Vol.2021-HPC-181 No.7
2021/9/27

WaitIO-Socket:異種システム上の複数 MPI プログラムを結合する通信ライブラリの試作

住元 真司[1]　荒川 隆[4]　坂口 吉生[1]　松葉 浩也[2]
八代 尚[5]　塙 敏博[2]　中島 研吾[2,3]

概要：今年より稼働した Wisteria/BDEC-01 システムでは Arm ベースの大規模シミュレーションノード群と x86_64+GPU を主体とするデータ・学習ノード群を高性能ネットワークで融合し、計算＋データ＋学習を一体化したエクサスケール時代の革新的シミュレーション手法の研究開発が行われる。本論文ではこれら 2 つの異種ノード群上のアプリケーション間を結合する通信ライブラリ WaitIO-Socket の試作・初期評価ならびに WaitIO-Socket 上で開発中の通信基盤 MP の概要・初期評価について述べる。

1. はじめに

計算科学が，理論，実験に続く「第三の科学」と呼ばれるようになって久しく，スーパーコンピューティングは計算科学を支える重要な基盤であったが，近年はデータ科学，機械学習，AI などの新しい分野への応用も盛んになっている。スーパーコンピューティングは，従来の計算科学・計算工学シミュレーションに加えて，データ科学，機械学習等の知見を融合した新しい手法を適用することによって，サイバー空間（仮想空間）とフィジカル空間（現実空間）を高度に融合したシステムを形成し，Society 5.0〔1〕が目指す人間中心の社会の実現に大きく貢献すると期待されている。

東京大学情報基盤センター（以下「東大 ITC」）では，このような状況を想定し，2015 年頃から，「シミュレーション（Simulation）＋データ（Data）＋学習（Learning）（S+D+L）」融合を目指した研究開発に取り組み，一方でその実現のためのスーパーコンピュータシステムとして「『計算・データ・学習』融合スーパーコンピュータシステム」（通称 BDEC（Big Data & Extreme Computing））構築を目指して，様々な研究開発を進めてきた〔2〕。

2021 年 5 月 14 日に運用を開始した「Wisteria/BDEC-01」〔3,4〕は，BDEC システム構想に基づくシステムの第 1 号機であり，シミュレーションノード群（Odyssey，ピーク性能 25.9 PFLOPS，Arm ベース）とデータ・学習ノード群（Aquarius，同 7.2 PFLOPS，x86_64+GPU）の 2 つの計算ノード群を有するヘテロジニアスなシステムである。Odyssey と Aquarius は 2 TB/s の高性能ネットワークで結合され、計算＋データ＋学習を一体化したエクサスケール時代の革新的シミュレーション手法の研究開発を進める予定である。

しかし、現在主流の通信ライブラリである MPI ライブラリは、基本的に単一システム内又は同一 MPI ライブラリ内に閉じた実装になっており、複数システムでの異種 MPI ライブラリ間での通信は想定されていない。複数の異種システム環境上のアプリケーション間で通信を実現する手段が必要である。

この複数システム間でのデータ連携を実現するために h3-Open-SYS/WaitIO〔3〕を開発している。第 1 弾として Oakbridge-CX（OBCX，東大 ITC）〔4〕上で API としては通信インタフェースを採用しながら高性能ストレージを用いてデータ連携を実現させた。今回は、Wisteria/BDEC-01 システムの持つ大容量のネットワーク資源を活用するために WaitIO の機能を通信によって実現することになった。本論文はこの WaitIO 機能を通信に対応させた WaitIO-Socket の試作について述べる。

2. Wisteria/BDEC-01 システム

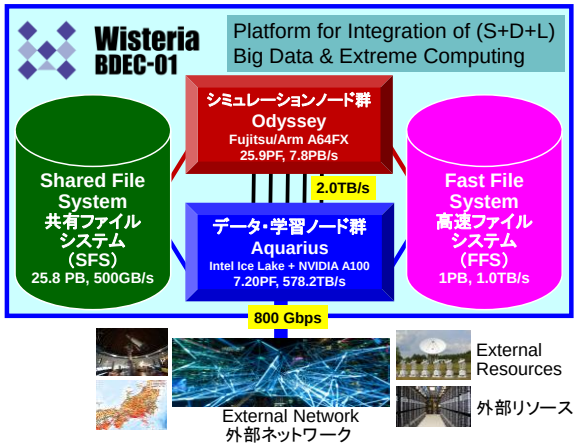

図 1. Wisteria/BDEC-01 システム構成〔3〕

図 1 に Wisteria/BDEC-01 システム(以降、Wisteria)の構成を示す。Wisteria/BDEC-01 は、25.9 PFLOPS の Arm ベースの大規模シミュレーションノード群(Odyssey)と 7.2 PFLOPS の x86_64+GPU を主体とするデータ・学習ノード群(Aquarius)を合計 2 TB/s の InfiniBand EDR/HDR で結合したシステム構成となっている。Odyssey ノードからはラック

1 富士通
2 東京大学 情報基盤センター
3 理化学研究所　計算科学研究センター
4 高度情報科学技術研究機構(RIST)
5 国立環境研究所

ⓒ2021 Information Processing Society of Japan

当たり 8 本ずつの InfiniBand EDR リンク(100 Gbps)を持ち、Aquarius ノード 45 ノード(ノード当たり InfiniBand HDR 200 Gbps x4 本)と、高速ファイルシステムと共有ファイルシステムも含めてフルバイセクションバンド幅の Fat tree で接続されている。2 つのシステムは現在それぞれ独立したジョブ管理の下で運用され、フロントエンドシステムから TCP/IP を用いて各計算ノード群にアクセス可能となっている。

表 1 Wisteria/BDEC-01 各構成システム仕様〔3〕

	Odyssey	Aquarius
総理論演算性能	25.9 PFLOPS	7.2 PFLOPS
総ノード数	7,680	45
インターコネクト	Tofu-D (6 次元メッシュ/トーラス)	InfiniBand HDR (200Gbps) x 4HCA (Full Fat Tree)
プロセッサ/ノード	A64FX(Armv8.1+SVE), 2.2GHz, 1 socket (48 Core+2 or 4 アシスタントコア)	Intel Xeon Platinum 8360Y, 2.4GHz 2 sockets(36+36)
メモリ量/ノード	32 GB	512GiB
メモリバンド幅/ノード	1024GB/s	409.6GB/s
GPU/ノード	-	NVIDIA A100 x8
コンパイラ、MPI	富士通コンパイラ、富士通 MPI	Intel コンパイラ、インテル MPI、(Open MPI)
オペレーティングシステム	Red Hat Enterprise Linux 8	Red Hat Enterprise Linux 8

表 1 に構成ノード群である Odyssey と Aquarius システムの仕様を示す。2 つのシステムを比較するとオペレーティングシステムは同じバージョンであるが、プロセッサ、インターコネクト、コンパイラ、MPI 通信ライブラリが異なる。

各ノード群へのネットワークアクセスについて、Aquarius システムは各計算ノードに InfiniBand HDR が搭載されているが、Odyssey システムの各計算ノードは GIO と呼ばれる Gateway 用の計算ノードを除き Tofu-D インターコネクトのみを搭載している。GIO 以外の計算ノードは GIO を経由した TCP/IP 通信によりアクセス可能である。

3. h3-Open-BDEC

3.1 Wisteria/BDEC-01 による「計算・データ・学習」融合

「計算＋データ＋学習（S+D+L）」融合のためには，Wisteria/BDEC-01 のようなこれまでにない革新的なハードウェアが必要であるが，様々なアプリケーション，ワークロードを Wisteria/BDEC-01 上で開発，実行していくためのソフトウェア群も重要である。

東大 ITC で開発した「ppOpen-HPC（自動チューニング機構を有するアプリケーション開発・実行環境）」〔6〕，「h3-Open-BDEC（「計算＋データ＋学習」融合のための革新的ソフトウェア基盤）」〔7,8〕を利用し，高性能なアプリケーションを容易に開発することが可能である。

3.2 h3-Open-BDEC 概要

東大 ITC では，センター内外の計算科学，計算機科学，数値アルゴリズム，データ科学，機械学習の専門家と協力して，エクサスケール時代のスパコンの能力を最大限活用し，科学的発見を持続的に促進するために，(計算＋データ＋学習)融合による革新的シミュレーション手法を提案し，最小限の計算量・消費電力で融合シミュレーションを実現する研究開発，ソフトウェア基盤実装を実施している〔7,8〕。本研究では，Wistria/BDEC-01 を（計算+データ+学習）融合のためのプラットフォームと位置付け，①変動精度演算・精度保証・自動チューニング（Automatic Tuning，AT）による新計算原理に基づく革新的高性能・高信頼性・省電力数値解法，②機械学習に基づく革新的手法である階層型データ駆動アプローチ（hDDA）の 2 項目を中心に研究開発を実施し，革新的ソフトウェア基盤「h3-Open-BDEC」を開発する（図 2）〔7,8〕。h3-Open-BDEC を Wisteria/BDEC-01 上で様々なアプリケーションに適用，効果を検証し，(計算＋データ＋学習）融合と変動精度演算により，従来手法と同等の正確さを保ちつつ，10 倍以上の飛躍的な計算量・消費電力削減の達成を目指している。h3-Open-BDEC をスーパーコンピュータ「富岳」，次世代 HPCI 計算機資源等へ展開し，(計算＋データ＋学習）融合手法の普及を図るものである。

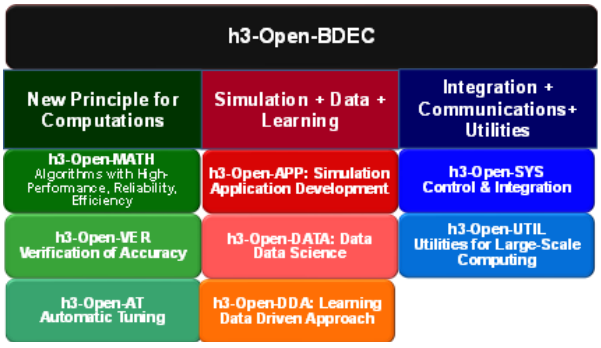

図 2 h3-Open-BDEC の概要〔7,8〕

革新的ソフトウェア基盤「h3-Open-BDEC」（図 2）を構成する h3-Open-MATH（数値アルゴリズム），h3-Open-VER（精度保証），h3-Open-AT（自動チューニング），h3-Open-APP（アプリ開発），h3-Open-DATA（データ科学），h3-Open-DDA（データ駆動アプローチ），h3-Open-UTIL（並列ユーティリティ），h3-Open-SYS（統合・制御）は複数の構成要素

