jh230056

近代的メニーコアシステムにおける性能モデリング手法

星野哲也（名古屋大学）

概要

複雑化する昨今の計算環境において，アプリケーションの性能に影響を及ぼす指標は多岐に渡る．ピーク性能やメモリ性能はもちろん，コア数，キャッシュの階層・速度・サイズ，ベクトル長と命令レイテンシ，コア・ノード間の通信レイテンシなどである．このように複雑化する計算環境において性能モデリングは，アプリケーションの性能理解や，自動最適化のための要素技術として非常に重要である．本課題の目的は， 幅広く実用的・先端的なアプリケーションを用いた，様々なアーキテクチャにおける性能モデルを構築することで，それに基づく自動最適化手法を構築することである．本年度の課題においては，Intel Sapphire Rapids をはじめとした，最新のプロセッサを用いたマイクロベンチマークによる性能モデルの構築，ステンシル計算・混合精度演算・階層型行列法など，近年盛んに実施されている最先端の手法を対象とし，自動最適化に向けての手動最適化，性能評価，性能モデル化，半自動最適化などを行なった．

1 共同研究に関する情報

1.1 共同研究を実施した拠点名

- 東北大学 サイバーサイエンスセンター
- 東京大学 情報基盤センター
- 名古屋大学 情報基盤センター
- 京都大学 学術情報メディアセンター

1.2 課題分野

- 大規模計算科学課題分野

1.3 共同研究分野 (HPCI 資源利用課題のみ)

- 超大規模数値計算系応用分野

1.4 参加研究者の役割分担

本研究課題では，主に以下の 6 点についての研究を実施した．

a) マイクロベンチマークによる近代的プロセッサの詳細性能モデリング
b) 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
c) 低精度演算を含む非線形ソルバの性能モデリング
d) 階層型行列法を対象とした性能モデリングによる性能理解
e) FMO プログラムの二電子積分ルーチンの性能モデリング
f) 性能モデルの自動最適化への応用手法の検討

参加研究者の役割分担は以下である．

- 星野哲也 (課題代表者, 名大): 研究統括及び研究 b の主導，a∼d の性能モデリング
- 塙敏博 (東大): b のベンチマーク実施
- 河合直聡 (名大): c の評価及び高性能化
- 伊田明弘 (JAMSTEC): d の階層型行列演算の評価及び高性能化

- 片桐孝洋 (名大): f の実施，c・e に関するアドバイス
- 満田晴紀 (名大，修士学生): e の高性能化及び評価

2 研究の目的と意義

計算機が複雑化する中で，アプリケーションの最適化において重要である性能モデリングについて，それ自体をテーマとしての研究は我が国ではあまり行われていない．マイクロベンチマークのレベルではなく，幅広くより実用的・先端的なアプリケーションを用いた，様々なアーキテクチャにおける性能モデリングが求められている．本課題では，アプリケーション最適化，自動チューニング， ハードウェアやスパコンの設計に詳しい計算機科学の専門家，非線形ソルバや階層型行列法に詳しい計算科学分野の専門家による共同研究を推進し，実アプリケーションに近い計算カーネルと最新のアーキテクチャを対象とした実用的な性能モデリング手法の開発を目的とする．

性能モデリングには主に 3 つの意義:(a) アプリケーション性能の理解，(b) 自動最適化の要素技術，(c) ハードウェアの設計，があると考える．特に本課題では，実用的なアプリケーションを高速化する際に重要となる (a) 及び (b) に着目する．

3 当拠点公募型研究として実施した意義

本研究の目指す実用的な性能モデリングを行うためには，様々な CPU，GPU，メモリ，通信装置，ノード構成からなるシステムにおいて検証を実施する必要性がある．本課題で利用を申請する Camphor 3 （京大），Wisteria-Aquarius（東大），Oakbridge-CX（東大），不老 Type I - II（名大），AOBA-B（東北大）は Intel Xeon CPU の 3 世代（Sapphire Rapids, IceLake, CascadeLake），NVIDIA GPU の 2 世代（A100, V100），A64FX，AMD EPYC を搭載しており，このような多様な計算機資源を提供する本公募型共同研究にて本研究を実施した意義は大きい．

