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

准教授　　塙　敏博（客員，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　12 名、学類生　6 名

学内共同研究員

安永　守利，和田　耕一，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が挙げられる。

- 多重複合型演算加速スーパーコンピュータ Cygnus のシステムソフトウェア，及びアプリケーションソフトウェアの開発
- アプリケーション分野との共同研究による演算加速アプリケーション開発
- 「富岳」における数値計算ライブラリ開発及び性能評価
- 高速ストレージシステムと並列 I/O の性能及び拡張性の向上
- ノードローカルストレージを用いたシステムソフトウェアの研究
- 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
- 大規模データ解析アプリケーション、深層学習フレームワークの高速化
- 複数右辺ベクトルを持つ行列計算の高速化と精度向上

3.研究成果

[1] OpenACC による GPU+FPGA 混合プログラミングフレームワーク（朴）

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のが現状である。

一方，我々は Cygnus をプラットフォームとする CHARM(Cooperative Heterogeneous Acceleration with Reconfigurable Multidevices, 再構成可能な多重複合型演算加速デバイスによる高性能計算)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 Acceleration 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することで，CHARM 向けの実行コードとランタイムシステムを実現する。しかし，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。

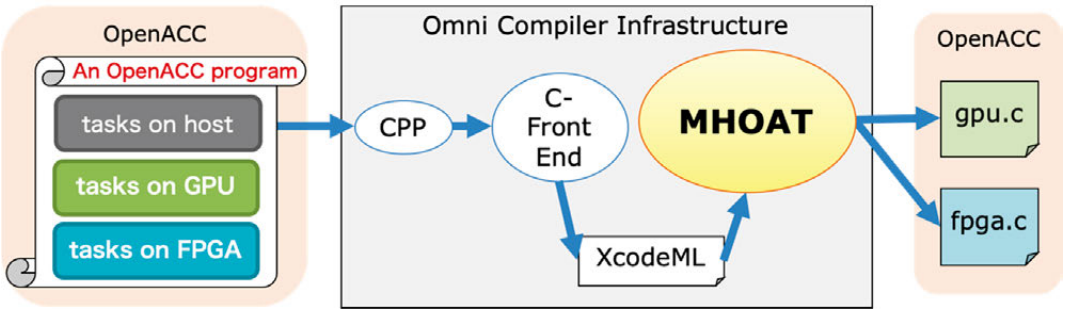


図 1 MHOAT の処理フロー

MHOAT によって FPGA と GPU という 2 種類のデバイス用に分離された OpenACC コードは，それぞれのバックエンドコンパイラ（GPU: PGI Compiler, FPGA: OpenARC）によってオブジェクトが生成され，最終的に gcc によるリンクを経て，1 つのホストプログラムから FPGA と GPU の各デバイス用カーネルプログラムを起動可能なオブジェクトが完成する。

2019 年度の研究では，GPU で行列計算（matrix-matrix multiply）を行い，その結果生成された行列を係数行列とする連立一次方程式の求解を FPGA 上で CG 法で行うというテストコードを MHOAT によって処理し，PGI Compiler と OpenARC をそれぞれ呼び出して全体を連携する処理が可能であることを示した。2020 年度の研究では，MHOAT をより高度な実用的プログラムに適用可能とするため, プロトタイプで問題となっていたいくつかの問題に取り組んだ。主要な改造は，FPGA にオフロードされる部分について，そのままコードを書き込むことができず，オフロードのための directive（"#pragma acc kernels" 等）の直後に，処理コードを閉じ込めた関数を一旦書き, その関数の中で実際の処理を書く必要があったこと，関数内で必ず特定のヘッダを include しなければいけなかったこと等である。これらの主要因は, FPGA 用のバックエンドコンパイラとして用いている OpenARC の作りの問題で，ユーザプログラムのコードは C 言語なのに対し，そこから生成される OpenCL（Intel 社の FPGA 用の高位合成言語は OpenCL が標準である）プログラムは C++に準拠しているという複雑な特性を持っていることによる。この問題は ORNL のグループと協議しているが，解決が困難で現在も検討中である。

また, 昨年度までの簡単なテストコードに加え，本格的なアプリケーションコードを対象とするため，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 CHARM 向けの宇宙初期天体シミュレーションコード ARGOT の移植を進めている。ARGOT は本研究部門の小林助教を中心とする研究により，CHARM による GPU と FPGA の組み合わせにより，最大で GPU のみの計算に比べ 17 倍もの高速化が可能であることが示されている。よって，MHOAT でこのコードがコンパイルでき，そのコードが十分な効率で実行可能となれば CHARM コンセプトの有効性を示す重要な例となる。本研究も現在継続中である。

2020 年度の成果として，MHOAT に関する査読付き論文を国際会議 HLPP2020 において発表した[2]。また，その基本となる研究を 2019 年度に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において発表した成果が認められ, 2020 年度の情報処理学会 CS 領域奨励賞が授与された。

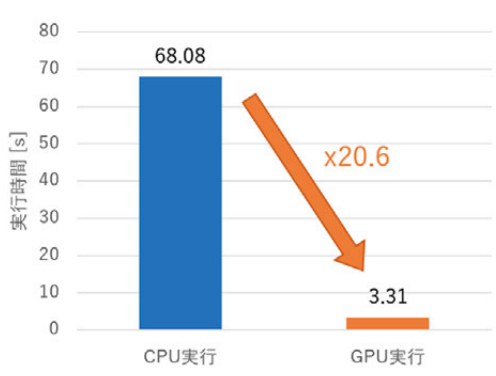
[2] 気象アプリケーション City-LES の GPU 向け高速化（朴）

GPU と FPGA の融合による演算加速のアプリケーションの一つとして，本センター地球環境研究部門の日下教授グループで開発中の都市気象モデルシミュレーション City-LES の

GPU 化を進めている。2019 年度までの研究で，これまで CUDA で開発してきたコードを全面的に OpenACC 化し，今後の OpenACC による統一的プログラミングへの準備を進め，単純な力学過程に基づく流体コードを主とする部分について Cygnus 上で最大 32 ノード（128GPU, NVIDIA Tesla V100）を用いて加速化し，weak scaling において CPU のみの場合の約 9 倍の性能，strong scaling においても約 6 倍の性能が得られていた。

2020 年度の研究では，その性能をさらに向上させ，また対象をこれまでの単純なモデル（建物なし，地表植生なし）ではなく，複雑構造の建物，植生，さらに太陽の移動に伴う太陽光の移動に対応した，完全な City-LES のフルモデルを GPU 化する研究を行った。昨年度の研究成果により，CUDA ではなく OpenACC のみで記述しても，少なくとも NVIDIA Tesla V100 GPU では速度はほとんど低下しないことと，特定のカーネルが CPU から大きく速度向上していない場合でも，そのカーネルを CPU に残すことで CPU・GPU 間のデータ転送時間の方がボトルネックとなり，返って性能が低下することがわかっており，このコンセプトの下で詳細モデルの GPU 化を進めた。

まず，昨年度の研究において少数の CUDA カーネルが積み残されていたので，これを完全に OpenACC 化した。その上で，複雑な建物構造に対応するため，空間メッシュの計算が単純モデルでは 2 次精度の中央差分だったものを 5 次精度の風上差分とし，これを GPU 化した。対応する関数は advection と呼ばれるカーネルで，その基本的なループ構造は単純で，OpenACC による GPU 化に非常にうまく対応した。結果として，advection カーネルの GPU 実行性能は CPU 実装に比べ 20.6 倍に高速化された。実測は，Cygnus の 1 ノード上で 4 つの MPI プロセスを走らせて行った。CPU 版はノード上の 24 のコアを MPI プロセスに均等分割し，プロセス当たり 6 スレッドで実行した。GPU 版は各 MPI プロセスに GPU を 1 つずつ割り当てた。従って，この性能向上は Cygnus で同一のリソース（計算ノード）を用いた場合に得られる結果である。性能向上の結果を図 2（左）に示す。結果は 200 タイムステップの時間発展における所要時間である。



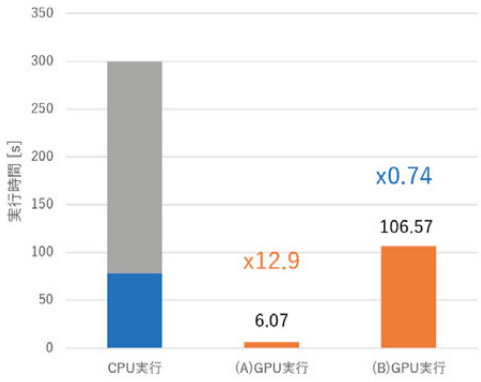


図 2 （左）advection カーネルの GPU 化による性能向上，（右）surface_driver カーネルの GPU 化による性能向上（中央は人工排熱なし，右は人工排熱ありのデータによる）

次に，太陽光の移動に伴うレイトレーシング処理を含むカーネルである surface_driver の GPU 化を行った。このカーネルの基本ループでは配列への reduction 処理が発生するため，ループ内に一時的なスカラー変数を設け，この処理低下を回避した（なお OpenACC 2.7 以降では redcution の対象として配列を扱えるようになったため，この工夫は不要になった）。さらに，ループ内にデータ依存性が一部存在するため，その部分はやむを得ず逐次実行 directive ("!$acc loop seq") を含め，結果の正しさを保証している。これらを踏まえて GPU 化を行った結果，人工排熱を考慮しないデータでは GPU 版の性能は CPU 版の 6.07 倍に高速化された。一方，人工排熱を考慮したデータでは，GPU 版の性能は CPU 版の 0.74 倍となり，速度が低下してしまった（図 2（右））。しかし，GPU 版が CPU 版よりも少し性能が低い場合でも，そのカーネルを CPU に残した方が返ってデータ転送オーバヘッドを招いて性能が逆転するというケースを昨年度にも経験しており，surface_driver でもその場合のデータ転送時間を測定した結果，予想通りカーネルの実行時間を遥かに上回る時間（200 回の時間反復で GPU 実行時間の 2 倍以上，図 2（左）の灰色部分）がかかることがわかった。従って，いずれの場合でも GPU でカーネルを実行し，その前後の GPU 上のカーネルとの間で GPU メモリを共有したままの方が総合的な性能が高いことがわかった。

以上の GPU 化の結果，City-LES 全体での速度向上は，CPU のみの実行に比べ，advection 及び surface_driver を CPU 側に残した場合では 1.49 倍に留まるが，両カーネルとも GPU 化した場合は 4.3 倍に高速化されることがわかった。これらの性能向上を図 3 に示す（中央は両カーネルを CPU に残した場合，右端は全てを GPU 化した場合）。今年度の研究成果は国際会議 ISPDC2020 において査読付き論文として発表された[1]。

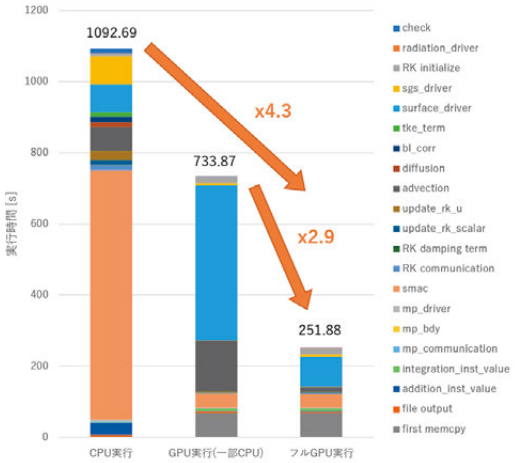


図 3 City-LES の詳細モデル対応コードの完全 GPU 化による性能向上

[3] 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（高橋）

高速 Fourier 変換（fast Fourier transform、以下 FFT）は、科学技術計算において今日広く用いられているアルゴリズムである。並列三次元 FFT においては、三つの次元（x、yおよびz軸）のうちの一つの次元のみ（例えばz軸）のみを分割した場合、z軸におけるデータ数は MPI プロセス数以上となる必要があり、問題サイズに制約が生じることになる。この問題に対処する方法として、二次元分割を用いた並列三次元 FFT が提案されている。しかし、OpenMP による通信スレッドを導入した計算と通信のオーバーラップを用いた二次元分割による並列三次元 FFT の自動チューニングはまだ提案されていないのが現状である。

そこで、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を自動チューニングし性能評価を行った結果について述べる。

