Vol.2019-HPC-168 No.5
2019/3/5

理研ポスト京プロセッサシミュレータの評価

児玉 祐悦[1,a)]　小田嶋 哲哉[1]　安里 彰[2]　佐藤 三久[1]

概要：
ポスト「京」向けのアプリケーションの早期開発を目的に、理研ではポスト京プロセッサシミュレータを開発している。本シミュレータは汎用プロセッサシミュレータ gem5 をベースとしたもので、ポスト京プロセッサを忠実にシミュレーションするものではない。しかし、詳細なパラメータチューニングと機能拡張を行い、アウトオブオーダ実行の命令パイプラインをシミュレーションすることで、十分な精度の結果が得られると考えている。本シミュレータでは、アプリケーションチューニングが可能となる相対的な評価が行えるような精度で、ポスト京プロセッサにおける 1 ノードのアプリケーションの実行速度を見積もることを目指している。本稿では、本シミュレータの実装の詳細を示すとともに、ポスト京のテストチップの実行サイクル数と比較してその精度を検証する。

1. はじめに

理化学研究所では、日本における次世代のフラグシップ計算機としてポスト「京」を開発している。ポスト「京」は 2021 年春の一般共用を目指して開発を進めているが、実機の一部をユーザが先行的に利用できるアーリーアクセスまでもまだ 1 年以上ある。できるだけ早期にポスト「京」でのアプリケーション開発を行う事を目的に、理研では理研ポスト「京」プロセッサシミュレータ（以下、理研シミュレータ）を開発してきた。

理研シミュレータは汎用プロセッサシミュレータ gem5 [1] をベースとしている。gem5 では、基本アーキテクチャモデルに基づくアウトオブオーダ実行をシミュレーションすることができ、詳細な実行サイクル数を見積もることができる。この基本アーキテクチャモデルは、ポスト「京」のプロセッサである富士通 A64FX プロセッサ [2], [3] とは細部では異なっている。しかし、理研シミュレータでは、パイプラインレベルの詳細なパラメータチューニングを行うとともに、主にメモリ階層における機能拡張によって、十分に正確なシミュレーションが行えると考えている。本シミュレータでは、アプリケーションチューニングが可能となる相対的な評価が行えるような精度で、ポスト京プロセッサにおける 1 ノードのアプリケーションの実行時間を見積もることを目指している。

本稿では、まず、ポスト「京」のプロセッサである A64FX の概要について述べるとともに、ベースとしている汎用プロセッサシミュレータ gem5 について説明する。次に、理研シミュレータの実装の詳細を示すとともに、本シミュレータの実行時間を、A64FX のテストチップの実行時間と比較し、その精度を検証し、最後にまとめを述べる。

2. ポスト「京」プロセッサ A64FX

図 1 にポスト「京」プロセッサ A64FX の構成を示す。A64FX は、13 個のコア（うち 1 コアはアシスタントコアと呼ぶ OS サポート用のコア）と L2 キャッシュおよびメモリコントローラから構成される CMG (Core Memory Group)4 基がリングバスで接続されている。L2 キャッシュは CMG あたり 8MiB の容量で、900GB/s を超えるスループットを持ち、CMG 間のコヒーレンスをサポートしている。メモリは CMG あたり 8GiB の容量の HBM2 で同一パッケージ内に搭載されており、CMG あたり 256GB/s のスループットを持つ。

各コアは、Armv8.2-A の 64 ビットアーキテクチャで、SVE(Scalable Vector Extension) [4] と呼ぶ新しい SIMD 拡張機能をサポートしている。各コアは 512 ビットの SIMD パイプラインを 2 つ持ち、プロセッサ全体の演算性能は 2.7TFLOPS を超えている。SVE は倍精度、単精度の浮