4 前年度までに得られた研究成果の概要

新規課題であるため該当しない．

5 今年度の研究成果の詳細

1.4 に示した項目について，それぞれ報告する．

5.1 マイクロベンチマークによる近代的プロセッサの詳細性能モデリング

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．下記の成果については，[7] にて発表済みであるため，詳細についてはそちらを確認していただきたい．

5.1.1 stream ベンチマーク

まず，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 1 に示す．実験に用いた Camphor システムの Sapphire Rapids のコア数は 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．またコンパイラオプションとして，`-qopt-streaming-stores` がある，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．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` の設定を変更して実験した．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．1 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により，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 2 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 2 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．

5.1.2 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 3 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

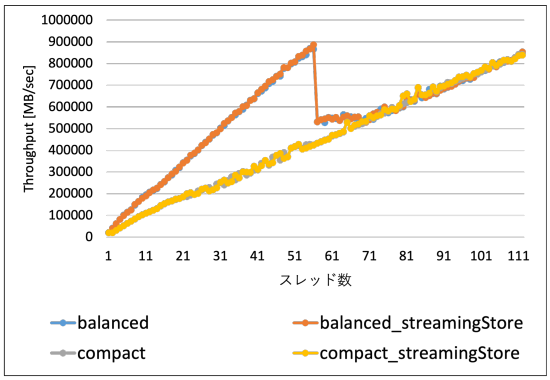

図 1　Sapphire Rapids における Stream Triad.

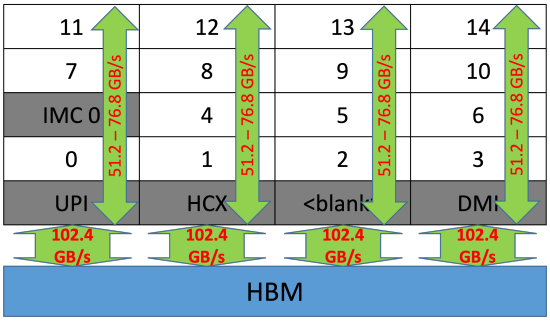

図 2　Sapphire Rapids のコア構成（1 numa 分）と帯域幅

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 4 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 4 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．

