Vol.2013-OS-124 No.1
2013/2/28

SMYLE OpenCL デバイスにおける並列プログラミングモデルの実現と評価

江谷典子[†1]　稗田拓路[†1]　冨山宏之[†2]

組込みシステム向けの高性能かつ低消費電力なメニーコアシステムの実現を目指して，1)組込みシステムを意識した効率的な超並列処理の実現，2)大幅な動作時消費電力の削減，3)ソフトウェアの生産性の向上，を重要な課題として捉えて，ヘテロジニアス並列コンピューティング環境を実現するために，Khronos グループによって標準化が進められて OpenCL(Open Computing Language)環境の構築に取り組んでいる．本開発では，独立行政法人新エネルギー・産業技術総合開発機構(NEDO)のプログラム『極低電力回路・システム技術開発（グリーン IT プロジェクト)』の中で，組込みシステム向けメニーコアアーキテクチャを実現することを目指した『低消費電力メニーコア用アーキテクチャとコンパイラ技術』プロジェクトの研究により開発された FPGA を用いたメニーコアアーキテクチャ SMYLEref の評価環境を用いて，OpenCL が提供している「データ並列プログラミングモデル」と「タスク並列プログラミングモデル」を実現する．本稿では，組込みシステム向けメニーコア用 SMYLE OpenCL デバイスにおける並列プログラミングモデルの実現方法を述べ，従来の逐次処理や並列処理の処理速度と比較するために，並列ベンチマークテストの評価結果を示す.

Implementation and Evaluation of Parallel Programming Model On Device Cores in SMYLE OpenCL

NORIKO ETANI[†1]　TAKUJI HIEDA[†1]
HIROYUKI TOMIYAMA[†2]

In order to realize the embedded many-core system which performs high-speed and low-power, 1) realizing efficient massive parallel processing for embedded system, 2) reducing the large electricity consumption, 3) enhancing the software productivity are the important research themes. So, a platform for OpenCL, which standardization is promoted by Khronos Group, has been developed in order to build a heterogeneous parallel computing environment. In a program of “Extremely Low-power Circuits and Systems (Green IT Project)” sponsored by New Energy and Industrial Technology Development Organization (NEDO), a platform on FPGA in order to evaluate SMYLEref architecture for many-core processor was developed as result of the research by a project of "many-core architecture for low energy consumption and its compiler technology". On this evaluation platform, "data parallel programming model" and "task parallel programming model" provided by OpenCL are implemented on device cores for SMYLE OpenCL. This paper describes a design and its implementation of parallel programming model on SMYLE OpenCL devices, and shows an evaluation result of the parallel benchmark test compared with conventional sequential processing and parallel computation with thread code.

1. はじめに

コンピュータを高速化するため，CPU のビット幅の拡張，動作クロックの高周波化，キャッシュの大容量化などが図られてきたが，CPU クロックの高周波化にともない，消費電力や熱，ノイズ，周辺デバイスとの性能バランスの崩壊などさまざまな問題が顕在化した．そこで，チップあたりのプロセッサコア数の増加により，並列性を高めることで，プロセッサの高性能化は促進されている[1].

そこで，組込みシステム向けの高性能かつ低消費電力なメニーコアシステムの実現を目指して，1)組込みシステムを意識した効率的な超並列処理の実現，2)大幅な動作時消費電力の削減，3)ソフトウェアの生産性の向上，を重要な課題として捉えて，それらを解決する一方策として「仮想アクセラレータ(VAM: Virtual Accelerator on Many-core)」の概念を導入し，その実行プラットフォームとしてメニーコアアーキテクチャ SMYLEref とプログラム開発環境の構築を行っている[2][3][4]．この取組みの中で，アプリケーション開発のための高レベル API を策定し，ヘテロジニアス並列コンピューティング環境を実現するために，Khronos グループによって標準化が進められている OpenCL(Open Computing Language)環境の構築に取り組んでいる[5][6][7].

この OpenCL プラットフォームを構成する計算要素には，ホストと OpenCL デバイスがある．ホストとは，制御側のソフトウェアが動作する計算環境であり，通常は CPU といったプロセッサおよびそれに付随するメモリを含む．OpenCL デバイスとは，演算用のソフトウェアが動作する

†1 立命館大学総合科学技術研究機構
Research Organization of Science and Technology, Ritsumeikan University
†2 立命館大学理工学部
College of Science and Engineering, Ritsumeikan University

計算環境であり，通常は，GPU，DSP，Cell/B.E.，CPU などの複数の演算用プロセッサおよびそれらに付随するメモリを含む[8].

