社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－ARC－172 (44)
2007－HPC－109 (44)
2007／3／2

ClearSpeed製コプロセッサの並列ベンチマークによる性能評価と性能向上手法の提案

西川由理[†] 鯉渕道紘[††]
吉見真聡[†] 天野英晴[†]

本稿では，ClearSpeed 社の開発したコプロセッサ CSX600 のメモリ間や計算コア間転送の性能評価，およびその結果の考察を述べる．評価には姫野ベンチマークを用い，同社により提供される SDK(Software Developement Kit) により CSX600 向けにデータフローの異なる 4 種類の実装を行った．実行時間の測定結果およびプロファイリングの解析結果より，CSX600 の性能は一般的な PC(Xeon 2.80GHz) の 1.4 倍程度であるが，これは実際の演算時間が実行時間の 6.7%程度に留まることに起因する．この性能改善手法として 1) ローカルメモリからメインメモリへの直接参照ならびに計算コアの同時アクセスの実現，2) アーキテクチャの側面から計算コア間のバックグラウンド転送のサポート，3) 各コアの持つローカルメモリの利用率を向上させるためのコンパイラの改良の 3 点を示す．

A Proposal of Performance Improvement Based on A Parallel Benchmark Evaluation on a ClearSpeed Coprocessor

Yuri Nishikawa,[†] Michihiro Koibuchi,[††] Masato Yoshimi[†] and Hideharu Amano[†]

This paper focuses on performance evaluation of ClearSpeed's CSX600 coprocessor when it encounters frequent memory transfers between shared and local memories, or between local memories. For this aim, four versions of Himeno Benchmark was implemented on CSX600. The source code was programmed using SDK (Software Development Kit) provided by ClearSpeed Technology, and the dataflow was modified based on the chip architecture. As the results of time study and profiling, the performance of CSX600 was perceived to be comparable to that of Xeon 2.80GHz. Actual floating-point caluculation time is projected to be about 6.7% of overall execution time. To improve the performance on such a case, we show three key points: 1) a mechanism for direct memory reference from poly- to mono regions, 2) a mechanism for asynchronous inter-core communication, and 3) improvement on compiler for upgrading utility of local memories within each core in the array process.

1. はじめに

ハイパフォーマンスコンピューティングが求められる応用分野は，天体力学や流体解析などの科学技術演算はもちろん，データマイニングや金融工学，バイオ分野におけるシミュレーションなど，多岐に渡っている．このような発展は，汎用プロセッサの性能の飛躍的な向上によるところが大きく，近年では多数の計算機をネットワークで接続し，スーパーコンピュータに匹敵する性能を安価に実現する PC クラスタシステムを構成する例が多い．しかし，このようなシステムは消費電力の面などで大きな問題を抱えている．さらにその設置場所や空調にかかるコストも無視できない．

このような問題に対処するため，プロセッサのマルチコア化や，汎用プロセッサと協調動作するコプロセッサにより，高性能を実現するシステムも登場した．例えば，天体シミュレーションに特化した GRAPE-6[1] は，専用 LSI1024 チップを並列動作させることで，地球シミュレータを 2 割上回るピーク性能を実現している．昨年には FPGA を最大 7 個搭載する GRAPE-7 が$105/GFLOP という価格対性能比を実現した[2]．

最近では Cell Broadband Engine[3] がゲーム用途だけでなく，高性能が要求される他のアプリケーションのアクセラレータとしても注目を集めている．昨年には，単精度の高速フーリエ変換 (FFT) で 46.8GFLOPS の性能を達成したとの報告もある[4]．

科学技術計算，DSP，組み込みシステムなどの幅広

† 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University
†† 国立情報学研究所
National Institute of Informatics

