POSIX スレッドを用いた Cell プロセッサ向け API の提案

町田智志[†1] 中西悠[†1,*1]
平澤将一[†1,†2] 本多弘樹[†1,†2]

Cell Broadband Engine(CBE) は，その高性能計算能力から注目を集めている．しかし，Cell プロセッサの性能を引き出すプログラムを作成するためには，Cell プロセッサ向けに用意された API を用いて，Cell プロセッサ特有の制御処理を記述する必要があり，プログラマの負担となる．そこで本研究では，POSIX スレッドで記述したソースコードを Cell プロセッサ向けに変換するツールを作成し，評価を行った．その結果，プログラマが Cell プロセッサの制御処理を意識することなく，POSIX スレッドによるソースコードを記述するだけで，Cell プロセッサの性能を活用した PPE/SPE ソースコードが得られることが確認できた．

POSIX Thread API for Cell Processor

Satoshi Machida,[†1] Yu Nakanishi,[†1,*1] Shoichi Hirasawa[†1,†2]
and Hiroki Honda[†1,†2]

Cell Broadband Engine(CBE) with high efficiency computing power attracts attention. However, to draw the performance of Cell processor, a program must be described with API prepared for for Cell processor. In addition, it burdens programmers because the API is a thing peculiar to a Cell processor. In this paper, we developed a tool to convert the source code that was described in a POSIX thread into for a Cell processor and evaluated it. Experimental results show that the proposed tool enables programmers to create PPE/SPE source codes for Cell processor easily without discriptions to control Cell processor.

1. はじめに

Cell Broadband Engine[1)2)] は，汎用プロセッサコアである Power Processor Element(PPE) と，複数の計算専用プロセッサコア Synergistic Processor Element(SPE) を同一チップに搭載する異種混合アーキテクチャのマルチコアプロセッサであり，その高性能計算能力から注目を集めている．Cell プロセッサの技術を用いたメディアプロセッサ Spurs Engine も発表されており，Cell の計算能力を利用できる環境が整いつつある．

Cell プロセッサを用いた並列プログラムを開発する場合，プログラマは PPE 用と SPE 用のソースコードを記述する．そして，PPE 用のソースコード中で，SPE を利用する処理を明示的に記述することで，PPE と SPE の両方を用いたプログラムを開発することが出来る．

SPE を利用するためのライブラリとして，東芝の SPE Runtime Environment(SPERE)[5)]，IBM の SPE Runtime Management Library(libspe)[3)] を用いることができる．しかし，プログラマは実際に行いたい計算処理だけでなく，SPE の管理や PPE-SPE 間のデータ転送など，Cell 特有の制御処理を意識しながら記述する必要がある．

本研究では，POSIX スレッドの API を用いて記述したソースコードを，PPE/SPE のソースコードに変換する方式を提案する．これによりプログラマは，POSIX スレッドの記述方式のみを利用することで Cell の並列プログラムを開発することができるため，プログラマの負担を減らすことが出来る．

これまでに，POSIX スレッドによるソースコードを Cell プロセッサ向けのソースコードに変換するツールの基本部分を実装した．また，その変換ツールを利用して評価用プログラムを変換し，評価を行った．

†1 電気通信大学
The University of Electro-Communications
*1 現在，株式会社東芝 セミコンダクター社
Presently with TOSHIBA Corporation Semiconductor Company
†2 独立行政法人科学技術振興機構, CREST
Japan Science and Technology Agency,CREST

2. 背　　景

2.1 Cell Broadband Engine のアーキテクチャ

Cell broadband Engine は異なる種類のプロセッサコアを同一チップ上に搭載する異種混合のアーキテクチャで構成されているマルチコアプロセッサである．4GHz 動作時における単精度浮動小数点演算のピーク性能は，プロセッサ全体で 296GFLOPS であり，同じ動作周波数における Pentium4 の 16GFLOPS に比べて 18.5 倍と高性能である．

