大規模配列の GPU 高速処理プログラミングを支援する開発環境

1. 背景

GPU とは Graphics Processing Unit の略語であり、コンピュータシステムにおいて描画処理を担当する装置である。近年の GPU は、3D 描画などの莫大な規模の演算量に応じて、多数コアによる並列処理を導入しており、CPU より高い並列処理能力を持っている。この並列処理能力を、描画処理以外の計算資源として活かそうとするのが GPGPU(General Purpose GPU)である。近年、CUDA 等の GPGPU 言語が登場するなど、徐々に GPU を計算資源として利用する動きが活発になりつつある。

しかし、未だ GPGPU プログラミングは初心者にとって難しいものである。最初に、プログラマは通常のプログラミングで意識する必要のなかったメモリを CPU 側と GPU 側の 2 つに分離して認識する必要が生じる。また、現在の GPGPU 言語にはオブジェクト型のよう抽象度の高い型が存在しておらず、処理したいデータを CPU 側でプリミティブな型の配列の形に変換して、それを GPU 側に転送する煩雑なことをしなければならない。

次に、GPGPU 言語のプログラミング概念に適応することが難点である。GPGPU 言語で GPU 処理を記述するにも、CPU 側と GPU 側の処理両方を記述しなければならない。しかも、GPU 内の配列の計算を担当するカーネルは、並列処理プログラミングに基づいているが、こういった並列処理プログラミングは未だ一般に広まっている訳でもなく、大半の場合は概念に適応するのが要求される。

最後に、大規模配列を扱うプログラミングのデバッグ問題がある。GPU で扱うデータの配列はその規模が大きい上、単なる数値の配列であるため、通常のテキストベースのデバッガやテキスト出力（所謂 printf デバッグ）ではデバッグが難しい。

以上のことは正しく動作する GPU プログラムが出来上がるまでの時間を遅らせる要因となり、初心者への入門障壁となっている。

2. 目的

本プロジェクトでは、GPU プログラミングをする過程で必要でありながらも、実質的に GPU 処理の本質に関わらない部分はなるべく簡略に書け、GPGPU 初心者のプログラマが GPU 内の処理部であるカーネルの記述だけに集中させるシステムを開発することを目的とする。

3. 開発の内容

nVidia 社の GPGPU 言語である CUDA を対象としたビジュアルインタプリタを開発した。図 1 はこのソフトウェアを利用した GPU プログラミングのワークフローを表している。

GPGPU プログラムは GPU 内の並列処理を行う GPU コードと CPU 側で制御を行う CPU コードに分かれる。従って、入力においても同様にそれぞれのハードウェアに対し

て異なるコードの入力を必要とする。GPU コードに関しては、実際 GPU 内での並列処理命令であり、ビジュアルインタプリタで呼び出すために必要な情報を加えて実際バイナリコードとしてコンパイルさせ、ビジュアルインタプリタ側でそのバイナリを呼び出す。CPU コードに関しては、閲覧や修正が容易になるように C 言語の構文規則に従いながら、ファイル入出力や可視化コードなど、GPU 処理に本質的に関係はないが、手続きとして必要な部分を簡略に記述できるような疑似コードの仕様になっている。GPU コードの各々のカーネルは、この CPU コード上で個別の関数として呼び出される。

デバッグの際に、プログラマは自分の書いた CPU 側のコードをビジュアルインタプリタ上で実行させることによって、書いたプログラムの挙動を観察することが出来る。この挙動観察の結果、もし問題になる処理が発見されたら、入力コードの修正を行った上、再びデバッグに入る。

この過程を経て、プログラムが意図する挙動を示すようになったら、ビジュアルインタプリタが補間していた簡略化部分を埋めた CPU コードをエクスポートさせる。この CPU コードと入力に用いた GPU コードとを合わせてコンパイルすることで、実際の GPU 上で動作可能なプログラムが完成する。

