Vol.2013-HPC-140 No.38
2013/8/2

大規模SIMD型アクセラレータの検討

児玉 祐悦[1,2,a)]　山口 佳樹[2]　中里 直人[4]　牧野 淳一郎[3]　朴 泰祐[1,2]　佐藤 三久[1,2]

概要：将来のアクセラレータとして有望な大規模 SIMD 型プロセッサについて、検討を行っている。そのアーキテクチャの基本設計方針や、性能パラメータの検討結果を示すとともに、より詳細な評価を行うために、検討のたたき台としてのアーキテクチャの設計を行った。そのアーキテクチャを紹介するとともに、そのアーキテクチャに基づくサイクルベースシミュレータを作成して、いくつかのベンチマークを用いて性能見積もりを行った。また、今後の評価の方針や検討課題について考察する。

1. はじめに

京コンピュータをはじめとして、10PFLOPS を超えるスーパーコンピュータが次々と開発されている。TOP500[1] の性能を外挿すると 2018-20 年ころには 1EFLOPS を超えることが予想される。もちろんこの予想は、性能向上の技術開発が順調に行われた場合の予測であり、何もしなくても達成される性能ではない。海外でもエクサフロップスを目指した調査・研究が行われているが、日本でも 2011 年からエクサフロップスに向けた技術課題やアプリケーションからの要求性能などがロードマップ [2] としてまとめられるとともに、2012 年度より文科省による「将来の HPCI システムに関する調査研究」として 3 つのシステム評価と 1 つのアプリケーション評価がフィージビリティスタディとして行われている。

上記のロードマップでは、アプリケーションの要求するメモリ容量とメモリ帯域に着目してアプリケーションを分類し、a) メモリ容量はそれほど大きくないが高いメモリ帯域を必要とする容量削減型、b) メモリ容量はそこそこ必要だがメモリ帯域はそれほど大きくない演算重視型、c) メモリ容量、メモリ帯域ともにそこそこ必要な汎用型、d) メモリ容量・メモリ帯域ともに高い性能が必要な容量・帯域型の 4 つがあることを明らかにした。

我々筑波大学、東工大、会津大、理研、並びに日立のチームは、演算重視型および容量削減型のアプリケーション向けにアクセラレータを用いたシステムの検討を行っている。基本方針として、電力性能比を向上させるために、シンプルな構成を取りつつ、演算性能およびオンチップメモリ性能を向上させることとした。基本的な構成について、2018-20 年の LSI テクノロジとして 14nm 程度を想定し、規模や性能について次のように想定した。

- チップサイズを 20mm x 20mm としてメモリ (SRAM) 換算で 1GB/チップが可能と想定
- コア (PE) とメモリの比を 1:1 として、90nm で製造した GRAPE-DR[3] からの外挿によりチップあたり 4096 PE とした。チップ内メモリは 512MB/チップであり、各 PE に分散させると 128KB/PE となる
- 動作周波数は 1GHz を想定
- 各 PE の演算器として 128 ビットの FMA 型演算器を仮定すると、倍精度演算で 4GFLOPS/PE となり、16TFLOPS/チップとなる。全 PE が単一命令流で実行する SIMD を想定
- PE 間は 2 次元メッシュの隣接通信をサポート
- ホストとの間は次世代の PCI Express あるいはその他の標準的な I/O 規格を想定
- ホストを介さずにアクセラレータ間で通信を行うチップ間ネットワークを想定

この他、オンチップメモリだけでは容量が足りないアプリケーション向けに、HMC(Hybrid Memory Cube)[4] の次世代版など今後可能になるとみられているインターポーザを介した 2.5 次元実装のスタックメモリの利用も検討する。容量は 16-32GB/chip、バンド幅は 512MB-1TB/s を想定する。通常の DDR メモリは想定しない。電力は 270W/chip を目標として 60GFLOPS/W 以上を想定する。

このような基本構成を元にして、システムの実現可能性

[1] 筑波大学 計算科学研究センター
Center for Computational Sciences, University of Tsukuba
[2] 筑波大学大学院 システム情報工学研究科
Graduate School of System and Information Engineering, Tuniversity of Tsukuba
[3] 東京工業大学
Tokyo Institute of Technology
[4] 会津大学
University of Aizu
[a)] kodama@cs.tsukuba.ac.jp

を含めたアーキテクチャの詳細化、命令サイクルレベルでの詳細な性能見積もり、アプリケーションレベルでの本基本構成への適用可能性および大まかな性能見積もりなどを並行して行う必要があった。以下ではまず、命令サイクルレベルでの評価を行えるようにアーキテクチャの一部を具体化してシミュレータを構築し、命令サイクルベースの評価について述べるとともに、他のレベルの評価についてもその概要を述べる。さらに、それらの評価から要求される改良方針を検討し、今後の評価項目について考察を行い、まとめを行う。