データ数Nを$N = N_1 \times N_2 \times N_3$とし、MPI プロセスが$P \times Q$の二次元にマッピングされるとする。$P \times Q$個の MPI プロセスを持つ分散メモリ型並列計算機では、三次元配列$x(N_1, N_2, N_3)$は二番目のN_2次元と三番目のN_3次元に沿って分散される。N_2がPで割り切れ、N_3もQで割り切れる場合、$N_1 \times (N_2/P) \times (N_3/Q)$個のデータが各 MPI プロセスに分散される。図 4 は、y軸とz軸の初期データを二次元分割した並列三次元 FFT を示している。

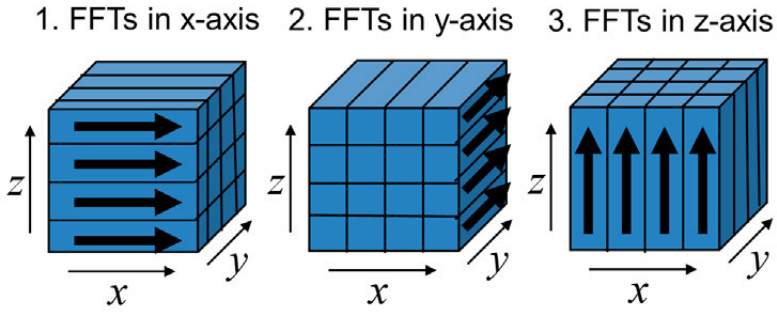


図 4 二次元分割を用いた場合の並列三次元 FFT

計算と通信のオーバーラップには、MPI 非同期通信が広く使われているが、OpenMP による通信スレッドを導入した計算と通信のオーバーラップ手法が提案されているため、これを用いた。

図 5 は、パイプライン化された計算と通信のオーバーラップを示している。図 5 において、NDIV は計算と通信のオーバーラップの分割数である。

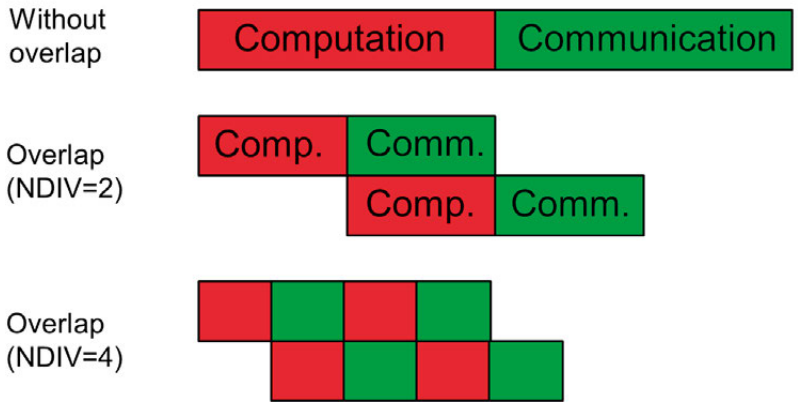


図 5 パイプライン化された計算と通信のオーバーラップ

二次元分割を用いた並列三次元 FFT の計算量と通信量のオーバーラップを自動チューニングする手法を提案する。自動チューニングは 3 つのステップで構成されている。(1) MPI プロセスグリッド（$P \times Q$）の選択、(2) 計算と通信のオーバーラップの分割数 NDIV の選択、(3) ブロックサイズ NB の選択。

通常、MPI プロセスの総数が$P \times Q$となるようなPとQは$P \approx Q \approx \sqrt{PQ}$となるように選択する。$P$と$Q$の全ての組み合わせを検索することで、最適な$P$と$Q$の組み合わせを調べることができる。MPI プロセス数$P \times Q$が 2 のべき乗の場合、PとQの全ての組み合わせを調べたとしても、探索空間は$\log_2(PQ) + 1$となる。

また、計算と通信のオーバーラップの分割数を増やすと、オーバーラップ率も高くなる。一方で、メッセージサイズを小さくすると、全対全通信の性能が低下する。このように、オーバーラップ率と全対全通信の性能はトレードオフの関係にある。

オリジナルの FFTE 7.1alpha のデフォルトのオーバーラップパラメータは NDIV=4 である。今回の実装では、オーバーラップパラメータ NDIV を 1、2、4、8、16 の間で変化させている。

オリジナルの FFTE 7.1alpha のデフォルトのブロッキングパラメータは NB=32 である。最適なブロックサイズは問題サイズに依存する可能性があるが、ブロックサイズ NB も変化させることができる。今回の実装では、NB を 8、16、32、64 の間で変化させている。

性能評価にあたっては、FFTE 7.1alpha（オーバーラップなし）、FFTE 7.1alpha（オーバーラップあり、NDIV=4）、FFTE 7.1alpha に自動チューニング（AT）を行ったもの、および FFTW 3.3.9 と性能を比較した。測定に際しては、weak scaling と strong scaling における経過時間を測定した。なお、FFT の計算は倍精度複素数で行い、三角関数のテーブルはあらかじめ作り置きとしている。入力と出力では同じデータ分散を使用した。FFTW では、” measure” planner を使用した。Xeon Phi クラスタとして、最先端共同 HPC 基盤施設（JCAHPC）に設置されている Oakforest-PACS（8208 ノード）の 1～2048 ノードを用いた。コンパイラは FFTE に対しては Intel Fortran Compiler version 19.0.5.281 を、FFTW に対しては Intel C Compiler version

19.0.5.281 を用いた。コンパイルオプションは-O3 -xMIC-AVX512 -qopenmp を指定した。MPI ライブラリは Intel MPI 2019.5.281 を用いた。各ノードあたりの MPI プロセス数は 4、各 MPI プロセスあたりのスレッド数は 17 に設定し、環境変数 KMP_AFFINITY=balanced を設定して flat/quadrant モードで MCDRAM のみを用いて実行した。$N = 2^m$点 FFT の GFlops 値は $5N \log_2 N$より算出している。

並列三次元 FFT の weak scaling 性能（$N = 256 \times 512 \times 512$ ×MPI プロセス数）を図 6 に示す。図 6 に示すように、自動チューニングを用いた FFTE 7.1alpha が最も高速であることが分かる。

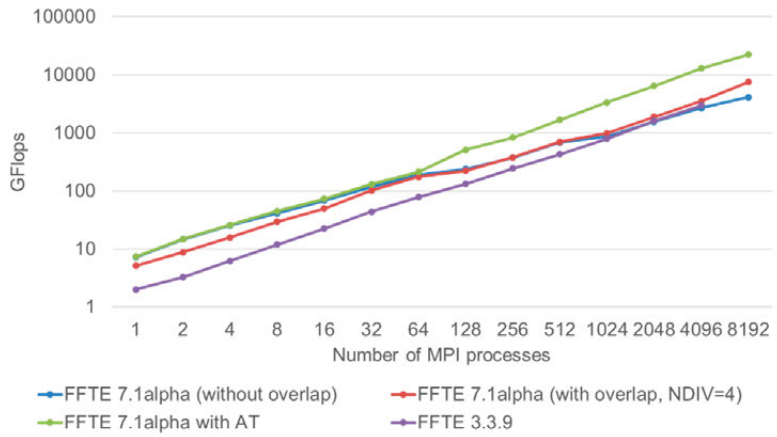


図 6 並列三次元 FFT の weak scaling 性能（$\boldsymbol{N = 256 \times 512 \times 512}$ ×MPI プロセス数）

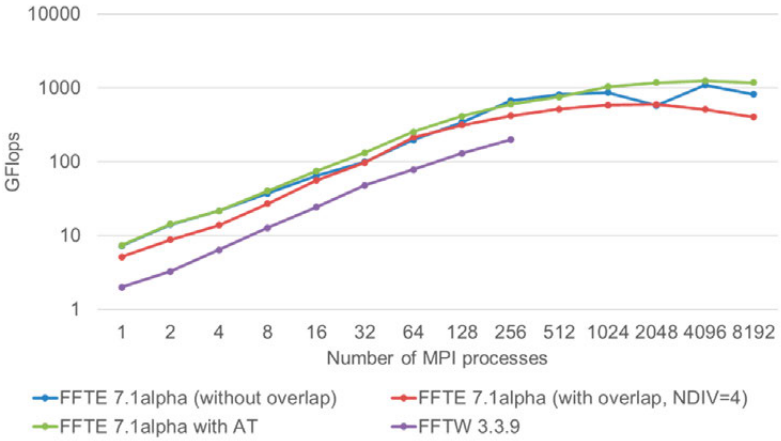


図 7 並列三次元 FFT の strong scaling 性能（$\boldsymbol{N = 256 \times 512 \times 512}$）

並列三次元 FFT の strong scaling 性能（$N = 256 \times 512 \times 512$）を図 7 に示す。図 7 から分かるように、MPI プロセスが 256 および 512 の場合を除き、自動チューニングを用いた FFTE 7.1alpha は FFTE 7.1alpha（オーバーラップなし）よりも高速である。

[4] Intel AVX-512IFMA 命令を用いた複数の Montgomery 乗算の高速化（高橋）

乗算剰余演算は計算機代数や暗号などの分野で広く用いられている演算の一つである。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乗算剰余演算を行うことができることが知られている。Intel Advanced Vector Extensions 512（Intel AVX-512）は 512 ビットのベクトル命令セットである。ベクトル命令を用いる Montgomery 乗算アルゴリズムや、Intel AVX-512IFMA（Integer Fused Multiply-Add）を用いた高速な二乗剰余演算が提案されている。複数の Montgomery 乗算については、Intel Streaming SIMD Extensions 2（Intel SSE2）命令を用いた実装や Cell プロセッサにおける実装が提案されているが、Intel AVX-512IFMA 命令を用いた実装は知られていない。

Montgomery 乗算は通常数百ビット以上の整数に対して行われるが、本研究では 52 ビット以下の場合を考える。このようなビット数に対する複数の Montgomery 乗算は、モジュラー高速 Fourier 変換（FFT）アルゴリズムで用いられている。

Algorithm 1 修正 Montgomery 乗算アルゴリズム

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.

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半分と上半分をそれぞれ生成する vpmadd52luq および vpmadd52huq 命令をサポートする。これらの下半分と上半分は 64 ビットのアキュムレータに加算される。Algorithm 1 に修正 Montgomery 乗算アルゴリズムを示す。Montgomery 乗算のオペランドが 52 ビット以下の場合、Intel AVX-512IFMA 命令を用いると Algorithm 1 において$\beta = 2^{52}$とすることができる。Algorithm 1 の 1 行目において、ABをβで割った余りは vpmadd52luq 命令を用いてABの下位 52 ビットとして計算する。2 行目において、μCをβで割った余りは vpmadd52luq 命令を用いてμCの下位 52 ビットとして計算する。3 行目において、ABとqNの上位 52 ビットは vpmadd52huq 命令を用いて計算する。4 行目と 5 行目における$q \neq 0$の場合の条件付き増分$C + 1$は、最小値演算と符号なし整数値qとCの加算$C + \min(q, 1)$で置き換えることができる。この最小値演算$\min(q, 1)$は、vpminuq 命令を用いて行う。Algorithm 1 の 6 行目と 7 行目の条件付き減算も、符号なし整数のラップアラウンドを用いると vpminuq 命令を用いて実行することができる。

Intel AVX-512 組み込み関数を用いたベクトル化された 52 ビット整数の複数の Montgomery 乗算を図 8 に示す。このプログラムでは、Intel AVX-512 組み込み関数_mm512 madd52lo_epu64()、_mm512_madd52hi_epu64()、および_mm512_min_epu64()は、それぞれ vpmadd52luq、vpmadd52huq、および vpminuq 命令に対応している。このプログラムは、ベクトル長 VLEN が 8 で割り切れると仮定している。もし VLEN が 8 で割り切れない場合、剰余ループを実行する必要がある。

```
#include <stdint.h>
#include <immintrin.h>

void vmulmod52(uint64_t *c, uint64_t *a, uint64_t *b, uint64_t *N, uint64_t *mu)
/*  c[:] = (a[:] * b[:] * 2^-52) mod N[:]. */
    We need mu[:] = -N[:]^-1 mod 2^52. */
{
  __m512i q, t;
  int i;

  for (i = 0; i < VLEN; i += 8) {
    t = _mm512_madd52lo_epu64(_mm512_set1_epi64(0), _mm512_load_epi64(&a[i]),
                              _mm512_load_epi64(&b[i]));
    q = _mm512_madd52lo_epu64(_mm512_set1_epi64(0), t, _mm512_load_epi64(&mu[i]));
    t = _mm512_madd52hi_epu64(_mm512_min_epu64(q, _mm512_set1_epi64(1)), q,
                              _mm512_load_epi64(&N[i]));
    t = _mm512_madd52hi_epu64(t, _mm512_load_epi64(&a[i]), _mm512_load_epi64(&b[i]));
    _mm512_store_epi64(&c[i], _mm512_min_epu64(t, _mm512_sub_epi64(t,
                                                _mm512_load_epi64(&N[i]))));
  }
}
```

