GPUへのコンパクト差分の実装と圧縮性流体計算への適用

出川智啓（電気通信大学）

概要

- GPU上で圧縮性流体の直接数値計算を実行するためにコンパクト差分をGPU上に実装したい
- コンパクト差分によって生じる行列式の求解の高速化が必要
- 行列式の解法をいくつか選び
  - 1次元移流方程式
  - 1次元移流方程式＋不連続捕獲
  - 2次元移流方程式

  における適用性を調査

背景

次世代超音速旅客機の実現における課題

- 環境適合性
  - ソニックブーム低減，騒音低減，排ガス正常化
- 経済性
  - 軽量化，低抵抗化，エンジン低燃費
- 機体の低抵抗化
  - 旅客機に作用する全抵抗の約40%が摩擦抵抗
    - 境界層の乱流遷移により摩擦抵抗増大
  - 境界層を層流に保持する翼形状や乱流化を遅滞させる流れ制御による抵抗低減
    - 遷移機構の解明と遷移予測精度の向上
    - 高い解像度と安定性を備えた計算法

背景

- コンパクト差分近似
  - 関数ϕの微分値ϕ'を未知数として近似式を構成

$$\beta\phi'_{i-2}+\alpha\phi'_{i-1}+\phi'_i+\alpha\phi'_{i+1}+\beta\phi'_{i+2} = a\frac{\phi_{i+1}-\phi_{i-1}}{2\Delta x}+b\frac{\phi_{i+2}-\phi_{i-2}}{4\Delta x}+c\frac{\phi_{i+3}-\phi_{i-3}}{6\Delta x}$$

  - 一般的な“陽的”差分と比較して，
    - 同一ステンシル幅でより高い打ち切り精度を達成
    - 同一空間精度であっても，より高い解像度をもつ
    - 行列式を解く必要があるため，計算負荷が高い
  - 多次元空間でのコンパクト差分近似は同一の係数行列・異なる右辺ベクトルをもつ多数の行列式を解く

目的

- GPUへのコンパクト差分の実装 [1]
  - 単一の行列式はLU分解により逐次求解
  - 1 Blockが1行列式を解くようにし，複数の行列式を同時に解くことで高速化を狙う
  - 単一の行列式に対する高速な解法が必要
- 行列式の解法の適用性を検討する
  - 1次元移流方程式
  - 1次元移流方程式＋不連続捕獲
  - 2次元移流方程式

[1] 出川他2名, 第15回計算工学講演会, Vol.1, 129-132, 2010.

コンパクト差分[2]

- 関数ϕの微分値ϕ'を未知数として近似式を構成

$$\beta\phi'_{i-2}+\alpha\phi'_{i-1}+\phi'_i+\alpha\phi'_{i+1}+\beta\phi'_{i+2} = a\frac{\phi_{i+1}-\phi_{i-1}}{2\Delta x}+b\frac{\phi_{i+2}-\phi_{i-2}}{4\Delta x}+c\frac{\phi_{i+3}-\phi_{i-3}}{6\Delta x}$$

$$\alpha=\frac{1}{3},\ \beta=0,\ a=\frac{14}{9},\ b=\frac{1}{9},\ c=0 \quad \Rightarrow \quad \text{空間6次精度}$$

$$\begin{bmatrix} 1 & 3 & & & & & \\ 1/4 & 1 & 1/4 & & & & \\ & 1/3 & 1 & 1/3 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1/3 & 1 & 1/3 & \\ & & & & 1/4 & 1 & 1/4 \\ & & & & & 3 & 1 \end{bmatrix} \begin{Bmatrix} \phi'_1 \\ \phi'_2 \\ \phi'_3 \\ \vdots \\ \phi'_{Nx-2} \\ \phi'_{Nx-1} \\ \phi'_{Nx} \end{Bmatrix} = \begin{Bmatrix} f(\phi_1,\phi_2,\phi_3,\phi_4) \\ f(\phi_2,\phi_4) \\ f(\phi_{i-2},\phi_{i-1},\phi_{i+1},\phi_{i+2}) \\ \vdots \\ f(\phi_{i-2},\phi_{i-1},\phi_{i+1},\phi_{i+2}) \\ f(\phi_{Nx-2},\phi_{Nx}) \\ f(\phi_{Nx-3},\phi_{Nx-2},\phi_{Nx-1},\phi_{Nx}) \end{Bmatrix}$$

片側コンパクト
空間4次精度
空間4次精度
片側コンパクト

$$[A]\{\phi'\}=\{f(\phi)\}$$

[2] Lele, S.K., J. Comput. Phys., Vol.103, 16-42, 1992.

行列式の解法

- コンパクト差分で作られる行列式
  - 係数行列が小規模（多くの場合数百×数百，3重対角行列）
  - 離散化精度を決定する係数であり，時間・空間的に変化しない
  - 行列式の解法としてLU分解が有効に利用されている
- GPU上での行列式の解法
  - 3重対角行列のLU分解
  - Cyclic Reduction法
  - 自乗共役勾配法（CGS法）
  - 安定化双共役勾配法（Bi-CGSTAB法）

3重対角行列のLU分解

- コンパクト差分の計算における主解法
  - 計算量は$O(2n)$
  - 計算開始時に係数行列を作り，CPUによってLU分解
  - LU分解した係数行列をConstant memoryへ転送し，全Threadから参照

$$
\begin{cases}
\phi_i = \phi_i - L_i \phi_{i-1}, & i = 2,3,\ldots N_x \\
\phi_{N_x} = \phi_{N_x} / D_{N_x} & \\
\phi_i = \left(\phi_i - U_i \phi_{i+1}\right) / D_i, & i = N_x - 1, N_x - 2, \ldots 1
\end{cases}
$$

$$
A = LDU = \begin{pmatrix} * & 1 & 0 & & \\ l_2 & 1 & & & \\ & l_3 & 1 & & \\ & & l_4 & 1 & \\ & & & l_5 & 1 \\ & & & l_6 & 1 \\ 0 & & & l_7 & 1 \\ & & & & l_8 \end{pmatrix}
\begin{pmatrix} d_1 & & & & \\ & d_2 & & 0 & \\ & & d_3 & & \\ & & & d_4 & \\ & & & d_5 & \\ & & & d_6 & \\ & 0 & & & d_7 \\ & & & & d_8 \end{pmatrix}
\begin{pmatrix} 1 & u_1 & & & \\ & 1 & u_2 & 0 & \\ & & 1 & u_3 & \\ & & & 1 & u_4 \\ & & & 1 & u_5 \\ & & & 1 & u_6 \\ 0 & & & 1 & u_7 \\ & & & & 1 \end{pmatrix}
$$

Cyclic Reduction法

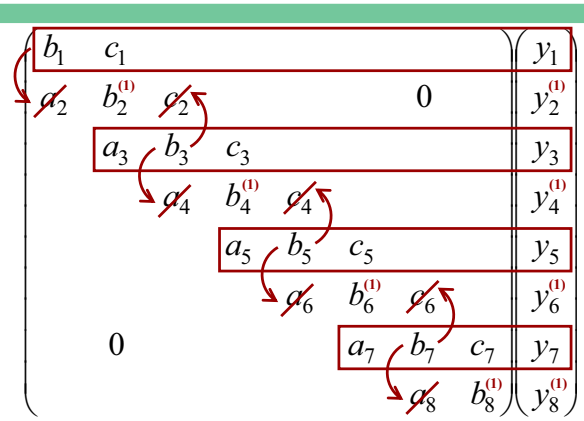
- ガウスの消去法の一種
  - 前進消去を分散的に行うことで並列性を確保
  - 帯幅は増加するが記憶領域は増加しない
  - 各処理を並列に行えれば，計算量は$O(2\log_2 N_x)$
  - 行列の係数，右辺ベクトルを全てShared memoryへ格納
  - GPUへの実装[3][4]が報告されている
  - GPUチャレンジ2010において，スプライン曲線の係数を求めるために使用した報告[5]もある

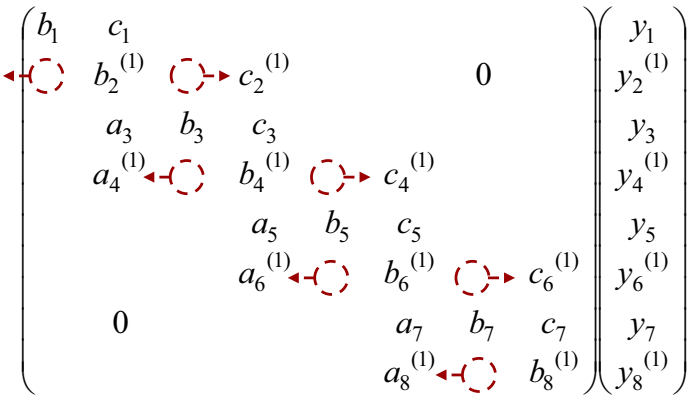
[3] Zhang, Y. *et al.*, Proc. 15th ACM SIGPLAN Sympo. PPoPP 2010, pp. 127-136, 2010.

[4] Göddeke, D. and Strzodka, R., preprint of IEEE TPDS.

[5] Waskito, P.他3名, SACSIS2010 論文集, pp.139-140, 2010.

