Vol.2021-HPC-180 No.13
2021/7/20

A64FXにおけるSPEC CPUおよびSPEC OMPの評価

児玉 祐悦[1,a)]　近藤 正章[1]　佐藤 三久[1]

概要：スーパーコンピュータ「富岳」で用いられているプロセッサ A64FX について、SPEC CPU および SPEC OMP ベンチマークを用いて評価を行った。その結果、A64FX の SPEC CPU int では Xeon よりも低いが、SPEC CPU fp ではソケット当たりの性能で評価すると Xeon を上回っていた。SPEC OMP では Xeon の Hyperthread の効果でソケット当たりでも Xeon を下回ったが、ベンチマークによっては高いメモリバンド幅により大きく上回るものもあった。また、A64FX の電力制御機構を用いた性能や電力の比較などを行い、コアリテンションを有効にすることにより、全コアを利用していないときには性能に影響を与えずに電力を削減できることを確認した。

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スーパーコンピュータとして「富岳」を開発し、2021 年 3 月より一般供用が開始された。本稿では、富岳で用いられているプロセッサ Fujitsu A64FX について、広くプロセッサ評価に用いられている SPEC ベンチマークを用いて評価を行ったので、その結果について報告する。また、その結果について他のプロセッサとの比較やコンパイラによる差異、電力性能などを考察する。

本稿では、まず、A64FX の概要について述べた後、A64FX における電力制御機構について説明する。次に、SPEC ベンチマークについて概要を述べ、A64FX の結果について報告する。その後、その結果についての種々の観点からの評価を行い、最後にまとめを述べる。

2. A64FX の概要

富岳の各ノードは 1 チップのプロセッサ Fujitsu A64FX[1] から構成される。図 1 にプロセッサの構成を示す。A64FX は Armv8-A の 64 ビットアーキテクチャであり、SVE(Scalable Vector Extension) を実装した最初のプロセッサである。A64FX は計算コアを 48 個持つメニコアプロセッサであり、内部では 4 つの CMG（Core Memory Group）に分かれている。各 CMG は 12 個の計算コア、1 個のアシスタントコア、8MB の共有 L2 キャッシュ、メモリコントローラから構成されている。

各コアは 512 ビットの SIMD 演算パイプラインを 2 本持っており、通常 2.0GHz で動作し、理論倍精度浮動小数

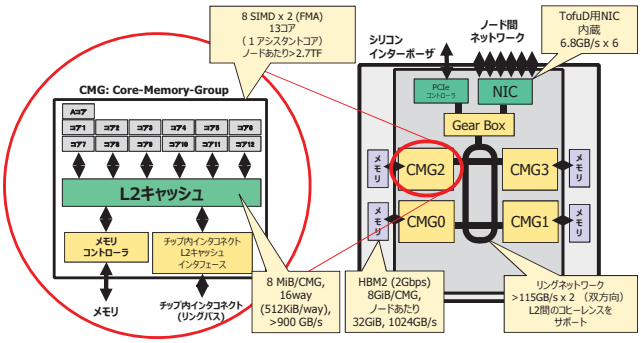

図 1　A64FX の構成

点演算性能は 64GFLOPS である。L1 キャッシュは各コアにデータ・命令各 64KB あり、ともに 4way で、キャッシュラインサイズは 256 バイトと SVE 向けに長めに設定されている。1 サイクルに最大 64 バイトのロード 2 回あるいはストア 1 回が可能で最大キャッシュバンド幅は 256GB/s である。L2 キャッシュは CMG 内に 16way で 8MB あり、CMG 内のコアで共有される。L2 キャッシュの最大バンド幅は 1024GB/s であるが、各コアの最大 L2 アクセスバンド幅は 128GB/s である。A64FX のマイクロアーキテクチャについては、[2] に詳細が公開されており、また各パラメータのコデザインによる決定については [3] に報告されている。

A64FX はパッケージ内に搭載された HBM2 を 1CMG あたり 1 基、1 チップでは 4 基搭載しており、容量 32GB、スループット 1024GB/s のメモリ性能を実現している。4 つの CMG はリングバスで接続されており、CMG 間のキャッシュコヒーレンスが維持される。ノード間ネットワークは Tofu Interconnect D (TofuD) という京と同様の 6 次元メッシュ／トーラス接続であり、ネットワークコン

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

トローラや PCI Express コントローラもチップ内に実装している。

電力効率の向上のため、A64FX では、最先端の半導体プロセスとして TSMC 社の N7 を採用するとともに、電力効率を高める回路設計を行った。さらに、次にあげるようないくつかの電力制御の機構を備えている。

2.1 ブーストモード

電力効率を重視して、富岳ではノーマルモードとして周波数は 2.0GHz としている。一方で、少しでも性能を向上させたい、という要求に応えるために、2.2GHz のブーストモードを用意している。しかし、A64FX ではノーマルモード時の電源電圧は可能な限り低く抑えているため、2.2GHz で動作させるためには電源電圧を上げる必要があり、ブーストモードでは電力効率は低下してしまう。また、A64FX では周波数は全コアで共通であるため、あるコアのみを 2.2GHz にすることは出来ない。

2.2 エコモード

A64FX では、アプリケーション性能を出来るだけ維持しつつ、そのアプリケーションでは使用しない回路の電力を削減することにより、電力効率を上げるという基本コンセプトに基づき、いくつかのパワーノブを有している。その中に、2 本ある浮動小数点演算パイプラインを 1 本に制限するパワーノブ FLAonly がある。しかし、大きな電力変動が起きても安定して動作させるためのスタンバイパワー等により、この FLAonly だけでは、電力を削減できないことが明らかとなった。そのため、FLAonly を適用した際の最大電力に合せてスタンバイパワーを制御するモードをエコモードとして新たに設定した。

