Vol.2012-ARC-201 No.6
2012/8/1

1600万計算コア超メニーコアアーキテクチャのシミュレーション

泊 久信[1,a)]　平木 敬[1,b)]

概要：8080 と SH-2 プロセッサを用い、メニーコアアーキテクチャで評価した。メニーコアでは、複雑なプロセッサコアを用いると搭載できるコア数が少なくなってしまうため、従来のプロセッサより小型のコアが用いられることがある。コア数を最大化するため単純化されたコアを用いると、コアあたりの性能が下がるため、コア数とコアあたりの性能を勘案して、性能の総和が大きくなるような設計にする必要がある。もっとも単純なプロセッサの一つである 8080 と、パイプライン動作をするが小型の実装が可能な SH-2 を用い、メニーコアアーキテクチャに当てはめて性能を評価することで、超メニーコア時代に最適なプロセッサコアのバランスを調べた。

Simulation of a Many-Core Architecture with 16 Million Processing Cores

Hisanobu Tomari[1,a)]　Kei Hiraki[1,b)]

Abstract: 8080 and SH-2 processors are evaluated as building blocks for a many-core architecture. In many-core architecture processor core designs simpler than conventional ones are often used because the number of processing elements that are integrated on a chip is limited by the size of the processor core. A many-core system design intends to maximize the throughput of instruction execution through the balance between the number of processor cores and the performance of a processor core. We put the 8080, which is one of the simplest processors, and the SH-2 pipelined processor in our many-core design to examine the optimal balance of simplicity and performance for the processor core in many-core designs.

1. はじめに

メニーコアの設計は、極単純な小型のコアを大量に並べるものから、複雑で高性能、巨大なコアを少数並べる設計まで、幅広い選択肢がある。多くのプロセッサは、巨大なコアを少数並べることで、シングルスレッドの性能を犠牲にしないような方針で設計が行われている [3]。一部のメニーコア指向のプロセッサでは、幾分シンプルなコアを用いることで、プロセッサコア数を多く設計した製品も存在する [4]。しかし、これらの設計は、本来考えられる選択肢の幅のごく一部、比較的複雑なコア寄りの選択のみを行っている。パイプライン処理をしないプロセッサは、効率面でより複雑なプロセッサに劣るものの、実装規模を小さく収められる利点がある。また、パイプラインプロセッサでも、その設計の幅は多様であり、どの程度のハードウェア資源を犠牲にしてどのレベルの性能を実現しているかも方式により異なる。使うプロセッサの複雑さに応じて、1 つのチップに実装できるプロセッサの数が決定されるため、複雑さの異なるプロセッサを用いて性能と実装効率を勘案する必要がある。

既存のメニーコア・アーキテクチャは、Intel Single-chip

[1] 東京大学
The University of Tokyo
[a)] tomari@is.s.u-tokyo.ac.jp
[b)] hiraki@is.s.u-tokyo.ac.jp

Cloud[2] (48 コア)、Xeon Phi (50+コア)、IBM Cyclops[9] (64 コア)、Cavium Octeon II (32 コア)、Tilera Tile-GX (100 コア) など、数十から百コアを対象にしている。しかし、半導体製造技術の向上により、同等のコアを使う場合でも搭載コア数は増加する。よりシンプルなプロセッサコアを用いた場合、それを越える数のプロセッサコアが実装可能になる。増大したプロセッサコアを効率的に動作させることが可能な超メニーコアアーキテクチャとして、本研究では我々が過去に提案した、マルチレベル相互結合網の一種であるシャッフルエクスチェンジの各段に PE を配置した方式を用いた [11][7]。この方式は計算の際に直接アクセスするメモリはすべてオンチップで搭載され、入出力はネットワークの端に存在するプロセッサコアのみが行うことにより、ソフトウェアによる最適化で演算機の最大の性能が実現できるように設計されている。