Cyclic Reduction(1/[2log$_2$(8)])





Cyclic Reduction(2/[2log$_2$(8)])

$$\begin{pmatrix}
b_1 & c_1 & & & & & & \\
 & b_2^{(1)} & c_2^{(1)} & & & 0 & & \\
 & a_3 & b_3 & c_3 & & & & \\
 & a_4^{(1)} & & b_4^{(2)} & & c_4^{(1)} & & \\
 & & a_5 & b_5 & c_5 & & & \\
 & & a_6^{(1)} & & b_6^{(1)} & & c_6^{(1)} & \\
 & 0 & & & a_7 & b_7 & c_7 & \\
 & & & & a_8^{(1)} & & b_8^{(2)} &
\end{pmatrix}
\begin{pmatrix} y_1 \\ y_2^{(1)} \\ y_3 \\ y_4^{(2)} \\ y_5 \\ y_6^{(1)} \\ y_7 \\ y_8^{(2)} \end{pmatrix}$$

$$\begin{pmatrix}
b_1 & c_1 & & & & & & \\
 & b_2^{(1)} & c_2^{(1)} & & & 0 & & \\
 & a_3 & b_3 & c_3 & & & & \\
 & & & b_4^{(2)} & & & c_4^{(2)} & \\
 & & a_5 & b_5 & c_5 & & & \\
 & & a_6^{(1)} & & b_6^{(1)} & & c_6^{(1)} & \\
 & 0 & & & a_7 & b_7 & c_7 & \\
 & & & a_8^{(2)} & & & b_8^{(2)} &
\end{pmatrix}
\begin{pmatrix} y_1 \\ y_2^{(1)} \\ y_3 \\ y_4^{(2)} \\ y_5 \\ y_6^{(1)} \\ y_7 \\ y_8^{(2)} \end{pmatrix}$$

Cyclic Reduction(3/[2log$_2$(8)])

$$\begin{pmatrix}
b_1 & c_1 & & & & & & \\
 & b_2^{(1)} & c_2^{(1)} & & & 0 & & \\
 & a_3 & b_3 & c_3 & & & & \\
 & & & b_4^{(2)} & & & c_4^{(2)} & \\
 & & a_5 & b_5 & c_5 & & & \\
 & & a_6^{(1)} & & b_6^{(1)} & & c_6^{(1)} & \\
 & 0 & & & a_7 & b_7 & c_7 & \\
 & & & a_8^{(2)} & & & b_8^{(3)} &
\end{pmatrix}
\begin{pmatrix} y_1 \\ y_2^{(1)} \\ y_3 \\ y_4^{(2)} \\ y_5 \\ y_6^{(1)} \\ y_7 \\ y_8^{(3)} \end{pmatrix}$$

$$\begin{pmatrix}
b_1 & c_1 & & & & & & \\
 & b_2^{(1)} & c_2^{(1)} & & & 0 & & \\
 & a_3 & b_3 & c_3 & & & & \\
 & & & b_4^{(2)} & & & c_4^{(2)} & \\
 & & a_5 & b_5 & c_5 & & & \\
 & & a_6^{(1)} & & b_6^{(1)} & & c_6^{(1)} & \\
 & 0 & & & a_7 & b_7 & c_7 & \\
 & & & & & & b_8^{(3)} &
\end{pmatrix}
\begin{pmatrix} y_1 \\ y_2^{(1)} \\ y_3 \\ y_4^{(2)} \\ y_5 \\ y_6^{(1)} \\ y_7 \\ y_8^{(3)} \end{pmatrix}$$

Cyclic Reduction(4/[2log$_2$(8)])

$$\begin{pmatrix} b_1 & c_1 & & & & & & \\ & b_2^{(1)} & & c_2^{(1)} & & & 0 & \\ & a_3 & b_3 & c_3 & & & & \\ & & & b_4^{(2)} & & & c_4^{(2)} & \\ & & & a_5 & b_5 & c_5 & & \\ & & & a_6^{(1)} & & b_6^{(1)} & c_6^{(1)} & \\ & 0 & & & & a_7 & b_7 & c_7 \\ & & & & & & 1 & \end{pmatrix} \begin{pmatrix} y_1 \\ y_2^{(1)} \\ y_3 \\ y_4^{(2)} \\ y_5 \\ y_6^{(1)} \\ y_7 \\ \phi_8 \end{pmatrix}$$

$$\begin{pmatrix} b_1 & c_1 & & & & & & \\ & b_2^{(1)} & & c_2^{(1)} & & & 0 & \\ & a_3 & b_3 & c_3 & & & & \\ & & & 1 & & & & \\ & & & a_5 & b_5 & c_5 & & \\ & & & a_6^{(1)} & & b_6^{(1)} & c_6^{(1)} & \\ & 0 & & & & a_7 & b_7 & c_7 \\ & & & & & & 1 & \end{pmatrix} \begin{pmatrix} y_1 \\ y_2^{(1)} \\ y_3 \\ \phi_4 \\ y_5 \\ y_6^{(1)} \\ y_7 \\ \phi_8 \end{pmatrix}$$

Cyclic Reduction(5/[2log$_2$(8)])

$$\begin{pmatrix} b_1 & c_1 & & & & & & \\ & b_2^{(1)} & & c_2^{(1)} & & & 0 & \\ & a_3 & b_3 & c_3 & & & & \\ & & & 1 & & & & \\ & & & a_5 & b_5 & c_5 & & \\ & & & a_6^{(1)} & & b_6^{(1)} & c_6^{(1)} & \\ & 0 & & & & a_7 & b_7 & c_7 \\ & & & & & & 1 & \end{pmatrix} \begin{pmatrix} y_1 \\ y_2^{(1)} \\ y_3 \\ \phi_4 \\ y_5 \\ y_6^{(1)} \\ y_7 \\ \phi_8 \end{pmatrix}$$

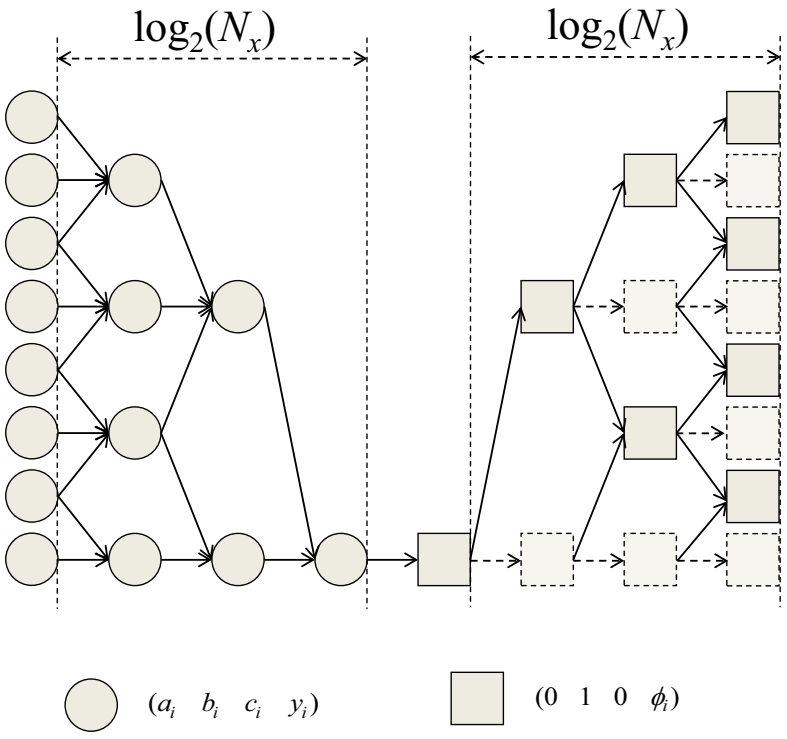
$$\begin{pmatrix} b_1 & c_1 & & & & & & \\ & 1 & & & & 0 & & \\ & a_3 & b_3 & c_3 & & & & \\ & & & 1 & & & & \\ & & & a_5 & b_5 & c_5 & & \\ & & & & & 1 & & \\ & 0 & & & & a_7 & b_7 & c_7 \\ & & & & & & & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ \phi_2 \\ y_3 \\ \phi_4 \\ y_5 \\ \phi_6 \\ y_7 \\ \phi_8 \end{pmatrix}$$

Cyclic Reduction(6/[2log$_2$(8)])

$$\begin{pmatrix} b_1 & c_1 & & & & & & \\ 1 & & & & & & 0 & \\ a_3 & b_3 & c_3 & & & & & \\ & & 1 & & & & & \\ & & a_5 & b_5 & c_5 & & & \\ & & & & 1 & & & \\ 0 & & & & a_7 & b_7 & c_7 & \\ & & & & & & 1 & \end{pmatrix} \begin{pmatrix} y_1 \\ \phi_2 \\ y_3 \\ \phi_4 \\ y_5 \\ \phi_6 \\ y_7 \\ \phi_8 \end{pmatrix}$$

$$\begin{pmatrix} 1 & & & & & & & \\ & 1 & & & & & 0 & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & 1 & & \\ & 0 & & & & & 1 & \\ & & & & & & & 1 \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi_5 \\ \phi_6 \\ \phi_7 \\ \phi_8 \end{pmatrix}$$

データの依存性



CGS法，Bi-CGSTAB法

- 非対称行列でも適用可能なCG法系統の解法を選択
  - 行列の係数はConstant memoryへ転送，右辺ベクトルおよび補助ベクトルを全てShared memoryへ格納
  - 行列ベクトル積はUnrolling

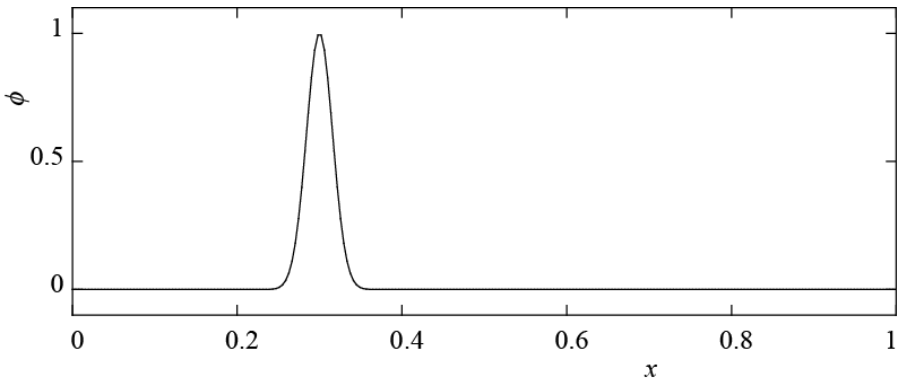
CGS法

$r_0=b-[A]x_0$ (x_0 :initial guess)
$r^*=r_0$
$\beta_{-1}=0$
for n=0,1,・・・・until $|r_n|<\varepsilon|b|$
begin
  $p_n=r_n+\beta_{n-1}z_{n-1}$
  $u_n=p_n+\beta_{n-1}(z_{n-1}-\beta_{n-1}u_{n-1})$
  $\alpha_n=(r^*\bullet r_n)/(r^*\bullet[A]u_n)$
  $z_n=p_n-\alpha_n[A]u_n$
  $x_{n+1}=x_n+\alpha_n(p_n+z_n)$
  $r_{n+1}=r_n-\alpha_n[A](p_n+z_n)$
  $\beta_n=(r^*\bullet r_{n+1})/(r^*\bullet r_n)$
end