このエコモードは、ブーストモードと直交した設定であるため、次の 4 つの組み合わせが可能である。ただし、ブーストモードは全コアで共通に設定されるのに対して、エコモードはコアごとに設定が可能であるため、ノーマルとノーマルエコ（以下ではエコと略記する）、あるいはブーストとブーストエコはコアごとの混在が可能である。

ノーマル　2.0GHz で浮動小数点演算パイプラインを 2 本使えるモード。

ノーマルエコ　2.0GHz で浮動小数点演算パイプラインを 1 本に制限したモード。

ブースト　2.2GHz で浮動小数点演算パイプラインを 2 本使えるモード。

ブーストエコ　2.2GHz で浮動小数点演算パイプラインを 1 本に制限したモード。

2.3 リテンション

ジョブが割り当てられないノードの電力を削減する目的で、富岳ではノードリテンションという状態を設定している。これは、1 つのアシスタントコア以外のコアをすべてコアリテンションという状態にする。コアリテンションとは、エコモードの説明で述べたスタンバイパワーなどをオフにした状態であり、アイドル状態のコア電力を削減することができる。コアリテンションの状態でもメモリ内容などは全て保持されているため、コアリテンションから通常のアイドル状態へは数ミリ秒で遷移可能である。コアリテンションはコアごとの設定が可能である。

ただし、直接、コアリテンションへの移行を指定できるわけではない。ユーザが、あるコアをコアリテンション可能と設定すると、OS がコアにプロセスが割り当てられていないことを判断して、そのコアをコアリテンションへと遷移させる。したがって、コアリテンション可能と設定していてもプロセスが実行されている間は通常と同じくコアは動作する。

2.4 Power API

上記で述べた電力制御は、富士通のジョブスケジューラのスクリプトでジョブ単位で設定することが可能である。また、Power API を用いてプログラム内で制御することも可能である。この Power API は Sandia Power API[4] をベースに富士通が開発したライブラリ [5] として富岳、および FX1000 で提供されている。ただし、FX700 ではサポートされていない。

富岳ではこの Power API を用いて、実測電力と推定電力の 2 種類のノード電力を取得できる。実測電力は CMU(Core Memory Unit) とよぶプロセッサボード上の POL(Point of Load) デバイスで測定した電力に基づく電力であり、約 5 ミリ秒間隔で更新される。実測電力には、ノード毎にばらつきがある。一方、推定電力はコア内の各ユニットの稼働率などから計算される電力であり、約 1 ミリ秒で更新される。推定電力は、ノード間のばらつきがなく、アプリケーションの電力評価・チューニングに用いることができる。推定電力は、ノード電力だけでなく、CMG 単位のコア電力や L2 キャッシュ電力、メモリ電力など電力内訳を取得することもできる。また、一部の推定電力は PMU(Performance Monitor Unit) を通じて取得することも可能で、FX700 でも利用可能である [6]。

3. SPEC

SPEC（Standard Performance Evaluation Corporation）[7] は、計算機の性能とエネルギー効率を評価するための標準化されたベンチマークとツールを開発・維持するために設立された非営利団体であり、1988 年から活動している。ベンチマークには、SPEC CPU®、SPEC ACCEL®、SPEC MPI®、SPEC OMP®、SPEC Cloud® など、さまざまな種類がある。

SPEC CPU は、もっとも古くからあるベンチマークス

イートで、これまで何度か更新されており、最新版はSPEC CPU 2017である。さまざまな計算機システムのワークロードを比較するベンチマークスイートで、いくつかのメトリックが使われている。ratioは、リファレンスマシンと比較した相対的なパフォーマンスが、各ベンチマーク毎に評価される。SPEC CPU 2017のリファレンスマシンはUltraSPARC-IV+(2.1GHz, 2cores x 4Chips)である。SPEC CPUの性能は、全ベンチマークの幾何平均をとる。全ベンチマークに共通のコンパイルオプションを適用した値をbase, 各ベンチマークに最適なオプションを適用した値をpeakと区別する。SPEC CPUには、整数演算性能を評価するintと、浮動小数点演算性能を評価するfpがある。また、シングルタスク（スレッド並列処理は可）の性能を評価するspeedと、マルチタスクスループットの性能を評価するrateがある。本稿では、base speedのみを評価した。

SPEC OMPは、OpenMPによるスレッド並列実行の評価を行うベンチマークスイートで、最新版はSPEC OMP 2012である。SPEC CPUと同様に、baseのみを評価した。ただし、リファレンスマシンはSPEC CPUとは異なり、Sun Fire X4140(AMD Opteron 2384, 2.7GHz, 4core x 2chips)である。

4. 評価

富岳上で、SPEC CPU 2017およびSPEC OMP 2012をインストールして、評価を行った。ただし、SPEC CPU 2017は標準のinstallコマンドで、ツールの構築が行えたが、SPEC OMP 2012はツール自体がやや古いため、arm版Linuxに対応しておらず、個別にツールのインストールが必要であった。