超メニーコアでトータルスループットを最大化するためのプロセッサアーキテクチャの特徴を調べるため、実際に 2 つの複雑さの異なるプロセッサコアを用いた超メニーコアのシミュレータを作成した。その上で偏微分方程式を解くアプリケーションを実装し、それぞれのコアでの性能を測定することにより、性能面でどのような特性があるかを測定した。実装したエミュレータは、Nsim[10] などのネットワークシミュレータと異なり、プロセッサも含めシステム全体をエミュレーションする。この方式は、ネットワークの状況によるプロセッサの振る舞いの動的な変化を観察できる一方、シミュレーションに時間がかかる方式である。プロセッサコアに、最も小さいプロセッサの代表として 8080[1]、およびパイプライン動作を行うプロセッサとして SH-2[5] を用いた。エミュレータはコア数のスケーラビリティを考慮して設計されており、SH-2 実装で 1600 万コアまで動作を確認した。現在の超大型の計算機では、例えば「京」の場合、70 万コアで構成されている [8]。将来的にはこのような数のプロセッサコアも 1 チップで実装できるようになるはずであり、このような実験環境を用いることで、その際のハードウェアとソフトウェア問題点を事前に洗い出すことができる。

2. 手法

2.1 ターゲットマシンのアーキテクチャ

メニーコアのアーキテクチャは我々の過去の研究をベースにしている。チップないの接続は、プロセッサとネットワーク管理回路、ローカルメモリをまとめた Processing Element (PE) 単位で管理される。PE の接続は Shuffle Exchange ネットワークの 1 段を用いて行われる。PE を 2 次元に配置し、それぞれの次元を Pipe (図中縦方向), Rank (図中では横方向) と名付け、隣り合う Rank を Shuffle Exchange の 1 段を使って接続する (図 1)。外部との I/O は最小のランクと最大のランクのみが行う。想定する動作は、最小のランクが必要なデータを外部のメモリから読み、PE を通過して計算を行い、結果を最大のランクが外部のメモリに書き込むというものである。Shuffle Exchange を利用する利点は、任意のパイプに到達するためにかかる hop 数がランク数の対数に比例するので、ランク数が大きくなったときでもメッシュに比べ小さな hop 数で目的地に到達できること、およびルーティングが目的地のパイプ番号の各ビットで決定できるため、ビット操作のみで決定できる点である。

ネットワークは、Reflective Memory 方式 [6] で、アドレス空間にマップされたメモリ領域として PE のプロセッサから操作する。それぞれの PE では、アドレス空間上にネットワークがマップされている。ランク n のメモリ空間には、ランク $n-1$ の接続されている PE のアドレス空間にマップされているメモリが存在する (図 2)。同様に、ランク n のアドレス空間には、ランク $n+1$ で接続されている PE のメモリ空間がマップされていて、これらのメモリ空間を使うことで高速な片側通信を実現している。それぞれの PE 間のパスについて、別のアドレスをマップしているため、通信は常に 1 PE 対 1 PE であり、複雑なハードウェアおよびソフトウェアを必要としない。

今回使ったプロセッサは 2 種類で、8080 と SH2 である。8080 は 8 ビット世代のプロセッサで、命令長は 1 バイトから 3 バイトまでの可変、乗算器は不要という特徴があり、小型の実装が可能である。1 命令を実行するのに演算命令で最小 4 クロックかかり、最大の命令で 16 クロックかかる。SH-2 は 32 ビットのプロセッサで、全命令が 16 ビット幅であり、分岐や乗算などをのぞく多くの命令で 1 命令/サイクルのスループットを持つパイプライン実行を行うため、8080 に比べ命令実行は高速である。今回エミュレーションで使用するメニーコアアーキテクチャでは、メモリはすべてオンチップで PE 毎にローカルに接続されており、命令ストリームはこのローカルメモリから読まれる。8080 では多くの算術命令が 1 バイトでエンコードできていたのに比べ、全命令が 16 ビットとなる SH-2 ではコードのサイズが大きくなり、ローカルメモリで演算結果の一時保存に利用できる領域が圧迫される。その他の命令セットでは、拡張のない MIPS と ARM, PowerPC では命令長が 32 ビットであり、SH よりさらに多い命令領域が必要になると考えられる。また、68000 では最小命令長が 16 ビットで、命令およびアドレッシングモードによって最長 80 ビットまで命令幅がのびるため、SH と同レベルの命令領域が必要になるうえ、デコードの処理が複雑になる。8080 と SH-2 は、両者ともに開発用ツールが揃っており、開発用のプログラムの作成が容易に行える。さらに、SH-2 は GNU のツール群を利用することが可能で、今回のテストプログラムのアセンブル、ROM イメージの作成はすべて GNU Binutils を用いて行った。現在はすべてアセンブリ

