VII. 高性能計算システム研究部門

1. メンバー

教授　　朴　泰祐，高橋　大介，建部　修見，額田　彰，塙　敏博（客員教授，東京大学）

助教　　多田野　寛人，小林　諒平，藤田　典久

研究員　　平賀　弘平

学生　　大学院生　13 名、学類生　7 名

学内共同研究員

安永　守利，櫻井　鉄也，山口　佳樹，今倉　暁

（以上，システム情報系）

学外共同研究員

小柳　義夫（RIST），石川　裕（理化学研究所），

松岡　聡（理化学研究所），中島　浩（京都大学），

天野　英晴（慶應義塾大学），後藤　仁志（豊橋技術科学大学），

関口　智嗣（産業技術総合研究所），中尾　昌広（理化学研究所），

佐野　健太郎（理化学研究所），川島　英之（慶應義塾大学），

田中　昌宏（慶應義塾大学）

2. 概要

本研究部門では，高性能計算システムアーキテクチャ，並列プログラミング環境，GPU 利用技術，FPGA 利用技術，並列数値処理の高速化研究，分散システムソフトウェア，ストレージシステム等の研究を行っている。

今年度の研究としては以下のテーマを主に行った。

- GPU 及び FPGA の協調計算に関する研究
- GPU による都市気象コードの高速化（地球環境研究部門との共同研究）
- メニーコアプロセッサにおける数値計算ライブラリの開発及び性能評価
- 機械学習，ビッグデータサイエンスにおけるストレージ性能の向上
- Gfarm ファイルシステムの研究開発
- GPU におけるデバイス間接続ネットワークの有効活用に関する研究
- 鞍点型連立一次方程式に対する階層並列型数値解法の並列実装と性能評価，及び 3 次元画像再構成問題への応用
- 統一的なプログラミングモデルによる GPU-FPGA 連携の実現のための実践的検討
- FPGA 向け実アプリケーションの開発に資するライブラリの開発

3. 研究成果

[1] OpenACC による GPU・FPGA 協調プログラミング（朴，小林，藤田）

OpenACC は近年注目されている GPU を中心とした演算加速装置のプログラミングを, 汎用 CPU における OpenMP のように, 逐次プログラムをベースに演算加速集中部分に directive（指示文）を挿入することでコンパイラが演算加速デバイス用のカーネルコードを生成するようにし，incremental にプログラムを高速化可能な言語フレームワークである。現状では，商用 OpenACC コンパイラは主として NVIDIA 社製 GPU 向けで，FPGA を対象とするものは存在しない。このため，GPU 用と FPGA 用，それぞれ異なったバックエンドコンパイラを用いてプログラミングを行っている。

我々は Cygnus をプラットフォームとする多重複合型演算加速スーパーコンピューティングの研究の一環として，複数の性質の異なる演算加速装置，すなわち GPU と FPGA を統一的にプログラミング可能なシステムの開発を続けており, この言語フレームワークを CAMP (Cooperative Accelerating by Multi-device Programming)と呼んでいる。現在の CAMP のターゲットは OpenACC で GPU と FPGA の両アクセラレータを単一のプログラム中でアクセスできるようにすることで， 1 つの OpenACC ベースのソースコード中でプログラマが directive によってどの部分をどのアクセラレータにオフロードするかを指定できるようにする。FPGA 用の商用の OpenACC コンパイラは現状では存在しないため，米国 ORNL (Oak Ridge National Laboratory)の FTG (Future Technology Group)が開発を進めている研究用コンパイラである OpenARC を用いる。同グループとの国際共同研究により OpenACC 向けの CAMP を実現するトランスレータである MHOAT (Multi-Hybrid OpenACC Translater)を開発中である。図 1 に MHOAT の処理の概念図を示す。

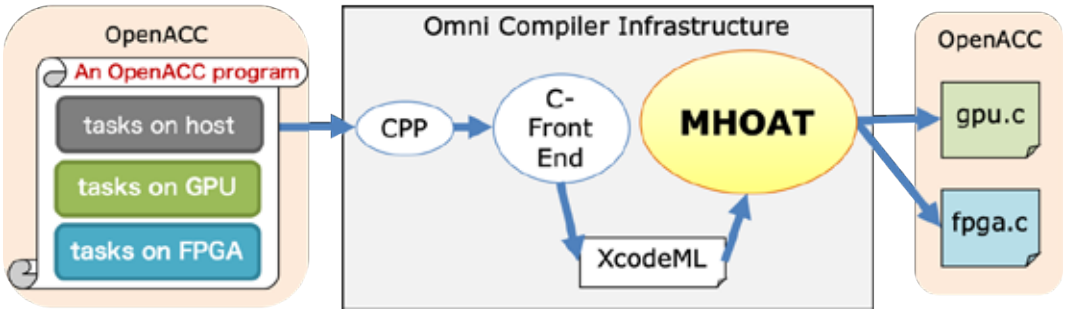


図 1　MHOAT の処理フロー

令和 3 年度の研究では，本センターの宇宙物理学研究部門・梅村教授のグループと共同で開発している宇宙初期天体シミュレーションコード ARGOT を MHOAT のターゲットコー

ドとしてコンパイルできるように MHOAT の改良を行った。これまで MHOAT は簡単なサンプルコードを対象に実装と性能評価を行ってきたが，ARGOT のような実アプリケーションコードを対象とした実装実験を行うことで同システムの有用性を示した。

ARGOT コードに FPGA または GPU への計算オフローディングを行うが，従来の研究により，同コードの計算中核部分を成す 2 種類の演算である ARGOT 法と ART 法について，それぞれ GPU 及び FPGA で機能分散的に使用デバイスを分けることで性能が大幅に向上することは昨年度までの研究で明らかになっている。今年度はいよいよ ARGOT コード全体を MHOAT でコンパイルし，これに成功した。この過程において，ORNL の OpenARC コンパイラの細かい不整合について開発者と議論し対応してもらった。さらに，元コードの一部については FPGA 実行に適合するよう，修正を行った。本来であれば，可能な限りコードの本体部分の修正は行うべきではないが, 現在の実装上の制限のワークアラウンドとして行っている。

現状では，同コードのコンパイルが可能となり，計算結果は CPU 版及び GPU 版と比べ，数%の誤差が生じることがわかった。この誤差が FPGA 上における数値計算誤差（演算の一部は厳密に IEEE 浮動小数点処理に準拠していない）によるものか，MHOAT の何らかのバグによるものかは現在検証中である。また，MHOAT の開発において高位言語からの FPGA コンパイルは必須であるが，Cygnus に搭載されている Intel Stratix10 のような高性能向け大規模 FPGA に ART 法のような実コードを実装する場合，コンパイル時間が 10 時間以上かかってしまう。FPGA の最終的な回路イメージ（Bit Stream）の生成は論理合成だけでなく，チップ内でのワイヤリングやフロアプランの最適化を行う必要があり, これはコンパイラによってヒューリスティックに行われるため, 膨大な時間がかかることはこの研究分野では常識となっている。しかし，MHOAT のような複雑なシステムの開発では研究の turn around time があまりに長くなってしまい進捗効率の大きな妨げとなる。そこで，Intel SDK に包含されているエミュレータモードを活用する。エミュレータモードは FPGA 上の動作をホスト CPU 上で仮想的に実行するもので，実際の FPGA へのオフローディングは行われない。しかし，CPU 上でのソフトウェア処理のため，Bit Stream 生成の手間が省け，コンパイル時間は数分程度と大幅に短縮される。ただし，あくまでエミュレートのため，機能検証はできるがタイミング検証はできない。

我々は MHOAT の出力としてバックエンドコンパイラに渡される OpenACC コードに対し，GPU オフロード部分は OpenACC 対応した NVIDIA HPC コンパイラにそのまま渡し，FPGA オフロード部分を OpenARC でコンパイルする際にこのエミュレーションモードでコンパイルするという処理を組み合わせることに成功した。これにより ARGOT コードが MHOAT で処理できることが実証できたが，Bit Stream による FPGA 上での検証と性能評価は今後の喫緊の課題となっている。

[2] GPU による都市気象シミュレーションの高速化（朴，日下（地球環境研究部門））

本センターの地球環境研究部門・日下博之教授のグループとの共同研究により，彼らの CPU 版 City-LES コードを GPU によって高速化する研究を継続している。令和 3 年度をこの共同研究の最終年度と位置づけ，これまでの集大成としての GPU 化研究を行った。

City-LES は元々，FORTRAN で記述された MPI 並列コードであり，以前の我々との共同研究でこれを OpenMP スレッド並列化したものが使われている。本研究はこれを OpenACC で書き換え，GPU へのオフローディングを行うことで処理性能を飛躍的に向上させようとする試みである。昨年度までの研究で，コードの大部分は OpenACC 化され，その部分部分においては 10 倍以上の高速化が達成されていたが，計算時間としては極めて小さい複数の関数が CPU 上に残っていた。しかし，少しでも CPU 上に関数が残っていると，GPU 化された関数とのデータ参照において両デバイス間通信が発生し，この部分が CPU 上の関数の処理時間を上回ってしまい，Strong Scaling における並列処理性能の妨げとなっていた。また，コード中でポアソン方程式の求解を行うが，この部分は解の収束性を高めるために Multi-Grid 法で実装されている。これまでこの部分の GPU 化は処理の複雑さから保留されていたが，今年度の研究ではこちらも GPU 化し，結果としてコード全体の完全 GPU 化を実現した。

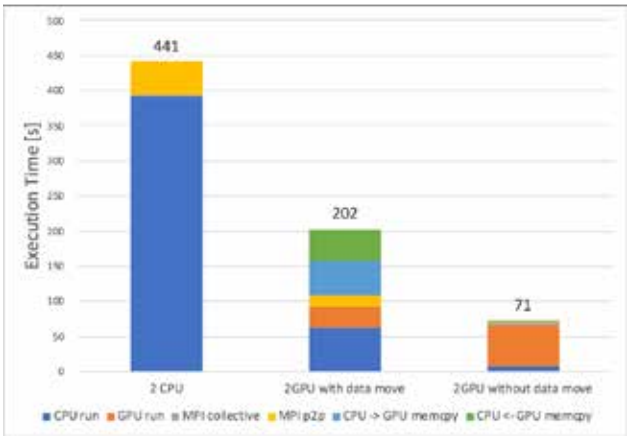


図 2　City-LES の 2CPU 及び 2GPU における性能向上とデータ転送オーバヘッド削減による実行時間短縮の様子

図 2 に Cygnus 単体ノードにおける CPU 版から GPU 版，さらにそこから CPU・GPU 間データ転送が除かれた場合の実行時間短縮の例を示す。全処理を GPU 化することで，大幅な処理時間の短縮が図られていることがわかる。実行時間の短い微小関数は並列性が低いことが多いため，場合によっては GPU オフローディングにより CPU よりも実行時間が長くなることがある。しかし，ここに示すようにデータ転送時間のオーバヘッドの方が多いことで，結果的に総合性能は向上する。

今年度の研究では，さらに GPU 上のメモリ管理についても最適化を行った。これまで，GPU 版の City-LES ではメモリ確保のオーバヘッドを削減するために，一度確保したデータを GPU デバイス上に定常的に残していたが，これは我々が期待していた効果を与えておらず，サブルーチンの呼び出し毎にメモリが動的に確保・解放されていることがわかった（図 3）。この部分を改良し，一度確保した配列を使い回すようにコードを変更した結果，当該部分を含む処理が Cygnus の 4 ノード実行で約 30%まで短縮された（図 4）。

```
subroutine acc_advection ( …略… )
 …略…
 real(8), dimension(ims_vc:ime_vc,jms_vc:jme_vc,kms:kme) :: adv_au, &
                                                            adv_av, adv_aw
 !$acc declare device_resident(adv_au, adv_av, adv_aw)
…
```

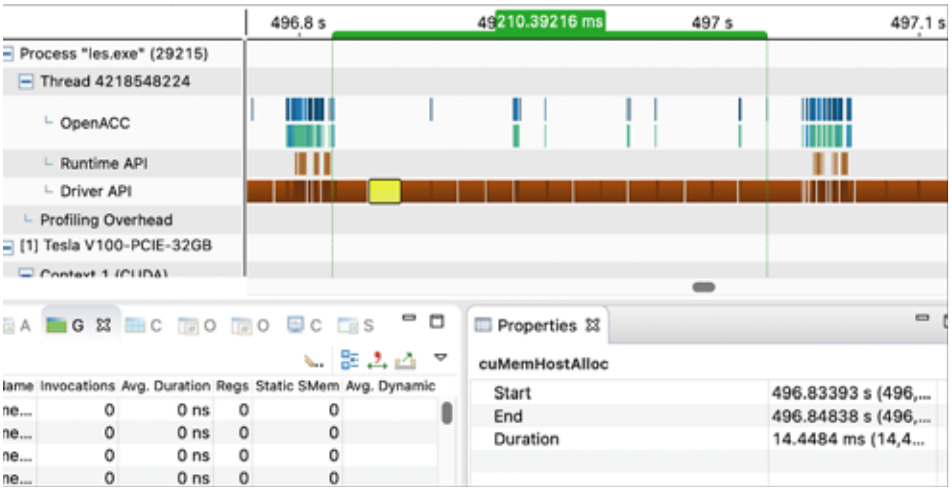


図 3 GPU 上でメモリの静的確保が動的アロケート/フリーを引き起こしてしまう様子

これらの改善により，総合的な GPU 版の性能は CPU 版に比べ最大で 22.5 倍の高速化を達成した。両者の性能差は利用するノード数に依存するが，Strong Scaling においては Cygnus の 4 ノード実行時（16 GPU 対 8 CPU）が最も速度向上が顕著であった。しかし，並列化効率が低下する 32 ノード実行時（128GPU 対 64CPU）でも性能差は 11.2 倍で，GPU 版の優位性が証明された。両バージョンの性能比較と，GPU 版の Strong Scaling 並列性能について図 5 に示す。

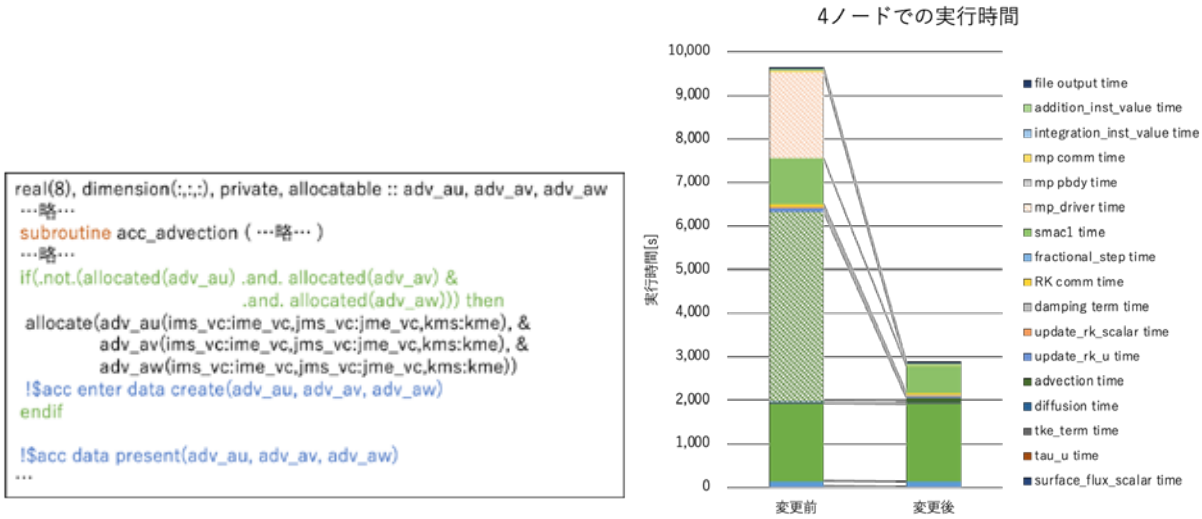


図 4　メモリ管理方法を変更し配列を使い回すことによる速度向上

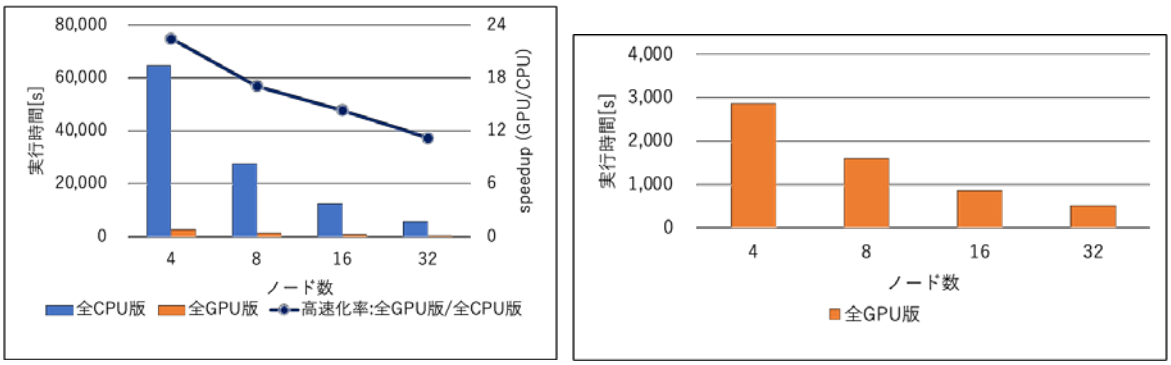


図 5　Cygnus（4 ノード～32 ノード）上での Strong Scaling 性能：CPU 版対 GPU 版の速度差と GPU 版の並列化効率（左）と GPU 版の実行時間（右）