[1] 理化学研究所 計算科学研究センター
[2] 富士通（株）
[a)] yuetsu.kodama@riken.jp

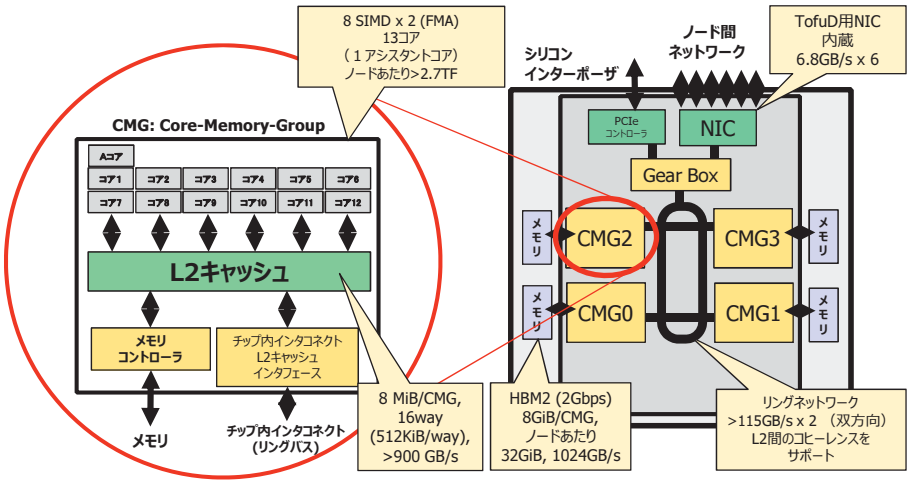

図 1　ポスト「京」プロセッサ A64FX の構成

動小数点数の他に半精度をサポートするとともに、64bit, 32bit, 16bit, 8bit の各整数もサポートしている。各コアは 64KiB のデータおよび命令 L1 キャッシュを持つ。データキャッシュのロード性能は 230GB/s を超え、ストア性能は 115GB/s を超える。

A64FX は、京と同じ 6 次元トーラスネットワーク (TofuD) をサポートするネットワークインターフェースをプロセッサ内に搭載している。各リンクスピードは 6.8GB/s に拡張されており、転送エンジンを 6 個に増強している。

3. プロセッサシミュレータ gem5

理研シミュレータは、オープンソースのプロセッサシミュレータ gem5 をベースとしている。gem5 の主な特徴は以下の通りである。詳細については、http://gem5.org を参照のこと。

- 複数の命令セットアーキテクチャ (ISA) に対応している。例えば、Alpha, Arm, SPARC, x86 など。理研シミュレータでは Armv8-A アーキテクチャのみを対象としている。
- 2 つのシステムモードに対応している。OS などもシミュレータ上のプロセッサでシミュレートする full system(FS) モードと、システムコールはソフトウェアでエミュレートする system-call emulation(SE) モードがある。理研シミュレータでは SE モードのみをサポートしている。
- 複数の CPU モデルに対応している。1 サイクル 1 命令実行の CPU モデル (Atomic)、Atomic にメモリ参照のタイミングを追加した CPU モデル (Timing)、インオーダのパイプラインをシミュレーションする CPU モデル (InOrder)、アウトオブオーダのパイプラインをシミュレートする CPU モデル (O3)。理研シミュレータでは Atomic および O3 のみをサポートしている。
- 2 つのメモリシステムを持つ。1 つはシンプルな Classic モデルで、もう一つはコヒーレントメモリシステムを柔軟に構成できる Ruby モデルである。理研シミュレータでは、現状は Classic モデルを用いている。

4. 理研シミュレータ

gem5 は Armv8-A ISA に対応しているが、理研シミュレータの開発を始めた 2016 年当時は、新しい SIMD 拡張である SVE(Scalable Vector Extension) には、Atomic モードのみの対応予定で、O3 には対応の予定もなかった。そのため、理研で独自に O3 モードの開発を行うこととした。具体的には、gem5 の Armv8-A ISA については、Arm Research がメンテナンスしており、SVE 向けの Atomic モードの開発は行っているということで、その実装を提供してもらい、理研で独自に O3 モードに対応することとした。

