Vol.2011-HPC-129 No.13
2011/3/16

HxABCLibScript: 非均質計算機向け自動チューニング記述言語拡張

片桐孝洋† 大島聡史† 平澤将一†† 本多弘樹††

本稿では，CPU および GPU(Graphics Processing Unit)を混載した非均質計算機において，任意のプログラムの一部分が，適する計算資源上で実行される最適化を実現する自動チューニング専用言語 HxABCLibScript を提案する．性能評価の結果，HxABCLibScript 記述から自動生成されるコードは，問題サイズや反復回数に応じ，CPU と GPU 間で適切に計算資源を切り替えることで最適化されることを確認した．

HxABCLibScript: An Extension of an Auto-tuning Language for Heterogeneous Computing Environment

TAKAHIRO KATAGIRI† SATOSHI OHSHIMA†
SHOICHI HIRASWA†† and HIROKI HONDA††

In this paper, we propose HxABCLibScript, which is a dedicated language for auto-tuning description on heterogeneous computer environment, which includes CPU and GPU (Graphics Processing Unit), to adapt arbitrary parts of programs. Results of performance evaluation indicated that the automatically generated codes from the description of HxABCLibScript can select the best computer resources between CPU and GPU according to problem size or the number of iterations on the program.

1. はじめに

計算機システムを取り巻く環境がますます複雑化している．ノードあたり 64 コアを超えるマルチコアプロセッサは現実となっている.キャッシュも多くの階層をもち，かついくつかのコアで共有されている．電力あたりの計算性能が高いため，CPU に加え GPU(Graphics Processing Unit)を搭載した計算機（非均質計算機）が登場した．現在 GPU を計算用途に使う GPGPU の研究が花盛りである．複雑化された非均質計算機では，高性能ソフトウェア開発のため性能チューニングの手間（コスト）が増大する．高性能化は高度に専門的となり，技術者も少なく，作業量も多い．その結果，開発手順（工数）は増大して，開発コストは高くなる．

このような背景から，安価に高性能なソフトウェアを開発する技術が待望されている．この要求に答える技術の 1 つとして研究されているのが，ソフトウェア自動チューニング（以降，AT）技術である．AT 技術は，数値計算ソフトウェアでその有効性が証明されてきた．たとえば，PHiPAC[1], ATLAS[2], FFTW[3], ABCLib[4], OSKI[5]など，成功例は枚挙にいとまがない。一方で，AT 技術をできるだけ汎用的な対象（プログラム）に拡大し，コード自動生成を中心とする性能チューニングの工数削減を目的にする研究は多くはない．コンパイラ側のアプローチとして ROSE プロジェクト[6]，ランタイムライブラリ側のアプローチとして Active Harmony プロジェクト[7]が知られている．また国内においては，ユーザ知識をもとにしたプリプロセッサのアプローチとして ABCLibScript[8]がある．

以上を鑑み，本稿は以下の構成からなる．まず 2 節で，非均質計算機向けの ABCLibScript の機能拡張を提案する．3 節は，2 節の機能を GPU と CPU の環境を用いて性能評価する．最後にまとめを行う．

2. *HxABCLibScrpt*: 非均質計算機向け自動チューニング記述言語拡張

2.1 概要

本節では AT 機能の付加のための専用言語 ABCLibScript[8]の機能をもとに，CPU と GPU の非均質計算機向の拡張 AT 言語 HxABCLibScript (Heterogeneous extended ABCLibScript)を提案する．

提案する HxABCLibScript は，HxABCLibScript の記述を解釈しコード生成するプリプロセッサ ABCLibCodeGen により処理される．ABCLibScript のコンセプトは，自動

† 東京大学情報基盤センター スーパーコンピューティング研究部門
Supercomputing Research Division, Information Technology Center, The University of Tokyo
†† 電気通信大学 大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications.

