Vol.2010-ARC-190 No.3
2010/8/3

コヒーレントでないメモリシステムへのアーキテクチャ支援

泊　久信[†1]　平木　敬[†1]

マルチコア化が進み、数千を越えるプロセッサコアが 1 つのチップに集積されるようになると、従来同様の対称型で共有メモリを実現するようなアーキテクチャを実現するのは困難になる。これは、チップ外に実装されるメモリとの接続の面と、チップ内の通信がキャッシュの更新の通信で溢れてしまうという問題からである。本研究では、こういった困難を克服するアーキテクチャとしてシストリック・アレイの拡張を提案し、エミュレータの実装を報告する。最後に、FFT を当アーキテクチャで動作させる際のアルゴリズムを示し、実際に本アーキテクチャが高性能計算の高速化に有用であることを示す。

Scalable Computer Architecture without Cache Coherency Support

Hisanobu Tomari[†1] and Kei Hiraki[†1]

As multicore designs gain popularity, it becomes difficult to support cache coherency among symmetric processor cores when the core count hits over 1,000. One cause of the problem is the lack of sufficient inter-chip I/O. Another reason is overwhelming number of intra-chip communication. In this paper, we propose a modified variant of systolic array as a solution to these problem. We developed a software emulator of our architecture. Finally, we showed an optimized FFT for our architecture to prove that this architecture actually contributes to speed increase of numerical applications.

†1 東京大学 大学院 情報理工学系研究科 コンピュータ科学専攻
Deptartment of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo

1. はじめに

マルチコアの設計は普遍的にみられるようになり、1 つのチップに集積されるプロセサ・コアの数は増加している。今後もコア数は増加することが予想され、数千コアがひとつのチップに収まるようになると予想されている[3]。一方で、共有メモリで動作するマシンは、大きな規模のものを作るのが困難で、そのプロセサ数は NUMA 構成のアーキテクチャを含めても、数千程度までとなっている[8)10)]。数千を越えるコアを 1 つのチップに集積する際にも、こういったネットワークで接続された大規模な共有メモリマシンを作る際と同じ問題がある。問題のうちの一つは、数千のコアを対象にチップ外のメモリシステムに接続するのは困難か不可能であるという点である。もう一つは、コアの性能を引き出すために利用されるキャッシュメモリのコヒーレンシを数千あるコアで確保するのが困難であるという点である。

本研究では、数千のプロセッサコアが 1 つのチップに集積することを、シストリックアレイを拡張することで効率的に実現することを提案する。シストリックアレイでは、入出力は端の PE からのみ行われる。共有メモリでは全プロセサからメモリにアクセスするが、実際のハードウェアでは電力や面積の制約から実装されるメモリコントローラの数はプロセサ数より小さくならざるをえない[7]。また、外部メモリへのアクセスが多いことは消費電力の面で不利である[5]。シストリックアレイでは、通信は直接接続された少数の PE とのみ行われる。一方、共有メモリを実現するためには、各プロセッサ同士が通信できるような機構が要求され、ルーティングを含めたネットワークをチップ上で実現することになる。以上のように、共有メモリを実現する場合に比べて、シストリックアレイは実際のハードウェアの特徴をそのままコンピュータ・アーキテクチャとして実現している。このことによって、ユーザプログラムはハードウェアをより効率的に利用できるものになり、性能の向上・消費電力の削減が期待できる。

シストリックアレイの PE をそのまま高機能な CPU に置き換えると、PE 間の通信が CPU の性能につり合わなくなる。Warp のような 1 次元の接続だと、PE の数を増やしたときに入力から出力までのレイテンシが大きくなってしまうのと、解ける問題の大きさが PE 1 つのメモリ量に依存してしまう[1]。iWarp は 2 次元の接続を実現しているが[2]、近隣の PE とのみの接続であり、PE の数が増えたときに、近傍にない PE にデータを伝播するときの hop 数が大きくなってしまい、今日の高性能計算で求められるアルゴリズムを実行するのに適さなくなる。以上のような問題を克服するために、従来のシストリックアレイを

