Vol.2021-HPC-180 No.25
2021/7/21

遠隔手続き呼出しを用いたタスク並列モデルによるクラスタ連携に関する研究

辻 美和子[1,a)]　宮島 敬明[1,2]　佐野 健太郎[1]　佐藤 三久[1]

概要：複数のアーキテクチャが並び立つポストムーア時代に向けて，それらの性質を理解し，アプリケーションに適したハードウエアを利用することがより重要になると考えられる．単一のアプリケーションであっても，ワークフローとタスクプログラミングを用いてカーネルごとにハードウエアを使い分ける協調計算により，アプリケーション全体の実行を加速が期待される．本研究では，CPU+GPU などの異なるアーキテクチャからなる単一のヘテロジニアスクラスタや性質の異なる複数のクラスタの連携を実現するために，遠隔手続き呼出し (Remote Procedure Call, RPC) を用いたタスク並列モデルの使用を検討する．このために，マルチコア CPU からなる汎用並列システムである「富岳」から，富岳と InifiniBand で接続された FPGA クラスタ「ESSPER」へ一部の処理をタスクとしてオフロードする検証を行った．実験により，富岳上のクライアントプログラムから，FPGA クラスタ上のワーカープログラムに，FPGA を用いた計算の依頼が技術的に可能であることが示された．

1. はじめに

半導体の集積率は 18 か月で 2 倍になるとしたムーアの法則だが，2000 年代前半からすでに性能カーブはこの曲線上にはなく，近年では 2 倍の性能を得るために数年を要するようになっている [2]．半導体性能の伸びに限界が見られるなかで，従来の計算機とはまったく異なる量子力学的な現象を利用して問題を解く量子コンピュータや，汎用マイクロプロセッサにはない回路を自由に構築し問題に対して最適化できる FPGA (Field Programming Gate Array) への期待が高まっている．一方で，すでに十分に普及しており豊富なエコシステムを持つ CPU や GPU の果たす役割もまだまだ無視できない．

複数のアーキテクチャが並び立つポストムーア時代に向けて，それらの性質を理解し，アプリケーションに適したハードウエアを利用することがより重要になると考えられる．さらに，単一のアプリケーション内でカーネルによって異なるハードウエアを使い分ける協調計算は，アプリケーション全体の実行を加速し効率化するための重要な手段となり得る．たとえば，小林らの放射伝達シミュレーションにおいては，ARGOT アルゴリズムを GPU で実行し，ART アルゴリズムを FPGA で実行する協調計算を行い，高速化を実現している [4]．

本研究では，CPU+GPU などの異なるアーキテクチャを持つ単一のヘテロジニアスクラスタや性質の異なる複数のクラスタの連携を実現するために，遠隔手続き呼出し (Remote Procedure Call, RPC) を用いたタスク並列モデルの使用を検討する．RPC を用いたタスク並列は，マスターワーカー型の並列プログラミングモデルであり，マスタープログラムが遠隔のノードに起動するリモートワーカープログラムに通信を介してタスクの実行を依頼する．本研究では，グリッド計算機環境に向けた RPC ライブラリ OmniRPC [5] を用い，マスタープログラムとして HPC アプリケーション開発者に馴染みの深い OpenMP-task プログラミングを用いるタスク並列モデルを検討した．

提案する RPC を用いたタスク並列モデルを実証する環境として，スーパーコンピュータ「富岳」および FPGA クラスタ「Elastic and Scalable System for High-Performance Computing, ESSPER」を用いた．富岳は Arm ベースのプロセッサである A64FX からなるホモジニアスなスーパーコンピュータであり，ESPPER は X86 プロセッサと 2 つの FPGA を持つノードを 8 ノード連結したヘテロジニアスクラスタである．富岳と ESSPER の間は Inifiniband ネットワークで接続されている．

実験により，富岳上のマスタープログラムから，RPC により ESSPER の X86 部のリモートプログラムに計算の一

[1] 理化学研究所計算科学研究センター
RIKEN Center for Computational Science
[2] 明治大学理工学部情報科学科
School of Science and Technology, Meiji University
[a)] miwako.tsuji@riken.jp

