Vol.2013-ARC-205 No.3
Vol.2013-OS-125 No.3
2013/4/25

次世代高性能並列計算機のための システムソフトウェアスタック

石川 裕[1,2]　堀 敦史[2]　Gerofi Balazs[1]　高木 将通[3]　島田 明男[2]　清水 正明[4]　佐伯 裕治[4]
白沢 智輝[5]　中村 豪[6]　住元 真司[7]　小田和 友仁[7]

概要：PC サーバにメニーコア CPU を接続した計算ノードから構成される PC クラスタやメニーコアを計算ノードとしたクラスタ向けのシステムソフトウェアスタックとして，OS カーネル，低レベル通信ライブラリ，MPI 通信ライブラリ，階層化ストレージシステムを含むファイル I/O ライブラリを設計実装している．想定計算機環境および高効率スケーラブルなマシン実現に向けた課題について述べた後，現状の研究開発状況を報告する．

System Software Stack for the Next Generation High-Performance Computers

Yutaka Ishikawa[1,2]　Atsushi Hori[2]　Gerofi Balazs[1]　Masamichi Takagi[3]　Akio Shimada[2]
Masaaki Shimizu[4]　Yuji Saeki[4]　Tomoki Shirasawa[5]　Gou Nakamura[6]　Shinji Sumimoto[7]
Tomohito Otawa[7]

Abstract: A system software stack, consisting of OS kernel, a low-level communication library, an MPI library, and file I/O library with a hierarchical storage system, have been designed and implemented for two types of clusters, one is a PC cluster whose compute node consists of a PC server with manycore CPUs and another is a manycore-based cluster. In this paper, after a machine environment, being considered, and its challenges to provide a highly efficient and scalable system are described, the current research achievements are reported.

1. はじめに

計算機単体の性能対消費電力を向上させるために CPU コア数が増大してきている．Tilela 社 TILE-Gx プロセッサや Intel 社 Xeon Phi Coprocessor など 60 以上の汎用コアが搭載されたプロセッサが登場している．2012 年に製品発表された Xeon Phi Coprocessor では単体で理論浮動小数点演算性能は 1 TFlops を越えている．今後汎用コア数の増大は見込めないだろうとする予測 [1] もあるが，コア数 O(2) のプロセッサが今後の高性能並列計算機の要素技術として使われるだろう．

我々は，エクサフロップス級スーパコンピュータを目指すポストペタフロップススーパーコンピュータの計算ノードアーキテクチャとして汎用メニーコアを想定し，そのためのシステムソフトウェアスタックの設計と実装を進めている．2018 年前後の技術を想定すると，2000m^2 の計算機室と 30 MW 消費電力の制限の中で，数万ノードの汎用メニーコアから構成される 200〜400 PFlops マシンが実現できるであろうと予想している．

現在クラスタシステムのシステムソフトウェアスタックで一般に使われている Linux カーネル，MPI 通信ライブラリ，ファイルシステムが，ポストペタフロップススーパーコンピュータ上に改変することなく移植してもスケーラビ

1 東京大学
2 理化学研究所
3 日本電気
4 日立製作所
5 日立ソリューションズ東日本
6 日立ソリューションズ
7 富士通

表 1　想定ハードウェア諸元と京コンピュータ
Table 1　Specifications of Assumed Hardware Environment and K Computer

	想定	京
計算ノード性能	3〜5 TF	128 GF
コア数	64〜	8
ノード数	80K	82.944K
総メモリ容量	4〜10 PB	1.26 PB
総演算性能	240〜400 PFlops	10.62PF
インターコネクト性能	10〜20GB/sec	5GB/sec
トポロジ	6-D Mesh/Torus, or Dragonfly, or ...	6-D Mesh/Torus
File System 容量	1EB	11PB〜(local) 30PB〜(global)

京コンピュータ諸元：http://www.aics.riken.jp/jp/k/system.html

```
1:  for (day = 1; day < 365*10; day++) {
2:      1 日分の計算
3:      10 変数（1 変数 10 TB データ）を 10 ファイルに書き込み
4:  }
```

図 1　COCO のファイルアクセスパターン
Fig. 1　File Access Pattern in COCO

リティを保った効率良い実行環境が提供できないと予想している．

本稿では，まず，我々が想定するシステム環境について紹介したのち，OS カーネル，通信ライブラリ，ファイルシステムについて，それぞれの課題および現在の検討状況について報告する．なお，本稿で上げている数値目標は今後検討を詳細化していく中で変更される．