大きな 2 次元のアレイにしても離れた PE とのデータの交換が効率的に行え、また高機能な PE に合わせて、計算と PE 間通信が並行して行えるように拡張した。

本論文では、第 2 章において高機能なプロセッサをシストリックアレイの PE として利用するために施したシストリックアレイの拡張、およびシミュレーション等の高性能計算で応用する際の利便性を考慮して施した PE 間結合の拡張について述べる。第 3 章では、本アーキテクチャの特性を調べるため実装したエミュレータの設計と実装について説明する。第 4 章では、高性能計算の例として、当アーキテクチャ上で高速フーリエ変換を動作させる際のアルゴリズムおよびプログラム可能性について考察する。

2. 高機能なプロセッサによるシストリックアレイ

プロセッサコアの数がチップの中で増えたときに問題になるのは、外部のメモリへのアクセスが多くなりすぎてしまうという点と、コアに固有なキャッシュの内容を維持するための通信がコアの数に比例して増加し、実際にデータを移動するための帯域を圧迫することである。シストリックアレイは、プロセサ (CPU) を一つ含むチップの構成要素 Processing Element (PE) を基本的な構成要素とする。チップ外部とのデータの入出力は端にある少数の PE からのみ行われるため、内部の大量の PE が外部メモリにアクセスすることはない。また、CPU は計算を PE 内のメモリをのみを使って行い、他の PE との通信は共有メモリのときと異なり、ローカルにのみ発生するため、チップ内の帯域を有効に利用できる。

図 1 に PE の構成を示す。 PE 内には、CPU の他にメモリが含まれている。まず、計算時のスタックやヒープなどを置くことができる Local Memory (LM) という領域がある。LM は各 PE 固有のメモリ領域で、他の PE からはアクセスすることができない。次に、データの受け渡しに利用する Input Buffer (IB) がある。IB は、ひとつ前のランクの PE と通信をするためのメモリ領域で、ダブルバッファリングして、PE 内での計算と他 PE からのデータの受信が同時に行えるように 2 バンク存在する。ある時点で PE 内の CPU から見えるのは片方のバンクのみである。どちらのバンクが PE 内の CPU から見え、外から書き込みを受け付けるかは PE 内の CPU が制御する。最後に、プログラムや定数を記録しておく ROM がある。通常プログラムはここに配置し、動作中には書き換えない。

シストリックアレイでは、データは一連の PE 列を通っていきながら計算される。データが通っていくための、PE を一列に並べたものをパイプと呼ぶ。パイプ内の PE は順序が付いている。ランク 0 の PE の IB はチップの入力に繋がっており、パイプ内で最大のランクの PE の出力はチップの出力に繋がっている。ランク n の PE の出力はランク $n+1$ の PE

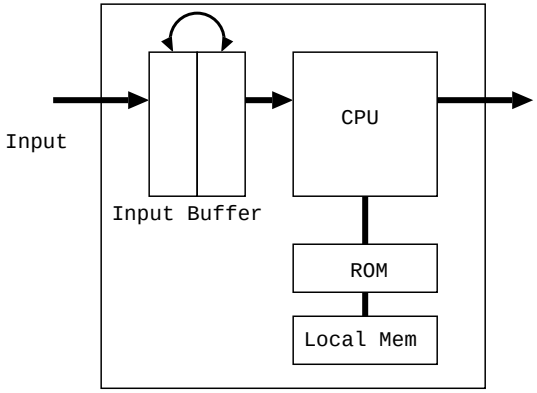

図 1 PE の構成

内の IB 入力に繋がっていて、計算は左端から右端まで PE がデータを受け渡しながら行う。

2.1 パイプ間結合

多くの PE を効率的に利用できるように結合するために、複数本のパイプを束ねることになる。前述の通り、パイプ内の PE は 1 次元の結合を持っていて、それぞれの PE は自分より 1 つ大きなランクのの PE にデータを移動することができる。各パイプのランク n の PE が全パイプのランク $n+1$ の PE に結合しているのが理想だが、n_p 本のパイプについてエッジの数が $O({n_p}^2)$ となり、パイプの数を増やすとネットワークに使うリソースが大きくなりすぎてしまう。隣のパイプとのみ結合する場合だと、高速フーリエ変換など、遠くのノードとの通信が発生する場合、目的のパイプにデータが到着するのにパイプ数分だけ先のランクまでルーティングを繰り返さないといけない。

