B-003

異なる並列プログラミング言語で書かれた PTX コードの比較

Comparison of PTX codes written in different parallel programming languages

魚谷　智志 *　南　昇吾 *　祢津　祐 *　増田　信之 *
Satoshi Uotani　Shogo Minami　Tasuku Netsu　Nobuyuki Masuda

1. まえがき

近年，計算機の性能向上に伴い大規模かつ高精度な数値計算の需要が高まっている．最近では CPU よりも計算の理論性能が高い GPU を数値計算に応用した GPGPU が広く用いられている．GPGPU の開発環境として，CUDA や AMD APP などのベンダー依存のものと，OpenCL や OpenACC などのベンダー依存でないものがある．これらの異なる環境で同じ数値計算を行った場合，どのような過程で計算処理が行われているかを知るためには，命令セットアーキテクチャを解析する必要がある．そこで本研究では，CUDA と OpenCL で記述された 2 次元 FDTD 法計算プログラムから PTX ファイルを作成し，どのような命令が出ているのか検証を行った．

2. FDTD 法

FDTD 法とは，電磁界現象の支配方程式であるマクスウェルの方程式を差分化し，時間領域で解く手法である．FDTD 法は，極めて単純なアルゴリズムであることから，1990 年代からアンテナやマイクロ波，電磁環境など様々な分野に応用されてきた [1]．本稿では，境界条件として Mur の吸収境界条件を用いた二次元 FDTD 法について解く．二次元の問題では，電界または磁界成分が 1 方向に対して不変で，電界または磁界成分がその方向のみとなる．電界成分が 1 方向のみとなる TM 波が入射するとき，マクスウェル方程式は次のようになる．

$$\begin{cases} \frac{\partial H_x}{\partial t} &= -\frac{1}{\mu}\frac{\partial E_z}{\partial y} \\ \frac{\partial H_y}{\partial t} &= -\frac{1}{\mu}\frac{\partial E_z}{\partial x} \\ \frac{\partial E_z}{\partial t} &= -\frac{1}{\varepsilon}\left(\frac{\partial H_x}{\partial y} - \frac{\partial H_y}{\partial x}\right) \end{cases} \tag{1}$$

ここで，H_x は x 軸方向の磁場，H_y は y 軸方向の磁場，E_z は z 軸方向の電場を表す．パラメーター ε と μ はそれぞれ誘電率と透磁率を表す．式 (1) をそれぞれ時間，空間に対して中心差分を行い，差分方程式で表すと次のようになる．

$$H_x^{n+\frac{1}{2}}(i,j+\frac{1}{2}) = H_x^{n-\frac{1}{2}}(i,j+\frac{1}{2}) - \frac{\Delta t}{\mu\Delta y}\{E_z^n(i,j+1) - E_z^n(i,j)\} \tag{2}$$

$$H_y^{n+\frac{1}{2}}(i+\frac{1}{2},j) = H_y^{n-\frac{1}{2}}(i+\frac{1}{2},j) - \frac{\Delta t}{\mu\Delta x}\{E_z^n(i+1,j) - E_z^n(i,j)\} \tag{3}$$

$$E_z^{n+1}(i,j) = E_z^n(i,j) - \frac{\Delta t}{\varepsilon\Delta y}\{H_x^{n+\frac{1}{2}}(i,j+\frac{1}{2}) - H_x^{n+\frac{1}{2}}(i,j-\frac{1}{2})\} + \frac{\Delta t}{\varepsilon\Delta x}\{H_y^{n+\frac{1}{2}}(i+\frac{1}{2},j) - H_y^{n+\frac{1}{2}}(i-\frac{1}{2},j)\} \tag{4}$$

ここで，空間離散間隔を Δx，Δy とし，時間離散間隔を Δt，時間ステップ数を n とする．

3. 実験結果

本研究では，CUDA と OpenCL で書かれた FDTD 法の計算プログラムを GPU で計算させ，計算時間を測定した．それぞれのプログラムは API 固有の表現以外は同一に記述し，計算時間の測定を行い，PTX コードの比較を行った．また，磁場 H_x，H_y の係数を一括りに変更した場合についても同様に比較を行った．

3.1 実験環境

本実験で使用した開発環境を表 1 に示す．計算時間の計測には，CUDA では nvprof，OpenCL では clGetEventProfilingInfo を用いた．メモリはグローバルメモリを用い，カーネル起動時間に関しては今回は無視した．従って，GPU において純粋な計算時間のみ測定した．時間ステップ数を 850，計算領域を $L \times L$，thread 数を x，y 両方向共に 16 とし，初期条件として磁界は 0，電界は半分の領域を 0，もう半分を 1 とした．計測時間は 850 ステップの平均をとり，1 ステップの計算にかかる時間を算出した．また，$\Delta x = \Delta y = 1, c_0 = 1, \Delta t = 0.5, \mu = 1, \varepsilon = 1$ とした．

表 1: 開発環境

CPU	Intel Xeon E5-2697 v2 (2.70 GHz) ×2
RAM	64 GB (32 GB ×2)
GPU	NVIDIA GeForce GTX 1080
OS	CentOS Linux 7.1.1503
compiler	gcc4.8.3 , nvcc 8.0

3.2 計測結果

本実験の計測結果を表 2 に示す．CUDA，OpenCL 共に計算時間が短い方を赤字で示した．表 2 より，計算量が少ない場合は CUDA，計算量が多い場合は OpenCL の方が計算時間が短いことが分かる．また，計算量が少ない場合において CUDA では計算カーネルの 1 箇所を変更した方が計算時間が短くなる傾向があることが分かる．しかし，OpenCL においてはこの傾向があるとは言えない．また，計算結果における差異は見られなかった．