2. 想定システム環境

2018 年前後のハードウェア諸元には幅があるが，本稿では表 1 に示す諸元を想定して議論を進める．表に示していない考慮事項について以下述べる．

2.1 メモリ階層

今後のメモリ階層は，L1， L2， L3，主記憶に加えて，OS が直接制御可能なメモリ階層が登場する可能性がある．現状の GPU クラスタにおいても，ホストコンピュータ上の主記憶はアクセラレータ上の主記憶の次の階層として位置づけることが出来る．

2.2 階層化ファイルシステム

ファイルシステム性能は，ファイルを読み書きする計算ノードとファイルシステムの間のネットワーク性能，同時にアクセスされるファイル数，並列ファイルシステムの性能に左右される．全てのノードが単一ファイルシステムにアクセスすると過負荷状態になり性能が著しく低下するため，京コンピュータのように大規模並列計算機では，計算ノード毎あるいは計算ノード群毎にローカルなファイルシステムを持ち，別途グローバルファイルシステムを持つ．プログラムが利用するファイルは実行前にグローバルファイルシステムからローカルファイルシステムにコピーされ（ステージイン），プログラム実行後，生成あるいは修正されたファイルがグローバルファイルシステムにコピーされる（ステージアウト）．想定システム環境においてもファイルステージングを想定するが，単一のグローバルファイルシステムが 1 エクサバイトのストレージ容量を提供するのはコスト的に難しく，さらに階層化したシステムを想定する．

また，本稿では，ファイルシステムに対する要求として東京大学大気海洋研究所で開発されている海洋大循環モデル COCO[2] を取り上げる．COCO は日本沿岸の小規模な湾の流れを太平洋スケールの外洋の海流とリンクさせたシミュレーションシステムであり，シミュレーション結果は 10 変数の値をそれぞれ別のファイルに保存している．問題規模 40,000 x 40,000 x 1,000 では，1 日のシミュレーション結果はそれぞれのファイルに 10 TB のデータ，10 ファイル合計で 100 TB のデータが格納される．ユースケースでは，10 年分のシミュレーションを 1 ヶ月以内に終了させたい．この間に生成されるデータ量は 365PB となる．

3. カーネル

3.1 課題

3.1.1 データの局所性と配置

一般にコアの数が増大するにつれてサーバ系コアに比べてパイプライン段数，キャッシュ容量や TLB サイズおよびタグ連想度（way 数），全体演算性能に対するメモリバンド幅は小さくなる．例えば，Intel 社 Xeon Phi（Knights Corner）の L1 および L2 キャッシュ容量はそれぞれ 32KB, 512 KB あるのに対し，Xeon E5-2670 (Sandy-bridge) の L1, L2, L3 キャッシュ容量はそれぞれ 256KB, 2MB, 20MB ある．メニーコアの能力を最大限引き出すため，キャッシュや TLB のミスヒット率を下げる必要がある．このためには，データアクセスの局所性を上げてワーキングセットを小さくするとともに，それらデータがキャッシュや TLB のタグ連想度以上のアドレス領域をアクセスしないようデータ構造を設計する必要がある．また，NUMA 構成の CPU の場合には，データ配置についても工夫する必要がある．

3.1.2 メモリ階層

想定アーキテクチャでも述べた通り，並列計算機全体でメモリ階層について考える必要がある．ユーザに対して OS はメモリ階層の透過性を維持しながらユーザあるいは実行時環境からのヒント情報を得てデータの最適配置を行う必要がある．

3.1.3 共有資源と排他制御

カーネル内にはコア間で共有される資源が多く，それら

資源の排他制御が必要となる．例えば，ページフォールト時にメモリを割り当てる場合には，ページテーブルおよび物理メモリを管理している構造体，ファイル I/O 処理では，ディレクトリキャッシュやファイルキャッシュなどの構造体，などがある．

Linux における RCU やトランザクショナルメモリ機構の導入などによって，共有資源参照に対するスケーラビリティを向上させるという取り組みが行われている．複数の異なるアプリケーションプロセスが実行されている環境下では，共有資源が同時に参照される頻度が低いため楽観的排他制御は効果がある．しかし，科学技術計算のようなデータ並列型アプリケーションは SPMD (Single Program Multiple Data) 実行モデルが基本である．このため全てのプロセスあるいはスレッドが同時に同じシステムコールを発行する．第 3.1.1 節で述べたデータ構造の局所性とともになるべく共有資源構造を必要としないカーネル設計が必要である．