多段結合網の構成を利用することで、ハードウェア量を抑えつつパイプ間で高速に全対全通信を実現することができる[11]。多段結合網は複数段に Switching Element を配置して、Switching Element の間を Shuffle Exchange Network で結ぶことで、n 本のパイプを $\log_2 n$ 段で繋ぐ[16]。ハードウェアの複雑さの削減のため、各 Switching Element に PE を対応させた。こうすると、ルーティングにかかる $\log_2 n$ ランクは計算をできないことになるが、複数の問題を同時に処理することでルーティングに使うランクはすべてスループットの向上に役立てることができる。PE には入力・出力が自パイプ内の PE との結合用以外に 2 本ずつ必要になる。この場合、他パイプとの PE の接続は、自分のパイプ番号と出力ポー

Vol.2010-ARC-190 No.3
2010/8/3

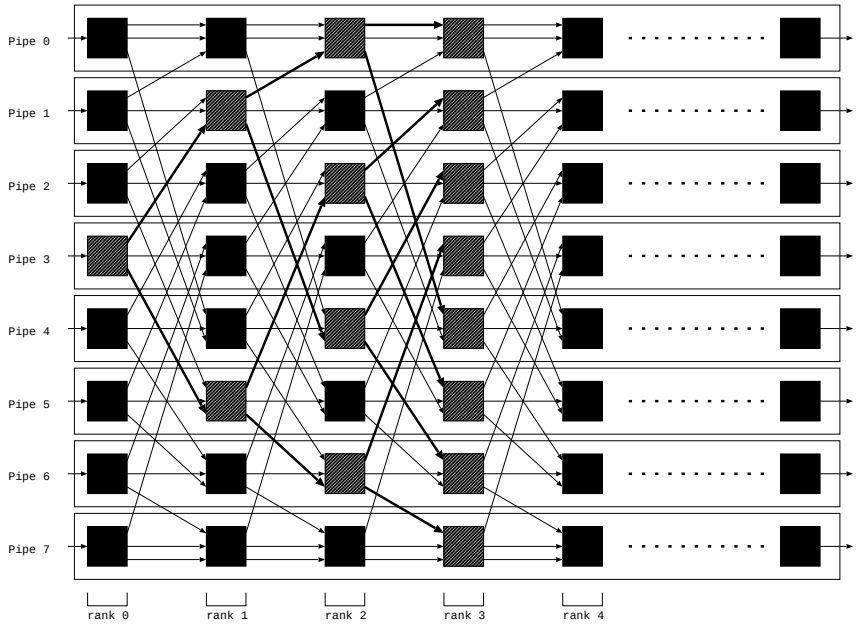

図 2 8 パイプ構成時のパイプ間・パイプ内接続

ト番号 0, 1 を連接させた値の、最上位ビットが相手先ポート番号、それ以外のビットが相手先パイプ番号に相当することになる。パイプ番号 1010_2 の出力ポート 1_2 は、次ランクのパイプ番号 0101_2 の PE の入力ポート 1_2 と接続される。16 パイプ構成だと、あるパイプから任意の他のパイプにデータを送るのに 4 ランク必要になる。以上のような構成にすることで、ある PE から見えるメモリ空間には以下の 4 種類の領域が含まれることになる。