Cell が搭載するプロセッサコアのうち，PPE は PowerPC アーキテクチャで実装されたコアである．主にプログラムやシステム全体の制御を行い，通常のプロセッサと同様に用いられる．SPE は独自アーキテクチャで実装され，主に多量のデータを処理する計算専用コアとして用いられる．

Cell にはメインメモリとして XDR DRAM が接続されている．PPE は直接メインメモリにアクセスすることが出来るが，SPE はメインメモリに直接アクセスすることが出来ない．また，SPE はそれぞれ，メインメモリとは独立したアドレス空間を持つ 256KB のメモリ (Local Storage; LS) を有し，この LS に対しては直接アクセスすることができるが，PPE は LS に直接アクセスすることが出来ない．従って，必要なデータはメインメモリ-LS 間で，プログラマの指示において明示的に DMA 転送処理を行う必要がある．

Cell 上で動作するプログラムは PPE 用と SPE 用の 2 種類が存在する．SPE 用のプログラムは，PPE 用プログラムから SPE の LS に送り込まれた後に実行される．そのため SPE 用のプログラムは，単独で実行されることはなく，常に PPE 用プログラムの制御下で動作する．

2.2 Cell プロセッサ向けのプログラム開発

プログラマが Cell のためのプログラムを作成する場合，PPE 上でのみ動作するプログラムは，PowerPC 向けの開発環境を用いることで，従来通りにプログラムを開発することができる．

しかし SPE も利用するプログラムを作成するためには，一般的には SPE のランタイムライブラリを導入し，そのライブラリの API を用いてソースコードを記述することになる．SPE ライブラリには，東芝の Cell 評価用環境である Cell Refernece Set 上で用いられている SPE Runtime Environment と，主に PlayStation3 上の開発環境で用いられている SPE Runtime Management Library が存在する．実装の差異はあるが，いずれも SPE を起動するための API や，PPE-SPE 間で通信をするための API などが定義されており，PPE のソースコード中でこれらの API を用いることで SPE を制御する．

メインメモリと LS のアドレス空間はお互いに独立しているため，PPE-SPE 間のデータ転送は以下の要領で行う．PPE ソースコードにおいては，転送したいデータのメインメモリにおけるアドレスとサイズを SPE に通知し，SPE ソースコードでは，DMA 転送のための API を通じて，通知されたアドレスに対して，指定サイズ分だけデータを受信または送信の指示を出す．

これらランタイムを用いて開発された PPE/SPE プログラムの，処理の大まかなフローは次のようになる．

(1) SPE プログラムを LS に読み込み，起動する
(2) SPE プログラムがメインメモリから LS にデータを読み込む
(3) SPE が計算を実行する
(4) SPE が結果をメインメモリに送り返す
(5) SPE プログラムを終了させる

このように，実際の計算処理以外に，Cell のアーキテクチャ特有の制御処理を意識しながらコーディングを行う必要があり，これらはプログラマの負担となる．

3. ソースコード変換ツール

ユーザが提案方式を利用して Cell プロセッサ向けプログラムを開発する場合，まず POSIX スレッドを用いて変換元ソースコードを作成した後，この変換ツールを用いて，PPE/SPE の各ソースコードを生成する．その後，PPE/SPE ソースコードをそれぞれコンパイルし，Cell 上で動作する実行ファイルを得る．

3.1 変換元ソースコード

図 1 に，プログラマが作成する POSIX スレッドによるソースコードイメージを示す．

`main()` 関数を含む主処理には，計算に必要なデータの初期化や，スレッドに対してデータの分配，各スレッドが計算した結果を集計する処理などを記述して行く．また，スレッド関数（図 1 では `thread_func()`）を呼び出す処理や，スレッドの終了を待ち合わせる処理など，スレッドを管理する処理もここに記述する．