ⓒ2021 Information Processing Society of Japan

を含み，緊密に関連し，以下の 3 層を構成している：

① 「変動精度演算に基づく新計算原理」層（h3-Open-MATH，h3-Open-VER，h3-Open-AT）
② 「(計算＋データ＋学習）融合」層（h3-Open-APP，h3-Open-DATA，h3-Open-DDA）
③ 「統合・通信・ユーティリティ」層（h3-Open-SYS，h3-Open-UTIL）

h3-Open-BDEC はエクサスケール時代のスパコンで（計算＋データ＋学習）融合を実現する世界初の革新的ソフトウェア基盤であり，計算科学の専門家のみで（計算＋データ＋学習）融合を容易に実現できる。ソースコード，マニュアル類も含めて一般に公開し，様々な環境で利用できるよう，普及に努める。h3-Open-BDEC 利用による（計算+データ+学習）融合シミュレーションにより，従来手法と同等の正確さを保ちつつ，大幅な計算量・消費電力削減を目指す。

3.3 研究事例：リアルタイムデータ同化と強震動シミュレーションの融合（h3-Open-APP，h3-Open-DATA，h3-Open-SYS, h3-Open-UTIL）

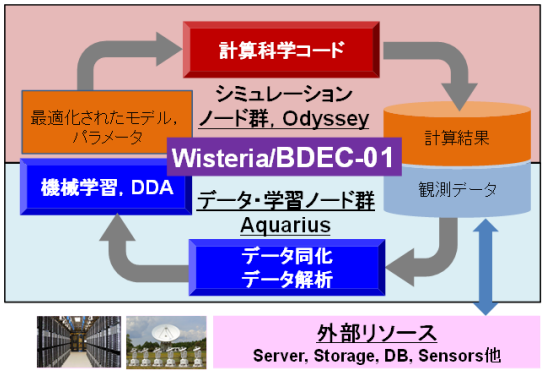

図 3 Wisteria/BDEC-01 利用による「計算・データ・学習」融合のイメージ〔7,8〕

計算科学シミュレーションは多くの場合，非線形な問題を扱うため，多数のパラメータスタディが必要である。Wisteria/BDEC-01 では，機械学習による最適パラメータ推定を，外部から取り込んだ実験・観測データによる同化と組み合わせて，正確な解をより短時間で求めることを目指している。

図 3 は，BDEC 上における「計算＋データ＋学習（S+D+L）」融合のイメージである。h3-Open-BDEC を使用することによって，シミュレーションノード群で計算科学シミュレーションコードを実行し，データ・学習ノード群では外部から取り込んだ観測データや，機械学習による推論等に基づきパラメータを最適化し，更に計算を実施するというサイクルを容易に実現することができ，またパラメータ最適化によって計算時間を全体として短縮できることが期待される。

シミュレーションとデータ科学の融合としては気候・気象シミュレーションとデータ同化の事例が良く知られているが，東大 ITC では東京大学地震研究所と協力して，三次元強震動シミュレーションとリアルタイムデータ同化を融合させた新しいシミュレーション手法の開発を実施している。古村等は，最適内挿法によるデータ同化と三次元強震動シミュレーションを組み合わせた Seism3D/OpenSWPC-DAF（Data-Assimilation-Based Forecast）の研究開発を実施しており，既に既存の地震観測データファイルを使用して「シミュレーション＋データ同化」融合に成功している〔9〕。

また，東大 ITC では，2019 年度からリアルタイム観測データを使用して「シミュレーション＋データ同化」融合を実施するためのフレームワークの研究開発を実施している。東大地震研，防災科技研，気象庁によって整備された JDXnet によって，全国 2,000 地点以上，100Hz で計測されている地震観測データをリアルタイムに取得できる。現在，東大 ITC の Oakbridge-CX システム（OBCX）の外部接続ノードを使用して JDXnet から得られる観測データを直接利用可能となっている〔10,11〕。

h3-Open-SYS は Wisteria/BDEC-01 のようなヘテロジニアスなシステムにおいて，シミュレーションとデータ処理実行を統合するソフトウェア群である。h3-Open-SYS/WaitIO〔10,11〕はその中核機能として，ファイルシステムを通じて複数の並列プログラムがデータの受け渡しを行うライブラリである。多くのスパコンで提供されている共有ファイルシステムをプログラム間のデータ連携手段として使用することで高い汎用性を確保し，ファイルを通信手段として用いる際に一般的に問題となる同期の問題を WaitIO ライブラリで解決する。

h3-Open-SYS/WaitIO は，本研究で Oakbridge-CX を使用して実施したリアルタイムデータ同化と強震動シミュレーションの融合に適用されている（図 4）〔10，11〕。2021 年

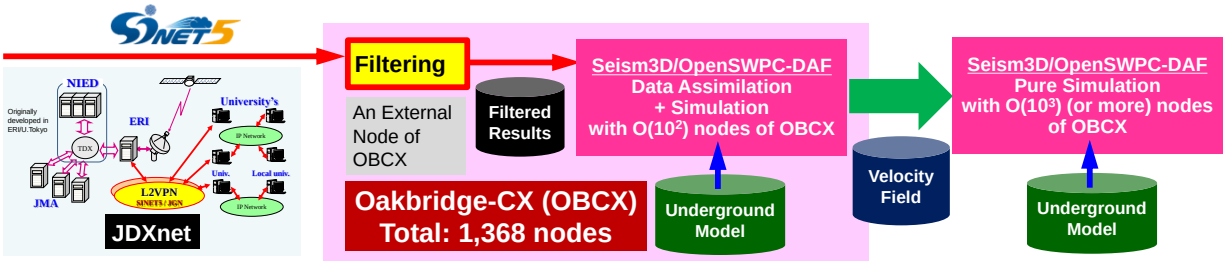

図 4 Oakbridge-CX 上で実施したリアルタイムデータ同化と強震動シミュレーションの融合〔10,11〕

度は，これらを Wisteria/BDEC-01 へ展開する。更に，h3-Open-BDEC の多機能カプラである h3-Open-UTIL/MP を使用して，データ同化，機械学習による三次元地下構造モデル構築システムを開発し，より精度の高いシミュレーションを実施するための検討を実施する。

4. h3-Open-SYS/WaitIO

h3-Open-SYS/WaitIO（図 4）〔10，11〕は２つ以上の並列プログラム間で通信を行うためのライブラリである。各並列プログラムは典型的には MPI プログラムであるが、MPI 以外の形の並列プログラムからも利用できるよう汎用的に設計されている。

h3-Open-SYS/WaitIO は多様な並列計算機環境を想定している。特に、通信を行う並列プログラムが同一の並列計算機で動作する場合はもちろん、前述の Odyssey と Aquarius のように各々の並列プログラム異なる並列計算機で動作する場合にも対応する。そのために通信経路としてファイルと TCP/IP を想定する。

ファイル経由での通信を行う h3-Open-SYS/WaitIO を特に h3-Open-SYS/WaitIO -File（以下、簡単に WaitIO-File）と呼ぶ。WaitIO-File は同時実行される並列プログラムの間の通信手段が共有ファイルシステムに限られる状況を想定し、共有ファイルを通信路として使用するものである。一般にスーパーコンピュータは、並列プロセスに対して割り当てたノード群から外への通信を許さない運用が多いため、共有ファイルを用いたデータの受け渡しは非効率であるものの現実的な手段である。WaitIO-File は通信を行うプロセスのペアそれぞれに対して通信方向別に 1 個ずつのファイルを準備する。送信側は送信データをファイル末尾に追記し続け、受信側はそのファイルを読み込む動作が基本であるが、特に受信側はファイルの末尾以降を読み込もうとした場合には新たなデータの追記を待つ点が通常のファイル I/O と異なる（WaitIO はこの特徴を表した名前である）。このファイルの追記を効率的に待つ手法、また、送受信が完了したデータを削除する方法について、ファイルシステムに依存した細かな最適化が必要であり、本稿執筆時点では基本的なアイディアを列挙している段階である。本稿では WaitIO-File については詳細には議論しない。

TCP/IP を用いて通信を行う h3-Open-SYS/WaitIO を特に h3-Open-SYS/WaitIO-Socket（以下、簡単に WaitIO-Socket）と呼ぶ。WaitIO-Socket は並列プログラムを構成する各プロセスが、別の並列プログラムの各プロセスに直接通信できる環境を想定している。前述のように現在のスーパーコンピュータの通常の運用ではこのような通信は許されていないことが多いが、並列計算機を構成するインターコネクトは TCP/IP での通信をサポートしていることが多いため、セキュリティー上の問題を考慮の上、運用方針を明確にすれば技術的には容易に実現可能な環境と言える。WaitIO-Socket の基本アイディアは、並列プログラムを構成する各プロセスが通信相手の並列プログラムの全プロセスに対して TCP/IP のコネクションを張り通信を行うものである。この基本アイディアはシンプルであるが、各並列プロセスが 1 万を超える数のプロセスから構成されるような大規模環境を想定する際に、通信ソケットだけでメモリ資源を使い尽くすようなことがないよう、設計、実装上の工夫が必須である。本稿ではこの WaitIO-Socket について詳細に議論する。

5. h3-Open-UTIL/MP

前節で述べたとおり h3-Open-SYS/WaitIO は並列プログラム間通信ライブラリである。このライブラリを基盤として複数の並列プログラムを連成するための上位ソフトウェア(カプラ)が h3-Open-BDEC プロジェクトの一環として h3-Open-SYS/WaitIO と並行して開発されている。このカプラを h3-Open-UTIL/MP と称する。h3-Open-UTIL/MP は異なる格子系を持つ複数のモデルコンポーネントに対して設定された時間間隔でデータを交換し格子変換を行う。更に、これらの基本機能に加えて、結合されたモデル群を並列に実行し統計処理を行う結合アンサンブル機能や Python アプリケーションを結合するための Python インタフェースを装備している。モデルと結合される Python アプリケーションには I/O や作画ライブラリなどが想定されるが、有力なアプリケーションの一つが機械学習である。シミュレーションモデルを機械学習と結合することで、計算負荷の高いプロセスを機械学習に代替させる、パラメータを用いない高解像度計算の結果をパラメータ化が必要な低解像度計算に反映させるといった、多くの利点が得られる。このような背景に基づき、大気モデル NICAM と機械学習ライブラリ PyTorch を結合するプロジェクトが進行中であり、既に予備的な結果が得られている。しかしながら、PyTorch による学習は計算負荷が高く実行のボトルネックとなっている。この状況を改善するにはシミュレーションモデルをシミュレーションノード群(Odyssey)で、機械学習ライブラリをデータ・学習ノード群(Aquarius)で実行し結合する方法が考えられる。この機能を実現するためのプログラム構成を図 5 に示す。最下層は機種をまたぐ大域通信を行うレイヤで MPI と WaitIO-Socket を併用し大域通信を実現する。第 2 層目はアプリ内通信とアプリ間通信を行うレイヤで、アプリ内は従来通りの MPI を、アプリ間は WaitIO-Socket を用いて通信を行う。第 3 層目は MPI の wrapper で、これにより従来の MPI ルーチンコールとシームレスに異機種間通信ができるようにする。

h3-Open-UTIL/MP が用いているコミュニケータの種類を図 3 に示す。図で色分けされた小さな四角の集合が一つのモデルコンポーネントを表す。用いるコミュニケータは全体通信に用いる Global, モデル間通信に用いる Model, 各