A64FXのビルドにはFujitsu compiler tcsds-1.2.30aを用いた。具体的なコンパイルオプションは以下の通りである。ただし、SPEC CPU intでは、一部実行時の結果チェックでエラーとなったため、`-ffj-no-fp-relaxed`を追加して、やや最適化を制限している。

```
COPTIMIZE = -Nclang -Ofast -mcpu=a64fx+sve
-ffj-no-fp-relaxed -ffj-eval-concurrent
-fsave-optimization-record -fopenmp -Nlst=t
-Koptmsg=2
CXXOPTIMIZE = -Nclang -Ofast -mcpu=a64fx+sve
-ffj-no-fp-relaxed -ffj-eval-concurrent
-fsave-optimization-record -fopenmp -Nlst=t
-Koptmsg=2
FOPTIMIZE = -Kfast,openmp -Nlst=t -Koptmsg=2
```

A64FXではノーマルモード(2.0GHz)で、以下の環境変数を指定して実行している。

```
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand
export XOS_MMM_L_ARENA_LOCK_TYPE=0
```

表 1　SPEC CPU 2017 intspeed

	Lang	Threads	A64FX	Xeon
600.perlbench_s	C	1	1.23	6.20
602.gcc_s	C	1	2.59	9.57
605.mcf_s	C	1	3.41	11.2
620.omnetpp_s	C++	1	1.27	7.31
623.xalancbmk_s	C++	1	1.54	9.46
625.x264_s	C	1	2.07	11.6
631.deepsjeng_s	C++	1	1.34	5.17
641.leela_s	C++	1	1.25	4.36
648.exchange2_s	F90	1	1.42	13.2
657.xz_s	C/OpenMP	48	8.29	23.5
int_base(g-mean)			1.97	9.07

1つ目の環境変数は、メモリ割り付けのタイミングを制御するもので、デフォルトではプログラムロード時にメモリ割り付けを行う。複数のCMGにまたがるマルチスレッド実行を行う場合には、ファーストタッチ時に割り付けを行うように指定する必要がある。2つ目の環境変数は、スレッド間でのメモリプールの方法を制御するもので、デフォルトではメモリ利用率を重視してスレッド間で共通のメモリプールを使用する。スレッド内で頻繁にメモリ割り当て要求を行う場合には、スレッド間の排他制御がボトルネックとなってしまうため、独立したメモリプールを用いるように指定する必要がある。

なお、本結果は`--reportable`というオプションを指定して実行しているが、一部コンパイラオプションの検証指定の不備によりinvalid runとなっている。ただし、コンパイラのオプションは正式なものであり、実行結果のチェックなどもパスしているため、有効な結果であると考えている。

4.1 SPEC CPU 2017 int

表1にSPEC CPU 2017 intの結果を示す。表には各ベンチマークの名前、記述言語、スレッド数、A64FXの結果、および比較のためにIntel Xeonの結果を示している。SPEC CPU 2017 intは10個のベンチマークからなり、1つを除いて、シングルコア実行となっている。これはベンチマーク的にはOpenMPを指定してコンパイルし、スレッド数48を指定して実行しているが、結果としてシングルコア実行されているという事である。

比較として挙げたIntel XeonはSPECのホームページでリストされているCisco UCS B200 M5(Intel Xeon Platinum 8168, 2.70 GHz, 24 cores x 2 sockets)の結果である。詳細は[8]を参照。

A64FXはXeonと比べて、1/4以下の結果となっている。これは、A64FXがHPC向けのメニコア型プロセッサで、周波数を低く抑え、アウトオブオーダリソースも限られているためと思われる。また、SPEC CPU intではSIMD化

表 2　SPEC CPU 2017 fpspeed

	Lang	Threads	A64FX	Xeon
603.bwaves_s	F	48	534	490
607.cactuBSSN_s	C++,C,F	48	67.6	182
619.lbm_s	C	48	74.6	44.0
621.wrf_s	F90,C	48	50.1	96.2
627.cam4_s	F90,C	48	41.9	105
628.pop2_s	F90,C	48	30.0	60.4
638.imagick_s	C	48	77.0	127
644.nab_s	C	48	46.6	288
649.fotonik3d_s	F90	48	86.3	82.6
654.roms_s	F90	48	79.1	155
fp_base(g-mean)			72.8	127

が難しい点も原因の一つであろう。

4.2 SPEC CPU 2017 fp

表 2 に SPEC CPU 2017 fp の結果を示す。表には int と同様に、各ベンチマークの名前、記述言語、スレッド数、A64FX の結果、および比較のために Intel Xeon の結果を示している。SPEC CPU 2017 fp は 10 個のベンチマークからなり、すべて 48 スレッド実行となっている。

比較として挙げた Intel Xeon は SPEC CPU int と同じシステムで、SPEC のホームページでリストされている Cisco UCS B200 M5(Intel Xeon Platinum 8168, 2.70 GHz, 24 cores x 2 sockets) の結果である。詳細は [9] を参照。

トータルスコアでみると、A64FX は Xeon と比べて、およそ 60 %の結果となっている。しかし、ソケット単位で比較すると、A64FX の方が 20 %ほど高いとみることもできる。また、SPEC CPU fp では、ベンチマークごとに A64FX と Xeon の差が逆転しているものもある。例えば、619.lbm_s では、A64FX の方が 70 %高い。おそらくメモリバンド幅が性能に与える影響が大きいと思われるが、詳細は考察で議論する。

当初は環境変数 XOS_MMM_L_ARENA_LOCK_TYPE を設定していなかった。627.cam4_s の性能が Xeon と比べて著しく低かったため、基本プロファイラを用いて、調べたところ、`__pthread_mutex_lock/unlock_full` で 8 割ほどの時間を費やしていることが分かった。関連のある環境変数として、この環境変数の設定を追加したところ、性能が 4 倍向上した。他のベンチマークについても再度評価をし直したところ、他に 2 つのベンチマークでも効果があり、他のベンチマークでは悪影響もほぼなかったため、この設定を採用した。これにより、トータルスコアでも 20 %ほど向上した。