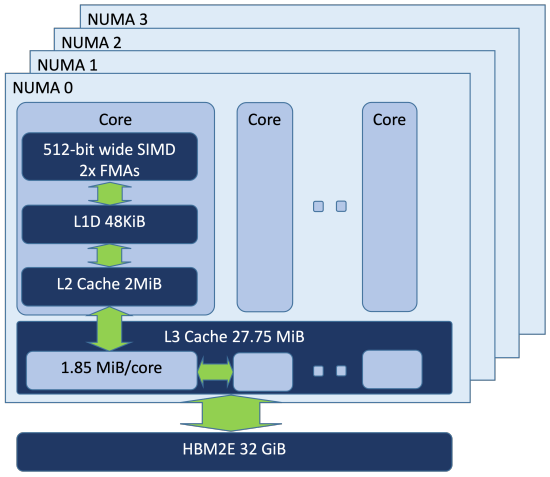

図 3　Sapphire Rapids のメモリ階層．

Sapphire Rapids 及び A64FX における結果をそれぞれ図 5 及び図 6 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 2 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．

```
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 4　$A = A + B$ を繰り返す，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．変数 id を変更することで，スレッドローカルなアクセスとスレッドを跨ぐアクセスを制御する．

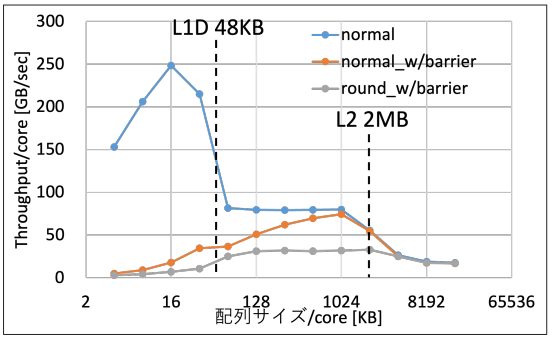

図 5　図 4 を Sapphire Rapids で評価した結果．

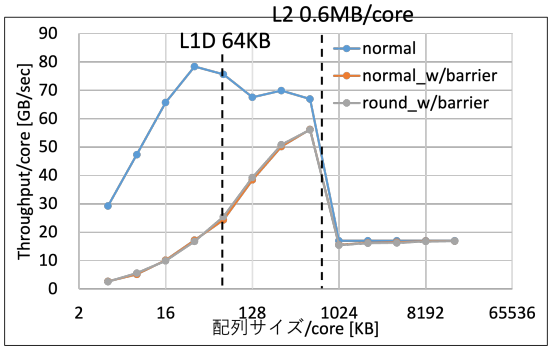

図 6　図 4 を A64FX で評価した結果．

5.1.3　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 7 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 7 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 8 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．

A64FX のように命令レイテンシの大きいプ

```
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 7　$c = c + a * b$ を繰り返す，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．

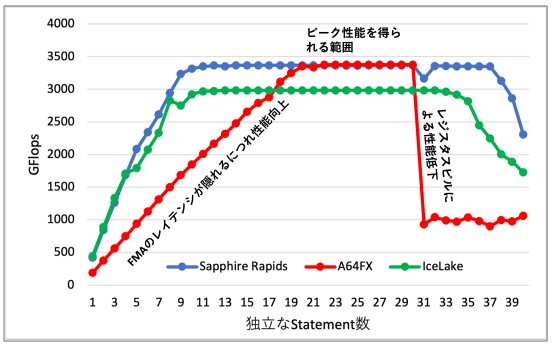

図 8　図 7 を Sapphire Rapids, A64FX, Intel Xeon IceLake で評価した結果．

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．

5.2　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 9）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．時空間ブロッキングは，依存性があるために単純には並列化できない時間ループをブロックに区切り，計算の依存性を解決しながら，ブロック内の計算をキャッシュ内で完結することで，メモリへのアクセス頻度を下げ，高速化する手法である．この手法は，計算の依存関係を解消するために冗長な計算やコア間の通信を行う必要があるため，メモリ性能，キャッシュ性能，コア間の通信性能，コア性能など，実に様々なパラメータが性能に寄与しており，性能モデル化が困難な題材の一つである．

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．各最適化の詳細については [7] を参照していただきたい．