生成されたコードを，多様な計算機環境上で，かつ，多様なコンパイラでコンパイルして実行可能コードを生成できることに主眼が置かれている．すなわち，AT 機能を実行することで，多様な計算機環境でも 1 つのプログラムから高性能化を実現する枠組みを提供することにある．

従来の ABCLibScript は CPU 上において，逐次実行，スレッド実行，および MPI を用いた分散メモリ型並列計算機での実行を目的にしていた．ここでは，GPU 上の実行を考慮した ABCLibScript の枠組みの提案を行う．

2.2 コード生成の流れ

図 1 に HxABCLibScript によるコード生成の流れを示す．図 1 ではまず，専用プリプロセッサにより ABCLibScript で記載された機能に基づくコード生成を行い，最適化候補となるコード（CPU 用）を複数生成する．その後，CPU 用のコードから GPU 用コードを生成できるコンパイラにより，CPU 実行と GPU 実行の双方ができる，最適化候補と AT 機能を含む実行可能コードを生成する．その後，ABCLibScript が提供する AT 機能により，CPU 実行，もしくは GPU 実行に適する最適化候補を探索する．ユーザが利用する時には，CPU 上，もしくは GPU 上で最適化されたコードによる実行が選択される．これが HxABCLibScript による CPU と GPU の最適化枠組みである。

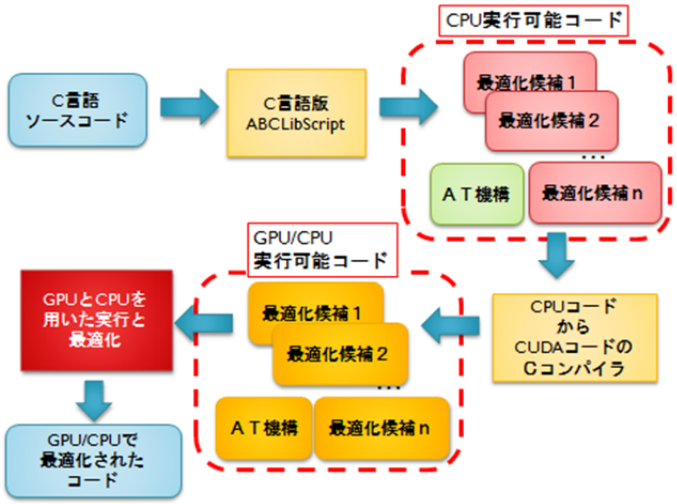

図 1　HxABCLibScript による記述プログラムの処理の流れ

2.3 拡張機能

拡張する ABCLibScript の機能（ディレクティブ）を図 2 に示す．

#pragma ABCLib allocate (<Object>)

- <Object> := {CPU | GPU | auto }
  - CPU : CPU 上での実行
  - GPU : GPU 上での実行
  - auto: CPU 上と GPU 上の双方で実行し高速な環境を選ぶ。

図 2　拡張ディレクティブとその機能

図 2 のディレクティブは，ABCLibScript による処理対象の範囲（以降，AT 領域とよぶ）に記載できる．AT 領域は，任意のプログラム中，たとえばループ中，関数呼び出し側／呼ばれる側に記載でき，ディレクティブ**#pragma ABCLib** <AT タイミング> <AT 機能> **region start** ～ **#pragma ABCLib** <AT タイミング> <AT 機能> **region end** で囲まれたプログラムの一部分である．

<AT タイミング>は FIBER 方式[8]で規定されている．インストール時，実行起動前時，実行時の 3 タイミングを記載する．<AT 機能>は大別して 3 通りあり，アンローリング機能(unroll)，パラメタ変動機能(variable)，領域選択機能(select)の 3 つが主機能である．特に，unroll 機能を指定する場合，AT 領域のコードにアンローリングを加えた複数のコードを自動生成する．かつ，それらを最適化候補として内部で関数化して，プログラムに自動付加する機能がある．variable 機能と select 機能は AT 領域のコードから新たな最適化候補を自動生成はしないが，パラメタを変動させ最適化する機能と AT 領域中の候補を自動選択する機能のコードを自動生成する．

性能モデル化の観点では，ABCLibScript は性能パラメタを自動抽出する機能，性能パラメタの値をユーザが指定する機能（ABCLib_BPset 関数），仮実行する範囲（サンプリング点）指定機能，実測データからを多項式近似し最適点を予測する機能（最小二乗法近似機能）が搭載されている．なお，デフォルトの性能モデルは全探索となっている．したがって，ユーザ知識の記述から AT 実行時間を削減することが利用の前提となっている．AT ソフトウェア開発コスト削減と，チューニング実行と検証に関する工数削減を目的に開発された言語である．

2.4 自動生成コード

先述のとおり ABCLibScript は，AT 領域に記載されているコードから所望の最適化を施したコードを関数化し，元のソースコードに自動付加する機能がある．この機能を拡張することで，図 2 の CPU と GPU の切り分け機能を実現する．