Bi-CGSTAB法

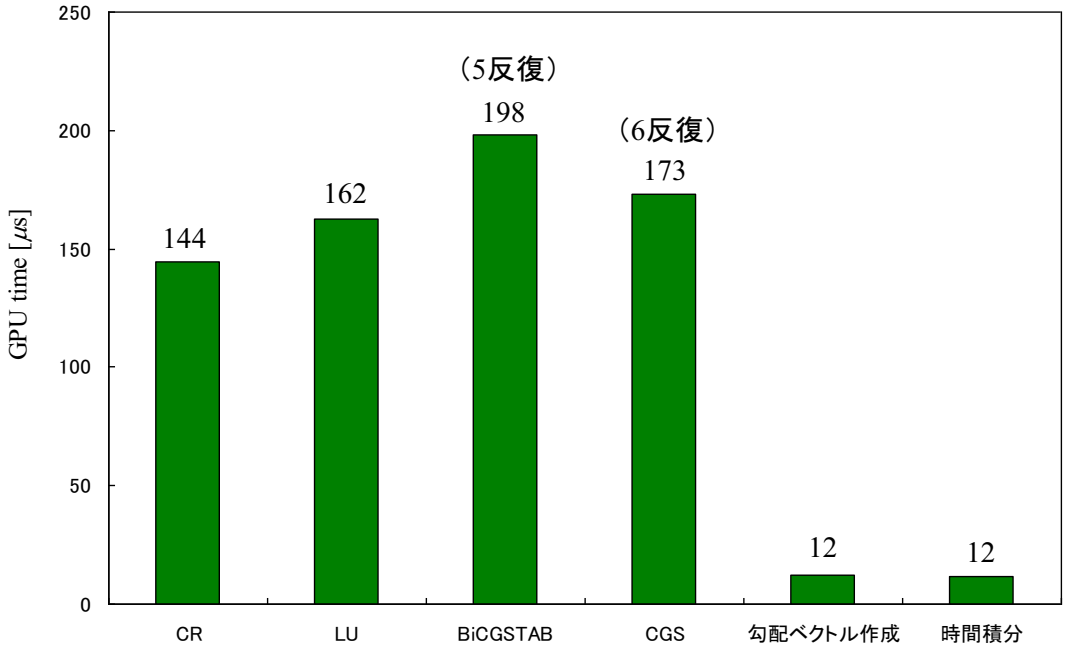
$r_0=b-[A]x_0$ (x_0 :initial guess)
$r^*=r_0$
$\beta_{-1}=0$
for n=0,1,・・・・until $|r_n|<\varepsilon|b|$
begin
  $p_n=r_n+\beta_{n-1}(p_{n-1}-\zeta_{n-1}[A]p_{n-1})$
  $\alpha_n=(r^*\bullet r_n)/(r^*\bullet[A]p_n)$
  $t_n=r_n-\alpha_n[A]p_n$
  $\zeta_n=([A]t_n\bullet t_n)/([A]t_n\bullet[A]t_n)$
  $x_{n+1}=x_n+\alpha_n p_n+\zeta_n t_n$
  $r_{n+1}=t_n-\zeta_n[A]t_n$
  $\beta_n=\alpha_n/\zeta_n\cdot(r^*\bullet r_{n+1})/(r^*\bullet r_n)$
end

1次元移流方程式



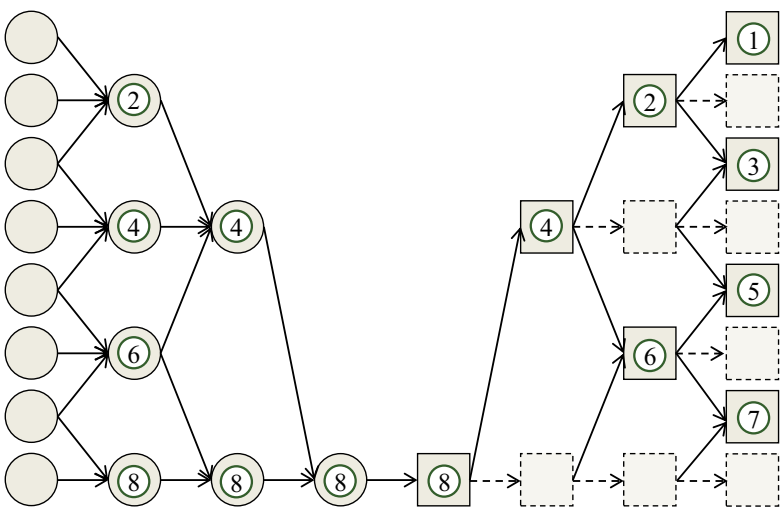
- 計算条件
  - $\frac{\partial\phi}{\partial t}+c\frac{\partial\phi}{\partial x}=0,\quad \phi=\left[\frac{1-\cos(2\pi x/L_x)}{2}\right]^{200},\ c=1$
  - 格子分割数256
  - Δt=0.001としてt=0.4まで
  - 空間離散化は6次精度コンパクト差分
  - 時間積分は4次精度Runge-Kutta法
  - 全て倍精度で計算
- 計算環境
  - CPU：Core i7 920
  - OS：Windows HPC Server 2008 (64bit)
  - GPU：NVIDIA Tesla C1060
  - CUDA 2.3 (for Windows Vista 64bit)
  - PGI アクセラレータコンパイラ 10.9 (for Windows 64bit)

各解法の所要時間



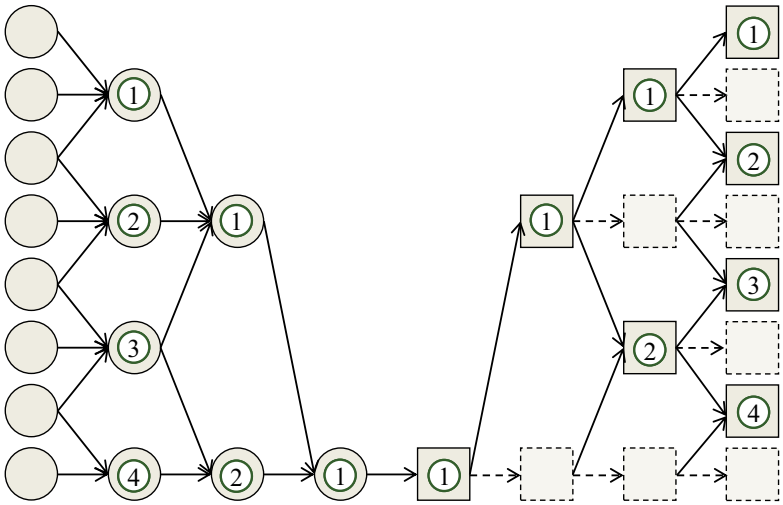
Cyclic Reduction法の改善

- データアクセスのパターンが総和計算に類似
  - 総和計算の改善方法を準用
    - スレッド番号の変更
    - 2の冪乗計算をビットシフト演算に(ishft)置き換え

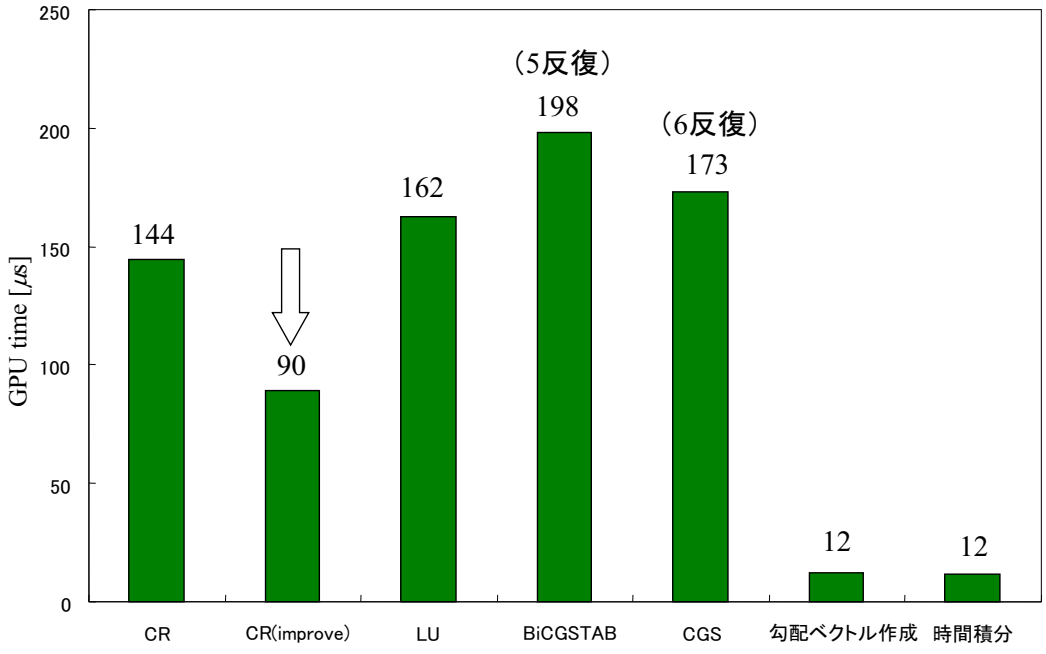


Cyclic Reduction法の改善

- データアクセスのパターンが総和計算に類似
  - 総和計算の改善方法を準用
    - スレッド番号の変更
    - 2の冪乗計算をビットシフト演算に(ishft)置き換え



各解法の所要時間



不連続捕獲スキームへの適用

- 散逸コンパクト差分[6]
  - 移流項をLax-Friedrish流束分割により分割
  - 特性速度の上流側に重みを付けたコンパクト差分

$$\frac{\partial \phi}{\partial t}+\frac{\partial F(=u\phi)}{\partial x}=0 \qquad F=F^{+}+F^{-}=\frac{1}{2}(u\phi+\lambda\phi)+\frac{1}{2}(u\phi-\lambda\phi) \qquad \lambda=u_{\max}$$