(1) ローカルメモリ
(2) ROM
(3) 同パイプの次ランクの IB
(4) 他パイプの次ランクの IB 2 バンク

図 2 に、8 パイプのときのパイプ間接続とパイプ内接続の例を示す。正方形が PE を示している。パイプ 3 がランク 0 で他のパイプにデータをコピーする際は、太線のエッジを使うことになる。8 パイプの時は、$\log_2 8 = 3$ なので、3 ランク先の任意のノードにデータをコピーするパスが存在することになる。網掛けされている PE がパイプ 3, ランク 0 からのデータを受け取ることができる PE である。配線がすべてローカルになっている点は、実際のチップの設計の上で有理である。ただし、PE の配置はランクごとに PE を密集させたタイルを配置した方が、図 2 のように PE を配置するよりも配線が短い可能性がある。

ルーティングはハードウェアでは行わず、PE の CPU が IB からデータを出力に書き込み動作することで、ソフトウェアにより行う。このことにより、ルーティングの途中の PE でデータをノーマライズする、といった柔軟なプログラムを書くこともできるし、ルーティングノードはルーティングするでー他と依存性のない別の問題を解くようなプログラミングようになり、ハードウェア利用効率を高めるためのアルゴリズムによる工夫を可能にしている。

この接続形態は、従来のシストリックアレイと類似性がある。従来の 2 次元正方形のシストリックアレイのうち、上端と左端からデータの入力、右端と下端からデータの出力をするものにおいて、各 PE を左側が x 方向、下側が y 方向として、(x, y) とアドレス付けする。この時、各 PE について、$k = x + y$ として、k をランクとすれば、このパイプ間接続は従来のシストリックアレイの PE 間のネットワークを、より遠くのノードに少ないランクで接続できるようにしたものであることが明らかになる。

2.2 同 期 機 構

複数のプロセサを使って計算をする場合、プロセサ間の同期機構が必要になる。全ての PE が同じメモリ領域を共有する場合と異なり、上記方法で結合したランク r, パイプ p の PE の同期は、直接接続されているランク $r-1$, パイプ p の PE と、パイプ間結合で Shuffle Exchange 相当の接続がされているランク $r-1$ の 2 つの PE との操作に限定できる。すべての PE の通信があるランクのものから次のランクの PE への書き込みの通信であって、IB のバンク切り替えはその IB を含む PE 内の CPU から操作可能なレジスタを用いる。以上のような PE の接続の特徴により、同期は以下のように行う。

まず、PE D に書き込み可能な PE A, B, C は、PE D の IB で現在書き込み側に存在する IB のバンク番号を常に読み出すことが可能である。A, B, C は、それぞれ D が参照できる領域にフラグを書き込める。ここに 0 以外の値を書き込むことで A, B, C は D にデータの書き込みが終了したことを通知することができる。D は D での処理が終了すると、A, B, C が書き込んだフラグを参照し、すべて 0 以外が書き込まれていれば、IB をバンク切り替えし、書き込まれたデータを使って次の計算をする。バンク切り替えと同時に全てのフラグは D によりクリアされる。バンク切り替えが起こると、A, B, C が書き込みを行える

バンクが切り替わるので、A, B, C は書き込みするバンク番号を読み出し、それが変化することをもって D の次データ受付の準備が整ったことを確認することができるようになっている。

以上のように、一般的な CPU を使ったプロセサ・アレイを構成できる。この構成は、シストリックアレイの利点を継承している[9]。まず、単純な構造の繰り返しになっている点。PE の構造はすべての PE で共通である。パイプ間接続のネットワークの構造も、どのランクのネットワークかによらず同じ構造の繰り返しになっている。次に、アルゴリズムの並列性を生かせる点は、本来のシストリックアレイ以上に高機能な PE を搭載することで強化されている。最後に、計算の能力と I/O が均衡をとることができる点も、PE の接続形態より、両端のみ I/O をするというプログラミングをプログラマに強制するので、自然に含まれている。逆に、プログラマが書けることを制限することで、効率の良い動作を可能にする。