スレッド関数には，並列に実行したい計算処理を記述して行く．スレッドとメインルーチンのデータのやりとりは引数に渡される構造体を通じて行うように記述する．各スレッドは，配分された構造体中のデータを元に計算を実行し，その結果データを構造体に書き戻すことで，メインルーチンに通知する．

```
void thread_func(void* arg){
  thread_data_t* t_arg =
                (thread_data_t *)arg;
  計算処理;
  構造体にデータを格納;
}

int main(void){
   /* データのやりとりに用いる構造体 */
  thread_data_t t_arg;
  pthread_t handle;

  計算データの初期化処理;
  pthread_create(&handle, NULL,
       thread_func, &t_arg);
  pthread_join(handle, NULL);
  結果の集計;
  終了処理;
}
```

図 1 POSIX スレッドを用いたソースコードのイメージ

```
void thread_func(void* arg){
  SPEプログラムイメージのロード;
  SPEプログラムの実行(arg);
  SPEプログラムが終了するまで待機;
  終了処理;
}

int main(){
  /* データのやりとりに用いる構造体 */
  thread_data_t t_arg;
  pthread_t handle;

  計算データの初期化処理;
  pthread_create(&handle, NULL,
       thread_func, &t_arg);
  pthread_join(handle, NULL);
  結果の集計;
  終了処理;
}
```

図 2 変換後の PPE ソースコードのイメージ

3.2 ソースコード変換ツールの機能と構成

変換ツールで生成される PPE/SPE ソースコードのイメージを図 2 と図 3 に示す．変換ツールの実装にあたり，言語は Java を用いた．入力された POSIX スレッドによるソースコードの解析は，String クラスで定義された検索や置換のメソッドを用いた簡単な文字列処理によって実装した．

3.2.1 PPE ソースコードの生成

POSIX スレッドによるソースコードのうち，`main()` 関数のコードは，PPE で実行される記述として切り出される．変換ツールはこの主処理の記述から呼び出されるスレッド関数の代わりに，SPE プログラムを起動するスレッド関数の定義の記述を新たに生成し，あわせて PPE ソースコードとして出力する．その結果，POSIX スレッドによるソースコード中において，計算処理が記述されたスレッド関数を呼び出していた主処理の記述は，PPE ソースコード中では，SPE プログラムの起動スレッドを呼び出す役割を果たすようになる．このことにより，SPE プログラムの起動を行う記述を POSIX スレッドによるソースコードから隠蔽する．

また，上記で生成される SPE プログラムの起動スレッドには，スレッド関数の引数として渡された構造体のメインメモリ上におけるアドレスを SPE に通知する処理も記述されている．SPE プログラムは，このアドレスを元に，計算に必要なデータとその結果の転送を行う．このことにより，変換ツールは Cell 上での PPE-SPE 間のデータのやり取りを，POSIX スレッドによるソースコード上におけるメインルーチンとスレッド間のデータのやり取りと対応させてデータの転送処理の記述を隠蔽する．

```
thread_data_t spe_arg;

int main(arg){
  DMA転送：argを元に計算データを
          spe_argに受信;
  thread_data_t* spe_arg_p = &spe_arg;
  spe_arg_pを元に計算処理;
  構造体に結果データを格納;
  DMA転送：結果データの送信;
}
```

図 3 変換後の SPE ソースコードのイメージ

3.2.2 SPE ソースコードの生成

POSIX スレッドによるソースコードのうち，スレッド関数に記述された計算処理は，SPE で実行される記述として切り出される．この計算処理では，スレッド関数の引数として渡された構造体のポインタを利用している．SPE 上では LS 上の変数を指すポインタとして振舞うため，計算処理の記述の前に LS へ必要なデータの転送を行い，かつ計算処理部分でデータが利用できるよう，転送された構造体を指し示すポインタを宣言する記述を生成し，挿入する．

変換ツールは，計算に必要なデータの転送の記述も生成する．変換ツールが生成した PPE ソースコードにおいて，SPE プログラム起動時の引数としてメインメモリ上における構造体のアドレスが渡されている．

そのアドレスを元に，メインメモリから DMA 転送によってデータを LS に取得する処理が追記される．更に計算終了後に結果をメインメモリに転送する処理が追記される．

