GPGPUにおけるシストリックアルゴリズムの効率的な実現について

茂木, 啓輔 / MOTEGI, Keisuke

(出版者 / Publisher)
法政大学大学院理工学・工学研究科
(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 理工学・工学研究科編 / 法政大学大学院紀要. 理工学・工学研究科編
(巻 / Volume)
57
(開始ページ / Start Page)
1
(終了ページ / End Page)
8
(発行年 / Year)
2016-03-24
(URL)
https://doi.org/10.15002/00013292

GPGPU におけるシストリックアルゴリズムの効率的な実現について

DESIGN AND IMPLEMENTATION OF SYSTOLIC ALGORITHMS ON GPGPU

茂木啓輔
Keisuke MOTEGI

指導教員　和田幸一

法政大学大学院理工学研究科応用情報工学専攻修士課程

Today, image processing have become popular in a wide range of fields such as environmental science medical, computational fluid analysis. GPU is characterized to perform image processing at high speed. But, initially the GPU was developed was a device for performing image processing only. Focusing on the high throughput, techniques for perform a general purpose computing is a GPGPU. And, GPGPU is expected to be applied in a variety of fields. In this paper,we propose an efficient implementation of the systolic algorithm of matrix multiplication on GPGPU.

Key Words：CUDA, GPU, GPGPU, Systlic Array, Matrix Multiplication

1.　はじめに

現在,コンピュータの性能は格段に向上し,パソコンであっても少し前では信じられないような機能が実現できるようになった.なかでもビデオカードの表示性能は高くなり,本物と見紛うばかりの非常にクオリティの高い描画を行うことができる.今日では 3D グラフィックスを扱う画像処理は医療,数値流体解析,環境科学などの分野で広く普及している.3D グラフィックス処理の分野では,長年,並列処理技術が利用されてきた.このような分野で 処理の高速化を目的として GPU(Graphics Processing Unit)は開発された.汎用的な計算を得意とする一般の CPU で同じ処理を行なった場合,GPU ほどのグラフィックス処理能力を得ることはできない.さらに,ここ数年では GPU は演算性能,メモリバンド幅とともに CPU のそれら大きく上回るようになってきている.

GPU が開発された当初は画像処理の高速化のみを目的とした処理装置であったが,この高い演算能力に着目し,グラフィックス分野で蓄積された並列処理技術の資産を応用しようというのが,3D グラフィックス処理以外の汎用計算を行わせる「GPGPU（General Purpose Computation on Graphics Processing Unit)」,あるいは「GPU コンピューティング」である.

本研究の目的は,シストリックアルゴリズムを GPU 上で効率的に実現することである.そしてその例として,行列積の演算を行う.そのために,本研究ではシストリックアルゴリズムを GPU 上で実現する際の理論評価および実装評価を行い,さらにベンチマークとして行列積を求める代表的なアルゴリズムとして 3 重ループを用いたアルゴリズムの理論評価および実装評価も行なう.

2.　GPGPU と CUDA

本章では GPGPU および開発環境である CUDA の特徴について説明していく.まず,GPU の機能である 3D グラフィックス処理をそれ以外の汎用的な計算行わせるようにできたのが「GPGPU（General Purpose Computation on Graphics Processing Unit)」,もしくは「GPU コンピューティング」と呼ばれるものである.GPGPU は,画像処理はもちろんのこと,流体計算,電磁波シミュレーション,天文シミュレーション,たんぱく質などの挙動を解析する数値シミュレーション,バイオ・インフォマティクス,金融工学など, 幅広い分野への適用が行われている.

CUDA とは,GPU を用いて汎用計算処理のプログラムを開発するための C 言語の総合開発環境である.NVIDIA

社から提供されており,本研究ではこの NVIDIA 社の GPU を対象としている.

CUDA の特徴として,まず GPU 側で処理をされるカーネル関数は,スレッドと呼ばれる実行単位で実行される.そして全てのスレッドで同じプログラムが走る.他の特徴として,スレッドはそれぞれ固有のスレッド番号を持っていることが挙げられる.この固有のスレッド番号を用いて,同じプログラムでもそれぞれのスレッドに固有の処理を並列して行わせることが出来る.