ⓒ2021 Information Processing Society of Japan

モデルの 0 番プロセスの集合である Leader, モデル内通信に用いる Local の 4 種類である。これらのコミュニケータに対して使われる MPI ルーチンの種類を表 2, 表 3 に示す。表でハッチを施されたセルが WaitIO-Socket を用いる異機種間通信である。

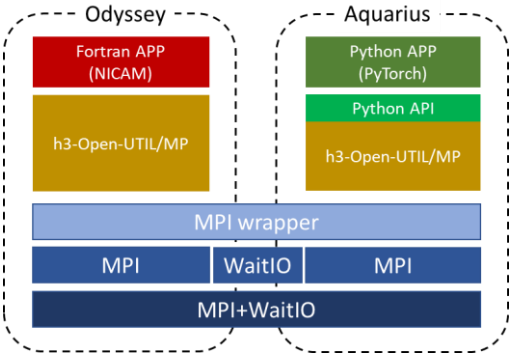

図 5　WaitIO-Socket と MPI を併用した異機種間結合プログラムの構成

表 2 の局所通信では MPI_Isend, MPI_Irecv, MPI_Wait, MPI_Waitall のみが用いられ、これらは WaitIO-Socket でも同様のルーチンが用意されていることからそのまま WaitIO-Socket に移行可能である。表 3 に示される大域通信で図 5 第 1 層にあたる MPI+WaitIO-Socket の併用が必要なルーチンは特殊なルーチンである Barrier を除くと Bcast, Gather, AllReduce, Reduce の 4 種類のみであり、これらを適切に実装すれば h3-Open-UTIL/MP を異機種間結合に対応させることができる。

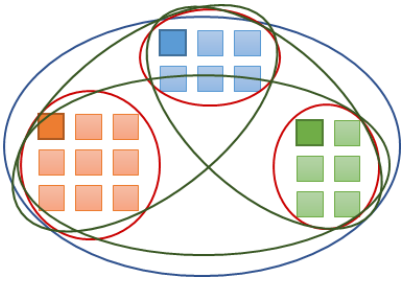

図 6　h3-Open-UTIL/MP が用いるコミュニケータ

表 2 各コミュニケータで用いられる局所通信ルーチン

局所通信	Global	Model	Leader	Local
Isend		●	●	●
Irecv		●	●	●
Wait		●	●	●
WaitAll		●		

実装に際しては WatiIO-Socket が開発中であることから WaitIO-Socket の API を MPI で実行するエミュレータを作成し、この上に大域通信のルーチン群を構築した。この際、WatiIO-Socket が担当するマシン間の通信は双方の 0 番プロセス(King)のみが行うものとしている。これは大域通信に関わる全プロセスが一斉に WaitIO-Socket による通信を行うと性能上の問題が生じる懸念があるためである。

表 3 各コミュニケータで用いられる大域通信ルーチン

大域通信	Global	Model	Leader	Local
Bcast	●		●	●
Gather			●	●
Scatter				●
GatherV				●
ScatterV				●
AllReduce	●			●
Reduce	●			●
Barrier			●	

MPI と WaitIO-Socket を併用した大域通信の例を図 7 に示す。図は MPI_Bcast の事例で、黄色い四角が Bcast の起点となる root プロセスを表す。まず root から root が所属する King にデータを送受信する。この通信は MPI で行われる。次いで King 同士の通信を WaitIO-Socket で行い、最後に各マシンの King が MPI_Bcast でデータを分配する。大域通信ルーチン群の実装は完了しているため、今後はこれらを h3-Open-UTIL/MP に適用する作業を進めるとともに、エミュレータを実際の WaitIO-Socket に置き換え異機種間結合を実現する予定である。

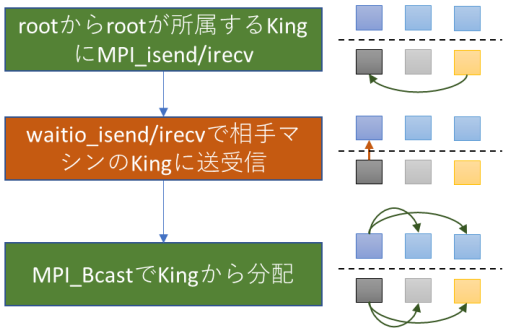

図 7　MPI と WaitIO-Socket を併用した大域分配の概要

6. WaitIO-Socket 実現の要件と設計方針

本章では 2 章で述べた Wisteria/BDEC-01 のシステム構成を元に WaitIO-Socket 実現の要件と設計方針を述べる。

6.1 WaitIO-Socket の要件

WaitIO-Socket 実現の要件を以下に述べる。

1. 異種システム上で動作可能： WaitIO-Socket では、Wisteria の 2 つのノード群上の複数アプリケーションを結合し相互のデータ交換を実現する必要がある。そのためには、ハードウェア、ソフトウェアが異なる 2 つのノード群上で動作する必要がある。
2. 複数のアプリケーション群を手間少なく結合可能で

ⓒ2021 Information Processing Society of Japan

あること： 既存ソフトウェア改変は最小限、かつ、結合部分のみのソフトウェア追加で実現可能なことが必要である。他のシステムソフトウェアと共存可能であるほか、プログラミングを容易にするために標準に近いプログラミングスタイルであることが必要である。

3. システム最大規模でアプリケーション実行可能であること: 数万プロセス規模で動作可能

以上の要件を満たす WaitIO-Socket を実現する。

6.2 WaitIO-Socket の設計方針

WaitIO-Socket の設計方針を以下に述べる。

1. Socket インタフェースの採用: 異種プロセッサ、異種インターコネクト、異種 MPI で構成されたプログラム間を結合するために通信プロトコルとして Socket インタフェースを採用する。Odyssey のインターコネクトである Tofu-D インターコネクトもインターコネクト上の TCP/IP 通信をサポートしており計算ノードとシステム外のシステムとの通信は Gateway ノードを経由して実現可能である。また、直接 Socket インタフェース上での通信ライブラリを実装することにより、MPI 他のシステムソフトウェアとの共存を可能とする。
2. MPI 仕様に準ずる API の提供: 提供される API や通信のセマンティックスは可能な限り通常利用の通信ライブラリと同じであることが望ましい。このため、WaitIO-Socket の API と通信のセマンティクスは MPI に準ずるものとする。
3. 最小限の計算機リソースで動作、全体同期の排除：数万プロセス規模で安定動作するためには、各プロセスが直接通信するプロセス数を最小限に抑制し使用するメモリ、CPU 等の計算機リソース使用を抑制する必要がある。通信プロセス数に比例して計算機リソースが必要だからである。また、全体プロセス間での同期は最小限とする必要がある。

以上の設計方針に基づき WaitIO-Socket を実装する。

7. WaitIO-Socket の実装概要

本章では WaitIO-Socket の実装概要について述べる。WaitIO-Socket の動作仕様、実装 API について述べた後、実装概要について説明する。

7.1 WaitIO の動作仕様

図 8 に WaitIO の動作イメージを示す。WaitIO は複数のアプリケーションを結合可能である。各アプリケーションは Parallel Block (以降、PB)と呼ばれる複数のプロセスグループで管理される。複数の PB から構成される WaitIO 全体は WaitIO Instance と呼ばれ、各 PB は PBID と呼ばれる識別子で管理されている。PBID=0 の PB を MASTER と呼ばれ、プログラムの起動時に初期化され、構成される PB で間で同期して各プロセス情報を交換する。初期化に必要な情報はシェルの環境変数で定義され、WAITIO_MASTER_HOST, WAITIO_MASTER_PORT, WAITIO_PBID, WAITIO_NPB (構成される PB 数)を各アプリケーションの実行時に設定する。

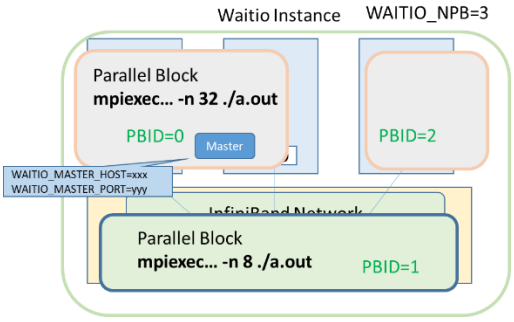

図 8 WaitIO の動作イメージ

7.2 実装 API の概要

表 4 に WaitIO-Socket の API を示す。初期化とサポート関数以外は 基本通信関数 Non-Blocking Send/Recv と Wait 関数だけである。

表 4 WaitIO-Socket API

WatiIO API	概要
int waitio_isend(waitio_group_t grp, int dst, char *buf, size_t len, waitio_tag_t tag, waitio_req_t *req);	Non-Blocking 送信
int waitio_irecv(waitio_group_t grp, int src, char *buf, size_t len, waitio_tag_t tag, waitio_req_t *req);	Non-Blocking 受信
int waitio_wait(waitio_req_t *req);	送受信完了待ち合わせ
int waitio_init(int timeout);	WaitIO 初期化
int waitio_get_nprocs(int **array);	PB 毎の参加プロセス数獲得
waitio_group_t waitio_create_group(int gid, int *rank[], int order[]); waitio_group_t waitio_create_group_func(int gid, waitio_filter_func_t func[], int order[]);	PB 間通信グループ生成 (メンバ配列指定、関数指定)
int waitio_group_rank(waitio_group_t grp, int *rankp);	グループ内 Rank の獲得
int waitio_group_size(waitio_group_t grp, int *sizep);	グループサイズの獲得
int waitio_pb_size(int *sizep);	全 PB のサイズ獲得
int waitio_pb_rank(int *rankp);	全 PB 内 Rank の獲得