3.1.4 OS ノイズ

一般に，複数のカーネルスレッドあるいはシステムデーモンによりシステムが管理されている．例えば，ファイル I/O や通信の非同期処理のためにカーネルスレッドを利用したり，定期的に統計情報取得や生存確認のためにデーモンを走らせている．これら OS アクティビティはユーザプロセスから CPU を横取りして実行する．このような OS アクティビティは OS ノイズと呼ばれ，OS ノイズによりユーザプロセスの実行が妨げられる．SPMD 実行モデルで動作している並列プロセスは頻繁に同期通信している．OS ノイズにより同期時に一番遅いプロセスに律速され実行時間が遅くなる．従来から OS ノイズは問題になっているが [3]，OS ノイズを最小限にする OS 構成法が必要である．

3.1.5 プロセス・スレッドモデル

HPC 系アプリケーションでは，ノード間並列は MPI 通信ライブラリをノード内並列は OpenMP 処理系を使う，MPI+OpenMP ハイブリッドプログラミングが主流である．一方で，PGAS モデル [4] のような分散共有メモリ型プログラミングモデルが注目されている．PGAS では，グローバルメモリ領域をスレッド毎に分割し，スレッドはローカルおよびリモートメモリを明示的に参照する．

PGAS モデルを採用している言語処理系のノード内実装では，コアの数だけプロセスを生成し，共有メモリ領域を確保してグローバルメモリ領域を実現している．カーネルは，PGAS モデルなど新しいプログラミングモデルあるいは実行モデルの実装に適したプロセス・スレッドモデルを提供していくべきである．

3.2 検討状況

我々が想定しているポストペタフロップススーパコンピュータに必要とされるシステムソフトウェアスタックの概要を図 2 に示す．現在検討を進めているのが黄色の部分である．

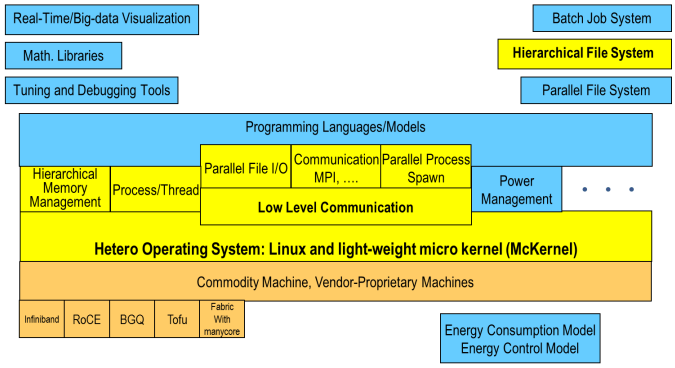

図 2 システムソフトウェアスタック
Fig. 2 System Software Stack

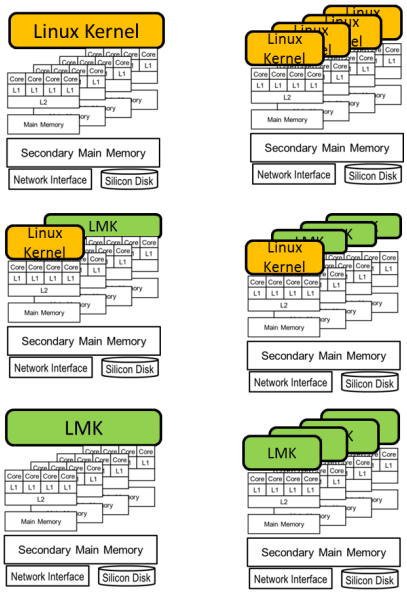

図 3 カーネル構成例
Fig. 3 Examples of Kernel Organizations

3.2.1 カーネル構成法

コモディティ並列計算機環境や一部の商用並列計算機では Linux API を提供している．ポストペタスケールスーパコンピュータにおいてもソフトウエア遺産の継承の観点から Linux API の提供は必須であると考えている．一方で前節で述べたような課題を解決するためにカーネルを再構築する必要がある．カーネルの構成法の可能性を図 3 に示す．大きく次の 3 つの方法が考えられる．

図 3 の最上段は Linux カーネルを元にする方法を示している．計算ノード全体で単一 Linux カーネルを動作させる，計算ノードを論理分割しそれぞれで Linux カーネルを動作させる方法である．単一 Linux カーネルの場合，前節で述べた課題を解決するためのコストだけでなく Linux カーネルの進化に対応しなければいけない．計算ノードを論理分割して Linux カーネルを動作させ，扱う資源量を小さくす

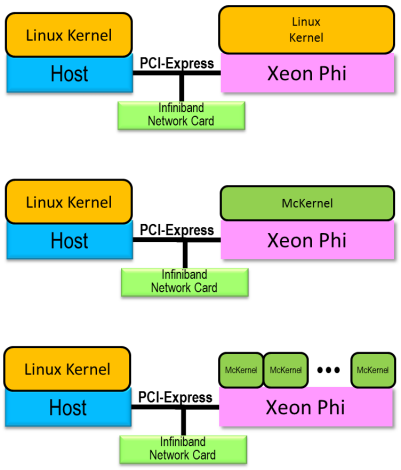

図 4 Xeon Phi 上での実装 (1)
Fig. 4 Implementation on Xeon Phi (1)