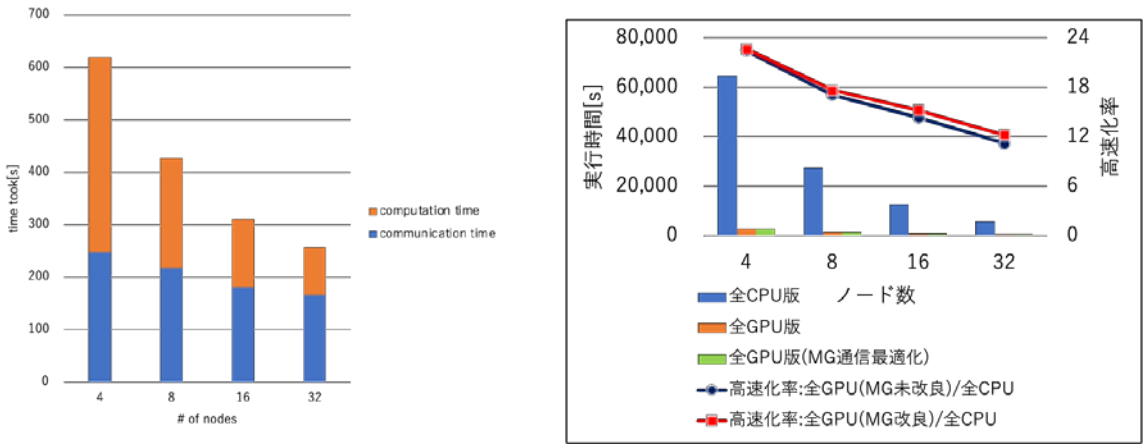


図 6　GPU 版の Strong Scaling 実行における並列通信オーバヘッド（左）と Multi-Grid 法の通信アルゴリズム改善による性能向上（右）

さらに，最終的な性能チューニングとして，ポアソン方程式の求解における Multi-Grid 法処理の GPU 化を行った。これまでの実装では，図 6（左）に示すように，Strong Scaling では通信処理が明らかなボトルネックになっていた。主要因が Multi-Grid 法の GPU 化における通信時間であることが判明し，いくつかのアルゴリズム改良を行ったが，劇的な性能向上には至らなかったが，最も並列処理効率が悪かった 32 ノードの場合，元々11.2 倍だった CPU 版との性能差が 12.2 倍に向上した。

完全 GPU 化と性能チューニングにより，例えば東京駅周辺の気象モデルである TOKYO2020 の Cygnus 上の 32 ノード実行について，従来の CPU 版では 154 時間かかったジョブが 8.7 時間に短縮される。Cygnus の運用規定では 1 ジョブの最大連続実行時間は 24 時間に制限されている。これまではジョブを 24 時間未満で打ち切り，暫定結果をファイル出力し，次の実行でこれを読み戻す処理を続けていたが，ジョブ実行が 24 時間未満となり，そのプロセスが不要となった。スーパーコンピュータの利用効率としてもこの差は極めて大きい。これらの研究成果は国際会議 NPC2021 及び情報処理学会第 183 回 HPC 研究会にて発表した。

本共同研究は令和 3 年度で終了し，完成した GPU 版 City-LES を今後は地球環境研究部門で実際の計算に用いていく予定である。

[3] 数論変換の高速化（高橋）

高速 Fourier 変換（fast Fourier transform、以下 FFT）は、現在科学技術計算において広く用いられているアルゴリズムである。FFT は複素数や実数を用いて計算されることが多いが、これらの変換は環や体でも計算できることが知られている。このような変換は、数論変換（number-theoretic transform、以下 NTT）と呼ばれる。NTT は準同型暗号、多項式乗算、多倍長精度乗算などに利用されている。本研究では、Intel Advanced Vector Extensions 512（AVX-512）命令を用いて NTT カーネルをベクトル化すると共に、OpenMP を用いて NTT を並列化して性能評価を行った。

n点 NTT は$\mathbf{F}_p = \mathbf{Z}/p\mathbf{Z}$（$p$は素数）において以下のように表すことができる。

$$y(k) = \sum_{j=0}^{n-1} x(j)\omega_n^{jk} \bmod p, \qquad 0 \le k \le n-1$$

ここで、ω_nは 1 の原始n乗根である。

n点 NTT を直接計算すると$O(n^2)$の演算回数を必要とするが、FFT と同様のアルゴリズムを適用することで、演算回数を$O(n \log n)$に削減することが可能である。Out-of-place FFT アルゴリズムとして知られる Stockham FFT アルゴリズムを基数 2 の NTT に適用したアルゴリズムを図 7 に示す。

Algorithm 1 Stockham radix-2 NTT algorithm

Input: $n = 2^q$, $X_0(j) = x(j)$, $0 \le j \le n-1$, and ω_n is the primitive n-th root of unity.

Output: $y(k) = X_q(k) = \sum_{j=0}^{n-1} x(j)\omega_n^{jk} \bmod p$, $0 \le k \le n-1$

1: $l \leftarrow n/2$
2: $m \leftarrow 1$
3: **for** t **from** 1 **to** q **do**
4: **for** j **from** 0 **to** $l-1$ **do**
5: **for** k **from** 0 **to** $m-1$ **do**
6: $c_0 \leftarrow X_{t-1}(k+jm)$
7: $c_1 \leftarrow X_{t-1}(k+jm+lm)$
8: $X_t(k+2jm) \leftarrow (c_0+c_1) \bmod p$
9: $X_t(k+2jm+m) \leftarrow \omega_n^{jm}(c_0-c_1) \bmod p$
10: **end for**
11: **end for**
12: $l \leftarrow l/2$
13: $m \leftarrow 2m$
14: **end for**

図 7　基数 2 の Stockham NTT アルゴリズム

NTT の計算では、剰余乗算が計算時間の大半を占めている。Montgomery 乗算を用いることで時間の掛かる除算を実質的に行うことなく、乗算、加減算およびシフト演算のみで剰余乗算を行えることが知られている。

図 7 のアルゴリズムの 8 行目と 9 行目には、剰余加算、剰余減算、剰余乗算が含まれている。$0 \le a, b < N$の場合の剰余加算$c = (a + b) \bmod N$は、加算$c = a + b$と、$c \ge N$の場合の条件付き減算$c - N$に置き換えることができる。このような条件付き減算は分岐を伴うが、符号なし整数値cおよびNに対する 2 の補数のラップアラウンド演算による最小値演算$\min(c, c - N)$に置き換えることにより、分岐を回避することができる。Intel AVX-512F（Foundation）命令セットでは、64 ビット符号なし整数の最小値演算に対する `vpminuq` 命令がサポートされている。

Intel AVX-512IFMA（Integer Fused Multiply-Add）命令セットは、Cannon Lake、Ice Lake、および Tiger Lake マイクロアーキテクチャでサポートされている。Intel AVX-512IFMA 命令セットは、52 ビットの符号なし整数の乗算を行い、104 ビットの中間結果の下位と上位をそれぞれ生成する `vpmadd52luq` 命令と `vpmadd52huq` 命令をサポートしている。これらの下位と上位の値は 64 ビットアキュムレータに加算される。このような演算は C 言語ではサポートされていないため、Intel AVX-512IFMA 命令を用いるには、Intel AVX-512 の組み込み関数を用いるか、インラインアセンブラを用いる必要がある。

Algorithm 2 Modified Montgomery multiplication algorithm

Input: A, B, N such that $0 \leq A, B < N$, $\beta > N$, $\gcd(\beta, N) = 1$, $\mu = -N^{-1} \bmod \beta$

Output: $C = AB\beta^{-1} \bmod N$ such that $0 \leq C < N$

1: $C \leftarrow AB \bmod \beta$
2: $q \leftarrow \mu C \bmod \beta$
3: $C \leftarrow \lfloor AB/\beta \rfloor + \lfloor qN/\beta \rfloor$
4: **if** $q \neq 0$ **then**
5: $\quad C \leftarrow C + 1$
6: **if** $C \geq N$ **then**
7: $\quad C \leftarrow C - N$
8: **return** C.

図 8　修正 Montgomery 乗算アルゴリズム

乗算の上半分を計算する命令を持つプロセッサでは、図 8 に示す修正 Montgomery 乗算アルゴリズムが効率的である。このアルゴリズムで$\beta = 2^{52}$とすると、3 行目で 52 ビット整数の加算が行われるが、vpmadd52huq 命令は 64 ビット整数の加算を行えるので、オーバーフローは発生しない。

```
__m512i _mm512_mulmod52_epu64(__m512i a, __m512i b, __m512i N, __m512i mu)
/*  Compute (a * b * 2^-52) mod N. We need mu = -N^-1 mod 2^52. */
{
  __m512i c, q, t;

  t = _mm512_madd52lo_epu64(_mm512_set1_epi64(0), a, b);
  q = _mm512_madd52lo_epu64(_mm512_set1_epi64(0), t, mu);
  t = _mm512_madd52hi_epu64(_mm512_min_epu64(q, _mm512_set1_epi64(1)),
                            q, N);
  t = _mm512_madd52hi_epu64(t, a, b);
  c = _mm512_min_epu64(t, _mm512_sub_epi64(t, N));

  return c;
}
```

図 9　Intel AVX-512 組み込み関数を用いたパックド 52 ビット整数の Montgomery 乗算

図 9 に、Intel AVX-512 組み込み関数を用いたパックド 52 ビット整数の Montgomery 乗算を示す。_mm512_set1_epi64()、_mm512_madd52lo_epu64()、_mm512_min_epu64()、_mm512_madd52hi_epu64()および_mm512_sub_epi64()は、それぞれ vpbroadcastq、vpmadd52luq、vpminuq、vpmadd52huq および vpsubq 命令に対応する。

また、six-step FFT アルゴリズムと呼ばれる手法は NTT にも適用可能であるので、OpenMP を用いて six-step NTT を並列化した。

性能評価として、以下の 4 つの実装の性能を Intel Xeon Platinum 8368（Ice Lake 2.4 GHz、38 コア、76 スレッド）を用いて比較した。

・提案する実装（AVX-512F）、modulus は 62 ビット

・提案する実装（AVX-512IFMA）、modulus は 52 ビット

・Intel HEXL 1.2.4（AVX-512DQ）、modulus は 62 ビット

・Intel HEXL 1.2.4（AVX-512IFMA）、modulus は 50 ビット

提案する実装を 1〜76 スレッドで実行した。順方向 NTT の経過時間が 1 秒を超えるまで繰り返し回数を 2 倍にし、平均経過時間を計測した。ひねり係数のテーブルは事前に用意した。Intel HEXL は並列実行をサポートしていないため、シングルスレッドで実行した。

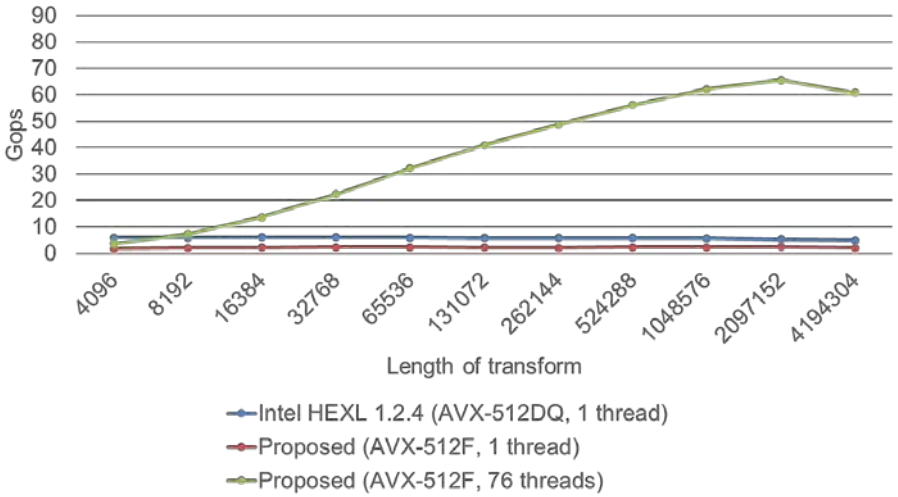


図 10　Intel AVX-512F または AVX-512DQ 命令を用いた NTT の性能（Intel Xeon Platinum 8368, 38 コア）

図 10 は、Intel AVX-512F または AVX-512DQ 命令を用いた NTT の性能を示している。図 10 に示すように、提案する実装（AVX-512F）はシングルスレッド実行において、Intel HEXL（AVX-512DQ）よりも遅いことが分かる。Six-step NTT は並列化に適しているが、3 回の行列転置が必要であり、この行列転置のオーバーヘッドが Intel HEXL より遅い理由であると考えられる。この性能評価で用いた Intel Xeon Platinum 8368 プロセッサは 57MB の L3 キャッシュを備えているため、最大で2^{21}点 NTT が L3 キャッシュに収まる。キャッシュブロッキングを用いた six-step NTT はキャッシュにデータが収まらない場合に有効であるが、Intel HEXL は2^{22}点 NTT までしか実行できなかったため、提案する実装の優位性が発揮されなかった可能性がある。

Intel HEXL（AVX-512DQ）が提案する実装（AVX-512F）よりも高速なもう一つの理由は、Intel HEXL（AVX-512DQ）が Intel AVX-512DQ（Doubleword and Quadword）命令セットでサ

ポートされている vpmullq 命令を用いて 64 ビット×64 ビット→64 ビットの整数乗算を行っているためである。提案する実装（AVX-512F）は、76 スレッドで$n \geq 2^{13}$の場合、Intel HEXL（AVX-512DQ）より高速に実行される。

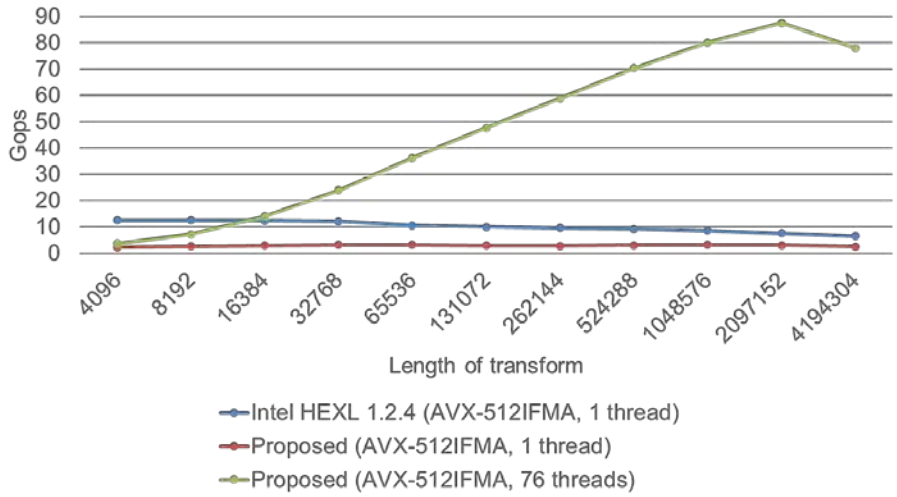


図 11　Intel AVX-512IFMA 命令を用いた NTT の性能（Intel Xeon Platinum 8368, 38 コア）

図 11 は、Intel AVX-512IFMA 命令を用いた NTT の性能を示している。図 11 に示すように、提案する実装（AVX-512IFMA）はシングルスレッド実行において、Intel HEXL（AVX-512IFMA）よりも遅いことが分かる。提案する実装（AVX-512IFMA）は、76 スレッド、$n \geq 2^{14}$で Intel HEXL（AVX-512IFMA）より高速に実行される。

図 10 と図 11 を比較すると、提案する実装（AVX-512IFMA）はシングルスレッド実行時および 76 スレッド実行時に、提案する実装（AVX-512F）よりも高速に実行された。これは、提案する実装（AVX-512IFMA）が Intel AVX-512F 命令を用いて複数の Montgomery 乗算を行うため、より少ない命令数で実行できるためである。ただし、modulus のサイズは提案する実装（AVX-512F）が 62 ビットであるのに対し、提案する実装（AVX-512IFMA）では 52 ビットに削減されている。

[4] 数学定数に対する二進 BBP 型公式の計算における Montgomery 乗算の使用（高橋）

πのような数学定数のn桁目の数字だけを計算することは、最初のn桁をすべて計算するよりも簡単ではないと広く信じられていた。ところが、1995 年に発見された BBP（Bailey–Borwein–Plouffe）型公式により、いくつかの超越数のn桁目の数字だけをさまざまな基数で計算できることが示された。BBP 型公式は数百桁以上の多倍長精度演算が不要（128 ビット程度の精度で十分）であり、容易に実装できる。またメモリをほとんど必要としないという特徴がある。