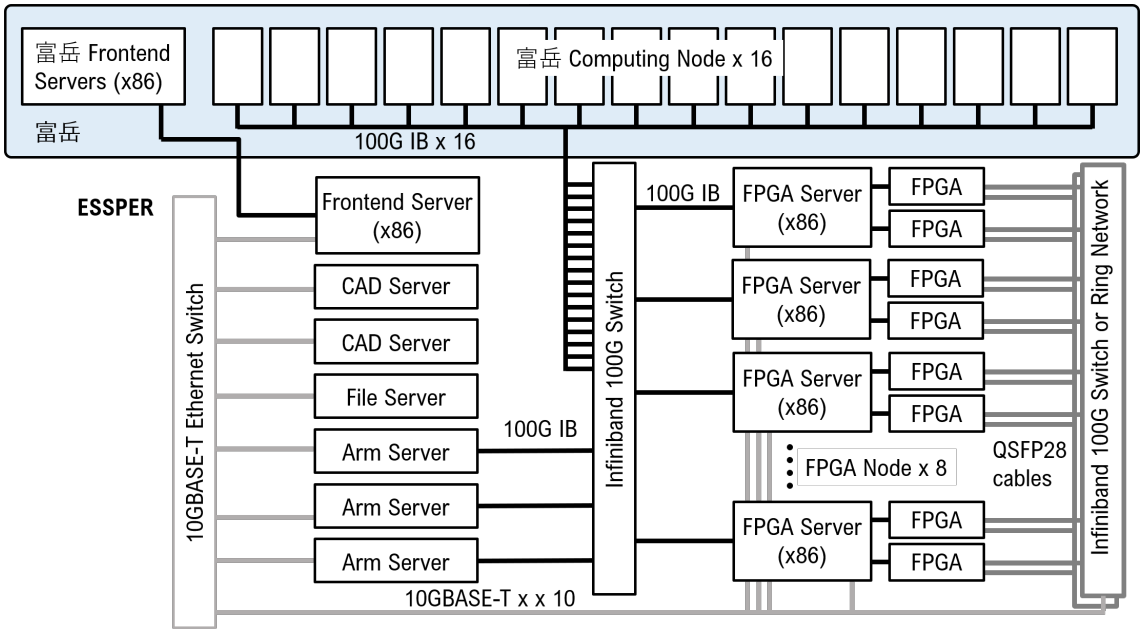

図 1　FPGA クラスタ ESSPER とスーパーコンピュータ富岳と接続の概要

部を依頼し，リモートプログラムから FPGA にカーネルをオフロード可能であることを確認した．これにより，遠隔手続き呼出しを用いたタスク並列モデルが CPU からなるスーパーコンピュータと CPU が苦手な計算を担う FPGA クラスタを連携させることが可能であることが示された．

2. FPGA クラスタ試作システム ESSPER

スーパーコンピュータ「富岳」の開発をになったフラグシップ 2020 プロジェクトでは，富岳の付加価値として「PCI を使った将来拡張の可能性」について調査検討をおこなってきた．この目的は，富岳に対して PCI による外部デバイスの拡張可能性および外部デバイス拡張のための技術的課題とその解決法を明らかにすることであった．拡張部として問題毎に専用回路を再構成可能なデバイスである FPGA クラスタが採用され，富岳との接続方式を含めた技術的検討のための FPGA クラスタ試作システム ESSPER (Elastic and Scalable System for High-Performance Reconfigurable Computing) が開発された [7], [8].

ESSPER および ESSPER と富岳との接続の概要を図 1 に示す．ESSPER は 8 つの計算ノードとフロントエンドノードからなる．ESSPER のフロントエンドは，富岳のフロントエンドを介してログイン可能であり，ESSPER のフロントエンドを介して 8 つの FPGA ノードにアクセスすることができる．ESSPER の計算ノードは，FPGA Server と呼ばれる CPU (Intel(R) Xeon(R) Gold 5122 CPU @ 3.60GHz) 1 つと，Stratix 10 SX を搭載した 2 つの D5005 FPGA カードからなる．FPGA ノードの間は，100G Infiniband (IB) で接続されており，同じスイッチに富岳計算ノードも 100G IB でつながっている．現在のところ，FPGA クラスタと直接通信できる富岳の計算ノードは特定のラックに配置された 16 ノードのみである．これらの 16 ノードは，Tofu とは別に ib インターフェースと IP アドレスを持つ．ESSPER 側のノードの ib インターフェースにも同様に IP アドレスが割り当てられており，これらを用いれば相互に通信が可能である．

3. クラスタ連携のための遠隔呼び出しを用いタスク並列モデル

ポストムーア時代に向けて複数の異なるアーキテクチャを効率的に利用するために，それらのアーキテクチャの性質を理解し，アプリケーションに適したアーキテクチャを選択することが重要である．さらに，単一のアプリケーション内でカーネルによって異なるハードウエアを使い分ける協調計算は，アプリケーション全体の実行を加速し効率化するための重要な手段となり得る．たとえば，小林らの放射伝達シミュレーションにおいては，ARGOT アルゴリズムを GPU で実行し，ART アルゴリズムを FPGA で実行する協調計算を行い，高速化を実現している [4].