ⓒ2021 Information Processing Society of Japan

7.3 実装概要

- 初期化：waitio_init()呼び出し時、すべての PB 上のプロセスで作成された IP アドレスと Port 番号のセットをすべての PB 上のプロセス間で共有する。このため、waitio_create_group()呼び出し時に選択されたプロセス情報でグループを作成する。このグループ作成はすべてプロセス内で処理が完了するため、MPI の MPI_Comm_split()などのコミュニケータ生成時のように全体同期は不要である。なお、グループ作成時はテーブル作成のみであり、各プロセス間の通信は初回の通信時に受信側から確立する。
- 通信プロトコル：各プロセス間の通信プロトコルは 128 byte まで制御・データ一体型の Eager プロトコル、128byte を超える場合は Rendezvous プロトコルを採用している。制御・データ分離型の Eager プロトコルについては unexpected メッセージによるメモリ使用量の増加が懸念されるため現状実装していない。また、TCP/IP 処理で 64KB 毎に分割されるため、64KB を超える場合は送信時に 64KB 単位に分割して送信している。
- 通信処理のプログレス処理：通信確立済みの Eager プロトコル以外、waitio_wait()関数呼び出し時に通信のプログレス処理が実行される。
- 全体通信処理実装概要:
  - 受信処理はすべて Linux の epoll(event polling)関数群を用いて実装している。
  - 送信処理は基本直接 write システムコールで実装、socket への write 処理が EINTR 要因他のエラーとなったときに epoll による処理に移行する。すべての write 処理が完了した時点で epoll 処理を解除する。
  - 通信のデータ構成：固定長の制御パケットと可変長のデータパケットから構成される。
  - 通信の信頼性確保：ソフトウェア、ハードウェアエラーによるデータロスを検出するため、制御パケットにはマジック番号とシーケンス番号を導入している。制御パケット受信時には、正しい制御パケットが正しい順序に受信されているか確認し、通信の信頼性を確保している。
- 利用するネットワークデバイスはシステムにより異なるためシステムごとに指定可能としている。
- プロセッサの Endian type は現状 Little のみ実装。
- 実装コード：C でライブラリ 5.7Kstep, テストプログラム 1Kstep である。

8. WaitIO-MPI Conversion ライブラリ

WaitIO はシンプルな通信に特化するため MPI の Datatype に相当する概念がない。このため、MPI で書かれたプログラムを WaitIO に書き直すのはひと手間かかる。プログラムを書き直すとプログラム実行の不具合につながる。したがって、可能な限り機械的に書き直せることが望ましい。

このため、MPI とのインタフェース変換を担う WaitIO-MPI Conversion ライブラリを準備した。この WaitIO-MPI Conversion ライブラリは、必要になった関数を随時加えることとしている。API としては FORTRAN 向け API も実装している。

現状、実装されている関数を表 5 に示す。非同期送受信関数(isend, irecv)の他、集団通信関数(bcast, reduce, allreduce)であり、扱う Datatype は数種の基本 Datatype に限定している。アプリケーションの必要に応じて拡張する予定である。現状、関数名と datatype, op などの MPI オブジェクト表記は WAITIO_の prefix を機械的に追加することで WaitIO 向けに書き換えるようにしている。

表 5 WaitIO-MPI Conversion ライブラリ API

WaitIO-MPI API (2021.08/07 現在)	概要
int waitio_mpi_isend (const void *buf, int count, WAITIO_MPI_Datatype datatype, int dest, int tag, WAITIO_MPI_Comm comm, WAITIO_MPI_Request *request);	WaitIO 実装版 MPI_Isend
int waitio_mpi_irecv (void *buf, int count, WAITIO_MPI_Datatype datatype, int source, int tag, WAITIO_MPI_Comm comm,WAITIO_MPI_Request *request);	WaitIO 実装版 MPI_Irecv
int waitio_mpi_reduce (const void *sendbuf, void *recvbuf, int count, WAITIO_MPI_Datatype datatype, WAITIO_MPI_Op op, int root, WAITIO_MPI_Comm comm) ;	WaitIO 実装版 MPI_Reduce
int waitio_mpi_bcast (void *buffer, int count, WAITIO_MPI_Datatype datatype, int root, WAITIO_MPI_Comm comm) ;	WaitIO 実装版 MPI_Bcast
int waitio_mpi_allreduce (const void *sendbuf, void *recvbuf, int count, WAITIO_MPI_Datatype datatype, WAITIO_MPI_Op op, WAITIO_MPI_Comm comm) ;	WaitIO 実装版 MPI_Allreduce
int waitio_mpi_waitall (int count, WAITIO_MPI_Request *array_of_requests,int *array_of_statuses);	WaitIO 実装版 MPI_Waitall
int waitio_create_universe (WAITIO_MPI_Comm *commp) ;	Waitio 初期化サポート関数

実装されている集団通信アルゴリズムは動作確認と通信負

荷を目的としたシンプルな 2 階層のシーケンシャル実装(2 Layered:以降 2L)と Binary(Binominal) Tree ベース(以降 BT)である。シーケンシャル実装の Bcast は root プロセスから各 worker プロセスに送信、reduce は各 worker プロセスから root プロセスに送信、root プロセスで演算する原始的なものである。また、Allreduce は reduce+bcast 実装である。ただし 1 プロセスに同時通信できるソケット数が限られるため、一定プロセス数以上の場合は階層的に通信する実装としている。Binary(Binominal) Tree ベースは Binary Tree 上にプロセスをマップしメッセージ交換を実施する。

なお、現状の WaitIO-MPI Conversion ライブラリは実験的に WaitIO-Socket インタフェースのみを用いて実装しているが、各 PB 上の MPI 実装を用いた Hybrid 実装であっても良い。ただし、Hybrid 実装とした場合には既に実装済みの MPI 利用のコードと競合が発生する可能性があるので、Hybrid で実装するかについては今後精査する予定である。

9. WaitIO-Socket の基本通信性能評価

本章は WaitIO-Socket 通信ライブラリが Wisteria システム上でどの程度通信性能を引き出せるかを検証することを目的として基本通信性能評価を実施した結果を述べる。

9.1 評価環境と評価プログラム

WaitIO-Socket の評価においては、より広いシステムでも稼働することを確認するため Wisteria システム(表 1)の他に Oakbridge-CX, Oakforest-PACS を用いて実施した。(表 6)

表 6 Oakbridge-CX, Oakforest-PACS のシステム仕様

	Oakbridge-CX (OBCX)	Oakforest-PACS (OFP)
総理論演算性能	6.61 PFLOPS	25 PFLOPS
総ノード数	1368	8208
インターコネクト	Omni-Path (100Gbps)	Omni-Path (100Gbps)
プロセッサ/ノード	Intel Xeon Platinum 8280 2.7GHz, 2 socket (28+28)	Intel Xeon Phi 7250 1.4GHz 1 socket (68)
メモリ量/ノード	192 GB	96(DDR4)+16(MCDRAM) GB
メモリバンド幅/ノード	281.6GB/s	115(DDR4)+490(MCDRAM) GB/s
コンパイラ、MPI	Intel コンパイラ、インテル MPI、(Open MPI)	Intel コンパイラ、インテル MPI、(Open MPI)
オペレーティングシステム	Red Hat Enterprise Linux 7, CentOS 7	Red Hat Enterprise Linux 7, CentOS 7

評価プログラムはシステム間のデータ通信性能を評価するため複数ストリームでの PingPong 通信の評価プログラムを作成した。これは、全 PB 内プロセスで 2 プロセスペアを作り PingPong 通信を行うものである。すべてのペアの PingPong 通信終了を MPI_Barrier()で待ち合わせて測定している。

- Intra-Node の場合：偶数 rank+1 のプロセスをペア
- Inter-Node の場合：rank+1/2*(PB サイズ)のプロセスをペアで通信する。

実行評価プログラムはそれぞれのシステムにインストールされているベンダ製コンパイラ＋ベンダ製 MPI で評価してグラフ化したが Xeon プロセッサ搭載システムでは Open MPI+GCC の組み合わせでも動作確認している。

9.2 1PB 内 1 対 1 通信評価(multi-stream)

本節では 1 PB 内でのマルチストリーム 1 対 1 通信評価について述べる。

9.2.1 PingPong 1/2 RTT 評価

本節では各システムで測定した PingPong1/2 RTT の性能評価について述べる。それぞれ、図 9、図 10 に Odyssey、図 11、図 12 に Aquarius、参考として付録の図 31、図 32 に Oakbridge-CX、図 33、図 34 に Oakforest-PACS の 1/2 RTT(ノード内、ノード間)の測定結果を示す。各データはノード数を固定、ノードあたりのプロセス数を増加させ、複数プロセスによる複数ストリームを 1000 回実行し MPI_Barrier で同期後の時間を回数分で割り算した平均値を示している。すなわち、各回のストリーム実行の最悪値の平均である。

表 7 各システムの 1/2RTT サマリ

usec(8B)	1/2 RTT(ノード内)	1/2 RTT(I ノード間)
Odyssey	26.8	37.1
Aquarius	6.57	21.1
OBCX	6.85	14.5
OFP	49.7	83.7

表 7 に各システムの 8 Byte の結果のサマリを示す。

表 7 より、Intel Xeon プロセッサシステムのノード内の測定結果は、ほぼ同等の実行時間である。一方、Odyssey, OFP については Xeon 搭載システムに比べ 4 倍弱(Odyssey)、9 倍弱(OFP)遅い結果となっている。相対的なプロセッサ処理の差であるほか、Odyssey の結果に関して、カーネル内処理はアシスタントコアで実行されるためにその遅延も含まれていると想定される。

全体的な傾向としては、通信プロトコルが Eager プロトコルから Rendezvous プロトコルに切り替わる地点で段差が見られる。ノードあたり 2 プロセスの結果ではノード内通信では 2 倍程度、ノード間通信では 3 倍から 5 倍程度の差であった。この実行時間はノード内プロセスが増加するにつれ増加傾向にあった。カーネル内処理の競合や同期の時間が関連していると想定される。