CUDA でのプログラムの流れは,CPU 側をホスト,GPU 側をデバイスと呼ぶ.ホスト側のプログラムは CPU 上で動作し,ホストのメモリを使用する.デバイス上で動作するプログラムをカーネルプログラムと呼び,これば GPU 上で処理される.以下の図 1 に CUDA を用いた際のプログラムの流れを示す.[1]

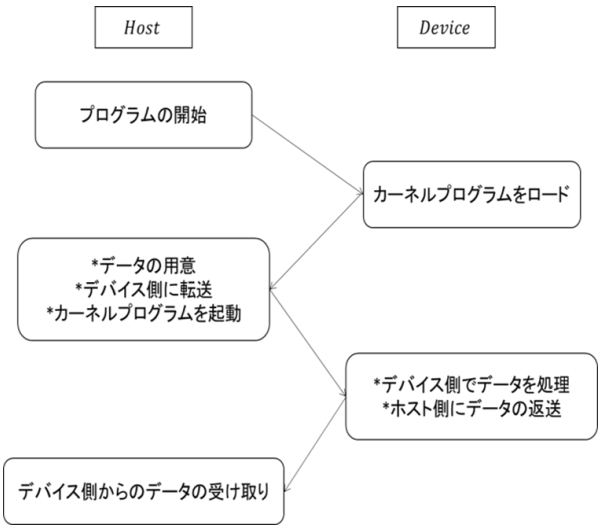

図 1. CUDA のプログラムの流れ

また,CUDA のメモリの特性でコアレッシングとバンクコンフリクトと呼ばれるものがある.バンクコンフリクトは,シェアードメモリの特性で, スレッドがシェアードメモリの同じメモリバンクへアクセスを行った際に,シーケンシャルに処理されてしまう.

コアレッシングはグローバルメモリの特性で連続するグローバルメモリ上のデータにプロセッサがアクセスするとき,32 バイト,64 バイト,128 バイトのまとまったデータ量を転送する事である.ワープ(32 または 16 スレッド)で転送される.

3. 評価に使用する理論モデル

本研究で使用する理論モデルは,既存のモデル[2]では考慮していなかったグローバルメモリのコアレッシングとシェアードメモリのバンクコンフリクトの特徴を踏まえて構成している.[3][4]グローバルメモリ,シェアードメモリへのアクセス時間を区別するために,Lgm,Lsm の 2 種類の独自のパラメータを設定した. 対象となる GPU は,NVIDIA 社の GPU である.以下に理論モデルの図を図 2 に,およびパラメータをまとめた表を表 1 に示す.

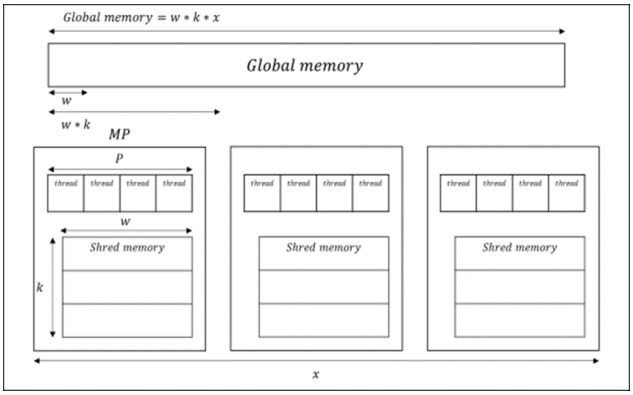

図 2.評価に使用する理論モデル

表 1.評価に使用する理論モデルのパラメータ

スレッド数	p=w*k/2
1バンク内のメモリ数	k
MP数	x
グローバルメモリのレイテンシ	Lgm
シェアードメモリのレイテンシ	Lsm
グローバルメモリのバンク数	wg=2*p
シェアードメモリのバンク数	ws
グローバルメモリ数	gm=N=w*k*x
MPあたりのシェアードメモリ数	sm=w*k
要素数	N

4. シストリックアルゴリズムについて

シストリックという名前は,ネットワークの振舞いが,血管中の血液の流れに似ているところからきている.心臓の拍動によって血液が循環系に送り込まれるように,シストリックアルゴリズムでは,データの流れが規則正しく波となって一列に進んでいく.[5]

シストリックアルゴリズムは,最高速のアルゴリズムというわけではないが,近接したモジュール間の相互結合を有効に使うという点で,非常に良い並列アルゴリズムである.また,アルゴリズムは問題のサイズに応じて,規則的に拡張

することができるという特徴があり,実用的なアルゴリズムである.そして,多くの GPU は SIMD(Single Instruction Multiple Data)のアーキテクチャであり,シストリックアルゴリズムは効率よく実現できるのではないかということが考えられる.

ここからは,シストリックアルゴリズムについて, サイズ N×N の行列同士の積算 A×B=C を求めることを例にして説明していく.

まず,シストリックアルゴリズムはセルという最小単位の演算器で構成されている.セル内には行列の要素同士の積算および積算の結果の足しこみを行なうためのプロセッサ,一時的に入力行列 A,B の要素を保持しておくためのレジスタ A,B,足しこみの結果を保持しておくためのレジスタ C が存在する.