本研究では，CPU+GPU などからなるヘテロジニアスクラスタや，富岳 ⇔ESSPER などの性質の異なる複数のクラスタの連携を実現するために，遠隔手続き呼出し (Remote Procedure Call, RPC) を用いたタスク並列モデルの使用を検討する．RPC を用いたタスク並列は，マスターワーカー型の並列プログラミングモデルであり，マスタープログラムが遠隔のノードに起動するリモートワーカープログラムに通信を介してタスクの実行を依頼する．本研究では，グリッド計算機環境に向けた RPC ライブラリ OmniRPC [5] を用い，マスタープログラムとして HPC アプリケーショ

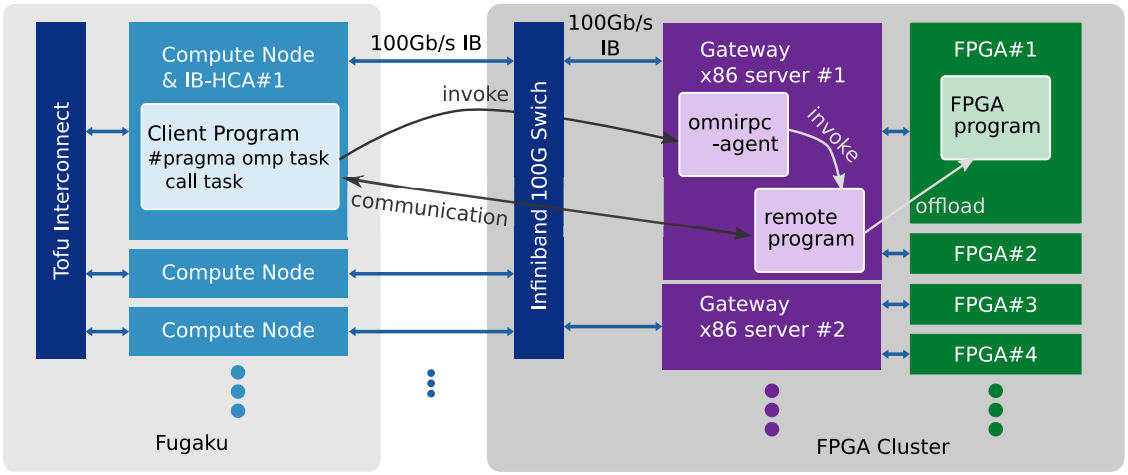

図 2 富岳から RPC により FPGA クラスタでプログラムを実行する例

ン開発者に馴染みの深い OpenMP-task プログラミングを用いるタスク並列モデルを検討した．OmniRPC は，クラスタや広域ネットワークで構成されたグリッドなど，さまざまな並列計算機環境においてマスターワーカー型の並列プログラムを実行するための開発実行環境およびライブラリである．クラスタ内はもちろん，エージェントと呼ばれる仲介プログラムを使用することで，ネットワークをまたいだ遠隔手続き呼び出しも可能である．

図 2 に，富岳と FPGA クラスタを用いた PRC プログラムの実行の概要を示す．クライアントプログラムは，IB で FPGA クラスタと接続された富岳の計算ノードで実行され，OpenMP4.0 で定義されたタスク指示文を用いて，RPC を行う．OpenMP によるスレッド並列を用いることで複数の独立の RPC を同時に行うことも可能である．クライアントプログラムの task 指示文によりフォークした OpenMP スレッドは，OmniRPC ライブラリを介して遠隔手続き呼び出しを実行し，FPGA クラスタの CPU 上の仲介プログラムを介して，同じく FPGA クラスタの CPU 上にリモートプログラムを起動する．リモートプログラムは，クライアントプログラムからの指示に応じて，FPGA へのタスクのオフロードを実行し，結果を返す．

具体的なクライアントプログラムの例を**図 3** に示す．この例では，Cholesky 分解の各カーネルを RPC で実行している．OpenMP の `#pragma omp task` 指示文によって指定されたコードブロックが，`OmniRpcCallAsync` を介して，RPC ライブラリを非同期で呼出している．`OmniRpcCallAsync` の第一引数はタスク名を，続く引数はタスクへの入出力を示す．クライアントプログラムの OpenMP の実行時ライブラリは，タスク指示文と付随する `depend` 節で指定されるデータ依存関係に従って，OpenMP タスクの実行順序を決定し，処理する．**図 4** にワーカープログラムの例を示す．