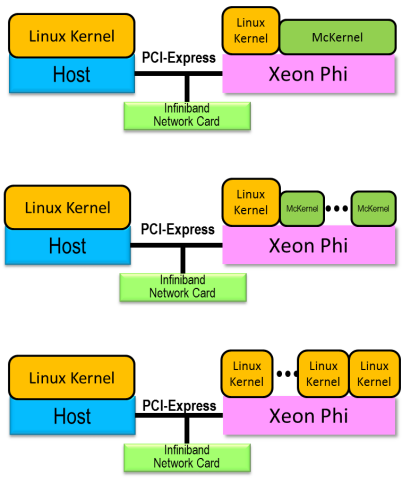

図 5 Xeon Phi 上での実装 (2)
Fig. 5 Implementation on Xeon Phi (2)

ることにより排他制御コストの減少やデータ構造の局所性を上げることが可能である．複数 Linux を動作させた場合には，2 つの利用形態が考えられる．一つは，これら複数 Linux および他の計算ノード上の複数 Linux とともに並列アプリケーションを動作させる利用形態である．計算ノード上では複数 Linux がデバイスを共有する必要がある．もう一つの利用形態は，OS サービスのための Linux カーネルと並列アプリケーションを動作させる目的の Linux カーネルの 2 種類にわける形態である．これにより OS ジッタを減少させることが可能になるだろう．

図 3 の中段は軽量マイクロカーネルと Linux カーネルの異種カーネル混在法を示している．軽量マイクロカーネルはプロセス／スレッド生成，メモリ管理，通信機構などの最低限の OS 機能を提供し，それ以外の Linux API は Linux カーネルに処理を移譲する．軽量マイクロカーネルは Linux の構造に左右されず新しく設計することが可能となる．

図 3 の下段は軽量マイクロカーネルのみの構成となっている．軽量マイクロカーネルで実現されない Linux API は，他の計算機上で動作する Linux サーバに移譲する．

3.2.2 軽量マイクロカーネル設計・実装

前節で述べた構成法の実現可能性を探るために PC サーバ（CPU は Sandy-bridge）に Intel 社 Xeon Phi coprocessor をつなげた計算ノードを Infiniband で接続したクラスタ環境で実装してきている．本環境において実証している OS 構成法を図 4 および図 5 に示す．McKernel は我々が開発している軽量マイクロカーネルである．McKernel は東京大学情報理工学系研究科博士課程に在学中に下沢拓氏によって開発された HIDOS マイクロカーネル [5]，SHIMOS[6]，および MEE 環境 [7] が基になっている．

図 4 の OS 構成法は，概念設計の実証だけでなく実際に本ハードウェア構成で広く利用されることも想定している．なお，図 4 最上段は Intel 社が提供している環境である．

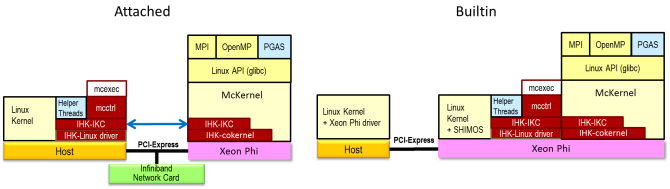

図 6 McKernel
Fig. 6 McKernel

Xeon Phi 上でも Linux カーネルが動作し，PCI Express 上の TCP/IP プロトコルが実装されている．ホストマシンから ssh で Xeon Phi にログインできる．Xeon Phi からホスト上のファイルは NFS 経由でアクセスできる．

現在，入手が容易かつプロセッサアーキテクチャが公開されているメニーコアプロセッサでかつ単独で動作するプロセッサはない．図 5 の OS 構成法は，メニーコアプロセッサが単独で動作した場合の構成を想定している．

現在，図 4 中段および図 5 上段の 2 つの構成を中心に開発を進めている．これら環境における McKernel の概要を図 6 に示す．図の左側および右側の構成をそれぞれ Attached 構成 McKernel，Builtin 構成 McKernel と呼ぶ．図中，赤色部分は Linux カーネルと McKernel とのインターフェイスを実現している IHK (Interface for Heterogeneous Kernel) 層である．

- IHK-Linux driver
  Linux 側のデバイスドライバモジュールで，Xeon Phi の物理メモリをホストメモリにマップする機能，DMA 転送機能，McKernel 機動機能などが実現されている．
- IHK-cokernel
  Xeon Phi 上のモジュールで，Linux カーネルと通信する機能を実現するための最小限の機能を提供するモジュール．
- IHK-IKC