しかし、その後、Arm Research でも SVE 向けの O3 モードへの対応を行っていると連絡を受けた。両者の実装について比較を行い、実装方法に大きな違いが無いこと、Arm 版は今後 gem5 本体に統合していく予定であることなどから、独自の実装をやめて、Arm 版の SVE O3 実装に移行することとした。2018 年 4 月より移行を開始し、10 月に移行を完了した。

SVE を含む Armv8-A ISA に関しては、Arm 版 gem5 として実装されているが、理研シミュレータの目的であるポスト「京」でのプログラム実行時間を高精度に見積もるためには、単に ISA として実行できるだけでは機能として不足している。

gem5 の O3 モードでは、パイプラインの各ステージのレイテンシやスループット、アウトオブオーダ実行の各リソースサイズなど詳細なパラメータを柔軟に指定することが可能である。これらのパラメータ値については、ポスト「京」の開発企業である富士通から、そのプロセッサである A64FX の詳細な値をもらい、それらのパラメータを調整

することにより、プロセッサ内の実行については十分な精度での性能シミュレーションができるようになった。ただし、gem5 の CPU モデルは Alpha 21264 を基本としたアウトオブオーダパイプラインに基づいており、A64FX のパイプライン構成とは基本的には異なっているため、当然違いは生じる。主な相違点は以下の通りである。

- リザベーションステーションが gem5 では 1 つであるのに対し、A64FX では、メモリアドレス計算用に 1 つ、演算用に 2 つ、分岐用に 1 つの計 4 つに分かれている。理研シミュレータでは 4 つの合計値に近い値で調整を行っているため、どれかに偏って処理ユニットが使われるなどした場合、両者の違いが大きくなる可能性がある。
- gem5 ではメモリアドレス計算が、メモリ命令ユニット内で行われるため、ストア命令ではメモリアドレスオペランドと書き込みデータオペランドが両方揃ってからストア命令が実行される。それに対し、A64FX では、メモリアドレスは独立したユニットで計算可能である。理研シミュレータでは、gem5 のままの実装としているため、アドレス計算のタイミングに違いが生じる可能性がある。
- gem5 では 1 つの命令は 1 つの処理ユニットで実行されることを前提としているのに対し、A64FX では一部の命令は複数の実行ユニットを並列あるいは遂次に利用する場合がある。理研シミュレータでは、そのような命令の頻度は少ないものとして、 1 つの実行ユニットに割り当てている。また、gather load 命令など一部の命令は gem5 のマイクロ命令機構を用いて、複数のマイクロ命令に分割して実行している。このような実行は、アウトオブオーダリソースを A64FX よりも多く使用することになり、実行時間の違いとして現れる場合がある。

理研シミュレータでは、現在 1CMG のシミュレーションのみをサポートしており、12 コアまでのマルチスレッド実行が可能である。プログラムの実行時間を高精度に見積もるためには、命令の実行時間だけではなく、キャッシュメモリ階層のアクセス時間を正確にシミュレーションすることが必要である。そのため、キャッシュメモリ階層の性能を A64FX に合せるように、理研シミュレータでは、gem5 の拡張を行った。主な拡張は以下の通りである。