タスクの記述には Interface Definition Language (IDL) が用いられる．IDL の処理部は C プログラムと同様である．タスクへの入出力については `IN/OUT/INOUT` を明確に記述する必要がある．これらの引数は前述のクライアントプログラムの `OmniRpcCallAsync` の第二引数以降と対応する．IDL により記述されたタスクは，OmniRPC 開発実行環境が提供するコンパイラにより処理される．このコンパイラは，IDL の処理部に，クライアントプログラムとの通信やデータ受信送信にかかわる部分を自動的に追加して，C プログラムを生成するソーストゥソースのコンパイラである．生成されたソースコードは gcc などのネイティブコンパイラで実行可能なバイナリへとコンパイルされる．リモートプログラムから FPGA へのカーネルオフロードを行う場合は，処理部にいわゆるホストプログラムそのものか，ホストプログラムへの呼出しを記述する．

図 5 に富岳から FPGA クラスタ上に起動されるリモートプログラムのコンパイルフローを示す．図では簡単のために OpenCL の例を用いているが，他の FPGA プログラム手法も同様である．クライアントプログラムは C および OpenMP などにより，OmniRPC の API を用いたリモートプログラムへのタスクの依頼と，タスク同士の依存関係が記述される（図左）．富岳の計算ノードは，Armv8.2-A 命令セットアーキテクチャーに基づく A64FX であるので，クライアントプログラムは対応するコンパイラでコンパイルされる．本実験では富士通製コンパイラが用いられた．FPGA クラスタの CPU 部分にクライアントプログラムによって起動されるリモートプログラムは，IDL によって記述される（図中央）．OmniRPC のコンパイラは，IDL の処理部に RPC のインターフェースを追加し，ソースコード（図 `*.rex.c`）を生成する．処理部からは，FPGA のいわゆるホストプログラムを呼び出す．これらは，ネイティブコンパイラによってコンパイルされ，ライブラリと共にリンクされる．FPGA のカーネルは，OpenCL や HLS などの FPGA プログラム言語で記述され，対応するコンパイ

```
void cholesky(const int ts, const int nt, double* A[nt][nt])
{
    OmniRpcRequest rq1, rq2, rq3, rq4;

#pragma omp parallel private(rq1, rq2, rq3, rq4)
#pragma omp single
    for (int k = 0; k < nt; k++) {
#pragma omp task depend(out:A[k][k])
{
        rq2 = OmniRpcCallAsync("rpc_potrf", ts, ts, A[k][k]);
        OmniRpcWait(rq2);
}
        for (int i = k + 1; i < nt; i++) {
#pragma omp task depend(in:A[k][k]) depend(out:A[k][i])
{
            rq3 = OmniRpcCallAsync("rpc_trsm", ts, ts, A[k][k], A[k][i]);
            OmniRpcWait(rq3);
}
        }
        for (int i = k + 1; i < nt; i++) {
            for (int j = k + 1; j < i; j++) {
#pragma omp task depend(in:A[k][i], A[k][j]) depend(out:A[j][i])
{
                rq1 = OmniRpcCallAsync("rpc_dgemm", ts, ts, A[k][i], A[k][j], A[j][i]);
                OmniRpcWait(rq1);
}
            }
#pragma omp task depend(in:A[k][i]) depend(out:A[i][i])
{
            rq4 = OmniRpcCallAsync("rpc_syrk", ts, ts, A[k][i], A[i][i]);
            OmniRpcWait(rq4);
}
        }
    }
#pragma omp taskwait
}
```

図 3　クライアントプログラムの例 (Cholesky 分解)

```
Define rpc_potrf(IN int ts, IN int ld, INOUT double A[ts][ld])
{
  fpga_host_main(ts, ld, A);
}
```

図 4　ワーカープログラムの例 (Cholesky 分解)

