CUDA による高速フーリエ変換の並列化についての研究

M2009MM018 小笠原将也
指導教員：杉浦洋

1 はじめに

Graphics Processing Unit(GPU) はグラフィックス処理の為に設計された並列プロセッサである．今日では, GPU は計算のスループットとメモリ帯域幅が CPU よりとても優れており, GPU をグラフィックス以外の処理の為に利用する努力がなされている．GPU による並列計算のプログラムを作る為には, グラフィックス API と GPU のアーキテクチャの詳しい知識が必要であった．しかし, NVIDIA が自社 GPU における GPU プログラミングの為に開発した CUDA は C 言語を拡張した仕様を持っており, グラフィックス API や GPU アーキテクチャの知識をあまり必要とせずに, C 言語のプログラムを作るかのように, GPU プログラムを作ることができる．本研究では, CUDA を利用して, 2 次元高速フーリエ変換を GPU で計算した場合の性能について考えた．基数 2 の Stockham アルゴリズム, 基数 4 の Stockham アルゴリズム, 基数 4 の Cooley-Tukey アルゴリズムを使った 2 次元並列高速フーリエ変換のアルゴリズムをそれぞれ考え, それらに基づいたプログラムをそれぞれ作り, 実行時間と FLOPS を調べた．2, 3, 4 章及び 6.1, 6.2, 6.3 節では文献 [1] を, 1, 5 章では文献 [2],[3] を参考にした．

2 いくつかの記法

本論文では, ベクトルや行列の添字は基本的に 0 から始まる．

行列 $A \in \mathbb{C}^{m\times n}$ の部分行列を $A(u,v)$ と書く．ここで, u, v は, 部分行列を定める A の行番号と列番号を成分とするような整数ベクトルである．例えば, $A=(a_{ij})$, $u=(0,2)$, $v=(0,1,3)$, $B=A(u,v)$ ならば

$$B=\begin{pmatrix} a_{00} & a_{01} & a_{03} \\ a_{20} & a_{21} & a_{23} \end{pmatrix}.$$

ベクトル $(0,1,...,n-1)$ の部分ベクトルで, 初項 i, 末項 j, 公差 k の整数列を成分とするベクトルを

$$i:k:j=(i,i+k,...,j),\quad 0\le i<n,\ 0\le j<n$$

と書く．公差 $k=1$ のときはそれを省略し,

$$i:j=(i,i+1,...,j)$$

と書く．特に, 元のベクトルを

$$:=(1,2,...,n-1)$$

と書く．

$x\in\mathbb{C}^n$, $n=rc$ ならば, $x_{r\times c}$ を次のように定める．

$$\begin{aligned} x_{r\times c} &= (x(0:r-1)|x(r:2r-1)|\cdots|x(n-r:n-1)) \\ &\in \mathbb{C}^{r\times c}. \end{aligned}$$

明らかに $(x_{r\times c})_{kj}=x_{jr+k}$ が成り立つ．

行列 $A\in\mathbb{C}^{p\times q}$, $B\in\mathbb{C}^{m\times n}$ に対して, クロネッカー積 $A\otimes B$ を次のように定める．

$$\begin{aligned} A\otimes B &= \begin{pmatrix} a_{00}B & \cdots & a_{0,q-1}B \\ \vdots & & \vdots \\ a_{p-1,0}B & \cdots & a_{p-1,q-1}B \end{pmatrix} \\ &\in \mathbb{C}^{pm\times qn}. \end{aligned}$$

3 離散フーリエ変換

$y=(y_0,...,y_{n-1})^T\in\mathbb{C}^n$, $x=(x_0,...,x_{n-1})^T\in\mathbb{C}^n$ とする．$k=0,...,n-1$ に対して

$$y_k=\sum_{j=0}^{n-1}\omega_n^{kj}x_j,\quad \omega_n=e^{-\frac{2\pi i}{n}} \tag{1}$$

が成り立つとき, y は x の $\mathbb{C}^n$ 上の離散フーリエ変換 (DFT) であるという．(1) を次のように表わすこともできる．

$$\begin{aligned} y &= F_n x, \\ F_n &= (f_{pq}),\ f_{pq}=\omega_n^{pq}=e^{-\frac{2\pi pqi}{n}} \end{aligned} \tag{2}$$

