Bulletin of Aichi Univ. of Education, 61（Natural Sciences）, pp. 25 − 28, March, 2012

OpenCLによる Common Lisp言語処理系の高速化の一考察

安本 太一

情報教育講座

A Study of Optimization for Common Lisp System Using OpenCL

Taichi YASUMOTO

Department of Information Sciences, Aichi University of Education, Kariya 448-8542, Japan

1 はじめに

OpenCL［1，2，3，4］をLisp言語処理系へ適用することを試みたので，その過程と結果を報告する．OpenCL（Open Computing Language）は，2008年12月にアップル社によって提案された，マルチコアCPUやGPU（Graphics Computing Unit）などで並列計算を行うためのフレームワークである．現在は，標準化団体クロノス・グループの作業部会によって仕様が管理されている．

最近のパーソナルコンピュータは，GPUを搭載しているのが普通であり，GPUは身近な存在となっている．普及価格帯のコンピュータにも比較的高性能なGPUが搭載され，ゲームなどの高いグラフィックス性能を必要とするアプリケーションを実行していない場合は，GPUがその能力を生かしきれず遊んでいる状態も少なくないと推測される．遊休状態にあるGPUの能力を汎用的に使用するという取り組みがある．

マルチコアCPUで並列計算を行うためのフレームワークには，GDC（Grand Central Dispatch）やPthreads（POSIX Threads）などがある．一方，GPUを汎用的な演算にも適用するというGPGPU（General Purpose computing on GPU）という考え方も取り込んだ，並列計算のためのフレームワークの一つとしてOpenCLがある．

OpenCLは特定のメーカに偏ることなく，AMD社やNVIDA社のGPUをサポートしていることから汎用性が高いと判断し，OpenCLのLisp言語処理系への適用を試みるに至った．

2 OpenCLの概要

OpenCLのモデルは，SIMD（Single Instruction Multiple Data）とSPMD（Single Program Multiple Data streams）の並列処理をサポートしている．このモデルにおいては，1個以上の“PE（Processing Element）とプライベートメモリの対”からなるCU（Computing Unit）あり，1個以上の“CUとローカルメモリの対”とグローバルメモリからなるOpenCLデバイスがある．

OpenCLデバイスの具体的なものとしては，GPU，マルチコアCPUやDSPなどがあげられる．OpenCLデバイスを使用したいホストは，OpenCLデバイスを取得し，（引数をセットした後に）カーネルとばれるOpenCLデバイス上で実行されるプログラムを呼び出し，カーネル終了後，ホストがカーネルの実行結果をOpenCLデバイスから受け取る．ホストは，プログラムの主要な流れの制御を司るコンピュータで，具体的にはディスクトップコンピュータなどが相当する．

カーネルはOpenCL C言語で記述する．OpenCL C言語はC99のサブセットに並列計算のための機能を追加したものである．OpenCL C言語で記述されたカーネルは，ホスト上のプログラムの実行中に（カーネル実行前に）オンデマンドでコンパイルされ，実行オブジェクトに変換される．カーネルは，C言語の関数のように記述するので，カーネル関数とよばれることある．カーネル関数のソースプログラムとなる文字列は，C言語の文字列としてホスト上のプログラムに埋め込んだり，ファイルに格納しておいて実行時にファイルから読む込む事もできる．

ホスト上のプログラムは，C言語，C++言語あるいはObjective Cを使って記述する．これらの言語から利用できるOpenCL APIを使って，カーネルのソースプログラムをコンパイルしたり，カーネルへの引数をセットしたり，カーネルを呼び出したりする．

3 OpenCLC言語の特徴

OpenCL C言語では，SIMD（およびSPMD）のプログラムを記述するための一般的な機能は備わっている．例えば，PE番号に相当するグローバルIDを取得するための機能があり，グローバルIDをもとに条件分岐（命令を実行する，実行しないの選択）を行うこと

ができる．グローバルIDは，1次元，2次元，3次元用がある．

barrierという名前の同期関数がある．これは，（OpenCLデバイスのハードウェアなどの状況によるが）各PEが別の命令を実行することできる（PE間でプログラムカウンタが一致しないことがありうる）からである．

データ型はC言語のcharからlongに相当するスカラデータ型が使用できる．また，これらの単純データ型からなるベクタデータ型も使用できる．2次元や3次元のイメージ型も使用できる．関数へのポインタや再帰はサポートされていない．OpenCLデバイスはCPU，GPU，DSPなど幅広い計算資源が想定されているので，最大公約数的な仕様にしていると考えられる．