$\log 2$に対する BBP 型公式は以下の式で表される。

$$\log 2 = \sum_{k=1}^{\infty} \frac{1}{k2^k}$$

2 進数で$n+1$ビット目から始まる$\log 2$の数ビットを計算することを考える。これは$\{2^n \log 2\}$（ここで$\{x\} \equiv x - \lfloor x \rfloor$とする）を求めることと等価になることから、以下の式が得られる。

$$\{2^n \log 2\} = \left\{ \left\{ \sum_{k=1}^{n} \frac{2^{n-k}}{k} \right\} + \sum_{k=n+1}^{\infty} \frac{2^{n-k}}{k} \right\} = \left\{ \left\{ \sum_{k=1}^{n} \frac{2^{n-k} \bmod k}{k} \right\} + \sum_{k=n+1}^{\infty} \frac{2^{n-k}}{k} \right\}$$

この式の 1 番目の総和における分数の分子において、2^{n-k}を$2^{n-k} \bmod k$に置き換えることができるのは、小数部分だけを計算すればよいためである。2 番目の総和については、分数の値が機械イプシロン以下になるまで高々数十項を計算すればよい。

この BBP 型公式の計算において最も時間を要するのは、べき剰余$2^{n-k} \bmod k$である。べき剰余計算はバイナリ法と Montgomery 乗算を用いることで効率的に行えることが知られている。Montgomery 乗算を 2 進数で計算する場合、剰余計算の法（modulus）は奇数でなければならない。しかし、1 番目の総和において分数の分母kが偶数である場合、べき剰余$2^{n-k} \bmod k$の法kも偶数になるため、Montgomery 乗算を直接用いることができない。そこで、$\{2^n \log 2\}$を計算する式を以下のように変形する。

$$\{2^n \log 2\} = \left\{ \sum_{k=1}^{n-\lfloor \log_2 n \rfloor} \frac{2^{n-k-\mathrm{ntz}(k)} \bmod D(k)}{D(k)} \right\} + \left\{ \sum_{\substack{k=n-\lfloor \log_2 n \rfloor+1 \\ n-k-\mathrm{ntz}(k) \geq 0}}^{n} \frac{2^{n-k-\mathrm{ntz}(k)} \bmod D(k)}{D(k)} \right\}$$

$$+ \left\{ \sum_{\substack{k=n-\lfloor \log_2 n \rfloor+1 \\ n-k-\mathrm{ntz}(k) < 0}}^{\infty} \frac{2^{n-k-\mathrm{ntz}(k)}}{D(k)} \right\}$$

ここで、$D(k) = k/2^{\mathrm{ntz}(k)}$、$\mathrm{ntz}(k)$は$k$の最下位ビットから連続する 0 のビット数（Number of Trailing Zero、以下 NTZ）である。$D(k)$は必ず奇数になることから、Montgomery 乗算を用いてべき剰余計算を行うことができる。多くのプロセッサは NTZ を求めるビットスキャン命令を持っているので、そのようなプロセッサでは$\mathrm{ntz}(k)$を高速に求めることができる。

BBP 型公式の一般形は以下の記法で表すことができることが知られている。

$$P(s, b, m, A) = \sum_{k=0}^{\infty} \frac{1}{b^k} \sum_{j=1}^{m} \frac{a_j}{(mk+j)^s}$$

ここで、、b、およびmは整数であり、$A=(a_1,a_2,\ldots,a_m)$は整数のベクトルである。次数はsであり、基数はbである。この記法を用いると、πおよび$\log 2$に対する BBP 型公式は以下のように簡潔に表すことができる。

$$\pi = P\big(1,16,8,(4,0,0,-2,-1,-1,0,0)\big)$$

$$\log 2 = \frac{1}{2}P\big(1,2,1,(1)\big)$$

b進数で$n+1$桁目から始まる$P(s,b,m,A)$の数桁は以下のようにして計算することができる。

$$\{b^n P(s,b,m,A)\} = \left\{\sum_{j=1}^{m} a_j \left\{\left\{\sum_{k=0}^{n} \frac{b^{n-k}}{(mk+j)^s}\right\} + \sum_{k=n+1}^{\infty} \frac{b^{n-k}}{(mk+j)^s}\right\}\right\}$$

$$= \left\{\sum_{j=1}^{m} a_j \left\{\left\{\sum_{k=0}^{n} \frac{b^{n-k} \bmod (mk+j)^s}{(mk+j)^s}\right\} + \sum_{k=n+1}^{\infty} \frac{b^{n-k}}{(mk+j)^s}\right\}\right\}$$

この式は、$b=2^c$とすると以下のように変形できる。

$$\{b^n P(s,b,m,A)\}$$

$$= \left\{\sum_{j=1}^{m} a_j \left\{\left\{\sum_{k=0}^{\lfloor (cn-\lfloor \log_2 (mn+j)^s \rfloor)/c \rfloor} \frac{2^{c(n-k)-\mathrm{ntz}((mk+j)^s)} \bmod D((mk+j)^s)}{D((mk+j)^s)}\right\}\right.\right.$$

$$+ \left\{\sum_{\substack{k=\lfloor (cn-\lfloor \log_2 (mn+j)^s \rfloor)/c \rfloor+1 \\ c(n-k)-\mathrm{ntz}((mk+j)^s)\geq 0}}^{n} \frac{2^{c(n-k)-\mathrm{ntz}((mk+j)^s)} \bmod D((mk+j)^s)}{D((mk+j)^s)}\right\}$$

$$\left.\left. + \sum_{\substack{k=\lfloor (cn-\lfloor \log_2 (mn+j)^s \rfloor)/c \rfloor+1 \\ c(n-k)-\mathrm{ntz}((mk+j)^s)<0}}^{\infty} \frac{2^{c(n-k)-\mathrm{ntz}((mk+j)^s)}}{D((mk+j)^s)}\right\}\right\}$$

性能評価として、πおよび$\log 2$に対する BBP 型公式を計算する際に、Montgomery 乗算を用いた提案手法と Montgomery 乗算を用いない従来手法の実行時間を比較した。

表 1　$\boldsymbol{\pi}$の 16 進$\mathbf{10^{10}}$桁目計算の実行時間（秒）

	従来手法	提案手法
Intel Core i3-8121U	2806.30	2213.19
Intel Xeon Gold 6230	1600.20	372.36

表 1 は、Intel Core i3-8121U と Intel Xeon Gold 6230 でπの 16 進10^{10}桁目を計算するのに必要な実行時間を示している。提案手法は従来手法に比べて Intel Core i3-8121U で約 1.27 倍、Intel Xeon Gold 6230 で約 4.30 倍高速であることが分かる。提案手法が従来手法よりも高速である理由としては、提案手法がべき剰余計算を行う際に Montgomery 乗算を用いているためであると考えられる。また、Intel Core i3-8121U では、Intel Xeon Gold 6230 よりも提案手法と従来手法の性能差が小さくなっているが、これは Intel Core i3-8121U が整数除算命令に要するサイクル数が少なくなっているためである。

表 2 $\log 2$の 16 進10^{10}桁目計算の実行時間（秒）

	従来手法	提案手法
Intel Core i3-8121U	2844.10	2234.79
Intel Xeon Gold 6230	1466.95	374.63

表 2 は、Intel Core i3-8121U と Intel Xeon Gold 6230 で$\log 2$の 16 進10^{10}桁目を計算するのに必要な実行時間を示している。提案手法は従来手法に比べて Intel Core i3-8121U で約 1.27 倍、Intel Xeon Gold 6230 で約 3.92 倍高速であることが分かる。

[5] ストレージシステムの研究（建部）

スーパコンピュータの計算ノードのストレージシステムを活用したアドホック並列分散ファイルシステム CHFS の設計を行った。アドホック並列分散ファイルシステムとは、計算ノードが割り当てられている間に計算ノードのローカルストレージを用いて一時的に構成される並列分散ファイルシステムである。並列ファイルシステムにおけるストレージ性能のボトルネックの解消を目的としている。

計算ノードのストレージシステムには NVMe SSD に加え不揮発性メモリも用いられる。不揮発性メモリは NVMe SSD と異なり、バイト単位でのアクセスが可能であり、従来のブロックデバイスを想定したファイルシステム等を用いたアクセスでは性能を活用することができない。本研究においては、不揮発性データ構造を用いたデータ管理を用い、具体的にはインメモリ不揮発性キーバリューストアを活用したストレージシステムの設計を行った。複数ノードの不揮発性メモリを用いて分散キーバリューストアを構成する。通信については、スーパコンピュータの高速ネットワークを活用するため RDMA を用いる。分散キーバリューストアについては、ノードの増減時にデータ転送を最小化するためにコンシステントハッシングを用いる。

分散キーバリューストア上に並列分散ファイルシステムを構成するための設計を行った。設計にあたり、ノード数に対するスケーラビリティを阻害する要因となる集中データ構造や逐次処理を避けた。HPC においては大規模ファイルの並列アクセスにおける性能向上が必須であるため、ファイルはチャンクで分割し分散させる。既存研究では GekkoFS が最も近いが、GekkoFS は NVMe SSD を想定し、ファイルデータはファイルシステムで管理し、メタデータは RocksDB で管理している。一方、CHFS では不揮発性メモリの性能を活用するためインメモリ不揮発性キーバリューストアをベースとした設計となっている。また、GekkoFS ではデータ分散はモジュラハッシングを用い、ノードの増減については考慮していない。

CHFS を Mochi-Margo を用いて実装した。Mochi-Margo は HPC 用の RPC ライブラリであり、InfiniBand 等の高速ネットワークを活用することが可能である。

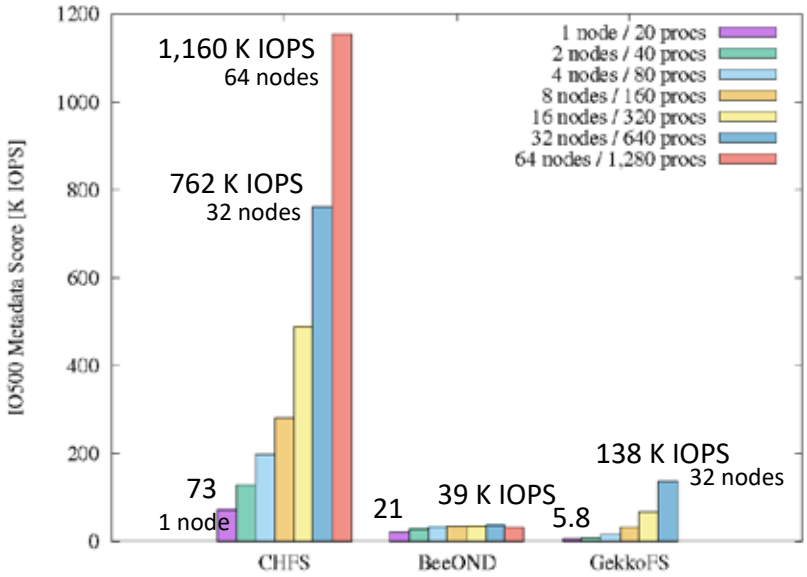


図 12　Cygnus における IO500 メタデータベンチマークの性能

図 12 に Cygnus を用いた IO500 メタデータベンチマークの CHFS の性能を示す。ノード数を 1 ノードから 64 ノードまで増やしていきメタデータ性能を計測した。IO500 メタデータベンチマークは 8 種類のメタデータアクセスパターンの性能の幾何平均である。ノード数を増加させるとメタデータ性能が向上していることがわかる。また既存システムである BeeOND と GekkoFS との性能の比較を行った。BeeOND ではノード数を増やしてもメタデータ性能はあまり向上しない。GekkoFS の性能は向上するが CHFS と比較すると性能差は 5.5 倍であった。なお、本研究成果は ACM 国際会議 HPC Asia において発表を行った。

[6] 分散ファイルシステムおよびグリッド・クラウト技術に関する研究（建部）

文部科学省が進める革新的ハイパフォーマンスコンピューティングインフラ（HPCI）のHPCI 共用ストレージ, 素粒子物理学データ共有システム JLDG のシステムソフトウェアとしても利用される Gfarm ファイルシステムの研究開発を行った。

HPCI 共用ストレージにおいて、北海道大学、東北大学から東京大学のストレージ機器に書込みを行うとたまに性能が悪くなる問題が発生した。ネットワーク、ストレージ等の調査を行った結果、東京大学内の特定のストレージ機器について問題発生頻度が高くなるものの、根本的な原因を突き止めるまでには至らなかった。そのため、Gfarm ファイルシステム側でのワークアラウンドにおいて対応を図った。ワークアラウンドでは、極端にデータ転送性能が落ちた場合に、敢えて通信を切断し、再接続して転送を行うように改修した。再接続すると転送性能が改善することがあるためである。このときの問題は、極端にデータ転送性能が落ちたことをどのように検知するかである。まず、一定時間における平均通信量を計測し、平均通信量が極端に低下したときに判定することを試みたが、平均通信量は安定しておらず、検知が難しいことが判明した。そのため、別の方法を模索した。結果、データ送信を行う send システムコールのタイムアウトで検出することが適当であることが分かった。タイムアウトの時間については設定ファイルで指定可能となるよう Gfarm ファイルシステムの改修を行った。本機能については、2021 年 4 月にソフトウェアリリースを行った。HPCI の各資源提供機関において更新後のソフトウェアを用いてファイル転送を行った結果、以前のような時々転送が遅くなる問題は発生しなくなった。

また、Society 5.0、IoT などで需要の高いオブジェクトストレージインタフェース（I/F）の対応のための研究開発を進めた。オープンソースのオブジェクトストレージサーバとしては MinIO がよく利用されている。MinIO のストレージインターフェースとして Gfarm を利用できるように設計、実装をすすめた。HPCI 共用ストレージでは myproxy を用いており、オブジェクトストレージサーバへの login 時に代理証明書を myproxy-server から取得する。サイズの大きいファイルはクライアントがファイルを分割して送付することとなるが、分割ファイルの送信、分割ファイルの結合、一時ファイルへの改名などの操作についてそれぞれ性能改善の余地があることが判明した。分割ファイルの送信については、分割したファイルを複数の通信ストリームを用いて並列転送することにより性能改善を図った。分割ファイルの結合については、分割されたファイルを結合しながら HPCI 共用ストレージに格納可能なように新しいコマンドを新設した。一時ファイルへの改名については、従来は２回行われていたがこれを１回に削減した。これらの改善を図ることによりアップロード性能が改善した。また、オブジェクトインターフェースを提供するサーバの導入を簡便にするため、コンテナ化を実施した。これにより、設定ファイルと CA の証明書、ユーザの対応表を準備してコンテナを作

成、起動することで導入が可能となった。成果は https://github.com/oss-tsukuba/gfarm-s3-minio-web で公開している。

海外の共同研究者が HPCI 共用ストレージに対して高速にデータ書込み、読込を行うため、GridFTP プラグインの整備を行った。GridFTP は標準となっている長距離高速転送プロトコルである。これにより、欧州などからの高速なデータ転送が可能となった。成果は https://github.com/oss-tsukuba/gfarm-gridftp-dsi で公開している。

GSI のサポート終了に伴う次期セキュリティに対する対応として、暗号通信の標準である TLS（Transport Layer Security）を HPCI 共用ストレージで用いるための概念設計、詳細設計を進め、基本的な実装を行った。さらに、現在 HPCI で用いている代理証明書をクライアント証明書として利用するための機能、信頼する中間認証局を指定するための機能等の実装を進めた。これにより、HPCI、JLDG において現在用いている代理証明書を利用して Gfarm ファイルシステムを安全にアクセスすることが可能となった。信頼する中間認証局を指定できるようにすることにより、信頼できない中間認証局が発行した信頼できない証明書を排除することが可能となる。

また、認証機構についての開発を行った。現在、HPCI 作業部会コアメンバー会議では認証方式は、OAuth2 が第一候補、Kerberos が第二候補となっている。本年度の開発においては、OAuth2 への対応を主としつつ、Kerberos の対応も行った。OAuth2 への対応として SASL の OAuth モジュールを用いる方法について検討を行い、試作を進めた。まず Gfarm ファイルシステムにおいて SASL を用いた認証の仕組みの設計を行い、SASL の OAuth モジュールにおいて指定した Keycloak に対しトークンの検証を依頼する仕組みについて設計を行った。また OAuth2 のトークンをコマンドラインで安全に取得､更新するための機構について検討を行い、試作を進めた。

本成果については Gfarm ワークショップにおいて発表した。

[7] マルチ GPU システム向けの Allreduce 処理の最適化（額田）

現在スパコンやクラウドの多くの計算ノードは NVIDIA GPU を搭載し、HPC 及び AI 分野の高速化に用いられている。深層学習ではその長い学習時間を短縮するため複数の GPU を用いる分散学習が盛んに行われている。よく用いられるデータ分散型においては GPU 間での Allreduce 処理が高速化のために特に重要である。

NVIDIA GPU は Pascal 世代以降で NV-Link という高速ネットワークをサポートし、GPU 間を直接、あるいはスイッチ経由で低レイテンシかつ高バンド幅で接続することが可能となった。これにより複数の GPU を利用する処理の効率が向上し、Allreduce の通信も高速化された。