1 回の実数の計算 (足し算または掛け算) を 1flops と数える．(2) を普通に計算すると $8n^2$ flops の実数の計算が必要である．しかし, 計算方法を工夫すればより少ない計算回数で計算できる．DFT をより少ない計算回数で計算するアルゴリズムを高速フーリエ変換 (FFT) という．

4 高速フーリエ変換

4.1 いくつかの定義

$n=pm$ のとき, mod p perfect shuffle $\Pi_{p,n}$ を次のように定める．

$$\begin{aligned} \Pi_{p,n} &= I_n(:,v), \\ v &= (0:p:n-1,1:p:n-1,...,p-1:p:n-1). \end{aligned}$$

$n=p_1\cdots p_t$, $\rho=(p_1,...,p_t)$ とする．$m=p_1\cdots p_{t-1}$ のとき, index-reversal permutation $P_n(\rho)$ を次のように定める．

$$P_n(\rho)=\begin{cases} I_n & (t=1), \\ \Pi_{p_t,n}(I_{p_t}\otimes P_m(\rho(1:t-1))) & (t>1), \end{cases}$$

また, 関数 $r_{n,\rho}:\{0,...,n-1\}\to\{0,...,n-1\}$ を次のように定める．

$$y=P_n(\rho)^T x \Leftrightarrow y_{r_{n,\rho}(k)}=x_k,\ k=0:n-1.$$

4.2 Cooley-Tukey アルゴリズムと Stockham アルゴリズム

$n = p_1 \cdots p_t$, $\rho = (p_1, ..., p_t)$ に対して次のように定める．$q = 1, ..., t$ のとき,

$$L_q = p_1 \cdots p_q,\ r_q = \frac{n}{L_q},\ \omega_{L_q} = e^{-\frac{2\pi i}{L_q}},$$
$$\Omega_{p_q, L_{q-1}} = diag(1, \omega_{L_q}, ..., \omega_{L_q}^{(L_{q-1})-1}),$$
$$B_{p_q, L_q} = (F_{p_q} \otimes I_{L_{q-1}}) diag(I_{L_{q-1}}, \Omega_{p_q, L_{q-1}}, ..., \Omega_{p_q, L_{q-1}}^{p_q - 1}).$$

もしも

$$A_q = I_{r_q} \otimes B_{p_q, L_q},$$
$$G_q = (B_{p_q, L_q} \otimes I_{r_q})(\Pi_{p_q, L_q}^T \otimes I_{r_q})$$

ならば, 次の因子分解を得る．

$$F_n = A_t \cdots A_1 P_n(\rho)^T, \tag{3}$$
$$F_n = G_t \cdots G_1. \tag{4}$$

(3), (4) をそれぞれ Cooley-Tukey factorization, Stockham factorization という．$p_1 = \cdots = p_t = 2$ のとき, (4) を利用して $x \leftarrow F_n x$ を計算するアルゴリズムを基数 2 の Stockham アルゴリズムという．そのアルゴリズムの計算回数は $5n \log_2 n$ flops である．$p_1 = \cdots = p_t = 4$ のとき, (3), (4) を利用して同様の計算をするアルゴリズムをそれぞれ基数 4 の Cooley-Tukey アルゴリズム, 基数 4 の Stockham アルゴリズムという．この 2 つのアルゴリズムの計算回数はいずれも $4.25n \log_2 n$ flops である．

4.3 2 次元高速フーリエ変換

$X \in \mathbb{C}^{n_1 \times n_2}$ とする．

$$X \leftarrow F_{n_1} X$$

を multicolumn DFT 問題という．この計算は X のそれぞれの列 X_j の DFT $F_{n_1} X_j$ を必要とする．

$x \in \mathbb{C}^n$, $n = n_1 n_2$ のとき

$$x \leftarrow (F_{n_2} \otimes F_{n_1}) x \tag{5}$$

を 2 次元 DFT 問題という．$x_{n_1 \times n_2}$ を $X \in \mathbb{C}^{n_1 \times n_2}$ とみなすとき, (5) は次のように変形できる．