- BASE: 9 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．
- Ydim: 9 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 10 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，5.1.3 節で説明した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，5.1.1 や 5.1.2 で説明した通り，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

```
 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 9　7 点ステンシルのカーネル

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．

5.3　低精度演算を含む非線形ソルバの性能モデリング

近年では倍精度から単精度への変更など，演算の低精度化による高速化手法について盛んに研究されている．単純に倍精度から単精度や半精度へと全ての演算を置き換えるのであれば，コードの書き換えは単純であるが，シミュレーションの精度が不足することが考えられる．性能と精度の両立を狙い，コードの一部分だけを

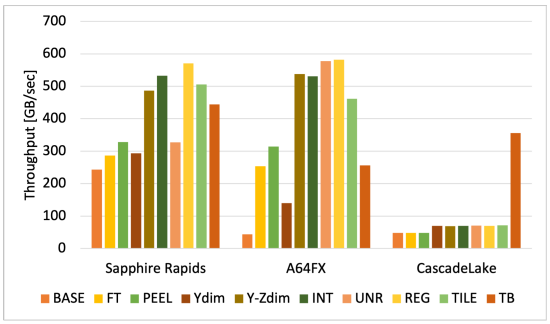

図 10　7 点ステンシルに順次最適化を適用した結果.

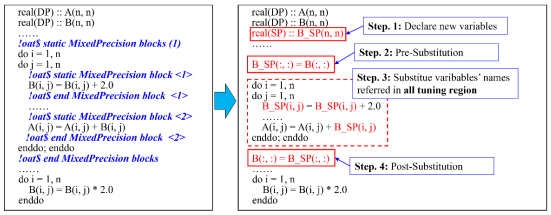

図 11　提案の MixedPrecision blocks 及び block 指示文の概要．Block 指示文中に出現する変数を全て抽出し，新たに宣言した低精度の変数と置き換える.Blocks 指示文は block 指示文の外側に宣言し，blocks 指示文が宣言された場所で変数の変換が行われる．

低精度化する手法が考えられるが，コードのどの部分を変更すればどの程度のシミュレーション精度と計算性能が得られるのかは明らかではなく，また人手によりしらみ潰しにコード変更を適用するのは現実的ではない．

本課題では，低精度化の部分適用の自動化に向け，自動チューニング言語 pp-OpenAT の指示文による，半自動の部分低精度化適用手法を開発し，評価を行なった [1]．提案の指示文の概要を 11 に示す．発表 [1] では，大気海洋シミュレーション NICAM の物理過程を 36 のブロックに区分して本指示文を適用し，ブロックの低精度化の効果をしらみ潰しに調査した．結果として，低精度化の部分適用によって性能向上が得られるブロックと，逆に性能が低下するブロックがあることが判明した．性能低下の要因は，精度変換によるオーバーヘッドや，倍精度と単精度の変数同士の演算では倍精度側にキャストされる都合上性能向上が得られないことなどが影響していることがわかった．

今後は変換によって得られる速度向上を性能モデル化し，pp-OpenAT の自動チューニング機構に組み込むことが目標である．

5.4　階層型行列法を対象とした性能モデリングによる性能理解

シミュレーションにおいて出現する密行列を低ランク行列で近似し高速化を目指す手法に関しても盛んに研究されている．例えば境界要素法を用いた地震シミュレーションでは，係数行列として現れる密行列を用いた密行列・ベクトル積に密行列・ベクトル積に性能が律速されるため，低ランク近似行列である $\mathcal{H}$-行列や格子 $\mathcal{H}$-行列を用いることによる高速化が試みられている．

しかしながら，$\mathcal{H}$-行列が持つ構造の複雑性から，スレッド並列手法やプロセス間の負荷分散に課題が生じる．格子 $\mathcal{H}$-行列法は $\mathcal{H}$-行列法の複雑性を緩和する手法であるが，複雑性を緩和する（格子サイズを小さくする）程に密行列に近づき，高速化の恩恵は得られにくくなる．自動による最適パラメータの抽出のためには性能モデリングが必要であり，そのためには実装の最適化が不足していた．特に，GPU クラスタ向けの実装及び最適化は十分に行われたこなかったため，本研究課題では，格子 $\mathcal{H}$-行列を用いた地震シミュレーションプログラムである HBI のマルチ GPU 並列化を行った．

発表 [6] では，GPU 向けの単体性能のチュー

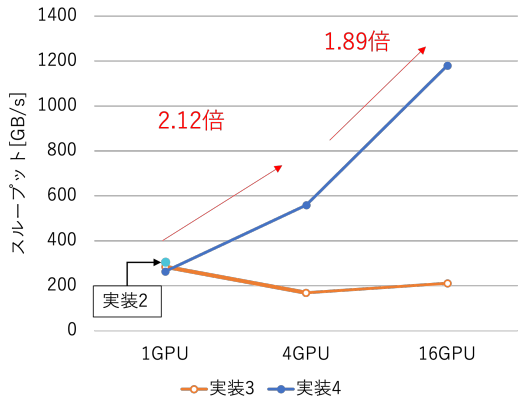

図 12　素朴なマルチ GPU 化（実装 3）と提案手法（実装 4）の比較.

ニング手法及び，マルチ GPU 向けの最適化手法の提案を行なった．図 12 は，格子 $\mathcal{H}$-行列・ベクトル積の素朴なマルチ GPU 実装と，格子をある程度ひとまとめにして GPU に処理させる手法を比較したものである．素朴な手法では台数効果による性能向上が得られなかったが，最適化によって性能向上が得られている．処理をまとめる単位は現状決め打ちであり，最適点の探索は今後の課題である．また自動最適化に向けて，性能モデリングを進めていく．

5.5　FMO プログラムの二電子積分ルーチンの性能モデリング

