984)])

$A\in\mathbb{R}^{n\times n}$, $\boldsymbol{b}\in\mathbb{R}^n$, $R:A^{-1}$ の近似，$I:n\times n$ 単位行列,
$e=(1,1,\dots,1)^T\in\mathbb{R}^n$
$\|RA-I\|_\infty<1$ のとき

$$|\widetilde{\boldsymbol{x}}-A^{-1}b|\le|R(A\widetilde{\boldsymbol{x}}-b)|+\frac{\|R(A\widetilde{\boldsymbol{x}}-\boldsymbol{b})\|_\infty}{1-\|RA-I\|_\infty}|RA-I|e.$$

計算量は、成分毎の評価とほとんど変わらない。

精度保証法（一般密行列・Krawczyk-Rumpの方法）

定理 (Rump [Ph.D thesis, 1980])

$A, R \in \mathbb{R}^{n\times n}$, $b, \widetilde{x} \in \mathbb{R}^n$
$[e] \in \mathbb{IR}^n$： $[e] \neq \emptyset$な有界閉区間ベクトル
もし

$$[y] := R(b - A\widetilde{x}) + (I - RA)[e] \subseteq \mathrm{int}([e]),$$

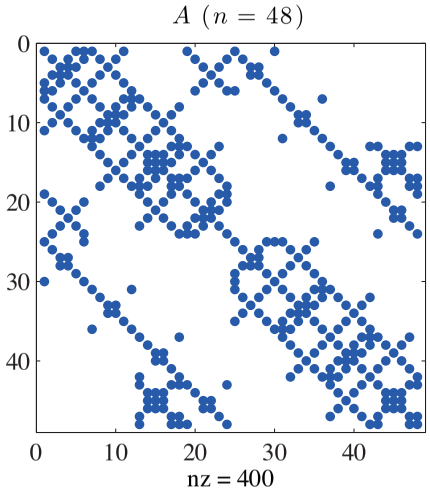
ならば、A及びRは正則で、$Ax = b$の厳密解$x^* = A^{-1}b$ は$x^* \in \widetilde{x} + [y]$をみたす。
ただし、$\mathrm{int}([e])$は$[e]$の内点。

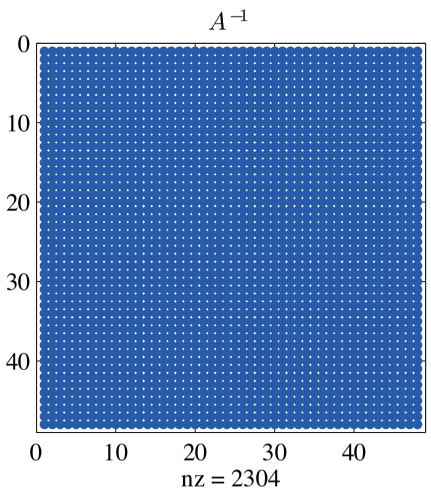
連立一次方程式についての精度保証で一番難しいところ

$\Longrightarrow$ 係数行列が疎行列（非ゼロ要素が極端に少ない行列）の場合

疎行列のデータを格納するのに必要なメモリ量: $\mathcal{O}(n)$

- 大規模問題の数値計算では，反復解法がよく用いられる
  - 計算量: $\mathcal{O}(n) \sim \mathcal{O}(n^2)$，メモリ量: $\mathcal{O}(n)$
- 本質的に逆行列の計算が超非効率的
  - 計算量: $\mathcal{O}(n^3)$，メモリ量: $\mathcal{O}(n^2)$





理想的な精度保証付き数値計算

従来の数値計算（近似解法）と比較して，計算量（計算複雑度）やメモリ量（空間複雑度）が極端に増えない

理想的な精度保証付き数値計算

従来の数値計算（近似解法）と比較して，計算量（計算複雑度）やメモリ量（空間複雑度）が極端に増えない

⇓

できれば２倍〜数倍くらいで

理想的な精度保証付き数値計算

従来の数値計算（近似解法）と比較して，計算量（計算複雑度）やメモリ量（空間複雑度）が極端に増えない

⇓

できれば２倍〜数倍くらいで

⇓

（欲を言えば，同じくらいで）

高速精度保証の現状

密行列	直接解法	非対称	◯ Oishi-Rump (2002)
		対称正定値	◯ Rump (1994), Rump-Ogita (2007)
		対称	△（非対称用の方式）
疎行列	直接解法	非対称	△ Rump (1994)
		対称正定値	◯ Rump (1994), Rump-Ogita (2007)
		対称	△ Rump (1995)
	反復解法	対角優位	◯
		単調行列*	◯ Ogita-Oishi-Ushiro (2001)
		H-行列*	◯ Ning-Kearfott (1997) ◯ Ogita-Oishi (2006)
		それ以外	×

*) 与えられた行列が，単調行列／H-行列であることを示すのはtrivialではない．

精度保証法（単調疎行列）

定理 (Ogita-Oishi-Ushiro [Computing, Suppl. 15 (2001)])

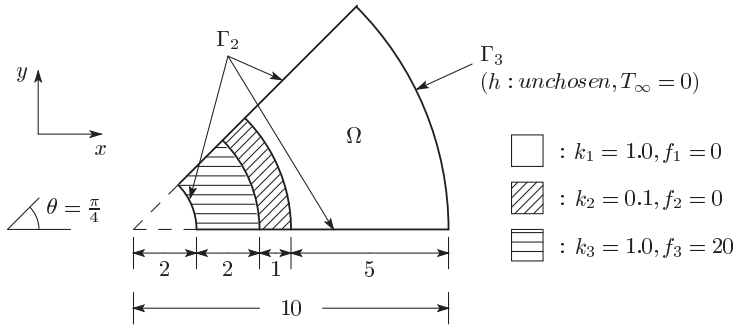
$A \in \mathbb{R}^{n\times n}$：単調行列（$A^{-1} \geq O$），$\boldsymbol{b} \in \mathbb{R}^n$,
$\boldsymbol{e} = (1, 1, \ldots, 1)^T \in \mathbb{R}^n$，$\widetilde{\boldsymbol{y}}$：$A\boldsymbol{y} = \boldsymbol{e}$ の近似解．
$\|A\widetilde{\boldsymbol{y}} - \boldsymbol{e}\|_\infty < 1$ のとき，

$$\|A^{-1}\|_\infty \leq \frac{\|\widetilde{\boldsymbol{y}}\|_\infty}{1 - \|A\widetilde{\boldsymbol{y}} - \boldsymbol{e}\|_\infty}.$$

- $A\boldsymbol{y} = \boldsymbol{e}$ を解くだけなので，$A\boldsymbol{x} = \boldsymbol{b}$ と同じソルバを適用できる．
- $\|A\widetilde{\boldsymbol{y}} - \boldsymbol{e}\|_\infty < 1$ さえ満たせば良い（反復解法向き）．

数値例（単調疎行列）

- $A, \boldsymbol{b}$: 2次元Poisson方程式を有限要素法で離散化
- 問題サイズ n は $10,000 \sim 250,000$
- MICCG法（停止条件: $\dfrac{\|A\widetilde{\boldsymbol{x}} - \boldsymbol{b}\|_2}{\|\boldsymbol{b}\|_2} \leq 10^{-12}$, $\|A\widetilde{\boldsymbol{y}} - \boldsymbol{e}\|_\infty \leq 10^{-3}$）



$$\begin{cases} \mathrm{div}\{-k \cdot \mathrm{grad}(u)\} = f & \text{in } \Omega \\ \{-k \cdot \mathrm{grad}(u)\} \times \boldsymbol{n} = 0 & \text{on } \Gamma_2 \\ \{-k \cdot \mathrm{grad}(u)\} \times \boldsymbol{n} = h(u - T_\infty) & \text{on } \Gamma_3 \end{cases}$$

Table 2: Intel Celeron 566MHz [Computing, Suppl. 15 (2001)]

次元 (n)	近似解 (sec)	精度保証 (sec)	相対誤差
10,000	3.3	1.7	4.1×10^{-10}
40,000	27.1	10.2	2.5×10^{-9}
90,000	90.7	32.3	7.1×10^{-9}
160,000	216.2	77.0	1.6×10^{-8}
250,000	458.5	146.8	3.3×10^{-8}

Table 3: HITACHI SR8000 (8 GFLOPS) [Computing, Suppl. 16 (2002)]
(A) 近似解の計算, (B) $\|A^{-1}\|_\infty$ の上限, (C) 近似解の精度保証

次元 (n)	(A)	(B)	(C)	(B) + (C)	Total
40,000	0.97	0.54	0.65	1.19	2.16
160,000	2.92	1.65	1.97	3.62	6.54
360,000	20.36	10.54	15.99	26.53	46.89
640,000	51.97	24.34	34.62	58.96	110.93
1,000,000	99.23	46.52	65.76	112.28	211.51

精度保証法（H-行列）

M-行列

$A = (a_{i,j}) \in \mathbb{R}^{n \times n}$ に対し，$a_{i,j} \leq 0$ for $i \neq j$，A が正則かつ $A^{-1} \geq O$.