4.3 SPEC OMP 2012

表 3 に SPEC OMP 2012 の結果を示す。表には SPEC CPU と同様に、各ベンチマークの名前、記述言語、スレッド数、A64FX の結果、および比較のために Intel Xeon の結果を示している。SPEC OMP 2012 は 14 個のベンチマークからなり、A64FX ではすべて 48 スレッド実行となっている。

表 3　SPEC OMP 2012

	Lang	Threads	A64FX	Xeon
350.md	F	48	2.63	62.6
351.bwaves	F	48	15.5	11.2
352.nab	C	48	3.00	12.9
357.bt331	F	48	5.82	16.0
358.botsalgn	C	48	5.22	10.5
359.botsspar	C	48	3.07	6.83
360.ilbdc	F	48	7.69	8.25
362.fma3d	F	48	4.28	11.3
363.swim	F	48	53.1	8.38
367.imagick	C	48	12.2	13.6
370.mgrid331	F	48	32.6	7.46
371.applu331	F	48	8.88	14.4
372.smithwa	C	48	12.8	11.8
376.kdtree	C++	48	3.22	9.24
base(g-mean)			7.77	12.0

比較として挙げた Intel Xeon は、SPEC のホームページでリストされている Cisco C240 M5(Intel Xeon Platinum 8288, 2.70 GHz, 28 cores x 1 sockets) の結果であり、SPEC CPU で比較したマシンとは異なっている。同じシステムが SPEC のホームページに載っておらず、24 コア x 2 ソケットのマシンもなかったため、コア数が 28 コアで Hyperthread を on にした 56 threads での結果を比較として挙げている。詳細は [10] を参照。

A64FX は Xeon と比べて、およそ 60 %の結果となっている。コア数を同じに正規化すると 70 %となる。しかし、ベンチマークによっては A64FX の方が高速なものも存在する。例えば、363.swim では A64FX が Xeon より 6 倍高速である。おそらくメモリバンド幅が性能に与える影響が大きいと思われるが、詳細は考察で議論する。逆に、350.md は Xeon が A64FX より 24 倍高速である。これについては、A64FX では SVE 化がほとんどされていないことが原因であり、マニュアルチューニングにより性能が向上することは報告されているが、コンパイラによる最適化はまだ適用されないため、今後の改良に期待したい。

5. 考察

5.1 SVE の効果

A64FX の評価を考察する上で、まず真っ先に気になるのが、SVE の効果がどれくらいであるかであろう。先の評価は、すべて最適化オプションとして Fortran では-Kfast、C では-mcpu=a64fx+sve として最大限の最適化を適用しており、SVE によるベクトル化が指定されているが、適用されるかどうかはプログラム依存である。そこで、最適化オプションで SVE を適用しない指定として Fortran

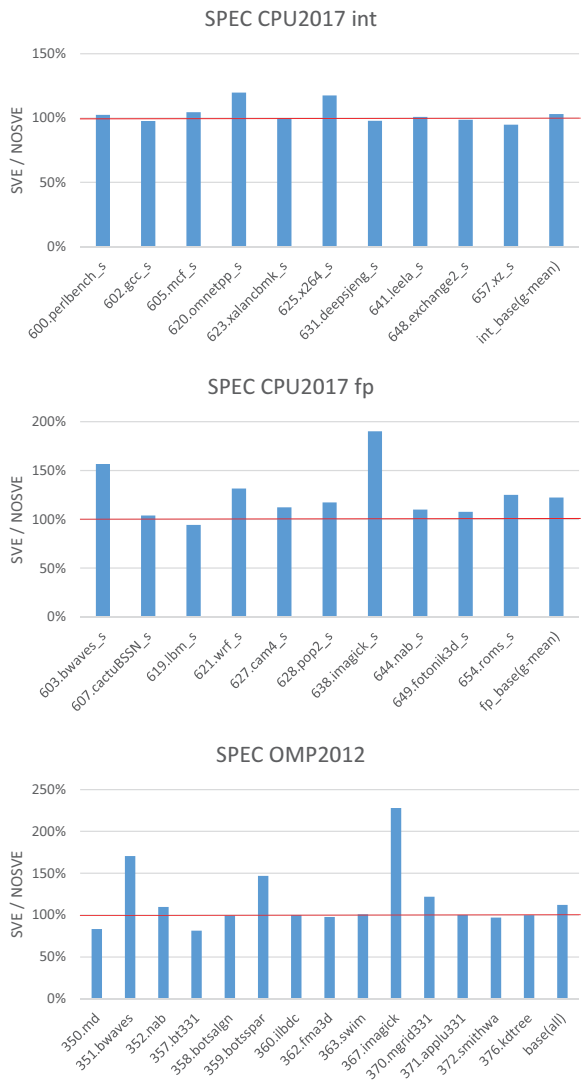

図 2 SVE の効果

では-KNOSVE、C では-mcpu=a64fx+nosve を指定し、他は同じ設定で評価してみた。結果を図 2 に示す。図では NOSVE の性能を 100 %として、SVE の性能を相対値で示している。

SPEC CPU int では SVE の効果は最大でも 20 %程度で、トータルスコアでは 3 %に過ぎない。整数系は SIMD が効きにくい傾向があるとはいえ、SVE では整数系の SIMD 命令もあるため、もう少し効果が出てほしいところである。ただし、SPEC ではソースを修正することは許されていないが、記述を少し変えるだけでも SIMD の効き方が変わってくる可能性はある。