タンパク質などの生体高分子における分子軌道計算のシミュレーションは，高分子ダイナミクスの理解と，その理解に基づく創薬の面から応用が強く期待され，例えば新型コロナウイルスのスパイクタンパク質や，メインプロテアーゼにおける分子同士の相互作用エネルギーの計算に活用されてきた．分子軌道計算では，一般にカットオフと呼ばれる手法により，分子間の距離が遠く相互作用を無視できる場合に計算を省略することで高速化を図る．しかしカットオフは分子軌道計算プログラムの最内ループ内で if 文として実装されるため，プログラムのベクトル化効率に悪影響を与える．加えてカットオフにより計算が省略される頻度は計算対象によって異なるため，適切な最適化戦略が異なる可能性がある．そこで自動最適化に向け，本研究課題では分子軌道計算シミュレーションソフトウェアである ABINIT-MP の主要ルーチンである，二電子積分生成を対象として，最適化及び評価を行なった．

発表 [5] では，カットオフを考慮した効率的なベクトル並列化手法や，ロードバランシング手法を提案した．

5.6　性能モデルの自動最適化への応用手法の検討

本課題の最終目標は，性能モデルを用いたプログラムの自動最適化である．発表 [8] では，説明可能 AI（XAI）SHAP による機械学習モデルの構築及び解釈を試みた．不完全コレスキー分解前処理共役勾配法の係数行列を画像化し，最大 fill-in レベルや閾値をパラメータとし，モデルの構築や性能推定を行なった．

6　今年度の進捗状況と今後の展望

本研究課題では，アプリケーションの性能理解や自動最適化に資する性能モデルの構築を目的とし，マイクロベンチマークによる評価や，各種実アプリケーションカーネルの最適化・評価を行なってきた．

性能モデルを構築する上で最適化や評価は欠かせないものであり，この点に関しては十分な進展があったといえる．また性能理解を促進するために作成したマイクロベンチマークは，特に Sapphire Rapids におけるアプリケーション性能を分析する上で非常に役に立った．一方で，自動最適化に資する，各種カーネルの性能モデル化には至っておらず，今後も自動最適化に向けて研究を進めていく予定である．

7　研究業績一覧（発表予定も含む）

国際会議プロシーディングス　（査読あり）

[1] Xuanzhengbo Ren, Masatoshi Kawai, Tetsuya Hoshino, Takahiro Katagiri, and Toru Nagai. Auto-tuning Mixed-precision Computation by Specifying Multiple Regions. 2023 Eleventh International Symposium on Computing and Networking (CANDAR)

[2] Masatoshi Kawai, Akihiro Ida, Toshihiro Hanawa, and Tetsuya Hoshino. Optimize Efficiency of Utilizing Systems by Dynamic Core Binding. HPCAsia '24 Workshops: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region Workshops

国際会議発表　（査読なし）

[3] Tetsuya Hoshino, Akihiro Ida, and Toshihiro Hanawa, Optimizations of $\mathcal{H}$-matrix-vector Multiplication for Modern Multi-core Processors, Japan Geoscience Union Meeting 2023

[4] Tetsuya Hoshino, Akihiro Ida, and Toshihiro Hanawa, Optimizations of $\mathcal{H}$-matrix-vector Multiplication for Modern Multi-core Processors, International Council for Industrial and Applied Mathematics

国内会議発表　（査読なし）

[5] 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：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, 研究報告ハイパフォーマンスコンピューティング（HPC）, 2023-HPC-190

[6] 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：OpenACC を用いた地震シミュレーションの GPU 並列化，情報処理学会全国大会 2024

[7] 星野哲也, 河合直聡, 伊田明弘, 塙敏博, 片桐孝洋:HPC カーネルベンチマークによる Sapphire Rapids HBM の性能評価, 研究報告ハイパフォーマンスコンピューティング（HPC）, 2024-HPC-193

[8] 中谷崇真，河合直聡，片桐孝洋，星野哲也，永井亨：ICTCG 法の実行時間予測モデルに対する説明可能な AI の適用, 情報処理学会全国大会 2024