3. 評　　価

提案した構成のシステムの動作を確認するために、ソフトウェアによるエミュレータを作成した。現在使われているマルチコアチップより大規模で、エミュレーションが可能な大きさとして 4,096 PE を想定する。これを実現するため、エミュレータは高速である必要がある。また、振る舞いを観察する上で、エミュレーションはできる限り正確なものである必要がある。作成したエミュレータは、Multiple Arcade Machine Emulator (MAME) をベースにした。MAME は本来、数千のアーケードゲーム機を正確にエミュレーションすることを目的に作られたソフトウェアである[13]。アーケードゲーム基板は、製造した会社や時期によって使用される CPU の種類や数が異なり、それらのアーキテクチャをエミュレートするために、MAME はかなり柔軟なエミュレーションシステムを持っている。

まず、物理的な基板に対応するエンティティとして、ドライバと呼ばれるコード部分がある。ドライバは、複数のゲームに対応し、各ゲームに対応する ROM イメージのファイル名とアドレス空間との関連付け、各ゲームでの CPU の種類と数、メモリの領域とエミュレーション処理との対応を定義する。アドレス空間は複数定義することができ、これを複数個の CPU で共有することも、分離することも、一部共有するといったことも可能である。CPU は複数種・複数個同時にエミュレートすることが可能で、クロック周波数をそれぞれ宣言する必要がある。これによって、複数のプロセッサが実際の基板上にあるときと同じタイミングで処理を行うことが可能になる。ドライバはそのほかに、デコーダの PLD など、基板特有のデバイスのエミュレーションコード、およびメモリアドレス領域の読み込みハンドラ・書き込みハンドラや、割り込み発生などを定義できる。

ドライバが各ゲーム毎に異なるのに対して、CPU や音源 IC、一部のグラフィック処理チップなどは複数の基板で共通して使われている。この部分は、Sinclair ZX-81 や PC-8801 の世代のコンピュータをエミュレートできるソフトウェア: MESS と共通のコードを使っている[12]。エミュレーションできる CPU として i386, 68000, MIPS, SH-2 を含む多種をサポートしている。MAME/MESS が利用する 68000 のエミュレータは、Karl Stenerud による MUSASHI がベースになっており、68000 から 68040 まで、一部割り込みを除いて FPU を含めてエミュレーション可能なものになっている。サイクル単位で正確なエミュレーションを提供するうえ、多くのゲーム基板で 68000 が使われていることによりエミュレーションの正確性が検証されているので、本研究は 68000 シリーズを使ってエミュレーションを行った。また、68000 のエミュレーションは比較的高速で、エミュレーションの実行時間が短縮できることが期待できる。68040 は FPU を内蔵しているので、68040 シリーズのエミュレーションを利用することにした。以上のような理由により、今回は 68040 を選択したが、エミュレータの設計上、今後 68000 ではなく MIPS にする、といった改変は比較的簡単に行うことができると考えられる。

基礎として利用した MAME のバージョンは 0.137u2 である。このバージョンは、以前ソースが分けられていた SDLMAME とソースが統合されているバージョンで、Linux 上でそのままのソースを使うことができる。このバージョンの MAME の欠点は、複数の CPU が存在するとき、グラフィックのレンダリングと CPU のエミュレーションは部分的に複数スレッドで動作するが、それぞれの CPU が複数のスレッドになることはなく、単一スレッドのみで動作する点である。こうすることで完全にサイクルアキュレートで正確なエミュレーションを可能にしている一方、本研究で目的としている大規模なマルチコアプロセサのエミュレーションへの応用という点では、マルチプロセッサシステムをホストにエミュレーションしても性能向上がほとんどないという点で不利である。また、MAME のメモリ管理として、MAME 内部でバンクと呼ばれるメモリ領域の割り当て・開放を行っており、8 ビットの管理番号を使ってこのバンクを管理している。エミュレートする RAM や ROM の領域は、このバンクから割り当てられる。オリジナルの MAME だと百数十の領域しか割り当てることができないので、PE 毎に ROM と RAM が存在する場合だと、割り当てられる領域以上には PE をインスタンシエートできない。4,096 プロセサを 1 つの実行イメージで処理しようとすると、以上のような困難がある。