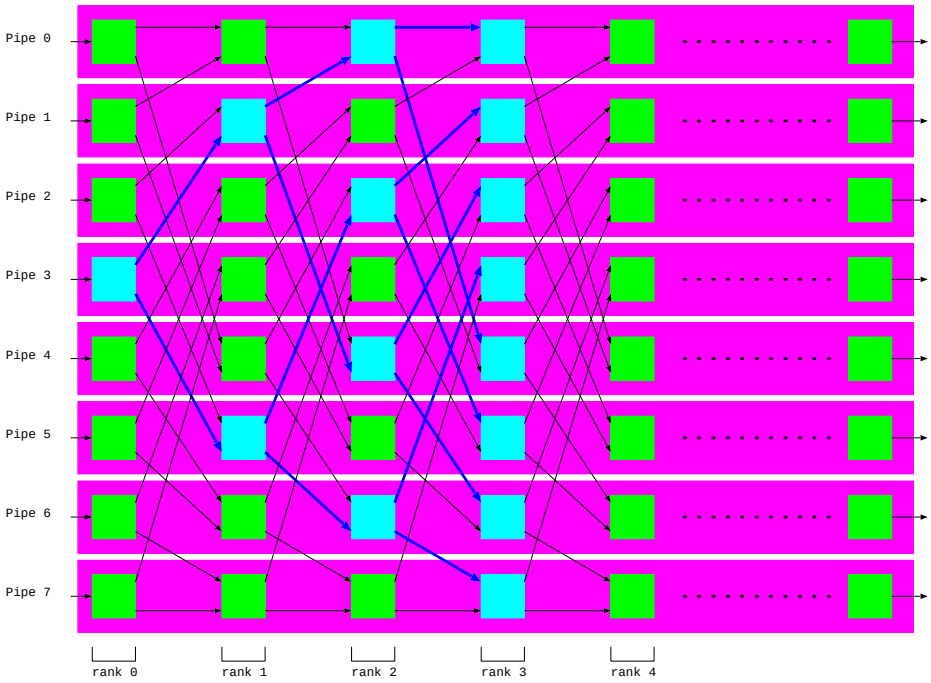

図 1 PE ネットワークの接続様式

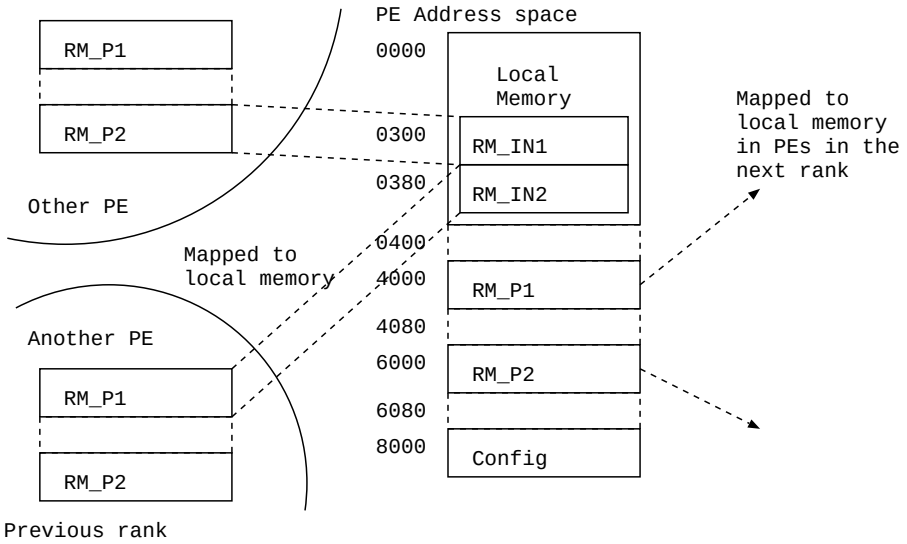

図 2 PE から見たネットワーク

でプログラム作成を行っているが、適切なリンカスクリプトを作成することで C 言語での開発も行えるようにする予定である。

命令の容量が小さい命令セットを選択したのは、ローカルメモリの容量を小さく保つためである。超メニーコアを実現するためには、単純なプロセッサを使うことでコアの面積を小さくするだけでなく、レジスタファイルを含めた周辺のメモリも小さくする必要がある。今回のプログラムは 8080 版でローカルメモリ 1 KiB、SH-2 版でローカルメモリ 2 KiB で動作する。ローカルメモリには、プロセッサが読み込む命令ストリーム、スタック、ヒープ、およびネットワークにマップされるメモリ領域が含まれる。ネットワークにマップされるメモリ領域は、8080 版で 1 入力ポートあたり 128 bytes で、本実装で用いた Shuffle Exchange では入力ポートは 2 ポートのため、1 KiB 中 256 bytes の領域がネットワークに使われる。SH-2 版はそれぞれの容量がすべて 2 倍になっている。

2.2 エミュレーション

プロセッサコアのエミュレーションは、アーケードゲームエミュレータ MAME のコードをベースにしている。MAME はタイミングが重要になるアプリケーションのエミュレータとして設計されていて、CPU エミュレーションコードでは各命令のクロック数をカウントし、要求されたクロック分の処理を実行するモデルが使われている。このモデルはメニーコアの高速エミュレーションで有効で、全プロセッサで特定のクロック数分の処理を行い、次のサイクルに移るという方式のエミュレーションを行った。メモリアクセスのサイクル数は、今回のアーキテクチャが全メモリがローカルに配置され、ネットワークもシャッフルエクスチェンジ上で隣接する PE との片側通信に限られるため、ウェイト 0 でエミュレーションを行っている。