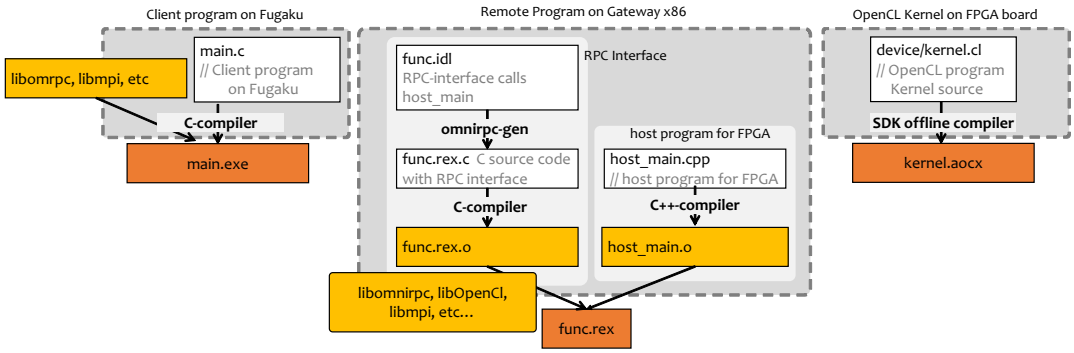

図 5 富岳から FPGA クラスタを用いる RPC プログラムのプログラミングフロー (OpenCL)

ラでコンパイルされる（図右）．現段階の実装では，FPGA プログラムのコンパイルおよび，リモートプログラムのリンクなどの一部の処理は手動で実行する必要がある．将来的には，OmniRPC コンパイラを拡張し，すべてのコンパイルは自動で行われる予定である．

4. 実験

提案する遠隔呼び出しを用いたタスク並列プログラミングモデルによる CPU クラスタ（富岳）と FPGA クラスタ（ESSPER）の挙動を確認するために，実機にてテストを行った．以下に述べるテスト手順のうちの多くが自動化される予定であるが，現段階ではいくつかの手順はマニュアル操作により行った．

4.1 OpenCL によるベクトル和

まず，OpenCL により記述された FPGA プログラムをオフロードする実験を行った．クライアントプログラムで適当なベクトル A および B を生成し，RPC により FPGA を用いて $C := A + B$ を実行する．FPGA 上のベクトル和のプログラムおよび FPGA ホストプログラムは，OpenCL のサンプルを用いた [3].

準備として，ESSPER と IB で接続された富岳の計算ノードに interactive job と呼ばれる計算ノードに直接ログインしてリアルタイムでこれを利用するジョブを投入する．富岳の計算ノード上で `ifconfig` などのコマンドを用いて，IB インターフェイスにふられた IP アドレスを確認する．富岳のクライアントプログラムから見える適当な場所に `hosts.xml` を作成し，以下のように記述する：

```
<OmniRpcConfig>
<Host name="fpgaserv8">
<Agent invoker="ssh" port="40001"/>
</Host>
<Host name="fpgaserv1">
 ...
</OmniRpcConfig>
```

`hosts.xml` はクライアントプログラムに引数として渡され，エージェントプログラムが起動されるノードのリストと起動方法，エージェントプログラムとの通信に使用される任意のポート番号が記述される．ただし，本実験の段階ではエージェントプログラムは手動で起動したため，これらのうちで利用されるのはポート番号だけである．

また，ESSPER の計算ノードから見える適当なディレクトリ（通常は `$HOME` の `.omrpc_registry` 以下）に，`stubs` ファイルを作成し，タスクを含むモジュール名，タスク名，リモートプログラムのフルパスを記述する：

```
# <モジュール名> <タスク名> <フルパス>
func rpc_add_vec /home/.../func.rex
```

富岳の計算ノードから以下のコマンドでクライアントプログラムを起動する．本実験ではクライアントプログラムは `main` とした：

```
$ ./main --hostfile ~/.omrpc_registry/hosts.xml\\
    --debug
```

別のターミナルを起動し，ESSPER の計算ノードにログインして CPU 上でエージェントプログラムを立ち上げる．引数として，クライアントプログラムが起動している富岳の計算ノードの IP アドレスが与えられる：

```
$ ./omrpc-agent -debug -workpath /tmp/       \\
    -host <IP Adress of Fugaku compute node>\\
    -port 40001
```

以上の操作により，

富岳計算ノードのクライアントプログラム `main` から，ESSPER 計算ノードの `omnirpc-agent` に IB を通じてタスクの実行が依頼され，

- `omnirpc-agent` は，`stubs` に記載された情報に従って，必要なリモートプログラム `func.rex` を起動し，
- `func.rex` の FPGA ホストプログラム部は `main` よりタスクの依頼と引数ベクトルを受け取り，
- `func.rex` の FPGA ホストプログラム部は，FPGA にベクトル和計算をオフロードし
- `func.rex` の FPGA ホストプログラム部は，FPGA カーネルから結果を受け取って，これを `main` に送信する