これを実現するために、1 つの MAME 実行イメージで 128 PE をエミュレートした。32

プロセス MAME を動作させることで、合計で 4,096 PE になる。1 つの MAME 実行イメージ中で、ドライバとして 128 PE のインスタンシエートと、従来の MAME のメモリ管理をスキップして直接標準ライブラリを呼び出す部分、および他 MAME プロセスと TCP を使って通信する部分を実装した。このようにすることで、複数のホストプロセサを活用できない欠点、および貧弱なメモリ管理により多量の PE をインスタンシエートできない問題を克服した。256 PE で一つのパイプを構成し、16 パイプを束ねることで合計 4,096 PE となる。PE 1 つあたりの LM は 512 KB で、 IB は 1 バンクあたり 256 KB とした。ROM は実験用途では 64 KB で十分だった。

エミュレータで動作するソフトウェアの開発には、GCC 4.5.0 と Binutils 2.20.1 を利用した。PE は ROM からプログラムを実行するようになっている。また、MAME もエミュレートするプログラムを別途 ROM イメージという形で MAME 実行時に読み出すのが基本的な動作となっているので、GCC でコンパイルし、リンクした ELF 形式の実行イメージを、objcopy を使って ELF のヘッダ情報を取り除き、ROM イメージに変換したものが利用できる。

4. 高速フーリエ変換の実装

本アーキテクチャでプログラムを実行するには、通信が常に 1 方向であることに気を付けてアルゴリズムを設計する必要がある。一方で、各 PE に内蔵される CPU コアはフルセットのものなので、SIMD のマシンに最適化してプログラムを組むときとは異なる点で、通常のプログラムと違う工夫が必要になる。分岐があっても、全てのランクで均一な計算時間がかかるようなものであれば、同一ランクでまったく異なる計算を行っても性能は低下しない。一方で、全ての PE で、データの出力までにかかる時間がほぼ同じでないと、あるランクのあるノードが処理に時間がかかって、次ランクの多くの PE が同期待ちになるだけでなく、そのランク以前の PE もデータ受け渡しで同期待ちになってしまい、チップ全体の効率が極端に落ちてしまう。

高速フーリエ変換 (FFT) は、数値計算プログラムでの普遍性と、高速化の工夫の種類の多さから、ベンチマークのプログラムの題材として人気がある。本研究では、Cooley-Tukey のアルゴリズムを実装した[6]。Cooley-Tukey のアルゴリズムでは、バタフライ演算を行うので、各ステップごとにデータの交換の必要があり、これが性能を出しにくい原因の一つになっている。本アーキテクチャでは、Shuffle Network を利用してパイプ間を接続しているため、データ交換を効率的に行うことができる。また、比較的大きな LM と IB を PE 毎