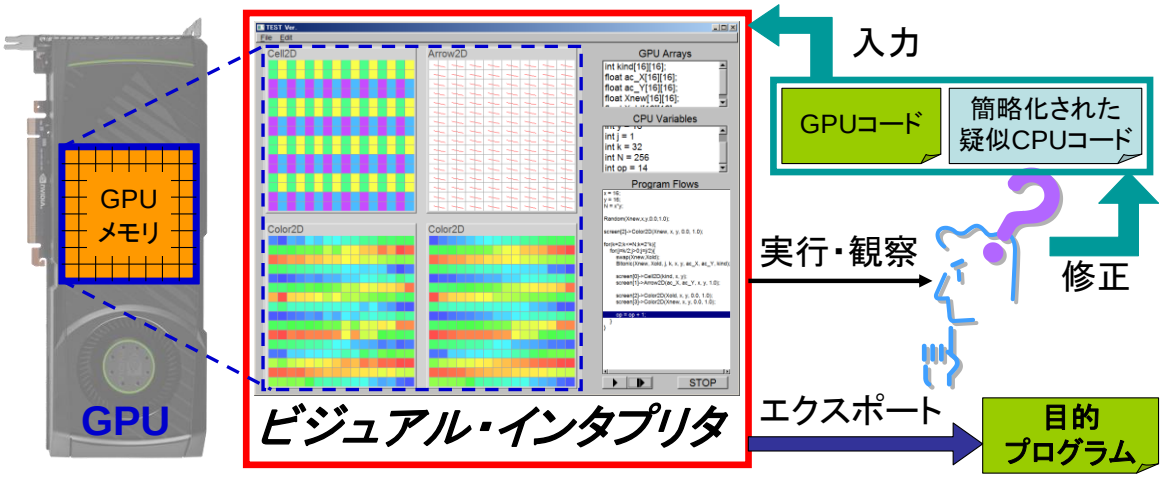

図 1. 本ソフトウェアのワークフロー

4. 従来の技術（または機能）との相違

4.1 一覧性のよいプログラム観察

GPGPU プログラムでは CPU 側のプログラムが GPU 処理の全体流れを制御することになっている。本ソフトウェアでは、その CPU 側のプログラムを把握しやすいように簡略化されたコードとして入力、表示される。特に、ファイルと GPU 配列間の入出力、可視化処理と言った定型的な処理を 1 行で処理できるようなインタプリタを用意している。

デバッグの際には、図 2 のように GPU 側の配列の型とサイズが「GPU Arrays」部で表示され、CPU 側の変数も同様に「CPU Variables」部で型と値が表示される。ま

た、現在プログラムの実行位置が「Program Flows」部で反転として表示される。図 2 では、左の状態の for 文の実行によって整数型変数 j の値が 4 から右の図の 2 に変わり、プログラムの実行位置は次の行に移っていることを示している。この仕組みによって、プログラマは自分の書いたプログラムを実行させながら観察することができ、CPU 側の制御や変数指定によるプログラムのバグを検出することも可能になる。