ノード内の複数 GPU 間の Allreduce 処理をさらに高速化するために、GPU 間を直接接続する NV-Link に加えて CPU を経由する PCI-Express によるネットワークを併用するハイブリッドアルゴリズムを提案する。GPU 間のデータ転送に関しては現状最短経路のみを使用するような実装になっているため、CPU を経由するためには一度 CPU のメモリに明示的に転送する必要がある。CPU 経由の場合には 2 段階の転送となるため PCI-Express の双方向のバンド幅を有効活用するために複数回の転送に分割してオーバラップさせる。

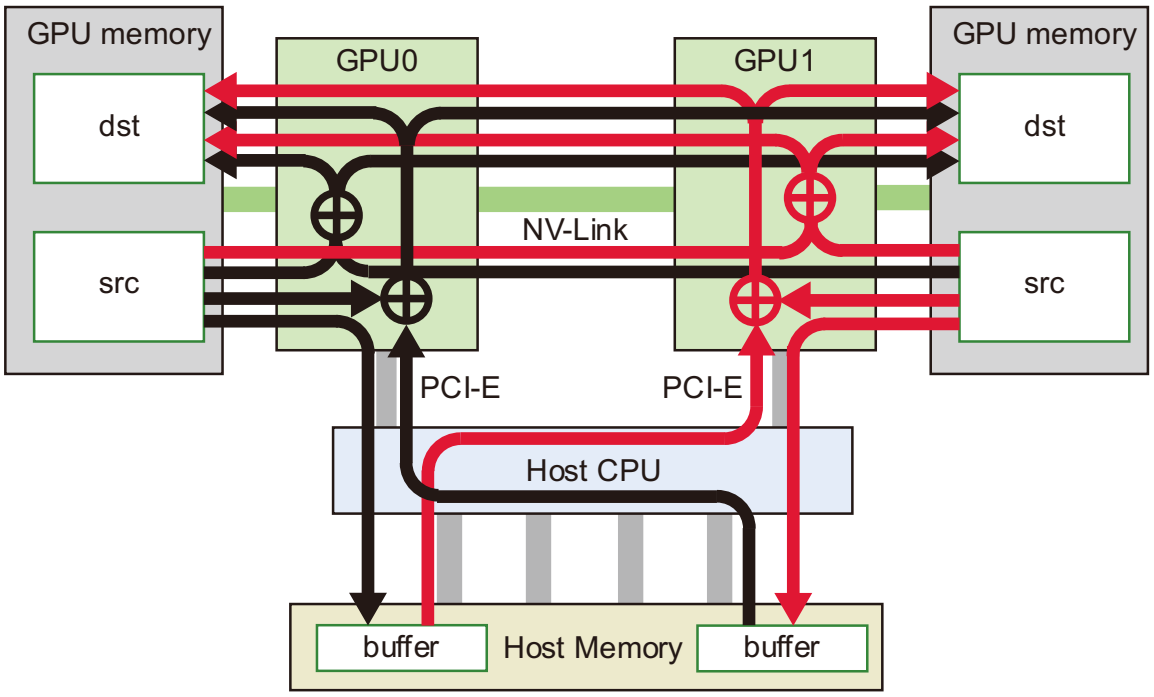


図 13　NV-Link と PCI-Express を併用するハイブリッド Allreduce アルゴリズム

Allreduce を行う配列データのサイズに合わせて最適なパラメータを選択しなければならない。NV-Link 側で処理する割合、CPU 経由通信の分割数に加えて、NV-Link 経由及び CPU 経由の２種類の Allreduce カーネルが同時に実行されるため、それらの優先度をスレッド数の設定により調整する。

ハイブリッドアルゴリズムに最適なパラメータを選択した上で、NVIDIA の提供する NCCL ライブラリ及び NV-Link だけを使用するアルゴリズムとの性能比較を行った。データサイズが小さい場合には CPU 経由の 2 段階転送の効率が低く、NV-Link だけを利用する方が高速である。一定以上のデータサイズになると CPU 経由の転送についてもバンド幅相応の転送速度が得られ、NV-Link だけの場合や NCCL よりも最大 19%の速度向上を実現している。

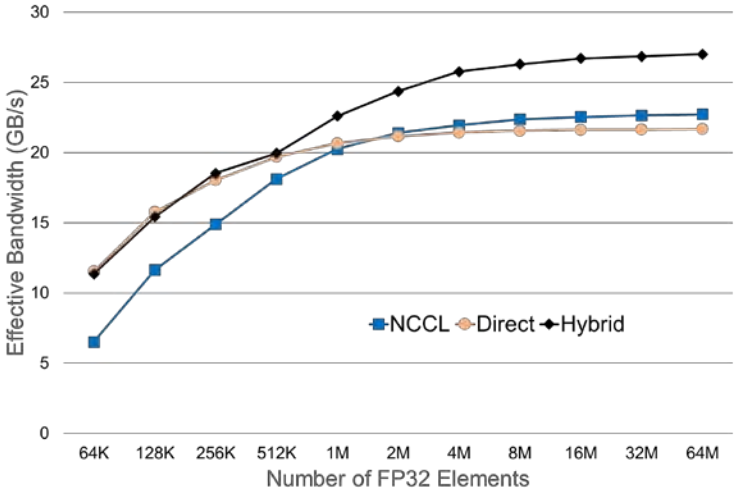


図 14 ハイブリッドアルゴリズムと既存手法との性能比較

[8] 遊休 GPU を利用するホスト・デバイス間通信の高速化（額田）

GPU 搭載スパコンの多くは複数の GPU を搭載する計算ノードを採用している。複数の GPU 及び複数の計算ノードを利用する大規模アプリケーションが実行される一方で、GPU を 1 基のみ利用するアプリケーションも依然として多いのが現状である。このように GPU を 1 基しか利用しない理由は様々で、複数 GPU を利用するプログラミングの難しさによることもあるが、並列性やクリティカルセクションなどの事情により GPU を 1 基だけ利用した方が高速に実行できる場合もある。このように計算ノードの GPU が 1 基しか利用されない場合に生じる遊休 GPU を有効活用する手法を提案する。

GPU アプリケーション自体には変更を加えない前提とすると、GPU 側での計算は引き続き GPU 1 基で実行する。そこでしばしば GPU アプリケーションの性能上のボトルネックとなるホスト・デバイス間通信の高速化に遊休 GPU を利用する。これには複数の経路を併用することによって CPU と計算に利用する GPU 間のネットワークバンド幅を向上させる必要があり、計算ノードの構成として各 GPU と CPU の間に独立した PCI-Express 等の接続があり、また GPU 間が NV-Link で接続されていることが条件となる。

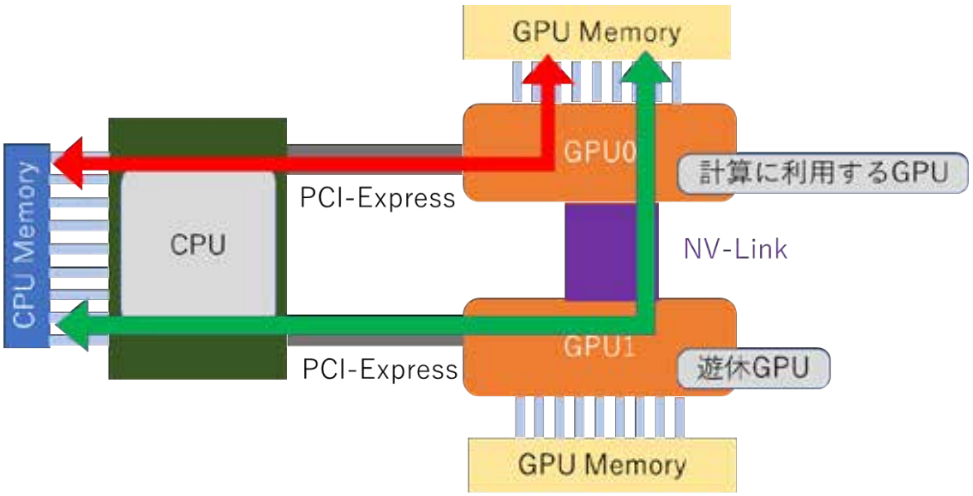


図 15 想定する複数 GPU 搭載ノードの構成

CUDA ではホスト・デバイス間転送に cudaMemcpy()などの API 関数を用いる。そこでアプリケーションがこの関数を呼び出した場合に複数のパスを併用して転送を行うように関数を上書きする方法を採用する。この方法では LD_PRELOAD 環境変数で専用ライブラリをロードするだけでアプリケーションのバイナリはそのまま用いることができる。

２つのパスがある場合に提案手法ではそれぞれのパスは半分のサイズのデータ転送を担当する。直行する GPU0 については cudaMemcpy()関数をそのまま呼び出すことで実現できるが、GPU1 経由の迂回ルートの場合は容易ではない。cudaMemcpy()関数を用いる場合には自動的に送受信バッファの場所を判定し、CPU と GPU0 の間の転送を GPU1 で行おうとしても GPU0 が転送を行ってしまい、複数のパスの併用という提案手法に用いることはできない。そこで代替手段として GPU で実行するカーネルによってこの転送を実現する。GPU0 は DMA コントローラで実行し、GPU1 はカーネルによって実行することになる。両 GPU で同じタイミングで転送が行われるようにするため GPU1 上のカーネルでの排他制御を採用し、各転送方向について同時に１ペアだけ転送を行うように制限する。

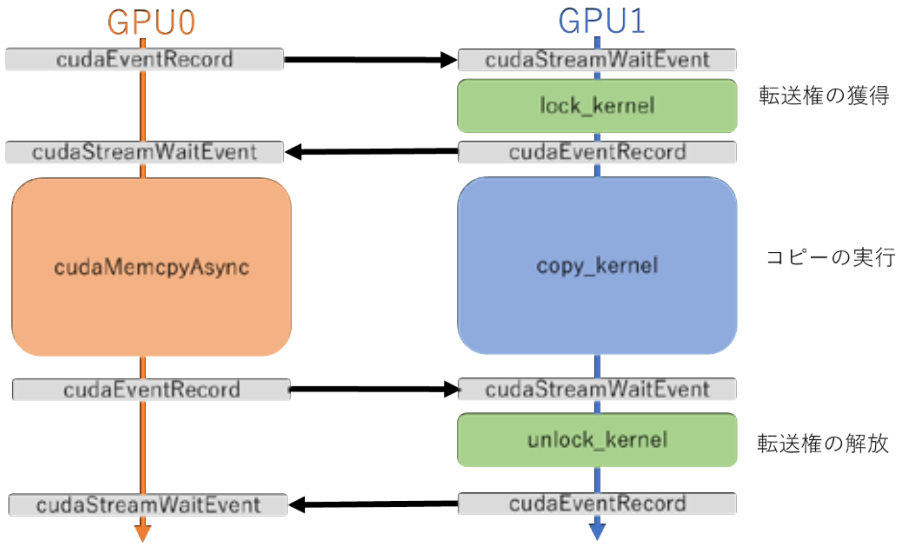


図 16 排他制御を用いる提案手法の cudaMemcpyAsync()関数の実行内容

ホスト・デバイス間で双方向転送を行う場合の提案手法の効果を評価した。データサイズが小さい場合にはもともとバンド幅ネックではなく、提案手法ではさらに 2 基の GPU を制御して同期をとるためのオーバヘッドがあり転送速度は改善しない。cudaMemcpy の転送アルゴリズムがサイズによって切り替わるためグラフが滑らかにならないが、1MB あたりで提案手法が上回り、サイズが十分大きくなれば複数のパスを併用するバンド幅向上の効果が発揮され、２倍近い転送速度が実現される。

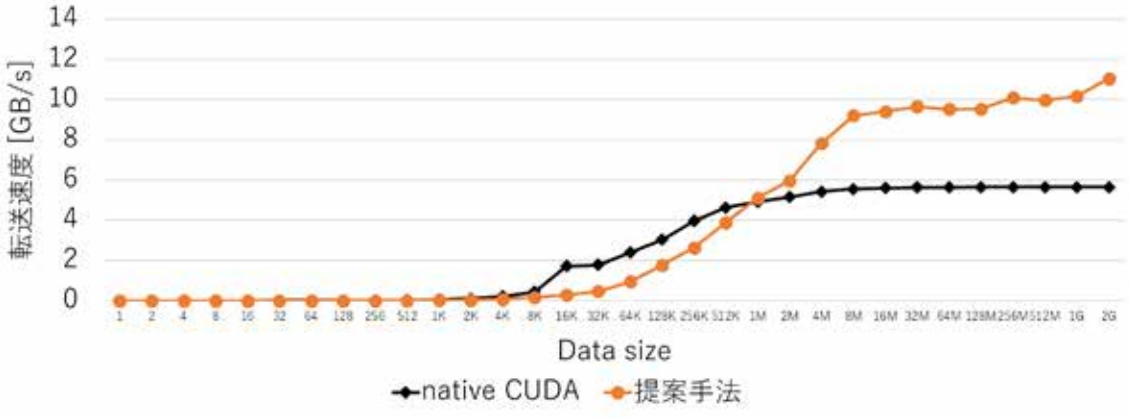


図 17 双方向転送時の提案手法による転送速度向上

[9] 鞍点型連立一次方程式に対する階層並列型数値解法の並列実装と性能評価（多田野）

鞍点型と呼ばれる連立一次方程式：

$$\begin{bmatrix} A & B \\ C^{\mathrm{T}} & O \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{g} \end{bmatrix}$$

は偏微分方程式の初期値・境界値問題に対するメッシュレス離散化法，構造解析，流体計算などの分野に現れ，求解に多くの計算時間を占める．ここで，A は n 次正則行列，B, C は $n \times m$ フルランク行列，O は m 次零行列，$\boldsymbol{x}, \boldsymbol{f}$ は n 次元ベクトル，$\boldsymbol{y}, \boldsymbol{g}$ は m 次元ベクトルである．同方程式が大規模である場合は反復法による求解をせざるを得ないが，行列 B, C の列数 m が多い場合は，行列 A が良条件であったとしても求解に非常に多くの反復回数を要したり，求解ができないことがある．

我々はこの鞍点型連立一次方程式をそのまま解くのではなく，同方程式のブロック構造を利用して問題を変形し，複数右辺連立一次方程式の求解を経由して解く手法を構築した（以下，提案法と呼ぶ）．鞍点型連立一次方程式の解ベクトル $\boldsymbol{x}, \boldsymbol{y}$ は以下のように表される．

$$\begin{cases} \boldsymbol{x} = A^{-1}\boldsymbol{f} - A^{-1}B\boldsymbol{y}, \\ \boldsymbol{y} = (C^T A^{-1} B)^{-1}(C^T A^{-1}\boldsymbol{f} - \boldsymbol{g}). \end{cases}$$

ここで，$U \equiv A^{-1}B, \boldsymbol{v} \equiv A^{-1}\boldsymbol{f}$ と定義し，さらに $\tilde{X} \equiv [U\ \boldsymbol{v}]$, $\tilde{B} \equiv [B\ \boldsymbol{f}]$ とおくと，行列 U とベクトル $\boldsymbol{v}$ は複数右辺連立一次方程式 $A\tilde{X} = \tilde{B}$ を解くことにより得られる．得られた解を用いることで，鞍点型連立一次方程式の解ベクトル $\boldsymbol{x}, \boldsymbol{y}$ を求めることができる．提案法における計算の主要部は，この複数右辺連立一次方程式の求解部分である．この部分を高速化することにより，提案法全体の求解高速化を達成することができる．

複数右辺連立一次方程式の右辺項 $\tilde{B}$ は互いに独立であるため，これらを分割することで並列に求解が可能となる．行列 $\tilde{X}$, $\tilde{B}$ を以下のように分割する．

$$\tilde{X} = \left[\tilde{X}^{(0)}\ \tilde{X}^{(1)} \dots \tilde{X}^{(P-1)}\right], \tilde{B} = \left[\tilde{B}^{(0)}\ \tilde{B}^{(1)} \dots \tilde{B}^{(P-1)}\right].$$

ここで，$\tilde{X}^{(j)}, \tilde{B}^{(j)}\ (j = 0, 1, \dots, P-1)$ は $n \times (m/P)$ 行列である．このように分割することで，以下の複数右辺連立一次方程式が得られる．

$$A\tilde{X}^{(j)} = \tilde{B}^{(j)}, j = 0, 1, \dots, P-1.$$

これらは互いに独立に求解が可能であるため，第 j 番目の方程式 $A\tilde{X}^{(j)} = \tilde{B}^{(j)}$ を第 j 番目の MPI プロセスに割り当てて求解を行う．さらに，各 MPI プロセスに割り当てられた複数右辺連立一次方程式も OpenMP を用いて並列求解することが可能である．したがって，提案法は階層型の並列性をもつ解法である．複数右辺連立一次方程式に対する有力な反復法として，ブロッククリロフ部分空間反復法がある．ブロッククリロフ部分空間反復法は，クリロフ部分空間反復法よりも少ない反復回数で求解できる可能性がある手法であり，同法を用いることでより高速な求解が期待できる．

数値実験により，提案法の並列性能を検証する．テスト行列として，行列 A には SuiteSparse Matrix Collection で公開されている行列 torso3（行列サイズ n：259,156，非零要素数：4,429,042）を用い，行列 B, C の列数 m は 3,000 とし要素は乱数で設定した．実験環境として，筑波大学計算科学研究センターのスーパーコンピュータ「Cygnus」を用いた．計算ノードの CPU は Intel Xeon Gold 6126 2.6GHz (12 cores)×2，コンパイラは Intel Fortran ver.