いアプリケーションをターゲットにした ClearSpeed 社の CSX600 という高性能かつ低消費電力な浮動小数点演算コプロセッサも製品化されている．計算サーバとの協調処理を実現するために開発された PCI-X アクセラレータボードは，CSX600 を 2 個と 1GB のメモリを搭載し，浮動小数点演算が必要なアプリケーションをコプロセッサ側で処理させることで消費電力の増大を抑えながらシステム全体の効率性を上げることを目標としている．このアクセラレータカードの採用例としては，東工大のキャンパスグリッドの計算資源であるスーパーコンピューティングシステム TSUBAME[5] が挙げられる．アクセラレータボード 360 枚を一部のノードに接続することで，システム全体の消費電力増大は 1%程度に抑えつつ，性能が 25% 向上したことが報告されている．Linpack では 47.38 TFLOPS の性能を記録しており，これは 2006 年 11 月の TOP500 において第 9 位であった．

このほか，DGEMM，FFT (いずれも倍精度) などのベンチマークによる性能評価も行われており，それぞれ 25GFLOPS，10GFLOPS の性能を実現している[6]．CSX600 のプログラミング環境として，BLAS などの演算の機能を提供する CSX ライブラリ (CSXL) が用意されている．そのため，このライブラリがサポートしている演算を含むアプリケーションの場合，汎用の CPU と CSX600 の協調動作を比較的容易に実現することができる．しかし，現状ではこのライブラリはごく一部の演算のみをサポートしているに過ぎないため，CSX600 を活用して様々なアプリケーションを実行するためには，ClearSpeed 社提供の SDK (Software Development Kit) を用いた低位のプログラミングによる実装が必要となる．そのため，CSX ライブラリ (CSXL) を活用できるアプリケーションに限られた評価が多く，プロセッサの演算性能に関する議論が主である．しかし，科学技術計算，DSP，組み込みシステムなどの幅広いアプリケーションをターゲットにしている CSX600 ではプロセッサのみならず，オンチップネットワークなどの周辺部も，対象とするアプリケーションによっては，大きく性能に影響する．

そこで，本研究報告では SDK を用いて，CSX600 のオンチップネットワークや I/O 性能に焦点を置いた性能評価を行う．まず，メモリ間または計算コア間通信時のメモリバンド幅が性能に影響するようなベンチマークアプリケーションとして，姫野ベンチマーク[7] を選択した．そして，計算コア間のデータフローに注目した実装を行った，

この実装をもとに演算時間とデータ転送時間のプロファイリングを行った．評価結果から，性能は 256MFLOPS と汎用プロセッサ以下に留まり，さらに実行時間に占める有効な浮動小数点演算の割合は，全体の 6.7% 程度であることが確認された．

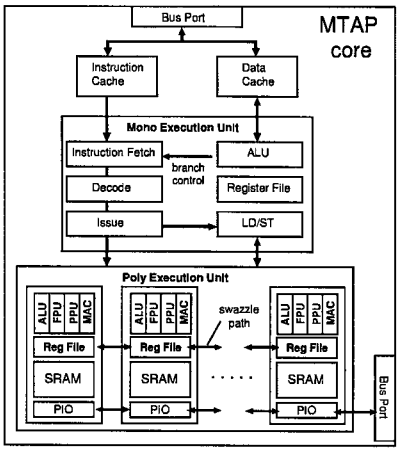

図 1　CSX600 内部の MTAP のアーキテクチャ

表 1　CSX600 のスペック

実効性能 (DGEMM)	25 GFLOPS
最大動作周波数	250 MHz
平均消費電力	10W
内部メモリ間転送速度	96 Gbps
外部 〃	3.2 Gbps

最後に，CSX600 が科学技術計算，DSP，組み込みシステムなどの幅広いアプリケーションを対象とした場合に必要となる性能改善手法として，1) poly 領域から mono 領域の直接参照，2) アーキテクチャの側面から計算コア間のバックグラウンド転送のサポート，3) 各コアの持つローカルメモリの利用率を向上させるためのコンパイラの改良の 3 点を示し，姫野ベンチマークのプロファイリング結果から，これらの効果について簡単な性能見積りを示す．