そしてセルには入力行列 A,B の要素をセルに入力するためのA_{in}, B_{in}, 積算および足しこみが完了した後に,隣接するセルに入力行列 A,B の要素を渡すためのA_{out}, B_{out}が存在する.

さらにシストリックアルゴリズムでは入力行列の要素同士の積算および積算の結果の足しこみの結果を隣接するセルに渡すことができるC_{in}, C_{out}を追加することができる.以下の図 3 に二次元の座標の場合のセルを示す.

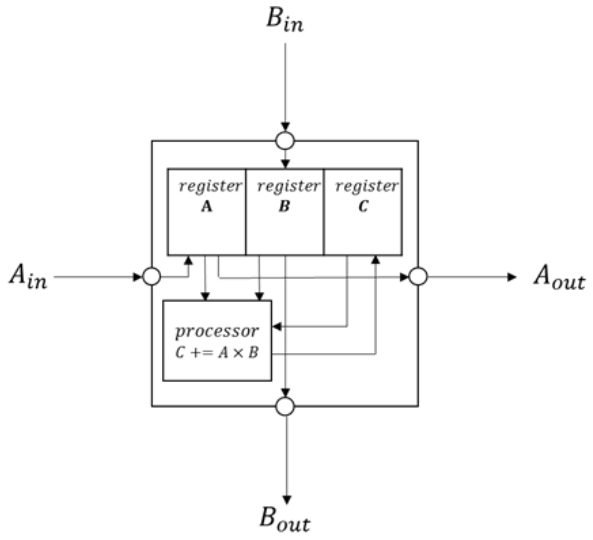

図 3.二次元の座標の場合のセル

以上の二次元の座標上にセルをつないだネットワークに対して時刻ごとに境界のセルに入力を与えることで行列積を求めることができる.このとき,時刻ごとに必要な入力行列 A,B の要素を与えるための配置,およびネットワークの接続関係を 3×3 の行列積を例に以下の図 4 に示す.

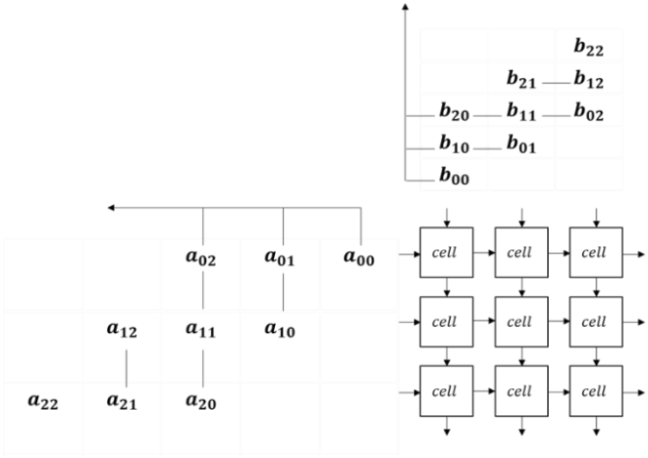

図 4. 3×3 の行列積を計算する場合の二次元の座標のセルの接続関係および入力行列 A,B の配置

5. GPU におけるシストリックアルゴリズムの実現方法

本章では GPU におけるシストリックアルゴリズムの実現方法を示し,ここでは例として行列積の計算を行う.また,ベンチマークとして現在 GPU 上で行列積を求める最速のアルゴリズムである 3 重ループを用いたアルゴリズムに対しても当研究室で開発した理論モデルを用いて理論評価を行う.

まず,GPU 上でのシストリックアルゴリズムの実現方法を考えると,大きく 2 種類の実現方法が考えられる.1 つ目が GPU 上のメモリをグローバルメモリのみを使用する方法である.この方法では,演算に必要な入力行列 A,B がグローバルメモリ上の配列に格納されており,一時的な計算結果の保存もグローバルメモリ上の配列に格納する実現方法である.最終的な結果である出力行列 C はグローバルメモリ上の配列に出力する.

2 つ目の方法が GPU 上のメモリであるグローバルメモリに加え,シェアードメモリを使用する方法である.この方法でも同様に, 演算に必要な入力行列 A,B がグローバルメモリ上の配列に格納されている.そして,一時的な入力行列 A,B の格納場所および一時的な計算結果の保存にシェアードメモリを使用する.グローバルメモリのみを使用する実現方法と同様に, 最終的な結果である出力行列 C を

グローバルメモリ上の配列に出力する.