- L1、L2 のキャッシュの容量や連想度、ラインサイズ、レイテンシについては、gem5 のパラメータ設定で実際の A64FX に合せて設定を行った。
- gem5 では、L1 に対して同一サイクルで load と store が同時にアクセスできることが前提となっている。それに対して、A64FX では 64byte x 2 の load か、64byte の store が可能となっている。理研シミュレータではこれに合わせた制御を可能にした。
- gem5 では、L2 から L1 へのキャッシュフィルが、コアの L1 へのアクセスと独立に制御されている。そのため、L1 の性能が高めに出てしまう。これらの間の排他制御を行うようにして、A64FX の L1 性能に近づけた。
- gem5 では、L1 へのアクセスがキャッシュラインを跨いでいるときには、複数のアクセスに分割しているためオーバヘッドが生じる。A64FX では、キャッシュラインを跨ぐアクセスでも性能低下を起こさないような設計となっている。理研シミュレータでもそのような機能を追加した。
- gem5 では、ソフトウェアプリフェッチがサポートされているが、リードアクセス型のプリフェッチしか実装されていなかった。ストアアクセス型のプリフェッチもメモリアクセスの最適化には重要であり、理研シミュレータで追加を行った。本機能については、すでに gem5 へパッチの報告を行い、受理されている。
- gem5 にもハードウェアプリフェッチの機能はあるが、単純なプリフェッチしかサポートされていない。理研シミュレータでは、京互換のハードウェアプリフェッチ機能を追加した。
- gem5 では、複数のコアで共有する L2 を簡単に指定できるが、それで生成される L2 はシングルバンクとなっているため、コア数が多くなると L2 アクセスがボトルネックとなりやすい。gem5 でも、L2 がモジュール構成になっているため、ユーザが個別に記述を追加することによりバンク化することは可能である。理研シミュレータでは、パラメータの指定のみでバンク数を簡単に変えられるように拡張を行った。
- gem5 では、L1/L2 間のバス幅は、1 つのパラメータとなっており、L1 から L2 への転送性能と L2 から L1 への転送性能が同一であることが前提となっている。一方、A64FX では、この 2 つの転送性能は異なる。理研シミュレータでは、この 2 つのバス幅を異なるパラメータとして指定できるように拡張を行った。L2/メモリ間のバス幅についても同様である。
- gem5 では HBM1 がサポートされているが、HBM2 はまだサポートされていない。理研シミュレータでは、HBM1 のパラメータをベースに HBM2 のパラメータを追加するとともに、バンクアクセススケジューリングポリシーを適切に設定した。gem5 の標準の機能では、メモリインターリーブ方式を A64FX と完全に合わせることはできなかったが、バースト長など他のパラメータとの組み合わせにより、方式は異なるが性能的には近づけるような調整を行った。

この他、以下のような、使い勝手を向上させるための機能拡張も行った。

- gem5 ではシミュレーションの各種統計情報を stats.txt

というファイルに出力するが、これにはプログラムの開始から終了までの情報が収集される。しかし、特定区間の統計情報を取得することは有用であるため、そのような機能を追加した。ただし、stats.txt を切り分けるためにはシミュレータ起動時に切り替え時刻を指定する必要があるため、まずは切り替えタイミングの時刻を取得するためにプレ実行を行い、次にその時刻を指定して本実行を行う方式として、そのような実行を行う python script を作成した。

- 富士通のスーパーコンピュータシステムでは PA データと呼ぶ詳細プロファイル情報が取得できる。gem5 でも同様の情報が取得できるように、機能拡張を行った。特に、0 命令コミット時の要因を、メモリ待ち、演算待ちなどに分類する機能などを追加した。さらに、stats.txt から富士通の PA データに準じる情報を抽出する python script を作成した。
- アウトオブオーダリソースのサイクルごとの利用率を取得する機能を拡張した。
- SIMD 命令などで要素単位の演算数をカウントする機能を追加した。さらに、predicate レジスタの値に応じて有効な演算数だけをカウントする機能を追加した。
- gem5 では OpClass と呼ぶ命令グループごとに、どの実行ユニットを使うか、命令レイテンシはいくらかなどを指定するようになっている。通常は命令の OpCode ごとに OpClass が決まっているが、A64FX にはオペランドタイプ（倍精度か単精度か）によって命令レイテンシが違う命令もある。そのような命令に対応するように拡張を行った。

これらの拡張機能については、富士通からの NDA 情報である各種詳細パラメータと分離して、公開していくことを検討している。

5. 評価

理研シミュレータが、A64FX の実行時間とどれだけあっているのかを調べるために、いくつかのプログラムを用いて評価を行った。評価対象は、A64FX の試作版テストチップであり、最終的なポスト「京」の性能を示しているわけではない。また、実行するプログラムを生成するためのコンパイラは富士通からポスト「京」向けのコンパイラのプロトタイプ版を提供してもらい、テストチップ評価で用いたものとほぼ同じコンパイラである 2018 年 10 月版を用いた。ほぼ、というのはテストチップ評価向けにはランタイムルーチンに若干の実機調整用の修正が加わっているという程度で、ユーザプログラムの実行命令列については同じである。