2. たたき台アーキテクチャ

本評価で、最初にたたき台として想定しているアクセラレータのアーキテクチャは図 1 の通りである。コントローラが 1 つあり、その中にデータメモリ（DM）と命令メモリを持つ。コントローラが、ホスト PC とのデータ交換、放送メモリへのデータ転送、各 PE への命令発行を行う。複数の放送メモリ（BM）があり、その BM に複数の PE が接続している。BM との接続とは別に、各 PE は隣接する PE と 2 次元接続されている。図 1 では 4x4 の PE 構成を示しているが、最大構成としては、BM は 64 個（各 BM の容量は 128KB）、BM あたりの PE の個数は最大 64 個としており、最大構成で 4096 (64 x 64) PE を想定している。各 PE はローカルメモリ 128KB、128 ビット汎用レジスタ 64 個、128 ビット浮動小数点乗算器 (64 ビット x 2 あるいは 32 ビット x 4 で動作、他の演算器も同様)、128 ビット浮動小数点加算器、128 ビット ALU などから構成される。ローカルメモリとは 128 ビット/ クロックのバンド幅、各演算器も 128 ビット/クロックのスループットを持つ。一方、データメモリ・BM 間、BM・PE 間、PE・PE 間はそれぞれ 64 ビット/クロックのネットワークで接続されている。チップ間のネットワーク、およびホスト間ネットワークについては詳細を検討中であり、3.2 でその検討内容について述べる。

2.1 サイクルベースシミュレータ

アーキテクチャを評価するために、その上で動くプログラムの開発、プログラムの動作の確認、並びにプログラム実行のサイクル数を見積もることが必要である。そのため、サイクルベースシミュレータを作成した。本シミュレータは、アセンブラファイルを読み込んで実行ファイルに変換する asm1 と、その実行ファイルのシミュレーションを行う emul の 2 つのプログラムからなる。

アセンブラプログラムではチップ全体の制御を行うフロー制御命令と、各 PE で SIMD に実行される PE 命令が混在する。フロー制御命令を大文字、PE 命令を小文字で表すことで区別する。

フロー制御命令としては、ホスト計算機からのデータを

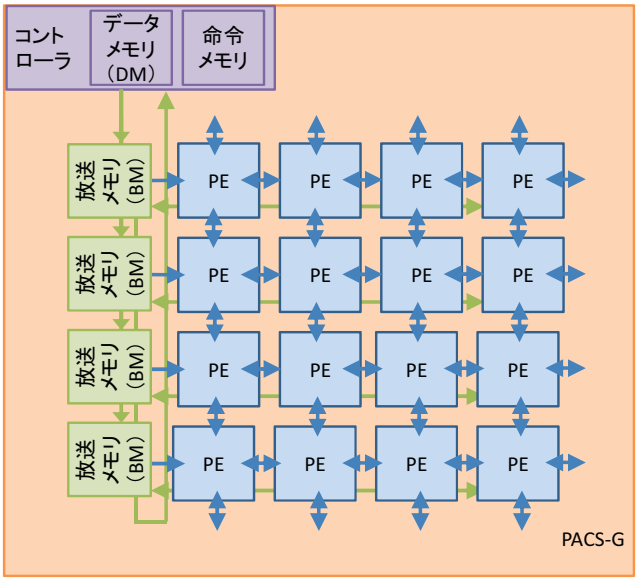

図 1 たたき台アーキテクチャ
Fig. 1 Straw man Architecture

受け取る領域（DM）上のデータ領域の指定を行う命令、PE 命令のフロー制御を行う分岐命令、分岐条件などを計算するレジスタ演算命令、DM 上のデータをレジスタに読み書きするロードストア命令がある。その他、DM から放送メモリ（BM）への転送を行う IDP 命令、BM からのデータを集約して DM へ格納する RRN 命令があり、これらのデータ転送は DMA により PE 命令とは独立に実行される。

各 PE で実行される命令は、基本的に以下の 3 つの領域からなる VLIW 的な命令フォーマットをもつ。

`fmul` 命令; `fadd/ialu` 命令; `bm/mv` 命令

これらの 3 つの命令は入力および出力オペランドの制約を満たす範囲内で並列に実行可能である。3 つの命令全体で汎用レジスタ 2 入力、ローカルメモリ 1 入力、あるいは専用レジスタを入力オペランドとして指定できる。また、汎用レジスタ 1 出力、ローカルメモリ 1 出力、あるいは専用レジスタを出力オペランドとして指定できる。専用レジスタとしては、乗算の結果レジスタ ($fb)、テンポラルレジスタ ($t)、PE アドレスレジスタ ($pe) などがある。例えば、

```
fmul m0 r0; fadd $fb $t $t; bm b0 r1
```

という 3 命令の組を考えると、ローカルメモリの値 m0 にレジスタの値 r0 を掛けて (暗黙的に) $fb に格納し、直前の乗算の結果 ($fb) をテンポラルレジスタ ($t) に加え、BM 上のデータ b0 をレジスタ r1 に格納する、という 3 演算を同時に実行できる。ただし、加算結果を$t ではなく汎用レジスタに格納しようとすると、bm 命令の結果格納とコンフリクトしてしまうため bm 命令との同時実行はできない。

現在の PE 命令は、たたき台アーキテクチャとして GRAPE-DR と同様に 1 命令を 4 クロック繰り返してベクトル実行を行う。演算器は倍精度 2 個、あるいは単精度 4