4　OpenCLによる組み込み関数の並列化の試み

OpenCLによる組み込み関数の並列化の試みとして，組み込み関数maxの並列化を試みた．maxは受け取った引数の最大値を返す．使用した言語処理系は，64ビット版のKCL（Kyoto Common Lisp）[5，6，7，8，9]である．今回は，引数がすべて固定長整数（fixnum）である場合の並列化を試みた．固定長整数以外の場合は，従来どおりの逐次版コードが実行される．

64ビット版のKCLでは，fixnumは，下位2ビットがデータ型を示す即値データとして実現されている．実質的には62ビットの整数が，ポインタ中に（符号ビットも含めて）左に2ビットシフトされている形になっている．OpenCLでは符号付きの64ビットの整数がサポートされているので，fixnumのポインタ表現のまま収容できる．OpenCLのカーネル関数におけるfixnumの大小比較においては，固定長整数に戻さず，ポインタ表現のまま大小比較をすることにした．そのため，固定長整数とそのポインタ表現の間の変換によるオーバヘッドはない．

オリジナルのKCLのmaxは，引数を（Lispデータへのポインタの）スタックに積むことになっていて，このLispデータのスタックは（Lispデータへのポインタの）配列で実現されている．OpenCLにおいては，ホストプログラムからカーネル関数へ，データ並列の引数データを渡す時は，連続したメモリ領域からなるメモリオブジェクトを確保し，そのメモリ領域に引数データをコピーし，そのメモリオブジェクトの先頭のポインタを（OpenCL APIを通じてカーネル関数を呼び出す前に）カーネル関数に通知することになっている．メモリオブジェクトへのデータ並列の引数データのセットは，連続したメモリ領域である配列からの単純コピーで済む．OpenCLのランタイムの実装にもよるが，OpenCLデバイスがマルチコアCPUである場合は，ホストとOpenCLデバイスが実質同じなので，コピーをせずにポインタを渡すだけで済むかもしれない．