$$X \leftarrow F_{n_1} X F_{n_2} = (F_{n_2} (F_{n_1} X)^T)^T. \tag{6}$$

(6) の計算回数は, それの計算において必要な 1 次元 DFT を基数 2 の Stockham アルゴリズムで計算した場合, 4.2 節より,

$$5n \log_2 n\ (flops), \quad n_1 = 2^{t_1},\ n_2 = 2^{t_2}$$

である．基数 4 の Cooley-Tukey アルゴリズムまたは基数 4 の Stockham アルゴリズムで計算した場合,

$$4.25n \log_2 n\ (flops), \quad n_1 = 4^{t_1},\ n_2 = 4^{t_2}$$

である．

5 GPU と CUDA

5.1 ブロックとスレッド

CUDA による GPU 並列プログラムにおいて, 並列処理はカーネル関数という void 型の並列処理用関数の中で行われる．カーネル関数を実行する直前に「ブロック数」と「1 ブロック内のスレッド数」を指定しなければならない．ブロック数を C, 1 ブロック内のスレッド数を D と指定した場合, カーネル関数の実行開始時に C 個のブロックが作られ, それぞれのブロックの中に D 個のスレッドが作られる．C 個のブロックを B_c $(c = 0, ..., C-1)$ とし, B_c を第 c ブロックというとする．また,「第 c ブロック内の第 d スレッド」$(c = 0, ..., C-1,\ d = 0, ..., D-1)$ を T_d^c とする．このとき, 以下のブロックとスレッドが作られることになる．

$$\begin{array}{ll} B_0; & T_0^0, T_1^0, ..., T_{D-1}^0, \\ B_1; & T_0^1, T_1^1, ..., T_{D-1}^1, \\ \vdots & \quad\vdots \\ B_{C-1}; & T_0^{C-1}, T_1^{C-1}, ..., T_{D-1}^{C-1}. \end{array}$$

スレッドはカーネル関数を実行する主体である．任意のスレッド T_d^c の実行内容を, c, d をパラメータとする 1 つのカーネル関数のコードとして書く．全ての T_d^c がそのコードを並列に実行することによって並列処理が実現する．

5.2 デバイスメモリ

カーネル関数内で扱われる変数が確保するメモリの種類には, グローバルメモリ, シェアードメモリ, コンスタントメモリ, レジスタがある．これらは総称してデバイスメモリといわれる．

グローバルメモリ 任意のスレッドがアクセスできて容量も大きいが, アクセス速度は遅い．カーネル関数の引数はグローバルメモリの変数である．主に「並列処理する配列データ」を格納する為に使われる．グローバルメモリの変数を $GA, GB, ...$ のように表わすとする．ただし, グローバルメモリの変数であることを意識する必要がない場合は $A, B, ...$ のように表わすとする．

コンスタントメモリ 任意のスレッドがアクセスできる．グローバルメモリより容量は小さいがアクセス速度は速い．カーネル関数内では読み込み専用であり, 書き込みはできない．定数データを格納する為に使われる．コンスタントメモリの変数を $CA, CB, ...$ のように表わすとする．変数はグローバル変数である．

シェアードメモリ 任意の $c \in \{0, ..., C-1\}$ に対して,「第 c ブロック内のシェアードメモリ」はそのブロック内のスレッド $T_0^c, T_1^c, ..., T_{D-1}^c$ だけがアクセスできる．容量は小さいがアクセス速度はとても速い．作業用メモリとして使われる．「第 c ブロック内のシェアードメモリ」の変数を $S_c A, S_c B, ...$ のように表わすとする．

レジスタ 任意の $c \in \{0, ..., C-1\}$, $d \in \{0, ..., D-1\}$ に

対して，「第 c ブロック内の第 d スレッドのレジスタ」はそのスレッド T_d^c だけがアクセスできる．普通のデータを格納する為に使われる．「第 c ブロック内の第 d スレッドのレジスタ」の変数を $R_d^cA, R_d^cB, ...$ または単に $A, B, ...$ のように表わすとする．

5.3 同期

同期には「ブロック内の全スレッドの同期」と「全スレッドの同期」の 2 種類がある．前者はカーネル関数内で実行できるが, 後者はできない．「全スレッドの同期」はカーネル関数の実行が終了する時に自動的に行われる．