5.1 1 コアによるカーネル実行の評価

まず、1 コアの実行の比較という事で、各種カーネルプログラムの実行時間を比較した。

```
subroutine calc01_add_r8(n, iter, dist, y, x1, x2)
 real*8 y(n), x1(n), x2(n)
 integer n, iter, i, j, dist

 do j = 1, iter
   do i = 1, n
     y(i) = x1(i) + x2(i)
   end do
 enddo

end subroutine calc01_add_r8
```

図 2 カーネルプログラムの例（倍精度加算）

表 1 カーネルプログラムリスト

名前	内容	サイズ	カーネル実行文
基本演算			
add	加算	2048	y(i) = x1(i) + x2(i)
sub	減算	2048	y(i) = x1(i) - x2(i)
mul	乗算	2048	y(i) = x1(i) * x2(i)
fma	積和	3072	y(i) = y(i) + c0 * x1(i)
div	割算	2048	y(i) = x1(i) / x2(i)
rev	逆数	3072	y(i) = 1 / x1(i)
sqrt	平方根	3072	y(i) = sqrt(x1(i))
型変換			
f2d	単精度からの変換	4096	y_r8(i) = dble(x1_r4(i))
i2d	整数からの変換	4096	y_r8(i) = dble(x1_i4(i))
d2f	単精度への変換	4096	y_r4(i) = real(x1_r8(i))
d2i	整数への変換	4096	y_i4(i) = int(x1_r8(i))
aint	aint 変換	3072	y_r8(i) = aint(x1_r8(i))
nint	nint 変換	4096	y_i4(i) = nint(x1_r8(i))
anint	anint 変換	3072	y_r8(i) = anint(x1_r8(i))
数値関数			
abs	絶対値	3072	y(i) = abs(x1(i))
max	最大値	2048	y(i) = max(x1(i), x2(i))
min	最小値	2048	y(i) = min(x1(i), x2(i))
mod	剰余	2048	y(i) = mod(x1(i), x2(i))
sign	符号	2048	y(i) = sign(x1(i), x2(i))
数学関数			
atan	atan 関数	3072	y(i) = atan(x1(i))
atan2	atan2 関数	2048	y(i) = atan2(x1(i), x2(i))
cos	cos 関数	3072	y(i) = cos(x1(i))
sin	sin 関数	3072	y(i) = sin(x1(i))
exp	exp 関数	3072	y(i) = exp(x1(i))
exp10	exp10 関数	3072	y(i) = exp10(x1(i))
log	log 関数	3072	y(i) = log(x1(i))
log10	log10 関数	3072	y(i) = log10(x1(i))
pwr	べき乗	2048	y(i) = x1(i) ** x2(i)

評価に用いたカーネルプログラムの例を図 2 に、評価に用いたカーネルプログラムのリストを表 1 に示す。カーネルとしては基本演算、型変換、数値関数、数学関数の 4 種類である。基本演算は、倍精度演算の加算、減算、乗算、積和、割算、逆数、平方数の計 7 個。型変換は、倍精度か