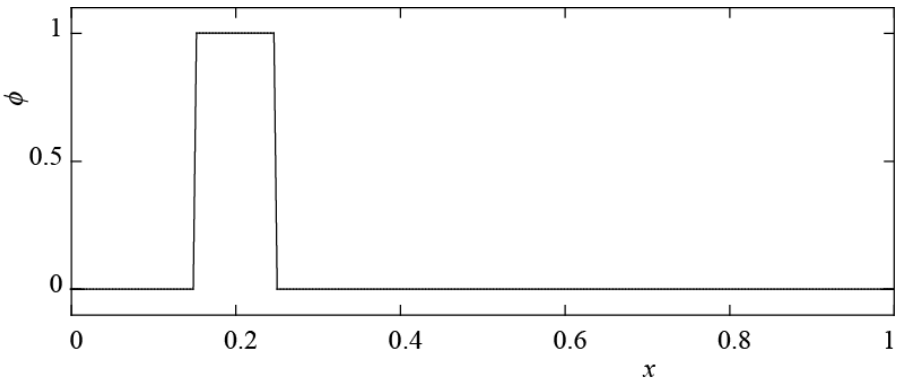
$$\frac{\partial F}{\partial x}=\frac{\partial F^{+}}{\partial x}+\frac{\partial F^{-}}{\partial x}$$

$$\alpha(1-\delta)F'_{i-1}+F'_{i}+\alpha(1+\delta)F'_{i+1} = a\frac{F_{i+1}-F_{i-1}}{2\Delta x}+b\frac{F_{i+2}-F_{i-2}}{4\Delta x}+\delta d\Delta x\frac{F_{i+1}-2F_{i}+F_{i-1}}{\Delta x^{2}}+\delta e2\Delta x\frac{F_{i+2}-2F_{i}+F_{i-2}}{4\Delta x^{2}}$$

$$\alpha=\frac{1}{3},\quad a=\frac{14}{9},\quad b=\frac{1}{9},\quad \mathrm{d}=4,\quad e=1 \qquad \delta=\pm 0.25 \quad \Rightarrow \quad \text{空間5次精度}$$

[6] Deng, X., *et al*., AIAA paper 96-1972, 1996.

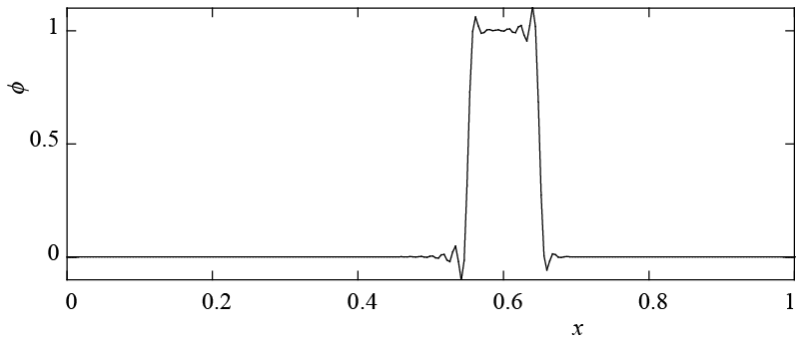
1次元移流方程式



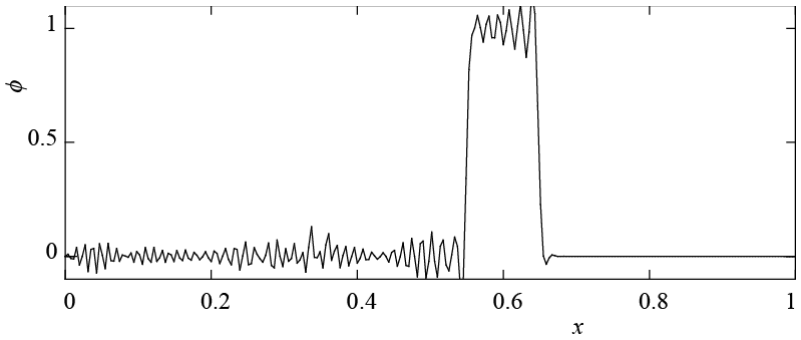
- 計算条件
  - $\frac{\partial \phi}{\partial t}+c\frac{\partial \phi}{\partial x}=0, \quad \begin{cases}\phi=1, & 0.15\le x\le 0.25\\ \phi=0, & \text{otherwise}\end{cases}, c=1$
  - 格子分割数256
  - Δt=0.001としてt=0.4まで
  - 空間離散化は5次精度散逸コンパクト差分
  - 時間積分は4次精度Runge-Kutta法
  - 全て倍精度で計算

計算結果（t=0.4）

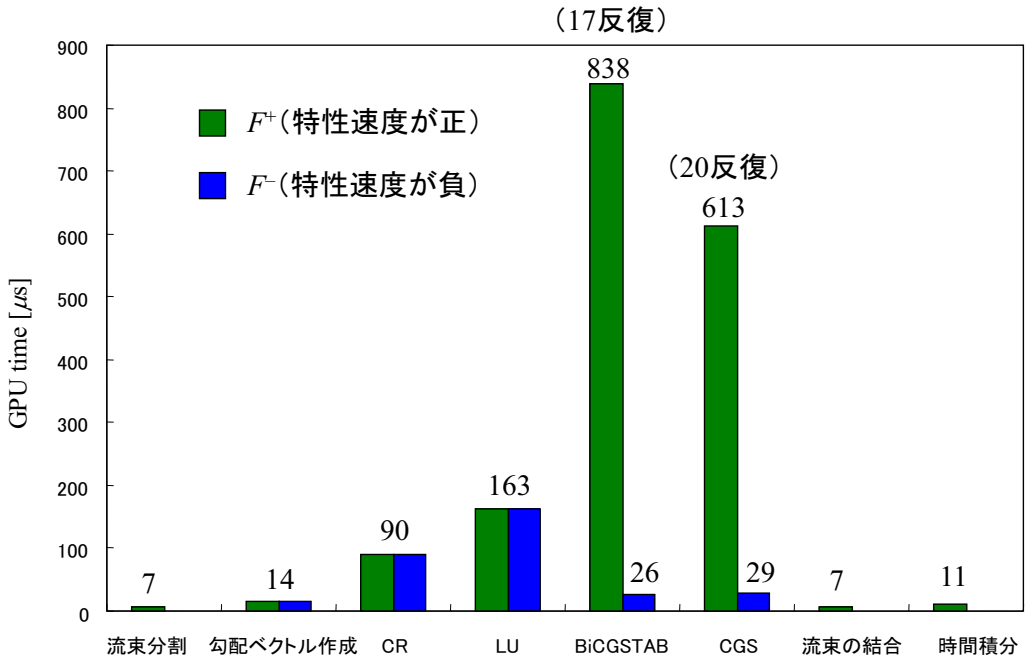
不連続捕獲有り



不連続捕獲無し



各解法の所要時間



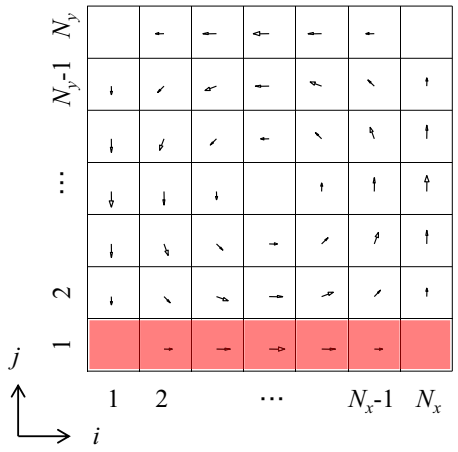
1次元コンパクト差分のまとめ

- GPU上でのコンパクト差分の求解において，Cyclic Reduction法は有用である
- CG法系統の解法を適用するには，内積がボトルネックとなりうる
- 不連続捕獲（散逸コンパクト差分）を用いた場合，CG法系統の解法は反復回数が著しく増加する
- LU分解以外の解法はShared memoryを大量に使用するため，256×256以上の行列式への適用は課題

2次元空間の1階コンパクト差分（x方向）

- N_x元連立一次方程式をN_y回解く

$$[A_x]\{\phi'_{i,1}\}=\{f(\phi_{i,1})\}$$

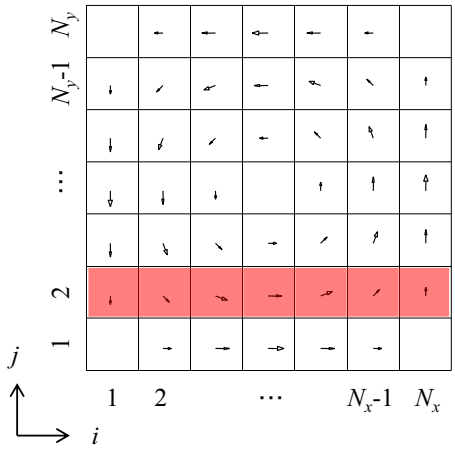



```
do j=1,2,・・・Ny
  φ'j=A-1f(φj)
end do
```

2次元空間の1階コンパクト差分（x方向）

- N_x元連立一次方程式をN_y回解く

$$[A_x]\{\phi'_{i,2}\}=\{f(\phi_{i,2})\}$$

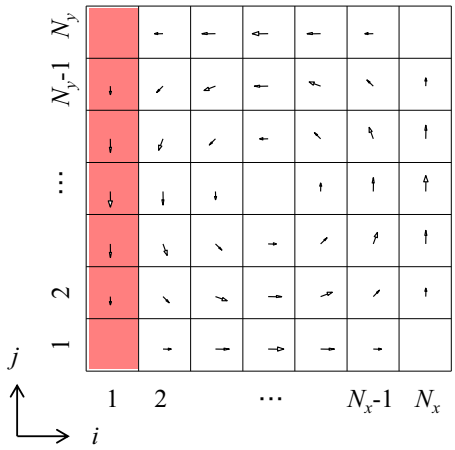



```
do j=1,2,・・・Ny
  φ'j=A-1f(φj)
end do
```


2次元空間の1階コンパクト差分（y方向）

- N_y元連立一次方程式をN_x回解く

$$[A_y]\{\phi'_{1,j}\}=\{f(\phi_{1,j})\}$$

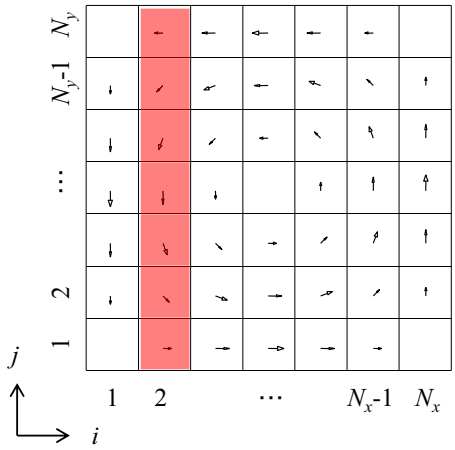



```
do i=1,2,・・・Nx
  φ'i=A-1f(φi)
end do
```