ⓒ2021 Information Processing Society of Japan

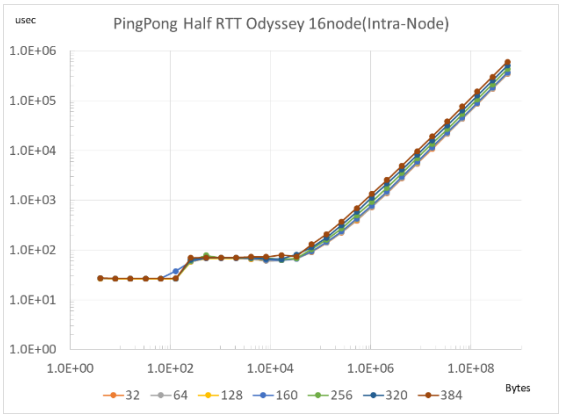

図 9　Odyssey 1/2 RTT 16node　ノード内通信

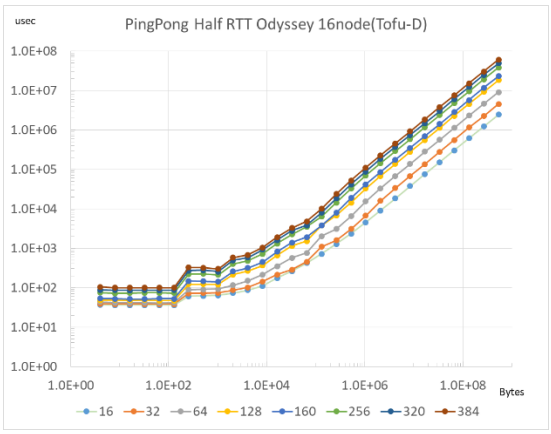

図 10 Odyssey 1/2 RTT 16node　ノード間(Tofu-D)

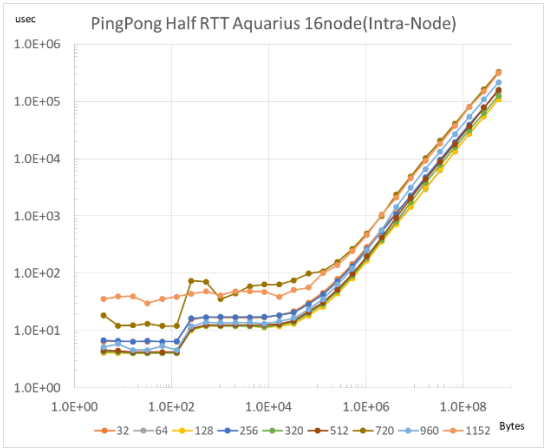

図 11 Aquarius 1/2 RTT 16node　ノード内

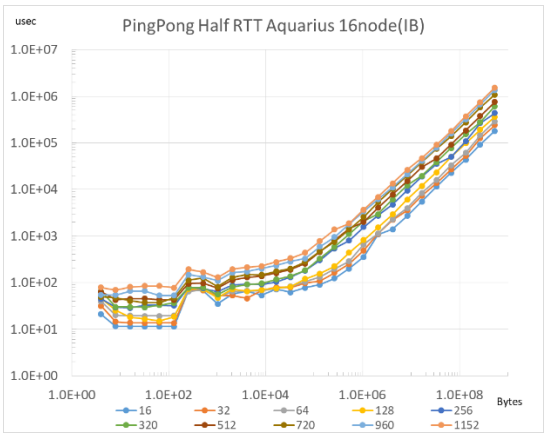

図 12 Aquarius 1/2 RTT 16node　ノード間(InfiniBand HDR)

9.2.2 PingPong バンド幅評価

本節では各システムで測定した PingPong バンド幅の性能評価について述べる。それぞれ、図 13 図 14 に Odyssey、図 15 図 16 に Aquarius、参考として付録、図 35 図 36 に Oakbridge-CX、図 37 図 38 に Oakforest-PACS の PingPong バンド幅(ノード内、ノード間)の測定結果を示す。各データは複数プロセスによる複数ストリームを 1000 回実行し MPI_barrier()で同期後の時間を回数分で割り算した平均値を示している。

表 8 各システムの PingPong バンド幅サマリ

PingPong BW GB/s	16 Nodes(8 sets)	Per N-to-N
Odyssey(Intra)	170.4	21.3
Odyssey(Tofu-D)	2.80	0.35
Aquarius(Intra)	2077.6	259.7
Aquarius(InfiniBand)	216.6	27.1
OBCX(Intra)	365.7	45.7
OBCX(OmniPath)	74.6	9.3
OFP(Intra)	65.6	8.2
OFP(OmniPath)	8.96	1.12

表 8 に各システムの最大値と 1 ノードあたりの PingPong バンド幅のサマリを示す。表 8 より、各システムのメモリ帯域、インターコネクト帯域、そしてコピー処理、通信プロトコルを実行するプロセッサの処理性能の影響が見られる結果となっている。

ⓒ2021 Information Processing Society of Japan

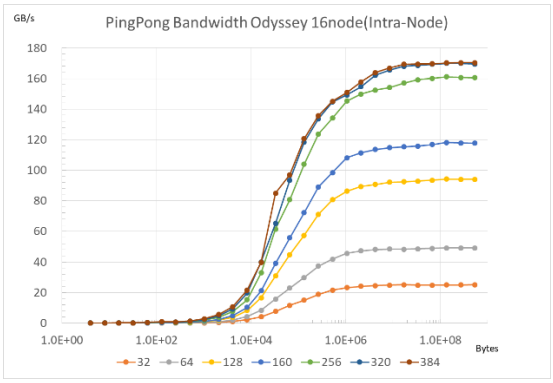

図 13 Odyssey 通信バンド幅 16node ノード内

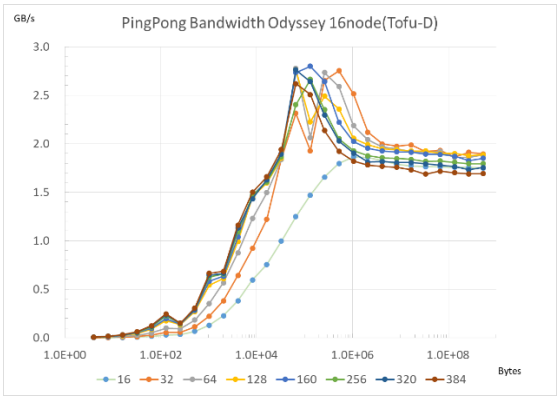

図 14 Odyssey 通信バンド幅 16node ノード間(Tofu-D)

最も高い性能を記録したのは、Aquarius である。ノード内性能ではノードあたり 260GB/s、ノード間性能でもノードあたり 27GB/s を観測した。PingPong 処理においては WaitIO-Socket ライブラリ-カーネル間コピーが 2 回発生する。このため Aquarius の 960 プロセス結果のメッセージ長が 8MB までの区間で大きく山が出ているのはキャッシュの影響であると想定される。

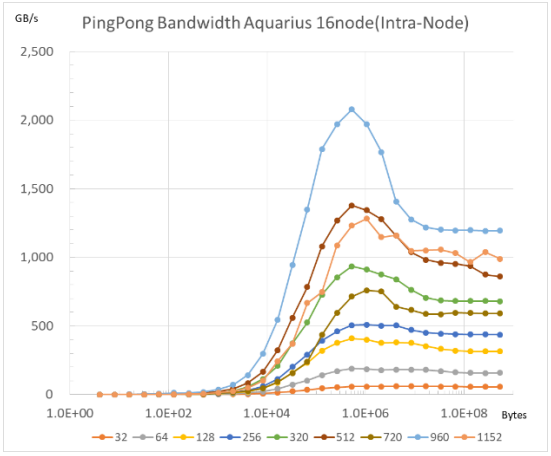

図 15 Aquarius 通信バンド幅 16node ノード内

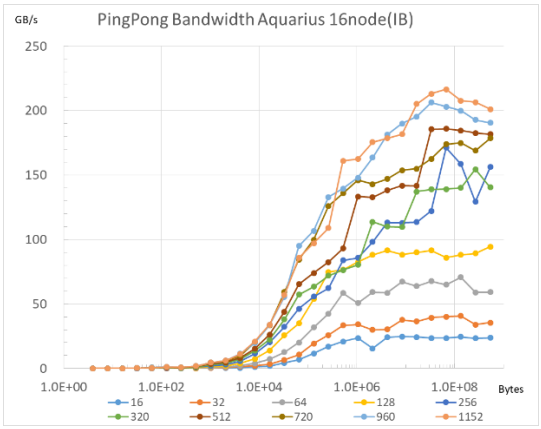

図 16 Aquarius 通信バンド幅 16node ノード間 (InfiniBand-HDR)

次にノード間のインターコネクト毎の性能差に着目すると、やはり最新の InfiniBand HDR 4 本を搭載する Aquarius(図 16)がプロセッサ性能とのバランスに優れ ノードあたり 27.1GB/s の通信性能を記録している。物理性能が半分の OmniPath を搭載している OBCX(表 8)は 9.3GB/s を記録している。その性能差は 2.9 倍であった。

一方、同じ OmniPath を搭載する OFP の結果(表 8、図 38)については 1.1GB/ s と OBCX に比べ 8.5 倍の性能差である。OFP に関してはノード内、ノード間とも他のシステムに比べ大きな性能差がないために、プロセッサコアの処理性能で律速していると想定される。

最後に Odyssey の結果(表 8、図 13、図 14)については、ノード内の通信性能に比べ、ノード間の性能が 61 倍低い結果となった。これはノード内通信が各計算コアによるシステムコール経由のメモリコピーで済むのに対して、ノード間通信はバックグラウンドで実行される TCP/IP 処理が 1 ノードあたり 2 or 4 コアのアシスタントコアで実行されるために、このアシスタントコア処理がボトルネックとなり性能が律速されていると考えられる。

9.3 大規模環境での評価

本節では Wisteria システムに置ける大規模プロセス環境での評価結果について述べる。測定は前節と同じ PingPong プログラムであるが、この測定では 1 ノードあたりのプロセス数を固定でノード数を変化させた結果である。

表 9 に Odyssey、Aquarius の最大 PingPong バンド幅の測定結果のサマリを示す。Odyssey では最大 49152 プロセスでの動作を確認した。

表 9 、図 21、図 22 より、Odyssey の 2048node システムにおいてはノード内合算 20.8TB/s、ノード間合算 208GB/s の通信バンド幅性能が得られている。ノード内の通信性能についてはプロセッサコア性能で通信性能が律速されているためメッセージサイズが増加しても性能劣化が見られない。また。ノード間の通信性能については、1 ノードあたりの通信性能は 0.2GB/s と高くないが、ノード数