個の演算を 1 クロックで行えるため、1 命令で最大倍精度 8 個、あるいは単精度 16 個の演算を指定できる。このベクトル実行のオペランドを指定するために、次のような指定が可能である。

```
fmul a[0].2v b[0].3s
```

a および b はレジスタあるいはメモリ上の倍精度型ベクトル変数である。変数を.2v と修飾することにより、ベクトル実行中にアクセス幅 2 でベクトルアクセスすることを指定する。一方、.3s と指定した変数はその 64 ビット値を 128 ビットオペランドの上位 64 ビットにコピーするとともにその値をベクトル実行中保持する。すなわち、8 演算に対して同じスカラ値を供給する。結果として上の 1 命令は次の 8 演算（4 クロック）を指定することになる。

```
fmul a[0] b[0]; fmul a[1] b[0]
fmul a[2] b[0]; fmul a[3] b[0]
fmul a[4] b[0]; fmul a[5] b[0]
fmul a[6] b[0]; fmul a[7] b[0]
```

変数がローカルメモリ上にある場合には v の後に値を指定することによりベクトルアクセスのストライドを指定することもできる。ただし 1 クロック内のデータは連続している必要がある。例えば以下の命令は次のような実行となる。

```
fmul a[0].2v10 b[0].2s
```

=>

```
fmul a[0] b[0]; fmul a[1] b[1]
fmul a[10] b[0]; fmul a[11] b[1]
fmul a[20] b[0]; fmul a[21] b[1]
fmul a[30] b[0]; fmul a[31] b[1]
```

PE 命令は全 PE に対して SIMD 命令として実行される。条件文のように一部の PE のみで実行を行なうために、PE 毎に実行フラグが設定されている。本シミュレータでは 4 ビットの実行制御フラグがある。指定したビットが 1 の PE のみ、以降の PE 命令が実行されるようになっている。通常は、常に 1 であるビット 0 が指定され、全ての PE で実行が行われる。また、指定したビットが 0 の PE のみ、以降の PE 命令を実行するよう指定することも可能で、これにより else 節の実行も容易に指定できる。上位 3 ビットの値は、演算結果のゼロフラグあるいはキャリーフラグに従って設定することができる。

ローカルメモリはアドレス直接参照およびレジスタ間接参照でアクセス可能であり、毎サイクル 128 ビットが参照できる。ただしアクセスデータは 64 ビットでアライメントされている必要があり、単精度 1 つ分ずれたデータのアクセスの場合はアライメントされた 2 つのメモリアクセスと、補正のための命令実行を行う必要がある。

隣接 PE 間の通信は東西南北に隣接する PE との通信を示す専用レジスタ$e, $w, $s, $n を用いて、データの送信と受信を明示的に指定する。例えばレジスタ r0 の値を右隣の PE のレジスタ r1 に送る場合は、次の 2 命令となる。

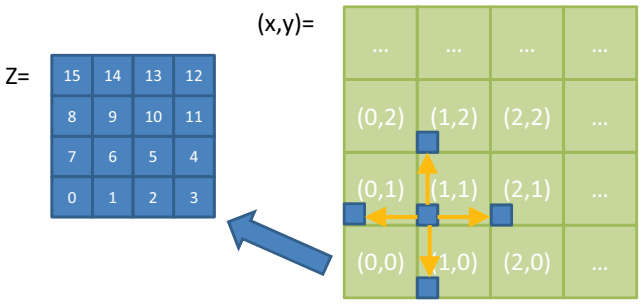

図 2 3 次元格子の 2 次元マッピング
Fig. 2 2D mapping of 3D grids

表 1 間接転送パラメータ
Table 1 Parameters for indirect transfer

方向	送信時	受信時
none	0x00	0x00
East(+X)	0x04	0x20
West(-X)	0x05	0x28
South(-Y)	0x06	0x30
North(+Y)	0x07	0x38

```
ipassa r0 $t $e
mv $w r1
```

最初の命令はデータの送信に相当し、必要なデータを$e レジスタに格納する。このレジスタは右隣の PE からは$w レジスタとして参照され、次の命令で送信先のレジスタに格納する。基本は SIMD なので全 PE が同じ命令を実行することになり、この命令列で全 PE が右隣の PE へのデータ転送を行う。端の PE の場合など、必要に応じて条件実行を用いることとなる。

このように、隣接 PE 間のデータ転送は基本的に Send ベースであり、データを保持している PE が必要とする PE にデータ転送する。複数ホップ離れた PE 間でデータ通信するためには、その間の PE が中継する必要がある。例えば図 2 は 16x16x16 の 3 次元格子を 64x64 の PE に割り当てる例を示している。ここでは 4x4 のグループで PE を 2 次元分割し、図 2 右のように、グループの 2 次元配置をそれぞれ x、y 次元に割り当てる。一方、グループ内を z 次元に割り当て、図 2 左のように z 次元を一筆書きで割り当てる。この時、x、y 次元の隣接通信は 4 ホップ離れた通信となる。x 次元のプラス方向にデータ転送を行う例が以下の命令列であり、5 命令必要となる。ネットワークは 1 クロックで 64 ビットデータの転送を仮定している。複数のデータを転送する場合は最後の命令がオーバラップ可能で 2Byte/clock のスループットとなる。

```
ipassa a[0].1v $t $e
mv $w $e
mv $w $e
mv $w $e
mv $w b[0].1v
```

