これからの並列計算のためのGPGPU連載講座(IV)
特別編 CUDAプログラミング Windows編

大 島 聡 史
東京大学情報基盤センター

1 はじめに

第二回および第三回の連載ではLinuxを対象としてCUDAプログラミングの流れや最適化方法を紹介してきた。しかし、CUDAに対応したGPUを搭載した計算機としてはWindowsがインストールされたPCのみを所有している利用者も多いようなので、今回はWindows向けのCUDAプログラミングについて説明する。

なお、本記事の内容はWindows7(64bit)での利用を中心としている。他のWindowsを利用する場合には適宜パスの読み替えなどを行っていただきたい。

2 CUIベースのCUDAプログラム開発環境

まずはLinuxを用いる場合と同様にコマンドラインで開発を行う手順を解説する。この方法はインストール手順こそWindowsに特化した内容ではあるものの、実際の開発手順はLinuxとほぼ同様のシンプルな方法である。

2.1 開発環境の導入

Windowsにおける開発環境構築方法は、すでに第二回の連載で述べたように、基本的にはLinuxと同様である。すなわち、まずはCUDA DriverとCUDA toolkitおよびCUDA SDKをインストールする。これらは全てNVIDIAのサイトにてインストーラ(exeファイル)の形式で配布されており、ウィザードに従って処理を進めるだけでインストール処理は完了する。

WindowsにおけるCUDA開発にはVisual Studioが必要である。本稿執筆時点の最新開発環境であるCUDA toolkit 3.1およびCUDA SDK 3.1ではVisual Studio 2005および2008に対応していることが記されている。筆者はCUDA toolkit/SDK 3.0もしくはCUDA toolkit/SDK 3.1とVisual Studio 2008 Professional Editionの組み合わせにて動作を確認している。

2.2 開発の手順

Windowsにおける開発の手順としては、Linuxと同様にnvccプログラムを実行してコンパイル・リンクを行えば良い。ただし、Visual Studioのコンパイラ本体であるcl.exe等へのパスが設定されていない(環境変数PATHが設定されておらず、コマンドプロンプト上でcl.exeを実行するにはフルパス指定が必要な状態) の場合にはパスを指定する必要がある。

図1にCUDAプログラムをコンパイル・実行する例を示す。ちなみに図中で入力しているコマンドは以下の通りである:

```
>nvcc -o chkdevice chkdevice.cu --compiler-bindir ‘‘C:\Program Files (x86)\Micr
 osoft Visual Studio 9.0\VC\bin’’ -I ‘‘C:\Program Files (x86)\Microsoft Visual
```

```
Studio 9.0\VC\include'' -I ''C:\ProgramData\NVIDIA Corporation\NVIDIA GPU Comp
uting SDK\C\common\inc'' -L ''C:\Program Files (x86)\Microsoft Visual Studio 9
.0\VC\lib'' -L ''C:\ProgramData\NVIDIA Corporation\NVIDIA GPU Computing SDK\C\
common\lib'' -lcutil32 -Xcompiler ''/MTd'' --machine 32
```

対象プログラムの中身は、システムに存在するCUDA対応デバイスの列挙をするだけのシンプルなプログラムである。この例ではcl.exeへのパスが設定されていない状況を想定して–compiler-bindirオプションを指定している。また、32bitのコード生成でcutilを使う想定をしているため-lcutil32を指定しているが、64bitのWindowsに64bitのCUDA toolkit/SDKを導入した場合、cutil64.libやcutil64.dllは存在するがcutil32.libやcutil32.dllは存在しないため、事前に用意しておく必要がある。これらを用意するには、`C:\ProgramData\NVIDIA Corporation\NVIDIA GPU Computing SDK\C\common`にあるcutil_vc90.slnをVisual Studioで開いてビルドすれば良い。