19.1.3，MPI ライブラリは OpenMPI ver. 4.0.3 であり，コンパイルオプション`-axCORE-AVX512 -align array64byte`を用いた．ハイブリッド並列を用い，各 CPU に MPI プロセスを割り付け，分割された連立一次方程式を求解した．CPU 内部では OpenMP を用いて並列化を行った．各連立一次方程式はブロッククリロフ部分空間反復法の 1 つである Block GWBiCGSTABrQ 法で求解し，前処理として近似逆行列前処理を用いた．前処理行列は全ての連立一次方程式で共通であるため，前処理行列計算は全計算ノードを用いて並列計算した．

図 18 に，ノード数 N の変化に対する鞍点型連立一次方程式の求解時間変化を示す．1 ノードあたり 2MPI プロセスが実行されているため，分割された各連立一次方程式の右辺ベクトル数は $m = 3{,}000/(2N)$ となる．$N = 1$ の場合，求解に要した時間は 358.7 秒であった．一方，$N = 2$ の場合は 124.5 秒であり，1 ノードを基準した場合の速度向上率は 2.88 であった．また，$N = 32$ の場合の計算時間は 10.2 秒であり，1 ノード基準の速度向上率は 35.2 であった．複数右辺連立一次方程式の求解にブロッククリロフ部分空間反復法を用いており，同法の 1 反復あたりの計算量は右辺ベクトル数の二乗に比例する．そのため，$N = 1$ の場合の計算時間が突出して長くなり，ノード数を上回る速度向上率が得られたと考えられる．以上より，提案法は並列計算環境と親和性があり，高い並列性能を示すことが確認された．この研究成果は，国際会議 JSST2021 にて発表した．

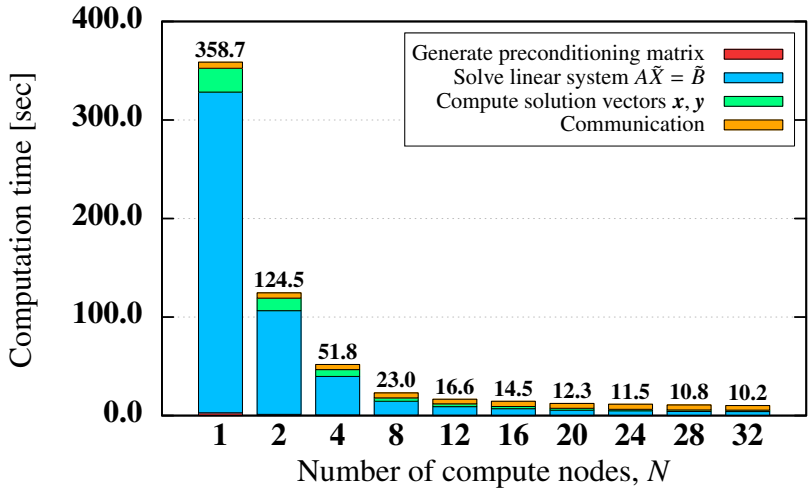


図 18 ノード数 N の変化に対する鞍点型連立一次方程式の求解時間変化

[10] 3 次元画像再構成問題で現れる鞍点型連立一次方程式の求解高速化（多田野）

複数枚の断層画像を用いて 3 次元オブジェクトを再構成する問題を，3 次元画像再構成問題と呼ぶ．同問題に対する手法として，拡張 CSRBF 法と呼ばれる方法がある．同法では，各

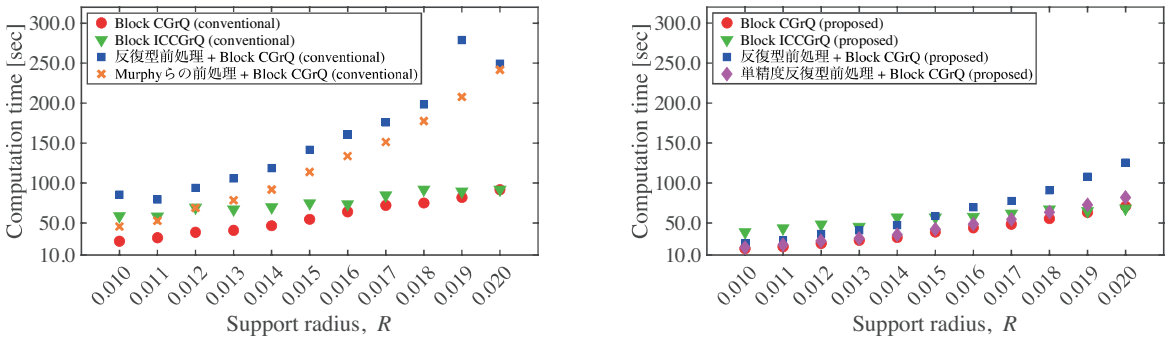
断層画像ごとに 2 次元補間関数を決定し，断層画像間の線形補間を行うことで 3 次元画像を再構成する．同法における 2 次元補間関数を決定するために，以下の鞍点型連立一次方程式を解く必要がある．

$$\begin{bmatrix} A & C \\ C^{\mathrm{T}} & O_3 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} F \\ O_{3,K} \end{bmatrix}.$$

ここで，A は n 次実対称行列，C は $n \times 3$ 行列，X, F は $n \times K$ 行列，Y は $3 \times K$ 行列，O_3 及び $O_{3,K}$ はそれぞれ 3 次零行列，$3 \times K$ 零行列である．K は断層画像の枚数を表し，行列 F の要素は断層画像の画素値によって決定される．拡張 CSRBF 法ではこの鞍点型連立一次方程式の求解に多くの計算時間を要するため，高速求解手法が必要である．そこで本研究では，前述の提案法をこの鞍点型連立一次方程式に適用し，高速化を図った．

数値実験により，拡張 CSRBF 法で現れる鞍点型連立一次方程式に対する提案法の有効性を検証する．数値実験では，カイコガ脳断層画像 127 枚の 3 次元画像再構成問題で現れる鞍点型連立一次方程式を扱った．行列 A のサイズは 262,144，$K = 127$ である．拡張 CSRBF 法にはサポート半径 R と呼ばれるパラメータがあり，このパラメータによって行列 A の非零要素数が変化する．数値実験には，筑波大学計算科学研究センターのスーパーコンピュータ「Cygnus」の 1 ノードを用いた．ノード内では OpenMP を用いて並列化し，24 スレッド並列計算した．コンパイラとして Intel Fortran ver. 19.1.0，コンパイルオプションは`-qopenmp -O3 -axCORE-AVX512 -heap-arrays` を用いた．反復法として，ブロッククリロフ部分空間反復法の一つである Block CGrQ 法を用いた．また，前処理法として不完全コレスキー分解前処理，Global CG 法による反復型前処理，鞍点型連立一次方程式に対する Murphy らの前処理法（従来法のみ）を用いた．提案法においては，反復型前処理を単精度で実行した場合も併せて性能評価を行った．

図 19 に，サポート半径 R の変化に対する鞍点型連立一次方程式の従来法と提案法の求解時間変化を示す．従来法では各前処理法ごとに計算時間のばらつきが見られ，反復型前処理と Murphy らの前処理を用いた場合は多くの時間を要する傾向にあった．また，提案法を用いることにより，従来法よりも高速に鞍点型連立一次方程式を求解することができた．本研究の成果は，日本応用数理学会 2021 年度年会において発表済みである．



(a) 従来法　　(b) 提案法

図 19 サポート半径 R の変化に対する鞍点型連立一次方程式の求解時間変化

[11] Intel oneAPI を用いた GPU・FPGA 複合型演算加速（朴，小林，藤田）

Intel oneAPI とは，Intel 社により提唱されたクロスアーキテクチャプログラミングフレームワークである．SYCL をベースにした DPC++による単一言語プラットフォームを提供し，複数のデバイス間における連携プログラミングが可能である．本研究ではまず始めに，従来は GPU のための CUDA プログラムと FPGA のための OpenCL プログラムを完全に分離してコンパイルしていたものを，oneAPI 環境で統合コンパイル・実行する環境を構築した．この環境を用いた GPU・FPGA 連携プログラミングの概要を図 20 に示す．

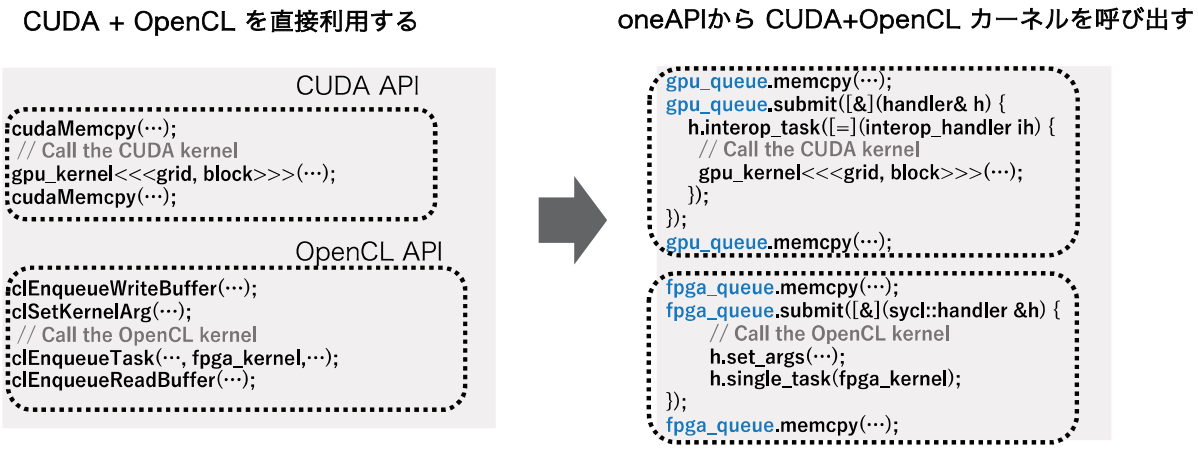


図 20　Intel oneAPI を用いた GPU・FPGA 連携プログラミングの概要

本研究では，既存の CUDA や OpenCL によるプログラム部分コードを組み合わせるためのフレームワークとして oneAPI を用いている．図に示すように，oneAPI から呼び出す計算カーネルの中身は CUDA や OpenCL で実装されており，それを oneAPI の Queue によって統一的に管理することに成功している．これにより，CUDA で記述された GPU 実装のコードや OpenCL で記述された FPGA 実装のコードを DPC++に書き直すことなく，oneAPI の枠組みで利用・連携できることが明らかとなった．

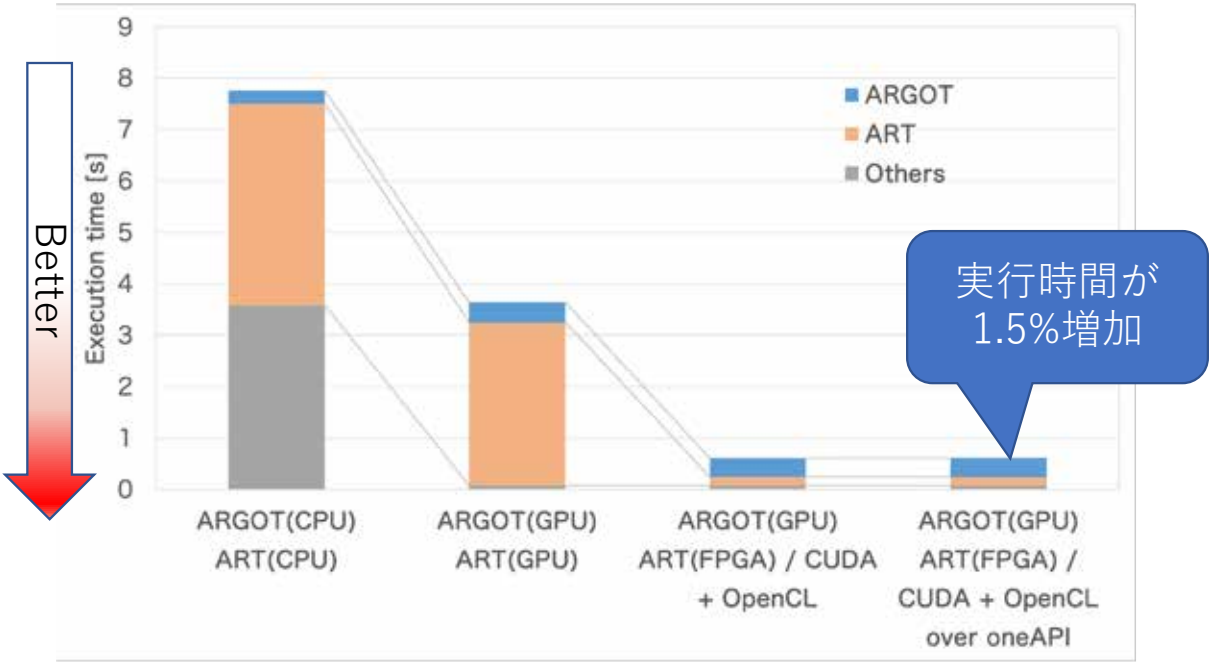


図 21　問題サイズ 32^3 における CPU，GPU，FPGA を利用した 各実装における ARGOT コード性能評価

このことを踏まえて，本研究では，これまでに CUDA と OpenCL で実装された GPU-FPGA 版 ARGOT コードを oneAPI の枠組み実行し，そのアプリケーション性能を評価した．その結果を図 21 に示す．評価には Intel Xeon Gold 6242 CPU を 1 台，PCIe カード型 NVIDIA Tesla V100 GPU を 1 台，Intel FPGA PAC D5005 を 1 台使用した．図の ARGOT(CPU) / ART(CPU) は ARGOT 法と ART 法がどちらも CPU 実装，ARGOT(GPU) / ART(GPU) はどちらも GPU 実装，ARGOT(GPU) / ART(FPGA) は ARGOT 法が GPU 実装，ART 法が FPGA 実装であることを表す．CPU 実装は C 言語ベースであり，OpenMP を用いたスレッド並列処理が適用されている．本研究では，単一の Xeon CPU (16 コア) を利用しており，1 コア 1 スレッドのマッピングでプログラムを実行した．GPU 実装は，CPU 実装をベースとしており，コードは CUDA で記述されている．問題サイズは 32^3 とした．

図に示されているように，GPU を使用したとしても ART 法は ARGOT コードの支配的な処理であることが分かる．これは，ART 法のアルゴリズムはベクトル長が部分的に縮小したり，メモリへのアクセスがランダムとなりキャッシュヒット率を低下させるなど SIMD-like なアーキテクチャに不向きな性質を孕んでいるためである．また，32^3 の問題サイズでは，5,120 個の CUDA コアを十分に利用するための並列性が不足していることも性能が低い要因として挙げられる．一方，ART 法を FPGA にオフロードすると，FPGA に実装されたパイプラインハードウェアによって ART 法の演算加速が実現されていることが分かる．今回の評価における問題サイズが 32^3 の場合，FPGA 実装版 ART は，GPU 実装版 ART と比較して 18.7 倍の性能向上を達成した．そして，CUDA+OpenCL による実装と oneAPI 実装を比較したところ，

oneAPI 実装の実行時間が 1.5%だけ増加した．つまり，oneAPI の枠組みで CUDA カーネル・OpenCL カーネルを呼び出し，制御するオーバーヘッドは無視できるぐらい小さく，CUDA+OpenCL で実装した GPU・FPGA 連携アプリケーションとほぼ同等の性能が得られることが明らかとなった．