ⓒ2021 Information Processing Society of Japan

を増やすことによりスケーラブルな通信性能が得られている。

特にノード内での通信は実質メモリコピーであるため同一ノードに 2 つのジョブが共用できる環境を作ることができれば一桁高いデータ転送性能が得られる。

表 9 Wisteria 大規模プロセス環境評価サマリ

PingPong BW GB/s	BW/# Nodes	Per node-to-node
Odyssey(Intra)	20,805/2048	20.8
Odyssey(Tofu-D)	208.2/2048	0.21
Aquarius(Intra)	2195.8/36	122
Aquarius(InfiniBand)	409.4/36	22.7

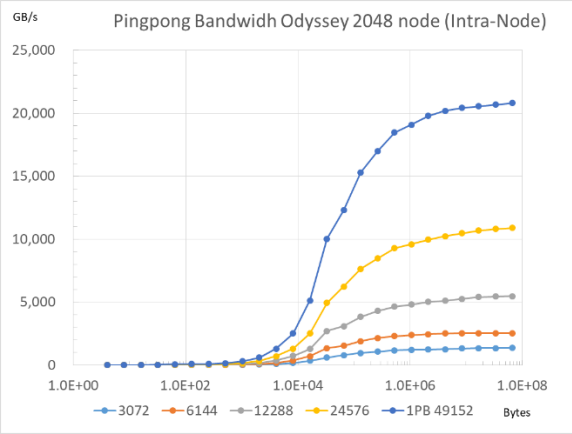

図 17 Odyssey 通信バンド幅 2048node ノード内

Aquarius については表 9 より、36node でノード内 2.2TB/s ノード間では 409GB/s の通信性能が得られている。ノード数は少ないが 1 ノードあたりの通信性能が高いといえる。また、図 19、図 20 よりメモリバンド幅の制約からノード内の 8MB 以上のメッセージサイズでは Odyssey と異なり性能劣化が見られた。

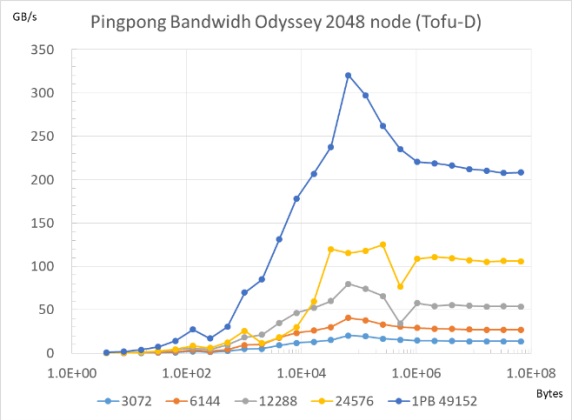

図 18 Odyssey 通信バンド幅 2048node ノード間 (Tofu-D)

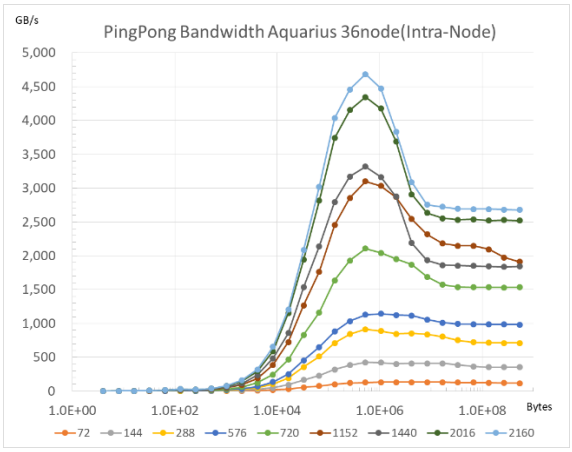

図 19 Aquarius 通信バンド幅 36node ノード内

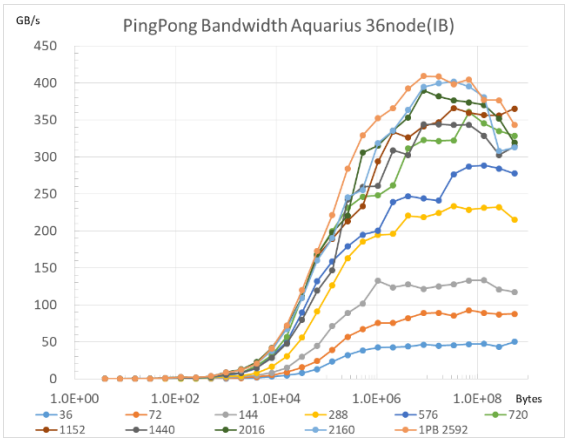

図 20 Aquarius 通信バンド幅 36node ノード間 (InfiniBand HDR)

9.4 2PB 結合での通信性能評価

本節では複数ジョブを WaitIO-Socket で結合した場合の通信性能について評価する。前節の大規模環境評価を 1 ジョブ(1 PB:Parallel Block)と 2 ジョブ(2 PB)で実行し性能差を評価する。

図 21、図 22 に Odyssey 1 PB(2048node) と 2 PB (1024+1024node)、図 23、図 24 に Aquarius 1 PB(36node)と 2 PB(18+18node)のノード内、ノード間の通信性能の比較結果を示す。

図 21 より、Odyssey のノード内通信性能についてはほぼ同等性能となっている。図 22 では少し性能にバタつきが見られるが傾向は類似という結果となった。

Aquarius については図 23、図 24 より 1PB より 2PB の方が高い性能となっているが概ね同様の傾向の性能結果となった。実装上 1PB でも 2PB でも相手の IP アドレスと Port 番号をベースに相手と通信する仕組みは変わらないので、1 ジョブ 1PB と 2 ジョブ 2PB の性能差が出る可能性としては、ジョブの割り当て場所、ランク割り当てノードとインターコネクト上の位置の違いが考えられる。

ⓒ2021 Information Processing Society of Japan

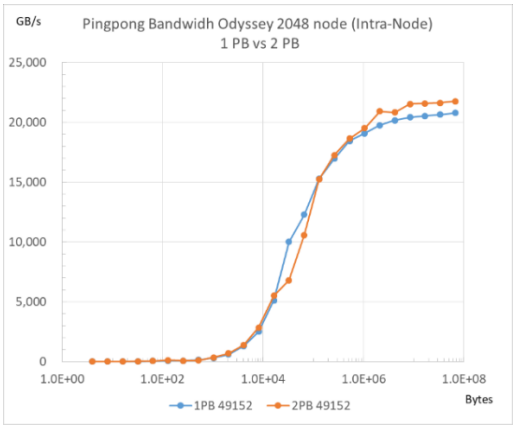

図 21　Odyssey 通信バンド幅 2048node ノード内 1PB vs 1PB

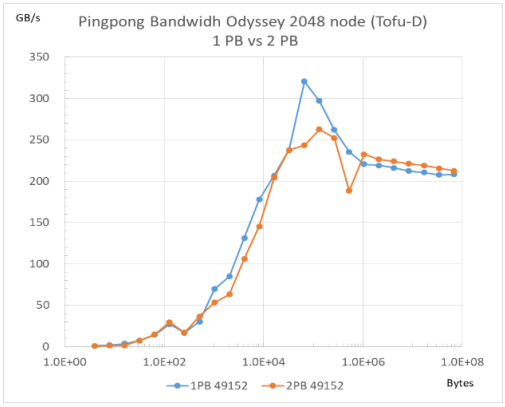

図 22　Odyssey 通信バンド幅 2048node ノード間 (Tofu-D) 1 PB vs. 2 PB

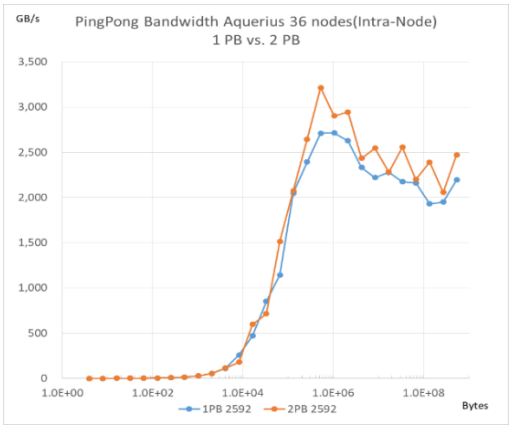

図 23　Aquarius 通信バンド幅 36node ノード内 1 PB vs. 2 PB

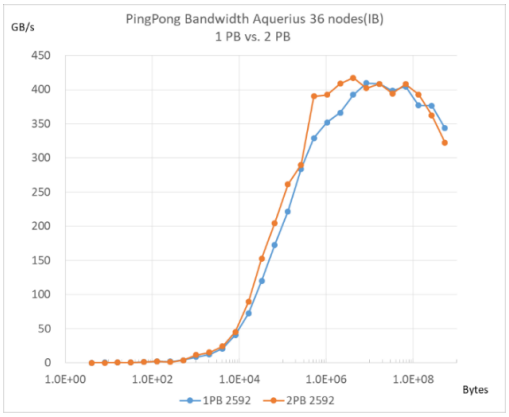

図 24　Aquarius 通信バンド幅 36node ノード間 (InfiniBand HDR) 1PB vs. 2 PB

なお、Odyssey16 ノードとフロントエンドマシン 1 ノード (4 プロセス)での 2PB 実行についても PingPong プログラムが正常に実行できることを確認した。

以上のように、大規模複数 PB での実行、異種プロセッサ、異種インターコネクト間での通信を実現できることを確認した。

10. h3-Open-UTIL/MP の評価

本章では h3-Open-UTIL/MP (以降 MP)の通信性能を評価する。MP は PB 内通信を高速なインターコネクト利用の MPI で通信、PB を跨る部分のみ WaitIO を利用する通信ライブラリである。

比較対象として WaitIO を用いた実装の WaitIO-MPI Conversion ライブラリ 2 種(2 階層シーケンシャル 2L と Binary Tree ベース BT)の通信性能を測定し比較する。