以下，2 章において CSX600 の概要を説明し，3 章では性能測定に用いた姫野ベンチマークの特徴を述べ，4 章にてそのデータフローに特化した実装について述べる．5 章では，実際に CSX600 で性能評価を行い，6 章にてその結果について議論する．最後に 7 章でまとめを述べる．

2. ClearSpeed CSX600 の概要

2.1 CSX600 のアーキテクチャ

ClearSpeed 社の SIMD 型コプロセッサ CSX600 は，単精度または倍精度演算に対し 48GFLOPS の理論性能，また行列積の演算ルーチン DGEMM により 25GFLOPS の実効性能を持つ[8]．主なスペックを表 1 に示す．

CSX600 は，96 個のプロセッシングエレメント (PE)

```
#include <lib_ext.h>

#define PI 3.14159265358979

int main() {
  poly double sin, angle;
  poly int pnum;

  // get_penum() obtains ID number of each core
  pnum = get_penum();
  angle = pnum * PI;

  // sinp() returns a vector(poly) value
  sin = sinp(angle);
}
```

図 2 C^n によるプログラム例

が一次元アレイ状に接続されたマルチコアプロセッサを持つ．これを multi-threaded array processor (MTAP) と呼び，その構造を図 1 に示す．各 PE はそれぞれ，32/64-bit の浮動小数点乗算器および加算器，6Kbyte のローカルメモリ (SRAM)，および整数演算用の ALU を持つ．これにより，PE 単独で 0.5GFLOPS の理論性能が得られている．さらに，各 PE は swazzle パスと呼ばれる 64-bit 幅のデータパスを介し，隣接 PE と通信を行う．

CSX600 は MTAP のほか，内部 SRAM，外部 DRAM インタフェース，および PE 間を接続する高速 I/O ポート等により構成される．特に，内部 SRAM，外部 DRAM は "mono" メモリ，96 個の PE 内メモリは "poly" メモリと呼ばれる．なお，各サブシステムのインターコネクトは ClearConnect と呼ばれる独自の Network-on-Chip により実現されている．

MTAP は優先度の異なるスレッドを 8 つまで同時に実行できる．例えばセマフォ管理を行うことで，他の演算中に mono - poly 間でデータをバックグラウンド転送することが可能である．ただし，PE 間通信のバックグラウンド転送は行えない．

2.2 C^n 言語

ClearSpeed 社は，96 個の PE に対するプログラミング環境として，独自の SDK とプログラミング言語 C^n を提供している．C^n は標準 C と多くの部分で共通しているが，最も大きな特長は，"poly" なる概念を導入したことにある．poly とは C^n 言語における予約語であり，MTAP の各 PE に明示的に変数等を割り当てるとき，すなわちベクトル値を宣言するときに用いる．C^n によるプログラム例を図 2 に示す[8]．このコードでは，`get_penum()` という関数で各 poly の ID 番号 (0〜95) を取得し，その値に基づいて正弦を求めている．`sinp()` は標準 C の sin 関数と異なりベクトル値を返すため、各 poly 変数 `sin` の値が異なる．

poly と対照的な概念は "mono" であり，このように宣言された変数は通常の C と同様にスカラー値となる．特に指定しなければ，変数等は mono 領域に割り当てられる．したがって並列プログラムの記述においては，データを mono 型と poly 型のどちらで扱うかに留意する必要がある．なお，poly 側から mono 領域に格納された値の直接参照は許されていない．したがって計算に用いるデータは，予め mono から poly メモリにコピーし，計算終了後に書き戻す必要がある．

表 2 姫野ベンチマークの計算サイズ

size	$i \times j \times k$
S	$65 \times 65 \times 129$
M	$129 \times 129 \times 257$
L	$257 \times 257 \times 513$

3. 姫野ベンチマークによる性能評価

本章では，ClearSpeed 上に実装する姫野ベンチマークの特徴を述べる．

3.1 姫野ベンチマークとは

姫野ベンチマークとは，非圧縮流体解析コードの性能評価のために考案された，ポアソン方程式をヤコビの反復法で解く際の主要な処理の演算速度を計測するもので，科学技術計算に対するシステム性能評価に広く用いられるベンチマークプログラムである．同種のベンチマークと比べてシステムに対する要求条件が厳しく，