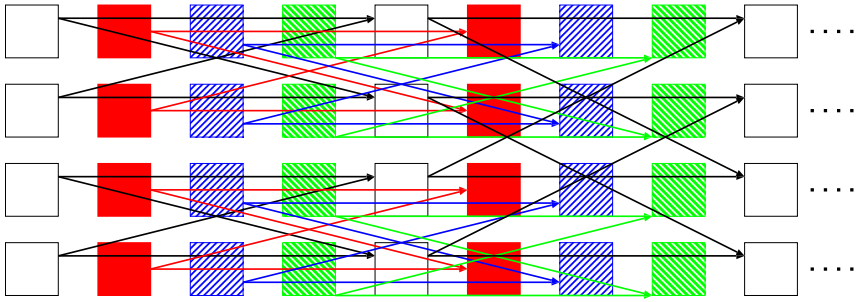

図 3 複数の高速フーリエ変換を解く

に持っているので、1 つの PE が担当する要素数を、問題の規模に合わせて調節することができる。一方で、第 2 章で述べた通り、Switching Element を PE に対応させているので、あるランクで計算した結果を次のランクですぐには使えない。データを移動するのでパイプ数 n_p に対し $\log_2 n_p$ ランク先までは計算を行えなくなってしまう。しかし、複数の FFT を同時に解き、FFT の問題を解くスループットを高めるためにこの途中の PE を使うことができる。図 3 は、4 つより先のランクとしか自由にデータが交換できない場合に、4 つの FFT を同時に計算する際のバタフライダイアグラムである。正方形が 1 つの PE を示していて、矢印はデータの移動を示している。ただし、実際にどういったパスを通っるかと、白以外のデータの 2 回目以降のデータ移動は簡便のため省略した。データが移動されるかは描いていない。左側からランク $0, 1, 2, \ldots$ とすると、ランク $4n$, $4n+1$, $4n+2$, $4n+3$ は別の問題を解いているが、全ての PE はデータをルーティングするために他の問題のデータも IB から読み込み適切なポートに出力するプログラムを含む必要がある。こうすることで、全ての PE の負荷が均一になり、PE の利用効率を大幅に高めることができる。

本アルゴリズムは AMD64 プラットフォーム上の Linux で動くシミュレータ上で動作を検証した。シミュレータは、エミュレータと異なり C 言語のサブルーチンを書くことで本アーキテクチャの動作を模倣するものである。

5. 関 連 研 究

Carlson ら[4] は、Shuffle Exchange Network を使って PE を接続することで、大きな規模の問題を少数の PE で効率的に処理できることを示した。本研究では、多数の PE を Shuffle Exchange Network を使って接続することで、プログラムのスループットを効率よく上げる

ことができることを示した。Midorikawa ら[17] は、MIN を含む複数のトポロジで CPU を接続する並列計算機を作った。本研究は、MIN のスイッチングエレメントに相当する部分で計算を行っている。

柴村ら[18] は、ネットワークの振舞を予測するための大規模並列システムのエミュレータを作成した。これは、計算部分にはディレイを宣言することで高速にネットワークをエミュレーションするものである。本研究で作成したエミュレータは、ネットワークに加え PE での計算もエミュレートすることができるものである。

Sedukhin[15] は、フーリエ変換をシストリックアレイで計算する方法を提案した。この場合、PE が DFT のための専用のものを使っているが、本研究では PE は高機能なプロセッサであり、数値計算の幅広い応用で高性能が出せる工夫を施した。

6. ま と め

本研究では、数千を越える多数のコアを 1 つのチップに集積する際のスケーラブルなアーキテクチャとしてシストリックアレイを提案した。従来のシストリックアレイをそのまま適用すると不利になると考えられる、プロセサが性能向上しても通信と計算を並行して行えるようにするためのダブルバッファリング機構と、多数の PE を小さなハードウェアコストで小 hop 数で通信を実現するために PE 間ネットワークに Shuffle Exchange Network を利用する方法を示した。シストリックアレイをベースとすることで、PE 数に関して、チップ外への I/O とチップ内の通信に割かれる資源がうまくバランスのとれたアーキテクチャを構成した。さらに、本アーキテクチャは PE 間の通信路がすべてローカルなので配線長が短くなり、高性能を出し易いという利点と、入出力の構成が 1 次元的になるので、複数のチップを連結してランク数を増やすことが容易にできるという利点もある。

エミュレータを柔軟なシステムを持つ MAME を改変しその上に実装することで、実際に提案した方式で動作するシステムを構成可能なことを示した。実際の数値計算のアルゴリズムとして高速フーリエ変換をとりあげ、これを提案したアーキテクチャで効率的に動作させる方法を示し、シミュレータでチップ外との I/O を増やすことなく、PE に問題を分割することでスループットを上げられることを示した。

今後、エミュレータを改良して様々なプログラムを動作させられるようにすることが必要である。Shuffle Exchange Network 以外にも、PE を接続するネットワークとして有用であると考えられる他の方法についても、本システムでの有効性を確かめる必要がある[14]。また、実際に FPGA 上で単純化した CPU を使って数百 PE 規模のシステムを実装し、どういった点がハードウェアの実装の上で改善の必要があるのかも確認する必要がある。