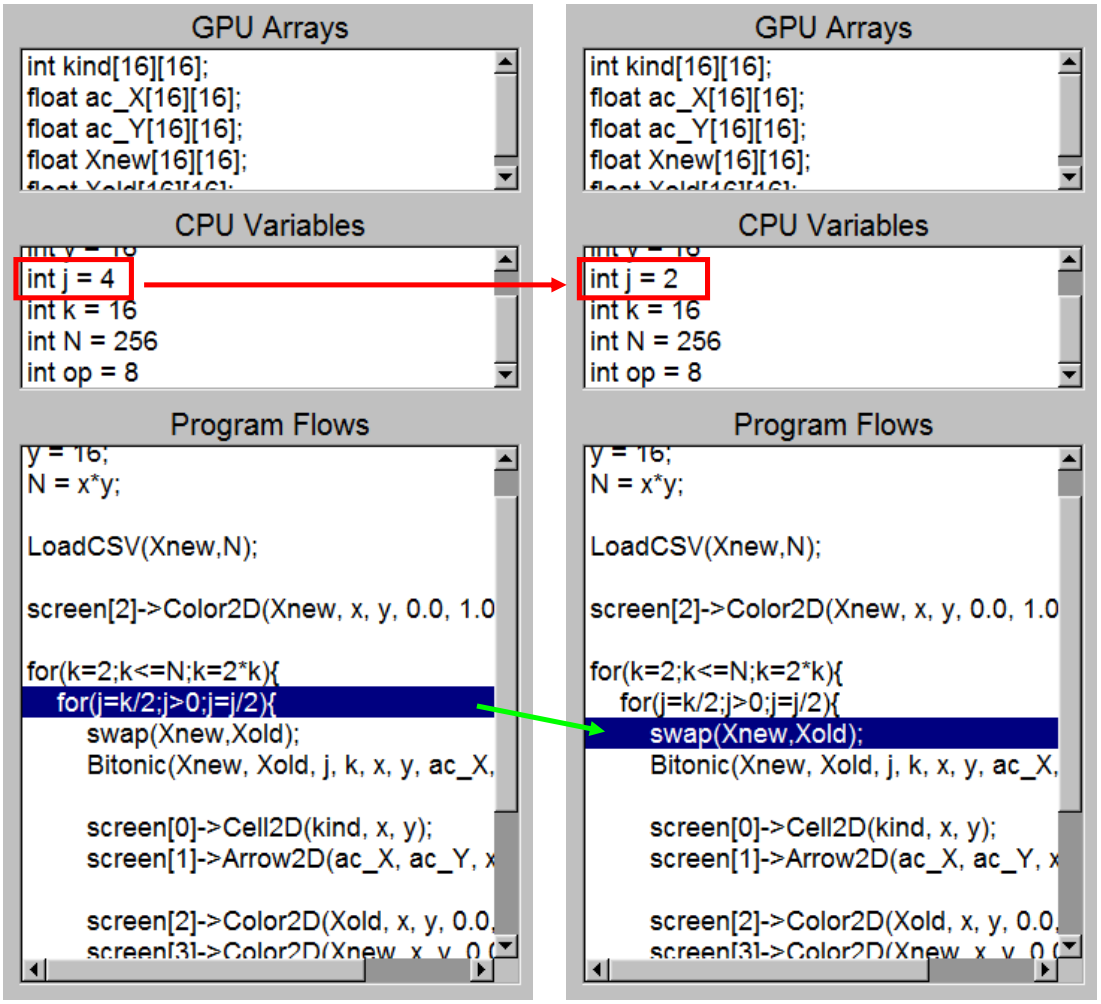

図 2. ビジュアルインタプリタの実行によって変化するプログラム状態の例

4.2　可視化を導入したプログラミング仕様

本システムは GPU が特性として配列を処理することが主になっているため、OpenGL を利用した配列の可視化機能を備えている。本システム上に可視化をさせるためには、簡略化された CPU コード側に可視化手法とパラメータを指定する必要がある。これは目的の処理自体には無関係なコードではあるが、所謂 printf デバッグの要領でこのコードを挿入することが可能なため、プログラマにとっては自然な手順と言える。図 3 は点群（図 3 左）を離散ボロノイ図（図 3 右）にする際に各々の格子点で最も近い母点の位置（図 3 中）を可視化している。この各々の図は、それぞれ 1 構文のみで可視化が可能となっている。

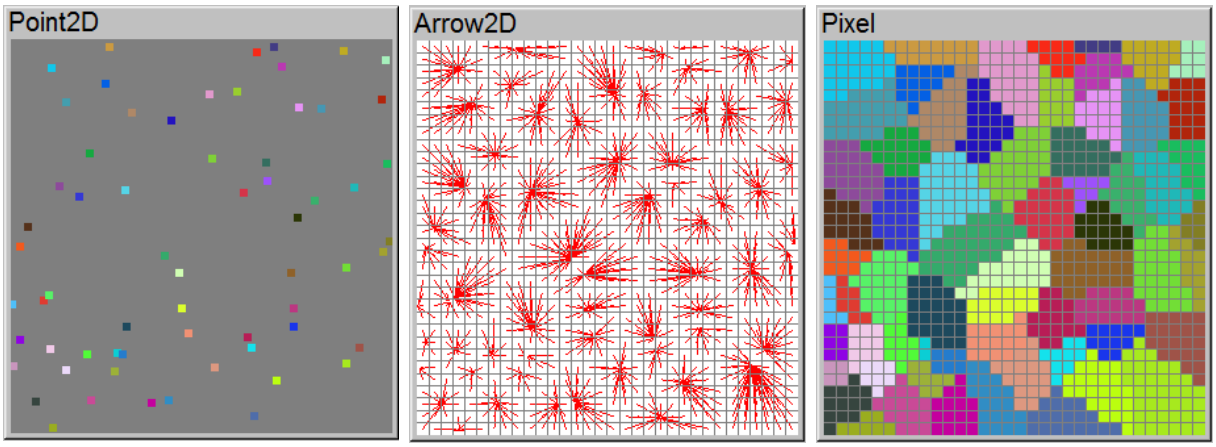

図 3. 離散ボロノイズの計算においての可視化

4.3　実用的な経験蓄積が可能

教育用の標語を立てているシステムがよく陥る問題として、教育対象をあまりにも単純化してしまった結果、システムを使っても実用的な経験の蓄積には役立たなかったり、対応可能な事例があまりにも限定的で、それ以外の広範囲の事例に対しては適用できなくなったりする問題がある。本ソフトウェアでは、GPU 処理部分に関してはユーザに実際の GPU コードを書かせる経験を与えているため、ユーザに実用的な経験を与えていると言える。

4.4　プロトタイピングツール

本システムは GPGPU プログラミング用のプロトタイピングツールでもあるため、動くプログラムを得るまでの時間が長い GPU プログラムの特性上、これは上級者にとってもメリットがあると言える。

5. 期待される効果

デスクトップはもちろん、ラップトップにも GPGPU が当たり前のように搭載されている現状にも関わらず、GPU プログラミング自体の難しさから、未だ GPU を計算資源として使いこなしているプログラムは希である。本システムを用いることで GPGPU がプログラム開発分野全般に広まることになり、この眠っている計算資源を有効に活用できるようになる事が期待できる。

6. 普及（または活用）の見通し

本環境を用いた GPU プログラミングのチュートリアルや試演が必要だと思われる。近日中、そういったドキュメンテーションを備えたウェブページを公開する予定である。

また、このソフトウェアを実際の教育現場などで活用させ、有効性の検証と共に論文執筆などで広報した上、オープンソースの形で公開する。

7. クリエータ名（所属）

盧　承鐸（東京大学大学院情報理工学系研究科コンピュータ科学専攻）