図 8 Intel AVX-512 組み込み関数を用いたベクトル化された 52 ビット整数の複数の Montgomery 乗算

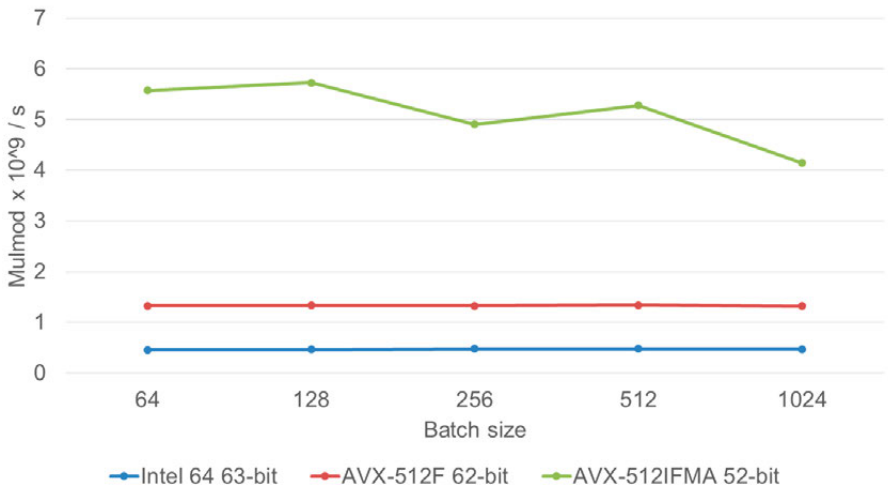


図 9 さまざまな命令セットを用いた複数の Montgomery 乗算の性能

性能評価にあたっては、Intel AVX-512IFMA 命令を用いた複数の 52 ビット Montgomery 乗算、Intel AVX-512F (Foundation) 命令を用いた複数の 62 ビット Montgomery 乗算、および Intel 64 命令を用いた複数の 63 ビット Montgomery 乗算の性能の比較を行った。

Montgomery 乗算$C = AB\beta^{-1} \bmod N$の計算において、Nは1から$2^{52}-1$の範囲の奇数の乱数であり、AとBは、$0 \leq A,\ B < N$の範囲の乱数である。バッチサイズを 64 から 1024 に変化させて Montgomery 乗算を 100 万回実行し、その平均の経過時間から 1 秒あたりの Montgomery 乗算回数（Mulmod $\times 10^9$/s）を算出した。

評価環境として、Intel Core i3-8121U の 1 コア、1 スレッドを用いた。コンパイラは Intel C compiler 190.5.281 を用い、コンパイルオプションは icc -O3 -xCANNONLAKE を用いた。

Intel 64、Intel AVX-512F、および Intel AVX-512IFMA 命令を用いた複数の Montgomery 乗算の性能を図 9 に示す。図 9 からバッチサイズが 128 の場合、Intel AVX-512IFMA を用いた提案した実装は、Intel Core i3-8121U プロセッサにおいて Intel 64 および Intel AVX-512F 命令を用いた実装と比較して、それぞれ約 12.22 倍および約 4.30 倍高速であることが分かる。

[5] ノードローカルストレージを活用する MPI-IO の研究（建部）

MPI-IO は HPC アプリケーションにおいて標準的に利用される並列 I/O インターフェースである。並列プロセスからの並列 I/O に用いられる。一方、並列ファイルシステムに対するアクセス性能はアクセスパターンにより変わり、特によく利用される単一共有ファイル（single shared file）パターンでは性能が低下することが知られている。単一共有ファイルパターンとは、並列プロセスが単一のファイルにアクセスするパターンである。本研究では、この単一共有ファイルパターンにおいて、ノードローカルストレージを活用してストレージ性能を向上するための研究を行う。

ノードローカルストレージは、計算ノードのローカルストレージである。ジョブが投入され、計算ノードが割当てられ、ジョブの実行が開始してから終了するまでしか利用することはできない。これまで、ローカルストレージは、プロセス毎ファイル（file per process）パターンに対し、単純な write back cache として利用される例はある。また、単一共有ファイルでは、計算ノードのローカルストレージにより構成されるアドホック分散ファイルシステムである BurstFS, BeeOND, Gfarm/BB, GekkoFS を用いた例がある。ただし、BeeOND や GekkoFS などストライピングにより分散させるファイルシステムでは書込み性能をあまり向上させることはできない。BurstFS は並列書込みに特化したファイルシステムであり、読込性能を犠牲にしている。Gfarm/BB では、単一共有ファイルは 1 ノードに格納されるためスケーラビリティの問題がある。本研究では、これらの問題に対し MPI-IO 層での解決を図り、NS-MPI の提案を行うものである。Gfarm/BB は、ファイル作成についてはノードローカルストレージに直接書き込む特性を持っている。そのため、プロセス毎ファイルの場合は非常に高い性能を発

揮する。一方で、単一共有ファイルの場合はファイルを分割しないため全プロセスが同一ファイルをアクセスすることとなり性能が低下する。この特性を利用し、NS-MPI においてファイル分割を行うことで、単一共有ファイルへのアクセスに対し、プロセス毎ファイルのアクセス性能を実現することを目指す。

これを実現するため、NS-MPI では sparse segments という新しいファイルフォーマットを提案した。Sparse segments は、単一共有ファイルを分割したものであり、分割したファイルはスパースファイルとなっている。スパースファイルとは、データを書き込んだ領域にはデータが存在し、書き込んでいないところは穴（hole）と呼ばれる領域となるファイルである。穴の部分を読み込むと 0 を読むこととなる。スパースファイルでは穴の部分は保持されないためストレージ容量は節約される。NS-MPI では、単一共有ファイルへのアクセスに対し、各プロセスが sparse segments フォーマットのファイルとしてアクセスする。書込みについては、各プロセスが各自のスパースファイルへの書込みとなるため、プロセス毎ファイルへの書込みと同様の性能を実現することが可能となる。読込みについては、どのスパースファイルに書き込まれたか調べることが必要となるが、書込み時に書き込んだ範囲を管理しておき、そのデータを用いることにより調べる。これらの設計に基づき、NS-MPI の実装を行い、東工大の TSUBAME3.0 スーパコンピュータにより評価を行った。

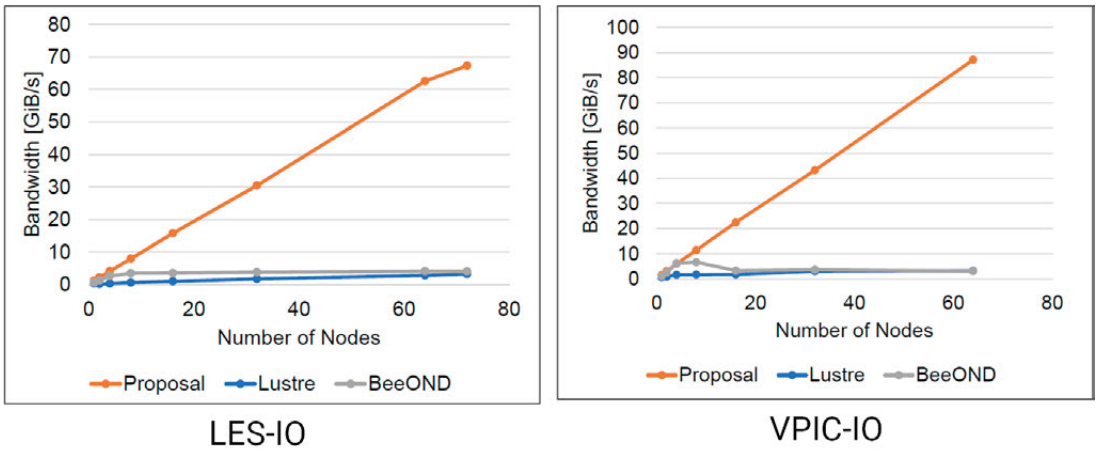


図 10 LES-IO と VPIC-IO における提案手法（NS-MPI）と Lustre、BeeOND との性能比較

アプリケーションベンチマークである LES-IO と VPIC-IO における評価結果を図 10 に示す。この図は、計算ノード数を増やしたときのストレージ性能を示している。従来の BeeOND を用いた手法では、計算ノード数が増加しても性能は頭打ちとなるが、提案手法ではスケーラブルに性能が向上していることが分かる。なお、Lustre はノードローカルストレージを用いず、直接並列ファイルシステムをアクセスしたときの性能である。

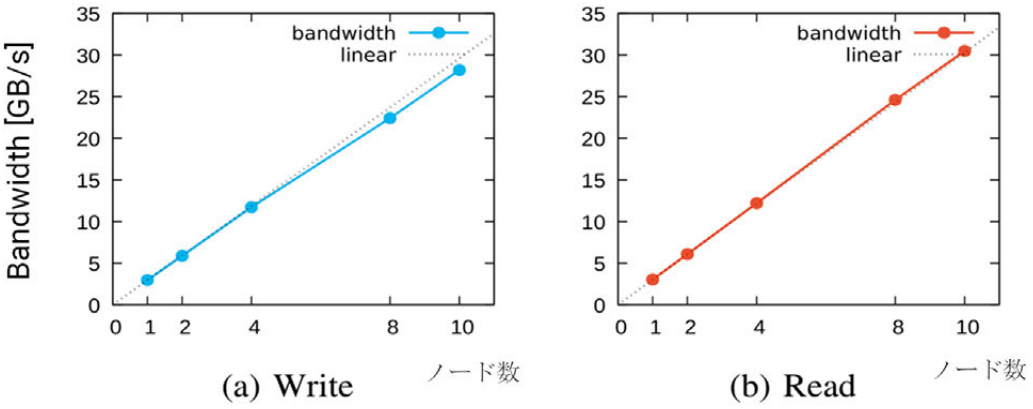


図 11 IOR による単一共有ファイルへの書込、読込性能

図 11 にベンチマーク IOR における単一共有ファイルへの書込、読込性能を示す。この性能評価は筑波大学の Cygnus スーパコンピュータで行った。書込みだけではなく、読込みについても計算ノード数に比例して性能が向上していることが分かる。本成果は IEEE International Workshop on High-Performance Storage および、6th IEEE International Conference on Data Science and Systems (DSS)において発表を行った。

[6] 大規模メタゲノムデータ解析の研究（建部）

メタゲノム解析は環境サンプルなどから抽出したゲノムを網羅的に解析するものであり，特定のゲノムの培養が難しいケースに対しても解析が可能となる。シーケンサの処理能力の向上により，クエリデータ，参照データベースの双方のデータ量が増えている。これにより，メモリ量，演算速度の問題により，1 ノードでの処理が難しくなってきた。この問題を解決するために，これまでクエリデータを分割して分散解析する方法が提案されている。ところが，参照データベースのデータ量の増加により，参照データベースについても分割することが必要となってきた。本研究では，クエリデータと参照データベースの両方を分割し，分散データ解析する手法を提案する。両データを分割することにより，分散データ解析した結果の集計処理が複雑となるが，本研究では Pwrake ワークフローシステムを用いることにより解決を図った。また，Gfarm/BB により，ノードローカルストレージを効率的に用いることで大規模処理時においても，I/O 性能がオーバヘッドとならないこと，演算性能がスケールすることを示した。