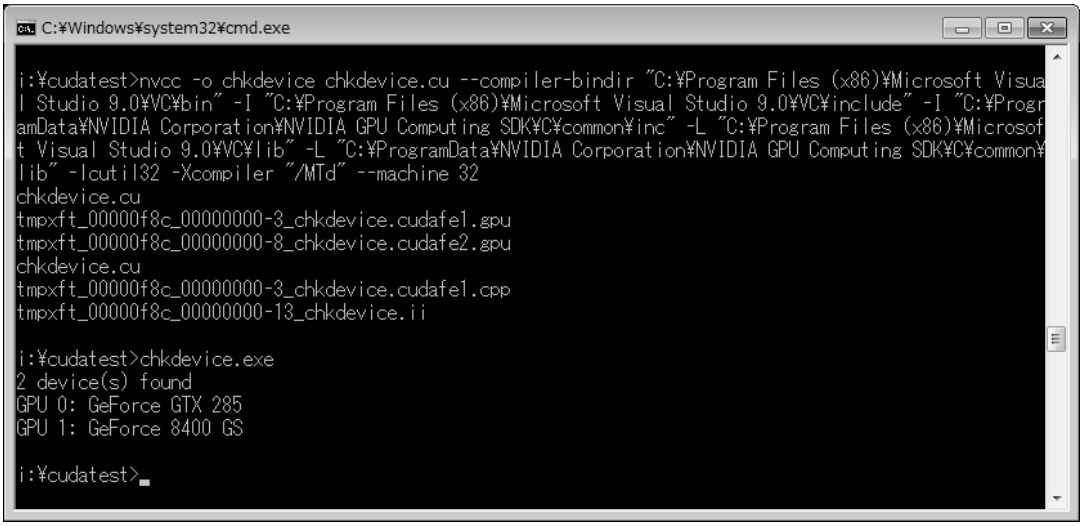

図 1　プログラムのコンパイルおよび実行の例

CUIでの開発については以上の通りである。Linux上での開発と比べると、nvccに指定するオプションに若干の違い(ほとんどのオプションは同じように利用できる)やシェル(コマンドプロンプト)の機能差による使い勝手の違いはあるが、基本的に同じように開発できることがわかっていただけただろう。

3　GUIベースのCUDAプログラム開発環境

続いて現在ベータ公開されている[1]開発環境NVIDIA Parallel Nsight(以下Nsightと記す)について解説する。

3.1　NVIDIA Parallel Nsight の特徴

NsightはVisual Studioに対するアドオンとして提供されている開発環境である。Nsightの主な機能(特徴)は以下の通りである。

- Visual Studioと統合されたGUIインタフェース
- GPU実機を用いた高速なデバッグ実行
- GPUカーネル内のステップ実行(トレースデバッグ)が可能
- プロファイラ等の機能も統合されている
- グラフィックス開発用の機能も搭載
- 有償版ではより強力なボトルネック解析機能等が提供される

Nsightはこのように様々な機能を提供しており、本稿で全てを解説することは困難である。よって、今回は導入手順とGPUカーネル内のステップ実行方法を中心に基本的な使い方を解説する。なお、詳細な手順はNsight(Host)をインストールすると閲覧できるようになるNVIDIA Parallel Nsight User Guideにて解説されているので、あわせてご覧戴きたい。

3.2 Nsight の導入

まずはNsightの入手方法についてであるが、Nsightを入手するためには無料登録可能なBeta accountを入手する必要がある。Beta accountを入手すると、HostとMonitorの2つのプログラムを入手できるようになる。HostはVisual Studioと統合される開発環境、Monitorは実際にGPUへアクセスするプログラムであり、1台のPCで開発を行う場合(ローカルデバッグ)にはHostとMonitorの両方をインストールする。また開発環境とデバッグに使うCUDA対応GPUを別のPCにする場合(リモートデバッグ)には、開発環境にHostを、CUDA対応GPUが搭載されたPCにMonitorをインストールする。

なおローカルデバッグを行うためには、デバッグ対象となるCUDA対応GPU以外にもう1つGPUが存在する必要がある。また、起動時にディスプレイと接続されていないGPUはWindowsから認識されないため注意しなくてはならない。

Nsight(Host,Monitor)のインストール後は、さらに以下の手順にそって開発の準備を行う必要がある。Monitorの設定はNsight Monitor起動後にタスクトレイに格納されるアイコンの右クリックメニューから、Hostの設定はVisual Studioに追加されているメニュー(図2)から行うことができる。

1. TDRの無効化
   Monitorのオプションから“WDDM TDR enabled”をFalseに設定して再起動する。これにより、ディスプレイドライバが2秒以上応答しなかった場合にWindowsによってプロセスが終了させられるという問題(Windowsの仕様)を回避することができる。
2. WPFのD3Dアクセラレーションの無効化(ローカルデバッグのみ)