AがM-行列 $\Rightarrow$ Aが単調行列

H-行列

$A = (a_{i,j}) \in \mathbb{R}^{n \times n}$ の比較行列 $\mathcal{M}(A) = (\widetilde{a}_{i,j})$ が M-行列.

$$\widetilde{a}_{i,j} = \begin{cases} |a_{i,j}| & (i = j) \\ -|a_{i,j}| & (i \neq j) \end{cases}.$$

A がH-行列ならば

$$|A^{-1}| \leq \mathcal{M}(A)^{-1} \Longrightarrow \|A^{-1}\|_\infty \leq \|\mathcal{M}(A)^{-1}\|_\infty$$

精度保証法（その他）

- （対称）$\lambda_i(A)$: 固有値

$$\|A^{-1}\|_2 = \frac{1}{\min|\lambda_i(A)|}$$

- （非対称）σ_i: 特異値（A^TAの固有値の平方根）

$$\|A^{-1}\|_2 = \frac{1}{\min \sigma_i(A)}$$

行列固有値問題に対する精度保証付き数値計算法

固有値問題

$A, B \in \mathbb{C}^{n\times n}$について

$$Ax = \lambda x \quad \text{あるいは} \quad Ax = \lambda Bx$$

をみたす$(\lambda, x) \in \mathbb{C} \times \mathbb{C}^n$を求める。

固有値の精度保証

$$|\lambda_i - \widetilde{\lambda_i}| \leqq \varepsilon_i, \quad i = 1, 2, \ldots, n$$

基本的な道具

- Gershgorinの定理
- 相似変換
- Weylの定理

Gershgorinの定理

$A = (a_{ij}) \in \mathbb{C}^{n\times n}$とする。
Aのすべての固有値は

$$\Lambda = \bigcup_{1\le i\le n} U_i \tag{1}$$

に含まれる。ただし

$$U_i = \{z \in \mathbb{C} \mid |z - a_{ii}| \le \sum_{j\neq i} |a_{ij}|\} \tag{2}$$

相似変換

$A \in \mathbb{C}^{n\times n}$とする。正則な行列$X \in \mathbb{C}^{n\times n}$に対して

$$\lambda_i(A) = \lambda_i(X^{-1}AX), \quad i = 1, 2, \ldots, n$$

対称行列$A = A^T \in \mathbb{R}^{n\times n}$の場合は、$D$を$A$のすべての固有値を並べた対角行列、$X$を$A$のすべての固有ベクトルを並べた行列とすると

$$AX = XD \iff X^{-1}AX = X^TAX = D$$

$$D = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix}, \quad X = \left(\begin{array}{c|c|c|c} x_1 & x_2 & \cdots & x_n \\ & & & \end{array}\right)$$

Weylの定理

$A = A^T, B = B^T \in \mathbb{R}^{n\times n}$とする。

$$|\lambda_i(A) - \lambda_i(B)| \le \|A - B\|_2, \quad i = 1, 2, \ldots, n \tag{3}$$

$B = A + E$ならば

$$|\lambda_i(A) - \lambda_i(A + E)| \le \|E\|_2, \quad i = 1, 2, \ldots, n \tag{4}$$

対称行列の固有値の精度保証

数値計算で$AX \approx XD$のようなXとDを得たとすると

$$\begin{aligned} |\lambda_i(A) - \lambda_i(D)| &= |\lambda_i(X^{-1}AX) - \lambda_i(D)| \\ &\le \|X^{-1}AX - D\|_2 = \|X^{-1}(AX - XD)\|_2 \\ &\le \|X^{-1}\|_2\|AX - XD\|_2. \end{aligned} \tag{5}$$

$X^{-1} \approx X^T$であるので、$\|X^TX - I\|_2 < 1$ならば

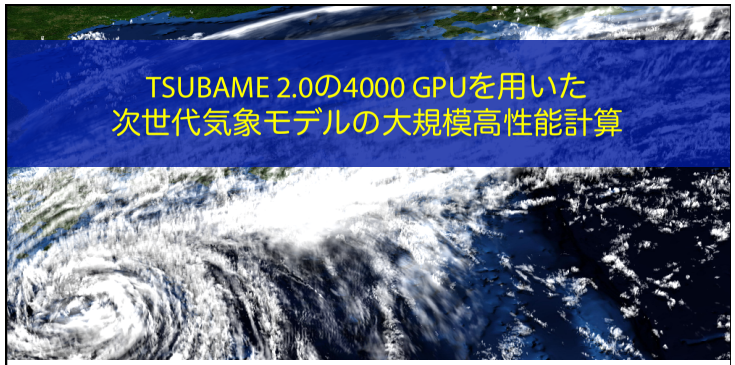
$$\|X^{-1}\|_2 \le \frac{\|X^T\|_2}{1 - \|X^TX - I\|_2}. \tag{6}$$

$\|X^T\|_2 = \|X\|_2$であり、式(6)を式(5)に代入すると

$$|\lambda_i(A) - \lambda_i(D)| \le \frac{\|X\|_2\|AX - XD\|_2}{1 - \|X^TX - I\|_2}.$$

結論

- 精度保証付き数値計算の概要を説明した。
- 連立一次方程式に対する精度保証法の概要について説明した。
- 固有値問題に対する精度保証法について簡単に触れた。



TSUBAME 2.0の4000 GPUを用いた 次世代気象モデルの大規模高性能計算

下川辺　隆史
東京工業大学
創造エネルギー専攻（学術国際情報センター）

Dec. 6, 2010　　第８回ASE研究会（東京大学　情報基盤センター）

もくじ

- GPUとは？ GPUコンピューティングとは？
- 気象シミュレーションコードASUCA
- ASUCAの単一GPUへの実装と計算性能
- TSUBAME 1.2 スパコンを用いたASUCAのマルチGPU計算における最適化手法と計算性能
- TSUBAME 2.0 スパコンを用いたASUCAの計算性能
- まとめ

2

Graphics Processing Unit and GPU Computing

3

What's GPU ?

- Graphics Processing Unit
- もともと PC の3D描画専用の装置
- パソコンの部品として量産されてる。= 非常に安価

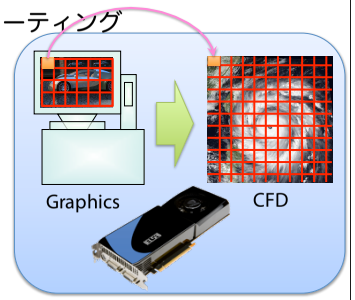
http://www.nvidia.co.jp

4

GPGPU

GPGPU
http://gpgpu.org/

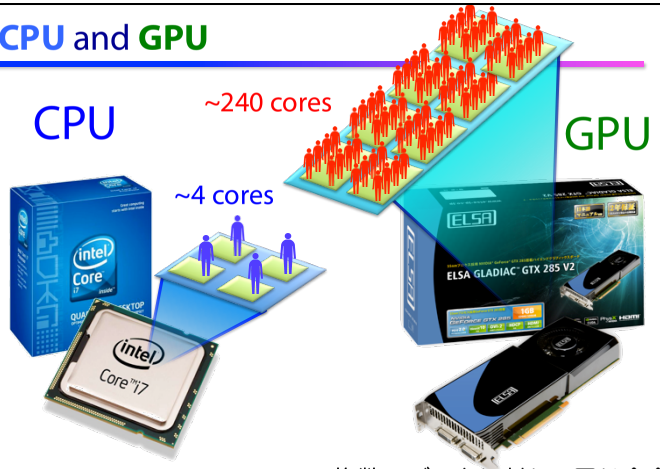
- General Purpose computation on GPU
  汎用GPU計算、GPUコンピューティング
  - ✓ 数値流体力学（CFD）
  - ✓ N体問題
  - ✓ 高速フーリエ変換（FFT）
  - ✓ …
- プログラムはGPUむけの
  開発言語・環境を用いる
  - ✓ CUDA (NVIDIA)
  - ✓ ATI Stream (AMD)
  - ✓ OpenCL (Khronos Group)