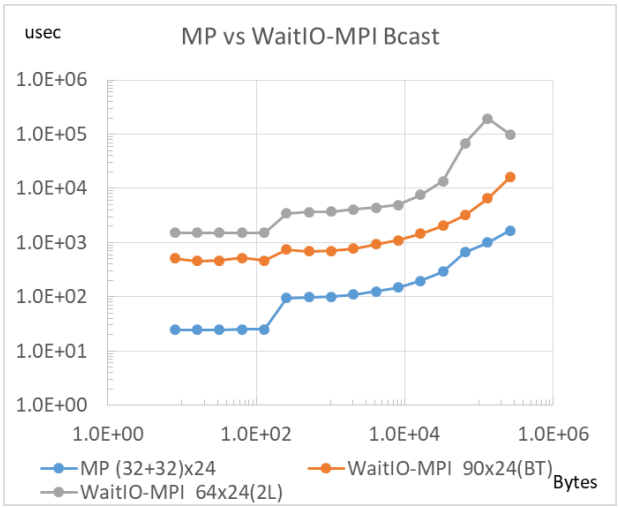

図 25　MP vs WaitIO-Socket 実行時間 Bcast

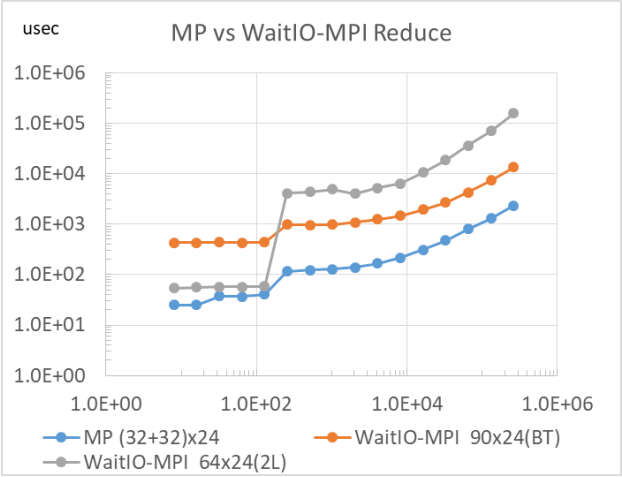

図 26 MP vs WaitIO-Socket 実行時間 Reduce

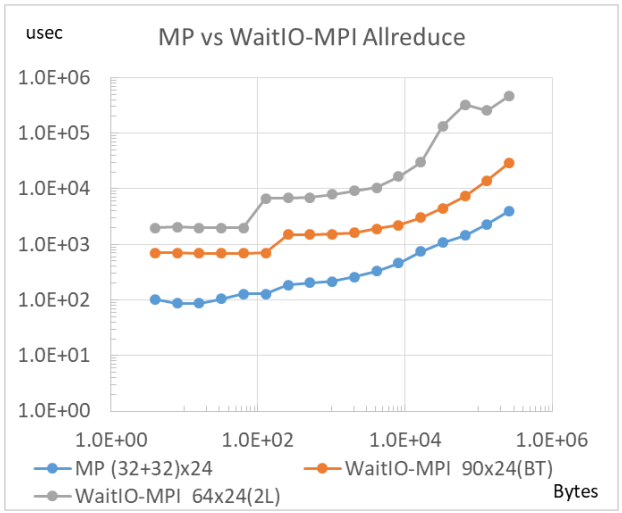

図 27 MP vs WaitIO-Socket 実行時間 Allreduce

図 25, 図 26, 図 27 に Odyssey 16 ノード 1536 プロセス(BT は 2048 プロセス)で実行したそれぞれ Bcast, Reduce, Allreduce の結果、表 10 に比較のための MP と BT の性能結果サマリを示す。

表 10 MP vs WaitIO-Socket(BT)実行時間サマリ

	MP usec	WaitIO-MPI Lib(BT) usec	WaitIO-MPI Lib(BT)/iMPI
Bcast 8 bytes	24.7	509.3	20.6
Bcast 256 bytes	94.4	744.2	7.9
Reduce 8 bytes	24.9	433.6	17.4
Reduce 256 bytes	116.2	985.0	8.5
Allreduce 8 bytes	102.0	698.5	6.8
Allreduce 256 bytes	186.7	1500.0	8.0

表 10 の結果より、WaitIO-MPI BT との比較では 256Byte の結果はおおむね 8 倍程度 MP での実装が速い結果となった。8 Byte の結果は Bcast, Reduce が 20 倍程度、Allreduce が 7 倍の結果となり、ハイブリッド実装の効果は明確にあることが分かった。一点、図 28 の 2L の 128Byte 以下の結果が MP の結果とそん色がないのは、2L の実装では各プロセスは Eager プロトコルで ROOT プロセスに送信して完了、ROOT プロセスは各プロセスとの同期なく受信したデータを演算するのみで済むからである。

11. アプリケーションを用いた WaitIO-MPI Conversion ライブラリの評価

WaitIO-MPI Conversion ライブラリを用いて pHEAT-3D アプリケーション[17]を移植し実行時間を評価した。pHEAT-3D は GeoFEM[18]プロジェクトで開発された並列有限要素法アプリケーションを元に開発した三次元定常熱伝導解析コードである。実行環境は OBCX(表 6)であり比較対象は Intel MPI(OmniPath)である。評価結果はノードあたり 8 プロセス、1 プロセスあたり 6 スレッド、1000STEP の実行時間の結果である。WaitIO-Socket 版 pHEAT-3D では現状の実装関数の制限より Solver 部分のみ WaitIO-MPI Conversion ライブラリ(表 5)で実装している。移植作業はエディタ上で機械的に実施可能で実際には WAITIO_MPI_Isend(), WAITIO_MPI_Irecv(), WAITIO_MPI_Allreduce() を用いて置き換えている。集団通信の利用アルゴリズムは BT である。現状の実装の制限で性能測定に関連する Solver 以外の通信は既存実装どおり MPI を用いている。このように Intel MPI ライブラリとの同時利用も可能である。

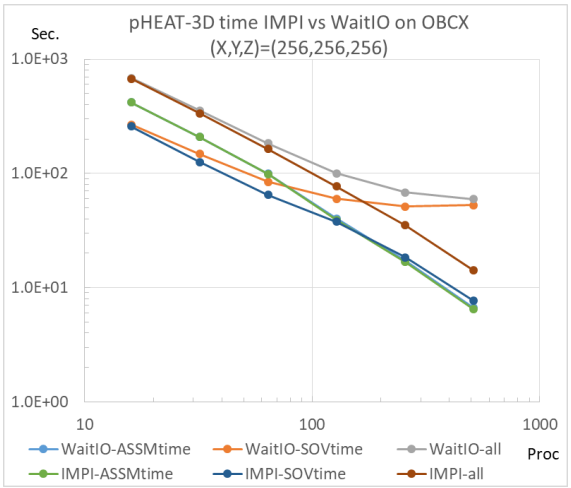

図 29 pHEAT-3D 実行時間：
Intel MPI(IMPI) vs. WaitIO-Socket on OBCX

図 29 に配列サイズ(X,Y,Z)=(256,256,256)の測定結果を示す。図 29 より、Intel MPI(IMPI)は 512 プロセスまで性能向上しているが、WaitIO-Socket の測定結果では 256 プロセス以上では性能が頭打ちであった。

ⓒ2021 Information Processing Society of Japan

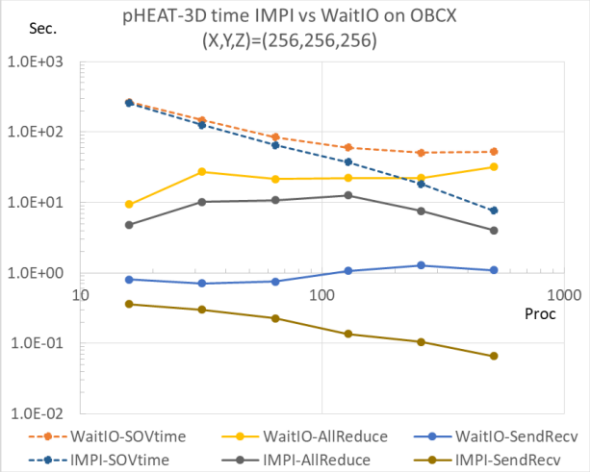

図 30 pHEAT-3D における通信時間

図 30 に pHEAT-3D で利用している 4 回の Allreduce、iSend/iRecv で実装された SendRecv 通信時間と Solver 全体の実行時間を示す。図 30 より AllReduce 通信が支配的であることが観測された。

なお、WaitIO-ASSEtime と Intel MPI-ASSEtime の性能差がほとんどないのは計算のみで通信を実施していないからである。

12. 関連研究

WaitIO-Socket で実装している複数の実行中のアプリケーションを動的に結合する方式に関する関連研究について述べる。

- MPI 規格[12]を用いる手法： MPI 標準規格実装である Open MPI[13], MPICH[14]を用いる方法としては MPI_Comm_spawn() と MPI_Comm_connect() / MPI_Comm_accept()を用いる手法がある。
  - Open MPI は同じ MPI ライブラリ内にでは異種ネットワーク、異種プロセッサ上でアプリケーション実行が可能であるが以下の課題がある。
  - 前者の MPI_Comm_spawn()はジョブスケジューラと MPI ライブラリが対応していれば、複数のアプリケーションを実行して相互通信が可能である。しかし、異種システム環境においては異種プロセッサ、異種インターコネクト上での MPI ライブラリの統一化は容易でないうえ、ジョブスケジューラの統合も既存の様々なシステム環境での実行を考慮すると現実的でない。
  - 後者の MPI_Comm_connect()/MPI_Comm_accept を用いる手法は実質的に各 MPI アプリケーションで Socket を生成するのと等価であるが一度通信が確立すると MPI ライブラリ通信として利用可能である。しかし、期待通りに動作しない場合があったため、採用していない。
- UCX[15], OFED[16]他複数プラットフォームに対応した低レベル通信を用いる手法：UCX, OFED ともに TCP/IP, 異種ネットワーク、異種プロセッサ上での動作実績がある。しかし、異種環境から構成される複数システムではすべての環境で利用可能であるかは不明なうえ、Odyssey のように UCX, OFED をユーザアプリケーションで直接サポートされていないシステムもあり利用に制約がある。

以上のように、既存の方式はそれぞれ制約があり WaitIO が目指す、より広範囲に動作する実行環境を提供するには課題がある。このため、我々は既存のシステムのほとんどが機能として持ち合わせている TCP/IP を用いた WaitIO-Socket を実現した。

13. おわりに

本論では、Wisteria/BDEC-01 システムの持つ大容量のネットワーク資源を活用するために h3-Open-SYS/WaitIO の機能を通信によって実現する WaitIO-Socket の試作について述べた。

WaitIO-Socket の実現においては、異種システム上で動作、複数のアプリケーション群を手間少なく実装、システム最大規模でアプリケーション実行するという要件の元、Socket インタフェースの採用、MPI 仕様に準ずる API、最小限の計算機リソースで動作、全体同期が初期化後には不要な通信ライブラリを実現した。

評価の結果、一通りの機能が正常動作することを確認し、50,000 プロセス規模での 2 つのジョブを結合して通信できることを確認した。

今後は、Odyssey の GIO を経由した Aquarius との結合試験を進めるとともに WaitIO-Socket 上での実アプリケーション実装と評価を進めていく。