プロセッサコア以外の部分について、MAME は数プロセッサまでの小規模なシステムをターゲットにした設計のため、利用可能なメモリ領域の大きさとメモリ使用効率、マルチスレッドの対応で問題がある。我々は、CPU 以外の部分について独自のエミュレータを用いた。エミュレータが動作するプロセッサをホストプロセッサ、その上で動作する制御の流れをホストスレッドと呼び、エミュレーションする 8080 と SH-2 と、それらの制御の流れをターゲットプロセッサ、ターゲットスレッドと呼ぶ。完全に正確なエミュレーションを行う場合、ターゲットプロセッサが 1 サイクル進む毎にホストスレッドを同期し、そのサイクルでのターゲットシステムでのネットワークへのメモリライトが完了したことを確実にしてからメモリイメージを新しいメモリイメージに張り替え、次のサイクルを処理することになる。正確さは保証されるものの、メモリ領域が 2 倍必要になるのと、ホストスレッドの同期処理に時間がかかるため、エミュレーションが大幅に遅くなる。エミュレーションの正確さをできる限り保ちつつこれらの問題を軽減するため、実装したエミュレータでは、ターゲットプロセッサアレイをパイプ単位で分割し、各ホストスレッドが複数のパイプのエミュレーションを行う。ホストスレッド毎に、各パイプに含まれる PE をランクが大きい方から順に SH-2 でターゲットの 7 クロック、8080 で 25 クロックずつ実行し、担当する全パイプで処理が終わった段階でほかのホストスレッドと同期する。PE のランクが大きい方から実行を行うのは、想定するデータの流れがランクが小さい方から大きい方への流れなので、一回に実行されるクロック数の中でメッセージが次ランクに伝播する場合でも、実際には起きない 2 つ以上先のランクへの情報の伝播を防ぐためである。同期と同期の間のターゲットのクロック数は、平均的な 1 基本ブロックを実行するのにかかるクロック数を概算して決定している。