また、SPEC CPU fp では、最大で 90 %の効果があるベンチマークがあり、トータルスコアでも 22 %の効果がある。しかし、SVE が最大限に発揮できれば double で 8 倍（NOSVE でも neon の SIMD 命令が使われるのでその場合は最大 4 倍）、float ではさらにその倍の性能向上が期待できるので、更なるコンパイラによる SVE 化を期待したい。

SPEC OMP では、最大で 230 %の効果があるベンチマークがある一方で、トータルスコアでは 12 %の性能向上に留まっている。これは、一部に SVE を無効にした方が性能が 20 %ほど高いベンチマークがあることも要因であると考えられる。SVE 化を有効にすると、gather/scatter の indirect アクセスのメモリ SIMD 命令が使われるため、これが逆に性能を低下させてしまう可能性もある。コンパイラの最適化により、真に有効な時のみ SVE を使うなどの改良が必要であろう。

5.2 gcc との比較

コンパイラによる違いについても、興味深い。4 節で示した結果は富士通コンパイラによる結果であった。ここではもう一つの選択肢である GNU Compiler による評価との比較を行う。富岳ではオープンソースのプログラムはパッケージ管理ツールである SPACK を用いてユーザが簡単に利用できるように整備されている。GNU Compiler 10.2.0 も SPACK を用いてロードするだけで、ユーザが利用することができる。gcc のオプションとしては以下を指定した。ただし、SPEC OMP 2012 では-flto(link time optimization) を指定すると、一部のベンチマークでビルドに失敗したのと、適用できたベンチマークでも-flto の効果はほとんどなかったため、-flto を省いている。

```
OPTIMIZE = -g -Ofast -ffast-math -flto
-march=native
```

結果を図 3 に示す。図では gcc の性能を 100 %として、fcc の性能を相対値で示している。

SPEC CPU 2017 int では、fcc が gcc の 50 %となるベンチマークがある一方で、fcc が gcc より 13 %高速になるベンチマークもあり、トータルスコアでは 11 %ほど gcc の方が性能が高いという結果となった。gcc では lto の効果が大きく、指定しないときに比べて最大で 35 %、トータルでも 8 %ほど性能向上があった。fcc でも lto を追加してみたのが fcc+lto である。ただし、コンパイラのバージョンも tcsds-1.2.31 に更新されている。fcc でも lto により最大で 27 %、トータルで 14 %の性能向上があり、トータルで gcc より 1 %良い結果となった。

SPEC CPU 2017 fp では、最大で fcc が gcc の 5 倍高速なベンチマーク（603.bwaves）もあるが、それ以外もすべて fcc の方が gcc よりも性能が高く、トータルでは 94 %高速、603.bwaves を除いても 75 %高速という結果であった。

SPEC OMP 2012 では、350.md で 7 倍、360.ilbdc では 32 倍と 2 つのベンチマークで gcc が極端に性能が低かった。トータルでは fcc の方が 57 %性能が高かった。しかし、この 2 つを除くと、fcc が 2 倍性能が高いベンチマーク（351.bwaves）から gcc の方が 2 倍程度性能が高いベンチマーク (376.kdtree) の間にあり、トータルでは 10 %の差に縮まる。

表 4　SPEC CPU/OMP における電力モードの比較

		SPEC CPU int		SPEC CPU fp		SPEC OMP	
power mode	retention	Ave.Power(W)	basespeed	Ave.Power(W)	basespeed	Ave.Power(W)	base
normal	disable	107.3 (100%)	1.94 (100%)	114.4 (100%)	72.8 (100%)	123.3 (100%)	7.77 (100%)
boost		117.3 (109%)	2.16 (111%)	124.6 (109%)	79.3 (109%)	134.1 (109%)	8.27 (107%)
eco		85.3 (80%)	1.96 (101%)	91.2 (80%)	66.8 (92%)	99.2 (80%)	7.45 (96%)
boost eco		91.1 (85%)	2.16 (111%)	95.9 (84%)	73.0 (100%)	105.7 (86%)	8.06 (104%)
normal	enable	42.3 (40%)	1.98 (102%)	93.6 (82%)	73.1 (100%)	113.0 (92%)	7.77 (100%)
boost		46.4 (40%)	2.16 (100%)	100.9 (82%)	79.0 (100%)	127.7 (95%)	8.43 (102%)
eco		42.0 (49%)	1.97 (101%)	77.9 (85%)	67.5 (101%)	95.0 (96%)	7.44 (100%)
boost eco		45.4 (50%)	2.16 (100%)	84.2 (88%)	73.2 (100%)	102.7 (97%)	8.07 (100%)

