様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

令和　６ 年　 ６ 月 １９ 日現在

機関番号：１３９０１
研究種目：若手研究
研究期間：2022～2023
課題番号：２２Ｋ１７８９８

研究課題名（和文）実アプリケーションの時空間ブロッキングによる高速化に関する研究

研究課題名（英文）A Study on Acceleration by Temporal Blocking for Real-world Applications

研究代表者
星野　哲也（Hoshino, Tetsuya）

名古屋大学・情報基盤センター・准教授

研究者番号：４０７７５９４６

交付決定額（研究期間全体）：（直接経費）　　1,100,000円

研究成果の概要（和文）：微分方程式を解析的に解く際に生じる時・空間の離散格子に対する特定の計算パターンはステンシル計算と呼ばれ、様々な流体シミュレーションにおいて頻出する重要なカーネルである。ステンシル計算の高速化は盛んに研究されており、時空間ブロッキング手法はその一手法であるが、非常に煩雑なプログラミングを要求するため、実アプリケーションへの適用例はほとんどない。さらに、時空間ブロッキングの性能は実行するプロセッサの性能パラメータに大きく依存するため、人手によって最適化することは現実的ではない。そこで本研究では時空間ブロッキングの自動最適化に必要な性能モデリングを、最新のCPUを用いて行った。

研究成果の学術的意義や社会的意義
本研究では、主にHigh Bandwidth Memory（HBM）を搭載した最新のCPUである、富岳スパコンのA64FXや、Intel Xeon Sapphire Rapids世代のCPUを用いて、性能モデル化を進めた点に大きな価値がある。時空間ブロッキング手法はその性質上、特にメインメモリの性能とラストレベルキャッシュの性能比に性能が大きく依存する。この性能比はHBMの登場によって既存のCPUと大きく変化し、本研究では性能モデルによってその影響を明らかにしたことが、高性能計算分野において意義のある成果である。また当初想定していなかった、命令レイテンシの影響を明らかにした点も意義がある。

研究成果の概要（英文）：The specific calculation pattern for a discrete grid in time and space that arises when solving differential equations analytically is called a stencil calculation, and it is an important kernel that frequently appears in various fluid simulations. Acceleration of stencil calculations has been studied extensively, and the temporal blocking method is one such method, but has rarely been applied to real applications because it requires very complicated programming. Furthermore, since the performance of temporal blocking is highly dependent on the performance parameters of the processor executing the blocking, it is not realistic to optimize the blocking manually. Therefore, in this study, the performance modeling required for auto-tuning of temporal blocking was performed using state-of-the-art CPUs.

研究分野：高性能計算

キーワード：高性能計算　ステンシル計算　時空間ブロッキング　自動チューニング

様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９（共通）

１．研究開始当初の背景

微分方程式を解析的に解く際に生じる時・空間の離散格子に対する特定の計算パターンはステンシル計算と呼ばれ、流体などのシミュレーションにおいて頻出する重要なカーネルである。ステンシル計算では、ある時刻 T+1 における空間格子点(x、y)を更新するために、時刻 T における近傍点、例えば(x、y)、(x+1、y)、(x-1、y)、(x、y+1)、(x、y-1)の値を参照する。一般に空間格子点の集合は配列としてメモリ上に保持され、時間ステップの進行の度にこの配列を更新するため、ステンシル計算の性能は多くの場合メモリアクセス速度に律速される。これに対し時空間ブロッキングは、メモリ性能による限界を超えてステンシル計算を高速化する手法として盛んに研究されている。時・空間格子をキャッシュメモリに乗るよう小ブロックに区分し、メモリに書き戻すことなく N 時間ステップ分の計算を行うことで、メモリアクセスコストを最大 1/N まで低減する。しかしステップ数 N が増えるに伴い、さらに外側の近傍点まで参照範囲が広がるため、ブロックサイズが肥大化し必要なキャッシュ容量が増加する。とりわけ、3 次元的でかつ計算に必要な変数の多い実アプリケーションでは、キャッシュ容量の問題はより深刻化する。またブロックは更新すべき領域より大きくなるため、他のブロックと重なる冗長計算領域が生じ、その分だけ浮動小数点演算数は増加する。近年では冗長計算が必須な overlapped 方式（図 1:左上）を避け、冗長計算の必要がない wavefront、 trapezoid、 diamond（図 1:右上、左下、右下）などの方式が提案されている。ただしこれらの方式はブロックの計算順序に依存関係があるために並列化が難しく、単純なステンシル計算であっても非常に煩雑なプログラミングを要求するため、実アプリケーションへの適用例はほとんどない。