本開発では，独立行政法人新エネルギー・産業技術総合開発機構(NEDO)のプログラム『極低電力回路・システム技術開発（グリーン IT プロジェクト)』の中で，組込みシステム向けメニーコアアーキテクチャの実現を目指した『低消費電力メニーコア用アーキテクチャとコンパイラ技術』プロジェクト[9]の研究により開発された FPGA を用いたメニーコアアーキテクチャ SMYLEref の評価環境を用いる.

図 1 には，SMYLE OpenCL のプラットフォームモデルを示す．現在の OpenCL の仕様について，高いリアルタイム性が要求される組込みシステムにも対応できるように，独自の制約と解釈を与えた SMYLE OpenCL を検討した[6]. 本開発では，SMYLE OpenCL を動作させるために用いるデバイス側の並列プログラミングモデルの実装と評価を行う.

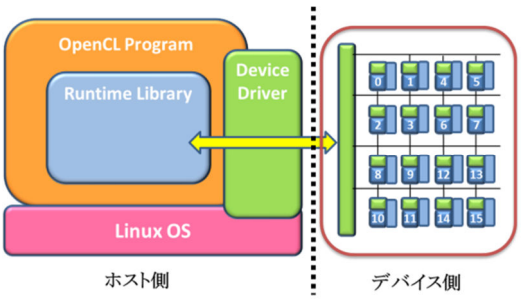

図 1　SMYLE OpenCL のプラットフォームモデル[5]

本稿では，FPGA を用いたメニーコアアーキテクチャ SMYLEref の評価環境を用いて，OpenCL が提供している並列プログラミングモデルである「データ並列」および「タスク並列」をデバイス側で実現する方法を述べ，従来の逐次プログラムやスレッドを用いた並列プログラムの処理速度と比較するために，並列ベンチマークテストの評価結果を示す.

本論文の構成を以下に述べる．まず 2 節で本開発におけるシステム構成を説明した後，3 節で SMYLE OpenCL における並列プログラミングモデルの実現について述べ，4 節でそのプログラミングモデルに則した逐次プログラムのコードおよびスレッドを用いたコードからの移行を説明する．その有効性を確認するための並列ベンチマークテストならびに結果について，5 節で示し，最後に 6 節でまとめる.

2. システム構成

2.1 開発環境

Xilinx 社製 FPGA チップである Virtex-6 を搭載する ML605 評価ボードをプラットフォームとして利用している．表 1 には ML605 評価ボード，および，表 2 には搭載する Virtex-6 チップの仕様を示す．なお，回路設計には，VerilogHDL を用い，論理合成，マッピング，配置配線には Xilinx 社の ISE を利用している[2][4].

表 1　ML605 ボードの仕様[11]

FPGA デバイス	Virtex-6 XC6VLX240T-1FFG1156
SDRAM	DDR3 SODIMM(512MB)
搭載 IO ポート	UART，USB，DVI 出力，CF，SMA 等
クロック入力	200MHz オシレータ
	66MHz ソケットオシレータ

表 2　Virtex-6 チップの仕様[12]

CMOS	40nm, 1.0V
Logic Cells	241,152
CLB Slices	37,680
Block RAM	14,976Kbit
ユーザーI/O 数	720
消費電力[13]	Static Power: 3.6W, Total Power: 6.5W

本開発では，表 3 の開発環境へターゲット OS である mips-geyser-linux[2]用クロスコンパイル環境を構築している．また，ベンチマークテストで比較評価を行う従来の逐次プログラムやスレッドを用いた並列プログラムは，この開発環境にインストールされている.

表 3　ソフトウェア開発環境

PC	Sony VAIO
CPU[14]	名称：Intel® Core™ i7-2640M プロセッサ
	動作周波数：2.80GHz
	消費電力(Max TDP)：35W
OS	Windows 7 Professional Service Pack 1
VM	VMware Player 4.0.3
HOST	32-bit Fedora 16

2.2 SMYLEref アーキテクチャの評価環境

SMYLEref のアーキテクチャを図 2 に示す.