ことが確認できた．

図 6 に ESSPER 側の実行時ログを示す．`fpgaserv8:...func.rex:25778` で始まるログは ESSPER 計算ノードの CPU 上のリモートプログラムからデバッグ情報として出力されたものである．`=== start ===` から `=== end ===`の間のログは FPGA ホストプログラムによる出力である．

4.2 HLS による 1DFFT

続いて，FPGA 向けにインテル HLS (High Level Synthesis) コンパイラを，ホスト部に c++(GCC) 8.3.1 コンパイラを用いて，1 次元 FFT を FPGA にオフロードして計算する実験を行った．本実験では ESSPER のチュートリアル用に作成された 1 次元 FFT サンプルを用いた．コードはチュートリアル用であるために，宮島ら [6] により検討された FPGA 向け 1 次元 FFT のうちメモリの読み書きのみを再現しており，FFT 計算は行っていない．

クライアントプログラムは 2 つの float 型の 1 次元配列を乱数を用いて，リモートプログラムにそれらを送信し，FFT の計算を依頼する．FPGA プログラムのホスト部となるリモートプログラムは，うけとった配列を持ちかえたうえで，FPGA のメモリに書き込み，FPGA に FFT 計算の実行を依頼する．この配列の持ちかえは，メモリチャネルを効率的に利用するために行われる．FPGA のメモリから計算結果を読み出し，もとの形式に持ちかえたうえで，クライアントプログラムに送信する．

図 7 に HLS コンパイラを用いて FPGA プログラムを作成する場合の RPC プログラムのプログラミングフローを示す．クライアントプログラムは図 5 の OpenCL の場合と同様なので省略した．基本的なプログラミングフローは OpenCL と同様であるが，コンパイルした AFU (Accelerator Functional Unit) イメージである，`*.gbs` ファイルを事前に `fpgaconf` コマンドで FPGA ボードにロードしておく必要がある．

その後，4.1 章と同様の手順により，富岳上のクライアントプログラムから 1 次元配列を ESSPER 上のリモートプログラムに転送し，FFT 計算部分を FPGA にオフロードして実行ができることを確認した．

図 8 にリモートプログラムの実行時間を図 9 にクライアントプログラムの実行時間を示す．

前述のように本実験にはチュートリアル用の FPGA の演算部が省略されたダミープログラムを用いているため，計測の主な対象は富岳⇔ ESSPER 間，ESSPER の CPU ⇔ FPGA 間のデータ転送および ESSPER の CPU 上で実行されるデータ転送のためのデータの持ち替え部分である．FPGA へのデータ転送のためのデータの持ち替えは，不連続なメモリ書き込みを含み，コストが比較的大きい．また，CPU から FPGA へのデータ転送部は計測に FPGA ボードの設定部分を含むが，後者が固定かつ大半を占めていると考えられる．以上の結果から，演算量の増加がデータサイズ N の増加に対して O(N) よりも急激なもので，CPU よりも FPGA に適した性質のカーネルについて，RPC を用いたタスクプログラミングモデルによる効果が得られると考えられる．D5005 ボードに向けて最適化された FFT は現在実装中であり，演算部も含めた計測は今後の課題としたい．

5. 関連研究

ヘテロジニアスな計算環境を効率的に用いるためのライブラリや，ヘテロジニアスなアーキテクチャ上で効率的に実行可能なアプリケーションの開発は広く行われている．

前述のように，小林らの放射伝達シミュレーションにおいては，GPU および CPU のノードからなるシステムにおいて，ARGOT アルゴリズムを GPU で実行し，ART アルゴリズムを FPGA で実行する協調計算を行い，高速化を実現している [4]．また，朴ら [9] は，汎用の計算並列 CP-PACS と重力計算専用システム GRAPE-6 を用いて宇宙輻射流体計算を行っている．StarPU[1] は，複数の種類のプロセッサ (PU) に対応するコードを生成し，実行するランタイムライブラリである．

本研究は，

- CPU + GPU などのヘテロジニアスなノードから構成される 1 つのクラスタではなく，2 つの性質の異なるクラスタを利用する枠組みを提案した，
- 富岳のクライアントプログラムから FPGA クラスタである ESSPER を用いてタスクを実行し，同時に CPU に適した計算についてはスーパーコンピュータの豊富な計算資源を用いることができる枠組みを示した

ことに特徴がある．

ESSPER を富岳から利用する別の方法として，FPGA カードのドライバである OPAE (Open Programmable Accelerator Engine) をソフトウエアブリッジ化した RemoteOPAE により gbs の書き込みなども含めた低レベルな操作を富岳側から行う研究も進められている [8]．本研究で用いた RPC を用いたタスクプログラミングでは，クライアントプログラム側はリモートプログラム側のアーキテクチャによらず一意な API でプログラミングすることが可能である．