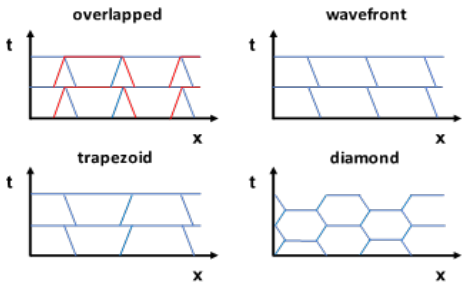

図 1：1 次元ステンシル計算における時空間ブロッキングの方式例。

一方で、Intel Xeon Ice Lake や AMD Milan など、スパコンに搭載される最新世代の CPU は、高い浮動小数点演算性能に加え、数十～数百 MB の大きなラストレベルキャッシュを有する。一部のメニーコアプロセッサや GPU が高いメモリ転送性能を持つ代わりに容量の小さい High Bandwidth Memory (HBM)を登載する一方、汎用 CPU には容量に優れる DDR メモリに加え強力な共有キャッシュを持たせるのが近年の傾向である。この大きな共有キャッシュを始め、最新世代 CPU のメモリや演算性能のバランスの変化により、並列化が容易で比較的単純なコード変形によって実現し得る overlapped 方式の時空間ブロッキングであっても、十分に高速化可能であることが研究代表者らの研究でわかってきているが、HBM を用いた CPU のように既存の汎用 CPU と大きく性能バランスの異なる最新の CPU において有効となる性能パラメータは明らかではない。

２．研究の目的

本研究の最終的な目的は、既存のステンシルアプリケーションへの時空間ブロッキング手法の自動適用および最適化である。本研究課題ではその前段階として、A) 最新世代の CPU における時空間ブロッキングの性能モデリング、B) 性能モデルに基づく実アプリケーションへの時空間ブロッキングの手動適用による高性能化、の 2 点を目的とする。時空間ブロッキングにおいて時・空間ブロックのサイズを最適化するためには、参照する近傍点の数や相対位置、近傍点が保有する物理量の数、CPU のメモリ・キャッシュ・レジスタの容量や性能、といった多数のパラメータが関わっており、性能モデリングを行うことは容易ではなく、特に最新世代の CPU の性能バランスを考慮した時空間ブロッキングの性能モデリングは例がない。

３．研究の方法

まず A64FX や Intel Xeon Sapphire Rapids などの最新世代 CPU について、各種ベンチマークにより詳細な評価を行い、性能モデルを構築するためのデータを蓄積する。その上で時空間ブロッキング手法において有効なパラメータの特定、有効パラメータを推定するための性能モデルの構築を目指す。

４．研究成果

近年のメニーコアプロセッサは、性能バランスが変わってきている。Intel の最新世代の CPU である Sapphire Rapids は、Intel 社の x86 製品としては初めて High Bandwidth Memory (HBM) が用いられている。同じく HBM を搭載した「富岳」のプロセッサである A64FX は、一般的な DDR メモリを用いた Intel Xeon CPU と異なる特性を持つことが知られており、アプリケーションの最適化戦略に影響を与えている。HBM を搭載した Sapphire Rapids についても、異なる最適化戦略が必要となる可能性があるが、Sapphire Rapids に関する評価は未だ十分ではない。そこで本課題では、複数のマイクロベンチマークを用いて、Sapphire Rapids を評価した。同じく HBM を採用する A64FX や、IceLake 世代の Intel Xeon CPU と比較を行った。

①　stream ベンチマーク

Sapphire Rapids with HBM のメモリ性能について、指標としてよく用いられる Stream Triad (c = alpha * a + b)の結果を図 2 に示す。実験に用いたのは京都大学の Camphor スーパーコンピュータシステムの Sapphire Rapids であり、コア数は 56 であるが、ハイパースレッディングが有効となっているため、それを考慮してスレッドとコアの割り付けを行う必要がある。また Intel コンパイラのオプションとして-qopt-streaming-stores があるが、これを-qopt-streaming-stores=always とすることで、ストリーミングストアの最適化が有効化される。ストリーミングストアとは、キャッシュへの書き込みを行わずにメモリへ結果を書き込むストアを行う方法で、キャッシュ書き込み分のオーバーヘッドを削減できる。