IKC (Inter Kernel Communication) を実現しているモジュールで Linux カーネルおよび McKernel 側の双方に存在する．

- mcctrl
  Linux のユーザプロセスに対して McKernel の起動と通信を可能とするデバイスを提供する．

McKernel 上で Linux システムコールを Linux カーネルに移譲する機能が実現され GNU libc も移植されている．OpenMP で記述されたプログラムも Intel 社コンパイラでコンパイルし McKernel 上で稼働する．McKernel から Linux カーネルへの Linux API 処理移譲方式の改良と性能評価は論文 [8] で述べられている．

3.2.3 メモリ階層

Xeon Phi 上の主記憶の次のメモリ階層として，ホスト上の主記憶を位置づけることが出来る．Attached 構成 McKernel 上で，Xeon Phi 上の主記憶とホスト上の主記憶の間をユーザ透過にページングするシステムを実現している [9]．我々は OpenMP などでスレッド並列化されたプロセス実行環境を想定している．通常の OS カーネルの構成法では，プロセスのスレッドを実行している各コアは同一のページテーブルを参照している．このため，ページングのためにコア群の TLB を無効化しなければならなく，このコストは大きい．論文 [9] では，スレッド毎にページング対象となっているページの中でスレッドが参照しているページテーブルのみを個別に保持することにより，ページング時にそのページをページテーブルに保持しているコアの TLB だけを無効化することによりページングのスケーラビリティ向上を実現している．

3.2.4 PGAS モデル支援

理化学研究所 AICS では，計算ノード内で PGAS モデルに基づく実行モデルを提供する PVAS と呼ばれるプロセス生成機能を設計・実装している [10]．複数のプロセスが同一の仮想アドレス空間を共有する．プロセスが同一の仮想アドレス空間上に存在するため，PGAS モデルに基づく処理系の実行時環境の実装が容易になる．また，同一仮想アドレス空間上のプロセス間のコンテキストスイッチでは TLB をフラッシュする必要がないため高速切り替えが可能となる．

3.2.5 ファイル I/O 処理の並列化

科学技術計算アプリケーションの OpenMP 等による並列化では，計算部分がスレッド並列化され，計算結果をファイルに書き出す処理は一つのスレッドが担当する．NUMA アーキテクチャの場合，スレッドが使用するメモリ領域はスレッドを実行しているコアから近い主記憶に割り当てる (thread affinity)．すなわち，ファイル書き出し処理時には，ファイルに書き出すべきデータは各コアの近くの主記憶に存在していることになる．システムコールが一つのスレッドから呼ばれても，カーネル内で thread affinity に応じたファイル I/O 処理を行うことにより無駄なデータコピーを排除して効率よいファイル I/O 処理を実現できる可能性がある．論文 [11] では，このようなカーネル内で複数のコアが affinity にしたがって I/O 処理することによりどの程度の性能向上が見込めるか事前調査している．

```
01: for (i = 0; i < N; i++)
02:     MPI_Recv_init(rbuf[i], ..., &req[i]);
03: for (I = 0; i < N; i++)
04:     MPI_Send_init(sbuf[i], ..., &req[i+N]);
05: do {
06:     /* Computation */
07:     MPI_Startall(N*2, req);
08:     /* Computation */
09:     MPI_Waitall(N*2, req, stat);
10:     / **** /
11: } while (…);
```

図 7 永続通信使用例
Fig. 7 An Example of Persistent Communication

4. 通信ライブラリ

4.1 課題

4.1.1 計算と通信の重複実行

計算ノードの性能は向上するが計算ノード内の主記憶容量および通信性能は同じスケールでは向上しない．第 2 節で示した想定システム環境では，演算性能は京コンピュータの約 40 倍，主記憶容量は約 8 倍，通信バンド幅は約 4 倍となっている．計算時間が短くなっても通信遅延は同じ割合で小さくならないことを意味する．アプリケーションプログラムの性質に依存するが，計算と通信の重複実行がますます重要となる．

MPI 通信ライブラリのノンブロッキング通信や後述する永続通信 (Persistent Communication) は，計算と通信の重複実行を保証していない．重複実行できるかは実装依存であり，多くの実装ではタイミングによって重複実行される可能性があるだけにすぎない．