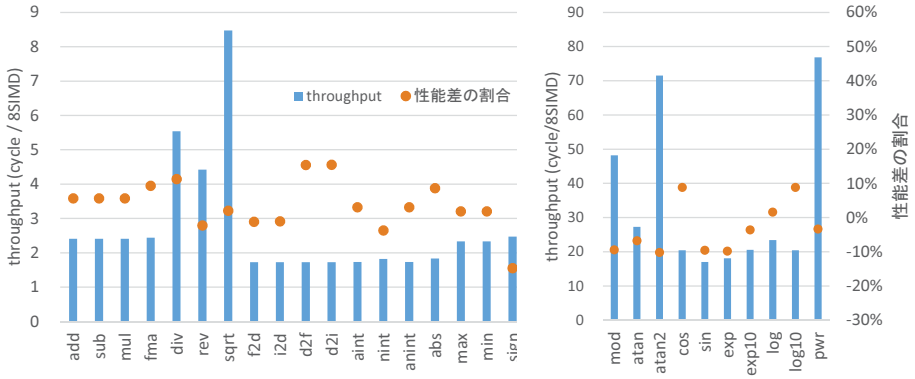

図 3 理研シミュレータにおける各カーネルの実行スループットおよびテストチップの性能差割合

ら単精度および 32 ビット整数への変換とその逆変換、および Fujitsu Fotran の組み込み関数である aint, nint, anint の倍精度からの変換の計 7 個。数値関数は、倍精度の絶対値、最大、最小、剰余、符号の 5 つの関数。数学関数は倍精度の atan, atan2, cos, sin, exp, exp10, log, log10, power の 9 つの関数。合計 28 個のカーネルについて評価を行った。表 1 のサイズの列は、カーネル部で参照する各配列のサイズを表しており、全体として L1 キャッシュの 3/4 を参照するサイズを選択しており、最内ループの繰り返し数 n に対応している。外側ループの繰り返し回数 iter は、テストチップでは 1000000 回としているが、理研シミュレータの実行速度は実機の 1 万倍くらい遅いため、外側ループの値は理研シミュレータはテストチップの 1/1000 の 1000 回としている。サイクルレベルシミュレータである理研シミュレータのタイマ精度は非常に高く、イテレーション数が少なくても十分な精度の時間測定が可能である。

これらのカーネルプログラムを、-Kfast オプションを指定してコンパイルしたバイナリを用いて、テストチップで実行した時間と、理研シミュレータで実行した時間とを比較した。基本的に富士通コンパイラではこれらすべてのカーネルは 8SIMD 実行されるとともに、数学関数もインライン展開され、ソフトウェアパイプラインによる最適化が有効となっている。除算や逆数についても、非パイプライン命令である除算命令を使うのではなく、パイプライン実行される reciprocal 命令を用いて計算している。

評価結果を図 3 に示す。棒グラフは、理研シミュレータの結果により求められた演算スループット (8 要素演算にかかるサイクル数）を表し、左軸に対応する。オレンジの点は、テストチップの結果と理研シミュレータの結果の性能差の割合であり、右軸に対応する。性能差が 10 %とは、理研シミュレータの実行時間がテストチップより 10 %長いことを、-10 %とは、テストチップの実行時間が理研シミュレータより 10 %長いことを示している。28 個のカーネル中、性能差が 10 %を超えるものは div, d2f, d2i, sign, atan2 の 5 つのみで、80 %以上のカーネルで性能差が 10 %以下となっていることを確認した。28 個の性能差の平均は 1.3 %、標準偏差は 7.8 %、絶対値の平均は 6.6 %である。

性能差の要因については、詳細を検討中である。d2f および d2i において、A64FX ではライトバッファにおける書き込みデータのマージ効果が期待できるのに対し、理研シミュレータではライトマージの機能がないことが原因であると考えられる。その他に、cos や log10 において理研シミュレータが 10 %近く遅いのは、gather load の違いと思われる。A64FX ではコンバインド gather load により最大 2 要素のアクセスを 1 サイクルで行えるの対して、理研シミュレータではコンバインド gather load をサポートしておらず、1 要素につき 1 サイクルかかるという違いがある。しかし、他の数学関数などは理研シミュレータの方が速いものもあり、その要因は現在検討中である。

5.2 マルチスレッドによる L2 キャッシュおよびメモリ性能の評価

次に、マルチスレッド実行時のメモリ階層性能を評価するために、2 つのデータサイズに対する Stream Triad の性能を、スレッド数を変更して比較した。

評価の 1 つ目は、L2 より少し小さいサイズに対する Stream Triad を評価した。図 4 に結果を示す。本結果は、L1 キャッシュに対してのみ 4 ライン先にソフトウェアプリフェッチを行う最適化をコンパイラオプションで指定した結果である。棒グラフが、同一データサイズに対してスレッド数を 1 から 12 に変化させた場合の、理研シミュレータにおけるトータル L2 スループットである。x 軸がスレッド数、y 軸は左の軸が対応している。スレッド数の増加に対して比較的スケーラブルな結果となっているが、8 スレッドを超えたあたりでバンド幅が飽和している。オレンジの点が、理研シミュレータと A64FX のテストチップのスループットの差の割合を示している。y 軸は右の軸が対応している。カーネルの評価と同じく、性能差が正の場合は理研シミュレータがその割合だけ A64FX のテスト