2次元空間の1階コンパクト差分（y方向）

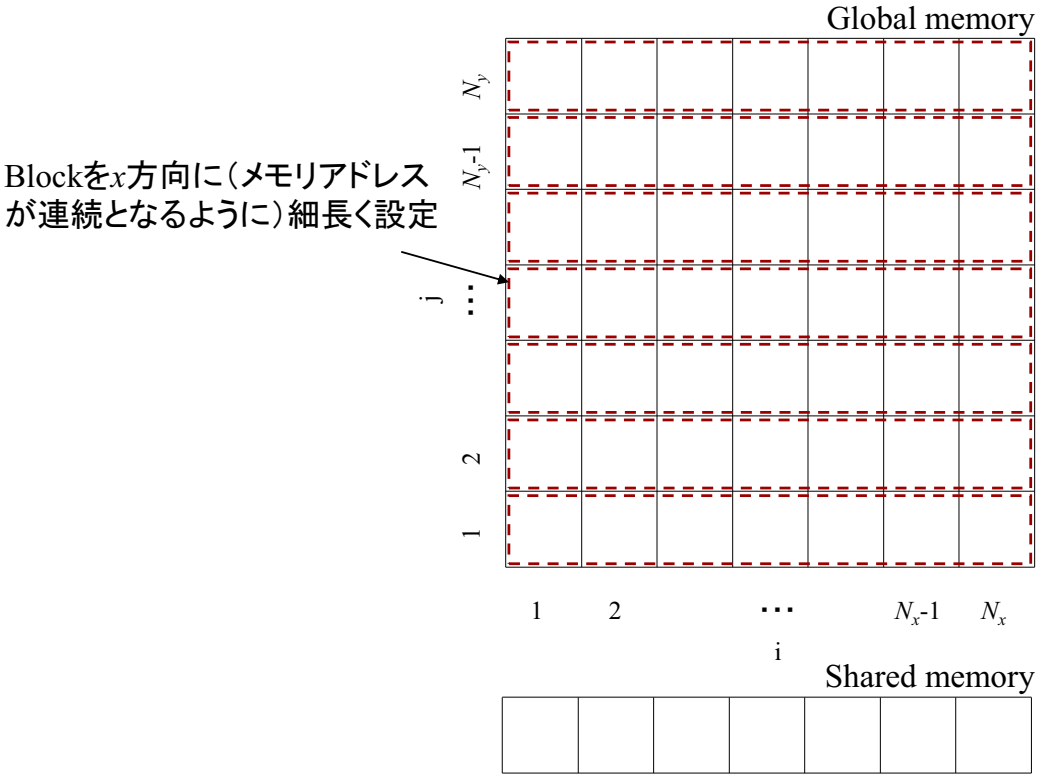
- N_y元連立一次方程式をN_x回解く

$$[A_y]\{\phi'_{2,j}\}=\{f(\phi_{2,j})\}$$

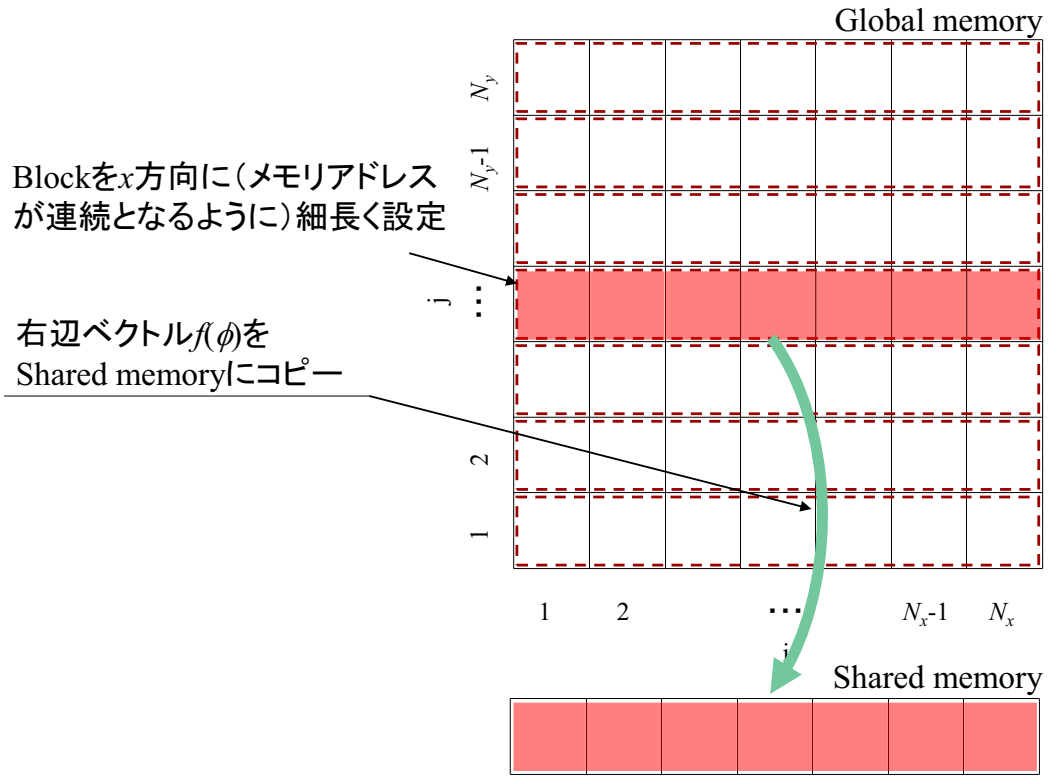



```
do i=1,2,・・・Nx
  φ'i=A-1f(φi)
end do
```

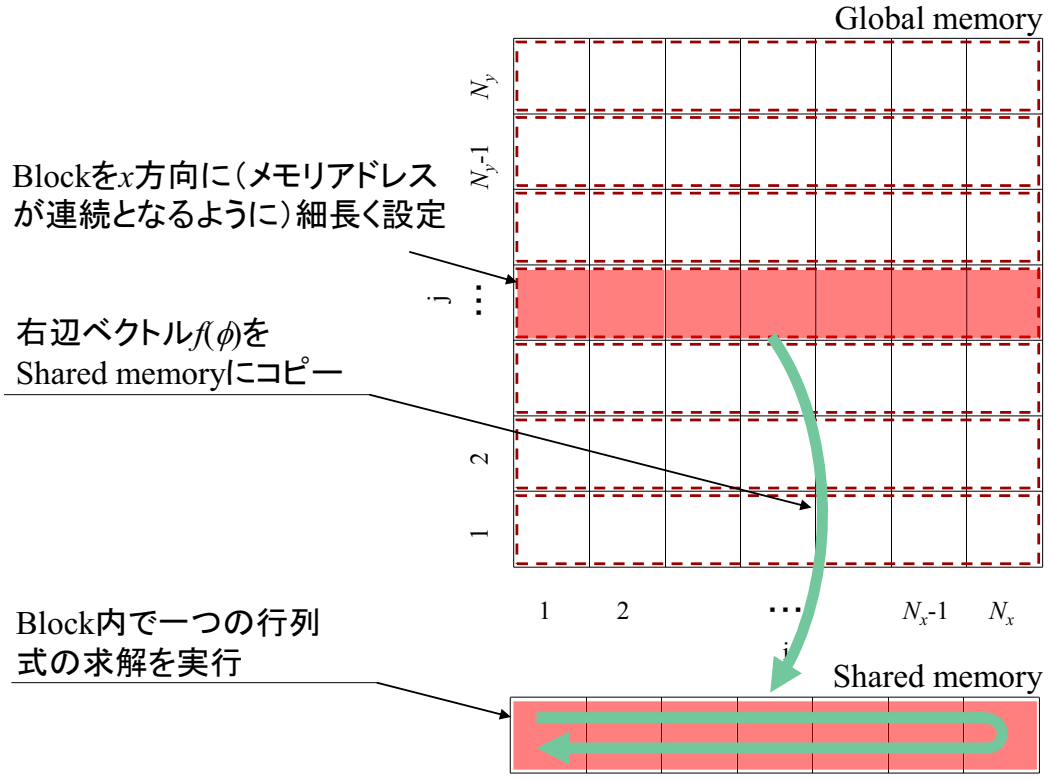

GPUを用いた2次元空間のコンパクト差分



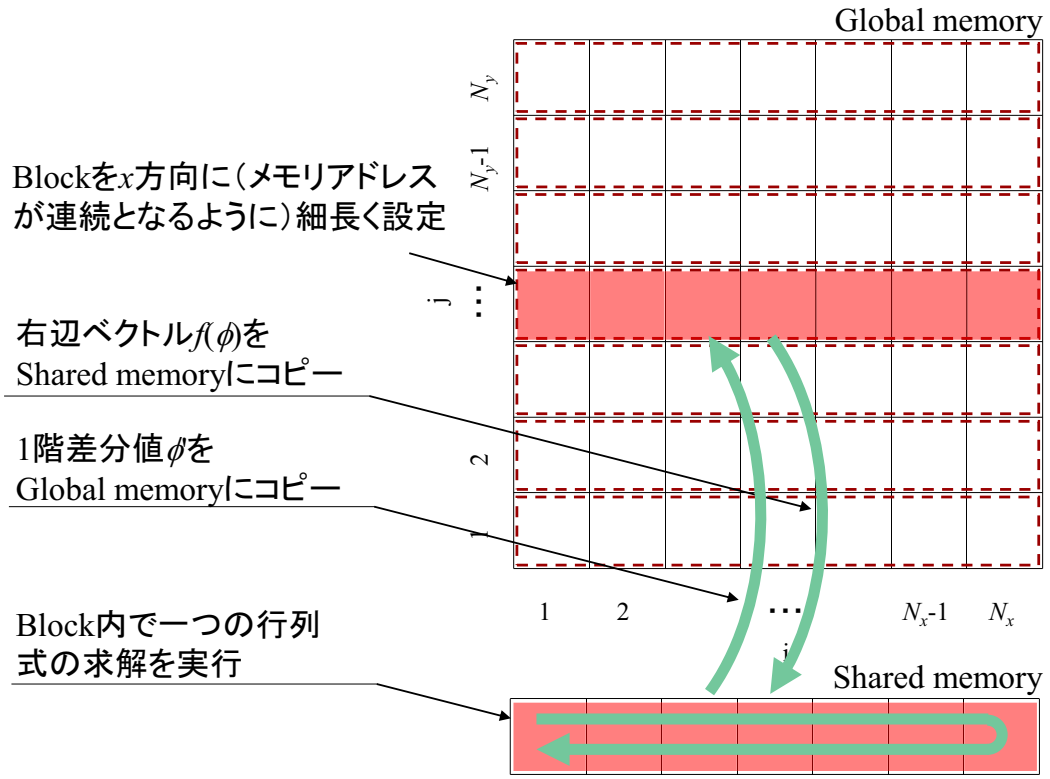
GPUを用いた2次元空間のコンパクト差分



GPUを用いた2次元空間のコンパクト差分



GPUを用いた2次元空間のコンパクト差分



2次元移流方程式

- Rotating Cone
  - 支配方程式

$$\frac{\partial \phi}{\partial t} + u\frac{\partial \phi}{\partial x} + v\frac{\partial \phi}{\partial y} = 0$$