   ```
   C:\Program Files (x86)\NVIDIA Parallel Nsight 1.0\Common\
   DisableWpfHardwareAcceleration.reg
   ```

   を実行して再起動する。
3. Aeroの無効化
   Monitorが動作するPCではAeroを無効化しておかねばならない。
4. Nsightの認証

HostのNsightオプションに認証のためのUIがあるので有効化する。

5. (オプション)セキュア接続設定
   MonitorとHost双方でセキュア接続の設定をすることができる。
6. (オプション)ヘッドレスモードの設定(ローカルデバッグのみ)
   Monitorが動作するPCでは、NVIDIAコントロールパネルにてデバッグ用GPUのディスプレイ出力を無効化し、PhyXアクセラレーション対象をデバッグ用GPUに設定することで、対象GPUをCUDAデバッグに専念させることができるようになる。これによりWindowsのデスクトップ処理等の影響が無くなるので良いとされている。

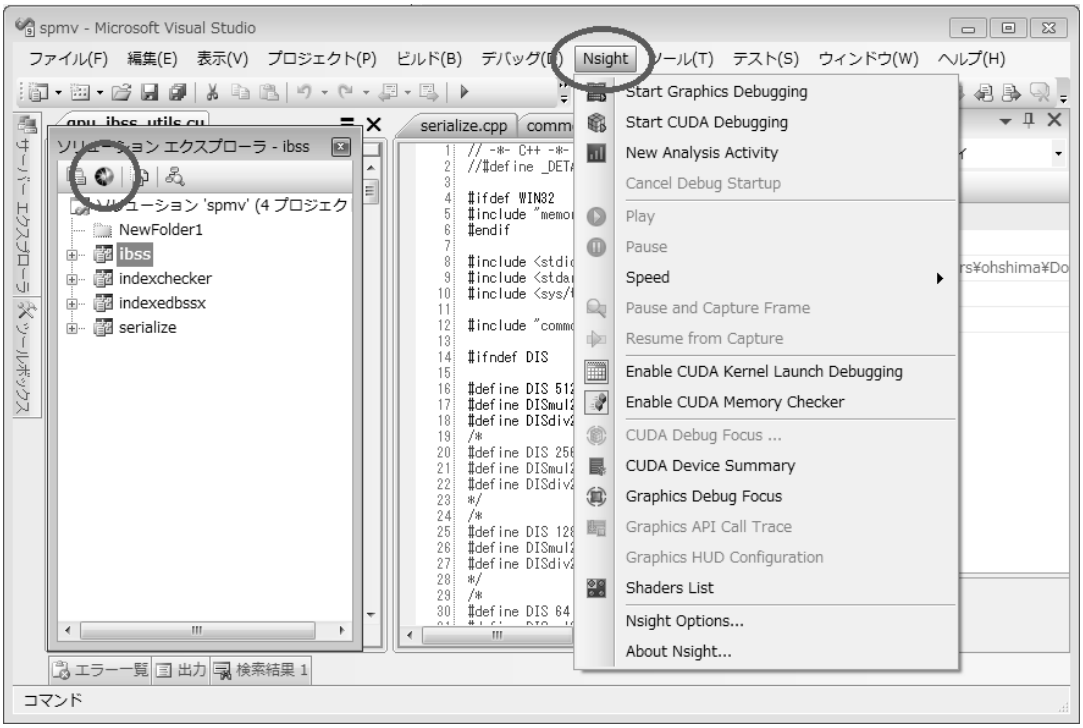

図 2　Nsight が組み込まれた Visual Studio の画面 (メニューに Nsight が、ソリューションエクスプローラのアイコンメニューに Nsight のアイコンが追加されている)

3.3　Nsight を用いたデバッグの手順

Nsightを用いてデバッグをする前に、まずはCUDAを用いたプログラムをVisual Studio上で作成する必要がある。以前からVisual Studioのカスタムビルド規則ファイルやCUDA用のウィザードがweb上でいくつか公開されていたが、現在ではNsightを導入するとカスタムビルド規則が追加されるようになっており、.cuファイルをプロジェクトに追加する際やプロジェクトの設定(ソリューションエクスプローラでプロジェクトを選択した状態でプロジェクトメニューを開き、カスタムビルド規則を選ぶ)を開けばNsight CUDA Runtime Api Build Rule v3.1.0などの名前の規則が選択できる。もしこれらの規則が見つからない場合は、規則設定画面で既存ファイルの検索を選び、`C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v3.1`から.rulesファイルを選択すれば良い。