- 高速なメモリアクセス
- 高速なノード間通信

の二点が性能向上の鍵である．つまり，計算機のメモリバンド幅やキャッシュの性能によって，結果が大きく変わることが知られている．

3.2 姫野ベンチマークの処理の流れ

姫野ベンチマークの主要な処理部では，三次元配列 (p とする) に対する浮動小数点演算が反復して行われる．プログラムコードでは，4 つのループが入れ子構造になっている．配列の値を全て用いた計算が終了すると，全要素の値が更新される．最も内側のループでは，34 回の浮動小数点演算 (加減算および乗算) が行われる．したがって，浮動小数点演算の総数は，配列サイズを $X \times Y \times Z$，反復回数を N とすると

$$34N \times (X-2)(Y-2)(Z-2) \qquad (1)$$

となる．性能測定は，およそ 1 分の間にプロセスの反復回数 N に基づき，単位時間当たりの浮動小数点演算数を求めることによって行う．

3.3 並列化された姫野ベンチマークにおける特徴

姫野ベンチーマークに並列プログラミングを適用して実行する場合，メモリ間，特にコア間通信に高いメモリバンド幅が求められる．この最小ループが 1 回実行される度に，座標 (i,j,k) に位置する要素 $p[i][j][k]$ は，三次元的に隣接する全ての要素の値を必要とする．一度に参照される要素を模式的に表したものを，図 3

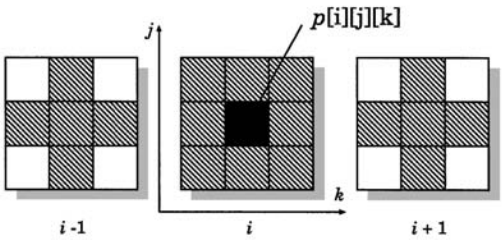

図 3　一回の浮動小数点演算で参照される要素

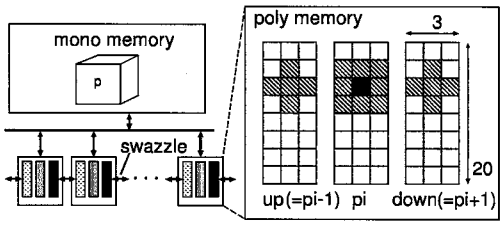

図 4　Type 2 における配列データの分配方法

の斜線部分で示す.

また，性能測定時には全ての計算コアで計算サイズを固定しなければ，公平な測定値が得られない．測定で用いた計算サイズを表 2 に示す.

姫野ベンチマークは，単一ノードで解くものと，MPI 方式による並列プログラムで解くものとの二種類が公開されている． 後者の並列化手法では，表 2 に示したサイズの配列を，ノード数と等しい個数に三次元的に分割する．例えば計算サイズ M で 128 プロセスに分割する場合，ネットワーク構成が $16 \times 4 \times 2$ ならば各ノードに $11 \times 35 \times 131$ の配列を配分する[9].

4. 実　　装

4.1 実装環境

本実装では CSX600 単体の性能を測定することを目的とし，その他のボード内のモジュールの性能への影響を最小限に抑えるため，アクセラレータボード上の 2 チップのうち 1 つを用いる． 実装環境は，ClearSpeed 社が提供する SDK を用いた．これには専用コンパイラ，ホスト PC からボードを利用する際のデバイスドライバ，デバッガ，実行ファイルからの逆アセンブラなどが含まれる．またコードは C^n 言語で記述し，インラインアセンブラによる最適化は行っていない.

4.2 データフローに特化した実装

CSX600 では，96 個の各 PE 間が一次元的に接続され，三次元的に隣接する PE の値を参照するには効率が悪い．さらに各 PE のローカル (poly) メモリが高々6kbyte であり，かつメイン (mono) メモリの直接参照も不可能なため，3 章で述べたような手法で配列データを配分することは困難である.