$$(u,v) = (-2\pi y, 2\pi x)$$

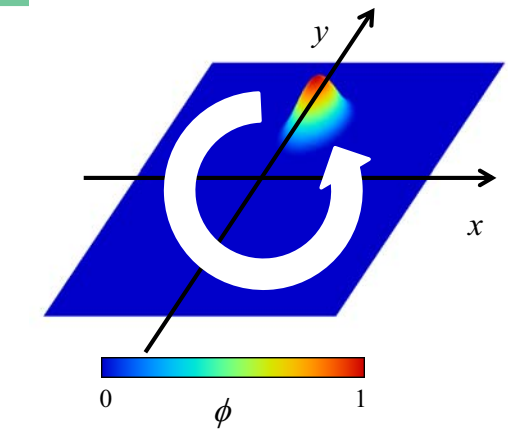
$$-1 \le x \le 1, -1 \le y \le 1$$

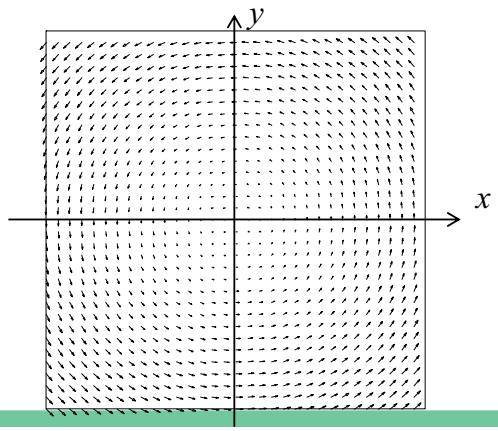
  - 初期条件

$$\phi = \begin{cases} 1 & -0.1 \le x \le 0.1, 0.4 \le y \le 0.6 \\ 0 & \text{otherwise} \end{cases}$$

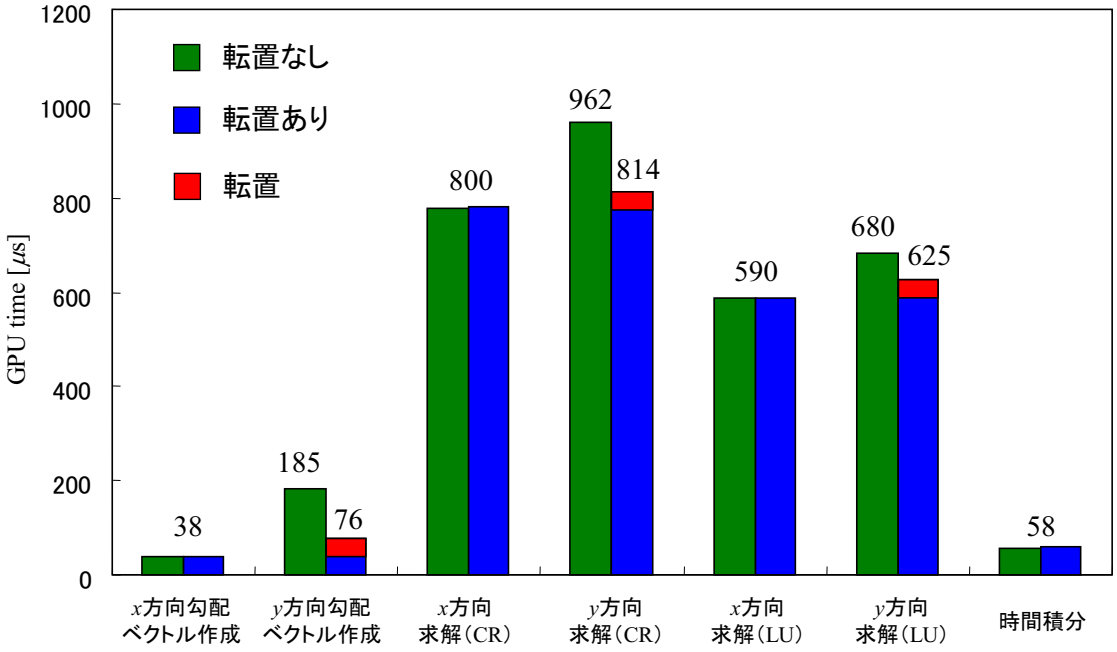
  - 境界条件

$$\phi = 0$$





各解法および行列転置の所要時間



プロファイラ出力

	Registers per Thread	Shared memory per Block	Occupancy
LU分解	10	2072	1
Cyclic Reduction	29	8240	0.125

	Active Threads per Multiporcessor	Active Warps per Multiprocessor	Active Tread Blocks per Multiprocessor
LU分解	1024	32	4
Cyclic Reduction	128	4	1

2次元コンパクト差分のまとめ

- メモリが不連続となる方向の差分は，配列の転置を行うことで実行速度が改善される
- 単一の行列を高速に計算できる解法を用いたとしても，Shared memoryやRegisterが多用されていると，全体の実行時間は増加する

単一渦の流出問題への適用

支配方程式

$\frac{\partial \rho}{\partial t}+\frac{\partial \rho u_i}{\partial x_i}=0$	（質量保存式）
$\frac{\partial \rho u_i}{\partial t}+\frac{\partial \rho u_j u_i}{\partial x_j}+\frac{\partial p}{\partial x_i}=\frac{\partial \tau_{ij}}{\partial x_j}$	（運動量保存式）
$\frac{\partial \rho E}{\partial t}+\frac{\partial \rho u_i E}{\partial x_i}+\frac{\partial u_i p}{\partial x_i}=\frac{\partial u_i \tau_{ij}}{\partial x_i}-\frac{\partial q_i}{\partial x_i}$	（エネルギ保存式）

空間離散化：6次精度コンパクト差分　境界では4次精度片側コンパクト差分と5次精度コンパクト差分の組み合わせ

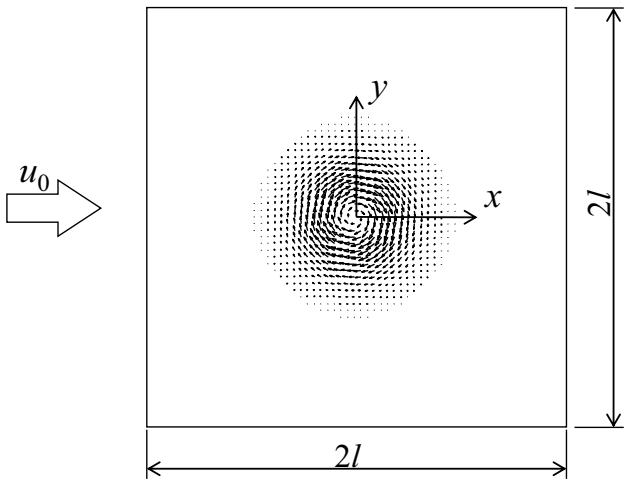
時間離散化：4次精度Runge-Kutta法

ρ :密度　p :圧力　E:全エネルギ=$u_k u_k/2+\varepsilon$　q :熱流束

u :速度　τ_{ij} :粘性応力　ε :内部エネルギ=$p/\rho(\gamma\text{-}1)$

計算対象

- 超音速一様流によって流出する単一渦[7]



条件

マッハ数：　Ma=u_0/c_∞=1.1

レイノルズ数：Re=$u_0 l/\nu$=10000

渦

$$\psi=\Gamma\exp\left(-\frac{x^2+y^2}{2r_c^{\,2}}\right)$$

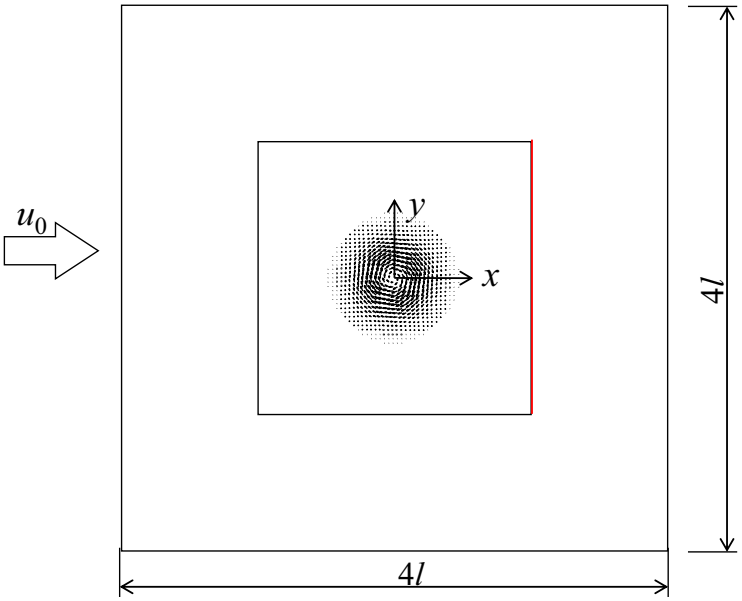
$$p-p_0=-\rho\frac{\Gamma^2}{2r_c^{\,2}}\exp\left(-\frac{x^2+y^2}{r_c^{\,2}}\right)$$

$\Gamma/(c_\infty l)$= -0.0005

r_c/l=0.15

[3] Poinsot, T.J. and Lele, S.K., J. Comput. Phys., Vol.101, 104-129, 1992.