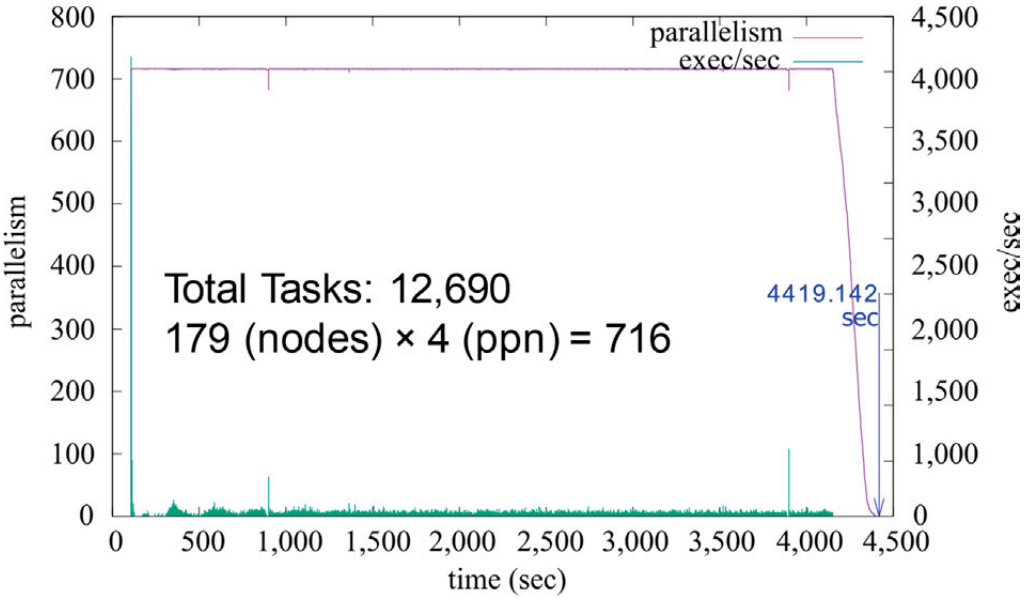


図 12 TSUBAME3.0 による大規模実行の模様

図 12 に東工大の TSUBAME3.0 スーパコンピュータの 180 ノードを用いた大規模実行の模様を示す。1 ノードは Gfarm/BB のメタデータサーバ，Pwrake ワークフローシステムのマスタサーバとして用い，残りの 179 ノードで演算を行っている。各ノードに GPU が 4 枚あるため，各ノードでは 4 プロセスを実行している。この実行の模様は最も時間を費やすアライメントプロセスを示しているが，実行開始から利用可能な計算ノードをフルに使っていることが分かる。62 GB のデータベースと 71 GB のクエリデータのメタゲノム解析を 180 ノード用いて前処理，後処理を合わせて 2 時間弱で実行することが可能であった。本規模は我々が知る限りで最大のメタゲノム解析である。本成果は情報処理学会論文誌コンピューティングシステム(ACS)において発表した。

[7] 分散ファイルシステム及びグリッド・クラウド技術に関する研究（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。

本年度は，フェイルオーバ機能の高度化，オブジェクトストレージインターフェース対応を行った。

Gfarm ファイルシステムは，メタデータ，ファイルデータが冗長化されており，単一障害についてはフェイルオーバにより隠蔽することが可能である。メタデータについては，マスターメタデータサーバと複数のスレーブメタデータサーバが動作しており，マスターメタデータサーバに障害が発生した時，各メタデータサーバのコミットログを確認し，最も新しいロ

グをコミットしているサーバがマスターメタデータサーバに昇格する。一方、Gfarm ファイルシステムはファイルシステムのインターフェースを提供しているため、ファイルを延々とオープンし続けることが可能であり、フェイルオーバの前後に、異なる複数のユーザが同一ファイルに書込みを行うなど、競合する操作が行われたときに複数のバージョンのファイルが作成されてしまう問題があった。本問題が生じるのは、フェイルオーバによってファイルのオープン状態が失われてしまうためである。そのため、スレーブサーバに対し、ファイルのオープン状態についても情報を共有するように改修を行った。これにより、フェイルオーバが行われてもファイルオープンの状態が引き継がれるようになり、問題を解決することができた。

オブジェクトストレージインターフェースについては，Society 5.0、IoT などで重要度が増している。そのため、デファクトスタンダードとなっているアマゾンウェブサービス（AWS）において用いられている S3 互換のインターフェースに対応するため、概念設計、詳細設計を行った。

図 13 Gfarm の S3 インターフェースにおける管理画面の例

図 13 に S3 インターフェースを用いるための管理画面を示す。これらはスマートフォンからも利用可能となっている。まず、ユーザ名とパスワードによりログインする。その後、S3 サーバの起動を行い、S3 サーバのアクセスに必要なアクセスキーとシークレットキーを確認することができる。データは複数ユーザ、グループで共有することができ、一番右の図は共有設定を行うためのページの例である。本成果については Gfarm ワークショップにおいて発表した。

[8] CUDA アプリケーションのチェックポイントに関する研究（額田）

GPU (Graphics Processing Unit)はもともと画像処理用に多数の演算器を搭載するアクセラレータであるが，汎用計算にも用いることができるプログラマビリティを備えている。そのコストパフォーマンスや電力効率の高さで注目され，今や多くのスパコンで採用されている。GPU によりアプリケーションが高速化されてきたが，その結果としてより大規模な計算を行うようになり，実行時間はむしろ長くなる場合も多い。スパコンなどの共用システムでは通常ユーザ間で利用機会が公平になるようにするために実行時間制限がある。この制限を超えるような実行を行いたい場合には，後から再開できるように実行の途中でアプリケーションの状態をファイル等に保存するチェックポイントという技術が使用される。

CPU のみを使用するアプリケーションのチェックポイント技術は確立されており，代表的な BLCR や DMTCP といったソフトウェアはアプリケーションを一切修正することなくチェックポイントを実現できる。これらは基本的に CPU のメモリ上のデータを保存しているが，GPU の状態は CPU のメモリ上にはないため GPU アプリケーションに対応していない。そこでチェックポイント処理に入る前後で、GPU の状態を CPU のメモリに回収して一緒に保存する前処理と，GPU の状態を復元する後処理を追加することによって GPU アプリケーションにも対応できるように拡張する NVCR を提案する。

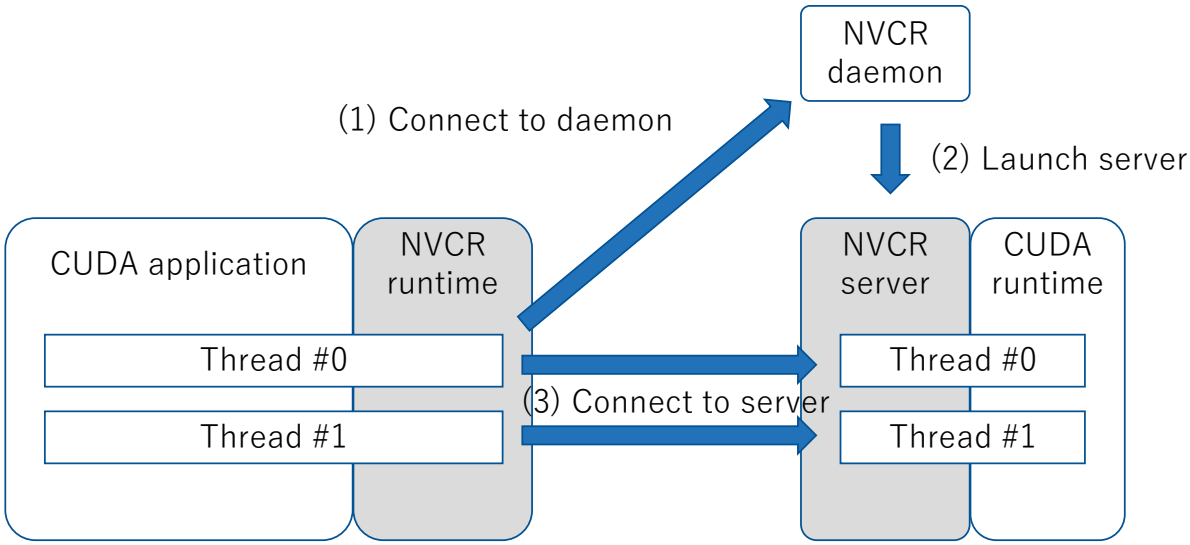


図 14 複数のプロセスで実装される NVCR の全体像

図 14 に NVCR の全体像を示す。左側にある GPU アプリケーションのプロセスに加えて，NVCR サーバと NVCR デーモンで構成される。NVCR では GPU アプリケーションは NVCR サーバ経由で GPU にアクセスする。NVIDIA が提供する CUDA ランタイムライブラリは GPU の初期化を２回以上行うことを想定しておらず，このためチェックポイントファイルからの再開時に GPU の初期化が行われないという問題がある。これを回避するために GPU へのア

クセスを行う使い捨ての NVCR サーバのプロセスを導入し，再開時には別のサーバプロセスを起動しなおす方式を採用する。

GPU アプリケーションは CUDA ランタイムライブラリで実装される専用の API 関数を用いて GPU を制御する。GPU アプリケーションに読み込ませる NVCR ランタイムライブラリはこれらの API 関数の呼び出しを全て NVCR サーバに伝達すると同時にモニタリングして，チェックポイントの前処理で保存するべき GPU 側のリソースの情報、及び実行を再開するときに GPU 側のリソースを再現するために必要な情報の収集を行う。

NVCR のランタイムライブラリが割り込むことによって追加のプロセス間通信が必要になり実行時間増加が懸念される。しかし GPU を操作する API 関数ではそもそも GPU という外部デバイスとの通信が必要である。CPU と GPU の間でデータのコピーを行う際にも GPU アプリケーションと NVCR サーバの間でプロセス間共有メモリを活用することによりコピー回数を増加させないことが可能である。GPU による高速化が大きく期待できるアプリケーションにおいては API 関数の遅延の影響を受けにくくホスト CPU との通信も少ない傾向にあり，NVCR による実行時間オーバヘッドは限定的であると言える。

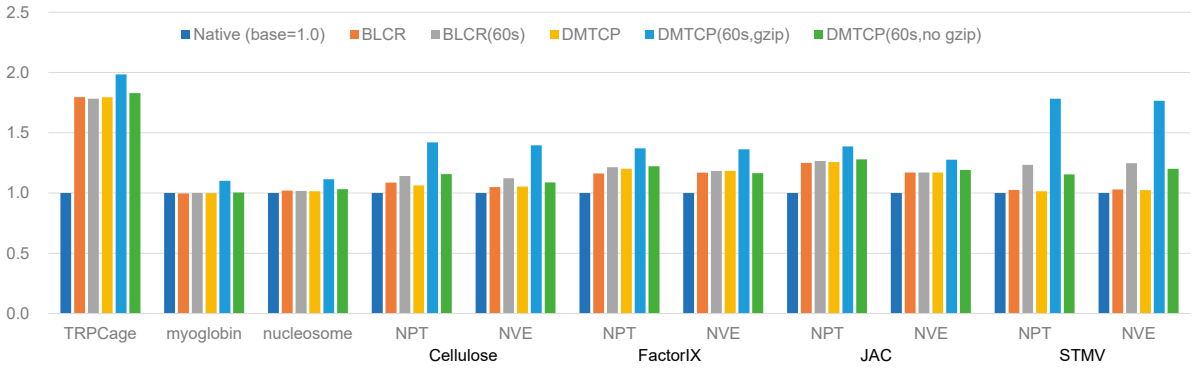


図 15 Amber による NVCR の実行時間オーバヘッド評価

図 15 は分子動力学シミュレーションソフトウェア Amber18 によるシミュレーション実行時間のオーバヘッドの評価である。NVCR を使用しない場合を 1.0 とする相対的な実行時間を示している。Amber Benchmark Suite のデータを利用しているが，分子のサイズによってオーバヘッドに差が出ている。TRPCage や FactorIX、JAC などは分子が小さく，GPU での実行時間が短くなり，結果として非常に高い頻度で API 関数の呼び出しが行われるため NVCR による API 関数の転送のオーバヘッドが大きくなっている。水色は DMTCP をベースとして 60 秒間隔でチェックポイントを行い, 出力ファイルを gzip で圧縮するモードであるが, 特に gzip で圧縮する処理に時間がかかっている。極端にストレージが遅い環境を除いては無圧縮でファイルを保存する方が早いと考えられる。いずれにしても 60 秒に 1 回という極端なチェックポイント間隔でもこの程度のオーバヘッドで抑えられており，十分に現実的であると言える。