一方、z 次元の隣接通信は 1 ホップで可能であるが、PE によって転送方向が異なるため、単純な命令のみでは条件実行が必要となる。SIMD 実行の場合、条件実行は逐次化されるためかなりのオーバヘッドとなる。そこで、方向を間接指定する命令を追加した。これは専用レジスタ$dr の値によって転送する方向を指定できる命令である。表 1 は設定値の一覧である。例えば z 次元プラス方向にデータ転送する場合、z=3 の PE では送信方向は北、受信方向は西であるので、$dr を 0x2f と設定してから$d レジスタにデータを書き込むと、送信指定した方向へ送信し、$d レジスタからデータを読み込むと、受信指定した方向からデータを読み込む。表で none とは送信あるいは受信を行わないことを示す。例えば上の例では z=0 の PE は東に送信はするが受信は行わないので、$dr を 0x04 と設定して受信を none に設定すると、受信を行わずに送信のみ行うので、条件実行を用いる必要はない。したがって、各 PE が適切な値を$dr に設定しておくことにより、以下の 2 命令で z 次元の隣接通信が行なえる。各 PE が設定すべき値は事前に表としてメモリに配布しておくことを想定している。複数のデータを転送する場合は 2 つの命令がオーバラップ可能で 8Byte/clock のスループットとなる。

```
ipassa a[0].1v $t $d
mv $d b[0].1v
```

上の 4 ホップ離れた x 次元の隣接転送の例では全 PE がデータ転送を行うため、送信と中継と受信を全 PE で実行すればよかった。しかし、特定の PE 間でデータ転送を行うためには、他の仕組みが必要である。1 データを送る場合には、条件実行を行えば、転送は可能であるが、オーバヘッドは大きくなる。例えば、ホップ数を h とすると、1 データ送るのに h+1 サイクルが必要であり、n データ送るためには n(h+1) サイクルが必要となる。このオーバヘッドを削減するために、$dr レジスタに専用フラグを設け、このフラグが 1 のときには$dr で指定した送受信レジスタ間の転送を条件実行フラグとは独立に行うこととした。これにより、中継すべき PE で適切に$dr を設定しておけば、送信 PE で条件実行で h サイクル送信を繰り返し、次に受信 PE で条件実行で h サイクル受信を繰り返すと 2h サイクルで h 個のデータ転送が行える。ホップ数とは独立に 32 ビット/サイクルのスループットが達成できるが、一度に送るブロックサイズはホップ数に応じて変更する必要がある。この転送は列方向、あるいは行方向では独立に転送が行える。また、BM 上に利用可能なバッファがあれば、BM とのデータ転送命令を用いることにより、同じスループットで実現できる。

BM 上のデータを各 PE に転送するには bm 命令を PE で実行する。bm 命令で指定された BM 上のデータが BM の制御回路によって読み出されて放送バスに出力される。各 PE ではその放送バス上のデータを指定したレジスタに格納する。全 PE が同一 bm 命令を実行することによりブロードキャスト転送となり、条件実行を行うことにより特定の PE のみへのマルチキャスト転送となる。各 PE のデータを BM へ格納するには送信 PE 指定付きの bm 命令を実行することになる。64PE の各データを BM に格納するには、それぞれの PE が送信元となる 64 命令が必要となる。

データメモリ (DM) から放送メモリ (BM) への転送は、フロー制御命令で DMA を起動することにより行う。この転送は PE 命令実行とは独立に行われる。DMA のパラメータは SETIDP 命令で静的に設定し、DMA の起動を IDP 命令で、DMA の完了待ちを IWAIT 命令で行う。DMA のパラメータにより、BM へのブロードキャストや、偶数番目の BM のみへの転送といったマルチキャスト、各 BM へ異なるデータ転送行うシーケンシャル転送などを指定できる。

放送メモリ (BM) の内容をデータメモリ (DM) に格納するのが RRN 命令である。この命令は各 BM の同じアドレスの内容をリダクションしながら（例えば総和）DM に格納する命令であり、リダクションはパイプライン的に処理されるため、BM が N 個でデータサイズが M の場合、M+logN サイクルで転送が完了する。RRN の転送も PE 命令実行とは独立に行われ、RRN の完了を待つ命令が RWAIT 命令である。

2.2 シミュレータによる評価

サイクルベースシミュレータによる評価として、差分法スキーム FDTD 法、姫野ベンチマーク、行列乗算、n 体問題のアセンブラプログラムを作成して実行した。以下では、アクセラレータの動作周波数を 1GHz と仮定した実行時間を示す。

2.2.1 差分法スキーム FDTD 法

FDTD 法は 3.3 に示すアプリケーション性能見積もりでも利用した地震波計算コードのカーネル部分である。各 PE あたり 8x8x8 格子を扱い、4096PE で 128x128x128 格子の演算を行う。各格子点では速度 3 変数、テンソル 6 変数等、計 16 の単精度変数を持ち、袖領域も含めて 118KB とほぼローカルメモリを使用している。

3 次元格子の 2 次元 PE へのマッピングは図 2 と同じとした。z 軸内隣接転送のための間接方向指定のテーブルを各 PE に適切に設定するために DM から BM へのマルチキャスト転送、および BM から PE への条件実行によるマルチキャスト転送を用いた。元にした Fortran ソースは 700 行ほどであった。これを m4 によるマクロを用いて記述することにより、アセンブラファイルで 1353 行と、非常に簡潔に記述することができた。展開後のファイルは 17153 行であった。ここで、時間ステップ内はすべてループ展開し、時間ステップ間はフロー制御命令によるループ