*東京理科大学基礎工学部

表 2: CUDA と OpenCL それぞれの計算時間

	CUDA [µs]		OpenCL [µs]	
計算領域	変更前	変更後	変更前	変更後
18^2	6.575	6.502	17.128	16.949
34^2	6.617	6.482	16.888	16.764
66^2	6.737	6.583	16.589	16.954
130^2	9.694	9.648	19.862	19.901
258^2	20.883	20.684	27.189	26.826
514^2	98.645	105.382	108.933	107.859
1026^2	430.279	445.510	440.877	441.992
2050^2	1499.40	1464.80	1417.14	1418.01
4098^2	9005.84	9224.41	8790.11	8797.03
8194^2	36776.2	37695.0	35785.0	35802.6
16386^2	147264	151414	143406	143299

ソースコード 1 に CUDA，ソースコード 2 に OpenCL の PTX コードの一部を示す．ここで示す PTX は計算領域 18×18 である．両者を比較すると，“fd8” に磁場 H_x を代入する演算までは同じ処理を行っていることが分かる．しかし，図 1 に示すようにそれ以降の演算の命令が異なっている事が分かる．また，CUDA では命令数が 31 であるのに対し，OpenCL では約 1.5 倍の 44 となった．

ソースコード 1: CUDA の PTX コードの一部

```
mad.lo.s32      %r8, %r6, %r5, %r7;
mad.lo.s32      %r9, %r4, 18, %r8;
mul.wide.s32    %rd7, %r9, 8;
add.s64         %rd8, %rd6, %rd7;
ld.global.f64   %fd1, [%rd8+152];
add.s64         %rd9, %rd5, %rd7;
ld.global.f64   %fd2, [%rd9+296];
ld.global.f64   %fd3, [%rd9+152];
sub.f64         %fd4, %fd2, %fd3;
fma.rn.f64      %fd5, %fd4, 0
    d3FE0000000000000, %fd1;
add.s64         %rd10, %rd4, %rd7;
ld.global.f64   %fd6, [%rd10+160];
ld.global.f64   %fd7, [%rd10+152];
sub.f64         %fd8, %fd6, %fd7;
mul.f64         %fd9, %fd8, 0
    d3FE0000000000000;
sub.f64         %fd10, %fd5, %fd9;
st.global.f64   [%rd8+152], %fd10;
```

ソースコード 2: OpenCL の PTX コードの一部

```
mul.lo.s64      %rd10, %rd6, 18;
add.s64         %rd11, %rd10, %rd9;
add.s64         %rd12, %rd11, 19;
cvt.s64.s32     %rd13, %rd12;
shl.b64         %rd14, %rd13, 3;
add.s64         %rd15, %rd3, %rd14;
ld.global.f64   %fd1, [%rd15];
add.s64         %rd16, %rd11, 37;
cvt.s64.s32     %rd17, %rd16;
shl.b64         %rd18, %rd17, 3;
add.s64         %rd19, %rd2, %rd18;
ld.global.f64   %fd2, [%rd19];
ld.global.f64   %fd3, [%rd19+-144];
sub.f64         %fd4, %fd2, %fd3;
fma.rn.f64      %fd5, %fd4, 0
    d3FE0000000000000, %fd1;
add.s64         %rd20, %rd11, 20;
cvt.s64.s32     %rd21, %rd20;
shl.b64         %rd22, %rd21, 3;
add.s64         %rd23, %rd1, %rd22;
ld.global.f64   %fd6, [%rd23];
ld.global.f64   %fd7, [%rd23+-8];
sub.f64         %fd8, %fd6, %fd7;
fma.rn.f64      %fd9, %fd8, 0
    dBFE0000000000000, %fd5;
st.global.f64   [%rd15], %fd9;
```

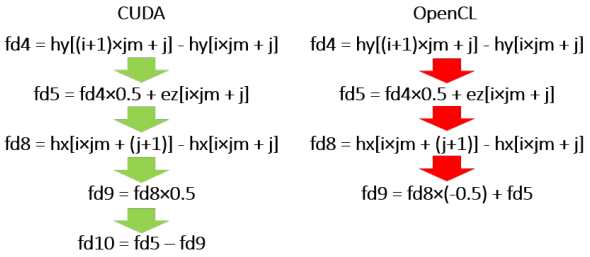

図 1: それぞれの PTX コードの計算順序

4. まとめと今後の課題

本稿では，CUDA と OpenCL で記述されたプログラムの PTX の比較を行った．使用する API が異なることから，出される命令に何らかの違いがあると仮定した．結果として，命令の出方や演算プロセス，命令数などが異なることが確認できた．命令数を見ると CUDA の方が少ないため，計算時間が短かったと考えられる．しかし，計算量が増加すると OpenCL の方が計算時間が短いことが分かった．この原因として，計算量が増えるにつれて，アドレスの加算量が大きくなってしまう事が挙げられる．CUDA では一つのアドレスにひたすら値を加算しているため，命令数や使用するレジスタ数が少なく済むが，計算量の増加に伴いアドレスの加算量が大きくなるにつれ，その部分がボトルネックになったと考えられる．一方で，予めアドレスの値を別のレジスタで代入している OpenCL では，CUDA よりもこの部分の影響が少ないため，計算時間が短かったと考えられる．

今後の展望として，直接 PTX を書き換えた場合に計算の高速化が行えるか検討する．

参考文献

[1] 宇野亨編著, 何一偉, 有馬卓司共著,『数値電磁界解析のための FDTD 法 -基礎と実践-』, コロナ社, p362, 2016.