[9] 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高速数値解法の構築（多田野）

本研究では，鞍点型と呼ばれる連立一次方程式：

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

を高速に解く数値解法の構築を目的とする。ここで，A は正則な n 次非対称行列，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。鞍点型連立一次方程式は，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おいて現れ，その求解時間は多くの割合を占める。したがって，同方程式の高速数値解法は必要不可欠である。連立一次方程式に対する有力な数値解法として，ガウスの消去法や LU 分解法などの直接法が知られているが，問題サイズが大規模な場合は多くの計算量とメモリ量が必要となるため, 適用は難しい。そのため，大規模問題に対してはクリロフ部分空間反復法などの反復法による求解をせざるを得ない。

鞍点型連立一次方程式に対する反復法による単純な求解アプローチとして，クリロフ部分空間反復法をそのまま適用することが考えられる。しかしながら，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係数行列 $\hat{A}$ は悪条件になることがあり，クリロフ部分空間反復法の残差の収束性が悪化する。GMRES 法のような頑健な方法では求解は比較的良い収束性を示すものの，同法は長い漸化式を用いる解法であるため多くの計算量とメモリ量を必要とする。BiCGSTAB 法のような短い漸化式に基づく反復法では，残差の収束性が悪く求解は困難である。そのため，短い漸化式に基づく解法でも求解が可能となるアプローチが必要である。

我々は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用いた求解手法を構築した。鞍点型連立一次方程式を変形すると，解ベクトル $\boldsymbol{x}$, $\boldsymbol{y}$は以下のように与えられる。

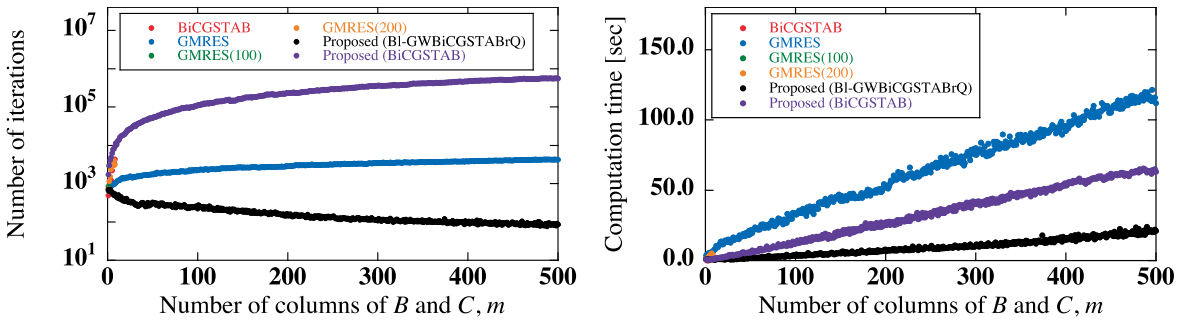
$$\begin{cases} \boldsymbol{x} = A^{-1}\boldsymbol{f} - A^{-1}B\boldsymbol{y}, \\ \boldsymbol{y} = \left(C^{\mathrm{T}}A^{-1}B\right)^{-1}\left(C^{\mathrm{T}}A^{-1}\boldsymbol{f} - \boldsymbol{g}\right). \end{cases}$$

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 $\boldsymbol{v}$ は複数右辺ベクトルをもつ連立一次方程式 $A\tilde{X} = \tilde{B}$ を解くことによって得られる。これを解いて得られた行列 U, ベクトル $\boldsymbol{v}$ を用いることで鞍点型連立一次方程式の解が求められる。提案アプローチにおける計算の主要部は，複数右辺連立一次方程式 $A\tilde{X} = \tilde{B}$ の求解である。同方程式の係数行列は A であるため，行列 A が良条件である場合はその性質を活かすことができ，短い漸化式に基づく反復法でも求解できる可能性がある。また，複数右辺連立一次方程式に対する反復法であるブロッククリロフ部分空間反復法は複数本の方程式を同時に解くことができ，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ことがあ

る。よって本研究では，提案アプローチ内で現れる複数右辺連立一次方程式の求解法としてブロッククリロフ部分空間反復法を適用し，その効果を検証する。

数値実験によって提案アプローチの有効性を検証する。テスト問題として，行列 A には SuiteSparse Matrix Collection で公開されている行列 epb2（行列サイズ：25,228，非零要素数：175,027），行列 B，C は乱数で与え，B と C の列数 m を変化させて実験を行った。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」の 1 ノード（CPU: Intel Xeon Gold 6126 2.6GHz (12 cores)×2，コンパイラ：Intel Fortran ver. 19.0.5，コンパイルオプション：-qopenmp -axCORE-AVX512）を用いた。また，計算は OpenMP を用いて 24 スレッド並列で行った。

従来アプローチでは鞍点型連立一次方程式に対して，クリロフ部分空間反復法の一つである BiCGSTAB 法，GMRES 法，及びリスタート付き GMRES 法（GMRES(100), GMRES(200)）を適用した。提案アプローチでは，内部で現れる複数右辺連立一次方程式 $A\tilde{X} = \tilde{B}$ に対して，ブロッククリロフ部分空間反復法の一つである Block GWBiCGSTABrQ 法を適用した場合と，BiCGSTAB 法を各右辺ベクトルに対して逐次的に適用した場合を比較した。図 16 に行列 B, C の列数 m を変化させたときの反復回数と計算時間の変化を示す。なお，各反復法において残差が収束条件を満たさなかったケースについては，グラフから除外した。図 16 (a)に示すように，従来アプローチにおいては GMRES 法以外の反復法は大半の m において残差が収束条件を満たさなかった。GMRES 法では全ての m に対して残差が収束条件を満たしたものの，m が増加するにしたがって反復回数が増加する傾向がみられた。一方，提案アプローチにおいては，m が大きくなっても残差が収束条件を満たしていることがわかる。特に，複数右辺連立一次方程式の解法に Block GWBiCGSTABrQ 法を用いた場合は，m の増加に伴って反復回数が減少していることがわかる。これはブロッククリロフ部分空間反復法の特長の一つであり，この性質が提案アプローチに対してうまく合致していることを示している。また，図 16 (b)に示すように，提案アプローチの計算時間は従来アプローチで GMRES 法を用いた場合よりも高速であった。特に，内部問題の複数右辺連立一次方程式にブロッククリロフ部分空間反復法を適用することで，より高速に求解できることを確認した。この研究成果は，国際会議 CEFC 2020 にて発表した。



(a) 反復回数の変化　　(b) 計算時間の変化

図 16 行列$\boldsymbol{B}$, $\boldsymbol{C}$ の列数 $\boldsymbol{m}$ の変化に対する反復回数と計算時間の変化

[10] 3 次元モデル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）

複数枚の断層画像から 3 次元オブジェクトを再構成する問題を，3 次元モデル画像再構成問題と呼ぶ。同問題では，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 次元ボリューメトリックデータを作成する。2 次元補間関数の作成に拡張 CSRBF 法と呼ばれる手法を用いると，その結合係数を決定するために以下の連立一次方程式を解く必要がある。

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

ここで，A は実対称疎行列，C はn行 3 列行列，X, F はn行K列行列，Y は 3 行K列行列，O_3, 及び $O_{3,K}$ はそれぞれ 3 次零行列，3 行K列零行列である。K は断層画像の枚数である。行列 F の要素は断層画像の画素値によって決定される。この連立一次方程式は係数行列 $\hat{A}$ が実対称行列の鞍点型連立一次方程式であり，この求解に多くの時間を要するため，高速求解手法が必要である。

本研究では，この鞍点型連立一次方程式の求解に，前述の提案アプローチを用いて高速化を図った。数値実験によりその性能を評価する。数値実験には，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用いた。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。数値実験は，CPU: Intel Xeon E5-2620 v3 2.4GHz (6 cores) × 2, コンパイラ : Intel Fortran ver. 19.0.0 を搭載した計算機上で行い, OpenMP で 12 スレッド並列で計算した。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，及び不完全コレスキー分解前処理付き Block CGrQ 法（Block ICCGrQ 法）を用いた。従来のアプローチでは鞍点型連立一次方程式にそのまま両手法を適用し，提案アプローチでは内部問題として現れる複数右辺連立一次方程式に対して両手法を適用した。図 17 に CSRBF 法のサポート半径 R を変化させたときの従来アプローチと提案アプローチの計算時間変化を示す。提案アプローチを用いることにより，Block CGrQ 法，Block ICCGrQ 法の両方において計算時間が減少した。前処理と提案アプローチの両方を用いることで，従来アプローチよりも鞍点型連立一次方程式を高速に求解できることを確認した。特に $R = 0.02$ の場合は，提案アプローチで Block ICCGrQ 法を用いたときのの計算時間は，従来アプローチで Block CGrQ 法を用いた場合の約 1/4 であった。本研究の研究成果は，日本応用数理学会をはじめとした学会，研究会において発表済みである。

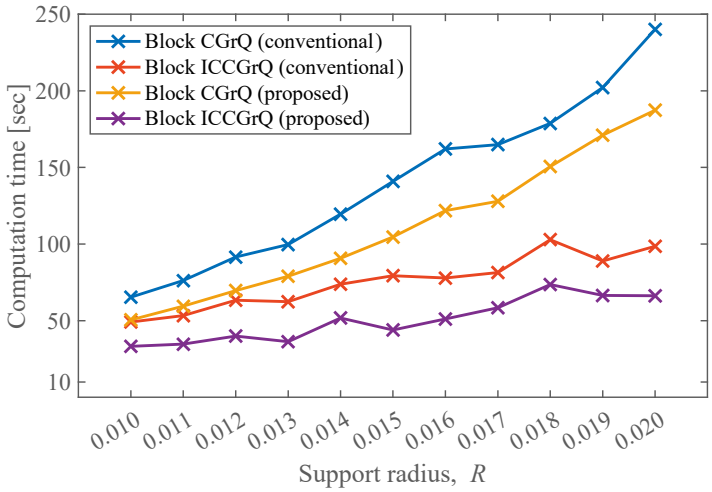


図 17 サポート半径 ***R*** を変化させたときの 3 次元モデル画像再構成問題で現れる連立一次方程式の求解時間変化

[11] 宇宙輻射輸送コードの GPU・FPGA 複合演算加速（朴，小林，藤田）

GPU・FPGA 複合演算加速が必要とされる理由は，複数の物理モデルや複数の同時発生する物理現象を含むシミュレーションであるマルチフィジックスアプリケーションに対して有効と睨んでいるためである．マルチフィジックスでは，シミュレーション内に様々な特性の演算が出現するので，GPU だけでは演算加速が困難な場合がある．そのため，GPU だけでは対応しきれない特性の演算の加速に FPGA を利用することで，アプリケーション全体の性能向上を狙う．

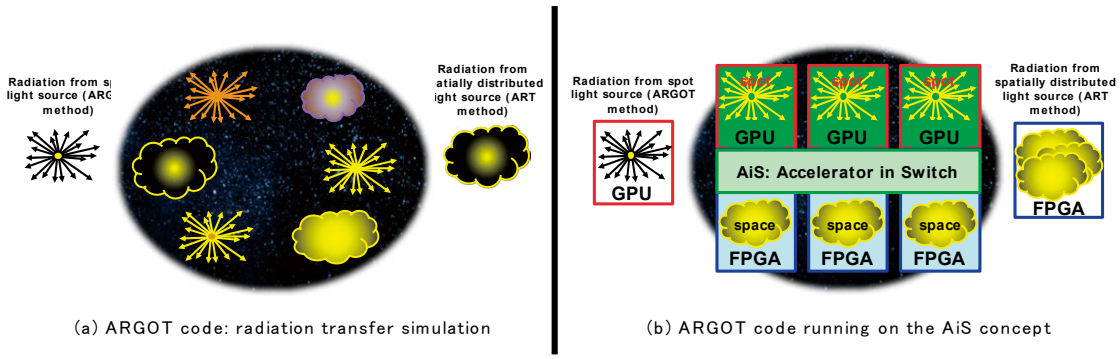


図 18 (a) 宇宙輻射輸送コード ARGOT の概観．(b) GPU・FPGA 連携による ARGOT コードの演算加速

本研究では，マルチフィジックスの例である，宇宙輻射輸送シミュレーションコード Accelerated Radiative transfer on Grids using Oct-Tree (ARGOT) を対象にする．ARGOT は計算科学研究センターで開発されている宇宙輻射輸送を解くプログラムであり，図 18 (a) に示すように，点光源と空間に分散した光源の 2 種類の輻射輸送問題を含む．なお，点光源からの輻射輸送の計算を ARGOT コードにおける ARGOT 法，空間に広がる光源からの輻射輸送の計算を ARGOT コードにおける ART 法と呼ぶ．ここで重要なのは，ART 法は ARGOT プログラムの中で 90%以上の計算時間を占めるアルゴリズムであり，本研究では ART 法の演算をこれまでに開発してきた FPGA カーネルを用いて加速させる．また，ARGOT 法の演算には既に ARGOT プログラムに実装されている GPU カーネルを用いることで，図 18 (b) に示すように主要演算部分を GPU と FPGA に適材適所的に機能分散して ARGOT コードを最適化する．