したがって，以下に示す手法で実装を行った．まず，96 個の PE の大半を稼動させるため，入れ子構造の最も外側のループ，すなわちサイズ $X \times Y \times Z$ の配列の X について展開し，各 PE には図 3 で示したような二次元の配列データを poly メモリ中に持たせる． しかし，poly メモリのサイズには限りがあるため，元の三次元配列を mono メモリ中に置き，各 PE は計算に必要なデータを他の演算中にコピーし，終了後に書き戻すことで計算を行う． このようにすれば，mono - poly 間のメモリコピーを繰り返しながら，アレイ上で左ないし右隣の PE から一部のデータを参照することで，計算サイズの大きい問題にも対応できる.

以上より，時間計測の対象となる計算ステップは次のようになる.

(1) mono メモリから配列 p の一部のデータのバックグラウンド転送を開始
(2) 両隣の PE と (1) のデータを送受信
(3) 浮動小数点演算を実行
(4) poly メモリ内の要素に対する演算が終了していれば (1) に戻り，でなければ (3) に戻る

ここで，(2) の隣接 PE の値の参照には，PE 間を接続する swazzle パスを用いる．これについて，以下の 2 通りの参照方法を試みた.

- Type 1: 計算途中に必要に応じて参照
- Type 2: 予め poly メモリ中に転送

これに従い，poly メモリのデータ配分をも二通りを用意した．Type 1 は，poly メモリの使用可能領域のほとんどを，独自の配列データのみとする方法である．ここではサイズが 3×80 の配列に格納した． Type 2 では，図 4 に示すように poly メモリを 3 分割し，1/3 を配列 p のデータの一部，残り 2/3 に両側の PE から転送されたデータを保持する．これら 3 種類のデータはサイズが 3×26 の配列に格納した． このように分けたのは，姫野ベンチマークで一回の演算毎に左右の PE から各 5 回の倍精度浮動小数点データ参照があり，データの上りと下りを兼ねる 64-bit 幅の swazzle パスがボトルネックになる可能性を調べるためである．なお，ここに示した poly の配列サイズは，SDK でコンパイル可能な最大値であった．このサイズは実際には，容量が 6kbyte である poly メモリの 1/3 程度しか活用できていないことを意味している.

さらに，mono プロセッサの性能測定のため，配列データを，図 5 に示す 2 通りの方法で更新した.

- Type A: poly メモリから mono メモリのテンポラリ領域に転送し，mono プロセッサ内で更新
- Type B: 読み出し用と書き込み用のメモリ領域を分離し，ループ毎に切り替えて使用