処理としている。袖領域の転送などは複数の配列に対して同様の処理を行うため、マクロ処理としてまとめることにより簡潔に記述できている。x、y 次元については周期境界条件が設定されており、袖領域転送後に、複数の配列の転送を各次元の両端間で行っている。この転送は 60 ホップ離れた PE 間の転送となり、また転送の送信元、送信先となる PE ペアが行あるいは列方向に 4 組づつあり、同時には転送できないため、袖領域の転送よりもオーバヘッドは大きい。この周期境界条件のデータ転送も配列ごとにマクロ処理としてまとめて簡潔に記述できている。一方、z 次元の境界部分については異なる式を適用しており、条件実行が必要となっている。

シミュレーション実行を行い、初期化に 3.0us, 速度計算とテンソル計算の 1 時間ステップにそれぞれ 32.7us、26.3us かかることを確認した。格子あたりの演算量はそれぞれ 57, 66 であり演算性能としてはそれぞれ 3.6TFLOPS、5.3TFLOPS、平均 4.4TFLOPS であった。

現在は袖領域の転送、周期境界条件の転送、格子内の演算をオーバラップしておらず、演算実行の割合が速度計算で 33 %、テンソル計算で 58 %であることと、演算量とメモリ参照量がほぼ等しくメモリアクセス性能が性能律速となっていることが、このような性能となっている原因と考えられる。速度計算とテンソル計算の性能の差は、参照する変数の数と更新する変数の数の差である。速度計算ではテンソル変数 6 個を参照して、速度変数 3 個を計算する。参照する変数はあらかじめ袖領域のコピーや境界条件の転送などが必要となる。一方、テンソル計算では速度変数 3 個を参照してテンソル変数 6 個を計算する。そのため、テンソル計算の方が計算量が大きく、袖領域などの通信量は小さいため、演算性能も高くなる。特に速度計算で周期境界条件の転送に 44 %と演算時間より多くの時間がかかっている。また、z 次元の条件実行によるオーバヘッドは端の部分の式は内部に比べて簡単になっていることもあり、速度計算、テンソル計算それぞれで演算実行の 15.5 %に抑えられている。しかし、この条件実行はそれぞれ z 次元の一方の端である 1/16 の PE のみで実行されており、残りの PE はアイドル状態となるため、演算効率的な影響はより大きいとみなすこともできる。

なお、シミュレーション自身の実行速度は、2.4GHz 6core の Xeon E5645 dual socket のサーバで OpenMP による 8 スレッドの並列化を行った場合で、1.1k クロック/秒のシミュレーション速度であり、1 スレッド実行に比べて 3.8 倍の性能向上を確認した。

2.2.2 姫野ベンチマーク

姫野ベンチマーク [5] は非圧縮流体解析コードを用いたベンチマークである。問題サイズとしては Middle を用い、各 PE あたり 8x8x16 格子を扱い、4096PE で 128x128x256 格子の演算を行う。各変数は単精度であり、3 次元格子の 2 次元 PE へのマッピングは FDTD 法と同じである。ローカルメモリは各格子に 13 変数で袖領域を含めて 100KB ほどとなった。シミュレーションを行い、初期化に 27.1us、1 時間ステップに 19.4us かかることを確認した。1 格子あたりの演算量は 34 と規定されており、演算性能は 7.35TFLOPS であった。1 時間ステップでは袖領域転送および誤差のリダクションにそれぞれ 1.5us、0.7us かかっており、演算実行割合は 88 %と高いが、メモリ参照がボトルネックとなり、このような性能となっている。

2.2.3 行列乗算

行列乗算は、DM に置かれた配列 A,B の積 C を求めるプログラムである。各要素は倍精度である。配列 A を列で分割して各 BM に分散し、BM 内では行で分割して各 PE に分散する。配列 B は行で分割して各 BM に分配し、BM 内では各 PE にブロードキャストする。各 PE の計算結果を BM に戻し、リダクションネットワークを用いて要素ごとの総和を取って DM に結果 C を格納する。

各 PE の演算単位を A[8][8] x B[8] = C[8] とし、A はすでに各 PE に、B はすでに BM に配置されているとすると、B のブロードキャストに 8 クロック、演算に 40 クロックで行列ベクトル積を実行できるが、結果 C の BM への格納は、PE 毎にシリアル化されるため、512 クロックかかってしまう。

これを演算とオーバラップするためには、各 PE の演算単位を A[8][256] x B[256] = C[8] して演算部を増やすことが必要である。これにより、演算と、B のブロードキャストおよび結果 C の BM への格納をオーバラップさせることができる。このとき、1026 クロックで 2048 個の乗算と加算を実行でき、演算カーネル部では、ほぼピークの 4GFLOPS/PE、チップ全体で 16TFLOPS を実現できる。実際にはこの部分が B の列に対するループになり、最後の結果 C の BM への格納はオーバラップできないため、ループを 8 とすると、効率は 93 %となる。このループを大きくすると効率は向上する。