いま，AT 領域に記載されたプログラムに対し，unroll 機能が記述されているとする．この候補数を 8 とする．この時，<内部関数名>_1 ～ <内部関数名>_8 という名称がつけられ，CPU 用の最適化候補が生成される．そこで，**GPU コードが生成できるコンパイラの専用ディレクティブを自動挿入した上**で，<内部関数名>_9 ～ <内部関数名>_16 という関数を自動生成する（図 3 参照）．また，AT によるパラメタ探索範囲を 1

～8 から 1～16 に拡張する．これにより，従来の ABCLibScript の基本機能を全く損なうことなく，AT の GPU 拡張が可能となる。

```
int ABCLib_InstallMyMatMul(int n,int iusw1) {
  switch(iusw1){
    case 1:   ABCLib_InstallMyMatMul_1(n); break;
    case 2:   ABCLib_InstallMyMatMul_2(n); break;
              ….                                   } CPU 用コード
    case 8:   ABCLib_InstallMyMatMul_8(n); break;
    case 9:   ABCLib_InstallMyMatMul_9(n); break;
              …                                    } GPU 用コード
    case 16:  ABCLib_InstallMyMatMul_16(n); break;
} }
```

図 3　GPU 用に拡張した内部関数コード

図 3 の GPU コードを生成するディレクティブは，現在，複数のコンパイラで提供されており，選択に幅がある．長期的には，規格化された API を設計し，自動生成コードの汎用性を高める必要がある．ここでは HxABCLibScript が提供する AT 機能の有効性評価に限定し，以下の 2 種のコンパイラのディレクティブを直接生成する．

2.5　PGI アクセレータコンパイラ[9]による実装

The Portland Group 社のコンパイラ（PGI コンパイラ）のアクセレータコンパイラ機能で，GPU コード（NVIDIA 社の CUDA コード）が自動生成可能である．図 4 に，行列 - 行列積コードを GPU 化する場合のディレクティブ例を載せる．

```
int ABCLib_InstallMyMatMul_9 (int n) {
int i, j, k;
#pragma acc region
{ // #pragma allocate region start.
        for(i = 0 ; i < n ; i++){
          for(j = 0 ; j < n ; j++){
            for(k = 0 ; k < n ; k++){
              A[i][j] = A[i][j] + B[i][k] * C[k][j];
            } } }
} // #pragma allocate region end.   }
```

図 4　PGI アクセレータコンパイラによる行列 - 行列積コードの GPU 化

以降で用いる HxABCLibScript の PGI アクセレータコンパイラによる GPU コードには，AT 領域コードに**#pragma acc region { }** を挿入したものを想定する．そのため，AT 領域に記述できるコードは，PGI アクセレータコンパイラのコード制約[a]を受ける．

2.6　OMPCUDA による実装

大島らによって提案された OMPCUDA[10]は，OpenMP で記述されたスレッド並列化された CPU コードを，その対象領域について GPU 化（CUDA 化）できる．OMPCUDA の GPU ディレクティブは OpenMP 記述そのものであるため，特殊なディレクティブ生成が不要となるメリットがあるばかりか，GPU コードの比較対象となる CPU コードが OpenMP でスレッド並列化されているため，現在のマルチコア CPU と GPU との非均質計算機環境に向く AT 機能が提供できる．

HxABCLibScript 適用上の問題は，CPU コードを GPU コードに変換しないため，単純に生成した HxABCLibScript コードでは，CPU コードか GPU コードかが OMPCUDA 側で判断できないことがある．そこで，OMPCUDA の拡張仕様として，以下のディレクティブに続くコードのみ GPU 化するという機能追加を行う．

#pragma OMPCUDA gpu region　…(1)

ABCLibCodeGen が生成する OMPCUDA 用の GPU コードは，式（1）が自動生成される．

3.　性能評価

3.1　評価環境

本性能評価で利用する CPU は，Xeon W3520 (nehalem quad-core, 2.67GHz)，メインメモリは 12GB, OS は CentOS5.5 x86_64 である．利用した GPU は Tesla C2050 である．CUDA toolkit は version3.2, GPU ドライバは 260.19.26 である．PGI コンパイラは version 11.0-0，オプションは” -fastsse -ta=nvidia -Mlist -Minfo=accel”を指定した．

OMPCUDA のビルドには主に gcc3.4 を使っている．OMPCUDA は Omni-1.6a[11]を用いて実装されている．本実験で用いる OpenMP コンパイラは Omni を用いている．