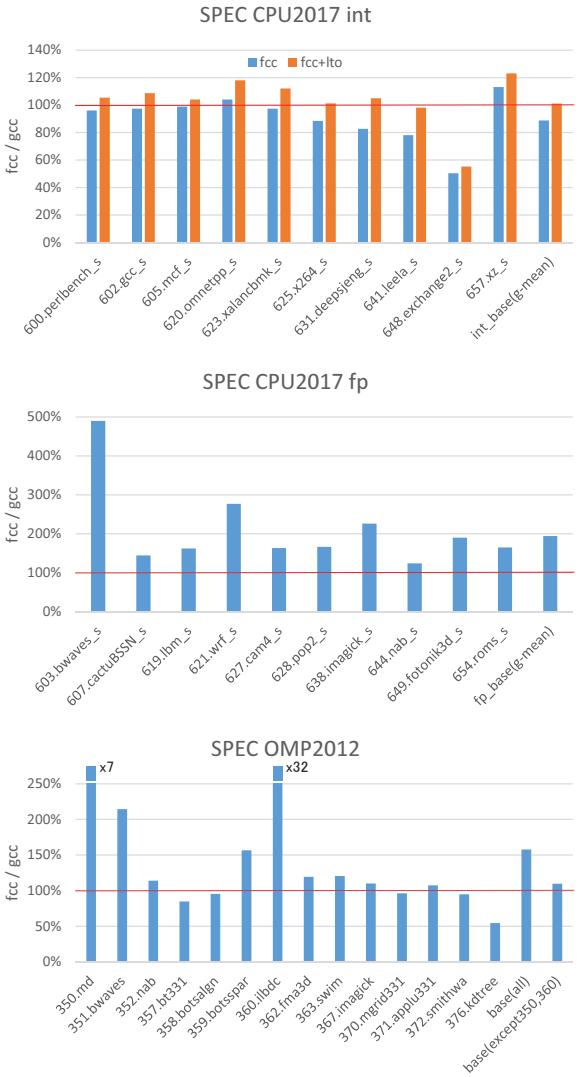

図 3　fcc と gcc の比較

5.3　電力モードによる比較

A64FX では、2.1-2.3 節に示したようにブーストモードやエコモード、コアリテンションなどいくつかの電力制御機構がある。電力モードとして 4 通りを示したが、リテンションの有無を組み合わせると合計 8 通りの電力モードがある。

SPEC CPU/OMP についてこの 8 通りを比較した結果を表 4 に示す。Ave.Power は各ベンチマーク実行時の見積もり電量を PowerAPI を用いて取得して、ratio に合せてその幾何平均を示している。

まずは、リテンションを disable にした状態を比較する。カッコ内の値はノーマルに対する割合を示す。ブーストモードの性能は SPEC CPU int, fp, OMP でそれぞれ 11 %、9 %、7 %向上し、電力はいずれも 9 %増加している。エコモードの性能は、SPEC int では変わらないが、SPEC fp および SPEC OMP では 8 %、4 %低下しており、電力はいずれも 20 %減少している。エコモードによる性能低下はベンチマークごとに異なり、SPEC CPU fp では最大 25 %低下しているものがあるが、変化が 5 %以下のベンチマークも 10 本中 5 本ある。SPEC OMP では最大 19 %低下しているが、変化が 5 %以下のベンチマークは 14 本中 9 本ある。ブーストエコモードでは、性能は 11 %、0 %、4 %向上しているが、電力は 15 %、16 %、14 %削減している。

次に、リテンションを enable にした状態を比較する。カッコ内の値はリテンション disable 時の同じ電力モードとの割合を示す。性能については、各電力モードでリテンションを disable とした場合とほぼ変わらない。一方、電力は、ノーマル状態で SPEC CPU int では 60 %削減されており、大幅な電力削減を確認できる。これは、SPEC CPU int の多くが 1 コアで実行されるために、残りの 47 コアがコアリテンション状態となり電力削減を行えるためである。

ただし、SPEC CPU int で唯一マルチスレッド化されている 657.xz_s でも 50 %の電力削減が確認された。本ベンチマークの実行を基本プロファイラで確認したところ、マルチスレッド実行されているのは全体の 25 %程度で、残りはシングルスレッドで実行されていることが分かった。また、OpenMP の制御を富士通に確認したところ、parallel region が終わるとマススタースレッド以外は busy wait となるが、200ms を超えると sleep mode に遷移するとのことであった。retention enable の場合は、このときに core retention state に遷移するため、電力削減効果が表れたと考えられる。このため、基本的に 48 スレッド実行される SPEC CPU fp や SPEC OMP でも retention enable を設定した場合、ノーマルモードで電力削減効果がそれぞれ

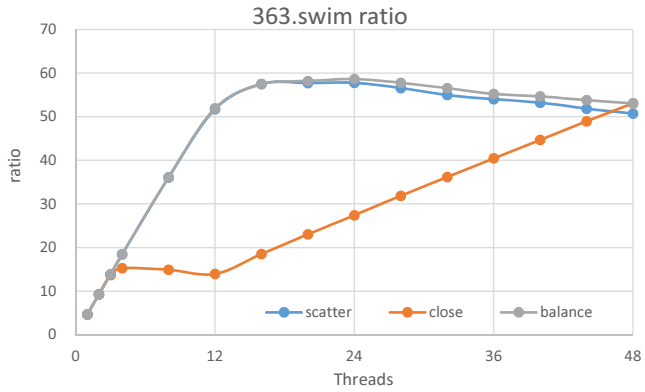

図 4　スレッドスケジューリングによる性能比較

17 %, 8 %確認できる。しかし、シングルスレッド実行の割合が小さいためか効果は小さくなっている。また、このような制御は busy wait が 200ms を超えた時だけであり、このような制御による性能への影響は確認されなかった。