GPUによるHigh Performance Computing が現実に。

5

CPU and GPU

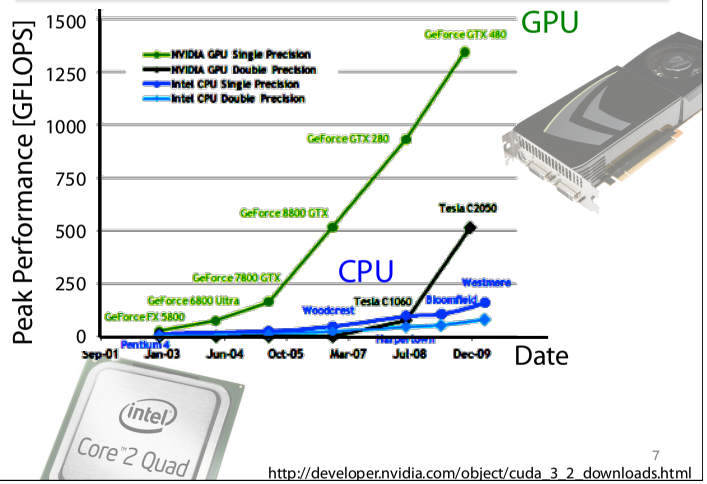


異なるデータに対して異なる命令を実行

複数のデータに対して同じ命令を同時に実行（SPMD, SIMD）

6

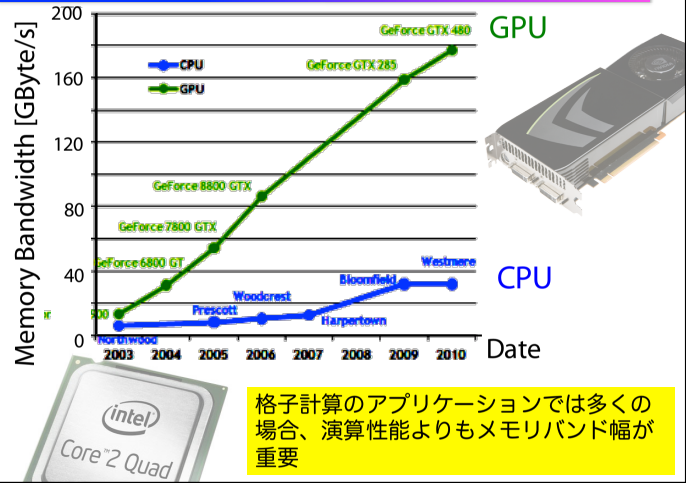
CPU and GPUの演算性能



http://developer.nvidia.com/object/cuda_3_2_downloads.html

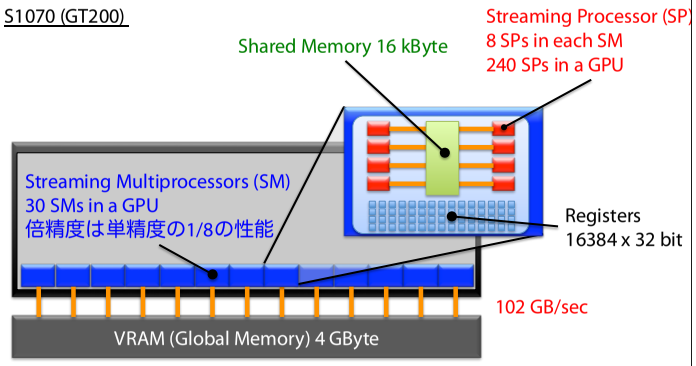
7

CPU and GPUのメモリバンド幅



格子計算のアプリケーションでは多くの場合、演算性能よりもメモリバンド幅が重要

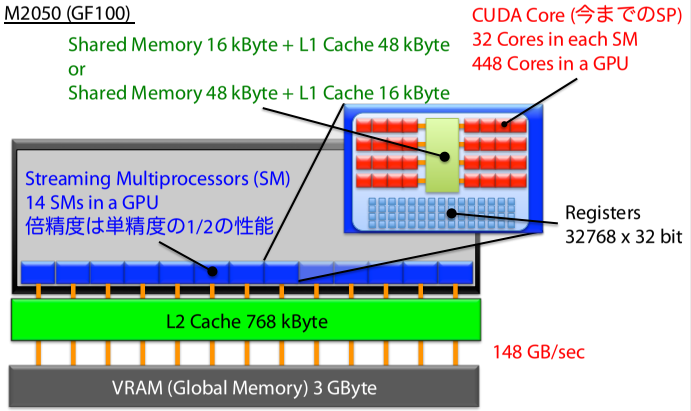
NVIDIA GPU Architecture



VRAM (Global Memory)	400-600 cycle, 4GB
Shared Memory	2 cycle, 16kByte
Register	1 cycle

9

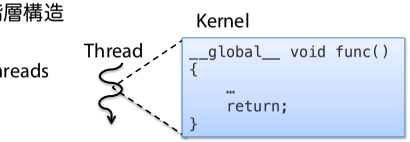
NVIDIA GPU "Fermi" Architecture

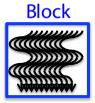


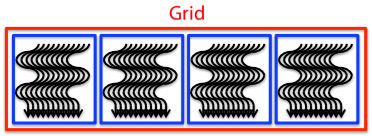
10

CUDA Basics 1

- CUDA用語：Thread, Block, Grid, と Kernel
  - Grid, Block, Thread と階層構造となっている。
  - 一般にGrid は >1000 threads から構成される
  - すべての threads は同じコード（Kernel）を実行（SPMD）
  - Threads / blocks は固有の ID を持ち、これをプログラム中で利用







CUDA Basics 2: C and CUDA codes

```
void calc(int n, float a, const float *x, float *y){
    for (int i = 0; i < n; i++)
        y[i] = a * x[i];
}

int main() {
…
    calc(n, 2.0, x, y);
…
}
```

Standard C Code

```
__global__ void calc_gpu(int n, float a, const float *x, float *y){
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    y[i] = a * x[i];
}

int main() {
…
    int nblocks = n / 256;
    calc_gpu<<<nblocks, 256>>>calc_gpu(n, 2.0, x, y);
…
}
```

CUDA Code

12

CUDA Basics 3: Kernel Variations

```
__global__ void kernel(int *a){
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    a[i] = 5;
}
                                        Output: 5 5 5 5 5 5 5 5 5 5 5 5

__global__ void kernel(int *a){
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    a[i] = blockIdx.x;
}
                                        Output: 0 0 0 0 1 1 1 1 2 2 2 2

__global__ void kernel(int *a){
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    a[i] = threadIdx.x;
}
                                        Output: 0 1 2 3 0 1 2 3 0 1 2 3
```

```
int main() {
…
    int nblocks = n / 4;
    calc_gpu<<<nblocks, 4>>>kernel(a);
…
}
```

CUDA Code

13

CUDA Basics 4: 2D Index

```
__global__ void kernel(int nx, int ny, int *a){
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    int j = blockDim.y * blockIdx.y + threadIdx.y;
    int ix = j * nx + i;

    a[ix] = a[ix] + 1;
}

int main() {
…
    dim3 threads(64, 4);
    dim3 blocks(nx/threads.x, ny/threads.y);

    kernel<<<blocks, threads>>>kernel(nx, ny, a);
…
}
```