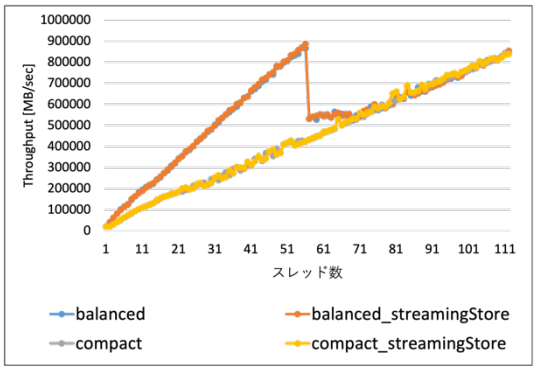

図 2: Sapphire Rapids における Stream Triad。

ベンチマークの配列サイズは 100M 要素とし、キャッシュに乗らないサイズに設定している。コンパイラには Intel compiler 2021.7.1、オプションとしては-xCORE-AVX512、-qopenmp、-O3 を用い、最大 56 コアを使用した。またコンパイラオプションの -qopt-streaming-stores=always の有無、環境変数として KMP_AFFINITY を balanced または compact の設定に変更して実験した。実行時には numactl コマンドにより、コアに近いメモリを利用するよう設定している。

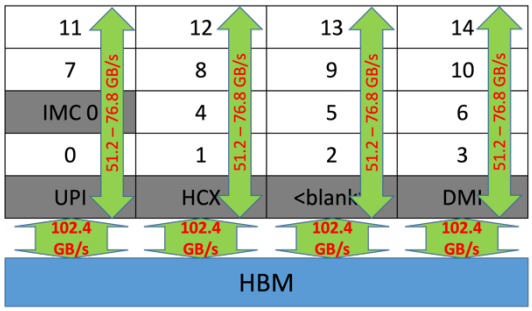

図 3: Sapphire Rapids のコア構成（1 numa 分）と帯域幅。

図 2 はスレッド数を 1 から最大まで変化させた時の性能を示している。Sapphire Rapids は最大 56 コアだが、ハイパースレッディングを考慮し、112 スレッドまで計測している。ここで重要なのは、Sapphire Rapids は理論値で 1.6TB/sec のメモリバンド幅性能を持つが、何らかの理由でこの性能を引き出し切れていないことである。性能の最も高い部分を見ると、-qopt-streaming-stores=always とすることで 3%程度性能が向上しているが、それでも 880 GB/sec 程度で、理論性能の 55%しか得られていない。A64FX が HBM の 80%以上の性能を引き出していることと比較するとかなり低い。

この理由は、John D. McCalpin (ISC 2023)らの報告（https://www.ixpug.org/images/docs/ISC23/McCalpin_SPR_BW_limits_2023-05-24_final.pdf）で示されているが、HBM の帯域幅をコア間のメッシュの Y 方向の通信性能が下回るためである。

図 3 は、Sapphire Rapids のコアの構成（1 numa 分）を示したものである。コア間は X-Y の二次元メッシュで接続されており、メッシュ間の 1 リンクの通信性能は 51.2～76.8 GB/s である。なお幅があるのは、熱によりクロック周波数が変動するためであり、メモリバンド幅を使い切るようなカーネル実行時には 51.2 GB/s に近づくと考えられる。通信リンクは 4 系統× X-Y の 2 方向あるため、メッシュの総バンド幅は HBM と同等か上回るが、HBM へのアクセス時には図 3 のように Y 方向の通信に律速されるため、HBM の性能が活かしきれない。

以上の結果から、アプリケーションの性能を考える上では、メモリバンド幅は単純なメモリ性能のみによって決まるわけではなく、コア間のメッシュ性能なども考慮する必要があることが判明した。

② キャッシュ性能に関するベンチマーク

近年の CPU は 100 MB を超える大きな共有キャッシュを備えるが、共有キャッシュの性能はコア間のメッシュの性能に依存するため、性能モデル構築の上で考慮する必要がある。図 4 は、Sapphire Rapids のメモリ階層を示したものである。同様に HBM メモリを備える A64FX は、L1 キャッシュがコアローカルのキャッシュで、L2 キャッシュが共有キャッシュであるが、Sapphire Rapids では L1 及び L2 キャッシュがコアローカルのキャッシュで、L3 キャッシュが共有キャッシュとなっている。