2010 年 11 月 1 日リリースの C 言語版 ABCLibCodeGen[12]に，本機能の実装を行った．本実験は ABCLibCodeGen による自動生成コードの機能の有効性検証が目的であるため，実装上の都合からコンパイルできないコードが出力される場合は，本質的な機能を損なわない範囲で，一部を手動による変更を施している．

a PGI アクセレータコンパイラ version 11.0 の制約では，プラグマで囲む GPU コードは三角ループ（上位ループのループ導入変数を下位ループが参照するようなループ構成）や関数コールの記述ができない．

3.2 ベンチマークの詳細

(1) 行列 - 行列積

図 5 に，HxABCLibScript による行列 - 行列積の最適化記述を示す．

```
#pragma ABCLib install unroll (i,j,k) region start
#pragma ABCLib name MyMatMul
#pragma ABCLib varied (i,j,k) from 1 to 2
#pragma ABCLib allocate (auto)
        for(i = 0 ; i < n ; i++){
          for(j = 0 ; j < n ; j++){
            for(k = 0 ; k < n ; k++){
                A[i][j] = A[i][j] + B[i][k] * C[k][j];
            } } }
#pragma ABCLib install unroll (i,j,k) region end
```

図 5 行列 - 行列積のアンローリング処理の適用

図 5 では，ループ導入変数 i，j，k のそれぞれについて 2 段のアンローリングをかけるため，CPU コードで 2×2×2＝8 種のコードが生成される．それと同様に，GPU コードも 8 種生成されるため，合計 16 種の最適化候補が自動生成される．この場合の性能パラメタは，問題サイズ n となる．

(2) Jacobi 法カーネル

図 6 に，Jacobi 反復法[13]のカーネルに選択処理（select 機能）を記述した例を載せる．図 6 から，HxABCLibScript の select 機能により，AT 領域のコードに対し CPU 実行と GPU 実行の 2 種が自動生成される．この場合の性能パラメタは，問題サイズ n となる．

(3) 姫野ベンチマーク

図 7 は姫野ベンチマークの主要反復部分全体に対し select 機能を指定した例である．

なお，文献[14]で紹介されているが，最外ループ n は意味上，並列実行ができない．ユーザが直接，GPU 上で並列実行を抑制するディレクティブ（PGI アクセレータコンパイラのディレクティブでは**#pragma acc for host**）を挿入することが前提である．現在の HxABCLibScript にこのディレクティブの自動挿入の機能はない．この場合の性能パラメタは，反復回数 nn となる．

```
#pragma ABCLib static select region start
#pragma ABCLib name SelectJacobi
#pragma ABCLib allocate (auto)
#pragma ABCLib select sub region start
        for (j=0; j<n; j++) {
            for (i=0; i<n; i++) {
                uold[j][i] = u[j][i]; } }
        for (j=1; j<n-1; j++) {
            for (i=1; i<n-1; i++) {
                resid = (ax*(uold[j][i-1] + uold[j][i+1])
                        + ay*(uold[j-1][i] + uold[j+1][i])
                        + b * uold[j][i] - f[j][i])*1.d0/b;
                u[j][i] = uold[j][i] - omega * resid;
                error = error + resid*resid;
            }  }
#pragma ABCLib select sub region end
#pragma ABCLib static select region end
```

図 6 Jacobi 反復法カーネルに対する選択処理の適用

3.3 実験結果

(1) PGI コンパイラでの実行

[行列 - 行列積]

行列 - 行列積において，行列サイズ n の開始サイズを 100，終了サイズを 1000，ストライドを 100 と設定して AT を行った．なお，GPU 化カーネルにおいて，AT 領域のコードそのまま（iusw=9），k ループ 2 段アンローリング（iusw=10）は正常動作した．しかし，それ以外のコード（iusw=11~16）はコンパイル時にエラー，もしくは実行時に領域不足で実行ができなかったので CPU コードに置き換えた．図 8 に AT 実行時の実行時間ログを載せる．

図 8 から，n が 300 未満では CPU と GPU では差がない．しかし，n が大きくなるにつれ，GPU での実行（iusw=9, 10）が高速となることがわかる．n=1000 では CPU で最速の候補（iusw1=5, 0.480[秒]）に対し，GPU 上で最速の候補は（iusw1=9 0.040[秒]）であり，約 10 倍も GPU のほうが速い．なお，最適化されたパラメタの一覧を ABCLibScript はファイルで出力する．以下にそれを載せる．