CUDA Code

14

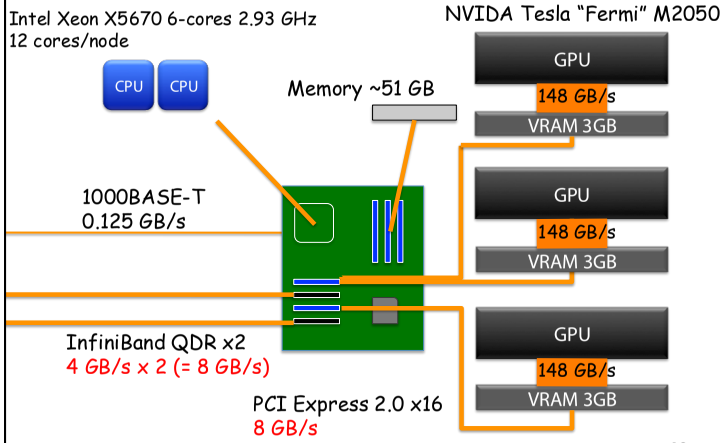
Tokyo Tech TSUBAME 2.0 Supercomputer

- GPUが演算性能の大部分を担う。2010年11月から稼働。

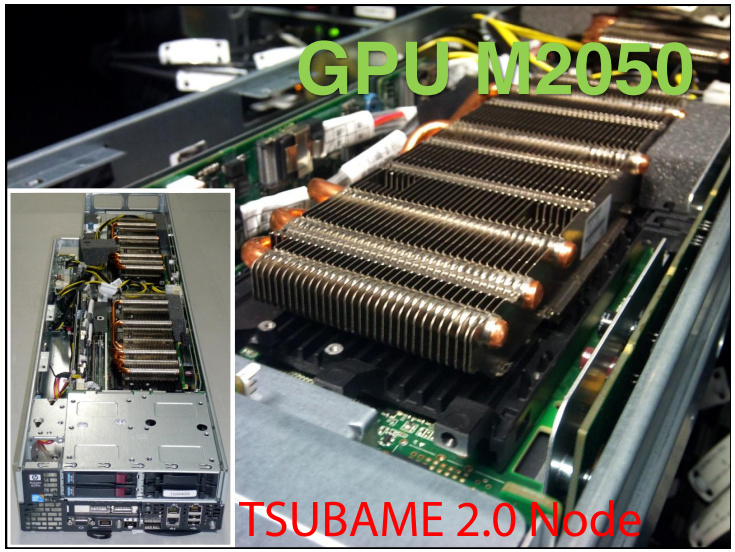
	TSUBMAE 1.2	TSUBAME 2.0 (2.4 PF)
CPU	AMD Opteron dual-core 2.4 GHz, 16 cores/node	Intel Xeon X5670 6-cores 2.93 GHz, 12 cores/node
GPU	NVIDIA Tesla S1070 691.2 TFlops (SP) 86.4 TFlops (DP) Memory BW 102 GB/s Memory 4 GByte 2 GPUs/node	NVIDIA Tesla "Fermi" M2050 1030 GFlops (SP) 515 GFlops (DP) Memory BW 148 GB/s Memory 3 GByte 3 GPUs/node
Total # of GPUs	680 GPUs	4224 GPUs
CPU-GPU	PCIe 1.0 x 8 (2 GB/s)	PCIe 2.0 x 16 (8 GB/s)
Inter-node Connection	Dual-Rail IB-SDR (2 GB/s)	Dual-Rail IB-QDR (8 GB/s)

15

TSUBAME 2.0 Node detail



16



TSUBAME 2.0 World Rankings (Nov. 2010)

The Top 500 (Absolute Performance)

- #1: **2.57** PetaFlops: China Defense Univ. Dawning Tianhe 1-A NVIDIA M2050 GPU
- #2: **1.76** Petaflops: US ORNL Cray XT5 Jaguar
- #3: **1.27** PetaFlops: China Shenzen SC Nebulae NVIDIA C2050 GPU
- #4: **1.19** PetaFlops: Japan Tokyo Tech. HP/NEC TSUBAME2.0 NVIDIA M2050 GPU
- #5: **1.05** PetaFlops: US LBNL Cray XE6 Hopper
- #33 (#2 Japan): **0.191** Petaflops:JAEA Fujitsu

The Green 500 (Performance/Power Efficiency)

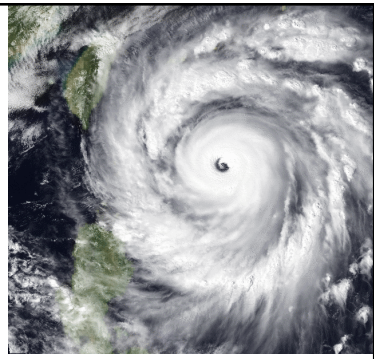
- #1: **1684.2** MFlops/W: US IBM Blue Gene/Q Prototype
- #2: **958.4** MFlops/W: Japan Tokyo Tech. HP/NEC TSUBAME2.0
- #3: **933.1** MFlops/W: US NCSA Hybrid Cluster Core i3 2.93Ghz Dual Core, NVIDIA C2050, Infiniband
- #4: **828.7** MFlops/W: Japan RIKEN K computer

18

詳細はE-Science Journal (ESJ)を御覧ください

- ✓ 東工大 学術国際情報センター(GSIC)で作成したパンフレット
  - ✓ TSUBAME 2.0 のシステムの詳細
  - ✓ TSUBAME で実行されたアプリケーション（ASUCA含む）
- ✓ http://www.gsic.titech.ac.jp/node/205 からダウンロードできます。東工大GSICに印刷された冊子もあります。

19

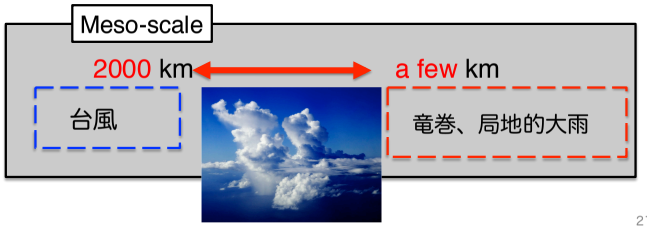


Weather Model ASUCA

20

気象シミュレーションコードASUCA

- ASUCA Production Code
  - ✓ 気象庁によって開発が進められる次世代高解像度気象シミュレーションコード
- なぜ GPU コンピューティング？
  - ✓ 速い, 安い, 低消費電力



21

http://wallpaper.free-photograph.net/

気象モデル

- ✓ 力学過程
  - ✓ 3次元の Navior-Stokes 方程式
    $$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla u = -\frac{1}{\rho}\nabla p - 2\Omega\times\boldsymbol{u} - \Omega\times(\Omega\times\boldsymbol{r}) - \boldsymbol{g} + \boldsymbol{F}$$
  - ✓ 計算格子全体にわたる時間発展の計算、アプリケーションの構造を決める
  - ✓ 一般にメモリアクセス律速な計算
- ✓ 物理過程
  - ✓ 雲物理プロセス、水蒸気や湿潤大気、太陽放射、凝縮、潜熱放出、大気境界層
  - ✓ 実験に基づいたパラメタリゼーションによる計算
  - ✓ 一般に計算律速（sin, cos, exp などを含む）

22

ASUCA : 支配方程式

- 完全圧縮・非静力学方程式系（格子計算）
  - ✓ フラックス形式
  - ✓ 一般座標系（起伏を表現）
- 運動方程式
$$\frac{\partial}{\partial t}\left(\frac{\rho u^i}{J}\right) + \frac{\partial}{\partial \hat{x}^j}\left(\frac{\rho u^i \hat{u}^j}{J}\right) + \frac{\partial}{\partial \hat{x}^n}\left(\frac{1}{J}\frac{\partial \hat{x}^n}{\partial x^i}p\right) - \frac{\rho g^i}{J} = \frac{F^i}{J}$$
- 温位の式（=温度の表現）
$$\frac{\partial}{\partial t}\left(\frac{\rho\theta_m}{J}\right) + \frac{\partial}{\partial \hat{x}^i}\left(\frac{\rho\theta_m \hat{u}^i}{J}\right) = \frac{F_{\rho\theta_m}}{J}$$
- 連続の式
$$\frac{\partial}{\partial t}\left(\frac{\rho}{J}\right) + \frac{\partial}{\partial \hat{x}^i}\left(\frac{\rho \hat{u}^i}{J}\right) = \frac{F_\rho}{J}$$
- 状態方程式
$$\rho = \frac{p_0}{R\theta_m}\left(\frac{p}{p_0}\right)^{C_v/C_p}$$

•Momentum :	ρu^i
•Potential temperature :	$\rho\theta$
•Mass-virtual potential temperature :	θ_m
•Density :	ρ
•Pressure :	p
•Constant reference value of pressure :	p_0
•Gravitational acceleration :	g
•Gas constant :	R
•Specific heat at constant pressure :	C_p
•Specific heat at constant volume :	C_v
•Force terms of the physical processes :	$F^i, F_{\rho\theta_m}, F_\rho$
•Jacobian :	J

ASUCA : 支配方程式

- 水物質の式
$$\frac{\partial}{\partial t}\left(\frac{\rho q_x}{J}\right) + \frac{\partial}{\partial \hat{x}^i}\left(\frac{\rho q_x \hat{u}^i_x}{J}\right) = \frac{F_{\rho q_x}}{J}$$
  - ✓ 7 つの水物質: 水蒸気、雲水、雨、雲氷、雪、あられ、ひょう
  - ✓ 現時点では、雲の微物理パラメタリゼーションとして Kessler-type warm-rain スキームを導入