3.2.3 ヘッダファイルの生成

PPE/SPE ソースコードで共通に用いられる変数やマクロ定義などは，ヘッダファイルとして生成される．データ転送に用いる構造体定義も，ここに記述されている PPE/SPE のソースコードには，このヘッダファイルをインクルードする処理も追記される．

4. 評　　価

4.1 評 価 環 境

本研究で評価のために作成したプログラムは，PlayStation3(CECH-A00) 上で動作させた．PlayStation3 に実装されている Cell プロセッサの動作周波数は 3.2GHz であり，ユーザアプリケーションは最大で 6 基の SPE を利用することができる．メインメモリは 3.2GHz 動作の 256MB XDR DRAM である．OS は Linux 2.6.16（PowerPC 版 Fedora Core 5），コンパイラは ppu-gcc 4.1.1, spu-gcc 4.1.1 を用いた．SPE の動作ライブラリとして SPE Runtime Management Library Version 2 を用いた．

4.2 評価用プログラム

本研究では変換ツールの評価のため，N クイーン問題の解求プログラムを作成した．このプログラムはバックトラッキングのアルゴリズムで実装されている．関数の再帰呼び出しは用いていない．スレッド化に際しては，初手の駒を N 列目のどこに配置するかによって，N 個のスレッドを分割した．

計算結果データの扱いについて，各スレッドで求められた解の個数のみを求めるプログラムと，個数だけでなく，駒の配置パターンも全て集計するプログラムの 2 種類を作成し，評価に用いた．比較に用いたソースコードは，逐次処理で求めるもの，POSIX スレッドで並列動作するもの，その POSIX スレッドによるコードを変換ツールで PPE/SPE 向けに変換したもの，筆者が PPE/SPE 向けに人手でコーディングしたものの 4 つである．

全てのソースコードで，コンパイルオプションにおいて，最適化オプションは O3 とした．これらのプログラムを Cell プロセッサ上で動作させ，実行時間を計測した．なお，逐次処理のプログラムと POSIX スレッドによるプログラムは PPE のみを利用して動作し，Cell 向けのソースコード 2 種は，PPE と SPE の両方を利用して動作する．

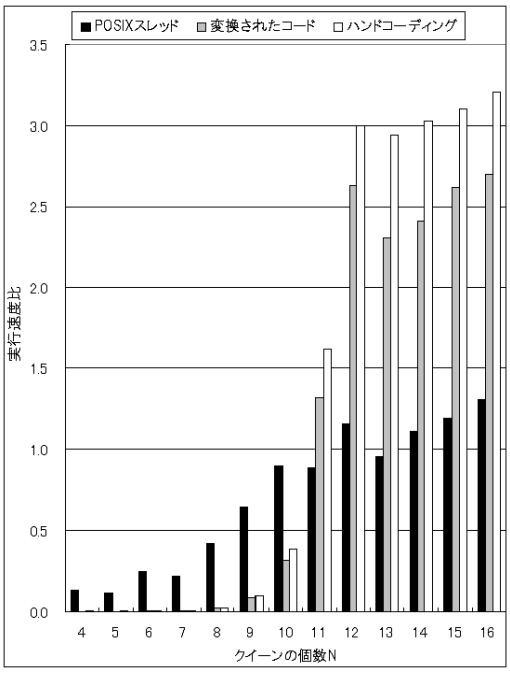

図 4　実行速度比：解の個数のみを求めるプログラム

4.3 評 価 結 果

解の個数のみを求めるプログラムを実行した結果を図 4 に，また，解の個数だけでなく，求められた駒の配置も集計するプログラムのを実行した結果を図 5 に示す．このグラフの横軸は N クイーン問題における駒の個数 N で，縦軸は逐次プログラムの実行速度を 1 としたときの実行速度比である．