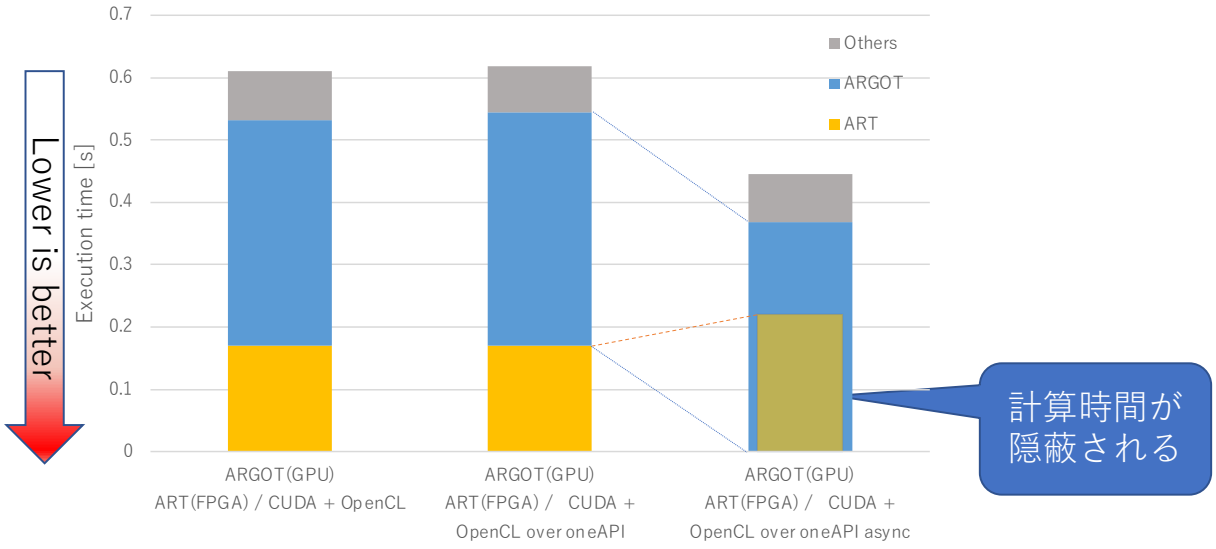


図 22　ARGOT 法および ART 法の非同期実行時の性能評価

ARGOT コードにおける ARGOT 法および ART 法は非同期に実行可能であり，その性能評価結果を図 22 に示す．ARGOT 法および ART 法の非同期実行は，それぞれの演算を担当する CUDA カーネル・OpenCL カーネルのインターフェースとなっている各 Queue を操作する OpenMP スレッドを生成し，並列実行することで実現できる．図に示すように，ART 法の実行時間が ARGOT 法の実行時間で隠蔽されており，全体として GPU 実行（ARGOT 法）と FPGA 実行（ART 法）を逐次的に処理する場合に比べ 1.38 倍の高速化を実現した．

[12] OpenACC+MPI による宇宙輻射輸送コード ARGOT の多ノード並列化（朴，小林，藤田)

現在における GPU・FPGA 複合演算加速は CUDA+OpenCL の混合プログラミングで実現しているが，このようなマルチリンガルプログラミングモデルはプラグラマに多大な負担を強いる．そのため，GPU-FPGA 連携のためのよりユーザービリティの高いプログラミング環境が必要とされる．これを実現するために，我々は指示文ベースのプログラミングモデルである OpenACC に着目しており，現在，我々と理化学研究所計算科学研究センター及び米国 Oak Ridge National Laboratory 間で，両演算デバイスを OpenACC による統一的記述によって利用可能にするプログラミング環境に関する共同研究を実施している．これらの背景を踏まえ，この研究では，ユーザビリティの高い GPU-FPGA 連携の実現を見据えた予備評価とし

て，ARGOT コードを OpenACC で実装し，OpenMP ベースの CPU 実装と CUDA ベースの GPU 実装との性能評価を行った．

OpenACC は GPU やメニーコアアクセラレータ向けのプログラムを容易に記述することを目的とした並列プログラミング言語規格である． C/C++や Fortran で記述されたプログラムに対し，OpenMP のようなコンパイラ指示文 #pragma を挿入することによって，アクセラレータにオフロードすべきプログラムのホットスポットをコンパイラに明示することができる．本研究では，OpenMP ベースの CPU 実装を参考に，GPU にオフロードすべき並列領域の指定を行った．すなわち，#pragma omp parallel for の箇所を #pragma acc parallel loop に置換し，並列領域における関数呼び出しは #pragma acc route seq にて OpenACC 化した．また，CPU と GPU 間のデータ転送は CUDA 版と等価になるように実装した．初期化時に #pragma acc enter data copyin() にて主要データを GPU メモリに常駐させ，CPU で実行しなければならない処理 (例：ログ書き出し) のみ #pragma acc update host() を使用し，必要なデータを GPU から CPU に転送した．

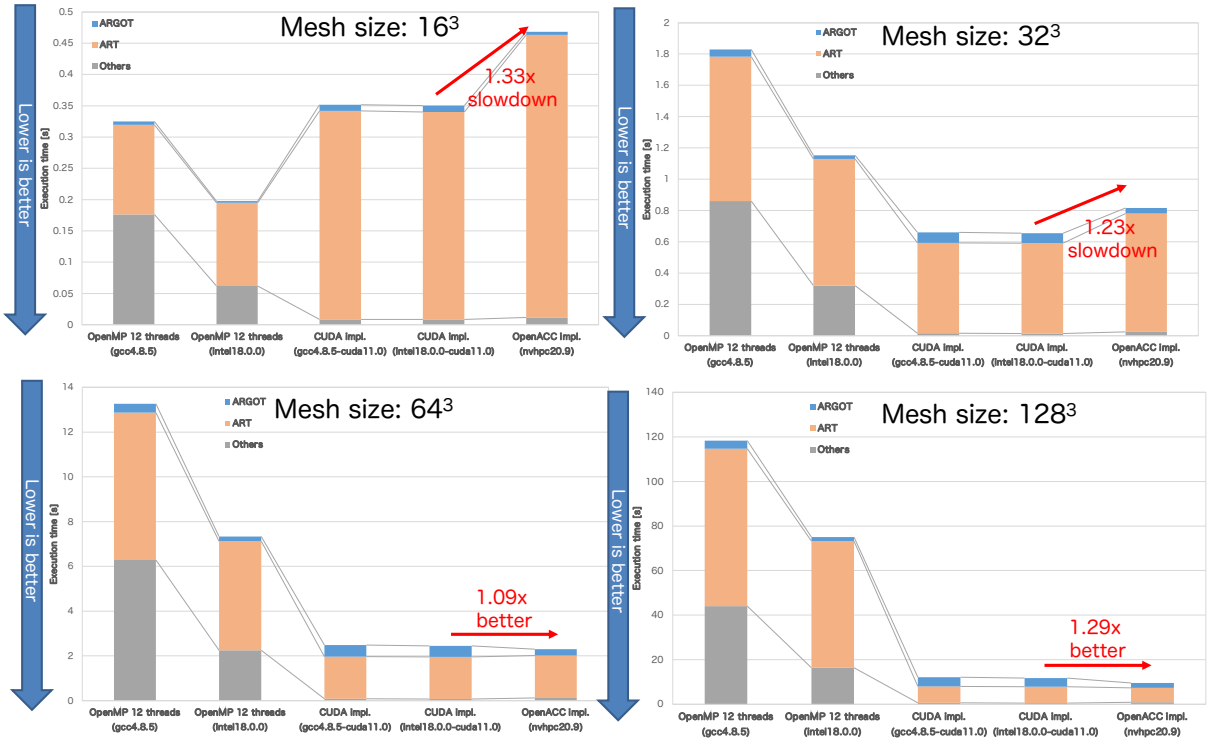


図 23　1 GPU を使用した OpenACC 実装の性能評価

OpenACC 化した ARGOT コードの性能を OpenMP ベースの CPU 実装および CUDA ベースの GPU 実装と比較した結果を図 23 に示す．性能比較には，Intel Xeon Gold 6126 CPU を 1 台，PCIe カード型 NVIDIA Tesla V100 GPU を 1 台使用し，問題サイズは 16^3 から 128^3 までとした．CUDA 実装と OpenACC 実装を比較すると，問題サイズが大きくなるにつれて OpenACC 実装の性能が CUDA 実装を最大 20 %程度上回ることが確認された．ただし，CUDA 実装の ARGOT 法では pinned メモリを繰り返し確保するコードが含まれていたため，この

影響によるところが大きい．従って，この pinned メモリを初期化時に 1 度だけ確保する方針に変更すれば，CUDA 実装と OpenACC 実装の性能は等価になることが予想される．逆に，問題サイズが小さい場合では，OpenACC 実装の性能は CUDA 実装の 70 % から 80% 程度となり，これは OpenACC 実装の ART 法の性能が CUDA 実装の ART 法に及んでいないことに起因している．性能が及ばない原因を調査するために，ART 法のコア演算部分のループにおけるコンパイラメッセージを確認したところ，Complex loop carried dependence of ... と表示されており，コンパイラがデータ独立であるかを上手く解析できていないことが分かった．また，16^3 や 32^3 の問題サイズでは，5,120 個の CUDA コアを十分に利用するための並列性が不足しているため，コンパイラの解析能力による影響をより受けたと解釈している．

次に，OpenACC 版 ARGOT コードを MPI を用いてマルチノード・マルチ GPU 化し，強スケーリングの条件下で ARGOT コードの性能を評価した．本研究では，図 24 に示すように 1 つの MPI プロセスに 1 つの GPU を割り当てるアプローチを採用している．

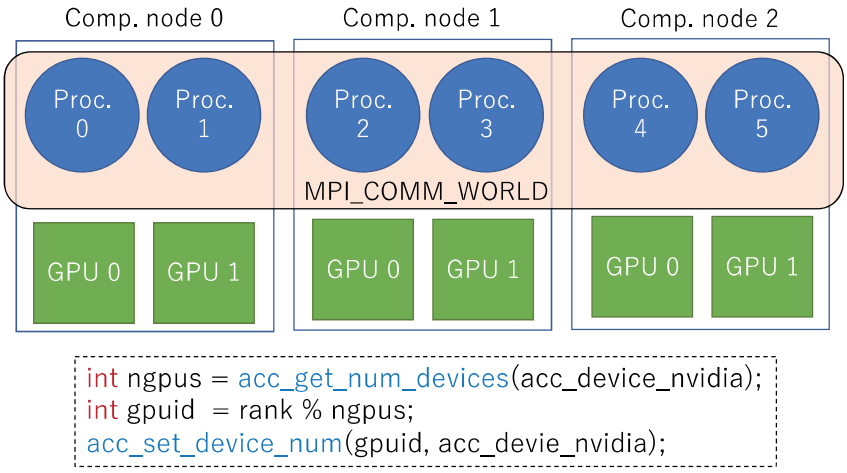


図 24　マルチノード・マルチ GPU 化けの方針 (1 GPU / MPI プロセス)

性能評価のプラットフォームとして，筑波大学計算科学研究センターで運用中の Cygnus スーパーコンピュータを用いる．Cygnus は，アクセラレータとして GPU のみを搭載した Deneb ノードと GPU および FPGA を搭載した Albireo ノードの 2 種類の計算ノードから構成されており，今回における性能評価では 2 台の Deneb ノード (8 GPU) を使用した．評価に用いる問題サイズは 128^3 とし，それを 1 から 8 プロセス (GPU 数) で並列実行した場合に要するシミュレーション時間を CUDA 実装と OpenACC 実装とで比較評価した．その結果を図 25 に示す．問題空間は 2 プロセスの場合は x 方向に沿って 2 分割，4 プロセスの場合は x および y 方向に沿って 2 分割，8 プロセスの場合は x, y, z 方向に沿って 2 分割されている．1 プロセスの場合における，CUDA 実装と OpenACC 実装における性能差の原因は，上述した ARGOT 法の pinned メモリの確保方法によるものである． そのため，

プロセス数を多くすると，プロセスあたりの問題サイズが 64^3 に近づき，CUDA 実装と OpenACC 実装の性能差は小さくなる．

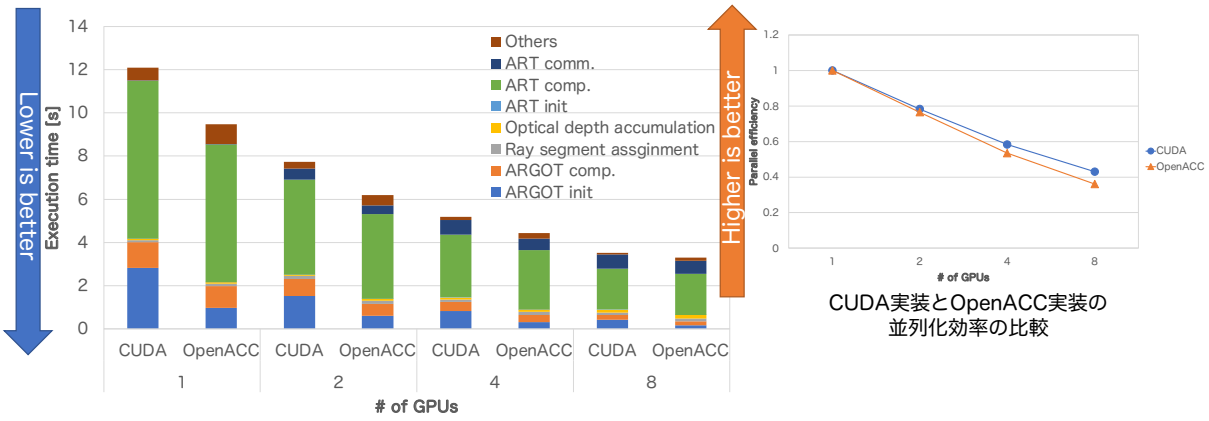


図 25　問題サイズが 128^3 の ARGOT コードを最大 8 プロセス (8 GPU) で並列実行した場合の性能比較

CUDA 実装および OpenACC 実装の並列化効率を比較すると，両者の差は数パーセント程度であるため，MPI＋OpenACC 実装であっても，MPI＋CUDA 実装と同等のノード並列化が実現できている．しかし，CUDA 実装においても並列化効率は 4 割程度であるため，現在，並列化効率が低下する要因について調査中である．

[13] OpenCL プログラミングによる FPGA 実装のためのソーティングライブラリ (朴，小林，藤田)

FPGA (Field-programmable Gate Array) は，その柔軟性により，アプリケーションに特化した計算パイプラインやデータ供給システムを構築できるため，高性能計算の研究において大きな注目を集めている．この柔軟性に加え，FPGA アプリケーションを OpenCL で開発するためのツールチェインが FPGA ベンダーから提供されており，プログラミングに必要な労力が軽減されている．しかし，OpenCL ベースの開発手法では抽象度が高いため，きめ細かな性能チューニングを行うことが難しいという弊害がある．この問題を解決する手段の一つとしては，FPGA にオフロードする実アプリケーションに頻出する汎用の計算カーネルを FPGA 向けに最適化し，ユーザーに IP コアやライブラリの形で提供することが挙げられる．

本研究では，基本的な算術演算であるソートを対象とし，FPGA 用 OpenCL プログラミングモデルで利用可能なソーティングライブラリを開発した．図 26 に示すように，開発したソーティングライブラリでは，ソート処理を実行するためのハードウェアエンジンがハードウェア記述言語 (HDL) で実装されており，それを OpenCL カーネルコードから関数として呼び出している．これにより，アプリケーション開発者は OpenCL の抽象度に留まりながら，RTL でチューニングされたソート機能を利用できる．

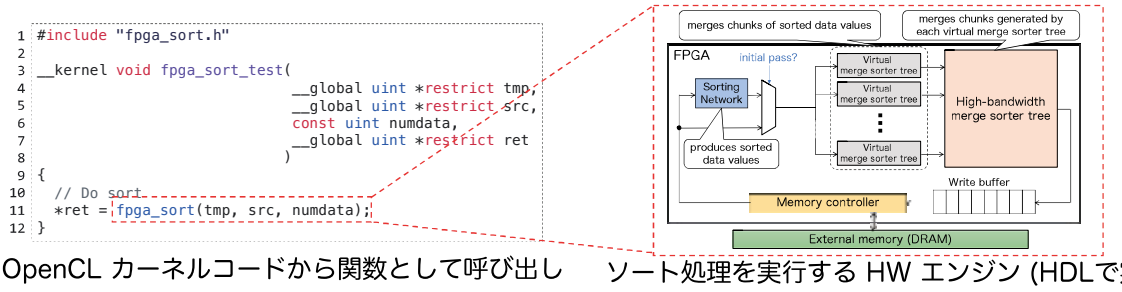


図 26　FPGA 用 OpenCL プログラミングモデルで利用可能なソーティングライブラリ

表 3　高位合成向けのマージソート ([14]: Kastner, R., Matai, J. and Neuendor ffer, S.: Parallel Programming for FPGAs, ArXiv e-prints (2018).) との比較結果．レコード数は 2^{29}

Integer									
	ALMs	(%)	Registers	(%)	M20Ks	(%)	DSPs	(%)	fmax [MHz]	Throughput [MB/s]
Our sorting library	76,644	8%	219,000	6%	350	3%	2	0.03%	**380.66**	**118**
The merge sort [14]	23,642	3%	48,755	1%	147	1%	0	0%	352.6	0.32

Floating point									
	ALMs	(%)	Registers	(%)	M20Ks	(%)	DSPs	(%)	fmax [MHz]	Throughput [MB/s]
Our sorting library	79,543	9%	229,761	6%	350	3%	2	0.03%	**380.66**	**116**
The merge sort [14]	18,263	2%	50,962	1%	147	1%	0	0%	351.0	0.32

表 3 に FPGA 用 OpenCL プログラミングモデル向けに再実装したマージソートとの比較結果を示す．我々のソートライブラリは再実装したマージソートに比べて少なくとも 2 倍のハードウェア資源を消費する一方，その動作周波数は 1.08 倍であり，ソートのスループットはソート性能は 3 桁向上する．動作周波数に差がそれほどないにも関わらず，ソートのスループットが 3 桁以上も違うのは，開発したソーティングエンジンが高度にパイプライン化されたアーキテクチャであるためである．ソートするデータが浮動小数点の場合でも，FPGA 上にパイプラインを構築しているため，整数の場合とほぼ同じ性能を実現できる．整数の場合と比較してスループットが 1.7%低下したのは，浮動小数型のデータを比較可能な整数型データに変換・復元するためのハードウェアモジュールを含むパイプラインを埋めるためのオーバーヘッドが原因である．これらのモジュールが含まれるため，ハードウェアリソースの使用量が約 1 %増加している．

本研究で得られた成果を論文以外の形でも積極的に普及していくために，成果物であるソーティングエンジンおよびその OpenCL ライブラリのソースコードを Github にオープンソースとして公開している (https://github.com/ac2-prod/fpga_sort).