5.4　スレッド数による比較

4.3 節で、363.swim で A64FX が Xeon より高い性能を示しているのは、メモリバンド幅によるものであろうと述べた。ここでは、それを検証するために、スレッド数を変化させて評価を行った。結果を図 4 に示す。横軸がスレッド数、縦軸が ratio による性能を示している。図では 3 種類のスレッドスケジューリングを評価している。

close がデフォルトの設定であり、4.3 節もこれを用いて 48 スレッド実行した結果である。close では 3,4 スレッドで性能がいったん飽和し、12 スレッドまで徐々に性能が低下し、その後性能が向上している。これは、A64FX では 48 個のコアは 4 つの CMG にクラスタ化され、各 CMG でメモリバンド幅が 256GB/s となっているためである。close は CMG0 から順にスレッドを割り当てる方式であり、12 スレッドまでは CMG0 のみにスレッドが割り当てられる。demand でメモリページ割り当てを行っているため、メモリも CMG0 のみに割り当てられる。そのため、12 スレッドまでは全体の 1/4 のメモリバンド幅しか使えない。16 スレッドでは 2CMG が利用され、最大メモリバンド幅は 2 倍となるが、CMG0 に割り当てられている 12 スレッドが 256GB/s を共有しているため平均 21.3GB/s のメモリバンド幅しか利用できない。それに対し、CMG1 に割り当てられている 4 スレッドは平均 64GB/s のメモリバンド幅を利用できる。各スレッドには同量の処理が割り当てられているため、CMG1 に割り当てられたスレッドは先に終了し、CMG0 のスレッドを待つことになる。そのため、12 スレッドを超えた後は、性能は最大メモリバンド幅ではなく、スレッド数の増加に比例して増加していく。

図 4 では、スレッドスケジュールとして close の他に、scatter と balance を用いた結果も示している。scatter は以下の GOMP_CPU_AFFINITY を指定して各 CMG にスレッドを均等にラウンドロビンで割り当てる方法である。なお、OMP_PROC_BIND=spread と設定することにより scatter と同様のスケジュールが期待されるが、この指定ではスレッドが各 CMG に均等に割り当てられるが、CMG 内での割り当ては OS に任せられており、同じコアにスレッドが割り当てられてしまい性能が低下してしまう場合もあるので注意が必要である。

```
export GOMP_CPU_AFFINITY="12-48:12 13-49:12
14-50:12 15-51:12 16-52:12 17-53:12 18-54:12
19-55:12 20-56:12 21-57:12 22-58:12 23-59:12"
```

一方、balance は同じく各 CMG にスレッドを均等に割り当てるが、各 CMG 内では連続して割り当てる方法である。例えば 16 スレッドの時には以下のように指定する。GOMP_CPU_AFFINITY による指定では、スレッド数ごとに設定を変更する必要がある。

```
export GOMP_CPU_AFFINITY="12-15 24-27 36-39
48-51"
```

scatter および balance では、12 スレッドまではスレッド数に比例して性能が向上するが、その後性能が飽和し、24 スレッドを過ぎると徐々に性能が低下していく。scatter と balance は 20 スレッドまではほぼ同じ性能であるが、24 スレッド以降は balance の方がやや性能が高い。48 スレッドでは balance は close と同じスレッドスケジューリングとなり性能も同じであるが、scatter は 48 スレッドの性能は他の 2 つより性能が 4 %ほど低くなっている。これは close や balance の方が L2 キャッシュにおけるデータ共有の可能性が高いことによるものと思われる。

balance の最大性能は 20 スレッドの時で、このときの性能は 48 スレッドの時より 10 %高い。これはスレッド数が多いとメモリや L2 キャッシュでのリクエストの衝突などのオーバヘッドが増えるためであると考えられる。このように、スレッド数及びスレッドスケジューリングをアプリケーションに合せて最適化することにより性能向上が図れる可能性がある。ただし、このようなベンチマークごとの最適化は base では許されておらず、peak としてスコアを示す必要がある。

また、利用しないコアがあるという事はリテンションによる電力削減効果も期待できる。そこで、balance を用いてスレッド数を変化させたときの各電力モードによる比較を行った。ratio による性能を図 5 に、1/Energy による電力効率を図 6 に示す。性能は、retention の設定ではほぼ違いはない。電力モードによる違いも小さく、12 スレッドでは normal と比べて、boost で 6 %高く、eco で 7 %低く、boost eco はほぼ同じである。24 スレッド以降はほぼ性能は同じである。最も性能が高いのは 24 スレッドの boosteco で 48 スレッドの normal より 11 %高い。一方、電力効率では 16 スレッド時の eco-retention が最も高く、同じ 16 スレッドの normal-retention より 3 %、normal より 45 %高い。こ

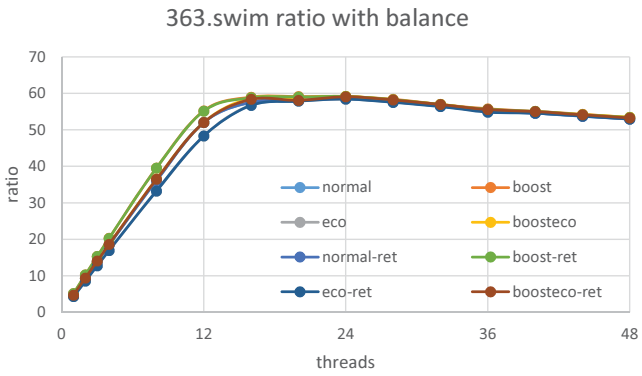

図 5　balance 時の電力モードによる性能比較

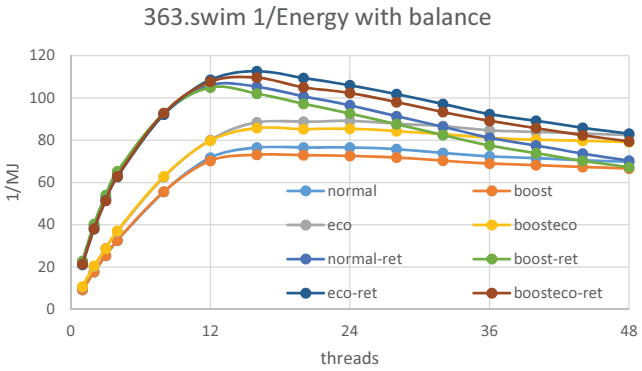

図 6　balance 時の電力モードによる電力効率比較

れを 48 スレッドの結果と比較すると、eco-retention より 35 %、normal より 61 %高い。その後、電力効率は徐々に低下していき、48 スレッドでは retention なしの各モードと同じになる。