4.1.2 通信機能のオフローディング

MPI 通信ライブラリ 3.0 ではノンブロッキング集団通信が規格化された．集団通信の内部処理をユーザプロセスと非同期に進めるためには，別スレッドで通信処理を進める必要がある．メニーコアにおいては通信スレッドの処理をアプリケーションが実行されているコア以外で実行するということも検討する必要がある．送信データは送信の直前にコアが送信データをアクセスし，受信データは受信後コアが受信データをアクセスすることが多い．また，Mellanox 社の Connext-X 以降の通信デバイスには集団通信の実装を支援するオフローディング機能が実装されている．今後，HPC 向けの通信デバイスにこのようなオフローディング機能が装備されていくだろう．このようなデータ局所性および通信デバイス機能を考慮して通信ライブラリを再設計する必要がある．

4.1.3 ネットワークインターフェイスの最適使用

多くの科学技術計算の並列プログラムでは，各プロセスは計算とプロセス間通信を繰り返す．プロセス間通信の相手先と送受信データ領域は固定されていることが多い．MPI 通信ライブラリが提供する永続通信はこのような通信パターンで使える．

図 7 に永続通信の使用例を示す．主ループ内で同じ送信パターン，受信パターンを発行する通信処理に対して，主ループに入る前にそれら要求を初期化し (MPI_Send_init, MPI_Recv_init)，MPI_Request 構造体の配列である req に要求を格納している．複数の要求を発行する MPI_Startall 関数を使い，MPI_Waitall 関数で処理の完了を待つ．ループの中で MPI_Isend/MPI_Irecv 関数を使うよりも簡素に記述できプログラムの見通しが良くなる．

以下のような 2 つの制限下で永続通信機能が使われていれば，複数の送受信要求をうまくスケジューリングして通信ハードウェアを効率よく使用することが可能となる．i) 主ループに入る前に送受信ともに永続通信機構を利用し通信相手が固定されていること．ii) MPI_Startall 時には集団通信と同様コミュニケータに属する全てのプロセスが関与すること．

例えば京コンピュータは通信の DMA エンジンを 4 基搭載している．MPI_Startall 時に送受信すべき通信パターンを調べて 4 基の DMA エンジンの利用率を上げることが出来，結果として通信遅延の減少に貢献する．しかし，現在の MPI 通信ライブラリの実装では通信デバイス層に対して送受信要求を繰り返し呼んでいるため，通信デバイス側での最適実装ができない．

4.1.4 低レベル通信ライブラリ

MPI 通信ライブラリが現在デファクトスタンダードであり，ポータビリティを考慮したアプリケーション，ライブラリ，並列プログラミング言語処理系を開発しようとすると MPI 通信ライブラリを使うしかない．MPI 通信ライブラリよりも低レベルあるいは設計思想の異なる通信ライブラリとしては，PAMI[12]，Portals4[13]，ARMCI[14]，GASNet[15] などがあるが，これらは MPI 通信ライブラリとの共存性あるいはポータビリティの観点から問題がある．

MPI 通信ライブラリ，並列ファイル I/O，並列プログラミング言語処理系などの実装に使用できる標準低レベル通信ライブラリの策定が重要である．これにより，低レベル通信ライブラリの上位層のソフトウェアスタックを効率良くかつポータブルに実装できる．

4.2 検討状況

4.2.1 DCFA および DCFA-MPI

メニーコア上での通信ライブラリを設計実装していくために，第 3 節で述べた Attached 構成 McKernel から直接 Infiniband デバイスをアクセスして通信する低レベル通信層 DCFA を実装した [16]．Xeon Phi は PCI デバイスであるため，Infiniband デバイスを直接初期化することも割り込みを受けることも出来ない．ホスト側で Infiniband デバイスの初期化後，Xeon Phi 側に Infiniband デバイスの PCI Express 上のアドレスを通知する．Xeon Phi 側のメモリ領域をホスト経由で Infiniband デバイスに通知した後，Xeon Phi 側は Infiniband デバイスのアドレスが分かっているので，Infiniband デバイスに直接送受信コマンドを発行することができる．

ホスト CPU である Sandy-bridge と Xeon Phi の組み合わせではメッセージサイズが大きくなると，Xeon Phi から直接 Infiniband にアクセスして送信するよりも Xeon Phi 上のデータをホスト CPU にコピーしホスト CPU から Infiniband 経由で送信した方が高い通信バンド幅が達成できる [17]．論文 [17] では，DCFA 上には MPI 通信ライブラリ実装の一つである YAMPII[18] を移植し，メッセージサイズに応じて Xeon Phi あるいはホストから送信処理を行う機構を実装し評価している．

4.2.2 Persistent Remote DMA Communication

課題で述べた MPI 通信ライブラリの永続通信機構の最適化については Persistent Remote DMA Communication (PRDMA) と称して京コンピュータおよび富士通 FX10 上でプロトタイプ実装および予備評価が終了している [19].