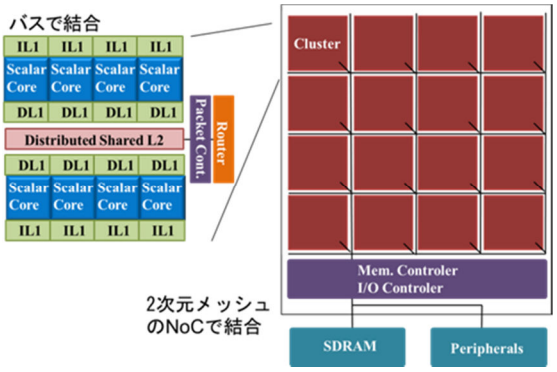

図 2　SMYLEref アーキテクチャ[2][4]

SMYLEref は，数個（図 2 の例では 8 個）のプロセッサコアをバスで結合したクラスタを，2 次元メッシュの NoC(Network on Chip)で結合したアーキテクチャを想定している[2][4]．クラスタには，コアや L2 キャッシュを備えるプロセッサクラスタと，チップ外部とのインターフェース(SDRAM コントローラなど)を持つペリフェラルクラスタがある．FPGA を用いた評価環境は，上述の 1 つのプロセッサクラスタ上で OS やシステムソフトウェアの評価・検証を可能にしている[2][4].

本評価環境では，ML605 ボードを 1 枚使用し，1 つのボード上には， 4 つのソフトコアが準備されている．図 3 では，SMYLEref アーキテクチャのコア構成を示す．1 つのコアには，マスタコアとしてメインスレッドの役割を，他のコアは，スレーブコアとしてスレーブスレッドの役割を持たせている．

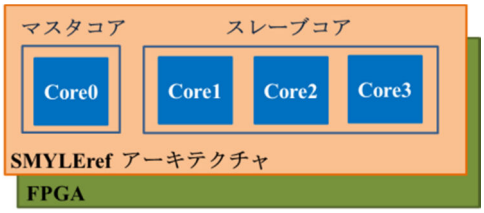

図 3　コアの構成

デバイス側の動作環境（図 4）は，SMYLEref アーキテクチャにある 4 つのソフトコアをデバイスとして用いる．アプリケーションプログラムは，マスタコアへロードされ，起動される．この時，マスタコアからスレーブコアを起動して，アプリケーションプログラムを並列実行させる．

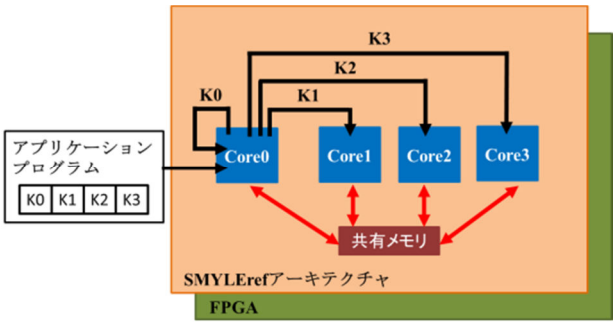

図 4　デバイスの動作環境

2.3 SMYLE OpenCL デバイスの構成

SMYLE OpenCL デバイスは，アプリケーションプログラム，SMYLE-Ref 層，IO ライブラリ，および浮動小数点ソフトウェアエミュレーションから構成される（図 5）．

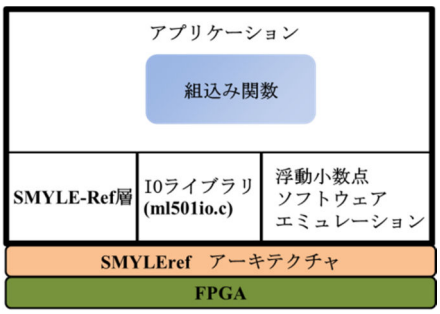

図 5　SMYLE OpenCL デバイスの構成

アプリケーションプログラムは，低レベルの制御や mutex 操作を実装する SMYLE-Ref 層と既存の IO ライブラリを利用して実装されている．SMYLE-Ref 層では，次の関数が準備されている．

- sr_mutex_init(sr_mutex *mutex)
  mutex を初期化する。
- sr_mutex_lock(sr_mutex *mutex)
  mutex をロックする。
- sr_mutex_unlock(sr_mutex *mutex)
  mutex をアンロック状態にする。

3. SMYLE OpenCL デバイスにおける並列プログラミングモデルの実現

3.1 データ並列とタスク並列

OpenCL の仕様は，図 6 で示した「データ並列プログラミングモデル」と「タスク並列プログラミングモデル」を提供している[10]．データ並列プログラミングモデルとは，1 つの実行タスクが，デバイス内の演算ユニットで，同時に動作するようなモデルである．タスク並列プログラミングモデルとは，デバイスで動作する複数の異なる実行タスクが，デバイス内の別々の演算ユニットに割り当てられて，動作するようなモデルである．