気象計算におけるGPUによる高速化

- The Weather Research and Forecast (WRF) の GPU 利用
  = 計算律速の物理モジュールをGPUで部分的に加速
  - 雲微物理モジュール WSM5 (WRF Single Moment 5-tracer) *
    水物質(水蒸気、 雲水、雨水、雲氷、雪)の混合比を予報
    WRF の 1 % のコード、25%の実行時間
    ⇒ 20 x 高速化
    (アプリケーション全体で 1.2-1.3倍の高速化)
  - 領域化学輸送モデル WRF-Chem**
    特定の領域における汚染気体の輸送や化学反応
    ⇒ 8.5 x 高速化

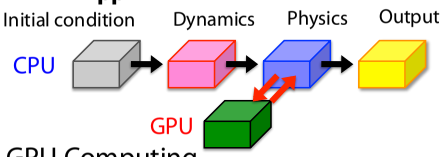
* Michalakes, J. and M. Vachharajani: GPU Acceleration of Numerical Weather Prediction. *Parallel Processing Letters Vol. 18 No. 4. World Scientific. Dec. 2008. pp. 531—548*
**John C. Linford, John Michalakes, Manish Vachharijani, and Adrian Sandu. Multi-core acceleration of chemical kinetics for simulation and prediction, *proceedings of the 2009 ACM/IEEE conference on supercomputing (SC'09), ACM, 2009.*

25

WRFとASUCAの高速化のアプローチの違い

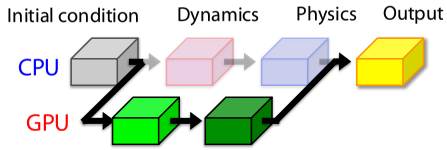
✓ WRF GPU Acceleration

Accelerator Approach



✓ ASUCA GPU Computing

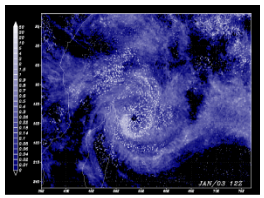
Full GPU Approach ➡ 数十倍の高速化の実現



26

これまでの大規模気象計算

- AFES (AGCM for Earth Simulator) @ 地球シミュレータ (2002) *
  - ✓ 海洋研究開発機構 (JAMSTEC)
  - ✓ 26.58 TFlops (SC02 Gordon Bell Prize)
  - ✓ 5,120 cores (640 nodes)



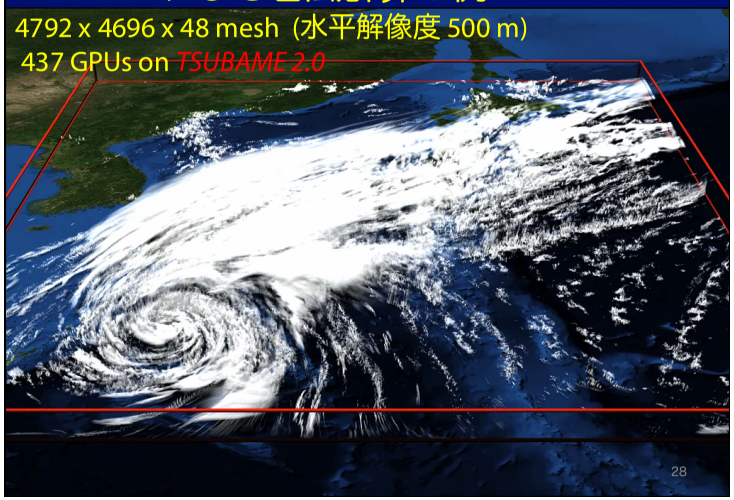
- WRF (Weather Research and Forecasting) モデル @ Jaguar (2009) **
  - ✓ US Oak Ridge 国立研究所 (ORNL)
  - ✓ 50 TFlops
  - ✓ 150,000 cores

* S. Shingu, H. Takahara, H. Fuchigami, M. Yamada, Y. Tsuda, W. Ohfuchi, Y. Sasaki, K. Kobayashi, T. Hagiwara, S. Habata, M. Yokokawa, H. Itoh, and K. Otsuka, "A 26.58 Tflops global atmospheric simulation with the spectral transform method on the Earth Simulator," *in Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing. Los Alamitos, CA, USA: IEEE Computer Society Press, 2002, pp. 1–19.*
** A. S. Bland, R. A. Kendall, D. B. Kothe, J. H. Rogers, and G. M. Shipman, "Jaguar: The world's most powerful computer," in *2009 CUG Meeting, 2009, pp. 1–7.*

27

ASUCA による台風計算の例

4792 x 4696 x 48 mesh (水平解像度 500 m)
437 GPUs on *TSUBAME 2.0*



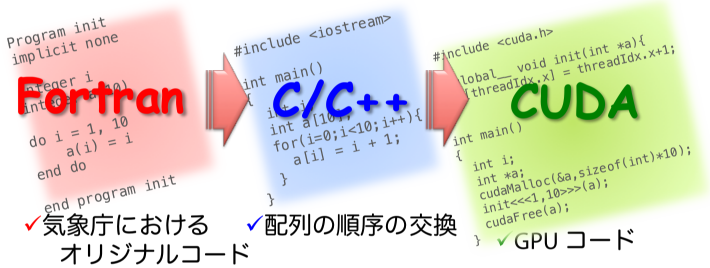
28

ASUCA on a single GPU

29

ASUCA: Fortran から CUDA へ

- フル GPU アプリケーション
- ゼロから書き換え



✓気象庁における オリジナルコード　✓配列の順序の交換　✓GPU コード

3 次元配列の要素順序

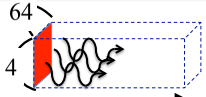
z,x,y (k,i,j)-ordering　*x,z,y* (i,k,j)-ordering　*x,z,y* (i,k,j)-ordering

GPUコードでのメモリアクセスパフォーマンスを向上

30

単一GPUへの実装の例

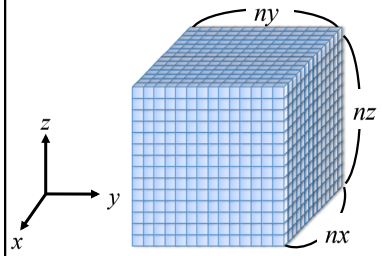
1 Thread　1 Block (64 x 4 threads)



スレッドのマーチング方向

CUDA の block を2次元に確保し、
残りの方向へ threads を移動しながら計算

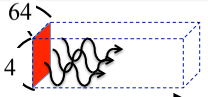
- 移流計算



31

単一GPUへの実装の例

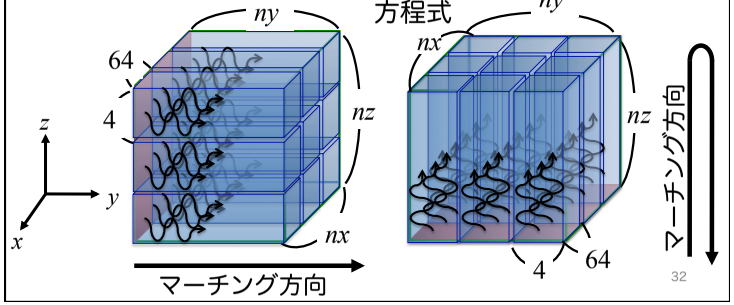
1 Thread　1 Block (64 x 4 threads)



スレッドのマーチング方向

CUDA の block を2次元に確保し、
残りの方向へ threads を移動しながら計算

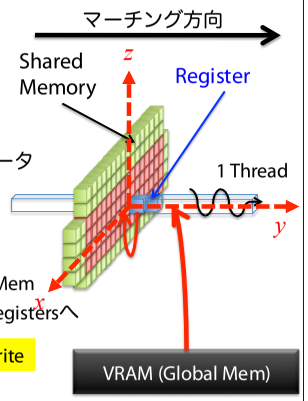
- 移流計算
- 気圧に対する1次元 Helmholtz 方程式



32

Shared memory とRegisters の利用

- Shared Memory (SMem)
  - ✓ Software Managed Cache
  - ✓ アクセス速度: 2 cycle
- Register
  - ✓ アクセス速度: 1 cycle
  - ✓ Threads 間で共有する必要のないデータに対して利用
- 移流計算: 各方向4ステンシル
  - ✓ xz平面のデータは (64 + 3) x (4 + 3) SMem
  - ✓ 各threadが使用するy方向データはregistersへ