以上の設定によってカスタムビルド規則を設定すれば、あとは通常のプロジェクト同様にビルドを行うことでCUDA対応アプリケーションが作成可能になる。プロジェクトのプロパティからカスタムビルド規則の設定画面(プロパティページの構成プロパティツリー最下部に追加されている) を開けば、nvccに与える各種オプションをGUIから選択することもできるので、追加のインクルードディレクトリなどが必要な場合は設定すると良いだろう。

Nsightを用いたデバッグについては、GPUカーネル内にブレークポイントを設置したりステップ実行したりするだけであれば、任意の位置にブレークポイントを設置した後でNsightメニューの中にあるStart CUDA Debuggingで実行すれば良いだけである。ただし、GPUカーネル内部のデバッグをしたい場合はNsightメニューのStart CUDA Debuggingを、ホスト側のデバッグをしたい場合はデバッグメニューのデバッグ開始を選ぶよう使い分ける必要がある。また、実行時引数の与え方についても、Start CUDA Debuggingから起動する場合にはNsight User Properties(ソリューションエクスプローラのアイコンメニュー、もしくはソリューションエクスプローラ上でプロジェクト名を選択して右クリックしNsight User Propertiesを選択) で表示されるUIを用いて引数を設定する必要がある。ローカルデバッグを行うかリモートデバッグを行うかの指定については、Nsight User Properties中のLaunch中にあるConnection name:をlocalhostにするか他のホスト(IPもしくは名前解決可能なホスト名)にするかで制御する。

幸い、Nsightは設定の不具合などがある場合にはポップアップや出力ウィンドウへの出力で具体的な問題点などを指摘してくれるため、各種の設定に抜けや間違いがあった場合でも比較的容易に気がつくことができるだろう。