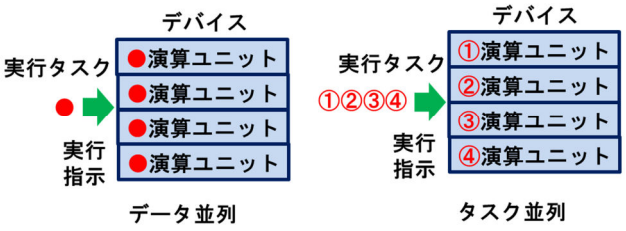

図 6　データ並列プログラミングモデル（左）と
タスク並列プログラミングモデル（右）

3.2 並列プログラミング機能

並列プログラミングは，「スレッド」「排他制御」「条件変数」の機能で実現することができる[15]．本システムでは，SMYLEref の基礎評価を目的に開発された簡易版 pthread ライブラリ（並列処理 API）[2]を利用して，デバイスコアをマスタスレッドおよびスレーブスレッドとして扱うことが

できる．ここでは，この API を利用しないで，SMYLEref 上のデバイスコアをスレッド相当として利用している．そこで，前述の並列プログラミングモデルを実現する上で，どのような機能が準備されているのかについて説明する．

(1) **デバイスコア**

データ並列の記述は，main プログラム中で，4 つのデバイスコアが実行するコードを定義する方式である．下記の例では，4 つのデバイスコアが printf 文を実行し，画面には”Hello, World!”が 4 回表示される．

```
int main()
{
      ・・・・・
      printf(“Hello, World! ¥n"); //データ並列
      ・・・・・
      exit(0);
}
```

タスク並列の記述は，main プログラム中で，各デバイスコア ID を指定して，各コア毎に実行するコードを定義する方式である．下記の例では，4 つの各デバイスコアが該当の printf 文を実行し，画面には，”core0⏎”, ”core1⏎”, ”core2⏎”,”core3⏎”と表示される．

```
int main()
{
      ・・・・・
//コア ID の取得
      sr_core_id_t my_id = sr_get_core_id();
//タスク並列
      if ( my_id == 0 ) printf(“core0¥n");
      if ( my_id == 1 ) printf(“core1¥n");
      if ( my_id == 2 ) printf(“core2¥n");
      if ( my_id == 3 ) printf(“core3¥n");
      ・・・・・
      exit(0);
}
```

デバイスコアのライフサイクル管理は，main プログラムの開始と終了が，並列プログラムの開始と終了となる．

SMYLE OpenCL デバイスでは，スケジューリングのアルゴリズムに関する規定はない．従って，デバイスコアのスケジューリングは，SMYLEref におけるソフトコアの実装に依存しており，データ並列では，確実に全デバイスコアが実行されることも厳密には保証していない．

(2) **排他制御**

排他制御を実現するためには，ロック(Lock)とクリティカルセクション(Critical Section)の概念が利用される．デバイスコア間で共有するリソースが利用される実行コードの部分をクリティカルセクションと呼ぶ．クリティカルセクションに入る前にロックを獲得し，コードの実行を行い，クリティカルセクションから出るときにロックを解放する．ロックは，同時にはただ 1 つのデバイスコアのみが獲得できる．1 つのデバイスコアがロックを獲得している時，他のデバイスコアが同じロックを獲得しようとすると，そのデバイスコアはブロックされる．ロックを獲得していたデバイスコアが，ロックを解放したときに，ブロックしていたデバイスコアが同じロックを獲得することが可能になる．

ロックを獲得および解放するためには，アプリケーションを実行する前にマスタコア部で mutex を初期化する．デバイスコアの起動は順不同である．よって，確実にマスタコアだけで mutex を初期化するために，他のコアを sleep 関数により停止させている．

```
sr_mutex_t __attribute__ ((section (".bss2"))) mutex;
int main()
{
  sr_core_id_t my_id = sr_get_core_id();
//マスタコアによる mutex 初期化
  if ( my_id == 0 )
  {
      sr_mutex_init( &mutex );
      sleep(1);
  }
  else
  {
      sleep(1);
  }
  kernel_app();  //並列アプリケーションの実行
  ・・・・・
  exit(0);
}
```

クリティカルセクションは，関数の内部のコードブロック単位で指定することができる．

```
extern sr_mutex_t mutex;
//共有リソース
int __attribute__ ((section (".bss2"))) global_counter = 0;
int kernel_app()
{
    sr_mutex_lock( &mutex );
    {
        global_counter++;
    }
    sr_mutex_unlock( &mutex );
  }
}
```

過度な排他制御によって，単にパフォーマンスの問題が生じるだけでなく，処理が完了しないという致命的な問題が発生することがある．例えば，深さ優先探索を行うためにバックトラックと再帰を用いる場合，共有している探索空間を分割して，タスク並列を行うと，共有している探索