しかし、BM の容量の制限から最大 16 ループしか一度に実行できず、そのたびに B の DM からの分散転送、および C のリダクション結果の DM への転送が必要となる。演算が 16K クロックに対して、8GB/s のネットワーク性能では B の転送に 256K クロックかかるため、この転送も含めた実効性能は 6 %程度となってしまう。BM への転送をオーバラップすることは可能であるが、実行時間の違いが大きすぎるため、あまり性能向上は見込めない。

そこで、スタックメモリの利用を想定し、512GB/s のメモリ転送性能を仮定すると B の転送が 4K クロックとなり、C のリダクション 8K クロックと合わせても演算実行とオーバラップが可能になる。A の転送時間を考慮した性能は A[512][16K] x B[16K][512] の場合で効率は 78 %程度（12.5TFLOPS）である。B の列数が増えればさらに効率

は向上し、A の行数が増えても性能は維持できる。

2.2.4 n 体問題

演算重視のアプリケーションとして直接法による n 体問題の評価を行った。力を受ける粒子を i 粒子、力を与える粒子を j 粒子と呼び、粒子数 n を 32K として各 PE に i 粒子を 8 個づつ割り当て、j 粒子は全 PE にブロードキャストにより転送する。全粒子の位置データは各時間ステップごとに DM に集約され、次ステップの j 粒子データとして再転送される。j 粒子のブロードキャストは DM より BM へブロードキャストし、さらに BM から各 PE にブロードキャストされるが、BM の容量の制限から 4K 粒子毎に分割して転送と演算を繰り返した。ダブルバッファリングを行って、この転送と演算をオーバラップすることも可能であるが、転送のオーバヘッドが DM への集約を含めて 2 %程度なので、現在は行っていない。

粒子間の力の計算に平方根の逆数を求める必要があり、この演算を演算器としてサポートすることを検討しているが、現在のシミュレータでは実装されていないため、ニュートン法により計算している。この演算を含めて 8 粒子のベクトル演算を行うことにより、2 粒子間の演算を 1 ペアあたり 35 クロック程度で実行できており、32K 粒子の 1 時間ステップの計算に 9.3 ms かかっている。このうち、j 粒子の転送に 130 us、粒子位置の集約に 100 us かかっている。各 PE の粒子数が 64 を超えると、粒子位置の集約が BM サイズを超えるため、分割する必要が出てくるが、実行効率的には影響はない。

3. シミュレーション以外の評価

我々フィージビリティ調査チームでは、上記サイクルベースシミュレータによる評価のほかにも、さまざまなレベルで評価を行っている。以下にそれらについて概要を紹介する。

3.1 電力量評価

2018-19 年に稼働するシステムで実現できる電力あたりの性能を、高い精度で評価することを目的として、東工大と会津大を中心に、演算プロセッサコアをある程度の数集積した LSI の試作と演算器の低電力設計を進めている。

試作 LSI は TSMC の 28nm プロセス (28HPM プロセス) を利用することを前提としている。なお、TSMC の 2012 年時点のロードマップでは、10nm プロセスが 2016 年始めに量産移行するとしており、その次の世代が利用可能になる可能性もあるが、見積もりは保守的に 14nm を想定して行った。GRAPE-DR のアーキテクチャをベースに 28nm での電力の見積もりと、プロセッサ試作のための論理設計、物理設計を行った。GRAPE-DR プロセッサは TSMC の 90nm プロセスで製造したものである。プロセッサコアは、 1 サイクルで倍精度加減算 1 つ、単精度乗算 1 つを実行でき、 2 サイクル毎に倍精度乗算を実行する。消費電力は 512 コア、 500MHz 動作で 65W であり、倍精度で 4GFLOPS/W、単精度で 8GFLOPS/W を実現していた。なお、ホスト計算機と組み合わせたシステム全体では、HPL での実測で 1.5GFLOPS/W であった。GRAPE-DR コアに、改良を行った RTL レベル設計を用いて、予備的な電力評価を 28nm プロセスを用いて行った。改良点のうち電力消費削減および汎用化に寄与するのは以下の項目である。

- 倍精度演算に最適化した乗算器と追加の単精度専用の演算器を持つ構成に変更し、演算性能を 2 倍にし電力あたり性能を大きく向上させた。
- レジスタファイルを IP マクロを使って実装し、面積、消費電力とも大きく減少させた。
- 動作周波数の目標値を低めに設定し、高速だが消費電力、特に静的リーク電流が大きい低しきい値トランジスタの利用を可能な限り避けた。
- プロセッサコア 1 つあたりのメモリ容量を 2kB から 8kB に増加した。
- PE 間に 2D トーラスネットワークを追加した。

消費電力見積もりはまだ予備的なものであるが、いわゆる”Typical”値では 44GFLOPS/W を得た。これは、供給電圧、動作温度、プロセスパラメータ等を全て標準的な値で評価したものである。28nm プロセスから 14nm プロセスに移行することで、消費電力は半分ないしそれ以下になると考えられる。このため、プロセッサ・チップについては 14nm プロセスで 70-90 GFLOPS/W が実現できる見込みである。

3.2 チップ間ネットワーク

プロセッサ・チップはチップ間ネットワークにより直接接続されることを想定している。これについては、筑波大、日立を中心に検討した。

チップ間ネットワークで接続するチップ数は、実装可能性や信号伝達のレイテンシ等を勘案して、全システムではなく、数筐体に跨る程度、すなわち、2048 チップを 1 グループとすることとした。チップ間ネットワークは、チップ内ネットワークが 2 次元メッシュアーキテクチャであることを受け、基本的にこのメッシュ構造を拡張してシステム全体をトーラスネットワークとして構築する。