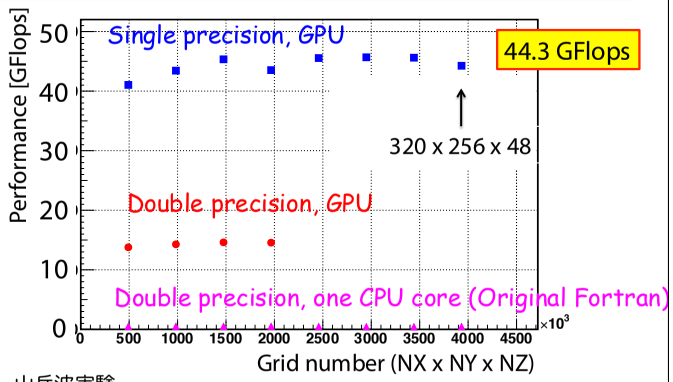


GMem へ直接アクセス　: 4 + 4 + 4 read, 1write

SMem と Registersの利用 : ~1 read, 1write

33

単一GPUによるASUCA の計算性能 (Tesla S1070)



- 山岳波実験
- 1GPU from NVIDIA Tesla S1070 card (Memory 4 GByte)
- CPU: AMD Opteron dual-core 2.4 GHz

34

Performance とArithmetic Intensity

$$Performance = \frac{FLOP}{\underbrace{FLOP/F_{peak}}_{\text{FP operations time}} + \underbrace{Byte/B_{peak}}_{\text{Memory access time}} + \alpha}$$

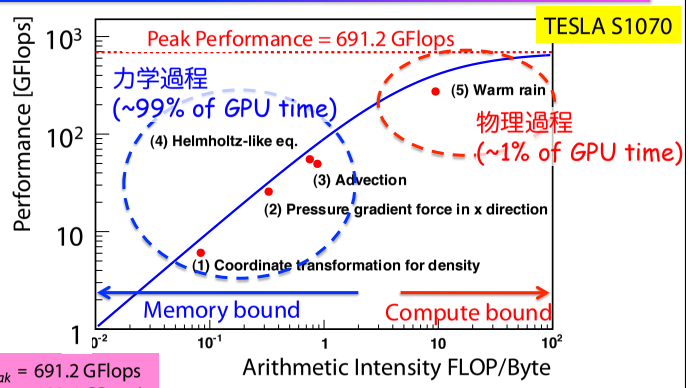
$$= \frac{FLOP/Byte}{FLOP/Byte + F_{peak}/B_{peak} + \alpha F_{peak}/Byte} F_{peak}$$

Arithmetic Intensity

- ✓ *FLOP* = アプリケーション中のFPオペレーションの数
- ✓ *Byte* = アプリケーション中のメモリアクセスのByte [Byte]
- ✓ F_{peak} = Floating- point オペレーションのピーク性能 [Flops]
- ✓ B_{peak} = ピークメモリバンド幅 [Byte/s]
- ✓ α = その他のオペレーションにかかる時間

35

ASUCAの5 kernelsの計算性能



F_{peak} = 691.2 GFlops
B_{peak} = 102.4 GByte/s

$$Performance = \frac{FLOP/Byte}{FLOP/Byte + F_{peak}/B_{peak} + \alpha F_{peak}/Byte} F_{peak}$$

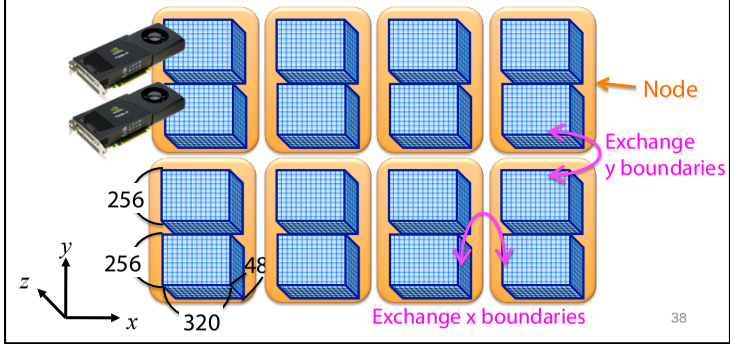
36

ASUCA on Multi-GPU

37

マルチGPU計算：領域分割

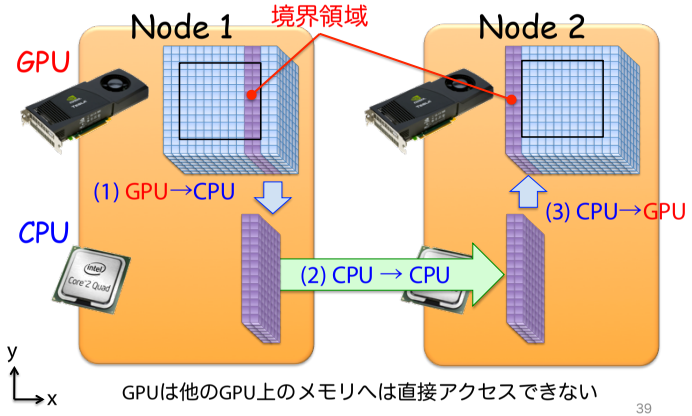
- 二次元領域分割 （x, y 方向）
  - ✓ 水平方向の格子数 ~ 1000 > 鉛直方向の格子数 ~ 100
  - ✓ 320 x 256 x 48 Sub-domain / GPU
  - ✓ 2 GPUs from NVIDIA Tesla S1070 / node on TSUBAME 1.2



38

マルチGPU計算：境界領域のデータ交換

- MPIを用いたGPUとCPUによるデータ交換



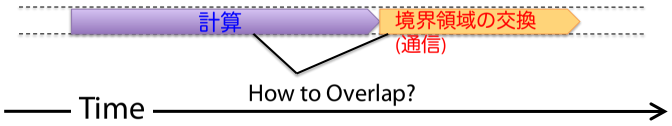
GPUは他のGPU上のメモリへは直接アクセスできない

39

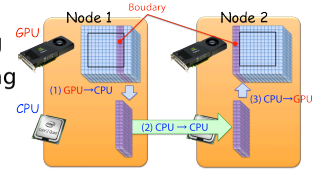
マルチGPU計算における最適化手法

- 計算と通信のオーバラップ

ある変数に対する計算の典型例



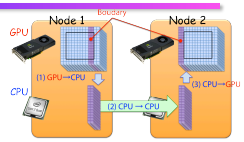
- 最適化手法
  - ✓ (1) Inter-variable Overlapping
  - ✓ (2) Kernel division Overlapping
  - ✓ (3) Fused Kernel Overlapping

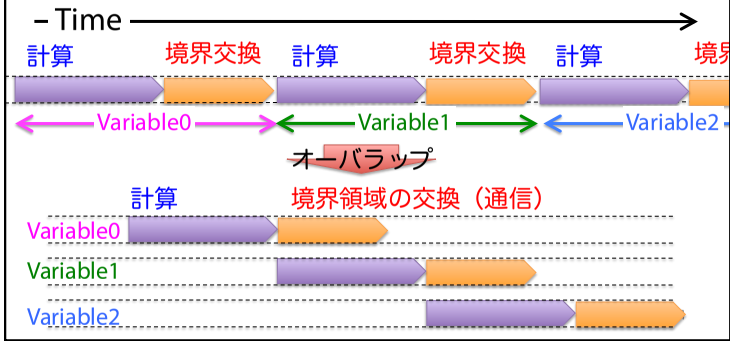


40

最適化手法１: Inter-variable Overlapping

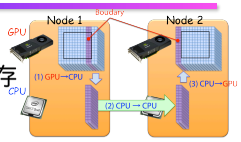
- 最適化手法１
  - ✓変数間の非依存関係の利用
  - ✓水物質の移流計算へ適用

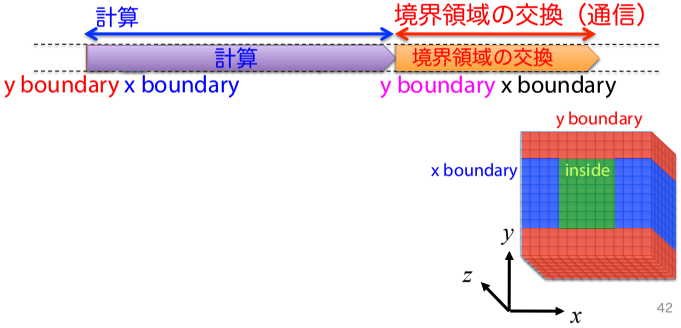




最適化手法２: Kernel division Overlapping

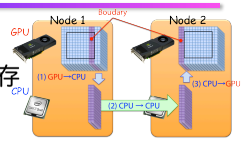
- 最適化手法２
  - ✓ひとつの変数内でのデータの非依存関係を利用
  - ✓1 kernel を 3 kernels へ分割