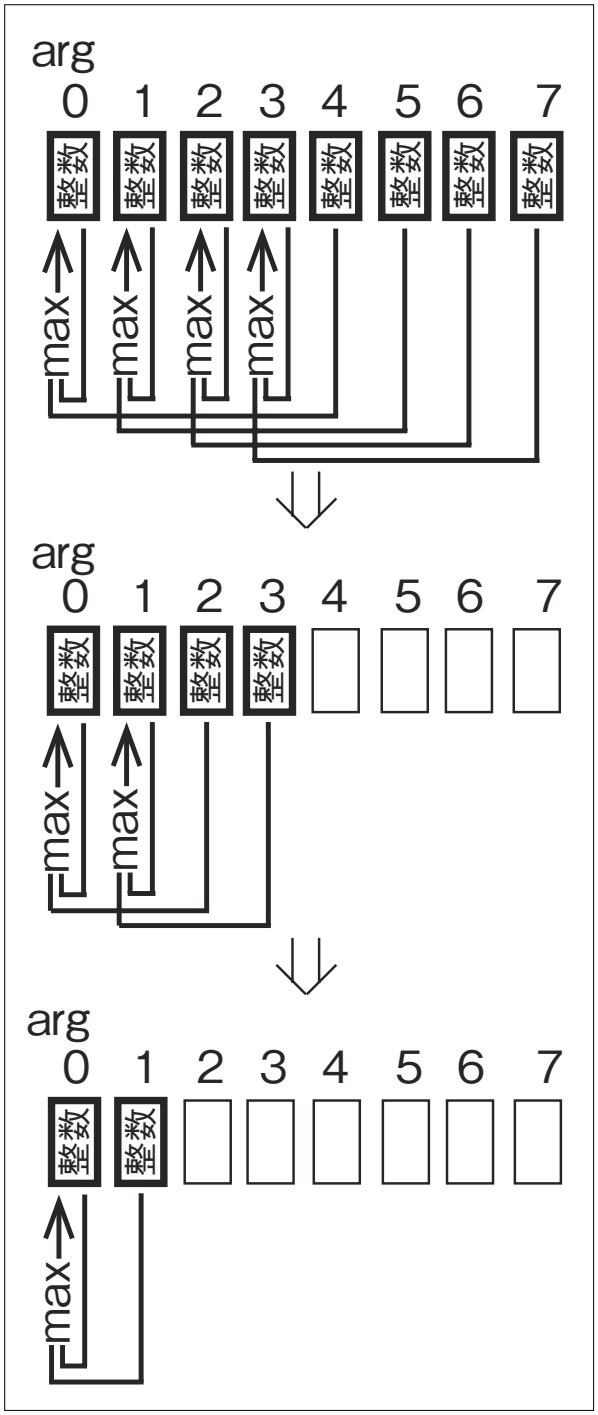

図1：maxの並列化のアルゴリズムの概要

今回実装するmaxの並列版では，図1に示すように，配列に配置されている2つの要素を比較して大きい方を添字の小さい方に格納することを並列に行うことを，繰り返す．

図2に（組み込み関数maxの本体から呼び出され）並列に実行されるカーネル関数findMaxの概要を示す．つまり，構文上は，Lispの組み込み関数maxの引数の数だけ，このカーネル関数findMaxが並行に実行される．すなわち，構文上，Lispの組み込み関数maxの引

```
__kernel void
findMax(int num, __global long *arg)
{
  int flag;
  int temp;
  int index = get_global_id(0);

  while(num >= 2){
    flag = num % 2;
    num = num / 2;

    if(index  < num)
      arg[index]
        = max(arg[index],
              arg[index + num]);

    barrier(CLK_LOCAL_MEM_FENCE
            | CLK_GLOBAL_MEM_FENCE);

    if(flag && (index == (temp = num * 2)))
      arg[0] = max(arg[0], arg[temp]);

    barrier(CLK_LOCAL_MEM_FENCE
            | CLK_GLOBAL_MEM_FENCE);
  }
}
```

図2：並列版maxのカーネル関数の概要

数だけ（並行に実行される）“制御の流れ”がある．各制御の流れにおいては，get_global_id（0）で得られるグローバルID（各制御の流れに付与される固有の番号，SIMD型並列計算機におけるPE番号のようなもの）を参照して，条件分岐をおこなう（命令を実行するか否かを決める）．図2の配列argの要素の一つが組み込み関数maxの引数の一つに相当し，添字付きでアクセスされるarg［グローバルID］が，そのグローバルIDをもった制御の流れが保持するfindMaxの引数の一つとなる．制御の流れのグローバルIDとargの添字が異なる場合は，制御の流れは他の制御の流れが保持するデータにアクセスする事になる．紛らわしいが，図2中のmaxは，OpenCL C言語のライブラリ関数maxであり，Lispの組み込み関数maxとは直接関係がない．

実際には，CUの分だけ，制御の流れが同時に並行実行される．引数の数（制御の流れ）がCUの数より大きい場合は，制御の流れをCUの数だけ同時に実行することがループにより繰り返される．CUで同時実行される制御の流れの具体的な数については，OpenCLランタイムシステムに任されるようである．

5　評価

5.1　評価実験

OpenCLを用いて並列化したmaxの性能を，オリジナル（逐次版）のmaxと比較した．（max 1 2 .... 29999 30000）を100回実行し，その実行時間を比較した．

使用した計算機は，アップル社製Mac Pro Early 2009（クワッドコアIntel Xeon5550（2.66 GHz）×2，メインメモリ12 GB，二次キャッシュ256 KB（コア単位），三次キャッシュ（プロセッサ単位）8 MB，グラフィックスはATI Radeon HD 4870 VRAM512MB，OSはMacOSX10.7.1）である．実行時間を表1に示す．

並列版（GPU）というのは，OpenCLデバイスとしてGPUを指定して，（max 1 2 .... 29999 30000）をdotimesで100回実行した場合である．並列版（CPU）は，同様に，OpenCLデバイスとしてCPUを指定した場合である．並列版（GPU）と並列版（CPU）は逐次版（オリジナル）のそれぞれ約22倍，約7倍とあまりにも遅い．そこで，maxから呼び出しているカーネル上の並列関数を100回呼び出し（大雑把にいうとfindMaxを100回実行し），Lispの組み込み関数maxは1回しか呼び出さないようにしたものが，それぞれ並列版（GPUカーネルで100回），並列版（CPUカーネルで100回）である．

5.2　考察

OpenCLを用いて並列化したのに，並列版（GPU）と並列版（CPU）は，逐次版（オリジナル）に比べて，実行時間が大幅に長くなってしまった．原因としては，カーネル関数実行のコストの高さにあると推測される．OpenCLデバイス側での並列計算の計算量が少ないと，OpenCLによる並列処理の恩恵が得られない．

表1からは，カーネル関数実行のオーバーヘッドは，OpenCLデバイスがGPUの場合の方がCPUの場合より高いといえる．dotimesでLispの組み込み関数maxを繰り返して何度もカーネル関数を呼び出しているのを，カーネル関数の呼び出しは一度だけにしてカーネル内での繰り返しに置き換えた場合の実行時間の短縮の割合が，GPUの方が大きいからである．CPUの場合は，実行時間の短縮の割合が小さい．OpenCLデバイス