チップ間ネットワークのバンド幅については、2 次元トーラスの拡張であるため、基本的にはチップの 4 方向のエッジのネットワークリンクをチップ外に延長する形になるが、そのままのバンド幅でチップ外に引き出すことは極めて困難である。このため、各エッジに通信バッファを配置し、チップ間の通信はこの通信バッファを経由し、チップ内 PE の通信要求を集約した形で行う。エッジ当たりのチップ間ネットワーク用信号ピンは、システム実装目標時

期のテクノロジ予測より、最大で 50 Gbps/channel、1 エッジ当たり 32 channel 程度が想定される。よって、エッジ当たりの双方向通信性能は 400GB/s となり、これはエッジ当たりの 64PE からの通信性能要求（1TB/s）の 40 %に相当する。

実装を考慮するとチップを複数搭載した演算加速装置ボードを各筐体内に多数設置し、これらのボード間を結合するネットワークリンクを筐体内及び筐体間に実装する形を想定する。ボード内はプリント配線による結合、ボード間はコネクタ及びケーブルによる配線となる。ボード上の実装密度とケーブルコネクタの配置から、各ボードには 16 個のチップを 4 × 4 の 2 次元配置とする。よって、ボード間の 2 次元トーラスのための 4 方向の各エッジに 32 channel × 4（各方向 4 チップのため）=128channel の 50Gbps リンクを引き出すことにより、チップ間ネットワークのリンク形状及びバンド幅を維持したまま、ボード間ネットワークを実現可能である。ボード間ネットワークは光リンクを用いる。

このように、チップ内ネットワークはチップ外との入出力においてチップ内バンド幅の 40 %に縮約した通信性能制限がかかるものの、チップ間及びボード間は等しい形状とバンド幅を維持したトーラスネットワークとして構築できる。性能上の問題は、チップ間の通信性能縮約で、これはチップ間通信バッファを利用し、ソフトウェア及びアルゴリズム上の工夫により、ネットワーク利用を理論的に階層化し、バンド幅縮約の制約を受けないよう工夫する必要がある。

3.3 アプリケーション性能

大まかな性能予測に基づき、アプリケーションの概算性能の見積もりを行った。見積もりを行ったアプリケーションは以下のとおりであるが、ここでは東工大と筑波大を中心に行った差分法スキームについてのみ概要を示す。

- 格子 QCD 領域分割ハイブリッドモンテカルロ法単精度ソルバー
- 磁気流体計算コード
- 重力多体計算 (N-body)
- 分子動力学計算
- 全球雲解像モデル NICAM
- 差分法スキーム (FDTD 法)

FDTD 法は、弾性問題シミュレーションや電磁界シミュレーションに広く使われている差分法スキームである。ここでは差分精度は、時間 2 次、空間 4 次で扱い、弾性体バージョンに基づいて評価を行った。場の変数としては粒子速度 3 変数と応力テンソル 6 変数、物性パラメータとしては密度と弾性定数 2 個の 3 変数を持つ。食い違い格子を用いた演算を、粒子速度と応力テンソルについて交互に時間積分を行う。

性能予測のパラメータとしては、以下を仮定した。

- 性能 /コア 単精度演算 8 GFLOPS
- オンチップメモリ /コア 128 KB (32K 単精度)
- オンチップメモリバンド幅 /コア 16 GB/s (2B/単精度 FLOP)
- 通信バンド幅 /コア（2D 隣接間ネットワーク）2D: 8 GB/s, 3D: 2 GB/s, 4D: 1 GB/s
- 行あるいは列方向に離れた場所への転送 4GB/s（ただし行あるいは列内では 1 コアのみ転送可能）
- コア /チップ 4096 (64 × 64 の 2D メッシュ接続)
- 通信バンド幅 /チップ（2D 隣接間ネットワーク, 4port/chip）2D: 409GB/s/port 4port 同時双方向通信可能

プログラムの解析などはサイクルベースシミュレータによる評価で述べているので省略し、結果および複数チップでの性能見積もりについて述べる。

時間発展ループ内には、粒子速度計算と応力テンソル計算のループがあり、それぞれについてメモリ参照回数と演算回数を見積もり、それぞれがボトルネックとなる場合の性能を評価した。その結果、メモリ参照がボトルネックになるとわかった。このとき、それぞれ 9.3us、および 10.7us が 1 時間ステップにかかる。また、ループ内の通信時間には、PE 間の袖領域の転送と、周期境界条件のためのデータ転送がある。転送に必要なデータ量及び転送パターンを見積もり、各ループにおける転送時間はそれぞれ 21.5 (9.2 + 12.3)us、10.7 (4.6 + 6.1)us が 1 時間ステップにかかる。合計で 1 時間ステップあたり 52.3 us かかり、コアあたりの演算性能は 1.4GFLOPS となる。この見積もりは、サイクルベースシミュレータによる評価 59.0 us に近く、この方法による見積もりが十分であることを裏付けている。