以上のことより,シストリックアルゴリズムに対して2種類の実現方法があることを述べた.さらに,シストリックアルゴリズムの場合には複数時刻の計算をまとめて行なう実現方法も考案したので,本章で示す.

また,シストリックアルゴリズムでは行列積を連続で計算する場合,パイプラインで計算を行なうことができる特徴がある.この特徴の GPU 上での実現方法はスレッド毎に割り当てられた座標を用いて入力行列が切り替わる時刻で書き込み先を切り替えることで実現できる.また,パイプラインを用いて連続で行列積を計算することで 1 回行列積を計算することと比較し,全てのセルが計算を行う時刻を増やすことができるので効率よく計算を行うことができる.

このような特徴を踏まえて,各アルゴリズムに対してサイズ N×N の行列積を 1 回計算する場合,S 回連続で計算する場合の 2 通りの評価を行なう.本紀要では,項数に限りがあるので,以下では二次元の座標のシストリックアルゴリズムのグローバルメモリを使用する方法,シェアードメモリを使用する方法について述べる.その他のアルゴリズムに関しては修士論文にて述べる.

5.1. グローバルメモリを使用した 1 時刻ごとに計算を行うシストリックアルゴリズム

グローバルメモリを使用した 1 時刻ごとに計算を行うシストリックアルゴリズムでは,グローバルメモリ上の配列に格納された入力行列 A,B,出力行列 C に対して,出力行列 C の要素数と同数のスレッドを用いて演算を行なう.このとき,スレッドの座標は CUDA 上の blockDim,blockIdx,および threadIdx を用いて一意に決定することができ,シストリックアルゴリズムのセルの座標とスレッドの座標がそれぞれ対応している.このスレッドがシストリックアルゴリズムの各セル内で行なう演算を行なう.

シストリックアルゴリズムでは時刻ごとにそれぞれのセルで必要な入力行列 A,B の要素が分かるので,時刻ごとにスレッドが演算に必要なグローバルメモリ上の配列に格納された入力行列 A,B にアクセスし読み込みを行い,その要素ごとの積算を行い,グローバルメモリ上の配列に格納されている出力行列 C に足しこみを行なう.以上の操作を入力行列 A,B がネットワーク上を全て通過する時刻数(二次元:3N-2)行なうことで出力行列 C を求めることができる.以下にグローバルメモリを使用したシストリックアルゴリズムの理論モデルを用いた評価を示す.

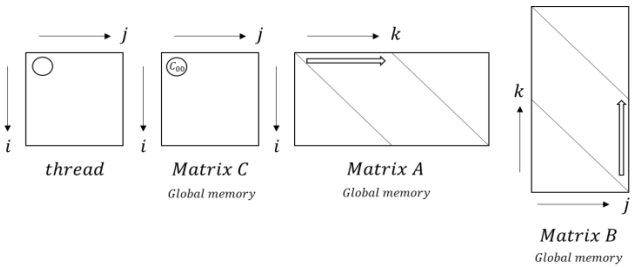

図 4.グローバルメモリを使用した 1 時刻ごとに計算を行うシストリックアルゴリズム

$$\left\{\frac{2N^2}{wk}\times Lgm+\frac{2N^2}{wk}\times Lgm\right\}\times\{3N-2+(S-1)(2N-1)\}$$

$$=\frac{4(2S+1)N^3-4(S+1)N^2}{wk}\times Lgm$$

数式1.S回計算する場合のグローバルメモリを使用した1時刻ごとに計算を行うシストリックアルゴリズムの理論評価

5.2. グローバルメモリを使用した複数時刻を同時に計算するシストリックアルゴリズム

グローバルメモリを使用した複数時刻を同時に計算するシストリックアルゴリズムでは,グローバルメモリ上の配列に格納された入力行列 A,B,出力行列 C に対して,出力行列 C の要素数と同数のスレッドを用いて演算を行なう.このアルゴリズムでも同様に,スレッドの座標は CUDA 上の blockDim,blockIdx,および threadIdx を用いて一意に決定することができ,シストリックアルゴリズムのセルの座標とスレッドの座標がそれぞれ対応している.

入力行列の要素の読み込みは,1 時刻ごとに計算を行うシストリックアルゴリズムと同様に行い要素ごとの積算を行う.その後,スレッド毎のレジスタに積算の結果の足しこみを行い,t 時刻毎にグローバルメモリ上の配列に格納された出力行列 C に足しこみを行なう.以上の操作を入力行列 A,B がネットワーク上を全て通過する$\frac{\text{時刻数}}{t}$(二次元:3N-2)行なうことで出力行列 C を求めることができる.以下でこのアルゴリズムの理論モデルを用いた評価を示す.