計算条件



格子分割数：512×512

境界条件：

Navier-Stokes Characteristic Boundary Condition[7]

計算環境

CUDA 2.3

CPU：Core i7 920

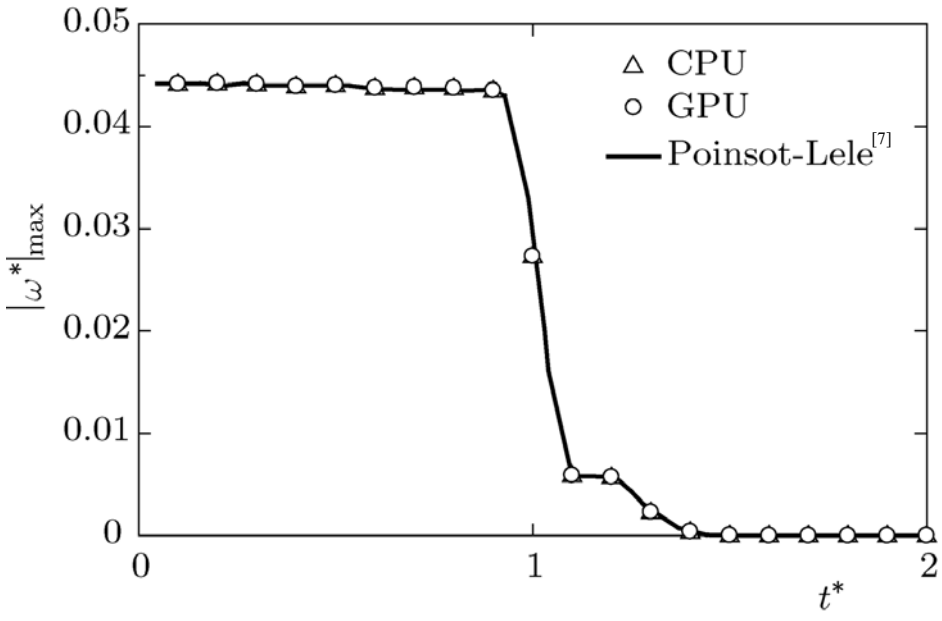
GPU：NVIDIA Tesla C1060

計算はすべて倍精度で実行

$4l \times 4l$の領域を1024×1024に分割して計算を行い，参照解とした

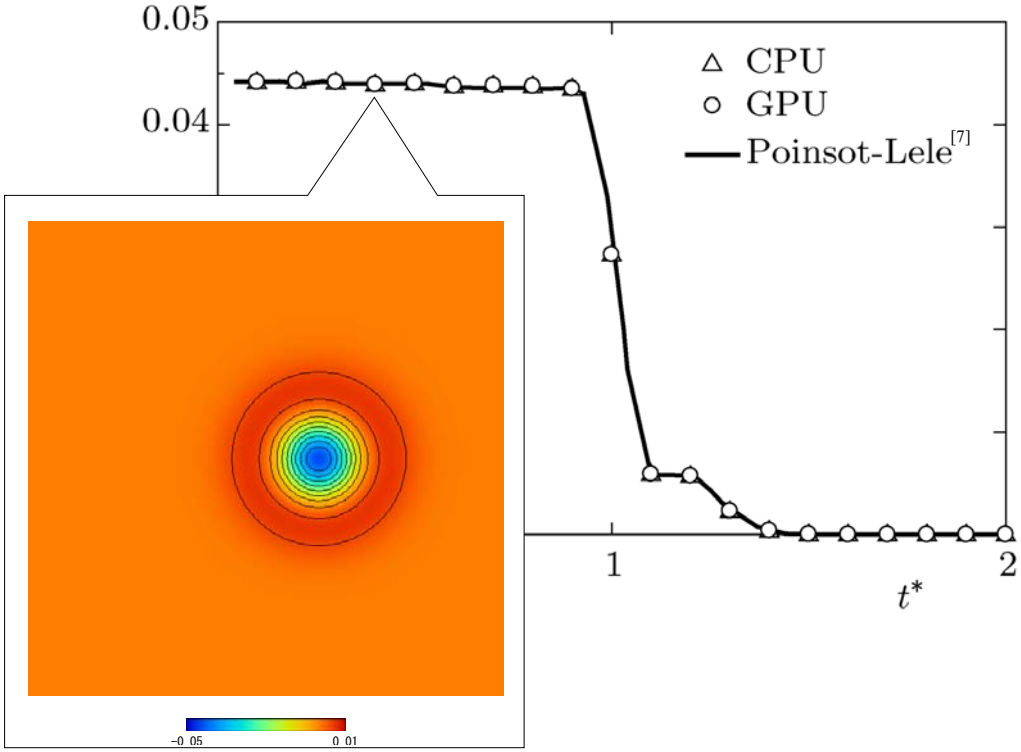
[7] Poinsot, T.J. and Lele, S.K., J. Comput. Phys., Vol.101, 104-129, 1992.

計算領域内の絶対渦度の最大値



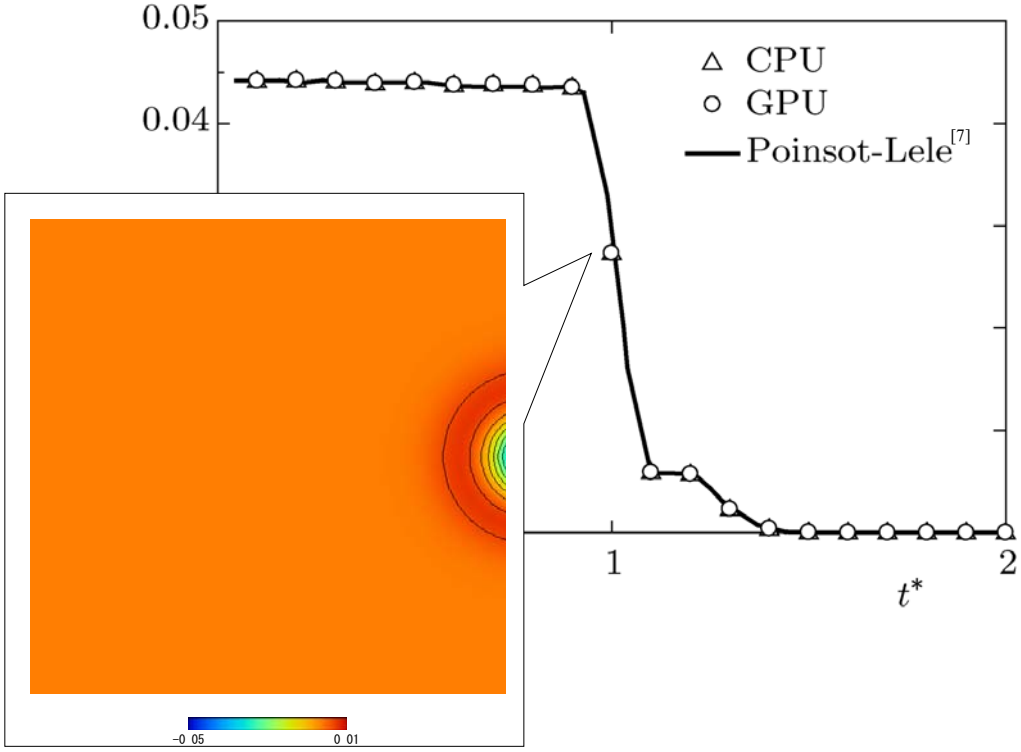
[3] Poinsot, T.J. and Lele, S.K., J. Comput. Phys., Vol.101, 104-129, 1992.

計算領域内の絶対渦度の最大値



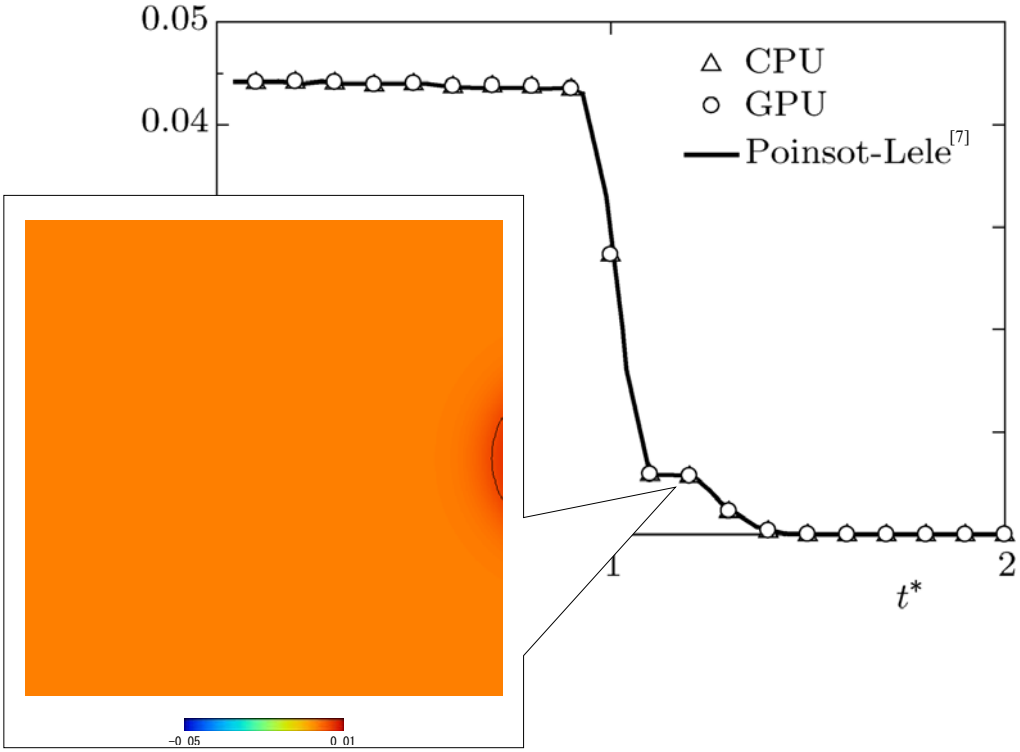
[3] Poinsot, T.J. and Lele, S.K., J. Comput. Phys., Vol.101, 104-129, 1992.