空間へのアクセス時にデッドロックが発生する．この場合は，対象探索空間を共有して，データ並列を行うとデッドロックを回避できる．

(3) 条件変数

排他制御は，共有リソースに対するアクセスが 1 度に 1 つのデバイスコアからだけであるように制御する機能であるが，ある条件が成立するまで並列処理を待機させる機能が必要である．このような機能を実現する仕掛けを条件変数と呼ぶ. 下記の例では, 共有データに定義されている remain 変数（キューにあるデータ数） が 0 である場合には，while 文によるループ待機を行い，キューにデータが入力されるまで，現在のデバイスコアを待機させる．

```
//キューから 1 つ取り出す
while (1) {
        while(remain == 0){}      //データがないので待機
        sr_mutex_lock( &mutex );
        { // !!!CRITICAL SECTION!!!
          i = queue[rp];
          rp++;
          remain--;
          if(rp == MAX_QUEUE_NUM) rp = 0;
        }
        sr_mutex_unlock( &mutex );
        if(i == END_DATA) break;
}
```

条件の判定の際に，if ではなく while を使用する．並列プログラミングでは，他のデバイスコアが同時に実行を行っているので，例えば，現在のデバイスコアが待機からリターンするために，条件を常に確認する必要がある．そのために，条件の成立の判定は，while 条件文を用いて，条件が成立するまでループ待機を行う．

3.3 プログラムブロック間の同期処理

逐次プログラムでは制御が移動する順に処理が行われ，現在処理中のプログラムが終了しない限り，次の処理には移動しない．ところが，並列化されたプログラムでは複数のブロックが並列に動作するため，処理順に依存性がある場合はそれらを制御しなければならない．処理順にルールがある場合，先に行う処理の終了を確実に待つために，逐次プログラムでは不要な同期処理が必要になる．SMYLE OpenCL では, 組込み関数として「同期関数(バリア関数)」を準備している[6].

4. 並列化コードへの移行

4.1 逐次プログラムのコード

while 文，for 文などのループ処理は，無限ループへ変更し，while 文や for 文の条件式の処理を分割するように設計を行ことで，デバイスコアの並列化を容易にする。

```
//逐次処理
int linear_search(int x, int *a, int num)
{       int n = 0; //配列の範囲内で目的の値を探す
  while(n < num && a[n] != x) {
        n++;
  }
  if(n < num) {
      return n;
  }
  return NOT_FOUND;
}
```

また，デバイスコアでは，カウンタ値や共有データを直接操作する場合は，排他制御を行う．

```
int LinearSearch0()
{
  sr_core_id_t my_id;
  n0 = 0;
  //配列の範囲内で目的の値を探す
  while(1){
    sr_mutex_lock( &mutex );
    if(n0 < DATA_NUM && data0[n0] != key) {
        n0++;
    } else {
      sr_mutex_unlock( &mutex );
      break;
    }
    sr_mutex_unlock( &mutex );
  }
  sr_mutex_lock( &mutex );
  if(n0 < DATA_NUM) {
      printf("data0:[%d](%d)=%d¥n",n0, data0[n0], key);
  }else printf("no-data0¥n");
  sr_mutex_unlock( &mutex );
}
```

4.2 スレッドを使ったコード

SMYLE OpenCL デバイスでは，プログラムの main 部分は，「データ並列」モデルが実行される．下記のスレッドを使ったコードでは，「スレッドの生成」を定義しているが，該当部分は不要となる．

```
for (i = 0; i < NUM_THREAD; i++)
{
//スレッド関数の引数データの初期化
    parg[i].thread_no = i;
    parg[i].thread_data = data;
//スレッドの生成
    pthread_create(&handle[i], NULL, (void *)thread_func, (void *)&parg[i]);
}
```

5. 評価

SMYLE OpenCL デバイスの並列プログラミングモデルを実現する方法を用いて，32-bit Fedora 16 で動作する並列プログラム 3 項目[16]および逐次処理プログラム 9 項目[17][18]を移植した．また，その処理速度を計測し，これら 12 項目の従来のプログラムと比較を行った．

5.1 ベンチマークの仕様

(1) **加算 1(Addition1)**
データ数 12 のデータを順次加算し，総計を表示するプログラム．

(2) **加算 2(Addition2)**
上記のプログラムについて，データ数 12 のデータを 4 分割し，分割単位でデータを加算し，最後に総計を演算して表示するプログラム．

(3) **生産者/消費者(P-C)**
2 つの生産者スレッドがキューにデータを最大 10 個まで追加し，2 つの消費者スレッドがキューにデータがあれば取り出すプログラム．