そこで A64FX と Sapphire Rapids の両者において、図 5 のコードを用いて評価を行なった。なお、A64FX では SVE 命令に置き換えたものを使用した。

図 5 のコードを OpenMP を用いて並列化し、以下の 3 種を試した。

- normal：スレッドローカル領域で A(:) = A(:) + B(:)を繰り返す。
- normal_w/barrier：for 文終了後に#omp barrier を呼ぶ。
- round_w/barrier：読み書きする領域をスレッド間でラウンドロビンに交換しながら A(:) = A(:) + B(:)を繰り返す。

Sapphire Rapids 及び A64FX における結果をそれぞれ図 6 及び図 7 に示す。A64FX では normal_w/barrier と round_w/barrier の性能がほぼ一致しているが、Sapphire Rapids では乖離している。これは A64FX が両者で L2 キャッシュを使う一方で、Sapphire Rapids は normal_w/barrier を L2 キャッシュ、round_w/barrier を L3 キャッシュを使って実行するためである。グラフの Y 軸はコアあたりの性能であるが、これを全コアでの性能に換算すると、Sapphire Rapids の round_w/barrier の性能は Stream triad のスループットの 2 倍強となる。これは、ラウンドロビンによるメッシュ間のデータ交換では、図 3 に示した X-Y 両方向のリンクを利用できているためと考えられる。

上述の結果から、アプリケーションの性能モデルを考える上では、コア間の通信方法、共有キャッシュの性能、コアローカルなキャッシュの性能など、考慮すべきパラメータがさらに増加傾向であることがわかる。

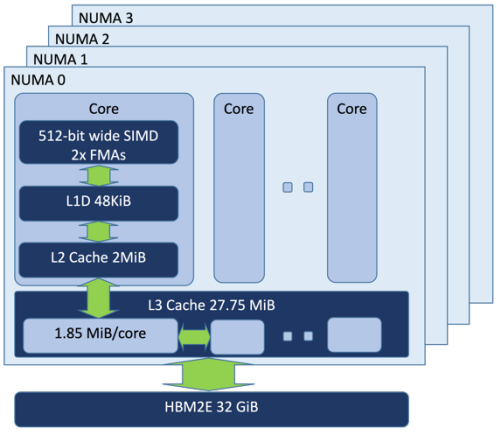

図 4: Sapphire Rapids のメモリ階層。

```
for(i = 0;i < size;i+=8){
  __m512d veca = _mm512_load_pd(A+i+id*size);
  __m512d vecb = _mm512_load_pd(B+i+id*size);
  veca = _mm512_add_pd(veca,vecb);
  _mm512_store_pd(A+i+id*size,veca);
}
```

図 5: キャッシュ性能を測るためのマイクロベンチマークコード。

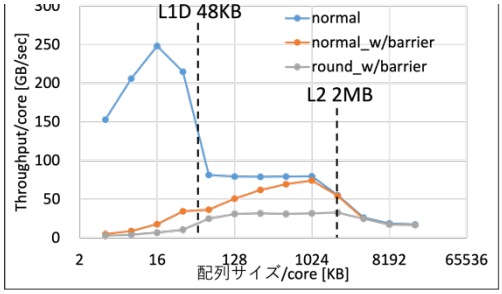

図 6: 図 5 のコードの Sapphire Rapids による評価結果

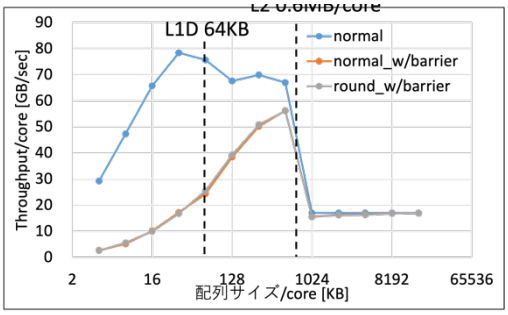

図 7: 図 5 のコードの A64FX による評価結果

```
for(i = 0; i < M; i++){
  vecc01 = _mm512_fmadd_pd(veca,vecb,vecc01);
  vecc02 = _mm512_fmadd_pd(veca,vecb,vecc02);
     :
  veccN  = _mm512_fmadd_pd(veca,vecb,veccN);
}                                          Statement数
 :
vecc = _mm512_add_pd(veccN ,vecc); // リダクション
```