以上から，行列サイズ n=300 未満で CPU 実装，行列サイズ n=300 以上で GPU 実装が選択されることがわかる．

```
float jacobi(int nn) {
  int i,j,k,n;
  float gosa, s0, ss;
#pragma ABCLib static select region start
#pragma ABCLib name SelectHimeno
#pragma ABCLib allocate (auto)
#pragma ABCLib select sub region start
    for(n=0 ; n<nn ; ++n){
      gosa = 0.0;
      for(i=1 ; i<imax-1 ; i++)
        for(j=1 ; j<jmax-1 ; j++)
          for(k=1 ; k<kmax-1 ; k++){
            s0 = a[0][i][j][k]*p[i+1][j][k]+a[1][i][j][k]*p[i][j+1][k]+a[2][i][j][k]*p[i][j][k+1]
                 +b[0][i][j][k]*(p[i+1][j+1][k]-p[i+1][j-1][k]-p[i-1][j+1][k]+p[i-1][j-1][k])
                 +b[1][i][j][k]*(p[i][j+1][k+1]-p[i][j-1][k+1]-p[i][j+1][k-1]+p[i][j-1][k-1])
                 +b[2][i][j][k]*(p[i+1][j][k+1]-p[i-1][j][k+1]-p[i+1][j][k-1]+p[i-1][j][k-1])
                 +c[0][i][j][k]*p[i-1][j][k]+c[1][i][j][k]*p[i][j-1][k]+c[2][i][j][k]*p[i][j][k-1]
                 + wrk1[i][j][k];
            ss = ( s0 * a[3][i][j][k] - p[i][j][k] ) * bnd[i][j][k];
            gosa+= ss*ss;      /* gosa= (gosa > ss*ss) ? a : b; */
            wrk2[i][j][k] = p[i][j][k] + omega * ss;
          }
      for(i=1 ; i<imax-1 ; ++i)
        for(j=1 ; j<jmax-1 ; ++j)
          for(k=1 ; k<kmax-1 ; ++k)
            p[i][j][k] = wrk2[i][j][k];
    }      /* end n loop */
#pragma ABCLib select sub region end
#pragma ABCLib static select region end
    return(gosa);
}
```

図 7　姫野ベンチマークの反復部分全体に対する選択処理の適用

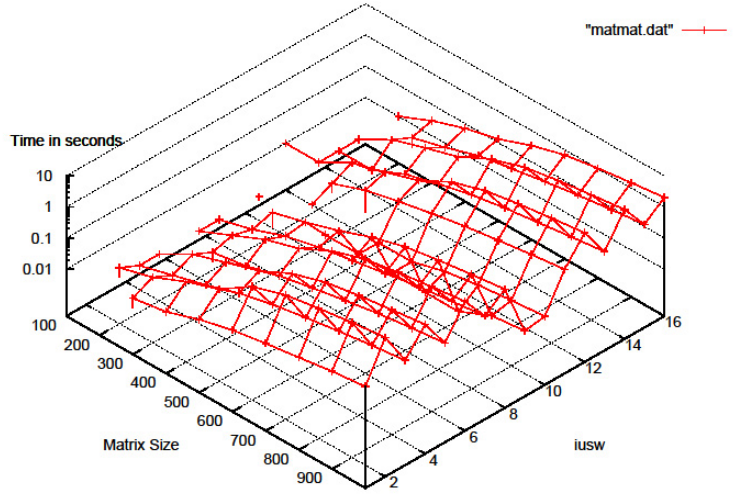

図 8 行列 - 行列コードの CPU と GPU での実行時間

```
●　行列 - 行列積の最適化済みパラメタのリスト
(MyMatMul
  (ABCLib_NUMPROCS 1)
  (ABCLib_SAMPDIST 100)
  (ABCLib_PROBSIZE 100 (MyMatMul_I 1) )
  (ABCLib_PROBSIZE 200 (MyMatMul_I 1) )
  (ABCLib_PROBSIZE 300 (MyMatMul_I 9) )
  (ABCLib_PROBSIZE 400 (MyMatMul_I 10) )
  (ABCLib_PROBSIZE 500 (MyMatMul_I 9) )
  (ABCLib_PROBSIZE 600 (MyMatMul_I 10) )
  (ABCLib_PROBSIZE 700 (MyMatMul_I 9) )
  (ABCLib_PROBSIZE 800 (MyMatMul_I 9) )
  (ABCLib_PROBSIZE 900 (MyMatMul_I 9) )
  (ABCLib_PROBSIZE 1000 (MyMatMul_I 9) )
)
```