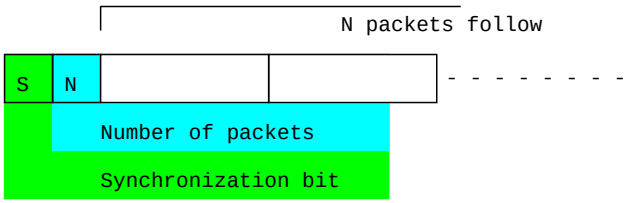

図 3 通信バッファの構造

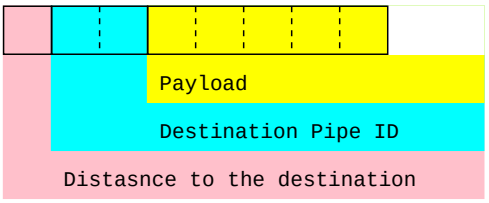

図 4 偏微分方程式でのパケットの構造

超メニーコアでは、各プロセッサに接続されるレジスタファイルやローカルメモリの総容量が巨大になる。8080 の場合、ローカルメモリが 1 KiB で、プロセッサのステートなどをあわせた結果、3200 万コアのエミュレーションで 36 GiB のホストメモリを消費する。同様に、SH-2 ではローカルメモリが 2 KiB で、SH-2 は内部状態のレジスタ幅が広いのと、プロセッサ内部に 512 バイトのメモリが必要なため、1600 万コアのエミュレーションに 50 GiB のメモリが必要である。これらのメモリ領域のそれぞれは、ホスト OS のページサイズよりも小さいレジスタファイルやプロセッサステート、ローカルメモリであるため、PE 毎に領域を確保すると容量効率が悪化する。本実装では、このような小さいメモリ領域をエミュレータで効率的に管理している。

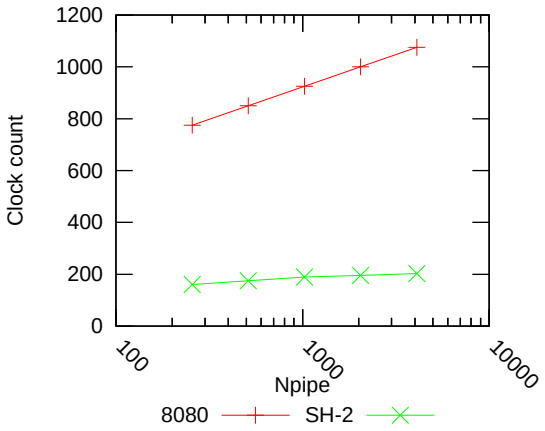

図 5 同期操作にかかるクロック数

3. 評価

3.1 同期レイテンシ

通信機構がメモリをマップした構造になっているので、同期はソフトウェアで行う必要がある。同期は、通信バッファ先頭の 1 ワード (8080 実装では 1 バイト、SH-2 実装では 1 ワード) を用いて、同期ビットで実現している (図 3)。通信バッファは、1 つの PE-PE パスの端にあるバッファ全体をさす。このビットが 0 のとき、通信バッファの所有権は送り側 (ランクが小さい方) にあり、このビットが 1 のとき通信バッファの所有権は受け側 (実際に通信バッファが存在する PE) にある。所有権を変更できるのは通信バッファを所有するノードのみとすることで、任意の PE からみたときの同期ビットの変化は単調となり、この方式で 2 つの PE の同期は成立する。

この同期方式を利用し、全パイプを同期するには、同期を開始するランクで両方の出力ポートで同期ビットをセットする。同期が完了する $\log_2 N_{pipe}$ 先のランクの間では、両方の入力ポートに同期ビットが揃ったら、両方の出力ポートで同期ビットをセットする。最後のランクで、両方の入力ポートに同期ビットが揃った段階で同期が完了する。なお、図 3 にあるパケット数のフィールドはこのプログラムでは使わない。

図 5 は、この同期操作をパイプ数をかえて行ったときの、8080 と SH-2 で同期が完了するまでのクロック数である。Shuffle Exchange を使っているため、全パイプに同期のトークンが行き渡る時間は $\log_2 N_{pipe}$ に比例している。SH-2 では 8080 に比べ、1/5 前後のクロック数で同期が完了する。8080 はバスアクセスの度に 5 クロック以上かかる上、アドレスの計算など基本的な部分で時間がかかるためである。