6 CUDA による 2 次元並列 FFT

4.3 節より, (6) を計算するには, 1 次元 DFT の計算が必要である．1 次元 DFT を基数 4 の Cooley-Tukey アルゴリズムで計算する場合の, 2 次元並列 FFT アルゴリズムについて考える．

$\rho_1 = (p_{11}, ..., p_{1t_1})$, $\rho_2 = (p_{21}, ..., p_{2t_2})$ (全ての $p_{ij} = 4$), $n_1 = p_{11} \cdots p_{1t_1} = 4^{t_1}$, $n_2 = p_{21} \cdots p_{2t_2} = 4^{t_2}$, $X \in \mathbb{C}^{n_1 \times n_2}$ とする．(3) より, 次のような流れで (6) を並列計算することができる．

1. $X \leftarrow P_{n_1}(\rho_1)^T X$ をカーネル関数の外で計算する,
2. $X \leftarrow A_{t_1} \cdots A_1 X$ をカーネル関数で並列計算する,
3. $Y \leftarrow X^T$ をカーネル関数で並列に実行する,
4. $Y \leftarrow P_{n_2}(\rho_2)^T Y$ をカーネル関数で並列計算する,
5. $Y \leftarrow A_{t_2} \cdots A_1 Y$ をカーネル関数で並列計算する,
6. $X \leftarrow Y^T$ をカーネル関数で並列に実行する,

第 $2, ..., 6$ 項のそれぞれの間には「全スレッドの同期」が必要である．つまりカーネル関数を 5 回実行してこれらの項の計算を行うことになる．

6.1 回転因子行列

回転因子行列 $W_{n_1}^{(long)} \in \mathbb{C}^{\frac{n_1-1}{3} \times 3}$ を次のように定義する．$q = 1, ..., t_1$ に対して,

$$
\begin{aligned}
&W_{n_1}^{(long)}(\frac{L_*-1}{3} + j, 0) = \omega_L^j, \\
&W_{n_1}^{(long)}(\frac{L_*-1}{3} + j, 1) = \omega_L^{2j}, \\
&W_{n_1}^{(long)}(\frac{L_*-1}{3} + j, 2) = \omega_L^{3j}, \\
&\quad j = 0, 1, ..., L_* - 1.
\end{aligned} \tag{7}
$$

ただし, L_q, L_{q-1} をそれぞれ単に L, L_* と表わす．

6.2 $Y = A_q X$

$q = 1, ..., t_1$ に対して, $Y = A_q X \Leftrightarrow Y_{col} = A_q X_{col}$ (X_{col}, Y_{col} は X, Y の第 col 列, $col = 0, ..., n_2 - 1$) である．

$$
Y_{col} = A_q X_{col}
$$
$$
\Leftrightarrow \begin{cases} Y_{col}(kL + j) = \tau_{jk0} + \tau_{jk2}, \\ Y_{col}(kL + L_* + j) = \tau_{jk1} - i\tau_{jk3}, \\ Y_{col}(kL + 2L_* + j) = \tau_{jk0} - \tau_{jk2}, \\ Y_{col}(kL + 3L_* + j) = \tau_{jk1} + i\tau_{jk3}, \end{cases} \tag{8}
$$
$$
j = 0, ..., L_* - 1, \ k = 0, ..., r - 1
$$

が成り立つ． ただし, r_q を単に r と表わし, $\alpha_{jk} = X_{col}(kL + j)$, $\beta_{jk} = \omega_L^j X_{col}(kL + L_* + j)$, $\gamma_{jk} = \omega_L^{2j} X_{col}(kL + 2L_* + j)$, $\delta_{jk} = \omega_L^{3j} X_{col}(kL + 3L_* + j)$, $\tau_{jk0} = \alpha_{jk} + \gamma_{jk}$, $\tau_{jk1} = \alpha_{jk} - \gamma_{jk}$, $\tau_{jk2} = \beta_{jk} + \delta_{jk}$, $\tau_{jk3} = \beta_{jk} - \delta_{jk}$ とする．