図 8:ピーク性能を達成する上で必要なパイプライン並列数、レジスタスピルの影響を測る目的のマイクロベンチマークコード。

③ 命令レイテンシに関するベンチマーク

上述の実験では、近年の CPU の性能モデルを考える上では、単にメモリの性能だけではなく、コア間のメッシュ性能など、様々な要因がアプリケーションの性能に影響することを示した。CPU のコア性能に関しても、単にピーク性能だけでなく、命令レイテンシやレジスタ容量についても考える必要がある。A64FX、Sapphire Rapids 及び Intel Xeon IceLake を用いて、図 8 のコードによる評価を行なった。なお、A64FX では SVE 命令に置き換えたものを使用した。

図 8 のコードは、c=c+a*b を繰り返すループを変形したものである。通常このループはコンパイラの最適化により高速に実行されてしまうが、intrinsics を用いて実装することで敢えてパイプライン並列化を阻害するコードとなっている。ループイテレーション間で命令の依存性がある（右辺に現れる変数 veccN が前ループイテレーションの左辺変数として現れている）ためであるが、ループボディ内で別変数に書き込む statement を増加させると、その分だけパイプライン並列化が可能となる。

各プロセッサにおいて、ループボディ内の独立な statement 数を 1～40 まで変化させた際の性能を、図 9 に示す。結果から、ピークに達する（つまり FMA 命令のレイテンシを隠し切る）までに必要な独立な statement 数（つまりパイプライン並列数）は、Sapphire Rapids 及び IceLake では 10、A64FX では 20 となることがわかる。これは FMA 命令+代入命令のレイテンシ（Sapphire Rapids と IceLake は 4 + 1 cycle、A64FX は 9 + 1 cycle）と FMA 演算器の数を（全プロセッサで 2 FMA）を考慮すると妥当な数値である。また A64FX は statement 数 31 で性能が大幅に低下している一方で、Sapphire Rapids と IceLake では性能低下が緩やかである。Statement 数 31 の時に必要なベクトルレジスタ数が 33 であり、32 を超えたことによる性能低下は妥当であるが、プロセッサ（あるいはコンパイラ）によってその度合いは大きく異なることがわかった。このように A64FX のように命令レイテンシの大きいプロセッサでは、ピーク性能を得られるスイートスポットが狭いことなども考慮して性能モデル化する必要性があることがわかる。

④ ステンシル計算の時空間ブロッキングを対象とした性能モデリング

アプリケーションに近いベンチマークとして、7 点ステンシルと呼ばれる、流体計算に頻出する 3 次元の拡散方程式のカーネル（図 10）を利用する。特に、時空間ブロッキングと呼ばれる

高速化手法に着目し、性能モデル化を目指す。7 点ステンシル計算に、時空間ブロッキングを含む以下の最適化を適用し、Sapphire Rapids、A64FX、Intel Xeon CascadeLake を用いて評価した。

- BASE: 図 10 のベースライン実装。
- FT: ファーストタッチを行った実装。
- PEEL: ループピーリングや branch hoisting と呼ばれる、最内ループ中の分岐をループの外に出す手法。例外処理の発生するループの端の処理のみ分離する。
- 図 10 の 2 行目の omp parallel for 指示文を、nowait 指示節を付与した上で 4 行目のループに適用し、y 軸分割を行う実装。
- Y-Zdim: omp parallel for 指示文を用いず、スレッド番号を利用し、y-z のループを 2 次元分割する。この際 z 軸ループは NUMA 単位で 4 分割し、y 軸ループはコア単位で分割する。
- INT: AVX512 や SVE の Intirnsics を利用した実装。
- UNR: ループアンローリングにより、ループ内で実行される独立な命令数を増やすことで、パイプライン実行の効率化を図った実装。
- REG: x 軸方向のレジスタブロッキングを行った実装。また AVX512 の _mm512 _alignr_epi64()命令を利用することで、非アラインドなメモリアクセスを回避する。
- TILE: タイリングによるキャッシュの利用効率化を行った実装。
- TB: 時空間ブロッキング（別名：テンポラルブロッキング）を適用した実装。