3.2 偏微分方程式

実用に近いプログラムとして、2 つのプロセッサコアを使ったそれぞれの場合について、1 次元の偏微分方程式を解くプログラムで性能を測定した。解く偏微分方程式は

表 1 8080 版のハードウェア実装でのスライス数

N_{rank}	$N_{pipe}=4$	8	16	32
2	3,269	6,010	12,059	22,824
4	5,594	12,314	23,897	–
8	12,578	24,146	–	–
16	24,270	–	–	–

$$\frac{\delta}{\delta t} r(n,t) = \frac{\delta^2}{\delta n^2} r(n,t) \qquad (1)$$

として表され、$r(n,t) = (x,y,z,w)$ となる 4 要素のベクトルについての計算である。各タイムステップではそれぞれの要素について、

$$x_{t+1}(n) = (x_t(n-1) - 2x_t(n) + x_t(n+1))/4 \qquad (2)$$

という計算を行う必要がある。今回のターゲットアーキテクチャでは、任意のパイプにデータを送信するためには、$\log_2 N_{pipe}$ ランク離れた PE にルーティングをすることになる。偏微分方程式のプログラムでは、計算の結果を隣接するパイプと計算が行われたパイプに送信する必要があるため、各タイムステップを $\log_2 N_{pipe}$ ランク毎に割り当てる。PE のプロセッサで行われる処理の流れは Algorithm 1 にある通りで、各プロセッサは到着したパケットが自分宛なら計算のためデータを保存し、そうでなければルーティングして出力ポートにコピーしている。計算が終わると、結果を自分より一個若いパイプと、自分と同じランク、自分より一個上のランクに計算結果を送信する。このプログラムでプロセッサ間を流れるパケットの構造は、まずパケットの到達点までの hop 数があり、次に宛先のパイプ ID がり、最後に偏微分方程式で使うデータがある (図 1)。偏微分方程式で使うデータには、パケットがどの方向から来たものなのかを識別するための値と、実際の計算で用いる数値データが格納されている。

最初のランクが単純にデータを流すだけだと、実際に計算を行うランクは $\log_2 N_{pipe}$ ランクに 1 ランクとなるが、ルーティングを開始するランクをずらすことで、全ランクで計算とルーティングを行うことができる。この場合でも、プログラムは先頭の目的地までの距離を使い、自分が計算を行うべきか、ルーティングするべきかで分岐するため、同一のプログラムを用いることができる。

8080 と SH2 のそれぞれについて、偏微分方程式を解いたときにかかるクロックの内訳を測定した (図 6)。計算を行うランクと、ルーティングのみのランクで分けている。Routing は、入力されたパケットを仕分けし、自分宛の場合にローカルメモリの領域にデータをコピーし、そうでない場合適切な送信ポートを選択しパケットを送信する時間が含まれている。Calc は偏微分方程式の 1 ステップを計算する時間、Send は計算の結果を自分と同じパイプと前後

Algorithm 1 PE で動作するプログラムの概要

```
loop
  wait(output port 0 sync bit=0)
  output port 0 'number of packets'← 0
  wait(output port 1 sync bit=0)
  output port 1 'number of packets'← 0
  for p ←input port 0 and input port 1 do
    wait(p sync bit=1)
    for n = 0 to p 'number of packets' do
      q ← pointer to the head of nth packet
      if distance to destination in q > 0 then
        route this packet to output port
      else
        copy payload to static region
      end if
    end for
    done(input port p, sync bit←0)
  end for
  do calculation
  output port 0, sync bit←1
  output port 1, sync bit←1
end loop
```