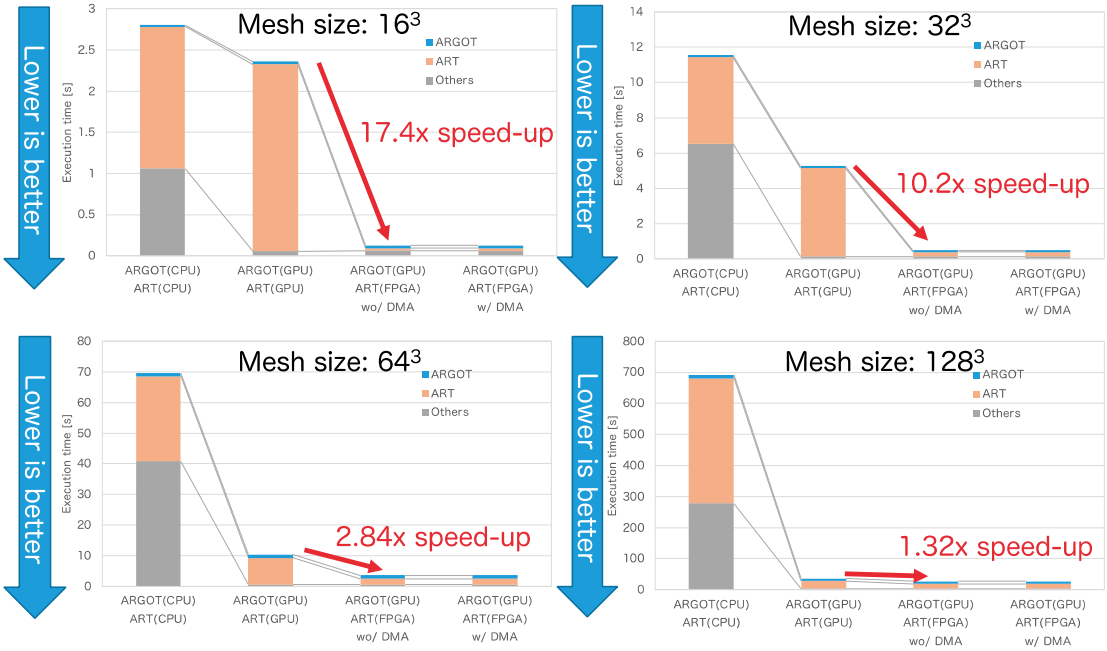


図 19 メッシュサイズに応じた ARGOT コードの性能比較

図 19 に CPU 実装，GPU 実装，FPGA 実装の性能比較を示す．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．また，w/ DMA，wo/ DMA はデバイス間 DMA を用いているか，用いていないか，をそれぞれ表す．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．本稿では，単一の Xeon CPU (14 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.

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．特に，メッシュサイズ 16^3 と 32^3 の場合は，P100 GPU にとって問題サイズが小さすぎることから，3,584 CUDA Core に対して十分な演算の並列度が得られず，その傾向が顕著である．

これに対して ARGOT(GPU) / ART(FPGA) は，デバイス間 DMA の有無によらず，どの問題サイズであっても常に GPU 実装より優れていることが分かる．DMA の有無によって性能差がほとんど生じないのは，ARGOT コード中の GPU--FPGA 間通信が全体の処理の 1%程度であるためである．先行研究で報告されている通り，FPGA 版 ART は空間並列だけでなくパイプラインによって時間方向にも並列計算を行う．そのため，問題サイズが小さい場合であっても性能を発揮でき，その場合，GPU を大幅に凌駕する性能を持つことを示している．

本研究の成果は国際会議 ASAP2020[12]及情報処理学会論文誌[11]に査読付論文として発表された。

[12] 宇宙幅射輸送コード ARGOT の OpenACC による GPU 実装（小林，藤田，朴）

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 に転送させた．

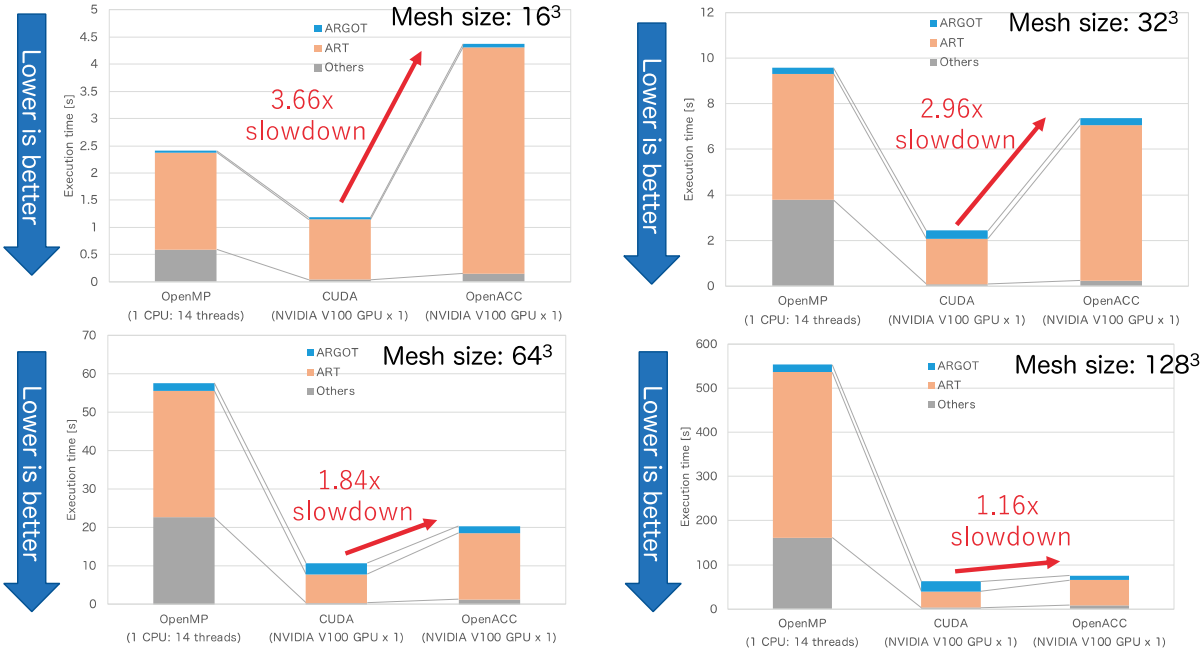


図 20 OpenMP, CUDA, OpenACC 実装による ARGOT コードの性能比較

図 20 に OpenMP, CUDA, OpenACC 実装による ARGOT コードの性能比較を示す．Others は各メッシュの化学反応と輻射熱・輻射冷却を計算するルーチンであり，各メッシュにおけるこれらの計算は独立であるため，CUDA および OpenACC 実装に非常に良く適合している．ARGOT 法については，メッシュサイズが大きくなるほど，CUDA 版の性能劣化が目立った．これは，CUDA 実装では，メモリサイズの小さい過去の GPU デバイスを使用していた影響から cudahostalloc での pinned メモリの確保・解放をイテレーション毎に実行していたため，今後は初期化時に必要なデータを一度で確保するように実装を変更する必要があることが分かった．そして，ART 法では，小さいメッシュサイズでの OpenACC 版の性能は CUDA 版の 25 %であり，大きいメッシュサイズの場合は CUDA 版の 86 %の性能を示した．現在，この原因を NVIDIA のプロファイリングツールを用いて調査中である．

[13] 宇宙輻射輸送コードの並列 FPGA 化（藤田，小林，朴）

我々はこれまでの研究で，早期宇宙の問題を輻射輸送計算で解決する Authentic Radiative Transfer (ART) 法を Intel Arria 10 向けに最適化を行っていた．本論文では，最新の Intel FPGA である Intel Stratix 10 向けに ART 法の最適化を行い，GPU 実装と複数ノードを用いて実行して性能比較を行う．我々は，OpenCL 上から FPGA が持つ複数の光リンクを用いて並列計算を行うためのフレームワークとして，Communication Integrated Reconfigurable CompUting System

(CIRCUS)というシステムを研究開発しており，本研究ではこれを ART 法に適用して並列計算を行うものである．なお，本研究で扱うプログラムは，CIRCUS を用いた初めての実アプリケーションを用いた並列計算の実現である．

FPGA 実装は，GPU 実装と比べて，1 ノード時 x4.54, 2 ノード時 x8.41, 4 ノード時 x10.64 の高速化を達成した．FPGA 間通信には前述した CIRUCS を用い，GPU 間通信には InfiniBand HDR100 ネットワークを用いて性能評価を行ったものである. FPGA 実装は 1 ノード時と 4 ノード時を比べて 94.2%の並列化効率を達成した．この効率は，CIRCUS が提供する低レイテンシで高スループットなパイプライン通信によって達成できたと考えている．

本研究の成果は国際ワークショップ H2RC2020(SC2020 内)[15]及び情報処理学会 ACS 論文誌[16]において査読付論文として発表された．

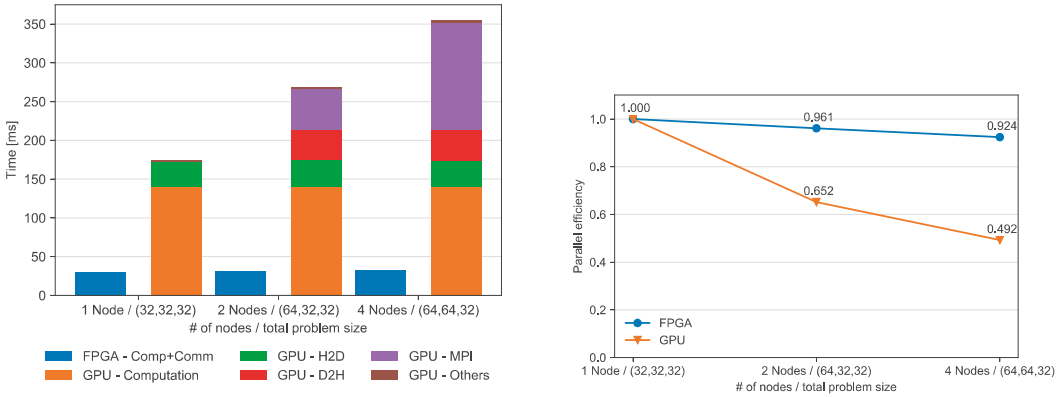


図 21 GPU と FPGA の性能比較．左は計算時間の比較，
右は 1 ノード実行を基準とした並列化効率を表す．

4. 教育

学生の指導状況

1. 佐藤駿一，修士（工学），GPU における SELL 形式疎行列ベクトル積の実装と性能評価，筑波大学大学院システム情報工学研究科修士論文，令和 3 年 3 月（指導：高橋大介）
2. 原山赳幸，修士（工学），オーバー・アンダーフローを抑えた高精度かつ高速な 2 ノルム計算手法，筑波大学大学院システム情報工学研究科修士論文，令和 3 年 3 月（指導：高橋大介）
3. 杉原航平，修士（工学），ノードローカルバーストバッファのための並列 I/O の研究，筑波大学大学院システム情報工学研究科修士論文，令和 3 年 3 月（指導教員：建部修見）

4. 高橋宗史，修士（工学），大規模データ処理のための分散オブジェクトストレージコネクタに関する研究，筑波大学大学院システム情報工学研究科修士論文，令和 3 年 3 月（指導教員：建部修見）
5. 畑中智之，修士（工学），HPC におけるコンテナ利用に関する研究，筑波大学大学院システム情報工学研究科修士論文，令和 3 年 3 月（指導教員：建部修見）
6. 菊池航平，学士（工学），地域気象シミュレーション City-LES の詳細モデル GPU 化に関する研究，筑波大学情報学群情報科学類卒業論文，令和 3 年 3 月（指導：朴泰祐）
7. 巨畠和樹，学士（工学），DAOS ファイルシステムの性能評価，筑波大学情報学群情報科学類卒業論文，令和 3 年 3 月（指導教員：建部修見）
8. 笠井大暉，学士（工学），分散キャッシュファイルシステムの設計，筑波大学情報学群情報科学類卒業論文，令和 3 年 3 月（指導教員：建部修見）
9. 河西優作，学士（工学），Persistent Memory を活用した Deep Learning フレームワークの研究，筑波大学情報学群情報科学類卒業論文，令和 3 年 3 月（指導教員：建部修見）
10. 菅沼夏樹，学士（工学），並列計算環境における複数右辺連立一次方程式の反復解法の高速化に関する研究，筑波大学情報学群情報科学類卒業研究論文，令和 3 年 3 月（指導：多田野寛人）