(4) **リニアサーチ 1(LS1)**
リストの先頭から終端に向かって目的の要素を探し出すアルゴリズム．ここでは，サイズ 25 の 1 次元配列を 4 つ準備し，この 4 つ目の配列にある最終位置に目的の要素がある場合．

(5) **リニアサーチ 2(LS2)**
上記(4)のアルゴリズムを使う．100 バイトあるデータの最終位置に目的の要素がある場合．

(6) **リニアサーチ 3(LS3)**
上記(4)のアルゴリズムを使う．ここでは，サイズ 50 の 1 次元配列を 4 つ準備し，この 4 つ目の配列にある最終位置に目的の要素がある場合．

(7) **リニアサーチ 4(LS4)**
上記(4)のアルゴリズムを使う．200 バイトのデータ最終位置に目的の要素がある場合．

(8) **数独 1(Sudoku1)**
9×9 のマス目において，すべての縦の列，横の列，3×3 のブロックに 1 から 9 までの数字をひとつずつ入れていくパズルのプログラム．解が 1 つの場合．

(9) **数独 2(Sudoku2)**
上記(8)のアルゴリズムを使って，解が 2 つの場合．

(10) **バブルソート(BS)**
データ数 100 を対象に，隣り合う要素の大小を比較しながら整列させるアルゴリズム．

(11) **クイックソート(QS)**
データ数 100 のデータの集合を基準値（先頭の値とする）より大きいものと小さいものとのグループに分け，それぞれのグループの中でも新しい基準値を使って同様の作業を行うという再帰処理により，データを順番に並び替えるアルゴリズム．

(12) **マージソート(MS)**
データ数 200 のデータを対象に，ブロックを前半と後半に分けてソートを行い，2 つのソートされたデータ列のマージを行うアルゴリズム．

5.2 処理速度の計測結果

前述の 12 の並列アプリケーションベンチマークを，下記のプログラミングモデルにおいて実施し，その処理速度を計測した．

(1) **sequential**
32-bit Fedora 16 で動作する逐次プログラム．

(2) **thread-task**
32-bit Fedora 16 で動作するスレッドを使ったタスク並列プログラム．

(3) **thread-data**
32-bit Fedora 16 で動作するスレッドを使ったデータ並列プログラム．

(4) **SMYLE-task**
SMYLE OpenCL デバイスで動作するタスク並列プログラム．

(5) **SMYLE-data**
SMYLE OpenCL デバイスで動作するデータ並列プログラム．

(6) **SMYLE-seq**
SMYLE OpenCL マスタコアのみによる逐次プログラム．

次に，SMYLE OpenCL デバイスにおける並列処理と処理速度を比較するために，ベンチマークテストの結果を示す．

図 7 では，ベンチマーク(1)(2)(3)を使って，スレッドコードを利用した並列処理と比較を行う．SMYLE は，並列処理におけるオーバーヘッドが小さいため，従来の並列処理の 2 倍以上の速さである．

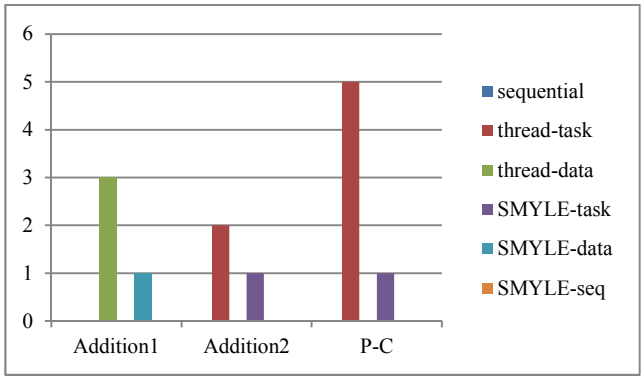

図 7 スレッドコードとの比較（単位：micro sec）

図 8 では，ベンチマーク(4)(5)(6)(7)を使って，リニアサーチの逐次プログラムと比較を行う．SMYLE は，データ分割の有無やプログラミングモデルの種類による処理速度