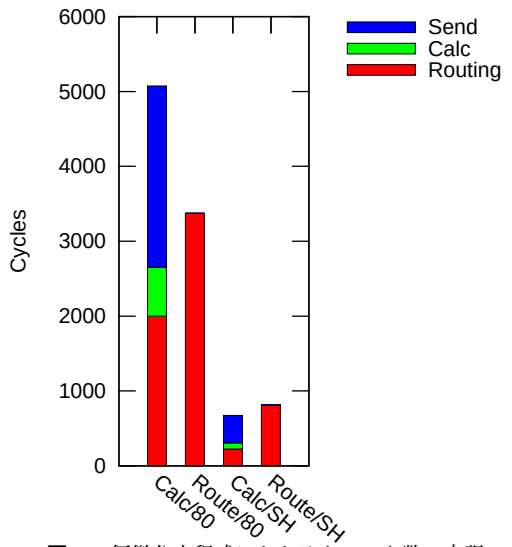

図 6 偏微分方程式にかかるクロック数の内訳

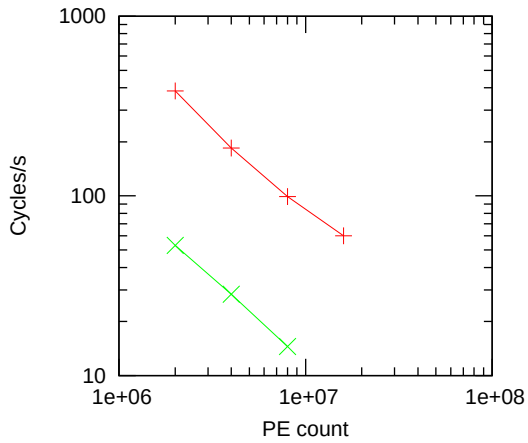

図 7 エミュレータのスループット

のパイプに伝播させるためにパケットを生成し、適切な出力ポートを選択してパケットを出力する時間が含まれる。SH-2 が 1/5 以下のクロック数で処理が完了していることがわかる。ルーティングのみをおこなうランクの実行時間が、8080 では計算ノードより短いのに対し、SH-2 では計算ノードより長くなっている。これは、8080 ではバスが 8 ビットだったのに対し、SH-2 の実装ではバスが 32 ビットになっており、4 バイト境界をまたぐメモリアクセスに時間がかかるようになったからと、SH-2 版では 1 ワード 16 ビットの SH-2 のアーキテクチャにあわせてパケットサイズが大きくなっているからである。SH-2 では 4 バイト境界をまたぐアクセスができないため、ソフトウェア側でコピーを分割している。ルーティングではデータのコピーが処理時間の大半を占めるため、性能はここで決まっている。なお、8080 と SH はともにシフト命令は 1 ビットのみなので、ループを使っている。

なお、この偏微分方程式のプログラムのオブジェクトは、8080 で 468 バイト、SH-2 で 624 バイトとなり、SH-2 は 33%長くなった。8080 と同様の 1 KiB のローカルメモリだと、これ以上複雑な動作をさせることは SH-2 では困難である。メニーコアの実現にあたっては、命令ストリームにデータ圧縮を施すなどして、命令領域のメモリ使用量を少なく抑えるための工夫が必要になる。

3.3 エミュレータの性能

エミュレータの性能を、6 コア Intel Westmere (2.93 GHz) を用いて測定した。エミュレータは 12 スレッド利用する。エミュレータの律速要因はホストスレッドの同期であり、8080 は 25 ターゲットサイクルでホストスレッドを同期しているのに対し、SH-2 は命令あたりのサイクル数が小さく、これに対応して同様の精度でエミュレーションを行うために 7 ターゲットサイクルで同期を行っている。これにより、SH-2 のエミュレータの性能は毎秒実行するターゲットプロセッサのサイクル数が 8080 の 1/7 である (図 7)。SH-2 は命令のエンコーディングは単純なものの、命令数が多いため、プロセッサコアのエミュレーションのプログラムが長い。8080 では 1 段階で命令を決定しているのに対し、SH-2 では命令を実行するまで 2 段階で実行する命令をデコードする構造のエミュレータになっていることが性能悪化の他の原因と考えられる。1 プロセッサのシステムをエミュレーションしたとして換算すると、8080 で 805 MHz、SH-2 で 112 MHz 相当の性能である。

3.4 ハードウェア資源効率

今回エミュレーションを行ったメニーコアアーキテクチャの 8080 での実装を FPGA で行った。実装を行う際に用いた 8080 コアは、エミュレータと同じく、オリジナルの 8080 と同じ命令レイテンシを持つ。パイプ数とランク

数を変えて、最大 64 プロセッサまでの資源の見積もりを行った。ターゲットの FPGA は Xilinx Virtex-6 シリーズの XC6VLX240T-1FF1156 である。配線まで終わった段階での使用スライス数を表 1 に示す。