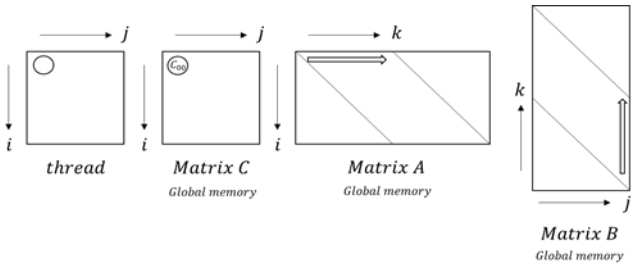

図 5.グローバルメモリを使用した複数時刻を同時に計算するシストリックアルゴリズム

$$\left\{\frac{2tN^2}{wk}\times Lgm+\frac{2N^2}{wk}\times Lgm\right\}\times\frac{3N-2+(S-1)(2N-1)}{t}$$

$$=\frac{\{(4S+2)t+(4S+2)\}N^3-\{2(S+1)t+2(S+1)\}N^2}{wk}\times\frac{1}{t}$$

$$\times Lgm$$

数式 2.S 回計算する場合のグローバルメモリを使用した複数時刻を同時に計算するシストリックアルゴリズムの理論評価

5.3. シェアードメモリを使用した 1 時刻ごとに計算を行うシストリックアルゴリズム

シェアードメモリを使用したシストリックアルゴリズムでは,グローバルメモリ上の配列に格納された入力行列 A,B,出力行列 C に対して,出力行列 C の要素数と同数のスレッドを用いて演算を行なう.このアルゴリズムでも同様に,スレッドの座標は CUDA 上の blockDim,blockIdx,および threadIdx を用いて一意に決定することができ,シストリックアルゴリズムのセルの座標とスレッドの座標がそれぞれ対応している.

シストリックアルゴリズムは時刻ごとにネットワークに入力される入力行列 A,B の要素が分かるので,各ブロック内のスレッドがその時刻に入力されるグローバルメモリ上の配列に格納された入力行列 A,B の要素にアクセスを行う.シェアードメモリを使用したアルゴリズムの場合は境界からセルに入力される入力行列およびブロックをまたいでアクセスが必要な入力行列要素へのアクセスのみとなる.グローバルメモリから読み込みを行った後,あらかじめシェアードメモリ上に確保されている一時的に入力行列 A,B の要素を格納するための配列 A`,B`に格納する.この操作は時刻ごとにシェアードメモリ上に格納されている配列 A`,B`に格納を行なうが, 格納する配列の場所は時刻ごとに配列 A`は行方向,B`は列方向に循環シフトしていく.そしてこの操作により,各時刻にそれぞれのスレッドが演算を行なうべき入力行列の要素がシェアードメモリ上の配列 A`,B`に格納されたので,ブロック内の各スレッドはシェアードメモリ上の配列に格納された入力行列 A,B にアクセスを行い,要素同士の積算を行なう.その後,積算の結果をシェアードメモリ上の配列に格納されている結果を一時的に保存するための配列 C`に足しこみを行なう.以上の操作を入力行列 A,B がネットワーク上を全て通過する時刻数(二次元:3N-2)行なうことで出力行列 C を求めることができる. 以下にシェアードメモリを使用したシストリックアルゴリズムの理論評価を示す.

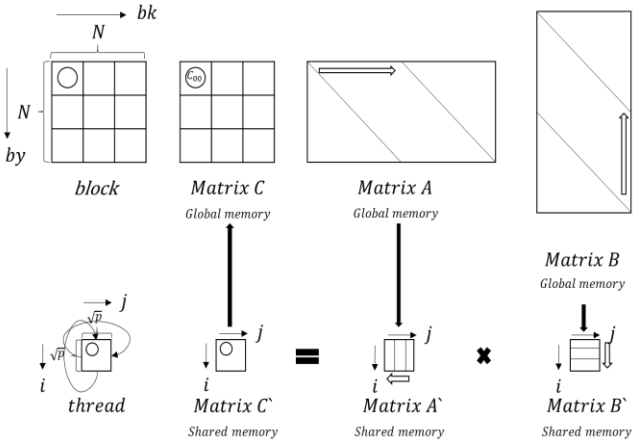

図 6.シェアードメモリを使用した 1 時刻ごとに計算を行うシストリックアルゴリズム

$$\left\{\frac{2N^2}{wk\sqrt{p}}\times Lgm+\frac{2\sqrt{p}}{w}\times Lsm+\frac{2p}{w}\times Lsm+\frac{2p}{w}\times Lsm\right\}$$