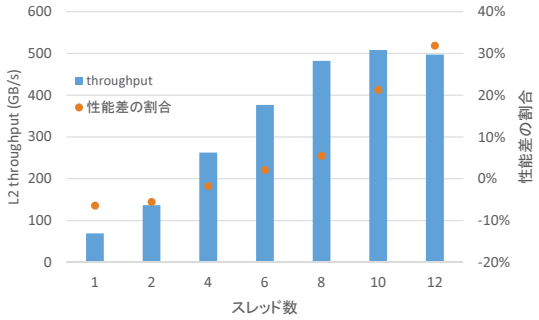

図 4　L2 に対する Stream Triad の理研シミュレータのバンド幅およびテストチップとの性能差

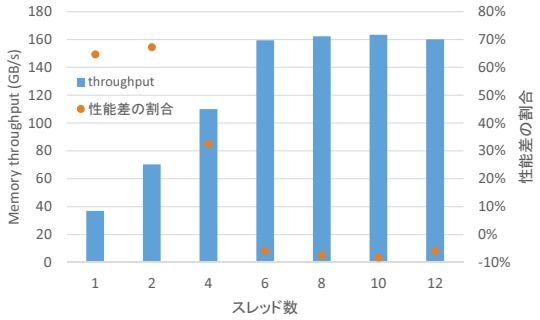

図 5　メモリに対する Stream Triad の理研シミュレータのバンド幅およびテストチップとの性能差

チップより遅いことを、負の場合は A64FX テストチップがその割合だけ理研シミュレータより遅いことを示している。図 4 よりスレッド数が少ない時には理研シミュレータがやや速く、スレッド数が大きくなると理研シミュレータが遅くなり、10 および 12 スレッドでは急激に差が大きくなり、12 スレッドでは 30 %以上遅くなっていることが分かる。

スレッド数が小さい時に理研シミュレータが速い理由は、L1 から L2 のライトバックの制御の違いと考えられる。A64FX では他の L1 アクセスとはライトバックが排他的に制御されているが、理研シミュレータでは独立に制御されている。この点については、A64FX に合せて修正を行う予定である。

スレッド数が大きい時に理研シミュレータが遅い理由の 1 つは、L2 キャッシュの公平性の制御と考えられる。A64FX では L2 キャッシュ側で各コアからの要求を公平にサービスする仕組みがあるが、理研シミュレータでは未実装である。そのため、スレッド数が多くなると、スレッド間の性能ばらつきが生じてしまい、一番遅いスレッドに性能が引っ張られるために、性能が低下してしまう事が考えられる。理研シミュレータでも何らかの公平性を制御する仕組みを検討中である。

また、A64FX の L1/L2 間クロスバはスレッド数が増えても性能低下が抑えられるように工夫されており、12 スレッドまで性能がスケーラブルに向上している。一方、理研シミュレータでは、宛先が異なる転送を 1 つの送信元がオーバラップして転送することができないため、クロスバの性能としては制限がある。このクロスバの性能制限を修正することは難しいと考えられるため、他の調整で性能を合わせられないかを検討中である。

評価の 2 つ目は、L2 の 2 倍のサイズに対する Stream Triad を評価した。これはメモリ性能を評価していることになる。図 5 に結果を示す。棒グラフが、同一データサイズに対してスレッド数を 1 から 12 に変化させた場合の、理研シミュレータにおけるトータルメモリスループットである。x 軸がスレッド数、y 軸は左の軸が対応している。6 スレッドでメモリスループットが飽和していることが分かる。オレンジの点が、理研シミュレータと A64FX テストチップのスループットの差の割合を示している。6 スレッドを超えたときのスループットの差の割合は 10 %以下である。一方で、スレッド数が小さい時にはその差の割合が 60 %を超え、かなり大きい。