6. おわりに

本研究では，異なるアーキテクチャを持つ単一のヘテロジニアスクラスタや性質の異なる複数のクラスタの連携を実現するために，遠隔手続き呼出し (Remote Procedure Call, RPC) を用いたタスク並列モデルの使用を検討した．このために，マルチコア CPU からなる汎用並列システムである「富岳」から，富岳と InifiniBand で接続された

```
[???:**func.rex:25778] arg[6]=-port
[???:**func.rex:25778] stub arg port=36389,52773 host=192.168.10.21
[fpgaserv8:**func.rex:25778] connect host=192.168.10.21:36389
[fpgaserv8:**func.rex:25778] check byte order ...
[fpgaserv8:**func.rex:25778] check byte order  done = 0...
[fpgaserv8:**func.rex:25778] stub init End ...
[fpgaserv8:**func.rex:25778] stub req start ...
[fpgaserv8:**func.rex:25778] reading request ...
[fpgaserv8:**func.rex:25778] request = 2
[fpgaserv8:**func.rex:25778] requested stub: name = rpc_vec_add
[fpgaserv8:**func.rex:25778] omrpc_send_stub_info ...
[fpgaserv8:**func.rex:25778] reading request ...
[fpgaserv8:**func.rex:25778] request = 4
[fpgaserv8:**func.rex:25778] requested call index=0
[fpgaserv8:**func.rex:25778] alloc work size=0x3d0900 addr=0x7f33becee010
[fpgaserv8:**func.rex:25778] stub: recv_array(1)
[fpgaserv8:**func.rex:25778] alloc work size=0x3d0900 addr=0x7f33be91d010
[fpgaserv8:**func.rex:25778] stub: recv_array(2)
[fpgaserv8:**func.rex:25778] alloc work size=0x3d0900 addr=0x7f33be54c010
[fpgaserv8:**func.rex:25778] stub: recv_array(3)
[fpgaserv8:**func.rex:25778] REQ_CALL end
[fpgaserv8:**func.rex:25778] Ninf_stub_SET_ARG(0)=0xf4240, 0xf4240
[fpgaserv8:**func.rex:25778] Ninf_stub_SET_ARG(1)=0xbecee010, 0x7f33becee010
[fpgaserv8:**func.rex:25778] Ninf_stub_SET_ARG(2)=0xbe91d010, 0x7f33be91d010
[fpgaserv8:**func.rex:25778] Ninf_stub_SET_ARG(3)=0xbe54c010, 0x7f33be54c010
[fpgaserv8:**func.rex:25778] Ninf_stub_BEGIN
===  start ===
1.000000 1.000000 1000000
Initializing OpenCL
Platform: Intel(R) FPGA SDK for OpenCL(TM)
Using 1 device(s)
sched_setaffinity: Invalid argument
sched_setaffinity: Invalid argument
  pac_s10_dc : Intel PAC Platform (pac_f000000)
sched_setaffinity: Invalid argument
Using AOCX: vector_add.aocx
sched_setaffinity: Invalid argument
sched_setaffinity: Invalid argument
Launching for device 0 (1000000 elements)
sched_setaffinity: Invalid argument

Time: 11.356 ms
Kernel time (device 0): 3.581 ms

Verification: PASS
===  end   ===
[fpgaserv8:**func.rex:25778] Ninf_stub_END begin
[fpgaserv8:**func.rex:25778] Ninf_stub_END send out args
[fpgaserv8:**func.rex:25778] Ninf_stub_END send array(3)
```

図 6　ESSPER 側の実行ログ

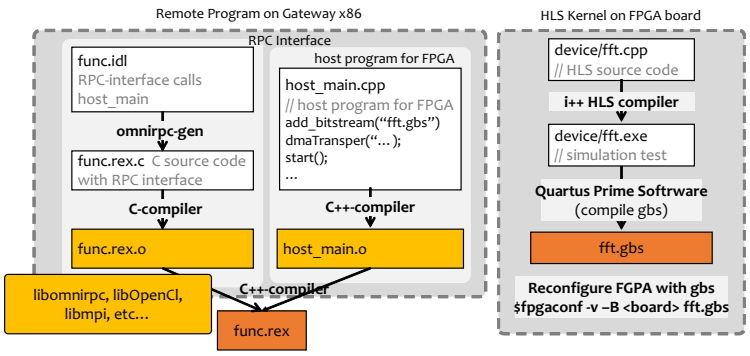

図 7　富岳から FPGA クラスタを用いる RPC プログラムのプログラミングフロー (HLS)