$$\times\{3N-2+(S-1)(2N-1)\}+\frac{2N^2}{wk}\times Lgm$$

$$=\frac{(4S+2)N^3+(2\sqrt{p}-2S-2)N^2}{wk\sqrt{p}}\times Lgm$$

$$+\frac{\left(4(2p+\sqrt{p})S+2(2p+\sqrt{p})\right)N-2\left((2p+\sqrt{p})S+(2p+\sqrt{p})\right)}{w}$$

$$\times Lsm$$

数式 3.S 回計算する場合のシェアードメモリを使用した 1 時刻ごとに計算を行うシストリックアルゴリズムの理論評価

5.4. シェアードメモリを使用した複数時刻を同時に計算するシストリックアルゴリズム

シェアードメモリを使用した複数時刻を同時に計算するシストリックアルゴリズムでは,グローバルメモリ上の配列に格納された入力行列 A,B,出力行列 C に対して,出力行列 C の要素数と同数のスレッドを用いて演算を行なう.このアルゴリズムでも同様に,スレッドの座標は CUDA 上の blockIdx,および threadIdx を用いて一意に決定することができ,シストリックアルゴリズムのセルの座標とスレッドの座標がそれぞれ対応している.

シストリックアルゴリズムは時刻ごとにネットワークに入力される入力行列 A,B の要素が分かるので,各ブロック内のスレッドがその時刻に入力されるグローバルメモリ上の配列に格納された入力行列 A,B の要素にアクセスを行い,あらかじめシェアードメモリ上に確保されている一時的に入力行列 A,B の要素を格納するための配列 A`,B`に格納する.このアルゴリズムでは, t 時刻分の計算をまとめて行なうために,t 時刻分の計算に必要なグローバルメモリ上の配列に格納された入力行列 A,B の要素へアクセスし読み込みを行い, あらかじめシェアードメモリ上に確保されている一時的に入力行列 A,B の要素を格納するための配列 A`,B`に格納する.この操作は時刻 t ごとにシェアードメモリ上に格納されている配列 A`,B`に格納を行なうが,格納する配列の場所は時刻ごとに配列 A`は行方向,B`は列方向に循環シフトしていく.そしてこの操作により,t 時刻分のスレッドが演算を行なうべき入力行列の要素がシェアードメモリ上の配列 A`,B`に格納されたので,ブロック内の各スレッドはシェアードメモリ上の配列に格納された入力行列 A,B にアクセスを行い,t 時刻分の要素同士の積算を行なう.その後,積算の結果をシェアードメモリ上の配列に格納されている結果を一時的に保存するための配列 C`に足しこみを行なう.以上の操作を入力行列 A,B がネットワーク上を全て通過する$\frac{\text{時刻数}}{t}$(二次元:3N-2)行なうことで出力行列 C を求めることができる.以下にシェアードメモリを使用した複数時刻を同時に計算するシストリックアルゴリズムの理論評価を示す.

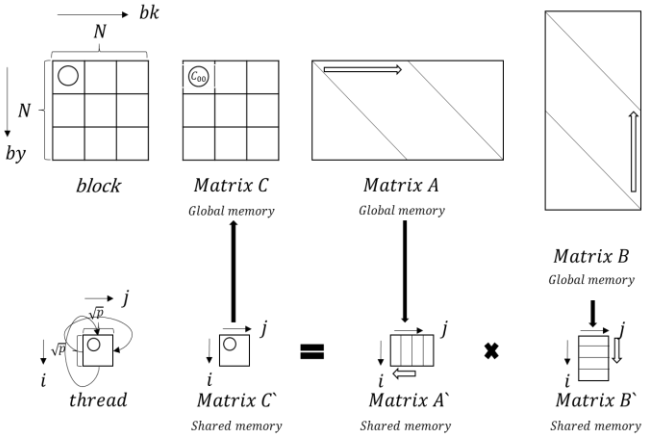

図 7.シェアードメモリを使用した複数時刻を同時に計算するシストリックアルゴリズム

$$\left\{\frac{2tN^2}{wk\sqrt{p}}\times Lgm+\frac{2t\sqrt{p}}{w}\times Lsm+\frac{2tp}{w}\times Lsm+\frac{2p}{w}\times Lsm\right\}$$

$$\times\frac{\{3N-2+(S-1)(2N-1)\}}{t}+\frac{2N^2}{wk}\times Lgm$$

$$=\frac{(4S+2)N^3+(2\sqrt{p}-2S-2)N^2}{wk\sqrt{p}}\times Lgm$$

$$+\left\{\frac{\left(4(p+\sqrt{p})S+2(p+\sqrt{p})\right)N-2\left((p+\sqrt{p})S+(p+\sqrt{p})\right)}{w}\right.$$

$$\left.+\frac{(4pS+2p)N-2(pS+p)}{w}\times\frac{1}{t}\right\}\times Lsm$$

数式 4. 行列積を S 回計算する場合のシェアードメモリを使用した複数時刻を同時に計算する二次元の座標のシストリックアルゴリズムの理論評価

また,最後に 3 重ループを用いたアルゴリズムのグローバルメモリを使用したアルゴリズムとシェアードメモリを使用したアルゴリズムの理論評価を以下に示す.