5. 受賞、外部資金、知的財産権等

受賞

1. 朴泰祐，情報処理学会 CS 領域功績賞, 2021 年 3 月 15 日

外部資金

1. 文部科学省高性能汎用計算機高度利用事業費補助金　朴泰祐（代表）, 2017～2021 年度, 全年度直接経費 92,192 千円（2020 年度直接経費 23,994 千円），「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 (B)（一般），朴泰祐（代表），2018～2020 年度，全年度直接経費 13,300 千円（2020 年度直接経費 4,700 千円），「再構成可能システムと GPU による複合型高性能計算プラットフォーム」
3. 理化学研究所受託研究，朴泰祐（代表）, 2015～2020 年度, 全年度直接経費 38,000 千円（2020 年度直接経費 5,000 千円），「ポスト京の並列プログラミング環境およびネットワークに関する研究」

4. 科学研究費補助金基盤研究 (C)，高橋大介（代表），H31～R3 年度，1,300 千円（R2 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」
5. 文部科学省委託研究，建部修見（分担），R2 年度，34,100 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）
6. NEDO，建部修見（分担），H30～R3 年度，4,394 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
7. 共同研究（富士通研究所），建部修見（代表），R2 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
8. 科学研究費助成事業（学術研究助成基金助成金）（若手研究），額田彰（代表），R2~R3 年度，1,700 千円（R2 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」
9. 科学研究費補助金基盤研究 (C)，多田野寛人（代表），R2～R4 年度，2,080 千円（R2 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」
10. 科研費 若手研究，小林諒平（代表），FPGA を用いた超高速ハードウェアソーティングアルゴリズムの開発，総計 3,380 千円 (期間: 2019 ~ 2020)，年度毎の配分金額 (2019: 2,470 千円，2020: 910 千円)

知的財産権

該当なし

6. 研究業績

(1) 研究論文

A) 査読付き論文

1. Daisuke Tsuji, Taisuke Boku, Ryosaku Ikeda, Takuto Sato, Hiroto Tadano, Hiroyuki Kusaka, "Parallelized GPU Code of City-Level Large Eddy Simulation", Proc. of ISPDC2020, Warsaw (virtual), 8 pages, Jul. 7th, 2020.
2. Ryuta Tsunashima, Ryohei Kobayashi, Norihisa Fujita, Taisuke Boku, Seyong Lee, Jeffrey S. Vetter, Hitoshi Murai, Masahiro Nakao, Mitsuhisa Sato, "OpenACC unified programming environment for GPU and FPGA multi-hybrid acceleration", Proc. of HLPP2020, Porto (virtual), 20 pages, Jul. 8th, 2020.
3. Ryuta Kashino, Ryohei Kobayashi, Norihisa Fujita, Taisuke Boku, “Performance Evaluation of OpenCL-Enabled Inter-FPGA Optical Link Communication Framework CIRCUS and SMI”, Proc. Of HPC Asia 2021, Jeju Island (virtual), 10 pages, Jan. 20th, 2021.

4. Kazuhiko Komatsu, Ayumu Gomi, Ryusuke Egawa, Daisuke Takahashi, Reiji Suda, and Hiroyuki Takizawa, "Xevolver: A code transformation framework for separation of system-awareness from application codes", Concurrency and Computation: Practice and Experience, Vol. 32, No. 7, e5577, 2020.
5. Daisuke Takahashi, "Fast Multiple Montgomery Multiplications Using Intel AVX-512IFMA Instructions", Proc. 20th International Conference on Computational Science and Its Applications (ICCSA 2020), Part V, Lecture Notes in Computer Science, Vol. 12253, pp. 655-663, Springer, 2020. (short paper)
6. Yukimasa Sugizaki and Daisuke Takahashi, "Fast Computation of the Exact Number of Magic Series with an Improved Montgomery Multiplication Algorithm", Proc. 20th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2020), Part II, Lecture Notes in Computer Science, Vol. 12453, pp. 365-382, Springer, 2020.
7. Kohei Sugihara, Osamu Tatebe, "Design of Locality-aware MPI-IO for Scalable Shared File Write Performance", Proceedings of the IEEE International Workshop on High-Performance Storage, pp. 1080-1089, doi: 10.1109/IPDPSW50202.2020.00179, 2020
8. 町田健太, 建部修見, 大規模メタゲノムデータに対する分散並列相同性検索システム GHOSTZ PW/GF の提案, 情報処理学会論文誌コンピューティングシステム(ACS), Vol. 13, No. 2, pp. 13-27, 2020 年 9 月
9. Takayuki Tanabe, Takashi Hoshino, Hideyuki Kawashima, Osamu Tatebe, "An Analysis of Consistency Control Protocols for In-Memory Databases with CCBench", Proceedings of the VLDB Endowment, Vol. 13, No. 13, pp. 3531-3544, doi: 10.14778/3424573.3424575, 2020
10. Kohei Sugihara, Osamu Tatebe, "Design of Direct Read from Sparse Segments in MPI-IO", Proceedings of the 6th IEEE International Conference on Data Science and Systems (DSS), pp. 1308-1315, doi: 10.1109/HPCC-SmartCity-DSS50907.2020.00168, 2020
11. 倉本亮世，多田野寛人，複数右辺ベクトルを持つ線型方程式に対するブロック積型反復解法の近似解高精度化，日本応用数理学会論文誌，Vol. 30, No. 4, pp. 290−319, 2020.
12. Ryohei Kobayashi, Norihisa Fujita, Yoshiki Yamaguchi, Taisuke Boku, Kohji Yoshikawa, Makito Abe, Masayuki Umemura, "Multi-Hybrid Accelerated Simulation by GPU and FPGA on Radiative Transfer Simulation in Astrophysics", IPSJ Journal of Information Processing/28/pp.1073-1089, 2020-12
13. Ryohei Kobayashi, Norihisa Fujita, Yoshiki Yamaguchi, Taisuke Boku, Kohji Yoshikawa, Makito Abe, Masayuki Umemura, "Accelerating Radiative Transfer Simulation with GPU-

FPGA Cooperative Computation", 2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)/pp.9-16, 2020-07

14. Norihisa Fujita, Ryohei Kobayashi, Yoshiki Yamaguchi, Tomohiro Ueno, Kentaro Sano, Taisuke Boku, “Performance Evaluation of Pipelined Communication Combined with Computation in OpenCL Programming on FPGA”, Proc. of AsHES2020 (Int. Workshop on Accelerators and Hybrid Exascale Systems) in IPDPS 2020, New Orleans (virtual), 10 pages, May 18th, 2020.
15. Norihisa Fujita, Ryohei Kobayashi, Yoshiki Yamaguchi, Taisuke Boku, Kohji Yoshikawa, Makito Abe, Masayuki Umemura, "OpenCL-enabled Parallel Raytracing for Astrophysical Application on Multiple FPGAs with Optical Links", 2020 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC), 2020, pp. 48-55, doi: 10.1109/H2RC51942.2020.00011.
16. 藤田典久, 小林諒平, 山口佳樹, 上野知洋, 佐野健太郎, 朴泰祐, OpenCL プログラミングを用いた並列 FPGA 処理システムの性能評価, 情報処理学会論文誌コンピューティングシステム（ACS）, 13 巻 3 号, pp. 13-28, 2020.

B） 査読無し論文

1. Naruya Kitai, Daisuke Takahasi, Franz Franchetti, Takahiro Katagiri, Satoshi Ohshima, and Toru Nagai, "Adaptation of A64 Scalable Vector Extension for Spiral", 情報処理学会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-178(9), 6 pages, 2021 年 3 月
2. 原山赳幸，工藤周平，椋木大地，今村俊幸，高橋大介，オーバー・アンダーフローを抑えた高精度かつ高速な 2 ノルム計算手法, 情報処理学会研究報告ハイパフォーマンスコンピューティング（HPC），2020-HPC-177(8)，9 pages, 2020 年 12 月
3. 杉崎行優，高橋大介，NVIDIA Volta GPU における浮動小数点演算を用いた剰余乗算の高速化，日本応用数理学会 2020 年度年会講演予稿集, 2 pages, 2020 年 9 月
4. 高橋大介，Intel AVX-512IFMA 命令を用いた複数の Montgomery 乗算の高速化，日本応用数理学会 2020 年度年会講演予稿集, 2 pages, 2020 年 9 月
5. 畑中智之, 建部修見, Cygnus 上での Docker Rootless Mode の利用の検討, 研究報告ハイパフォーマンスコンピューティング（HPC）, Vol. 2020-HPC-175, No. 24, pp. 1-6, 2020 年 7 月
6. 高橋宗史, 建部修見, ストレージコネクタ spark-ceph-connector の書き込み性能の改善, 研究報告ハイパフォーマンスコンピューティング（HPC）, Vol. 2020-HPC-176, No. 8, pp. 1-12, 2020 年 9 月

7. 巨畠和樹，建部修見, DAOS ファイル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）, Vol. 2021-HPC-178, No. 23, pp. 1-9, 2021 年 3 月
8. 建部修見, CHFS: コンシステントハ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. 2021-HPC-178, No. 22, pp. 1-8, 2021 年 3 月
9. 小林諒平, 藤田典久, 朴泰祐, OpenACC と OpenCL の混合記述による GPU-FPGA デバイス間連携, 研究報告ハイパフォーマンスコンピューティング(HPC) 2020-HPC-177(12) 1 - 7 2020 年 12 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, 宇宙幅射輸送コード ARGOT の OpenACC による GPU 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) 2020-HPC-175(7) 1 – 7, 2020 年 7 月
11. 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, 再構成可能システムと GPU による多重複合型演算加速, 計算工学講演会論文集 Proceedings of the Conference on Computational Engineering and Science / 日本計算工学会 編 25, 2020 年 6 月
12. 柏野隆太, 小林諒平, 藤田典久, 朴泰祐, "OpenCL 対応 FPGA 間光リンク接続フレームワーク CIRCUS と SMI の性能評価", 情報処理学会研究報告ハイパフォーマンスコンピューティング(HPC), 2020-HPC-175(16), pp.1-8, 2020 年 7 月 31 日.
13. Fan Ruochong, Ao Yun, Yamaguchi Yoshiki, Boku Taisuke, "A study of an FPGA-based cluster computing with high-speed serial links", 電子情報通信学会信学技報, CPSY2020-11, DC2020-11, pp.1-7, 2020 年 7 月 31 日.

(2) 国際会議発表

A) 招待講演

1. Taisuke Boku, "Multi-Hetero Accelerated Supercomputing ~System, Programming and Applications~", Keynote at AsHES2020 (in IPDPS2020), New Orleans (virtual), May 18th, 2020.
2. Taisuke Boku, "University of Tsukuba's Accelerated Computing with GPU+FPGA", Int. Workshop ADAC9 (virtual), Sep. 17th, 2020.

B）一般講演

1. Daisuke Tsuji, Taisuke Boku, Ryosaku Ikeda, Takuto Sato, Hiroto Tadano, Hiroyuki Kusaka, "Parallelized GPU Code of City-Level Large Eddy Simulation", Proc. of ISPDC2020, Warsaw (virtual), Jul. 7th, 2020.