4.2.3 Low Level Communication Library

現在，MPICHI 通信ライブラリの低レベル通信層として Low Level Communication Library (LLC) の API を策定している．LLC は RDMA 通信による one sided 通信を基本として，その上に two sided 通信機能を構築する．LLC は，Persistent Remote DMA Communication を容易に実現出きるような API も提供していく予定である．

5. ファイルシステム

5.1 課題

アプリケーションのファイル I/O の性質を考えず COCO が生成する 365 PB のデータをグローバルファイルシステムに直接格納することを考えると，グローバルファイルシステムの総データ転送スループットが 1TB/sec で約 4 日かかり 10TB/sec で 10 時間となる．これは 365PB のデータがローカルファイルシステムに蓄えられた後にグローバルファイルシステムにステージアウトする場合の性能である．しかし，ローカルファイルシステムに 365 PB の容量を持たせるのはコストが高く，非同期処理かつパイプライン処理によって，計算ノードからローカルファイルシステム，そしてグローバルファイルシステムにデータが流れていくべきである．

5.2 検討状況

ローカルファイルシステムに収まるだけの計算に分割し，

ジョブの実行とステージアウトを重複実行することを検討する．すなわち，一回のジョブ実行時に生成されたファイルをローカルファイルシステムからグローバルファイルシステムにコピーしている間に次の計算のためのジョブを同時に実行する．この場合，ローカルファイルシステムには2回分のファイルが格納される必要があるので，1つのジョブが利用できるファイルサイズはローカルファイルシステム全体の1/2の容量となる．ローカルファイルシステムの総量を10PBとするとジョブが利用できるのは5PBのファイルとなる．

COCOの場合，50日分のシミュレーションで5 PBのファイルが生成される．1ヶ月以内に10年分3650日のシミュレーションを終了させるためには，1日分のシミュレーションを12分以内に終了させる必要がある．50日分のシミュレーション1回の計算を10時間で終了させれば良いことになる．なお，1日分のシミュレーション結果であるデータをローカルファイルシステムに書き込みスループットが2TB/secあれば1分以内に書き込みを終了することができるので，シミュレーション時間そのものは11分以内に終了すればよい．

10時間以内に5 PBのデータをグローバルファイルシステムに書き込めば良いとするとローカルファイルシステムとグローバルファイルシステムの間のデータ転送スループットは139 GB/secあれば良いことになる．他のアプリケーションにおけるファイルI/O要求を精査しないと結論は出せないが，現在想定しているCOCOの問題サイズと要求実行完了時間ではファイルI/O性能を特別に考慮する必要はない．

しかし，1 EBのストレージをハードディスク装置で賄うにはコストが高くなるためテープ装置や光ディスクのようなさらなる階層化を考える必要がある．また，上記見積りでは，ユーザがジョブを複数に分けて実行することを想定したが，これでは効率が悪い．システムソフトウェアで，透過的に非同期かつパイプライン的にデータを移動する機能を今後検討する必要がある．また，他のアプリケーションのユースケースを調べ，ローカルファイルシステムおよびグローバルファイルシステムの適正な大きさやスループットを決めていく必要がある．また，今後，データの圧縮効果，ファイル間の重複ブロックデータ排除などにより実効データサイズの縮小を検討していく．

6. おわりに

本稿では，PCサーバにメニーコアCPUを接続した計算ノードから構成されるPCクラスタやメニーコアを計算ノードとしたポストペタスケール並列計算機環境のためのシステムソフトウェアスタックとして，OSカーネル，低レベル通信ライブラリ，MPI通信ライブラリ，階層化ストレージについての検討状況を報告した．現在，概念設計を進めるとともにシステムソフトウェアの実装も進めている．McKernelはOSカーネルの実現可能性を確認するためのプロトタイプシステムに留まらず実用化を目指している．Attached構成のMcKernel上で理研AICSで開発されているSCALE気象コードが稼働している．SCALEはFortranで記述されノード内OpenMP並列，ノード間MPI並列のハイブリッド並列化されたコードである．2012年SC国際会議の研究展示では，可視化ツールと連携しオンラインでシミュレーション結果を表示するデモを行った．

今後，ゲノム情報処理を含む他のアプリケーションのユースケースを検討しながらOS構成，通信機能，ファイルI/O処理を含む階層化ストレージを検討していく．

謝辞

本研究の一部は，文部科学省「将来のHPCIシステムのあり方の調査研究」のなかの課題名「レイテンシコアの高度化・高効率化による将来のHPCIシステムに関する調査研究」，科学技術振興機構CRESTポストペタ領域のなかの課題名「メニーコア混在型並列計算機用基盤ソフトウェア」，および国際科学技術共同研究推進事業研究領域「情報通信技術」研究課題名「ポストペタスケールコンピューティングのためのフレームワークとプログラミング」による．