謝辞 本研究の一部は科学研究費補助金（19H05662，代表：中島研吾）の助成を受けたものである。

参考文献

[1] Society 5.0（内閣府）：https://www8.cao.go.jp/cstp/society5_0/
[2] 中島研吾 他，Society 5.0 を実現する BDEC システム，AXIES 2020
[3] Wisteria/BDEC-01（東京大学情報基盤センター）：https://www.cc.u-tokyo.ac.jp/supercomputer/wisteria/service/
[4] 中島研吾 他，「計算・データ・学習」融合スーパーコンピュータシステム「Wisteria/BDEC-01」の概要，情報処理学会研究報告（2021-HPC-179-1）（第 179 回 HPC 研究会），2021
[5] Oakbridge-CX（東京大学情報基盤センター）：https://www.cc.u-tokyo.ac.jp/supercomputer/obcx/service/
[6] ppOpen-HPC: https://github.com/Post-Peta-Crest/ppOpenHPC
[7] h3-Open-BDEC: http://nkl.cc.u-tokyo.ac.jp/h3-Open-BDEC/
[8] Iwashita, T, Nakajima, K., Shimokawabe, T., Nagao, H., Ogita, T., Katagiri, T., Yashiro, H., Matsuba, H., h3-Open-BDEC: Innovative Software Platform for Scientific Computing in the Exascale Era by Integrations of (Simulation + Data + Learning), Project Poster, ISC-HPC 2020

ⓒ2021 Information Processing Society of Japan

[9] Furumura, T., Maeda, T., Oba, A., Early Forecast of Long‐Period Ground Motions via Data Assimilation of Observed Ground Motions and Wave Propagation Simulations, Geophys. Res. Lett., https://doi.org/10.1029/2018GL081163, 2018
[10] Nakajima, K., Matsuba, H., Hanawa, T., Furumura, T., Tsuruoka, H., Nagao, H., Integration of 3D Earthquake Simulation & Real-Time Data Assimilation on h3-Open-BDEC, MS290: Progress & Challenges in Extreme Scale Computing & Data SIAM Conference on Computational Science & Engineering (CSE21) (Online, March 4, 2021)
[11] SIAM News (March 10, 2021), Supercomputer Simulations of Earthquakes in Real Time (by Jillian Kunze), https://sinews.siam.org/Details-Page/supercomputer-simulations-of-earthquakes-in-real-time
[12] MPI: A Message-Passing Interface Standard Version 4.0: https://www.mpi-forum.org/docs/mpi-4.0/mpi40-
[13] Graham R.L., Woodall T.S., Squyres J.M. (2006) Open MPI: A Flexible High Performance MPI. In: Wyrzykowski R., Dongarra J., Meyer N., Waśniewski J. (eds) Parallel Processing and Applied Mathematics. PPAM 2005. Lecture Notes in Computer Science, vol 3911. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11752578_29
[14] MPICH: https://www.mpich.org/
[15] UCX: P. Shamis et al., "UCX: An Open Source Framework for HPC Network APIs and Beyond," 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects, 2015, pp. 40-43, doi: 10.1109/HOTI.2015.13.
[16] OFED: https://www.openfabrics.org/
[17] 中島研吾，住元真司，塙敏博. Urgent computing に向けたアプリケーション，情報処理学会研究報告（2020-HPC-178-12）（第 178 回 HPC 研究会），2021
[18] Nakajima, K., Parallel Iterative Solvers of GeoFEM with Selective Blocking Preconditioning for Nonlinear Contact Problems on the Earth Simulator. ACM/IEEE Proceedings of SC'03, https://doi.org/10.1145/1048935.1050164, 2003

付録

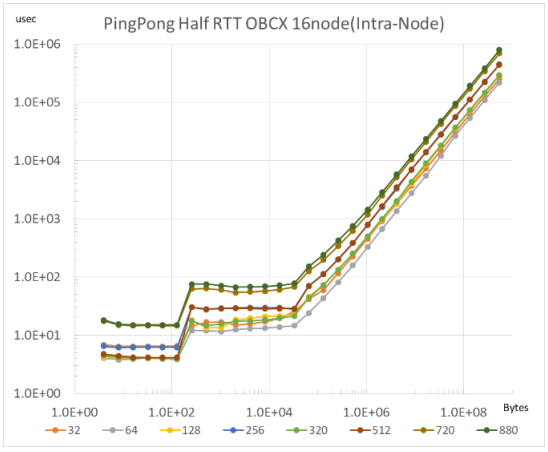

図 31 Oakbridge-CX 1/2 RTT 16node ノード内

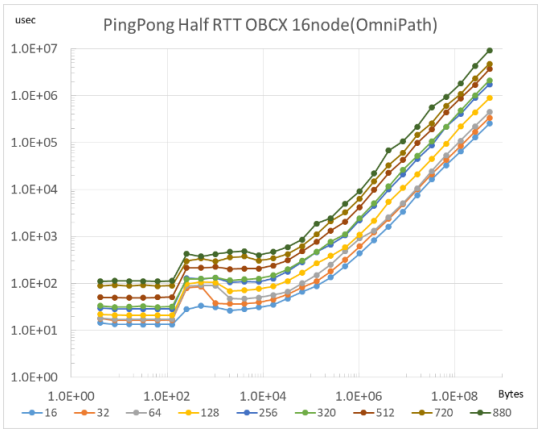

図 32 Oakbridge-CX 1/2 RTT 16node ノード間(OmniPath)

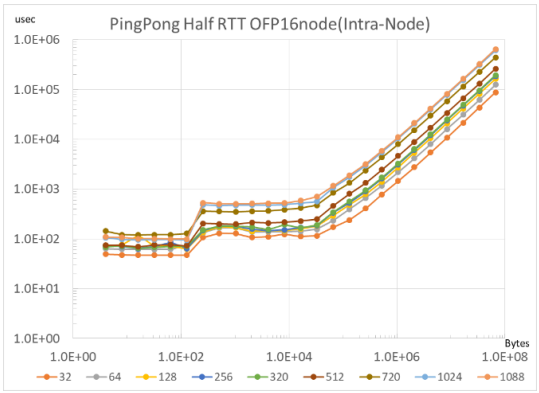

図 33 Oakforest-PACS 1/2 RTT 16node ノード内

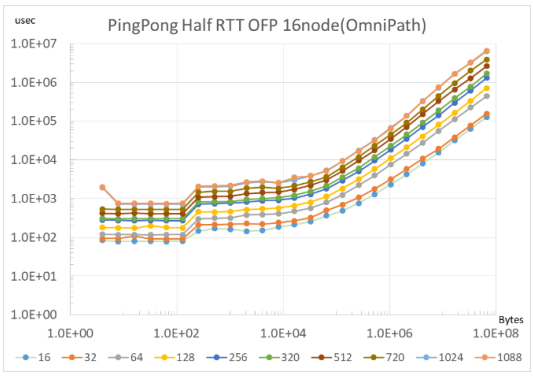

図 34 Oakforest-PACS 1/2 RTT 16node ノード間 (OmniPath)

ⓒ2021 Information Processing Society of Japan

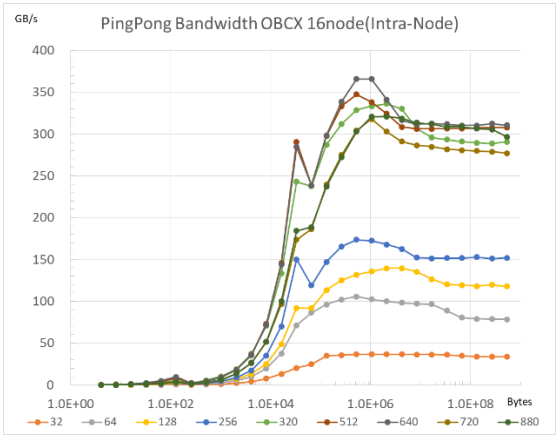

図 35　Oakbridge-CX 通信バンド幅 16node ノード内

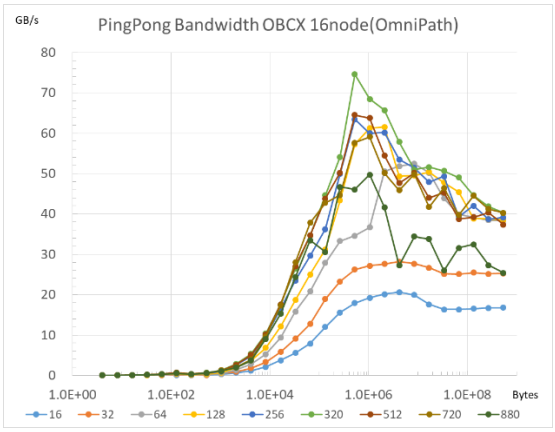

図 36　Oakbridge-CX 通信バンド幅 16node ノード間 (OmniPath)

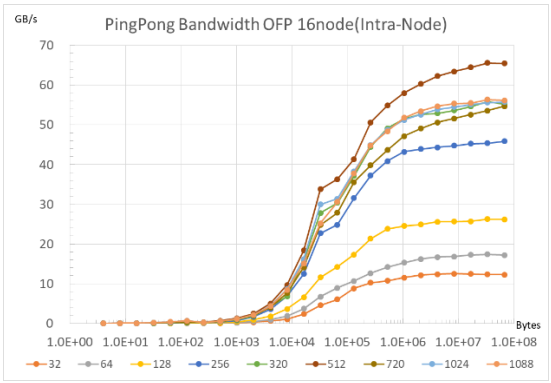

図 37　Oakforest-PACS 通信バンド幅 16node ノード内

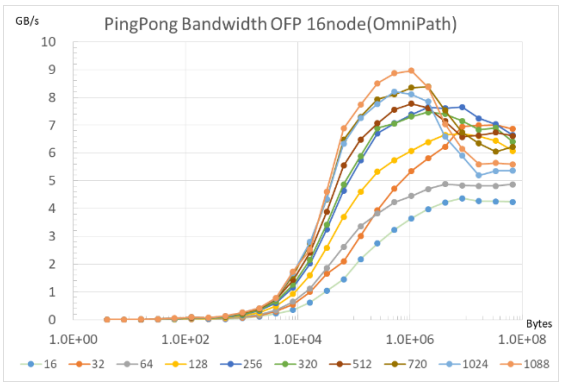

図 38　Oakforest-PACS 通信バンド幅 16node ノード間 (OmniPath)

ⓒ2021 Information Processing Society of Japan