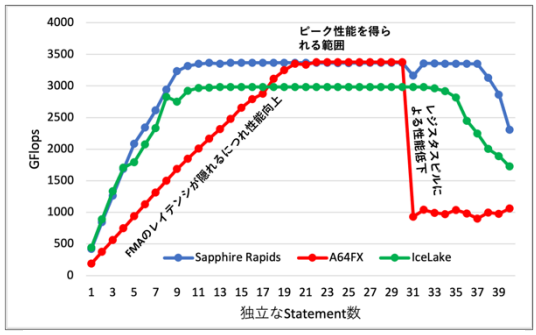

図 9: 図 8 のコードを Sapphire Rapids，A64FX，Intel Xeon IceLake で評価した結果。

```
do {
#pragma omp parallel for
  for (int z = 0; z < nz; z++) {
    for (int y = 0; y < ny; y++) {
      for (int x = 0; x < nx; x++) {
        int c =  x + y * nx + z * nx * ny;
        int w = (x == 0)    ? c : c - 1;
        int e = (x == nx-1) ? c : c + 1;
        int n = (y == 0)    ? c : c - nx;
        int s = (y == ny-1) ? c : c + nx;
        int b = (z == 0)    ? c : c - nx * ny;
        int t = (z == nz-1) ? c : c + nx * ny;
        f2_t[c] = cc * f1_t[c]
           + cw * f1_t[w] + ce * f1_t[e]
           + cs * f1_t[s] + cn * f1_t[n]
           + cb * f1_t[b] + ct * f1_t[t];
      }
    }
  }
  double *tmp = f1_t;
  f1_t = f2_t;
  f2_t = tmp;
  time += dt;
} while (time + 0.5*dt < 0.1);
```

図 10: 7 点ステンシルのカーネル。

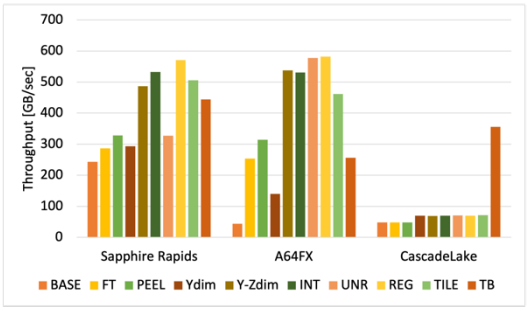

図 11: 7 点ステンシルに順次最適化を適用した結果。

図 11 に、上記最適化を順次適用した結果を示す。全体的な傾向として、Sapphire Rapids の結果は CascadeLake よりも A64FX の性能の傾向と似ている。上記最適化のうち、Y-Zdim が効果的であることがわかる。これは、NUMA 間で発生するキャッシュコヒーレントのための通信を最小化するための分割手法である。つまり、Sapphire Rapids では、NUMA 間通信が性能の足を引っ張ることが考えられるため、その点に気を付ける必要がある。UNR で表されるループアンローリングは、A64FX で有効である一方、Sapphire Rapids では性能を低下させる要因となった。A64FX で有効だった理由は、上述の命令レイテンシによるものと考えられる。ステンシル計算は一変数への足し込みを行うため、命令間に依存が生じる。ループアンローリングはループボディ内の独立な statement 数を増やす効果があるためである。一方 Sapphire Rapids で性能が低下した原因は性能モデルからは説明がつかず、他の要因を考える必要がありそうだが、原因の究明には至っていない。また TB で表される時空間ブロッキングは、DDR メモリを搭載する従来型の CPU である CascadeLake で非常に有効であるものの、Sapphire Rapids では有効ではなかった。時空間ブロッキングはメインメモリへの負荷を抑え、キャッシュの負荷を増加させる手法である。今回適用した時空間ブロッキング手法は、ラストレベルの共有キャッシュを利用する手法であったため、上述した通り、ラストレベルキャッシュが負荷の増加に耐えられなかったと考えられる。

A64FX においても時空間ブロッキングは有効ではなかったが、こちらはプロファイリングの結果を見るに、時空間ブロッキングで冗長な計算が増えたことによって、キャッシュではなくコアの負担増によって性能が低下したと考えられる。上記のように、プロセッサのパラメータによって最適化方針が全く異なることがわかり、また性能モデルから律速原因の説明が可能であることがわかった。A64FX 及び Sapphire Rapids での高速化のためには、単なるパラメータの調整ではなく、実装方針を切り替える必要があり、性能モデルに基づく自動最適化は今後の課題である。