複数チップにおける性能見積もりとして、簡単のために z 次元はチップ内で閉じるとする。この場合 x、y 次元はチップ間で物理的に隣接になる。ただし、x、y 次元の他チップとの隣接コアは 4 列あるため、各コアから通信バッファまでの転送性能の見積もりは、周期境界条件のデータ転送と同等になる。すなわち、粒子計算と応力テンソル計算を合わせて 18.4 us となる。チップ間の転送は 4 方向同時転送が可能であるとすると、1 方向のデータ量と転送バンド幅より、転送時間は粒子計算と応力テンソル計算を合わせて 2.9us となる。周期境界条件のためのデータ転送は複数チップ間にまたがる。このチップ間転送に要する時間を、ここでは仮に隣接通信の倍の時間と見積もって 36.9 us とする。これらの見積もりにより、1 時間ステップあたり 92.0 us かかり、この時のコアあたりの演算性能は 0.8 GFLOPS、チップ全体の演算性能は 3.4 TFLOPS となる。256 チップを組合わせたときには全体で 2048 × 2048 × 128 のデータセルに対して 861 TFLOPS の性能となる。

4. 今後の評価

現在のサイクルベースシミュレータのアーキテクチャは、たたき台としての評価のためのものであり、今後改良を進めていく予定である。

チップ間ネットワークの評価に合わせて、チップ内通信バッファの詳細化などを行い、サイクルベースシミュレータに組み込んで、複数チップの評価をできるだけ早急に行いたい。また、アプリケーション評価から、単純な 2 次元メッシュの隣接通信だけではなく、任意のホップ数を持つ PE 間の直接通信を行ないたいという要望もある。レイアウトレベルの評価からはビット幅を 8 ビット程度に制限することにより、次元内のルーティングが可能ではないかという評価もあり、今後チップ内ネットワークについては検討を行っていく予定である。

また、現在は 1 命令 4 サイクルのベクトル実行を基本としているが、倍精度で 8 個、単精度で 16 個のベクトル化を有効に利用するのは実際には厳しい。2.2 節で行った評価ではなんとかうまくベクトル化が行なえたが、128K バイトのローカルメモリに多数の変数を要求するような実アプリケーションでは、1PE で実行する格子サイズが小さくなるため、さらに厳しくなると予測される。さらに、袖領域の通信を隠ぺいするために、袖領域部分を先に処理して転送を早めに開始する場合は、それだけベクトル長が短くなる。そのような処理に対応するため、1 命令 4 サイクルのベクトル実行はやめて、1 命令 1 サイクルの通常の命令実行とすることを検討する。ただし、回路をシンプルに維持するために、演算器のパイプラインレイテンシの隠ぺいは命令レベルの静的スケジューリングで対応することが重要であり、そのための一時レジスタの追加などが必要となる。

プログラミングモデルの検討も重要な検討項目である。アクセラレータの利用を容易にするための方式としては、OpenACC のようなディレクティブベースのオフローディングを検討している。現在、そのためのコンパイラなどを開発中であるが、そこで想定した命令セットアーキテクチャと、サイクルベースシミュレーションで想定した命令セットアーキテクチャで、一部不整合が生じている。現在、この 2 つの命令セットアーキテクチャの統合を行っており、その結果をサイクルベースシミュレータにも反映させていく。

このようなアーキテクチャの改良で難しいのが、ナイーブな並列実装で評価していては正しくアーキテクチャの評価が行なえない点である。アーキテクチャ上のトレードオフを評価するためには、それぞれのアーキテクチャに向けた最適化を施したうえでの評価を行わないといけない。2.2 節で行った評価ではまだ通信の隠ぺいなどの最適化が適用されていない部分があるため、それらの最適化を行ったうえでの評価を行っていく必要がある。

5. まとめ

将来のアクセラレータとして検討している大規模 SIMD 型プロセッサについて、そのアーキテクチャの基本設計方針や、性能パラメータの検討結果を示した。より詳細な評価を行うために、検討のたたき台としてのアーキテクチャの設計を行い、そのアーキテクチャについて紹介するとともに、そのアーキテクチャに基づくサイクルベースシミュレータを作成した。このシミュレータを用いて、いくつかのベンチマークプログラムを実行し、本アクセラレータのチップ単体での性能見積もりを行った。今後は、チップ間ネットワークをシミュレータに実装し、複数チップの評価を行っていくとともに、本アーキテクチャの改良を検討し、本アクセラレータが適用可能なアプリケーションの拡大を検討していく。

謝辞 本研究は、文部科学省「将来の HPCI システムのあり方の調査研究」（平成 24 年度～平成 25 年度）の支援による。

参考文献

[1] http://www.top500.org/
[2] HPCI 技術ロードマップ白書, 2012 年 3 月. http://open-supercomputer.org/wp-content/uploads/2012/03/hpci-roadmap.pdf
[3] Junichiro Makino, Hiroshi Daisaka, Toshiyuki Fukushige, Yutaka Sugawara, Mary Inaba, Kei Hiraki, “The performance of GRAPE-DR for dense matrix operations,” Proc. of the International Conference on Computational Science, ICCS 2011, pp.888-897.
[4] Hybrid Memory Cube Consortium. http://www.hybridmemorycube.org/
[5] 姫野ベンチマーク. http://accc.riken.jp/2145.htm