[14] HBM2 搭載する FPGA のためのメモリシステムに関する研究(藤田，小林，朴)

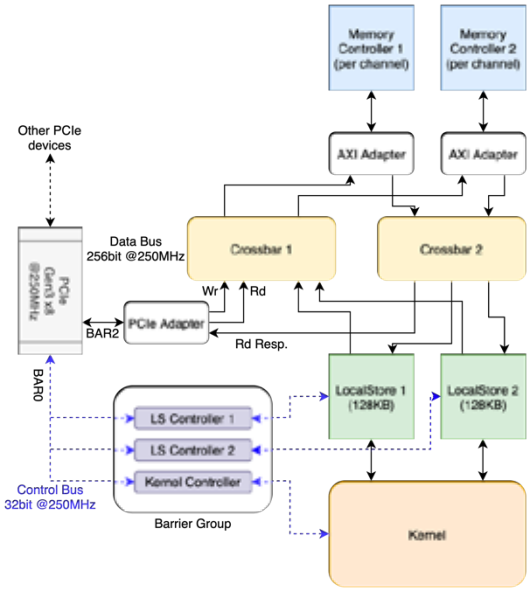
我々は Field Programmable Gate Array （FPGA）を HPC 分野で演算加速装置として用いる研究を行っており，早期宇宙の問題を輻射輸送計算の高速化や，複数の FPGA における並列計

算を FPGA 自身が持つ光リンクを用いて実現した．しかしながら，FPGA が持つ外部メモリ帯域は，HPC 分野で用いられる他のアクセラレータと比べて低く，HPC アプリケーションに対して FPGA を用いる際のボトルネックとなることが多く，FPGA 利用を進める上での障壁となることが多い．

近年 FPGA ベンダーより広帯域なメモリである High Bandwidth Memory 2（HBM2）が搭載されている FPGA ボードがリリースされつつある．Intel FPGA においては，HBM2 を用いることで最大 512GB/s の外部メモリ帯域を利用できる．HBM2 は比較的低速のメモリを多数束ねたアーキテクチャであり，全てのメモリチャンネルに同時アクセスしなければ，その高い理論ピーク性能を得ることが出来ない．HBM2 は他の演算加速装置でも利用されており，特に高性能な Graphics Processing Unit（GPU）に搭載されている．GPU においては，高性能なキャッシュや内部メモリネットワークを土台として，多数のコアによる並列アクセスが行われ，HBM2 の高い性能を引き出している．しかしながら，FPGA には HBM2 を扱うためのそのような（固定機能としての）専用メモリシステムが実装されていない．このようなメモリシステムを用いず，複雑なメモリアクセスを行うことが一般的である HPC アプリケーションを実装することは困難であり，HBM2 搭載 FPGA 向けかつ HPC アプリケーション向けメモリシステムが求められている．これらのシステムは，再構成可能な回路要素を用いて FPGA 内に実装しなければならない．

本研究では，この問題を解決するために，HBM2 搭載 FPGA 向けかつ HPC アプリケーション向けメモリシステムの提案と実装を行う．提案するシステムは 2 つの主要コンポーネントがあり，メモリアクセスの柔軟性を担保し複数のメモリチャンネルを効率よく扱うためのクロスバと，FPGA に内蔵されているメモリ（Block RAM: BRAM）を用いるキャッシュ（Addressable Cache）から構成される．また，アプリケーションの開発には，ソフトウェアで用いられている C 言語や C++言語を用いて FPGA ハードウェアを開発できる高位合成（High-Level Synthesis: HLS）を前提として設計を行っている．

図 27 に提案するシステムの概要図および性能評価の結果を示す．本研究は本年度より開始したものであるため，システムが期待通りに動作するか検証をするためのプロトタイプ実装を行った．プロトタイプ実装であるため，FPGA が持つ 32 チャンネルの内 2 チャンネルを利用できるものであり，理論ピークに対して 71%の性能が得られた．HBM2 とキャッシュ間の転送制御のオーバヘッドが大きく，それによって性能が低下していることがわかっており，最適化および全 32 チャンネルを利用できる実装を行うことが今後の課題である．



Mem. Type	Bandwidth	Theoretical Peak	Eff.
BRAM	6.687 GB/s	8.0 GB/s	83.6%
HBM2	5.682 GB/s	8.0 GB/s	71.0%

図 27 提案するメモリシステムの概要図（左）および性能評価結果（右）.

4. 教育

学生の指導状況

1. 柏野隆太，修士（工学），GPU・FPGA 複合型演算加速システムにおけるプログラミングと性能評価, 筑波大学大学院理工情報生命学術院システム情報工学研究群修士論文，令和 4 年 3 月（指導：朴泰祐）
2. 渡邉　孔英，修士（工学），都市気象シミュレーションコードの OpenACC による並列 GPU 化と性能評価，筑波大学大学院理工情報生命学術院システム情報工学研究群修士論文，令和 4 年 3 月（指導：朴泰祐）
3. 杉﨑行優，修士（工学），A fast algorithm and its implementation for computing the number of magic series，筑波大学大学院理工情報生命学術院システム情報工学研究群修士論文，令和 4 年 3 月（指導：高橋大介）
4. 倉本健，修士（工学），不揮発性メモリの揮発性利用に関する研究，筑波大学大学院理工情報生命学術院システム情報工学研究群修士論文，令和 4 年 3 月（指導教員：建部修見）
5. 石川翔大，修士（工学），画像再構成問題に現れる鞍点型連立一次方程式に対する高速求解手法の構築, 筑波大学大学院理工情報生命学術院システム情報工学研究群修士論文，令和 4 年 3 月（指導：多田野寛人）.
6. 山口博將，学士（情報科学），ルジャンドル予想の2^{32}以下の数値的検証，筑波大学情報学群情報科学類卒業論文，令和 4 年 3 月（指導：高橋大介）

7. 木下嵩裕，学士（情報工学），不揮発性メモリにおける高速トランザクションシステムの検討，筑波大学情報学群情報科学類卒業論文，令和 4 年 3 月（指導教員：建部修見）
8. 小山創平，学士（情報科学），グラフニューラルネットワークにおける分散学習システムの設計，筑波大学情報学群情報科学類卒業論文，令和 4 年 3 月（指導教員：建部修見）
9. CHOI YEON KYU、学士（情報工学）、レイトレーシング用ハードウェア RT コアの汎用計算利用の研究、筑波大学情報学群情報科学類卒業研究論文、令和 4 年 3 月（指導：額田彰）
10. 佐野由佳，学士（情報工学），アクセラレータ間の相互接続を考慮した GPU・FPGA 複合型演算加速と性能評価，筑波大学情報学群情報科学類卒業研究論文，令和 4 年 3 月（指導：小林諒平）

5. 受賞、外部資金、知的財産権等

受賞

1. IEEE Cluster 2021 Outstanding Service Award，高橋大介，2021 年 9 月 10 日
2. 情報処理学会 論文編集貢献賞，建部修見，2021 年 5 月 11 日
3. 情報処理学会 山下研究記念賞，藤田典久，小林諒平，山口佳樹，上野知洋，佐野健太郎，朴泰祐「スーパーコンピュータ Cygnus 上における FPGA 間パイプライン通信の性能評価」，2021 年.

外部資金

1. 文部科学省・高性能汎用計算機高度利用事業，朴泰祐（代表），H29～R3 年度，23,649 千円（R3 年度），「次世代演算通信融合型スーパーコンピュータの開発」
2. 科学研究費補助金基盤研究 (A)，朴泰祐（代表），H3～R6 年度，11, 700 千円（R3 年度），「多重複合演算加速機構を用いた次世代スーパーコンピューティング」
3. 科学研究費補助金基盤研究 (C)，高橋大介（代表），H31～R3 年度，1,170 千円（R3 年度），「エクサスケールシステムにおける高速フーリエ変換のアルゴリズムに関する研究」
4. 文部科学省委託研究，建部修見，分担，R3 年度，62,700 千円，HPCI の運営（HPCI 共用ストレージ用大規模分散ファイルシステムの機能整備等）
5. NEDO，建部修見，分担，H30～R5 年度，5,865 千円，実社会の事象をリアルタイム処理可能な次世代データ処理基盤技術の研究開発

6. 共同研究（富士通研究所），建部修見，代表，R3 年度，3,960 千円，Machine Learning アプリケーション向け高速データ供給技術の研究
7. 科学研究費補助金若手研究、額田彰（代表）、R2～R3 年度、1,600 千円（R3 年度）、「GPU アプリケーションに対するシステムレベルのチェックポイント技術の確立」
8. 科学研究費補助金基盤研究 (C)，多田野寛人（代表），R2～R4 年度，1,040 千円（R3 年度），「鞍点型連立一次方程式に対する階層並列型高速数値解法の開発」
9. 科学研究費補助金基盤研究 (A)、小林 諒平 (分担)、2021~2024 年度、500 千円(2021 年度)「多重複合演算加速機構を用いた次世代スーパーコンピューティング」

知的財産権

6. 研究業績

(1) 研究論文

A) 査読付き論文

1. Kazuki Furukawa, Tomoya Yokono, Yoshiki Yamaguchi, Kohji Yoshikawa, Norihisa Fujita, Ryohei Kobayashi, Taisuke Boku, Masayuki Umemura, " An Efficient RTL Buffering Scheme for an FPGA-Accelerated Simulation of Diffuse Radiative Transfer", Proc. of FPT'21, Auckland (virtual), Dec. 2021.
2. Koei Watanabe, Kohei Kikuchi, Taisuke Boku, Takuto Sato, Hiroyuki Kusaka, "High Resolution of City-Level Climate Simulation by GPU with Multi-Physical Phenomena", Proc. of NPC2021, Paris (virtual), Nov. 2021.
3. Takeyuki Harayama, Shuhei Kudo, Daichi Mukunoki, Toshiyuki Imamura, and Daisuke Takahashi, “A Rapid Euclidean Norm Calculation Algorithm that Reduces Overflow and Underflow”, Proc. 21st International Conference on Computational Science and Its Applications (ICCSA 2021), Part I, Lecture Notes in Computer Science, Vol. 12949, pp. 95-110, Springer, 2021.
4. Naruya Kitai, Daisuke Takahashi, Franz Franchetti, Takahiro Katagiri, Satoshi Ohshima, and Toru Nagai, “An Auto-tuning with Adaptation of A64 Scalable Vector Extension for SPIRAL”, Proc. 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW 2021), The 16th International Workshop on Automatic Performance Tuning (iWAPT 2021), pp. 789-797, 2021.
5. Osamu Tatebe, Kazuki Obata, Kohei Hiraga, Hiroki Ohtsuji, "CHFS: Parallel Consistent Hashing File System for Node-local Persistent Memory", Proceedings of the ACM

International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2022), pp.115-124, 10.1145/3492805.3492807, 2022.

6. Adrián P. Diéguez, Margarita Amor, Ramón Doallo, Akira Nukada, Satoshi Matsuoka. “Efficient High-Precision Integer Multiplication on the GPU”, International Journal of High Performance Computing Applications, SAGE Publications, Mar. 2022.
7. Hiroto Tadano and Shota Ishikawa, “An efficient approach for saddle point problems using block structure”, Journal of Advanced Simulation in Science and Engineering, Vol. 8, No. 1, pp. 114-127, 2021.
8. Hiroto Tadano and Shota Ishikawa, “Accuracy improvement of approximate solutions generated by the method for solving saddle point problems using block structure”, Proceedings of biennial IEEE Conference on Electromagnetic Field Computation (CEFC 2020), 4 pages.
9. Ryuta Kashino, Ryohei Kobayashi, Norihisa Fujita, and Taisuke Boku, “Multi-hetero Acceleration by GPU and FPGA for Astrophysics Simulation on oneAPI Environment”, HPCAsia2022: International Conference on High Performance Computing in Asia-Pacific Region, pp.84-93, 2022-01
10. Ryohei Kobayashi, Kento Miura, Norihisa Fujita, Taisuke Boku, and Toshiyuki Amagasa, “A Sorting Library for FPGA Implementation in OpenCL Programming”, Proceedings of the 11th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART '21)./(10)/pp.1-6, 2021-06
11. Norihisa Fujita, Ryohei Kobayashi, Yoshiki Yamaguchi and Taisuke Boku, "HBM2 Memory System for HPC Applications on an FPGA," 2021 IEEE International Conference on Cluster Computing (CLUSTER), 2021, pp. 783-786, doi: 10.1109/Cluster48925.2021.00116. (Short Paper)
12. Kazuki Furukawa, Ryohei Kobayashi, Tomoya Yokono, Norihisa Fujita, Yoshiki Yamaguchi, Taisuke Boku, Kohji Yoshikawa and Masayuki Umemura, "An efficient RTL buffering scheme for an FPGA-accelerated simulation of diffuse radiative transfer," 2021 International Conference on Field-Programmable Technology (ICFPT), 2021, pp. 1-9, doi: 10.1109/ICFPT52863.2021.9609944.

B) 査読無し論文

1. 渡邉孔英, 菊池航平, 朴泰祐, 佐藤拓人, ドアングアンヴァン, 日下博幸, "都市気象シミュレーション City-LES の OpenACC による完全 GPU 化と並列性能評価", 研究報告ハイパフォーマンスコンピューティング（HPC）, 2022-HPC-183(6), 1-8, 2022 年 3 月.

2. 綱島隆太, 小林諒平, 藤田典久, 朴泰祐, Seyong Lee, Jeffrey S. Vetter, 村井均, 中尾昌広, 辻美和子, 佐藤三久, "OpenACC による宇宙物理シミュレーションの GPU＋FPGA 協調計算の実装", 研究報告ハイパフォーマンスコンピューティング（HPC）, 2022-HPC-183(11), 1-9, 2022 年 3 月.
3. 高橋大介, “二次元分割を用いた並列三次元 FFT における計算と通信のオーバーラップの自動チューニング”, 日本応用数理学会 2021 年度年会講演予稿集, 2 pages, 2021 年 9 月
4. 平賀弘平, 建部修見, 計算ノード上の不揮発性メモリを用いた MPI-IO バーストバッファの設計, 研究報告ハイパフォーマンスコンピューティング（HPC）, Vol. 2022-HPC-183, No. 24, pp. 1-9, 2022 年 3 月
5. 建部修見, 計算ノードの不揮発性メモリを用いたキャッシュファイルシステムの設計, 研究報告ハイパフォーマンスコンピューティング（HPC）, Vol. 2022-HPC-183, No. 8, pp. 1-9, 2022 年 3 月
6. 小林諒平, 藤田典久, 山口佳樹, 朴泰祐, 吉川耕司, 安部牧人, 梅村雅之, “GPU クラスタを用いた宇宙輻射輸送コード ARGOT の OpenACC 実装と性能評価”, 研究報告ハイパフォーマンスコンピューティング（HPC）/2022-HPC-183(17)/pp.1-8, 2022-03
7. 小林諒平, 三浦賢人, 藤田典久, 朴泰祐, 天笠俊之, “FPGA 向け浮動小数点数型ソーティングライブラリの提案と実装”, IEICE-CPSY2021-8/IEICE-121(116)/pp.43-48, 2021-07
8. 小林諒平, 三浦賢人, 藤田典久, 朴泰祐, 天笠俊之, “コンパクション処理を活用した正規パス問合わせアクセラレータの FPGA 実装”, IEICE-RECONF2021-12/IEICE-121(59)/pp.62-67, 2021-06
9. 藤田典久, 小林諒平, 山口佳樹, 朴泰祐, “HBM2 搭載 FPGA をのための Addressable Cache を用いた HPC 向けメモリシステムの性能評価”, 研究報告ハイパフォーマンスコンピューティング（HPC）, 2021-HPC-183(9), 2022/03.
10. 藤田典久, 小林諒平, 山口佳樹, 朴泰祐, “FPGA における HPC アプリケーション向け HBM2 メモリシステムの提案と実装”, 研究報告ハイパフォーマンスコンピューティング（HPC）, 2021-HPC-180(17), 2021/07.

(2) 国際会議発表

A) 招待講演

1. Taisuke Boku, “Multi-Hetero Accelerated Computing - System and Applications –“, Mini Symposium “Targeting Future Exascale and Extreme Heterogeneity Era”, in SIAMPP22, on-line, Feb 26th, 2022.