$$\frac{4SN^3}{wk}\times Lgm$$

数式 5.行列積を S 回計算する場合のグローバルメモリを使用した 3 重ループを用いたアルゴリズムの理論評価

$$\frac{2SN^3+2\sqrt{p}N^2}{wk\sqrt{p}}\times Lgm+\frac{6\sqrt{p}SN}{w}\times Lsm$$

数式 6.行列積を S 回計算する場合のシェアードメモリを使用した 3 重ループを用いたアルゴリズムの理論評価

6. 結果と考察

まず，今回の実験環境のパラメータの値の詳細を表 2 に，理論評価を行い求めた式に値を代入し作成したグラフを，図 11 に行列積を 1 個求める場合のグローバルメモリを用いたアルゴリズムを,図 12 に行列積を 1 個求める場合のシェアードメモリを用いたアルゴリズムを示す.

表 2. 実験環境のパラメータの値の詳細

スレッド数	$p = 256$
バンク内のメモリ数	$k = 16$
MP数	$x = \frac{N^2}{256}$
グローバルメモリへのアクセス時間	$Lgm = 0.000000597372$
シェアードメモリへのアクセス時間	$Lsm = 0.00000000477897$
ワープ内のスレッド数	$w = 16$
要素数	N

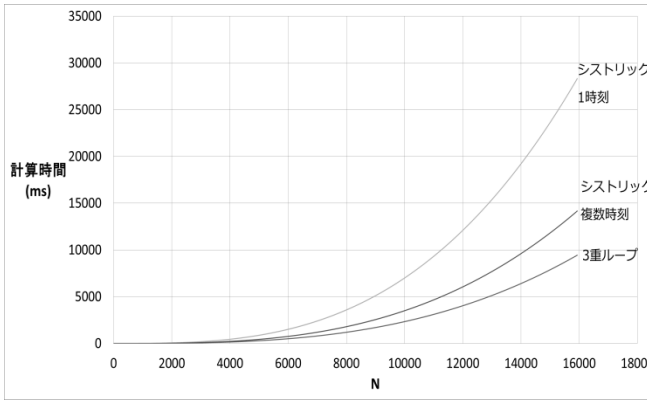

図 8. 行列積を 1 回求める場合のグローバルメモリを用いたアルゴリズムの理論評価

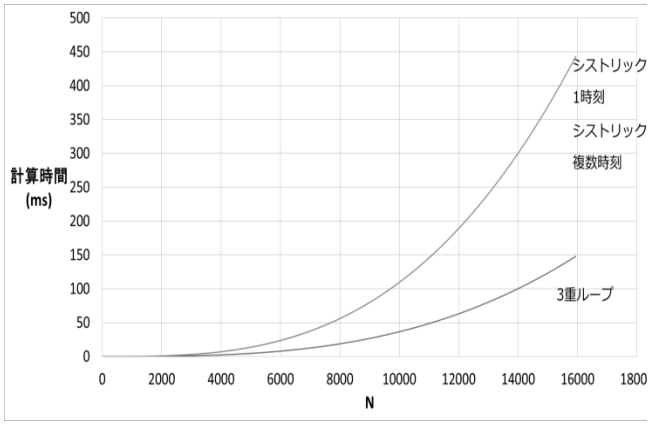

図 9. 行列積を 1 回求める場合のシェアードメモリを用いたアルゴリズムの理論評価

次に,実装結果を示す.図 13 にグローバルメモリを用いた両アルゴリズムおよびシェアードメモリを用いた 3 重ループのアルゴリズムの実装結果を示す.