(8) より, 1 つの col に対して $L_* r = \frac{n_1}{4}$ 個の組 (j, k) が存在する．そこで, $X_{col} \leftarrow A_q X_{col}$ を並列計算する場合, 列 col とブロック B_{col} を, 組 (j, k) とスレッド T_d^{col} $(d = 0, ..., \frac{n_1}{4})$ をそれぞれ対応させることができる．このとき, $(j, k) \leftrightarrow d = kL_* + j$ と対応させる．また, (7) も利用する．

6.3 $X \leftarrow A_{t_1} \cdots A_1 X$

以上より, $X \leftarrow A_{t_1} \cdots A_1 X$ を並列計算するカーネル関数のアルゴリズムは概して次のようになる． ただし, $W_{n_1}^{(long)}$ の値をコンスタントメモリ変数 CW_1 に, 整数ベクトル $Idx = (r_{n_2,\rho_2}(0), r_{n_2,\rho_2}(1), ..., r_{n_2,\rho_2}(n_2 - 1))$ の値をコンスタントメモリ変数 $CIdx$ にそれぞれ格納する．

1. $\{C = n_2,\ D = \frac{n_1}{4},\ T_d^{col}\ do\ this\ (col = 0, ..., C - 1,\ d = 0, ..., D - 1).\}$
2. $\{X = GX,\ Y = GY;\ W_1 = CW_1,\ Idx = CIdx\}$
3. $S_{col}x \leftarrow GX_{col}$
4. synchronize all T_d^{col} in B_{col}
5. for $q = 1 : t_1$
   (a) $L \leftarrow 4^q;\ L_* \leftarrow \frac{L}{4}$
   (b) $k \leftarrow floor(\frac{d}{L_*});\ j \leftarrow d - kL_*;\ v \leftarrow \frac{L_*-1}{3}$
   (c) $\alpha \leftarrow S_{col}x[kL + j]$
   (d) $\beta \leftarrow CW_1[v + j, 0] S_{col}x[kL + L_* + j]$
   (e) $\gamma \leftarrow CW_1[v + j, 1] S_{col}x[kL + 2L_* + j]$
   (f) $\delta \leftarrow CW_1[v + j, 2] S_{col}x[kL + 3L_* + j]$
   (g) $\tau_0 \leftarrow \alpha + \gamma;\ \tau_1 \leftarrow \alpha - \gamma$
   (h) $\tau_2 \leftarrow \beta + \delta;\ \tau_3 \leftarrow \beta - \delta$
   (i) $S_{col}x[kL + j] \leftarrow \tau_0 + \tau_2$
   (j) $S_{col}x[kL + L_* + j] \leftarrow \tau_1 - i\tau_3$
   (k) $S_{col}x[kL + 2L_* + j] \leftarrow \tau_0 - \tau_2$
   (l) $S_{col}x[kL + 3L_* + j] \leftarrow \tau_1 + i\tau_3$
   (m) synchronize all T_d^{col} in B_{col}
6. end
7. $GX_{col} \leftarrow S_{col}x$
8. synchronize all T_d^{col}

第 1 行は「ブロック数を $C = n_2$, 1 ブロック内スレッド数を $D = \frac{n_1}{4}$ とし, 1 つ 1 つのスレッド T_d^{col} $(col = 0, ..., C - 1,\ d = 0, ..., D - 1)$ が第 3 行以下のコードを実行する」ということを意味する．第 2 行は「グローバルメモリ変数 GX, GY, コンスタントメモリ変数 $CW_1, CIdx$ が利用できる」ということを意味する．入力データは GX に格納されており, 出力データも GX に格納される．第 $3, 7$ 行においては, 1 つのスレッド番号 d がいくつかの行番号と対応している．B_{col} 内の全てのスレッドが協力し

表 1 実行環境

OS	Fedora 12
プロセッサ	Core i7 CPU 860, 2.80GHz
GPU	GeForce GTX 480

表 2 基数 2 の Stockham アルゴリズムを利用した場合

	実行時間 (ms)	GFLOPS
CPU	74.9	1.40
GPU	4.03	26.0
$\frac{GPU}{CPU}$ (比)	0.0538	18.6

てデータのコピーを行う. 第 4, 5(m) 行は「ブロック内の全スレッドを同期する」すなわち「ここに来たスレッド T_d^{col} は B_{col} 内の他の全てのスレッドがここに来るまで待つ」ということを意味する. 第 5(a)-(m) 行は, 1 つ 1 つのスレッド T_d^{col} が協力して, $S_{col}x \leftarrow A_q S_{col}x$ を計算する. 第 8 行は「全スレッドを同期する」すなわち「ここに来たスレッドは他の全てのスレッドがここに来るまで待つ」ということを意味する. 第 8 行が終了すればこのカーネル関数は実行終了する.