参考文献

[1] Moore, C.: DATA PROCESSING IN EXASCALE-CLASS COMPUTER SYSTEMS, *The Salishan Conference on High Speed Computing* (2011).
[2] Hasumi, H.: Ocean Component Model (COCO) Version 2.1, Technical report, Division od Climate System Research, Atmosphere and Ocean Research Institute, the University of Tokyo (2000).
[3] Petrini, F., Kerbyson, D. J. and Pakin, S.: The Case of the Missing Supercomputer Performance:Achieving Optimal Performance on the 8,192 Processors of ASCI Q, *SC '03 Proceedings of the 2003 ACM/IEEE conference on Supercomputing* (2003).
[4] Carlson, W., Draper, J., Culler, D., Yelick, K., Brooks, E. and Warren, K.: Introduction to UPC and Language Specification, Technical Report CCS-TR-99-157, IDA Center for Computing Sciences (1999).
[5] Shimosawa, T.: Operating System Organization for Manycore Systems, Technical report, A Doctor Thesis submitted to the Graduate School of the University of Tokyo (2012).
[6] Shimosawa, T., Matsuba, H. and Ishikawa, Y.: Logical Partitioning without Architectural Supports, *The 32nd IEEE International Computer Software and Applications Conference (COMPSAC 2008)* (2008).
[7] 下沢　拓，石川　裕，堀　敦史，並木美太郎，辻田祐一：メニーコア向けシステムソフトウェア開発のための実行環境の設計と実装，情報処理学会SWOPP (2011).
[8] 佐伯裕治，清水正明，白沢智輝，中村　豪，高木将通，Gerofi, B.，思　敏，石川　裕，堀　敦史：ヘテロジニアス計算機上のOS機能委譲機構，情報処理学会 第124回システムソフトウェアとオペレーティング・システム研

究会 (2013).
[9] Gerofi, B., Shimada, A., Hori, A. and Ishikawa, Y.: Operating System Assisted Hierarchical Memory Management for Heterogeneous Architectures: Preliminary Results on Stencil Computation, *The 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2013)* (2013 (To appear)).
[10] Shimada, A., Gerofi, B., Hori, A. and Ishikawa, Y.: PGAS Intra-node Communication towards Many-Core Architecture, *The 6th Conference on Partitioned Global Address Space Programming Models* (2012).
[11] 小田和友仁，住元真司，　堀敦史，　石川裕：メニーコア向け NUMA 最適並列分散 I/O の予備検証，情報処理学会 第 124 回システムソフトウェアとオペレーティング・システム研究会 (2013).
[12] Kumar, S., Mamidala, A. R., Faraj, D., Smith, B. E., Blocksome, M., Cernohous, B., Miller, D., Parker, J., Ratterman, J., Heidelberger, P., Chen, D. and Steinmacher-Burrow, B. D.: PAMI: A Parallel Active Message Interface for the Blue Gene/Q Supercomputer, *IEEE 26th International Parallel and Distributed ProcessinSymposium*, pp. 763–773 (2012).
[13] Barrett, B. W., Brightwell, R., Hemmert, K. S., Pedretti, K., Wheeler, K. and Underwood, K. D.: Implementing OpenSHMEM and its Implications for Portals 4, *19th Annual Symposium on High-Performance Interconnects (HotI)* (2011).
[14] Nieplocha, J., Tipparaju, V., Krishnan, M. and Panda, D.: High Performance Remote Memory Access Comunications: The ARMCI Approach, *International Journal of High Performance Computing and Applications*, No. 2, pp. 233–253 (2006).
[15] Bonachea, D.: GASNet Specification, v1.8, Technical report, U.C. Berkeley Tech Report (UCB/CSD-02-1207) (2008).
[16] Si, M. and Ishikawa, Y.: Design of Direct Communication Facility for Manycore-based Accelerators, *CASS2012 in conjunction with IPDPS2012* (2012).
[17] Si, M., Ishikawa, Y. and Takagi, M.: Direct MPI Library for Intel Xeon Phi co-processors, *CASS2013 in conjunction with IPDPS2013* (2013).
[18] 石川　裕：YAMPII もう一つの MPI 実装，情報処理学会研究報告，pp. 115–120 (2004).
[19] Ishikawa, Y., Nakajima, K. and Hori, A.: Revisiting Persistent Communication in MPI, *EuroMPI 2012: Recent Advances in the Message Passing Interface*, Springer Netherlands, pp. 296–297 (2012 (poster)).