2. Taisuke Boku, “Successful FPGA Programming Methods and Tools for HPC”, BoF at SC21, St. Louis, Nov. 16th, 2021.
3. Taisuke Boku, “Multi-Hybrid Acceleration for Multi-Physical Simulation", Int. Workshop WSSSPP2021, Sendai, Dec. 13th, 2021.
4. Taisuke Boku, “Data Science and AI by Big Memory Supercomputer", Int. Workshop DRBSD7-2021, in conjunction with SC21, St. Louis, Nov. 14th, 2021.
5. Taisuke Boku, “OpenACC Unified Programming Environment for GPU and FPGA Multi-Hybrid Acceleration", OpenACC Summit 2021, Sep. 14th, 2021.
6. Taisuke Boku, “FPGA-centric multi-hybrid acceleration for HPC applications -Preparing for Extreme Heterogeneity in High Performance Computing-", ISC21, on-line, Jul. 1st, 2021.
7. Taisuke Boku, “Successfull FPGA Programming", Birds of Feather, ISC21, on-line, Jun. 30th, 2021.
8. Taisuke Boku, “FPGAs for HPC: Open Challenges for Infrastructure and System Operation - FPGA & GPU: compensate with each other toward perfect HPC hardware solution-", Int. Symp. HEART2021, on-line, Jun. 21st, 2021.

B) 一般講演

1. Kazuki Furukawa, Tomoya Yokono, Yoshiki Yamaguchi, Kohji Yoshikawa, Norihisa Fujita, Ryohei Kobayashi, Taisuke Boku, Masayuki Umemura, " An Efficient RTL Buffering Scheme for an FPGA-Accelerated Simulation of Diffuse Radiative Transfer", FPT'21, Auckland (virtual), Dec. 2021.
2. Koei Watanabe, Kohei Kikuchi, Taisuke Boku, Takuto Sato, Hiroyuki Kusaka, "High Resolution of City-Level Climate Simulation by GPU with Multi-Physical Phenomena", NPC2021, Paris (virtual), Nov. 2021.
3. Daisuke Takahashi, “Parallel Implementation of FFT in a Finite Field”, SIAM Conference on Parallel Processing for Scientific Computing (PP22), Seattle, WA, USA (Virtual Conference), February 26, 2022.
4. Takeyuki Harayama, Shuhei Kudo, Daichi Mukunoki, Toshiyuki Imamura, and Daisuke Takahashi, “A Rapid Euclidean Norm Calculation Algorithm that Reduces Overflow and Underflow”, The 21st International Conference on Computational Science and Its Applications (ICCSA 2021), Cagliari, Italy (Virtual Conference), September 13, 2021.
5. Naruya Kitai, Daisuke Takahashi, Franz Franchetti, Takahiro Katagiri, Satoshi Ohshima, and Toru Nagai, “An Auto-tuning with Adaptation of A64 Scalable Vector Extension for SPIRAL”,

The 16th International Workshop on Automatic Performance Tuning (iWAPT 2021), Portland, OR, USA (Virtual Conference), May 21, 2021.

6. Osamu Tatebe, Kazuki Obata, Kohei Hiraga, Hiroki Ohtsuji, "CHFS: Parallel Consistent Hashing File System for Node-local Persistent Memory", ACM International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2022), Virtual event, January 2022.
7. Akira Nukada, “Performance Optimization of Allreduce Operation for Multi-GPU Systems”, The Fifth IEEE International Workshop on Benchmarking, Performance Tuning and Optimization for Big Data Applications (BPOD 2021), IEEE, pp. 3107-3112, Dec. 2021.
8. Hiroto Tadano, “Parallel implementation of the approach for solving saddle point problems using block structure”, The 40th JSST Annual International Conference on Simulation Technology (JSST 2021), Virtual Conference, Sep. 2021.
9. Taiga Yamashita, Taku Itoh, and Hiroto Tadano, “A parallelization strategy for incomplete decomposition-based preconditioning for solving linear systems”, The 40th JSST Annual International Conference on Simulation Technology (JSST 2021), Virtual Conference, Sep. 2021.
10. Ryuta Kashino, Ryohei Kobayashi, Norihisa Fujita, and Taisuke Boku, “Multi-hetero Acceleration by GPU and FPGA for Astrophysics Simulation on oneAPI Environment”, HPCAsia2022: International Conference on High Performance Computing in Asia-Pacific Region, January 13, 2022
11. Ryohei Kobayashi, Kento Miura, Norihisa Fujita, Taisuke Boku, and Toshiyuki Amagasa, “A Sorting Library for FPGA Implementation in OpenCL Programming”, The 11th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART '21), June 21, 2021
12. Norihisa Fujita, Ryohei Kobayashi, Yoshiki Yamaguchi and Taisuke Boku, "HBM2 Memory System for HPC Applications on an FPGA," 2021 IEEE International Conference on Cluster Computing (CLUSTER), 2021, pp. 783-786, doi: 10.1109/Cluster48925.2021.00116. (Short Paper)
13. Kazuki Furukawa, Ryohei Kobayashi, Tomoya Yokono, Norihisa Fujita, Yoshiki Yamaguchi, Taisuke Boku, Kohji Yoshikawa and Masayuki Umemura, "An efficient RTL buffering scheme for an FPGA-accelerated simulation of diffuse radiative transfer," 2021 International Conference on Field-Programmable Technology (ICFPT), 2021, pp. 1-9, doi: 10.1109/ICFPT52863.2021.9609944.

(3) 国内学会・研究会発表

A) 招待講演

1. 朴泰祐，「『富岳』を含む HPCI 資源を用いた計算科学に対する期待」，「富岳」成果創出加速プログラム物質・材料課題合同研究会，オンライン，2021 年 12 月 9 日.
2. 朴泰祐，「oneAPI による FPGA と GPU の複合演算加速アプリケーション実行」，Intel FPGA Technology Day，オンライン，2021 年 12 月.
3. 朴泰祐，「筑波大学計算科学研究センターにおける PC クラスタのあゆみ」，PC クラスタシンポジウム 2021，オンライン，2021 年 12 月 8 日.
4. 朴泰祐，「アカデミアの HPC におけるクラウド利用（または可能性）」，HPC in Cloud ワークショップ，on-line，2021 年 5 月 19 日.

B) その他の発表

1. 渡邉孔英, 菊池航平, 朴泰祐, 佐藤拓人，ドアングアンヴァン，日下博幸, "都市気象シミュレーション City-LES の OpenACC による完全 GPU 化と並列性能評価", 情報処理学会ハイパフォーマンスコンピューティング研究会, オンライン，2022 年 3 月.
2. 綱島隆太, 小林諒平, 藤田典久, 朴泰祐, Seyong Lee, Jeffrey S. Vetter, 村井均, 中尾昌広, 辻美和子, 佐藤三久, "OpenACC による宇宙物理シミュレーションの GPU＋FPGA 協調計算の実装", 情報処理学会ハイパフォーマンスコンピューティング研究会, オンライン，2022 年 3 月.
3. 高橋大介，“二次元分割を用いた並列三次元 FFT における計算と通信のオーバーラップの自動チューニング”，日本応用数理学会 2021 年度年会，オンライン開催，2021 年 9 月 7 日.
4. 平賀弘平, 建部修見，計算ノード上の不揮発性メモリを用いた MPI-IO バーストバッファの設計，情報処理学会 HPC 研究会，オンライン，2022 年 3 月
5. 建部修見, 計算ノードの不揮発性メモリを用いたキャッシュファイルシステムの設計, 情報処理学会 HPC 研究会，オンライン，2022 年 3 月
6. 立木佑弥，額田彰，遊休 GPU を利用したホスト・デバイス間通信の高速化，情報処理学会研究報告, Vol. 2022-HPC-183, No. 4, 2022 年 3 月.
7. Akira Nukada, “Hybrid Allreduce Algorithm for On-node Multi-GPU Systems using both NV-Link and PCI-Express Networks”, NVIDIA GTC 2022, Poster, Online, Mar. 2022.
8. 多田野寛人，倉本亮世，Group-wise 更新による Block GPBiCG 法の近似解精度改善，日本応用数理学会 2021 年度年会，オンライン，2021 年 9 月.

9. 石川翔大，多田野寛人，齋藤歩，画像再構成問題で現れる鞍点型連立一次方程式に対するブロック構造を利用した前処理の適用と性能評価，日本応用数理学会 2021 年度年会，2021 年 9 月.
10. 小林諒平, 藤田典久, 山口佳樹, 朴泰祐, 吉川耕司, 安部牧人, 梅村雅之,“GPU クラスタを用いた宇宙輻射輸送コード ARGOT の OpenACC 実装と性能評価”，第 183 回ハイパフォーマンスコンピューティング研究発表会, 2022 年 3 月 18 日
11. 小林諒平, 三浦賢人, 藤田典久, 朴泰祐, 天笠俊之,“FPGA 向け浮動小数点数型ソーティングライブラリの提案と実装”, SWoPP2021: 並列／分散／協調システムとディペンダブルコンピューティングおよび一般, 2021 年 7 月 21 日
12. 小林諒平, 三浦賢人, 藤田典久, 朴泰祐, 天笠俊之, “コンパクション処理を活用した正規パス問合わせアクセラレータの FPGA 実装”, 2021 年 6 月リコンフィギャラブルシステム研究会, 2021 年 6 月 9 日
13. 藤田典久，小林諒平，山口佳樹，朴泰祐，“HBM2 搭載 FPGA のための Addressable Cache を用いた HPC 向けメモリシステムの性能評価”，研究報告ハイパフォーマンスコンピューティング（HPC），2021-HPC-183(9)，2022/03.
14. 藤田典久，小林諒平，山口佳樹，朴泰祐，“FPGA における HPC アプリケーション向け HBM2 メモリシステムの提案と実装”，研究報告ハイパフォーマンスコンピューティング（HPC），2021-HPC-180(17)，2021/07.

(4) 著書、解説記事等

1. 今村俊幸，荻田武史，尾崎克久，片桐孝洋，須田礼仁，高橋大介，滝沢寛之，中島研吾 : ソフトウェア自動チューニング—科学技術計算のためのコード最適化技術—，森北出版, 320 pages, 2021 年 9 月

7. 異分野間連携・産学官連携・国際連携・国際活動等

異分野間連携（センター内外）

- 素粒子物理研究部門と Japan Lattice Data Grid（JLDG）の構築、運用に関して連携を行っている。
- 「初期天体形成シミュレーションにおける GPU+FPGA 連携プログラミング及び実行に関する研究」計算科学研究センター・宇宙物理研究部門・梅村グループとの共同研究
- 「FPGA を活用したグラフ処理アプリケーションの高速化に関する研究」計算科学研究センター・計算情報学研究部門 (データ基盤分野)・天笠グループとの共同研究
- 「地域気象コード City-LES の GPU 化に関する研究」計算科学研究センター・地球環境研究部門・日下グループとの共同研究

- 量子物性研究部門と SALMON2 の OpenACC による GPU 化に関する連携

産学官連携

国際連携・国際活動

- GPU・FPGA 協調プログラミングシステム MHOAT の開発において米国 Oak Ridge National Laboratory と共同研究
- FPGA 間通信システムに関する共同研究をドイツ・パダボーン大学と実施
- Cygnus における GPU・FPGA 協調 MD 計算に関する共同研究を米国ボストン大学と実施

8. シンポジウム、研究会、スクール等の開催実績

- International Workshop on FPGA for HPC Cluster 2021，組織委員長，IEEE Cluster2021 併設ワークショップ（オンライン），2021 年 9 月 7 日
- Gfarm シンポジウム 2021，東京（ハイブリッド），2021 年 10 月 29 日
- Gfarm ワークショップ 2022，那覇（ハイブリッド），2022 年 2 月 10 日

9. 管理・運営

組織運営や支援業務の委員・役員の実績

1. 朴泰祐：筑波大学情報環境委員会委員
2. 朴泰祐：理化学研究所客員主管研究員
3. 朴泰祐：HPCI コンソーシアム理事長
4. 朴泰祐：PC クラスタコンソーシアム理事
5. 朴泰祐：学際大規模情報基盤共同利用・共同研究拠点（JHPCN）運営委員
6. 朴泰祐：「富岳」成果創出加速課題領域総括
7. 朴泰祐：HPCI 計画推進委員会委員
8. 朴泰祐：「富岳」課題推進委員会委員
9. 高橋大介：筑波大学情報環境機構学術情報メディアセンター運営委員会委員
10. 高橋大介：理化学研究所客員主管研究員
11. 高橋大介：HPCI 利用研究課題審査委員会レビューアー
12. 高橋大介：HPCI 連携サービス運営・作業部会委員
13. 高橋大介：学際大規模情報基盤共同利用・共同研究拠点（JHPCN）課題審査委員
14. 建部修見：HPCI セキュリティインシデント即応委員会委員
15. 建部修見：HPCI 連携サービス運営・作業部会委員

16. 建部修見：HPCI 利用研究課題審査委員会レビューアー
17. 建部修見：情報通信研究機構協力研究員
18. 建部修見：東京工業大学学術国際情報センター共同利用専門委員
19. 建部修見：特定非営利団体つくば OSS 技術支援センター理事長
20. 小林諒平：理化学研究所計算科学研究センター客員研究員
21. 小林諒平：計算科学研究センター計算機システム運用委員会委員

10. 社会貢献・国際貢献

1. Taisuke Boku: Steering Committee Chair, International Conference on High Performance Computing in Asia-Pacific Region (HPCAsia)
2. Taisuke Boku: Steering Committee Member, IEEE Cluster
3. Taisuke Boku: Steering Committee Member, International Conference on Parallel Processing
4. Taisuke Boku: Scientific Committee Member, Supercomputing Frontier Europe 2021
5. Taisuke Boku: Program Vice Co-Chair of CCGrid2022
6. Daisuke Takahashi: International Conference on High Performance Computing in Asia Pacific Region (HPC Asia 2022) Proceedings Chair
7. Daisuke Takahashi: Special Session on Auto-Tuning for Multicore and GPU (ATMG) in Conjunction with IEEE 14th International Symposium on Embedded Multicore SoCs (MCSoC-21) Program Committee Member
8. Daisuke Takahashi: 8th International Workshop on Large-scale HPC Application Modernization (LHAM 2021) in Conjunction with 9th International Symposium on Computing and Networking (CANDAR'21) Program Committee Member
9. Daisuke Takahashi: The 21st International Conference on Computational Science and Its Applications (ICCSA 2021) Publicity Committee Member
10. Daisuke Takahashi: 2021 IEEE International Conference on Cluster Computing (Cluster 2021) Program Committee Member
11. Daisuke Takahashi: The 20th International Symposium on Parallel and Distributed Computing (ISPDC 2021) Program Committee Member
12. Daisuke Takahashi: The International Conference on Computational Science (ICCS 2021) Program Committee Member
13. Daisuke Takahashi: The 16th International Workshop on Automatic Performance Tuning (iWAPT 2021) Program Committee Member
14. 高橋大介：情報処理学会ハイパフォーマンスコンピューティング研究会幹事
15. 高橋大介：情報処理学会論文誌査読委員

16. 高橋大介：電子情報通信学会／情報処理学会 第 20 回情報科学技術フォーラム（FIT2021）担当委員
17. Osamu Tatebe: Technical Papers Track Chair, IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SC21)
18. Osamu Tatebe: Program Committee, ACM International Symposium on High Performance Distributed Computing (HPDC 2021)
19. Osamu Tatebe: Program Committee, IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2021)
20. Osamu Tatebe: Publicity Co-chair and Program Committee, IEEE International Conference on Cluster Computing (Cluster 2021)
21. Osamu Tatebe, Program Committee, International Supercomputing Conference 2021
22. Osamu Tatebe: Program Committee, ACM International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2022)
23. Hiroto Tadano: Publication Co-Chair, The 40th JSST Annual International Conference on Simulation Technology (JSST 2021)
24. 多田野寛人 : 日本シミュレーション学会「非線形現象の高性能数値解析技術研究委員会」　副委員長
25. 多田野寛人 : 日本応用数理学会「行列・固有値問題の解法とその応用」研究部会　運営委員
26. Hiroto Tadano: Local Scientific Program Committee, 10th International Congress on Industrial and Applied Mathematics (ICIAM 2023)
27. 小林諒平：電子情報通信学会 コンピュータシステム研究専門委員会 幹事補佐
28. 小林諒平：電子情報通信学会 コンピュータシステム研究会専門委員
29. 小林諒平：電子情報通信学会 リコンフィギャラブルシステム研究会専門委員
30. 小林諒平：電子情報通信学会 英文論文誌編集委員
31. 小林諒平：電子情報通信学会 ISS ソサイエティ誌編集委員
32. 小林諒平：情報処理学会 ハイパフォーマンスコンピューティング研究会運営委員
33. 小林諒平：SWoPP 2021 実行委員 (コンピュータシステム研究会)
34. 小林諒平：SWoPP 2021 組織委員 (副組織委員長)
35. 小林諒平：xSIG 2021 プログラム委員
36. Ryohei Kobayashi：HPCAsia2022 Digital Chair
37. Ryohei Kobayashi：HEART2021 Publication Chair
38. Ryohei Kobayashi：HPCFPGA2021 Organizing Deputy Co-Chairs
39. Ryohei Kobayashi：HPCFPGA2021 Program Committee Member

40. Ryohei Kobayashi：FTAC 2021 Program Committee Member
41. Ryohei Kobayashi：CANDAR 2021 Program Committee Member
42. Ryohei Kobayashi：CANDAR 2021 CSA workshop Program Committee Member
43. Ryohei Kobayashi：COOL Chips 24　Program Committee Member

11. その他

海外長期滞在、フィールドワークなど