6.4 $Y \leftarrow P_{n_2}(\rho_2)^T(A_{t_1}\cdots A_1X)^T$

6 章の, 第 2, 3, 4 項をまとめて「2. $Y \leftarrow P_{n_2}(\rho_2)^T(A_{t_1}\cdots A_1X)^T$ をカーネル関数で並列計算する」とすることができる. 6.3 節のアルゴリズムの第 7 行を

$$GY^{(CIdx[col])} \leftarrow S_{col}x$$
$$(GY^{(CIdx[col])}\text{は } GY \text{ の第 } CIdx[col] \text{ 行})$$

とと変えればよい. 同様にして, 第 5, 6 項をまとめて「5. $X \leftarrow (A_{t_2}\cdots A_1Y)^T$ をカーネル関数で並列計算する」とすることができる.

以上によって, 1 次元 DFT を基数 4 の Cooley-Tukey アルゴリズムで計算する場合の, (6) の 2 次元並列 FFT アルゴリズムが作られた. (4) を用いた 2 次元並列 FFT のアルゴリズムも上とほとんど同様の考えで導くことができる.

7 数値実験

本研究では, 1 次元 DFT を計算する為に

1. 基数 2 の Stockham アルゴリズム,
2. 基数 4 の Stockham アルゴリズム,
3. 基数 4 の Cooley-Tukey アルゴリズム

を使った 2 次元並列 FFT プログラムをそれぞれ作った. それらを表 1 の計算機環境で実行し, 実行時間と FLOPS を調べた. ただし, GPU プログラムの実行時間はカーネル関数 (並列処理用関数) の実行時間であり, 入力行列のサイズは 1024×1024 である. 実行結果は表 2, 3, 4 のようになった.

表 2 と 3 を見比べると, 基数を増やすことによる高速化の度合いが, CPU プログラムよりも GPU プログラムの方が大きいことがわかる. また, 基数 2 よりも基数 4 の方が

表 3 基数 4 の Stockham アルゴリズムを利用した場合

	実行時間 (ms)	GFLOPS
CPU	57.9	1.54
GPU	1.74	51.2
$\frac{GPU}{CPU}$ (比)	0.0301	33.2

表 4 基数 4 の Cooley-Tukey アルゴリズムを利用した場合

	実行時間 (ms)	GFLOPS
CPU	48.4	1.84
GPU	1.15	77.4
$\frac{GPU}{CPU}$ (比)	0.0238	42.1

より効率的に並列計算できることがわかる. 基数 4 の 2 つのアルゴリズムは計算回数が等しいが, 表 3 と 4 を見比べると結果に差があることがわかる. これは, Cooley-Tukey アルゴリズムはインプレースな変換をするが Stockham アルゴリズムは作業用メモリを必要とする為であると考えられる. Stockham アルゴリズムよりも Cooley-Tukey アルゴリズムの方が効率的に並列計算できることが表よりわかる.

8 おわりに

より効率的な GPU 並列プログラムを作る為には, GPU のアーキテクチャを理解し, GPU の特性に適したブロックとスレッドの作成数や使い方, メモリアクセスの方法, FFT アルゴリズムの選択等について考える必要がある.

参考文献

[1] Charles Van Loan, *Computational Frameworks for the Fast Fourier Transform*, Society for Industrial and Applied Mathematics, 1992.

[2] Fermi, *Compute Architecture White Paper*, `http://www.nvidia.com/object/fermi_architecture.html`.

[3] NVIDIA, *NVIDIA CUDA Programming Guide Version 3.0*, `http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide.pdf`.