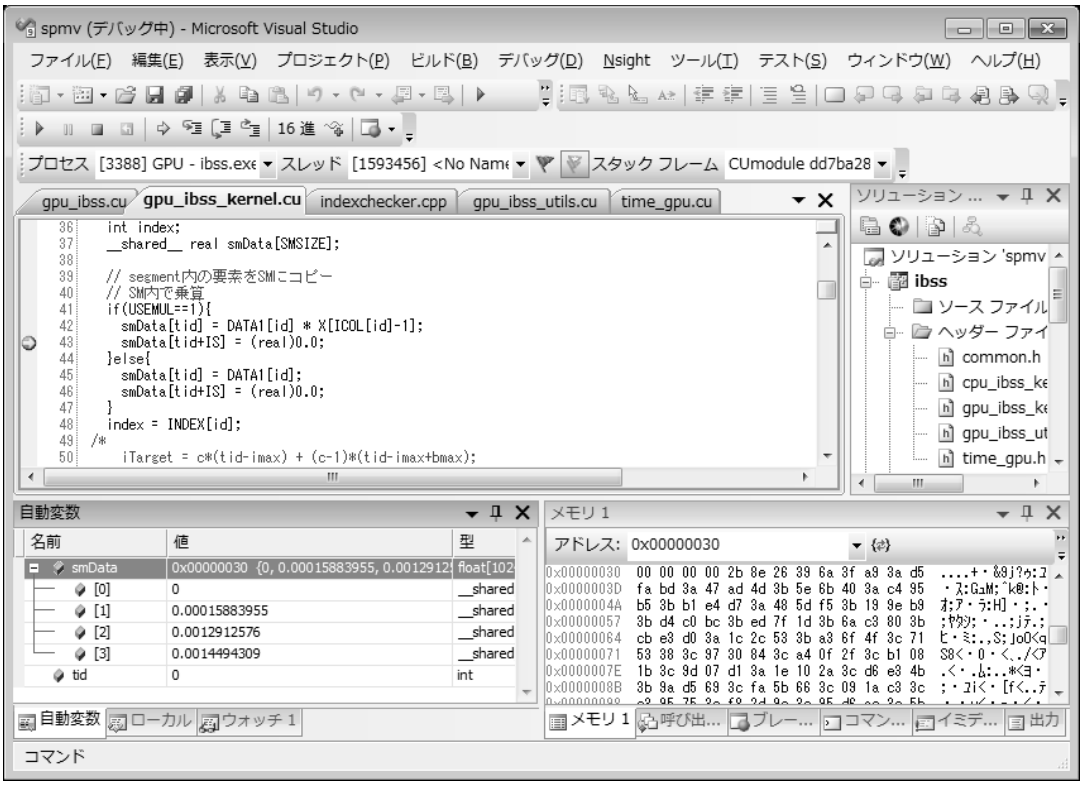

図 3　GPU カーネルのステップ実行例 (SharedMemory 上の配列 smData の中身を確認できている)

Nsightを用いたデバッグ(GPUカーネルのデバッグ)中の操作については、通常のVisual Studioと同様にステップ実行やメモリ内容の確認を行うことができる(図3)。一方でNsight独特の操作としては、BlockやThreadを切り替えてのステップ実行が挙げられる。Nsightメニューの CUDA Debug Focusメニューがこれに該当しており、ステップ実行時に本メニューから対象となるBlockおよびThreadを選択することが可能となる。

他の機能としてはプロファイラ機能が統合されている(図4)。NsightメニューからNew Analysis Activityを選ぶと表示されるUIにてプロファイル対象を選択してLaunchボタンを選択すれば、アプリケーション全体やGPUカーネルなどの実行情報を自動的に収集し、表やグラフによって表示してくれる機能である。デバッガとあわせて活用することでCUDA最適化プログラミングがより容易になるだろう。

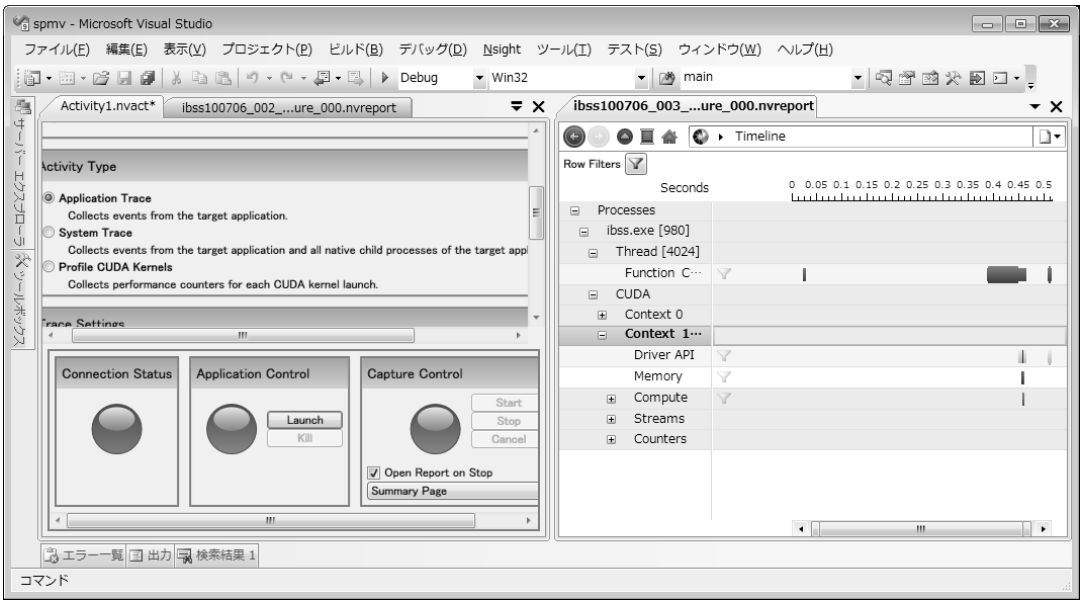

図 4　プロファイラ機能 (左側は設定画面、右側はタイムライン表示画面)

以上、第四回の今回は予定を変更してWindows向けのCUDA開発環境について紹介した。WindowsとLinuxで全く同じプログラムが動かせるかが使っているライブラリ等により左右されてしまうのは周知の事実であるが、Nsightのデバッグ機能は非常に強力なので、Linux向けのCUDAプログラミングを行っている人も使ってみる価値があるだろう。

(次回に続く)

参考文献

[1] NVIDIA. Nvidia parallel nsight, `http://developer.nvidia.com/object/nsight.html`.