FPGA クラスタ「ESSPER」へ一部の処理をタスクとしてオフロードする技術検証を行った．実験により，富岳上のクライアントプログラムから，FPGA クラスタ上のワーカープログラムに，FPGA を用いた計算の依頼が可能であることが示された．

今後の課題としては，さまざまなアプリケーションやベンチマークをによる遠隔呼び出しを用いたタスク並列モデルによる複数クラスタの協調計算の性能評価やオーバーヘッドの計測があげられる．

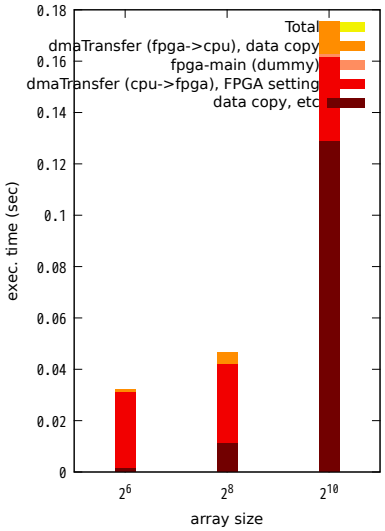

図 8 リモートプログラムの実行時間．積み上げグラフは上から，全体，FPGA からへのデータ転送とデータコピー，カーネル（本研究はダミーで計算はしていない），FPGA へのデータ転送と FPGA 設定，クライアントから送られてきたデータのコピーと FPGA 転送のための持ち替えを示す．

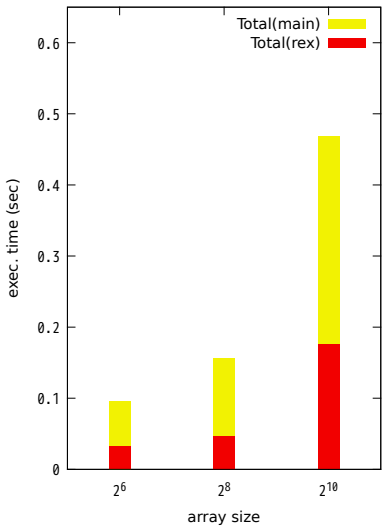

図 9 クライアントプログラムの実行時間．積み上げグラフは上から，API の呼出しから終了までの全体，リモートプログラムの実行時間全体を示す．

参考文献

[1] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier. Starpu: A unified platform for task scheduling on heterogeneous multicore architectures. In *Euro-Par - 15th International Conference on Parallel Processing, volume 5704 of Lecture Notes in Computer Science*, 2009.

[2] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach, Sixth Edition*. 2017.

[3] IIntel Corporation. OpenCL vector addition design example. https://www.intel.co.jp/content/www/jp/ja/programmable/support/support-resources/design-examples/design-software/opencl/vector-addition.html.

[4] R. Kobayashi, N. Fujita, Y. Yamaguchi, T. Boku, K. Yoshikawa, M. Abe, and M. Umemura. Multi-hybrid accelerated simulation by GPU and FPGA on radiative transfer simulation in astrophysics. *Journal of Information Processing*, 28:1073–1089, 2020.

[5] M. Sato, M. Hirano, Y. Tanaka, and S. Sekiguchi. Omnirpc: A grid rpc facility for cluster and global computing in openmp. In *International Workshop on OpenMP Applications and Tools, 2001*, pp. 130–136, 2001.

[6] 宮島, 上野, 佐野. 高性能計算のための高速フーリエ変換の FPGA 実装と評価. 信学技報, vol. 120, no. 339, RECONF2020-61, pp. pp.19–24, 2021.

[7] 佐野. FPGA による高性能計算の課題と将来展望〜FPGA クラスタ試作システム開発で分かったこと〜, 第 179 回ハイパフォーマンスコンピューティング研究発表会 招待講演.

[8] 佐野, 上野, 宮島敬明, J. Huthmann, 小柴. ESSPER: 高性能計算のためのスケーラブルかつ柔軟な FPGA クラスタシステムの開発. 信学技報, vol. 120, no. 339, RECONF2020-61, pp. pp.7–12, 2021.

[9] 朴, 佐藤, 小沼, 牧野, 須佐, 高橋, 梅村. HMCS - G ：グリッド環境における計算宇宙物理のためのハイブリッド計算システム. 情報処理学会論文誌コンピューティングシステム（ACS）, 44(SIG11(ACS3)):pp.1–13, 2003.