パイプ数が増えると、Shuffle Exchange 方式の規模が増加し、ランク数が増加すると、Shuffle Exchange の段数が増加する。従来の PE 毎に矩形領域を人手で割り当てていく場合だと、Shuffle Exchange の規模が大きくなると配置の自由度が増し、最適な配置を探索するのに多大な労力が必要であるが、今回は制約として I/O ピンのみを与え、配置はツールによる大域最適化が働いている。配置配線をすべてソフトウェアで行った場合、Shuffle Exchange の規模を大きくしてもリソースの大幅な増加につながらず、Shuffle Exchange はメニーコアチップでのネットワークの方式として有用であることがわかる。

4. まとめ

超メニーコアプロセッサで計算コアに求められる特性は、従来のコアとは異なり、大量のコアを並べた際のトータルのスループットの最大化である。これを実現するために、マルチサイクルのプロセッサから、資源を投じて高度に命令レベル並列性を利用するプロセッサコアまでの間でどういったものを用いるのが適しているかを調べるため、最も単純化されたプロセッサと、パイプライン実行するプロセッサを使い、実際にシミュレータを作成して評価を行った。実アプリケーションカーネルに近いベンチマークで、SH-2 は 8080 の 5 倍の処理性能がある一方、命令長が長くなるため、ローカルメモリはより多く必要になり、プロセッサの資源も大きくなると予想される。8080 版のプロセッサアレイについては、合成可能なハードウェア実装が完了し、最大 64 コアまでの資源の見積もりが可能になっている。SH-2 については、プロセッサの実装をしていないため、8080 との資源の比較は行えていない。SH-2 が 8080 の 5 倍以下の資源で実装できる場合、SH-2 を使うのが好ましいといえる。FPGA では、パイプラインを用いた場合でも資源の増大は小さく、MIPS コアの場合でも 8080 コアの 2 倍の資源で実装可能なことから、この場合はパイプラインプロセッサを用いた方がよいと予想される。極端にシンプルなコアを並べても、スループットは資源に見合わないと考えられる。

メニーコアプロセッサを設計する場合、シンプルなコアを並べると、複雑なコアを少数並べた従来のコアよりスループットが下がることがないかを確認し、従来考えられてきたメニーコアに最適なコアを再定義する必要がある。そのうえで、スループットを最大化できる超メニーコアアーキテクチャと組み合わせ、数値計算以外でも高速な処理が可能な設計が求められている。

参考文献

[1] Intel Corporation. intel 8080 microcomputer systems user's manual. September 1975.
[2] Jim Held. Single-chip cloud computer — an experimental many-core processor from Intel Labs. *Intel Labs Single-chip Cloud Computer Symposium*, 2010.
[3] R. Kalla, B. Sinharoy, W.J. Starke, and M. Floyd. Power7: Ibm's next-generation server processor. *Micro, IEEE*, 30(2):7 –15, march-april 2010.
[4] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: a 32-way multithreaded sparc processor. *Micro, IEEE*, 25(2):21 – 29, march-april 2005.
[5] Hitachi America Ltd. Superh risc engine sh-1/sh-2 programming manual. September 1996.
[6] S. Lucci, I. Gertner, A. Gupta, and U. Hegde. Reflective-memory multiprocessor. In *System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on*, volume 1, pages 85 –94 vol.1, jan 1995.
[7] Hisanobu Tomari. Design and evaluation of sea-of-core array architecture with 32 million processor cores. Masther Thesis, Dept. of Computer Science, the University of Tokyo, Mar. 2012.
[8] M. Yokokawa, F. Shoji, A. Uno, M. Kurokawa, and T. Watanabe. The k computer: Japanese next-generation supercomputer development project. In *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, pages 371 –372, aug. 2011.
[9] Ying Ping Zhang, Taikyeong Jeong, Fei Chen, Haiping Wu, R. Nitzsche, and G.R. Gao. A study of the on-chip interconnection network for the ibm cyclops64 multi-core architecture. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 10 pp., april 2006.
[10] 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. PSI-NSIM : 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.
[11] 泊 久信 and 平木 敬. 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. 研究報告 計算機アーキテクチャ（ARC） 2010-ARC-190(3), jul 2010.