[Jacobi 反復法カーネル]

Jacobi 法のカーネル[13]を実行する場合, 行列 - 行列積と同様の問題サイズ n の範囲では，常に CPU のほうが速くなり GPU 化の効果がなかった．これは，CPU から GPU へのデータ転送時間が大きいためである．そこで，このカーネルを呼び出す外部ループ（反復部分）を設定し，その外部ループに入る前にデータ転送を行うディレクティブ（**#pragma acc data region copy(u[0:n-1][0:n-1]) copyin(f[0:n-1][0:n-1]) local(uold[0:n-1][0:n-1]) local(resid)**）と，当該カーネル前にデータ転送をするディレクティブ（**#pragma acc region copyin(dx,dy,omega,b) copy(error)**）を挿入する場合[13]の AT 効果として実験した．なお，外部ループの反復回数は 100 回に固定している．

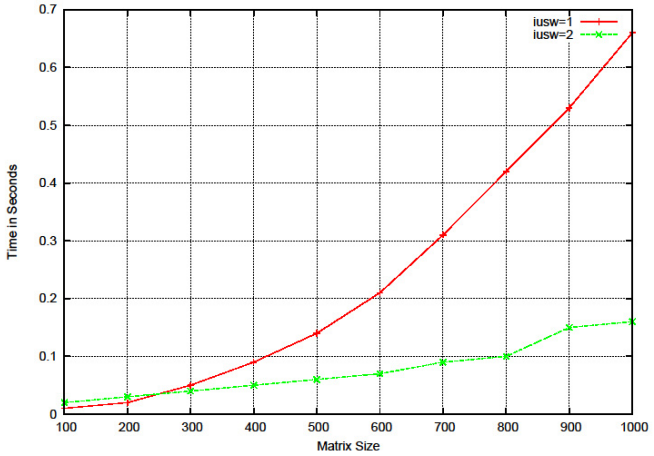

図 9　Jacobi 反復法の CPU と GPU での実行時間

図 9 から，行列サイズ n=300 未満では CPU 実行（iusw=1）が高速だが，行列サイズ n=300 以上で GPU 実行（iusw=2）が高速となる．なお，本データはデータ転送時間を含む GPU 時間を測定している点に注意する．

[姫野ベンチマーク]

姫野ベンチマークの CPU と GPU での実行時間を図 10 に載せる. ここで図 10 では，性能パラメタに反復回数を取っているので, 反復回数 nn=5 以下は CPU 実行（iusw=1）が高速，反復回数 nn＝6 以上で GPU 実行（iusw=2）が高速となる．

ところで姫野ベンチマークでは，実際の反復前に 3 反復の予備実行をして，そのあと，数百反復の本反復を行う．そこで本実験では，予備実行においては CPU，本実行では GPU にカーネルを自動切り替えされるか検証した．以下に，姫野ベンチマークの実行ログを記載する. 実行ログから, 3 反復の予備実行で CPU コードが選択され（iusw1: 1），本実行の 232 反復では GPU コードが選択される（iusw1: 2）ことがわかる．

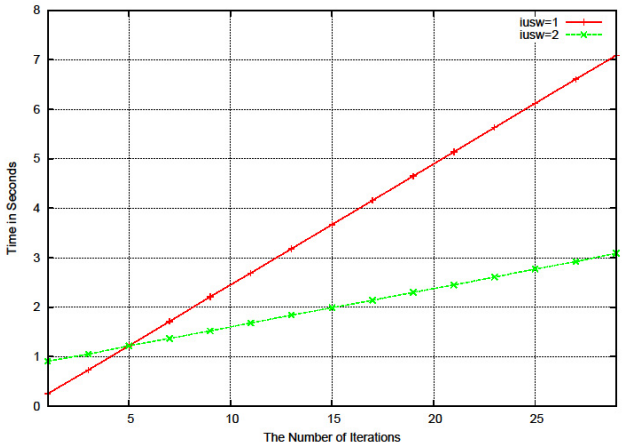

図 10　姫野ベンチマークの GPU と GPU での実行時間

● 姫野ベンチマークの実行ログ

```
mimax = 257 mjmax = 257 mkmax = 513
imax = 256 jmax = 256 kmax =512
 Start rehearsal measurement process.  Measure the performance in 3 times.
myid: 0  Static Routine: SelectHimeno  iusw1: 1
 MFLOPS: 4344.258720 time(s): 0.772542 8.578184e-04
 Now, start the actual measurement process. The loop will be executed in 232 times
 This will take about one minute. Wait for a while
myid: 0  Static Routine: SelectHimeno  iusw1: 2
 Loop executed for 232 times
 Gosa : 7.314291e-04
 MFLOPS measured : 13021.065031  cpu : 19.932327
 Score based on Pentium III 600MHz : 158.793476
```