５．主な発表論文等

〔雑誌論文〕　計3件（うち査読付論文　3件／うち国際共著　0件／うちオープンアクセス　0件）

１．著者名 Ren Xuanzhengbo、Kawai Masatoshi、Hoshino Tetsuya、Katagiri Takahiro、Nagai Toru	４．巻 none
２．論文標題 Auto-tuning Mixed-precision Computation by Specifying Multiple Regions	５．発行年 2023年
３．雑誌名 2023 Eleventh International Symposium on Computing and Networking (CANDAR)	６．最初と最後の頁 175-181
掲載論文のDOI（デジタルオブジェクト識別子） 10.1109/CANDAR60563.2023.00031	査読の有無 有
オープンアクセス オープンアクセスではない、又はオープンアクセスが困難	国際共著 －

１．著者名 Masatoshi Kawai, Akihiro Ida, Toshihiro Hanawa, Tetsuya Hoshino	４．巻 none
２．論文標題 Optimize Efficiency of Utilizing Systems by Dynamic Core Binding	５．発行年 2024年
３．雑誌名 HPCAsia '24 Workshops: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region Workshops	６．最初と最後の頁 77-82
掲載論文のDOI（デジタルオブジェクト識別子） 10.1145/3636480	査読の有無 有
オープンアクセス オープンアクセスではない、又はオープンアクセスが困難	国際共著 －

１．著者名 Hoshino Tetsuya、Ida Akihiro、Hanawa Toshihiro	４．巻 2022
２．論文標題 Optimizations of H-matrix-vector Multiplication for Modern Multi-core Processors	５．発行年 2022年
３．雑誌名 2022 IEEE International Conference on Cluster Computing (CLUSTER)	６．最初と最後の頁 462,472
掲載論文のDOI（デジタルオブジェクト識別子） 10.1109/CLUSTER51413.2022.00056	査読の有無 有
オープンアクセス オープンアクセスではない、又はオープンアクセスが困難	国際共著 －

〔学会発表〕　計5件（うち招待講演　1件／うち国際学会　0件）

１．発表者名 百武尚輝，星野哲也，小澤　創，伊田明弘，安藤亮輔，河合直聡，永井　亨，片桐孝洋
２．発表標題 OpenACCを用いた地震シミュレーションのGPU並列化
３．学会等名 情報処理学会全国大会
４．発表年 2024年

１．発表者名
Tetsuya Hoshino, Akihiro Ida, Toshihiro Hanawa

２．発表標題
Optimizations of H-matrix-vector Multiplication for Modern Multi-core Processors

３．学会等名
Japan Geoscience Union Meeting 2023（招待講演）

４．発表年
2023年

１．発表者名
Tetsuya Hoshino, Akihiro Ida, Toshihiro Hanawa

２．発表標題
Optimizations of H-matrix-vector Multiplication for Modern Multi-core Processors

３．学会等名
ICIAM

４．発表年
2023年

１．発表者名
星野 哲也 , 河合 直聡 , 伊田 明弘 , 塙 敏博 , 片桐 孝洋

２．発表標題
HPCカーネルベンチマークによるSapphire Rapids HBMの性能評価

３．学会等名
研究報告ハイパフォーマンスコンピューティング（HPC）

４．発表年
2024年

１．発表者名
満田 晴紀, 星野 哲也, 望月 祐志, 坂倉 耕太, 片桐 孝洋, 大島 聡史, 永井 亨, 河合 直聡

２．発表標題
分子軌道計算プログラムの性能評価と自動チューニング適用の検討

３．学会等名
研究報告ハイパフォーマンスコンピューティング（HPC）

４．発表年
2023年

〔図書〕　計0件

〔産業財産権〕

〔その他〕

―

６．研究組織

	氏名 （ローマ字氏名） （研究者番号）	所属研究機関・部局・職 （機関番号）	備考

７．科研費を使用して開催した国際研究集会

〔国際研究集会〕　計0件

８．本研究に関連して実施した国際共同研究の実施状況

共同研究相手国	相手方研究機関