Type A ではデータフローが循環的であるのに対し，

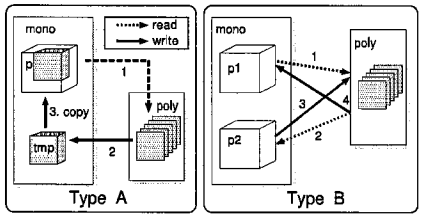

図 5 値の更新方法

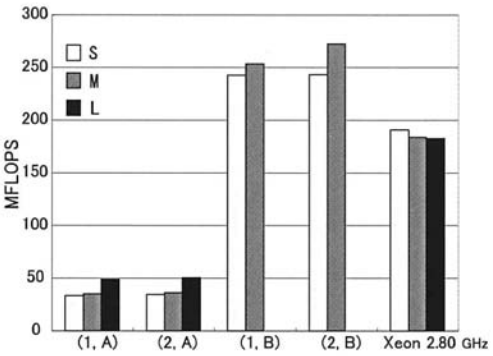

図 6 各サイズでの性能

Type B は一回の演算が終了する度に読み込み先と書き込み先が入れ替わり，mono プロセッサで値をコピーする手順が省略できる．

5. 姫野ベンチマークによる性能評価

本章では，前章で述べた手法で実装した姫野ベンチマークを CSX600 の実チップ上で動作させた評価結果を示す．

5.1 評価環境

評価方法として，姫野ベンチマークの実装 Type (1, A)，(2, A) を S，M，L サイズで，Type (1, B)，(2, B) を S，M サイズで実装し，1 分間の間に実行される有効な浮動小数点演算の総数を求めた．それぞれの手法で，オリジナルのソースコード (単一ノード用) を汎用プロセッサ (Xeon 2.80GHz) で実行した結果と等しいことを確認している．また時間計測は，ClearSpeed 社より提供されるデバイスドライバを用い，ホスト側から計算開始および終了のシグナルを送受信することで行った． この結果を図 6 に示す．なお，コンパイル時には CSX600，プロセッサでのいずれも最適化オプションを指定していない．

この結果より，Type A では Xeon 2.80GHz での実行速度と比較して 0.17〜0.28 倍，Type B で 1.27〜1.49 倍の性能を示すことが確認された．特に Type A で性能低下が著しいのは，配列データの更新を単一

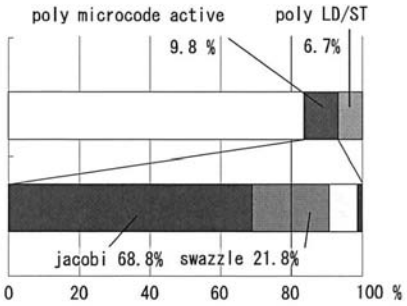

図 7 実装 (2, B) におけるプロファイル結果 (計算サイズ: M)

(mono) プロセッサのみで行うからである．4 章でも述べたように，Type A は mono プロセッサが値を移動する点以外は Type B と同一なため，両者の性能の差が mono プロセッサの持つ処理能力と見なせる．

1 チップの実効性能は 25GFLOPS と言われているが，それに比べて性能が大きく下落する原因を調べるため，(2, B) の計算サイズ M における処理の内訳を，同社により提供されるプロファイリングツール Cleartrace を用いて調査した．それを示したのが図 7 である． これより，各 PE の浮動小数点演算器の稼動率が全体の実行時間中の 9.8%，poly メモリにアクセスされる時間が 6.7%に留まることがわかった．加えて演算器稼動中も，姫野ベンチマークで計測対象となる処理にあたる時間は，そのうちの 68.8%，つまり全体の 6.7%程度であることが判明した．従って，C^n 言語設計における浮動小数点演算の性能は，以上の結果より 4.1 GFLOPS 程度であることが予想される．

さらに，PE が稼動しない約 83.5%のうち，56.9%が mono - poly 間転送の待ち時間であることが，解析の結果わかっている．3 章でも述べたように，姫野ベンチマークはキャッシュで性能が決まる．ここで，キャッシュに相当する poly メモリのおよそ 1/3 程度しか使えないことが，転送回数を増大させ，性能低下を引き起こしているといえる．

6. 議　　論

本章では，CSX600 のアーキテクチャに沿ったデータフローによるコードを用いたにも関わらず，性能が 5 章で示した程度に留まる理由を考察し，さらなる性能向上のために考慮すべき点について議論する．まず，CSX600 のアーキテクチャの面から述べる．

CSX600 は SIMD 型であるにも関わらずマルチスレッド化をサポートしているのが特徴で，この最も大きな利点の一つが mono - poly 間のデータのバックグラウンド転送機構だと言える．しかし，その機構を用いても転送の待ち時間が主要ループ処理全体の 6 割

程度を占める結果となった．図6によるとCSX600は問題サイズが大きくなるほど汎用プロセッサに対して有利だが，このような大規模なデータはmono領域にしか置けず，値の直接参照が行えないのが性能低下を引き起こしていると言える．

またPE間はswazzleパスにより一次元的に接続され，1回につき2クロックで64-bitのデータ転送が行われる．転送中は他の演算は中断される．これよりPE内部にDMAコントローラが必要なく消費電力の面で有利と言われているが，図6の(1, B), (2, B)の結果より，問題サイズが大きくなるにつれて隣接PE間のデータ転送効率が性能に影響すると思われる．