表1：オリジナルのmaxとOpenCL並列版のmaxの実行時間の比較

maxのバージョン	実行時間	比
逐次版（オリジナル）	0.125	1.0
並列版（GPU）	2.6965	21.572
並列版（CPU）	0.8695	6.956
並列版（GPUカーネルで100回）	0.0304	0.2432
並列版（CPUカーネルで100回）	0.6346	5.0768

（実行時間の単位は秒）

がCPUの場合は，ホストプログラムとカーネルは，同じ装置内で実行されるため，引数の受け渡しや制御の流れの管理のコストが低く，実行時間の短縮の割合が小さいのだろう．

評価実験に用いたMacPro Early 2009においては，グラフィックスカードはATI Radeon HD 4870はPCI Express 2.0レーン幅× 16であり，メモリはDDR3 1066MHzである．GPUが遅い速度で接続されているとは思わないが，引数などのデータ転送が効率的ではないのかもしれない．

また，現行ではmaxの呼び出しごとに，プログラムオブジェクトの作成（カーネル関数のソースコード文字列をコンパイルしてオブジェクトを作成）をしていて，カーネル関数の定義が変わらなくても，dotimesによるmaxの繰り返しごとにコンパイルしているのは無駄だといえる．ソースコード文字列に変更がなく，かつOpenCLデバイスに変更がない場合は，以前に作成したプログラムオブジェクトを使い回すといった最適化が必要だと思われる．

ただし，カーネル関数のコンパイルを実行時に行うというOpenCLの仕様自体は，動的な性質を特徴とするLisp言語にはあっている．

評価のため設定した特殊な環境とはいえ，並列版（GPUカーネルで100回）が逐次版（オリジナル）より実行時間が4分の1に短縮されたことは，GPUによる速度向上の可能性を示している．並列版（CPUカーネルで100回）が，逐次版（オリジナル）より速くならなかったことは，なんらかの理由で，OpenCLの枠組みの下でのマルチコアCPUによる並列化が適合しなかったことを示している．この原因の追及は今後の課題としたい．

6 まとめ

OpenCLにより，Lispの組み込み関数の一つmaxを並列化してみた．特殊な状況下ではあるが，OpenCLデイバイスにGPUを用いた場合に速度向上が得られる事がわかった．一回のカーネル関数の呼び出しにおいて，GPU側で多くの計算を並列にこなす場合に，OpenCLによる並列化の効果が得られる．

OpenCLは発表されたのが2008年であり，歴史が浅く，その実装も安定していないと筆者は感じている．筆者は，GPUが異なるMacを複数台有しているが，アップル社ホームページや市販の本で紹介されているOpenCLサンプルプログラムが（サンプルプログラムの開発環境が筆者と異なるせいか）動作しない例がいくつかある．しかしながら，様々なGPUやマルチコアCPUに対して，同一のフレームワークの下で，並列プログラミングを行う事ができることは高く評価できる．

プログラムの実行時に，ソースプロラムの文字列からコンパイルしてカーネルの関数のオブジェクトプログラムを得るというOpenCLの特徴は，ホストのCPUは同一であっても，コンピュータによって装着しているGPUが異なる事が多いとう現実に即したものである．装着しているGPUの情報は，プログラムの実行時にわかるので，装着しているGPUに応じて，オンデマンドでLispの関数をコンパイルして最適化といった応用が考えられる．

参考文献

[1] Khronos OpenCL Working Group: *The OpenCL Specification Version:1.1*, Khronos Group (2011).

[2] フィックスターズ：OpenCL入門マルチコアCPU・GPUのための並列プログラミング，インプレスジャパン（2010）．

[3] 池田成樹：OpenCL並列プログラミング，カットシステム（2010）．

[4] 奥園隆司：OpenCL入門GPU&マルチコアCPU並列プログラミング，秀和システム（2010）．

[5] Steele, G. L. Jr.: *Common Lisp the language*, Digital Press (1984).

[6] Yuasa, T. and Hagiya, M.: *Kyoto Common Lisp Report*, Teikoku Insatsu Publishing (1985).

[7] Yuasa, T.: Design and Implementation of Kyoto Common Lisp, *Journal of Information Processing*, Vol. 13, No. 3, pp. 284–295 (1990).

[8] 湯淺太一，安本太一：KCLにおける即値データの実装とその評価，電子情報通信学会春季全国大会講論集，D–357（1989）．

[9] 安本太一：Common Lisp言語処理系の64ビット化，愛知教育大学研究報告自然科学編，五十三輯，pp. 27–32（2004）．

（2011年9月16日受理）