2. Ryuta Tsunashima, Ryohei Kobayashi, Norihisa Fujita, Taisuke Boku, Seyong Lee, Jeffrey S. Vetter, Hitoshi Murai, Masahiro Nakao, Mitsuhisa Sato, "OpenACC unified programming environment for GPU and FPGA multi-hybrid acceleration", Proc. of HLPP2020, Porto (virtual), Jul. 8th, 2020.
3. Ryuta Kashino, Ryohei Kobayashi, Norihisa Fujita, Taisuke Boku, “Performance Evaluation of OpenCL-Enabled Inter-FPGA Optical Link Communication Framework CIRCUS and SMI”, Proc. Of HPC Asia 2021, Jeju Island (virtual), Jan. 20th, 2021.
4. Daisuke Takahashi, "Automatic Tuning of Computation-Communication Overlap for Parallel 3-D FFT with 2-D Decomposition", SIAM Conference on Computational Science and Engineering (CSE21), Fort Worth, TX, USA (Virtual Conference), March 4, 2021.
5. Daisuke Takahashi, "Fast Multiple Montgomery Multiplications Using Intel AVX-512IFMA Instructions", 20th International Conference on Computational Science and Its Applications (ICCSA 2020), Cagliari, Italy (Virtual Conference), July 2, 2020.
6. Yukimasa Sugizaki and Daisuke Takahashi, "Fast Computation of the Exact Number of Magic Series with an Improved Montgomery Multiplication Algorithm", Proc. 20th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2020), New York, NY, USA (Virtual Conference), October 4, 2020.
7. Kohei Sugihara, Osamu Tatebe, "Design of Locality-aware MPI-IO for Scalable Shared File Write Performance", IEEE International Workshop on High-Performance Storage, New Orleans, LA, USA (Virtual conference), May 2020
8. Kohei Sugihara, Osamu Tatebe, "Design of Direct Read from Sparse Segments in MPI-IO", 6th IEEE International Conference on Data Science and Systems (DSS), Yanuca Island, Cuvu, Fiji (Virtual conference), 2020
9. Hiroto Tadano, Shota Ishikawa, "A numerical study on the acceleration of solution of saddle point problems by using Block Krylov subspace methods", 19th Biennial IEEE Conference on Electromagnetic Field Computation (CEFC 2020), Virtual Conference, Nov. 2020.
10. Ryohei Kobayashi, Norihisa Fujita, Yoshiki Yamaguchi, Taisuke Boku, Kohji Yoshikawa, Makito Abe, Masayuki Umemura, "Accelerating Radiative Transfer Simulation with GPU-FPGA Cooperative Computation", 2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP), 2020 年 7 月 7 日
11. Norihisa Fujita, Ryohei Kobayashi, Yoshiki Yamaguchi, Tomohiro Ueno, Kentaro Sano, Taisuke Boku, “Performance Evaluation of Pipelined Communication Combined with Computation in OpenCL Programming on FPGA”, Proc. of AsHES2020 (Int. Workshop on

Accelerators and Hybrid Exascale Systems) in IPDPS 2020, New Orleans (virtual), 10 pages, May 18th, 2020.

(3) 国内学会・研究会発表

A) 招待講演

1. Taisuke Boku, "HPC for AI & AI for HPC ~time to meet together~", Keynote at HPC-AI Advisory Council Japan, Jan. 26th, 2021.

B) その他の発表

1. Naruya Kitai, Daisuke Takahasi, Franz Franchetti, Takahiro Katagiri, Satoshi Ohshima, and Toru Nagai, "Adaptation of A64 Scalable Vector Extension for Spiral", 情報処理学会第 178 回ハイパフォーマンスコンピューティング研究発表会，オンライン，2021 年 3 月 15 日.
2. 原山赳幸，工藤周平，椋木大地，今村俊幸，高橋大介，“オーバー・アンダーフローを抑えた高精度かつ高速な 2 ノルム計算手法”，情報処理学会第 177 回ハイパフォーマンスコンピューティング研究発表会，オンライン，2020 年 12 月 21 日.
3. 杉崎行優，高橋大介，“NVIDIA Volta GPU における浮動小数点演算を用いた剰余乗算の高速化”，日本応用数理学会 2020 年度年会，オンライン，2020 年 9 月 10 日.
4. 高橋大介，“Intel AVX-512IFMA 命令を用いた複数の Montgomery 乗算の高速化”，日本応用数理学会 2020 年度年会，オンライン開催，2020 年 9 月 8 日.
5. 畑中智之, 建部修見, Cygnus 上での Docker Rootless Mode の利用の検討, 情報処理学会 HPC 研究会, オンライン, 2020 年 7 月
6. 高橋宗史, 建部修見, ストレージコネクタ spark-ceph-connector の書き込み性能の改善, 情報処理学会 HPC 研究会, オンライン, 2020 年 9 月
7. 巨畠和樹, 建部修見, DAOS ファイルシステムの性能評価, 情報処理学会 HPC 研究会, オンライン, 2021 年 3 月
8. 建部修見, CHFS: コンシステントハ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, 2021 年 3 月
9. 石川翔大，多田野寛人，齋藤歩，3 次元モデル再構成問題に現れる鞍点型連立一次方程式に対する前処理付き反復法の開発，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　第 30 回研究会，オンライン，2020 年 12 月.
10. 石川翔大，多田野寛人，齋藤歩，3 次元モデル再構成問題に現れる鞍点型連立一次方程式に対する前処理付き反復法の性能評価，日本応用数理学会　第 17 回研究部会連合発表会，オンライン，2021 年 3 月.

11. 石川翔大，多田野寛人，齋藤歩，3 次元モデル再構成問題に現れる鞍点型連立一次方程式の求解高速化，【非線形問題の解法と可視化に関する研究会】2020 年度第 1 回研究会，オンライン，2021 年 3 月.
12. 小林諒平, 藤田典久, 朴泰祐, OpenACC と OpenCL の混合記述による GPU-FPGA デバイス間連携, 第 177 回ハイパフォーマンスコンピューティング研究発表会, 2020 年 12 月 21 日
13. 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, 宇宙幅射輸送コード ARGOT の OpenACC による GPU 実装, 第 175 回ハイパフォーマンスコンピューティング研究発表会 (SWoPP2020), 2020 年 7 月 30 日
14. 小林諒平, 藤田典久, 中道安祐未, 山口佳樹, 朴泰祐, 吉川耕司, 安部牧人, 梅村雅之, GPU･FPGA 複合演算加速による宇宙輻射輸送コード ARGOT の性能評価, 第 173/174 回ハイパフォーマンスコンピューティング研究発表会, 2020 年 5 月 12 日
15. 柏野隆太, 小林諒平, 藤田典久, 朴泰祐, "OpenCL 対応 FPGA 間光リンク接続フレームワーク CIRCUS と SMI の性能評価", 情報処理学会研究報告ハイパフォーマンスコンピューティング(HPC), 2020-HPC-175(16), pp.1-8, 2020 年 7 月 31 日.
16. Fan Ruochong, Ao Yun, Yamaguchi Yoshiki, Boku Taisuke, "A study of an FPGA-based cluster computing with high-speed serial links", 電子情報通信学会信学技報, CPSY2020-11, DC2020-11, pp.1-7, 2020 年 7 月 31 日.
17. 藤田典久, 小林諒平, 山口佳樹, 朴泰祐, HBM2 メモリを持つ FPGA ボードの性能評価, 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-178(6), pp.1-8, 2021 年 3 月
18. 藤田典久, 小林諒平, 山口佳樹, 上野知洋, 佐野健太郎, 朴泰祐, 高位合成を用いた FPGA 間通信機構の設計と性能評価, 計算工学講演会論文集, 2020 年 6 月
19. 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, 研究報告ハイパフォーマンスコンピューティング（HPC）, 2020-HPC-175(8), pp.1-8, 2020 年 7 月

(4) 著書、解説記事等

該当なし

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）

- 「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

- 「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
- 素粒子物理研究部門と Japan Lattice Data Grid（JLDG）の構築、運用に関する連携
- 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

該当なし

国際連携・国際活動

- "Research on Multi-Hybrid Accelerated Supercomputing with GPU and FPGA cooperation", International Collaboration with ORNL (PI: Jeff Vetter) under DOE-MEXT Exascale Computing Collaboration（朴）
- アルゴンヌ国立研究所とストレージシステムについて共同研究を行っている。
- オークリッジ国立研究所とバーストバッファシステムについて共同研究を行っている。

8. シンポジウム、研究会、スクール等の開催実績

- 朴泰祐, 実行委員長, FPGA for HPC Workshop（科学研究費基盤研究(B)「再構成可能システムとGPUによる複合型高性能計算プラットフォーム」終了ワークショップ）, virtual, 2021 年 2 月 26 日.
- Gfarm シンポジウム 2020，東京，2020 年 10 月 9 日
- Gfarm ワークショップ 2021，オンライン，2021 年 3 月 5 日
- GPU オンラインキャンプ，オンライン開催，2020 年 9 月 9 日～11 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. 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
7. 高橋大介：理化学研究所客員主管研究員
8. 高橋大介：HPCI 利用研究課題審査委員会レビューアー
9. 高橋大介：HPCI 連携サービス運営・作業部会委員

10. 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
11. 建部修見：HPCI 共用ストレージ運用部会副部会長
12. 建部修見：HPCI 連携サービス運営・作業部会委員
13. 建部修見：理化学研究所客員主管研究員
14. 建部修見：情報通信研究機構協力研究員
15. 建部修見：HPCI 利用研究課題審査委員会レビューアー
16. 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
17. 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
18. 建部修見：情報処理学会論文誌コンピューティングシステム編集委員長
19. 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: General Co-Chair, IEEE International Conference on Cluster Computing 2020 (Cluster2020)
3. Taisuke Boku: Steering Committee Member, IEEE Cluster
4. Taisuke Boku: Steering Committee Member, International Conference on Parallel Processing
5. Taisuke Boku: Steering Committee Member, Intel eXtreme Performance Users Group (IXPUG)
6. Taisuke Boku: Scientific Committee Member, Supercomputing Frontier Europe 2021
7. Taisuke Boku: Organizing Chair, IXPUG Workshop HPCAsia 2021
8. Daisuke Takahashi: The 20th International Conference on Computational Science and Its Applications (ICCSA 2020) Publicity Committee Member
9. Daisuke Takahashi: 2020 IEEE International Conference on Cluster Computing (Cluster 2020) Program Committee Member
10. Daisuke Takahashi: The 15th International Workshop on Automatic Performance Tuning (iWAPT 2020) Program Committee Member
11. Daisuke Takahashi: The International Conference on Computational Science (ICCS 2020) Program Committee Member
12. Daisuke Takahashi: The 23rd IEEE International Conference on Computational Science and Engineering (CSE 2020) Program Committee Member
13. Daisuke Takahashi: The 19th International Symposium on Parallel and Distributed Computing (ISPDC 2020) Program Committee Member

14. Daisuke Takahashi: The 18th IEEE International Symposium on Parallel and Distributed Processing with Applications (IEEE ISPA 2020) Program Committee Member
15. 高橋大介:電子情報通信学会／情報処理学会 第 19 回情報科学技術フォーラム（FIT2020）担当委員
16. 高橋大介：情報処理学会 2020 年度論文賞選定ワーキンググループ（ジャーナル）委員
17. 高橋大介：情報処理学会論文誌ジャーナル／JIP 編集委員会委員
18. 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
19. Osamu Tatebe: Program Committee, IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SC20)
20. Osamu Tatebe: Program Committee, ACM International Symposium on High Performance Distributed Computing (HPDC 2020)
21. Osamu Tatebe: Program Committee, IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2020)
22. Osamu Tatebe: Program Area Chair, IEEE International Conference on Cluster Computing (Cluster 2020)
23. Akira Nukada: Program Committee Member, 49th International Conference on Parallel Processing (ICPP 2020)
24. Hiroto Tadano: Publication Co-Chair, The 39th JSST Annual International Conference on Simulation Technology (JSST 2020)
25. 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
26. 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
27. 多田野寛人：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
28. 多田野寛人：情報処理学会論文誌コンピューティングシステム　編集委員
29. Ryohei Kobayashi: Program Committee, SC20
30. Ryohei Kobayashi: Program Committee, CANDAR'20
31. Ryohei Kobayashi: Program Committee, 8th International Workshop on Computer Systems and Architectures (CSA'20) held in conjunction with CANDAR'20
32. Ryohei Kobayashi: Registration Chair, IEEE Cluster 2020
33. Ryohei Kobayashi: Program Committee, IEEE Symposium on Low-Power and High-Speed Chips and Systems (COOL Chips 23)
34. 小林諒平：xSIG 2020 プログラム委員
35. 小林諒平：SWoPP2020 組織委員

36. 小林諒平：SWoPP2020 実行委員 (コンピュータシステム研究会担当幹事)
37. 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
38. 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

11. その他

海外長期滞在、フィールドワークなど

該当なし