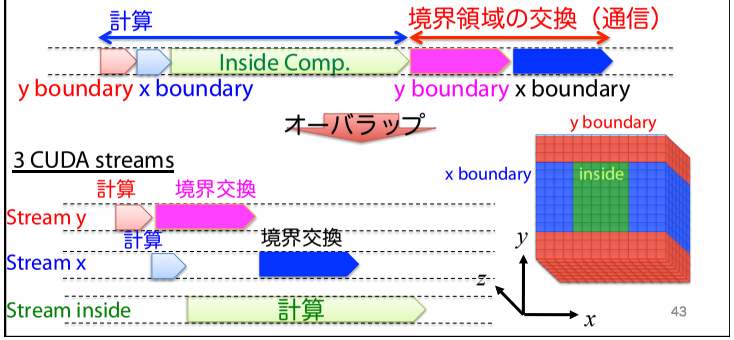




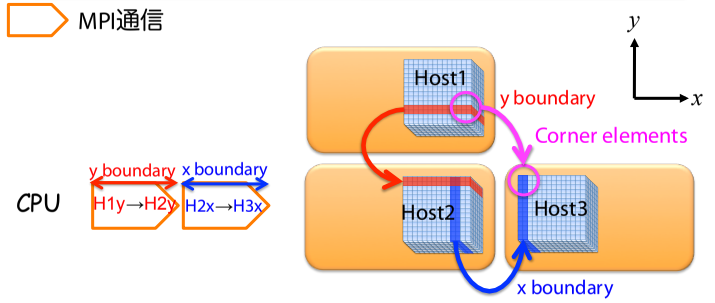
最適化手法２: Kernel division Overlapping

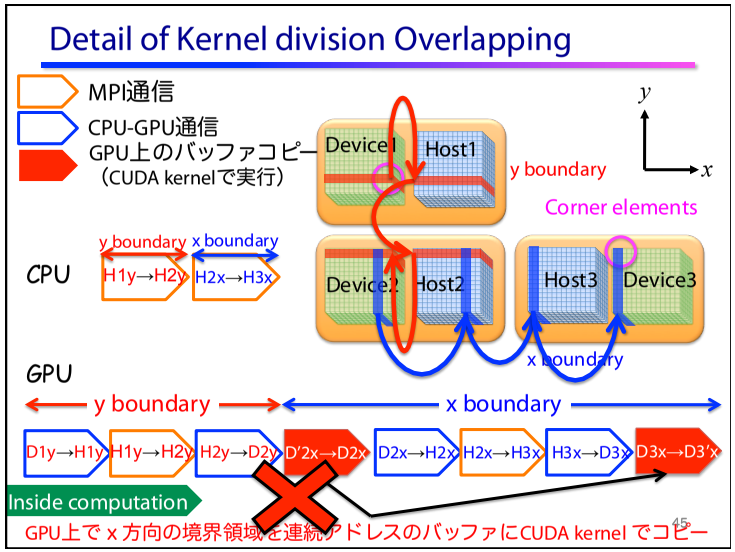
- 最適化手法２
  - ✓ひとつの変数内でのデータの非依存関係を利用
  - ✓1 kernel を 3 kernels へ分割

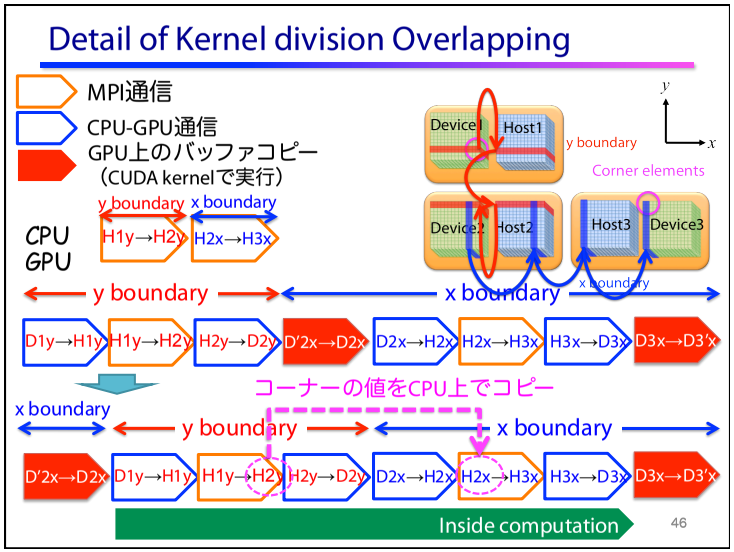


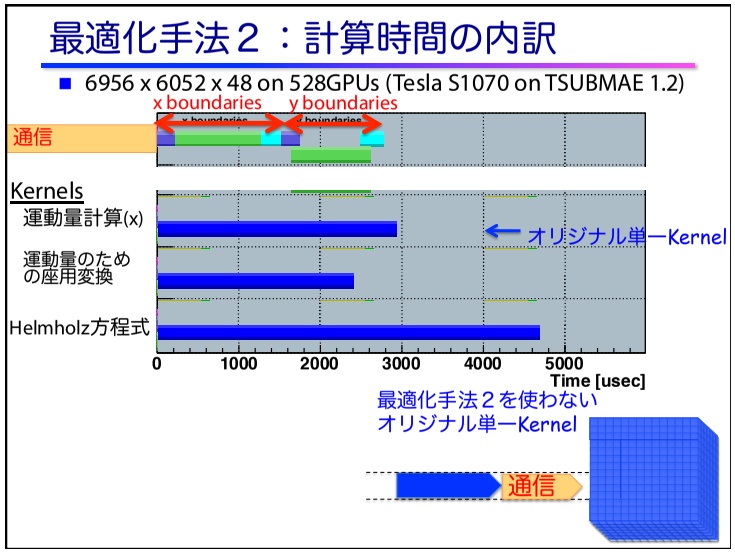


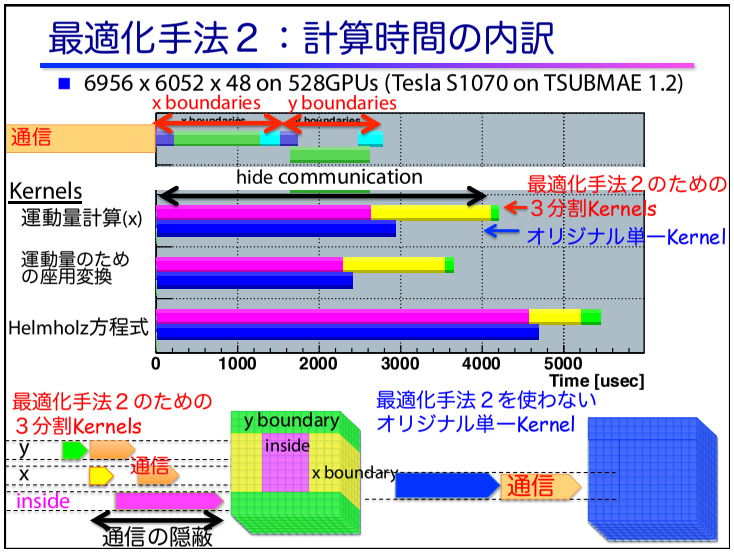
Detail of Kernel division Overlapping



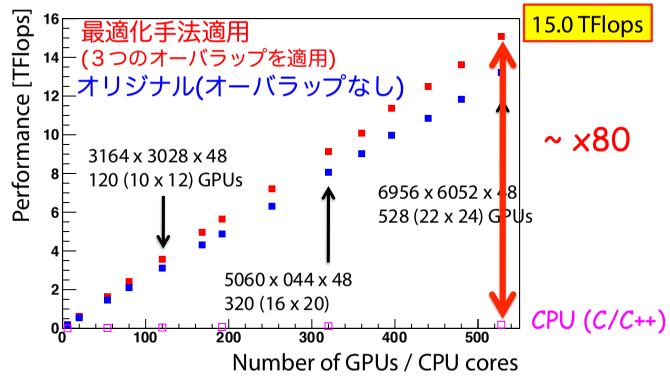

Detail of Kernel division Overlapping
MPI通信
CPU-GPU通信
GPU上のバッファコピー
（CUDA kernelで実行）
Device1
Host1
y boundary
Corner elements
CPU
y boundary x boundary
H1y→H2y
H2x→H3x
Device2
Host2
Host3
Device3
x boundary
GPU
y boundary
x boundary
D1y→H1y
H1y→H2y
H2y→D2y
D'2x→D2x
D2x→H2x
H2x→H3x
H3x→D3x
D3x→D3'x
Inside computation
GPU上で x 方向の境界領域を連続アドレスのバッファにCUDA kernel でコピー
45