(2) OMPCUDA コンパイラでの実行

[行列 - 行列積]

OMPCUDA の現在のバージョンの実装制約から，2 次元配列で記述した図 5 の行列 - 行列積が処理できなかった．そのため，図 11 のように，1 次元配列を用いた実装で性能評価をする．

```
#pragma ABCLib install select region start
#pragma ABCLib name MyMatMul-1D
#pragma ABCLib allocate (auto)
#pragma ABCLib select sub region start
#pragma omp parallel for private(k)
   for(i=0; i<N*N; i++){
        for(k=0; k<N; k++){
             tmp += B[(i/N)*N+k] * C[k*N+(i%N)];
        }
        A[i] = tmp;
   }
#pragma ABCLib select sub region end
#pragma ABCLib install select region end
```

図 11　行列 - 行列積の 1 次元配列実装例

図 11 では，OpenMP でスレッド並列化したコードと，OMPCUDA で GPU 化されたコードのうち最も高速であるコードが選択される．このとき，OpenMP の図 11 の最外ループ i の並列性は 4 であり，OMPCUDA での並列性は問題サイズ N と同一になるようにした．また，GPU での性能パラメタである，スレッド数とブロック数は，128 と 56 で固定した．図 12 に実行時間を載せる．

図 12 より，行列サイズ N が 100～600 までは OpenMP によるスレッド実行の CPU コード（4 スレッド）が GPU 実行より効果があるが，600 次元を超えるサイズで GPU コードのほうが高速になる．したがって，行列サイズに応じ有効なコードが CPU と GPU で異なる例であり，ABCLibScript が提供する AT 機能が有効となる．

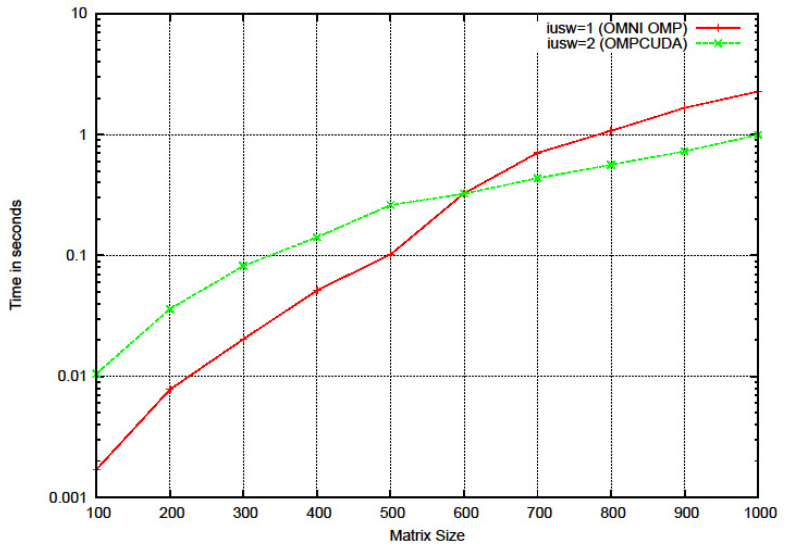

図 12　行列 - 行列積における OpenMP によるスレッド実行の CPU コードと GPU コードの実行時間

4. 関連研究

OpenMPC[15]は，OpenMP 記述から CUDA コードを自動生成するコンパイラであり，本研究で用いている OMPCUDA と機能が似ている．OpenMPC は，GPU 特有の性能パラメタであるブロック数やスレッド数を指定できる API を有するだけでなく，一部はコンパイラの機能として自動チューニングすることができる．

OpenMPC のアプローチと我々のアプローチの大きな違いは，HxABCLibScript は unroll 機能に代表される最適化候補の生成機能，および，select 機能に代表されるアルゴリズム選択機能を AT 機能として有することである．そのことで，最適化対象を広げることが目的である．また，CPU と GPU の計算資源に対し，適する計算資源上で問題サイズなどの性能パラメタの値に応じて柔軟にコードを切り替えることで最適化を行える点が異なる．