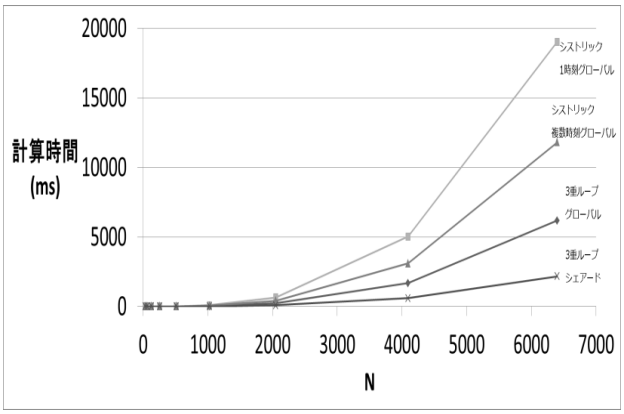

図 10. グローバルメモリを用いた両アルゴリズムおよびシェアードメモリを用いた 3 重ループのアルゴリズムの実装結果

次に,考察を行なう．まず,アルゴリズム同士の比較を行なうと,シストリックアルゴリズムではN^2の平面を二次$3N-2$回計算しているのに対し,3 重ループを用いたアルゴリズムではN^2の平面をN回計算していることになる.

次に,グローバルメモリを使用した 1 時刻ごとに計算を行うシストリックアルゴリズムでは,時刻毎にグローバルメモリ上の配列に格納された入力行列にアクセスを行なうことにより,アルゴリズム同士の比較で述べたように,入力行列の読み込みを行なう際に発生するグローバルメモリへのアクセスが約 3 倍あり,同様に結果の足しこみを行なう際に発生するグローバルメモリへのアクセスも約 3 倍あることが,ベンチマークである 3 重ループのアルゴリズムと比較し,計算時間が約 3 倍の結果となってしまった原因である.

そこで,改良策として,グローバルメモリを使用した複数時刻を同時に計算するシストリックアルゴリズムでは時刻毎に演算した結果をレジスタに足しこむことで,結果の足しこみの際に発生するグローバルメモリへのアクセスを減らすことができ,理論評価では,S 個連続で行列積を計算する場合ではほぼ等速で計算をすることができることが分かった.

また,シェアードメモリを使用した 1 時刻ごとに計算を行うシストリックアルゴリズムでは,時刻毎にグローバルメ

モリ上の配列に格納された入力行列のその時刻に必要な要素にアクセスを行なうことにより,グローバルメモリを用いたシストリックアルゴリズムよりはグローバルメモリへのアクセス回数は減ったものの,3 重ループのアルゴリズムと比較し,計算時間は約 3 倍の結果となった.

さらに,シェアードメモリを使用した複数時刻を同時に計算するシストリックアルゴリズムではステップ毎に演算した結果をレジスタに足しこむことで,シェアードメモリへのアクセスを減らすことができたが,理論評価では,1 時刻ごとに計算するアルゴリズムとほぼ同じ計算時間となってしまった.グローバルメモリ,シェアードメモリを用いたシストリックアルゴリズムに対しては,グローバルメモリ上の配列に格納された入力行列へのアクセスを wor-p_shuffle 関数を用いて減らすなどの改良が可能であると考えられる.

7. むすび

本研究では GPU を用いて,シストリックアルゴリズムの実装方法を提案し,例として,行列積の演算を行なった.理論評価の結果としては行列積を S 回計算する場合のグローバルメモリを用いた二次元のシストリックアルゴリズムではベンチマークである行列積を S 回計算する場合のグローバルメモリを用いた 3 重ループを使用したアルゴリズムとほぼ同じ速度で計算することができることが分かった.

今後は課題となっている入力行列の読み込み部分の効率を改善することである.

謝辞

本研究の機会を与えてくださり,ご指導を頂きました和田幸一教授ならびに大阪府立大学藤本典幸教授に深く感謝致します.また,本研究にあたりいろいろな面での貴重なご意見やご指導頂きました本研究室の皆様,先輩方に深く感謝致します.

参考文献

[1]青木尊之,額田彰 ”はじめての CUDA プログラミング” 光学社(2009)

[2]K Nakano “An Optimal Parallel Prefix-sums Algorithm on the Memory Machine Model for GPUs”, 電子情報通信学会信学技報,pp. 99-113, (2012)

[3]鈴木,遠藤,和田:GPGPU に対する理論モデル,2013 電子情報通信学会総合大会,
ISS 特別企画, 学生ポスターセッション,DK-1 (2013-03)

[4]松本: GPGPU に適した並列理論モデルの提案,法政大学理工学部応用情報工学科卒業論文(2014-03)

[5]上林弥彦,岡部寿男,浜口清治,武永康彦 “プレパラータ先生の超並列計算講義” 共立出版(1996)

学会発表

茂木,和田,藤本:“GPGPU におけるシストリックアルゴリズムの効率的な実現について”,第 11 回情報科学ワークショップ予稿集, pp.129-154,(2015-09)