Detail of Kernel division Overlapping
MPI通信
CPU-GPU通信
GPU上のバッファコピー
（CUDA kernelで実行）
Device1
Host1
y boundary
Corner elements
y boundary x boundary
H1y→H2y
H2x→H3x
CPU
GPU
Device2
Host2
Host3
Device3
x boundary
y boundary
x boundary
D1y→H1y
H1y→H2y
H2y→D2y
D'2x→D2x
D2x→H2x
H2x→H3x
H3x→D3x
D3x→D3'x
コーナーの値をCPU上でコピー
x boundary
y boundary
x boundary
D'2x→D2x
D1y→H1y
H1y→H2y
H2y→D2y
D2x→H2x
H2x→H3x
H3x→D3x
D3x→D3'x
Inside computation
46


最適化手法２：計算時間の内訳
6956 x 6052 x 48 on 528GPUs (Tesla S1070 on TSUBMAE 1.2)
x boundaries
y boundaries
通信
Kernels
運動量計算(x)
運動量のための座用変換
Helmholz方程式
オリジナル単一Kernel
0
1000
2000
3000
4000
5000
Time [usec]
最適化手法２を使わない
オリジナル単一Kernel
通信


最適化手法２：計算時間の内訳
6956 x 6052 x 48 on 528GPUs (Tesla S1070 on TSUBMAE 1.2)
x boundaries
y boundaries
通信
hide communication
Kernels
運動量計算(x)
運動量のための座用変換
Helmholz方程式
最適化手法２のための
３分割Kernels
オリジナル単一Kernel
0
1000
2000
3000
4000
5000
Time [usec]
最適化手法２のための
３分割Kernels
y
x
inside
通信
y boundary
inside
x boundary
通信の隠蔽
最適化手法２を使わない
オリジナル単一Kernel
通信

ASUCA のマルチGPU計算での計算性能



- 山岳波実験、単精度計算
- NVIDIA Tesla S1070 cards and AMD Opteron dual-core 2.4 GHz on TSUBAME 1.2

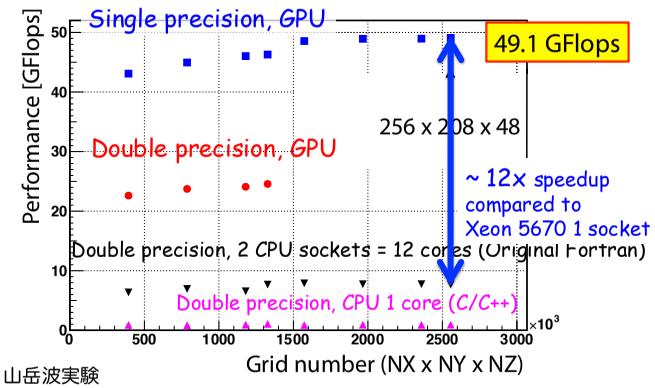
49

ASUCA on TSUBAME 2.0

50

TSUBAME 2.0 (1GPU)でのASUCA の計算性能

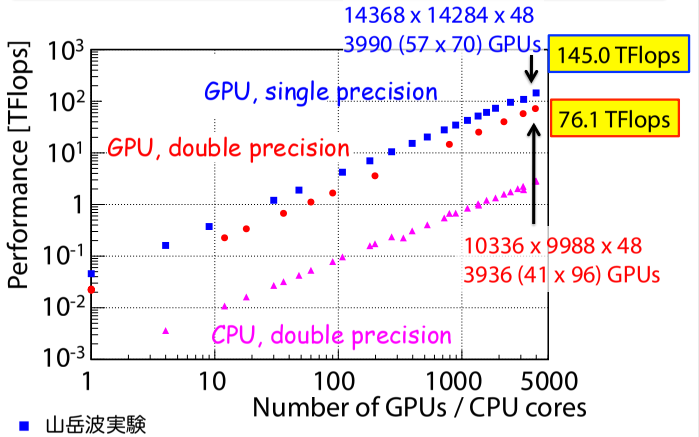
NVIDIA Tesla M2050 / Intel Xeon X5670 2.93 GHz



- 山岳波実験
- NVIDIA Tesla M2050 card / Intel Xeon X5670 2.93 GHz on TSUBAME 2.0

51

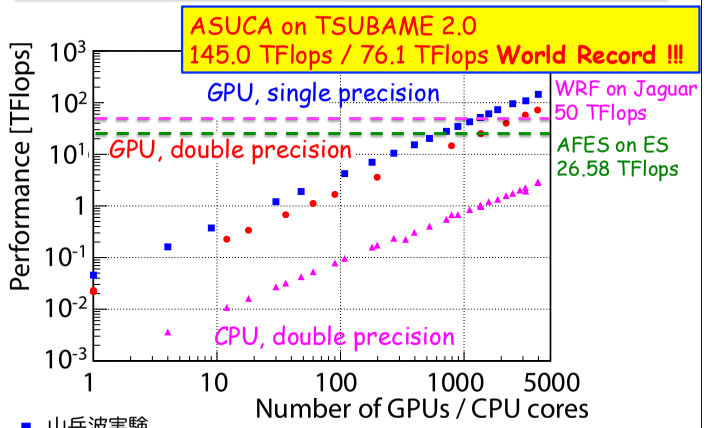
TSUBAME 2.0を用いたASUCAの計算性能



- 山岳波実験
- NVIDIA Tesla M2050 card / Intel Xeon X5670 2.93 GHz on TSUBAME 2.0

52

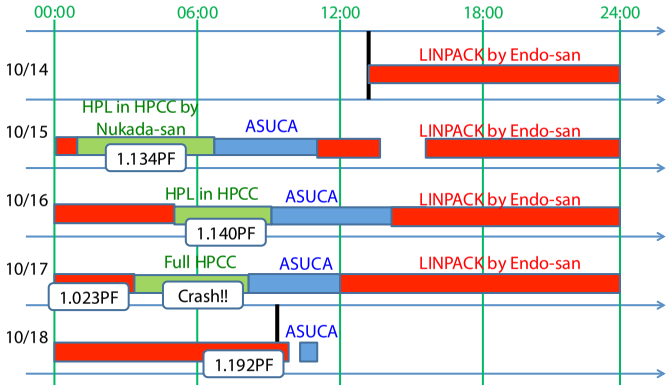
TSUBAME 2.0を用いたASUCAの計算性能



- 山岳波実験
- NVIDIA Tesla M2050 card / Intel Xeon X5670 2.93 GHz on TSUBAME 2.0

53

TSUBAME 2.0 ベンチマークの日々



We can use all nodes only for four days.

まとめ

気象庁の開発する次世代メソスケールモデル ASUCA のフルGPU計算について紹介した。

- ASUCA の単一GPU計算
  - ✓ 計算格子 256 x 208 x 48 に対して単精度計算で 49.1 GFlops を達成（TSUBAME 2.0 の Tesla "Fermi" M2050 を利用）
- ASUCAのマルチGPU計算
  - ✓ 最適化手法として通信と計算のオーバラップ手法の導入
- TSUBAME 2.0を用いたASUCA の計算性能
  - ✓ 計算格子 14368 x 14284 x 48 に対して単精度計算で3990GPUを利用して 145.0 TFlops を達成
  - ✓ 計算格子 10336 x 9988 x 48 に対して倍精度計算で3936GPUを利用して 76.1 TFlops を達成

［参考文献］

- ✓ Takashi Shimokawabe, Takayuki Aoki, Chiashi Muroi Junichi Ishida, Kohei Kawano, Toshio Endo, Akira Nukada, Naoya Maruyama, and Satoshi Matsuoka, **"An 80-Fold Speedup, 15.0 TFlops, Full GPU Acceleration of Non-Hydrostatic Weather Model ASUCA Production Code,"** *in Proceedings of the 2010 ACM/IEEE conference on Supercomputing (SC'10) , New Orleans, LA, USA, Nov 2010.*

55