この差の原因は、理研シミュレータでは、L2 に対するハードウェアプリフェッチ機能がまだ未実装であるためであると考えている。現在、L2 に対するハードウェアプリフェッチ機能は実装中であり、その完了を待って再度評価を行う予定である。

また、ここで示したメモリバンド幅、正確にはチップ内の CMG 数である 4 倍した値 640GB/s は、[2] で示しているプロセッサのメモリバンド幅 830GB/s より低い値となっている。これは、本評価の結果はハードウェアプリフェッチのみを用いたときのアプリケーションから見たメモリバンド幅であるのに対し、[2] の結果は、L2 キャッシュへのソフトウェアプリフェッチやストリーム書き込みの最適化を適用した場合の結果であるためである。現時点では、これらの最適化機能は理研シミュレータでは実装中であり、その完了を待って再度評価を行う予定である。

6. まとめ

理研ではポスト京プロセッサシミュレータを汎用プロセッサシミュレータ gem5 をベースとして開発している。本シミュレータは、詳細なパラメータチューニングと機能拡張を行い、基本アーキテクチャモデルに基づくアウトオブオーダ実行をシミュレーションすることができ、詳細な実行サイクル数を見積もることができる。この基本アーキテクチャモデルは、ポスト「京」のプロセッサである富士通 A64FX プロセッサとは、細部では異なっているが、アプリケーションチューニングが可能となる相対的な評価が行えるような精度で、ポスト京プロセッサにおける 1 ノードのアプリケーションの実行速度を見積もること目指している。

ポスト京のテストチップの実行サイクル数と比較することにより、本シミュレータの精度を確認した。28 個のカーネルプログラムでは、80 %以上の 23 個でその差が 10 %以

下であることを確認した。マルチスレッド実行の Stream Triad では、スレッド数に応じたスケーラブルな性能は確認できたが、L2 サイズデータではスレッド数が大きい時に、メモリサイズデータではスレッド数が小さい時に、性能差が大きいことが判明した。原因はターゲット L2 プリフェッチなどの機能が未実装であるためであり、これらの実装を継続して行う予定である。

本理研シミュレータを利用できる理研内の環境を、当初 FS2020 プロジェクトの重点課題や萌芽的課題のユーザに提供してきたが、昨年 9 月より RIST（高度情報科学技術研究機構）を通じてより広いユーザへの公募を開始している [5]。本シミュレータに興味のある人は、ぜひ利用を検討していただきたい。

謝辞 本稿の一部は，文部科学省「特定先端大型研究施設運営費等補助金（次世代超高速電子計算機システムの開発・整備等）」で実施された内容に基づくものである。

参考文献

[1] N. Binkert, B. Beckmann, G. Black, S.K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D.R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M.D. Hill and D.A. Wood. *The gem5 Simulator*, ACM SIGARCH Computer Architecture News, Vol.29, Issue 2, May 2011.

[2] Y. Yoshida, *Fujitsu High Performance CPU for the Post-K Computer*, 2018 IEEE Hot Chips 30 Symposium, 2.13, Aug. 2018.

[3] Y. Ajima, T. Kawashima, Takayuki. Okamoto, N. Shida, K. Hirai, T. Shimizu, S. Hiramoto, Yoshiro. Ikeda, T. Yoshikawa, K. Uchida and T. Inoue, *The Tofu Interconnect D*, IEEE International Conference on Cluster Computing, pp.646-654, Sep. 2018.

[4] N. Stephens, S. Biles, M. Boettcher, J. Eapen, M. Eyole, G. Gabrielli, M. Horsnell, G. Magklis, A. Martinez, N. Premillieu, A. Reid, A. Rico and P. Walker, *The ARM Scalable Vector Extension*, IEEE Micro, Vol.37, Issue 2, March/April 2017, pp.26–29.

[5] http://www.hpci-office.jp/pages/other_submission, ポスト「京」性能評価環境利用課題の募集について.