次に，CSX600でのC^n言語設計における問題点について議論する．最大の難点は，各PEの持つメモリを1/3程度しか活用できないことにある．利用できるpolyメモリの容量が少ない場合，mono - poly間の転送頻度を増やすことで対処せざるを得ないが，評価結果からも得られるように，転送は全体の性能を大きく低下させる一因である．さらに図7のプロファイル結果より，仮に浮動小数点演算器が常時稼動したとしても実効性能の16%程度しか得られないことが予想される．

以上より，科学技術演算，DSP，組み込みシステムなどの様々なアプリケーションにおいて必要となるCSX600の性能向上手法として，次の3点を提案する．

- polyメモリは容量が小さく転送もボトルネックになる可能性が高いため，各PEがmono領域の異なるデータにアクセスできる機構を設けるべきである．現在はpoly領域からmono領域の直接参照は禁止されているが，メモリ間のバンド幅を向上させることで，これを可能な限り実現することが効果的と考えられる．
- マルチスレッドを可能にする機構を活かし，隣接PE間も非同期転送を可能にすべきである．データの上りと下りの線を分離することにより，各PEがDMAC等を持たずともバックグラウンド転送が実現できると考えられる．
- 図6 (1, A), (2, A)より，monoプロセッサ単独による処理が大きな性能低下を引き起こすことが分かった．このようなプログラムの実行を未然に防ぐため，monoプロセッサのみが利用されたりPEの稼動率が著しく低い場合に、コンパイラがその箇所を自動検出し，予測される稼動率の表示とともに警告する機構を設けるべきである．さらに，浮動小数点演算の実効性能に近づけるにはチューニングされた様々な実数演算のライブラリが必要になるだろう．

7. まとめ

本稿では，ClearSpeed社の開発したコプロセッサCSX600のメモリ間やPE間転送の性能を評価するため，データフローに着目した姫野ベンチマークを同社から提供されるSDKを用いて実装した．実行時間の測定およびプロファイリングの結果より，CSX600の性能は汎用的なPC (Xeon 2.80GHz)の1.4倍程度であり，さらに実際の演算が行われる時間が実行時間全体の6.7%程度に留まることがわかった．この性能改善手法として，1) poly領域からmono領域の直接参照，2) アーキテクチャの側面から計算PE間のバックグラウンド転送のサポート，3) 各PEの持つローカルメモリの利用率を向上させるためのコンパイラの改良の3点を示した．今後は，SDKを用いてNAS Parallel Benchmarksのような並列アプリケーションについても性能評価を行う予定である．

謝辞

本研究の一部は，国立情報学研究所共同研究「ネットワークオンチップにおけるストリーミング処理に関する研究」による．

参考文献

1) Makino, J.: The GRAPE project , *Computing in Science and Engineering* (2006).
2) K&F Computing Research: `http://www.kfcr.jp/`.
3) 林宏雄, 斎藤光男, 増渕美生: Cell Broadband Engineの設計思想 , 東芝レビュー, Vol. 61, No. 6 (2006).
4) Chow, A., Fossum, G. and Brokenshire, D.: A Programming Example: Large FFT on the Cell Broadband Engine, *Proceeding of the 2005 Global Signal Processing Expo* (2005).
5) "遠藤敏夫, 松岡聡, 橋爪信明, 長坂真路, 後藤和茂": "ヘテロ型スーパーコンピュータTSUBAMEのLinpackによる性能評価", 情報処理学会研究報告 2006-HPC-107 (pp43-48,July 31) (2006).
6) ClearSpeed Whitepaper: Accelerating the Intel Math Kernel Library: `http://www.clearspeed.com/`.
7) 姫野ベンチマーク: `http://accc.riken.jp/HPC/HimenoBMT/index.html`.
8) ClearSpeed Whitepaper: CSX Processor Architecture: `http://www.clearspeed.com/`.
9) 中尾昌広, 廣安知之, 三木光範: 姫野ベンチのパラメータチューニング, *ISDL Report*, No. 200206100013 (2002).