参 考 文 献

1) M. Annaratone, E. Arnould, T. Gross, H. T. Kung, and M. Lam. The Warp computer: Architecture, implementation, and performance. *IEEE Trans. Comput.*, 36(12):1523–1538, 1987.
2) S. Borkar, R. Cohn, G. Cox, S. Gleason, and T. Gross. iwarp: an integrated solution of high-speed parallel computing. In *Supercomputing '88: Proceedings of the 1988 ACM/IEEE conference on Supercomputing*, pages 330–339, Los Alamitos, CA, USA, 1988. IEEE Computer Society Press.
3) Shekhar Borkar. Thousand core chips: a technology perspective. In *DAC '07: Proceedings of the 44th annual Design Automation Conference*, pages 746–749, New York, NY, USA, 2007. ACM.
4) DavidA Carlson and Binay Sugla. Adapting shuffle-exchange like parallel processing organizations to work as systolic arrays. *Parallel Computing*, 11(1):93 – 106, 1989.
5) Sylvain Collange, David Defour, and Arnaud Tisserand. Power consumption of GPUs from a software perspective. In *ICCS '09: Proceedings of the 9th International Conference on Computational Science*, pages 914–923, Berlin, Heidelberg, 2009. Springer-Verlag.
6) James Cooley and John Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.
7) Jim Held. Single-chip cloud computer — an experimental many-core processor from Intel Labs. *Intel Labs Single-chip Cloud Computer Symposium*, 2010.
8) Yasushi Kanoh, Tetsuya Hirose, Masaaki Nakamura, Takeo Hosomi, Kosuke Tatsukawa, Hiroyuki Araki, Tomoyoshi Sugawara, and Toshiyuki Nakata. Architecture of a parallel computer Cenju-4. *Innovative Architecture for Future Generation High-Performance Processors and Systems, International Workshop on*, 0:105, 1998.
9) H.T. Kung. Why systolic architectures? *Computer*, 15(1):37–46, 1982.
10) James Laudon and Daniel Lenoski. The SGI Origin: a ccNUMA highly scalable server. In *ISCA '97: Proceedings of the 24th annual international symposium on Computer architecture*, pages 241–251, New York, NY, USA, 1997. ACM.
11) D.H. Lawrie. Access and alignment of data in an array processor. *Computers, IEEE Transactions on*, C-24(12):1145 – 1155, dec. 1975.
12) The MESS team. The official MESS home page. 2010. `http://www.mess.org/`.
13) Nicola Salmoria and the MAME team. MAME — multiple arcade machine emulator. 2010. `http://mamedev.org/`.
14) M.R. Samatham and D.K. Pradhan. The de Bruijn multiprocessor network: A

Vol.2010-ARC-190 No.3
2010/8/3

versatile parallel processing and sorting network for VLSI. *IEEE Transactions on Computers*, 38:567–581, 1989.

15) S. G. Sedukhin. Systolic array architecture for two-dimensional discrete fourier transform. In *CONPAR 90 — VAPP IV*, pages 682–691, 1990.

16) H.S. Stone. Parallel processing with the perfect shuffle. *Computers, IEEE Transactions on*, C-20(2):153 – 161, feb. 1971.

17) T.Midorikawa, D.Shiraishi, M.Shigeno, Y.Tanabe, T.Hanawa, and Amano H. SNAIL-2: a SSS-MIN connected multiprocessor with cache coherent mechanism. In *the 3rd International Conference on Parallel and Distributed Computing Applications and Technologies*, pages 17–24, 2002.

18) 柴村 英智, 薄田 竜太郎, 本田 宏明, 稲富 雄一, 于雲青, 井上 弘士, and 青柳 睦. PSINSIM：大規模並列システムの性能解析に向けた並列相互結合網シミュレータ. *IEICE technical report. Computer systems*, 107(276):45–50, 2007.