解パターンも集計するプログラムについては，$N = 12$ 以上では実行できなかったため，この結果を上限とした．これは，SPE で求められた結果が大きく，LS の 256KB 以内の容量に保存することが出来なくなったためである．現在の変換ツールの実装で生成される PPE/SPE のソースコードでは，SPE プログラムの起動時に入力データを 1 回取得し，SPE での計算終了時に結果データを 1 回出力するような記述になっているためである．現在の変換ツールの実装では，大きなデータの扱いに課題が残った．

4.3.1 コーディング

いずれのプログラムも，実装にあたってプログラマがコーディングしたのは POSIX スレッドによるソースコードだけである．POSIX スレッドのソースコードは PPE 上でマルチスレッドのプログラムとして実行することもでき，変換ツールを用いて変換した後，PPE/SPE 上で実行することもできることが確認できた．このことから，この変換ツールを用いることで，

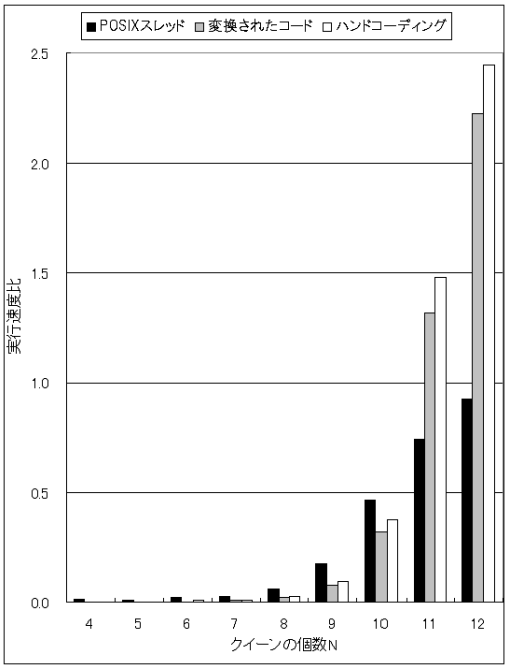

図 5 実行速度比：解の個数と，求められた解を集計するプログラム

プログラマは Cell プログラミングに特有の記述を意識することなく，POSIX スレッドによるソースコードから，PPE/SPE 向けのソースコードが作成できることが確認できた．

4.3.2 実 行 速 度

続いて，変換後のプログラムの実行速度について述べる．個数のみを求めるプログラムと，解を集計するプログラムのいずれも，変換後の PPE/SPE ソースコードは $N = 10$ までは逐次処理，POSIX スレッドによるプログラムよりも低速で動作し，PPE/SPE 向けに変換したことの恩恵を得られていない．これは計算量が SPE プログラム呼び出しや PPE-SPE 間の通信などのオーバヘッドに比べて少なく，結果としてプログラム全体の実行時間が大きくなってしまったためだと考えられる．$N = 11$ 以降は逐次処理，POSIX スレッドによるプログラムより高速に動作し，PPE/SPE 向けに変換した効果が得られた．

変換後の PPE/SPE ソースコードは，人手によりコーディングした PPE/SPE ソースコードよりも高速に動作させることは出来なかった．しかし，人手によるコーディングに対して平均 86%の速度が得られており，速度性能を大きく損なうことなく，ある程度の速度が得られている．

以上のことから，提案方式を用いることで，POSIX スレッドによるソースコードから，Cell の PPE/SPE を活用したソースコードを生成でき，人手によるコーディングに対しても，速度性能を大きく損なうことの無いことが確認できた．

5. ま と め

本論文では，POSIX スレッドで記述されたソースコードから Cell プロセッサ向けの PPE/SPE ソースコードに変換する手法を提案し，変換を行うツールを実装した．また，評価用のプログラムとして N クイーン問題の求解プログラムを作成し，評価を行った．提案方式を用いることで，プログラマは Cell プログラミング特有の記述を意識することなく，スレッドプログラミングのみを行うだけで，Cell プロセッサ向けのソースコードを作成できることが確認できた．

変換プログラムで変換可能なプログラムには制約を課したが，その制約下においても，実行する計算量が多い場合，POSIX スレッドによるソースコードに比べて高速に動作し，人手によるコーディングに比べても，速度性能を大きく損なうことの無い PPE/SPE ソースコードを得られることが確認できた．