の差がなく，逐次プログラムとの処理速度にも差はない．

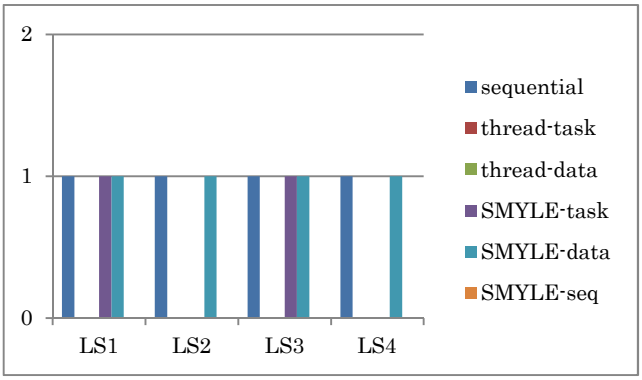

図 8　リニアサーチの逐次プログラムとの比較
（単位: micro sec）

図 9 では，ベンチマーク(8)(9)を使って，バックトラックや再帰処理を用いた逐次プログラムと比較を行う．SMYLE は，デッドロック回避のため，排他制御を使わず，データ並列プログラミングモデルを利用した．解が 1 つの場合，SMYLE の方が速い．解が 2 つの場合は，SMYLE は約 2 倍の速さである．

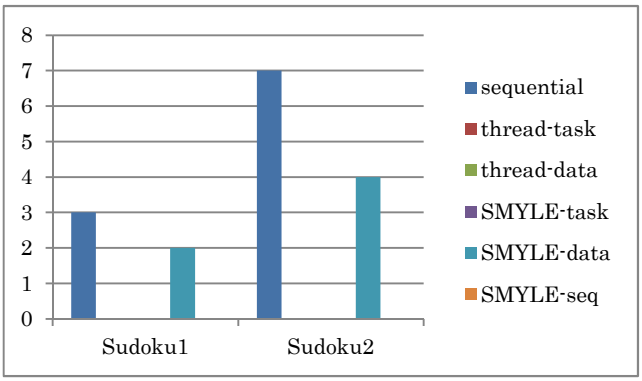

図 9　バックトラックと再帰処理を用いた逐次プログラムとの比較（単位: micro sec）

図 10 では，ベンチマーク(10)(11)(12)を使って，3 種類のソートの逐次プログラムと比較を行う．データ並列プログラミングモデルを用いた SMYLE は，逐次プログラムよりも遅くなる．これは，排他制御によるデッドロックやバッファ間のデータ転送の手間が起因していると考える．まず，ベンチマーク(10)(11)について，SMYLE では，排他制御によるロック回数が，(10)では 10576 回，(11)では 4 回であることを計測した．この排他制御によるデッドロックが発生したために，逐次プログラムよりも遅くなったと考える．そこで，SMYLE のマスタコアだけによる逐次処理を計測してみると，ベンチマーク(10)(11)は，ほぼ逐次プログラムと同じ処理速度である．次に，ベンチマーク(10)(11)(12)のバッファ間のデータ転送について，プログラムの記述では，(10)は同じバッファ内で 1 回，(11)は同じバッファ内で 4 回，(12)は異なるバッファ間で 4 回行われている．よって，ベンチマーク(12)について，SMYLE は，データ並列でもマスタコアの逐次処理でも，従来の逐次処理よりも遅いのは，バッファ間のデータ転送の手間により発生した遅延によるものと考える．

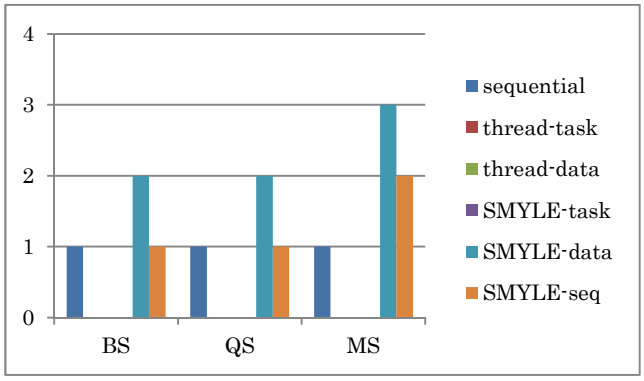

図 10　ソートの逐次プログラムとの比較（単位: micro sec）

6. まとめ

本稿では，組込みシステム向けメニーコアシステム用 SMYLE OpenCL デバイスにおける並列プログラミングモデルである「データ並列」および「タスク並列」を実現する方法について述べた．また，逐次プログラムやスレッドを用いた並列プログラムを SMYLE OpenCL デバイスの並列プログラムへ移植し，処理速度を計測するベンチマークテストとして実施した．その結果，SMYLE OpenCL デバイスでは，スレッドを用いた並列プログラムよりも速く，逐次プログラムとほぼ同じ速度，あるいは，より速いことを確認した．

今後は，ハードウェア／ソフトウェア開発向けの統合設計自動化ツール[19]を用いて，演算負荷の重い部分を見つけ出し，その部分を FPGA にハードウェアとして実装することで，より高速な SMYLE OpenCL デバイスの並列処理を目指したい．