計算領域内の絶対渦度の最大値



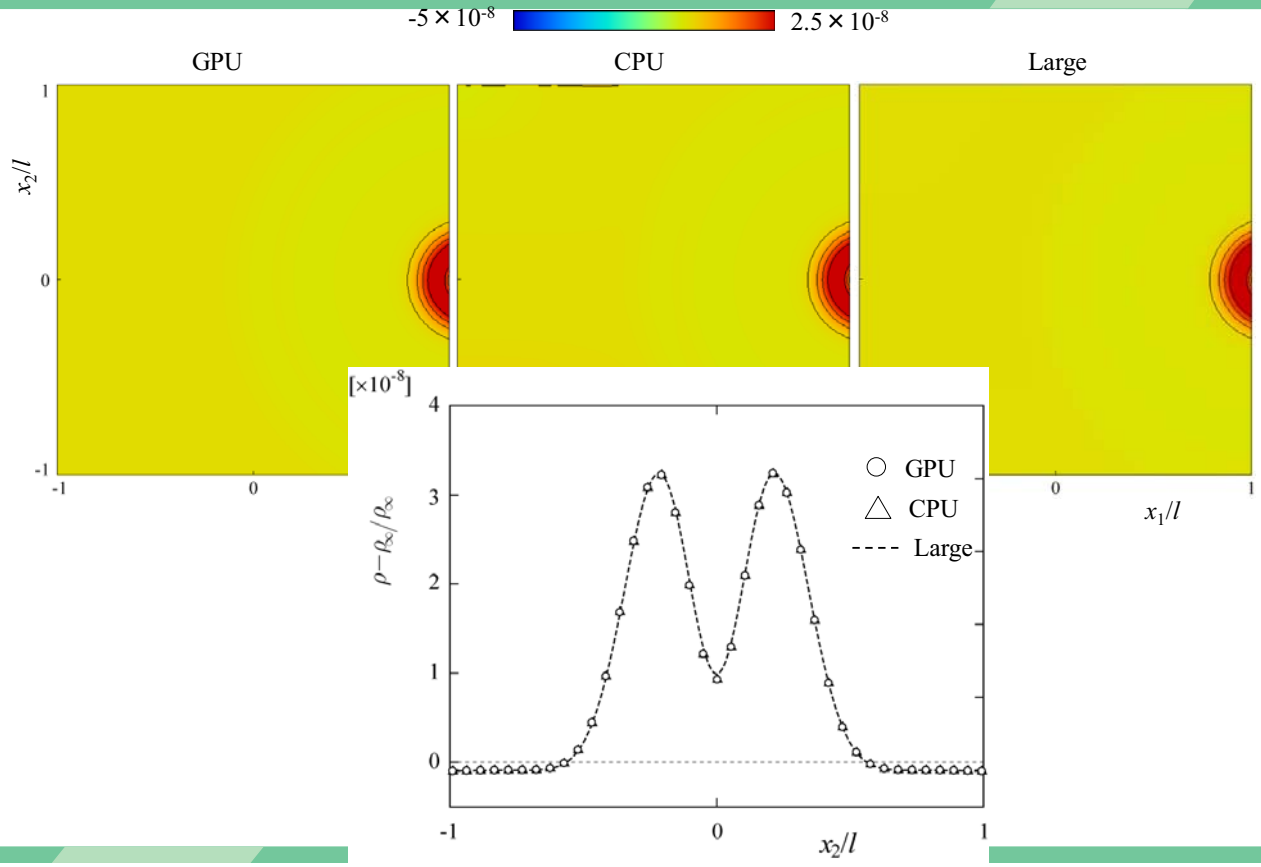
[3] Poinsot, T.J. and Lele, S.K., J. Comput. Phys., Vol.101, 104-129, 1992.

計算領域内の絶対渦度の最大値

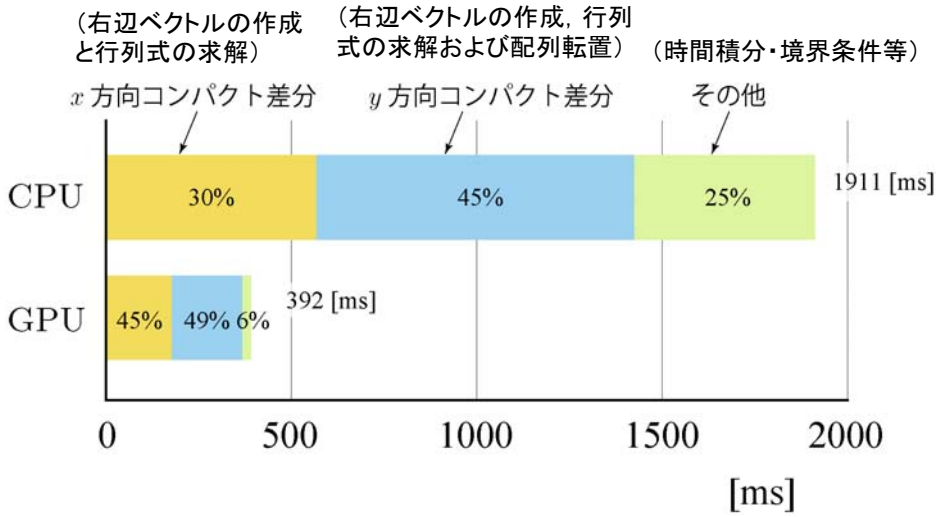


[3] Poinsot, T.J. and Lele, S.K., J. Comput. Phys., Vol.101, 104-129, 1992.

密度場$(\rho-\rho_\infty)/\rho_\infty$の瞬時分布($t$=1)



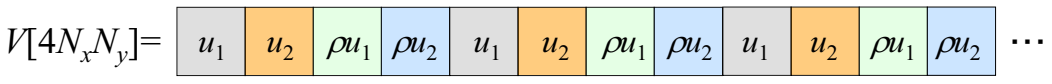
1計算ステップあたりの所用時間

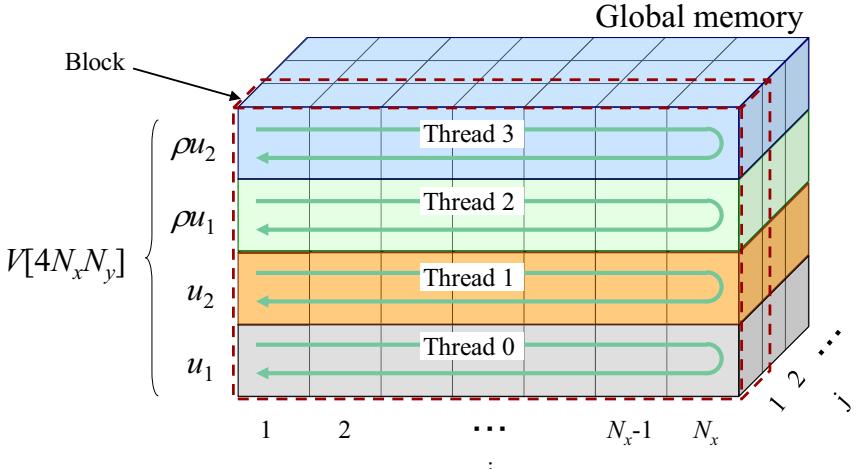


その他の処理は，GPUで計算を行うことにより，約20倍高速化
コンパクト差分は3～4倍程度

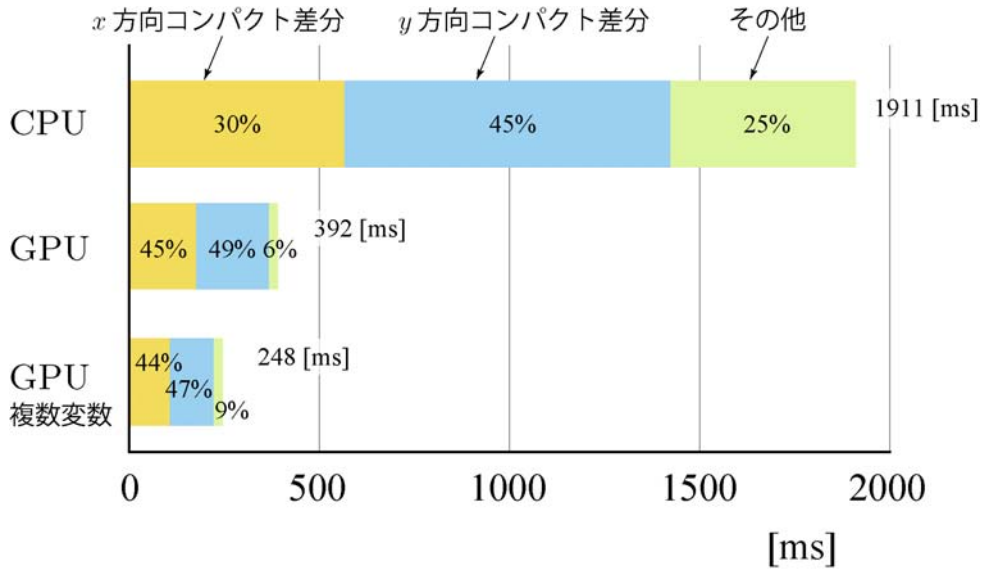
複数変数のコンパクト差分の実行

- Threadの同時実行数を向上
  - 1Blockが複数の変数に対してコンパクト差分を実行
  - 14変数のうち，8変数と6変数をまとめてコンパクト差分を実行





1ステップあたりの所用計算時間



まとめ

- GPUへコンパクト差分を実装するために，行列式の解法の適用性を調査した
- 1次元のコンパクト差分では
  - GPU上でのコンパクト差分の計算において，Cyclic Reduction法は有用である
  - CG法系統の解法を適用するには，内積がボトルネックとなりうる
  - 不連続捕獲（散逸コンパクトスキーム）を用いた場合，CG法系統の解法は反復回数が著しく増加する

まとめ

- 2次元のコンパクト差分では
  - メモリが不連続となる方向の差分は，配列の転置を行うことで実行速度が改善される
  - 単一の行列を高速に計算できる解法を用いたとしても，Shared memoryやRegisterが多用されていると，全体の実行時間は増加する
  - 並列化可能でShared memoryやRegisterの使用量が少ない行列式の解法が必要
- 圧縮性流体計算へ適用し
  - CPUに対して約5倍の高速化が達成された
  - 1Blockが複数の変数のコンパクト差分を計算するカーネルを実装したところ，CPUに対して計算が約7.7倍高速化された