5. おわりに

本稿では，CPU と GPU の混載環境である非均質計算機向けに ABCLibScript の機能を拡張した HxABCLibScript を提案した．性能評価の結果，行列 - 行列積，Jacobi 反復法，姫野ベンチマークへの適用に対して，問題サイズや反復回数に応じ，適切に計算機資源の切り替えができることを確認した．このことからまだ予備評価に留まるが，HxABCLibScript の提供機能が非均質環境においても有効であることを確認した．

今後の課題は多数ある．

まず，GPU 向きの専用コードの自動生成がある．これは，特定の計算カーネル，たとえば，差分法カーネル（ステンシルコード）に対して，直接高性能な CUDA コードを生成する機能の追加があげられる．

次に，PGI コンパイラや OMPCUDA コンパイラに対し，GPU のブロック数やスレッド数を ATするディレクティブとその ATのためのコードの自動生成機能の開発が必要である．ブロック数やスレッド数の調整機能を有するディレクティブは，HxABCLibScript の機能として規格化し，内部で複数のコンパイラのディレクティブに対応するよう自動変換すべきである．

さらに適切な位置に，GPU への配列データ入力と出力を指定するディレクティブ，および GPU 内の共有メモリ利用のためのディレクティブを，元のコードから全自動，もしくは半自動で挿入する機能の開発が，GPU 上で実行される最適化候補を高性能化するために必須である．

謝辞 本研究は，科学技術研究費補助金，基盤研究（B），課題番号：21300007 「メニーコア・超並列時代に向けた自動チューニング記述言語の方式開発」，および，科学技術振興機構 CREST 領域「情報システムの超低消費電力化を目指した技術革新と統合化技術」，平成 19 年度採択課題，「ULP-HPC: 次世代テクノロジのモデル化・最適化による超低消費電力ハイパフォーマンスコンピューティング」による．また，東京大学須田礼仁教授から，定例の打ち合わせにおいて有益なコメントを頂いた．ここに感謝の意を表したい．

参考文献

1) Bilmes, J., Asanovic, K., Chin, C.-W. and Demmel, J.: Optimizing Matrix Multiply Using PHiPAC: a Portable, High-Performance, ANSI C Coding Methodology, Proceedings of International Conference on Supercomputing 97, 340-347, 1997.
2) Whaley, R., Petitet, A. and Dongarra, J.~J.: Automated Empirical Optimizations of Software and the ATLAS Project, Parallel Computing, 27, 3-35, 2001.
3) Frigo, M.: A Fast Fourier Transform Compiler, Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation, Atlanta, Georgia, 169-180, 1999.
4) Katagiri,T., Kise,K., Honda,H., and Yuba,T: ABCLib_DRSSED: A Parallel Eigensolver with an Auto-tuning Facility, Parallel Computing, 32, 3, 231-250, 2006.
5) Vuduc, R., Demmel,J., and Yelick,K.: OSKI:A Library of Automatically Tuned Sparse Matrix Kernels, Proceedings of SciDAC 2005, Journal of Physics: Conference Series, 2005.
6) ROSE プロジェクト, http://www.rosecompiler.org/
7) Active Harmony プロジェクト, http://www.dyninst.org/harmony/
8) Katagiri, T.,Kise,K., Honda,H., and Yuba,T.: ABCLibScript: A Directive to Support Specification of An Auto-tuning Facility for Numerical Software, Parallel Computing, 32, 1, 92-112, 2006.
9) PGI Accelerator Compilers: http://www.pgroup.com/resources/accel.htm
10) Ohshima, S., Hirasawa, S., and Honda, H.: OMPCUDA : OpenMP Execution Framework for CUDA Based on Omni OpenMP Compiler, Proceedings of International Workshop on OpenMP (IWOMP2010), June 2010.
11) Omni Compiler Project, http://www.hpcs.cs.tsukuba.ac.jp/omni-openmp/
12) C 言語版 ABCLibCodeGen, Version Trial, Release 1, 11/1/2010 http://www.abc-lib.org/format.htm
13) Softek 社による Jacobi 反復法の GPU 高速化 http://www.softek.co.jp/SPG/Pgi/TIPS/public/accel/accel-jacobi.html
14) Softek 社による姫野ベンチマークの GPU 高速化 http://www.softek.co.jp/SPG/Pgi/TIPS/public/accel/accel-himeno.html
15) Lee, S. and Eigenmann, R.: OpenMPC: Extended OpenMP Programming and Tuning for GPUs, SC10: Proceedings of the 2010 ACM/IEEE conference on Supercomputing, November 2010.