謝辞　本研究は，独立行政法人新エネルギー・産業技術総合開発機構(NEDO)の委託により実施した．

参考文献

1) 電子・情報技術分野の技術ロードマップ 2009 解説，Ⅲ．コンピュータ技術分野，独立行政法人新エネルギー・産業技術総合開発機構(NEDO)，
http://www.nedo.go.jp/content/100085064.pdf
2) グェンチュオンソン，レイジャオ，近藤正章，平尾智也，井上弘士: FPGA を用いたメニーコア・アーキテクチャ SMYLEref の評価環境の構築，情報処理学会研究報告，Vol. 2012-ARC-198, No. 15, pp.1-7, 2012.
3) Koji Inoue: SMYLE Project: Toward High-Performance, Low-Power

Computing on Manycore-Processor SoCs, ASP-DAC2013 Proceedings of the 18th Asia and South Pacific Design Automation Conference, pp.558-560, IEEE, ISBN: 9784673-3028-2, 2013.
4) Masaaki Kondo, Son Truong Nguyen, Tomoya Hirao, Takeshi Soga, Hiroshi Sasaki, Koji Inoue: SMYLEref: A Reference Architecture for Manycore-Processor SoCs, ASP-DAC2013 Proceedings of the 18th Asia and South Pacific Design Automation Conference, pp.561-564, IEEE, ISBN: 9784673-3028-2, 2013.
5) 稗田 拓路，西山 直樹，谷口 一徹，冨山 宏之，井上 弘士: 組込みシステム向けメニーコア用 OpenCL 環境，情報処理学会研究報告，Vol. 2012-SLDM-155, No. 2, pp.1-6, 2012.
6) 江谷 典子，稗田 拓路，冨山 宏之: SMYLE OpenCL における組込み関数の開発と評価，情報処理学会研究報告，Vol. 2012-OS-123, No. 7, Vol.2012-EMB-27, No. 7, pp.1-8, 2012.
7) Hiroyuki Tomiyama, Takuji Hieda, Naoki Nishiyama, Noriko Etani, Ittetsu Taniguchi: SMYLE OpenCL: A Programming Framework for Embedded Many-core SoCs, ASP-DAC2013 Proceedings of the 18th Asia and South Pacific Design Automation Conference, pp.565-567, IEEE, ISBN: 9784673-3028-2, 2013.
8) 株式会社フィックスターズ: 改訂新版 OpenCL 入門 1.2 対応，株式会社インプレスジャパン，ISBN978-4-8443-3172-8, 2012.
9) 極低電力回路・システム技術開発（グリーン IT プロジェクト）実施方針：平成 24 年度,独立行政法人新エネルギー・産業技術総合開発機構(NEDO) ,
http://www.nedo.go.jp/content/100490844.pdf
10) Khronos OpenCL Working Group: The OpenCL Specification Version 1.2, 2012 ,
http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf
11) Xilinx Inc.: Getting Started with the Xilinx Virtex-6 FPGA ML605 Evaluation Kit, UG (v1.4) November 15, 2010.
12) Virtex-6 Family Overview, DS150(V2.4) January 19, 2012,
http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf
13) 7 シリーズ FPGA で消費電力を半減,
http://japan.xilinx.com/publications/archives/xcell/issue75-76/p12-19.pdf
14) Intel® Core ™ i7-2640 M Processor Specifications,
http://ark.intel.com/products/53464/Intel-Core-i7-2640M-Processor-4M-Cache-up-to-3_50-GHz
15) A. D. Birrell, An Introduction to Programming with Threads, SRC Research Report 35, 1989 ,
http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-35.pdf
16) 安田 絹子, 飯塚 博道, 青柳 信吾, 小林 林広, 阿部 貴之: マルチコア CPU のための並列プログラミング―並列処理&マルチスレッド入門，秀和システム，ISBN 978-4-7980-1462-3, 2006 ,
http://www.shuwasystem.co.jp/books/7980/1462-1/1462-1.html
17) 紀平 拓男, 春日 伸弥: プログラミングの宝箱 アルゴリズムとデータ構造 第 2 版，ソフトバンククリエイティブ，ISBN 978-4-7973-6328-9, 2011 ,
http://isbn.sbcr.jp/63289/
18) 数独を解くプログラム,
http://www.geocities.jp/motonaga_asao/Sudoku/Sudoku.html
19) アルテラ OpenCL 向けソフトウェア開発キット, 2012,
11-06-12_OpenCL SDK Press Release_Final(j).pdf,
http://monoist.atmarkit.co.jp/mn/articles/1211/07/news034.html