6. まとめ

スーパーコンピュータ「富岳」で用いられているプロセッサ A64FX について、SPEC CPU および SPEC OMP ベンチマークを用いて評価を行った。その結果、SPEC CPU int では Xeo よりも低いが、SPEC CPU fp ではソケット当たりで評価すると Xeon を上回ることを確認した。SPEC OMP では Xeon の Hyperthread の効果でソケット当たりでも 7 割程度であったが、ベンチマークによっては大きく上回るものもあった。

結果について、いくつかの観点から考察を行った。

SVE の効果については、SPEC CPU int ではほとんど効果がなく、SPEC CPU fp や SPEC OMP でもトータルでは 22 %および 12 %の効果しかなかった。しかし、一部のベンチマークでは高い効果が認められるものもあり、コンパイラによる今後の SIMD 化率の向上に期待したい。

富士通コンパイラと GNU コンパイラの比較では、SPEC CPU int では両者はほぼ同じ性能であるが、SPEC CPU fp や SPEC OMP では富士通コンパイラの方が性能が高かった。GNU コンパイラでは今後のリリースで march として a64fx が指定できるという話も聞くので、最適化が向上することに期待したい。また、Arm コンパイラなど他のコンパイラについても評価を行いたい。

A64FX の電力制御機構を用いた性能や電力の比較では、コアリテンションを有効にすることにより、使用しないコアがあるときには性能に影響を与えずに電力を削減できることを確認した。特にシングルコアでの実行の時には電力を半減できる。また、全コアを利用する場合でも、wait するスレッドがある場合には電力削減の効果が期待できる。ブーストモードでは、多くのベンチマークで性能向上が確認できたが、電力も増加してしまうため、利用するか判断が必要であろう。エコモードでは、性能に影響を与えずに電力を削減できるベンチマークもあるが、性能が低下してしまうベンチマークもあるため、利用するかどうかを判断するにはプログラムの特性を把握する必要がある。ブーストエコは両者の特徴を持つため、より多くのプログラムで性能を維持したまま電力削減の効果が期待できるが、性能への影響はエコモードと同様に注意が必要である。

スレッド数を変化させた比較では、特にメモリバンド幅律速なベンチマークではスレッド数がある程度以上になると性能が飽和してしまう事を確認した。A64FX は他のプロセッサに比べて高いメモリバンド幅を有しているが、それでも 12 スレッド程度でメモリバンド幅を使い切ってしまう。その場合、適切なスレッド数で実行することにより、性能の改善や電力効率の改善が行えることを確認した。

今回の評価では、全ベンチマークで同一の最適化フラグを用いる base により評価を行ったが、今後は各ベンチマークごとに最適なフラグや実行時最適化を行う peak による評価についても行っていきたい。そのような評価は、ベンチマークだけではなく、一般的なアプリケーションへの適切な最適化指針を定めるうえで有効であると考える。

謝辞　SPEC ツールのインストールや各ベンチマークのビルド方法などについて助言をいただいた富士通株式会社の皆様に感謝いたします。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] Y. Yoshida, *Fujitsu High Performance CPU for the Post-K Computer*, 2018 IEEE Hot Chips 30 Symposium, 2.13, Aug. 2018.
[2] https://github.com/fujitsu/A64FX/, *A64FX Microarchitecture Manual*
[3] M. Sato et al., Co-Design for A64FX Manycore Processor and Fugaku, SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, 2020, pp. 1-15, doi: 10.1109/SC41405.2020.00051.
[4] R. E. Grant, M. Levenhagen, S. L. Olivier, D. DeBonis, K. T. Pedretti and J. H. Laros III, "Standardizing Power Monitoring and Control at Exascale," Computer, vol. 49, no. 10, pp. 38-46, Oct. 2016.

[5] FUJITSU Software Technical Computing Suite V4.0L20, ジョブ運用ソフトウェア API ユーザーズガイド Power API 編, J2UL-2545-01Z0(00), Feb. 2020.
[6] E. Arima, Y. Kodama, T. Odajima, M. Tsuji and M. Sato, "Power/Performance/Area Evaluations for Next-Generation HPC Processors using the A64FX Chip," 2021 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS), 2021, pp. 1-6, doi: 10.1109/COOLCHIPS52128.2021.9410320.
[7] https://www.spec.org/, *SPEC: Standard Performance Evaluation Corporation*
[8] https://www.spec.org/cpu2017/results/res2018q2/cpu2017-20180529-06367.txt, *SPEC CPU2017 Integer Speed Result Cisco UCS B200 M5*
[9] https://www.spec.org/cpu2017/results/res2018q2/cpu2017-20180528-06124.txt, *SPEC CPU2017 Floating Point Speed Result Cisco UCS B200 M5*
[10] https://www.spec.org/omp2012/results/res2019q2/omp2012-20190313-00172.txt, *SPEC OMPG2012 Summary Cisco UCS C240 M5*