5.1 本システムの制約

本研究で作成した変換ツールの実装において，まずはスレッドソースコードで利用できる記述や変換ツールの解析動作に制約を課し，その制約下において変換の出来るツールの実装から始めた．現在の制約を以下に示すとともに，今後の方針を述べる．

(1) POSIX スレッド API としては，`pthread_create()`，`pthread_join()`，`pthread_exit()` の各関数のみを用いることが出来る．但し，スレッドの返り値を用いることは出来ない．

(2) 大域変数については，メインルーチン内，または単一スレッド内での書き換えは可能だが，書き換えられた後のデータを他のスレッドで参照することは出来ない．これは，大域変数として記述された変数宣言は，変換ツールによって PPE/SPE の両ソースコードの大域変数として記述されるため，メインメモリ中，または SPE の各 LS 中に確保され，それぞれは個別に参照することが出来るが，変更を加えた場合に各メモリ間で一貫性を保つことが出来ないためである．そのため，書き換えを伴うメインルーチンとスレッド間のデータのやり取りは，スレッドに渡す構造体を通じてのみ行うこととする．

(3) 関数の呼び出し関係などは解析していため，main 関数とスレッド関数以外に定義された関数については，呼び出しの有無には関係なく，

PPE/SPE のソースコード両方に出力される．場合によってはコンパイルエラーの原因となるため，プログラマが生成されたソースコードから該当部分を除去する必要がある．

(4) Cell 上で 16 バイト以上のデータの DMA 転送を行う場合，転送データが 16 バイトの倍数であり，かつ 16 バイト境界でアラインメントされていなければならないというアーキテクチャ上での制限があるが，データサイズの調節や，アラインメントを指定する記述の自動挿入を行っていない．そのため，プログラマが手動で記述する必要がある．

(1) については，利用頻度が高いであろう排他制御の API(`pthread_mutex`) を例に挙げると，東芝による SPE ライブラリ環境では排他制御用の API が用意されているため，これらを用いて実現することが出来る．それ以外の環境では，Cell に用意された LL/SC 命令を用いて排他制御を実装することが可能だと考えられる．

(2) については，データの書き込み状況を調べ，大域変数の書き換えが行われるたびにデータ転送をする記述を挿入すれば実現できそうだが，SPE で使うことのできる大域変数が LS のサイズに限定されるため，あまり現実的ではない．変換ツールのほか，生成後の PPE/SPE で利用するためのソフトウェア分散共有メモリの開発が必要である．

(3)，(4) については，変換ツールの実装を単純化するためにひとまず設けた制約であり，解析処理とコードの追記/削除機能を改善することで比較的容易に実現できると考えられる．

今後の課題としては，SPE の LS のサイズを超える，大きなデータを扱うプログラムへの対応や，同種の処理を行うスレッドのみを複数生成するだけでなく，異種の処理を行うスレッドを混在させるプログラムへの対応などを行いたい．また，今回の評価では N クイーン問題の求解プログラムのみを用いたが，それ以外のアプリケーションも実装・変換し，評価をしたいと考えている．

参 考 文 献

1) Pham,D. et al.:The Design and Implementation of a First-Generation CELL Processor, In Proceeding of the IEEE International Solid-State Circuits Conference(2005).
2) 黒澤泰彦，渡辺幸男，田胡治之，”次世代プロセッサ Cell Broadband Engine”，東芝レビュー，Vol.61, No.6, pp.9-15（2006）．
3) Cell Broadband Engine resource center. http://www-128.ibm.com/developerworks/power/cell/ .
4) IBM Corporation : SPE Runtime Management Library Version 2.0, Cell Broadband Engine Architecture Joint Software Reference Environment Series(2006).
5) 前田誠司，佐藤記代子，川上健，”Cell プログラム実行環境”，東芝レビュー，Vol.61, No.6, pp.42-46（2006）．