戦略的創造研究推進事業　CREST
研究領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」
研究課題「メニーコア混在型並列計算機用基盤ソフトウェア」

研究終了報告書

研究期間　平成23年4月～平成28年3月

研究代表者：堀　敦史
（国立研究開発法人　理化学研究所
計算科学研究機構　システムソフトウェア研究チーム、上級研究員）

§1 研究実施の概要

(1)実施概要

本研究プロジェクトには，メニーコアとマルチコアが混在するようなノードアーキテクチャにおける，a) メニーコア用 OS カーネル，b) 高スケーラブルな通信とI/O，c) 超軽量スレッドライブラリ，d) 故障レジリエンス，の 4 つの研究テーマがある．

a) メニーコア用 OS カーネル

本プロジェクトでは，メニーコアとマルチコアが混在するヘテロなアーキテクチャを想定し，軽量な OS カーネルを新たに開発することが目標の 1 つとしていた．このための軽量 OS カーネル開発を進めていたが，諸般の事情により，OS カーネル（McKernel）開発の主体は，本プロジェクトから別なプロジェクトに移行したため成果とはならなかったが，同時に研究開発を進めていたメニーコア向けの新しいタスクモデル PVAS（Partitioned Virtual Address Space）に注力することとなった．

PVAS では，複数のプロセスが 1 つの仮想アドレス空間を共有する．PVAS は既存の Linux カーネルを改変することで開発された．PVAS の有効性を検証のために PVAS を用いて MPI を実装した結果，様々な MPI 処理方式の改良が可能であり，その結果として高いノード内通信性能を省メモリで実現できることが確認できた．PVAS のアイデアは，メニーコアとマルチコアが混在するアーキテクチャ向けに拡張した M-PVAS（Multiple PVAS）の研究開発に拡張された． PVAS のアイデアは OS カーネルだけに留まらず，b) 高スケーラブルな通信と I/O，および，c) 超軽量スレッドライブラリにおける大きな柱となった．また，最終年度までに，PVAS を McKernel に移植が完了する予定である．

以上の結果より，OS カーネルの開発は発展的に解消されたものの，PVAS という全く新しいタスクモデルを提供する OS の開発と，その検証により有効性が示されたと考えられる．

b) 高スケーラブルな通信とI/O

本プロジェクトが想定するヘテロなアーキテクチャにおけるスケーラブルな通信や I/O 機構の研究開発を目的としていた．ここでメニーコアとマルチコア間の通信や同期の効率的な機構の実現が大きなチャレンジとなる．

これに対し，PVAS をヘテロアーキテクチャに拡張した Multiple PVAS（M-PVAS）を開発した．これは，メニーコア，マルチコアの双方で PVAS 対応 Linux が載っている環境において，メニーコア，マルチコア双方のプロセスが 1 つの仮想アドレス空間を共有する．これにより，メニーコア上のプロセスとマルチコア上のプロセス間での同期や通信が効率よく実現可能となる．また，上記 a)における PVAS による MPI 通信の高速化および省メモリ化に加え，M-PVAS を使ったヘテロアーキテクチャ上での MPI の新しい実装を試みた．メニーコアでアプリの並列計算をおこない，ノード間通信はマルチコアでおこなった結果，メニーコア単体より高い通信性能を実現することができた．さらに M-PVAS の有効性の検証のために Pthread 実装の MapReduce プログラムを，M-PVAS を用いた実装に書き換え，現在検証を進めている．一方，MPI-IO レベルでの I/O の研究も行われた．プロジェクト前半で得られた知見から，ファイルサーバに対する I/O 要求を制限する I/O throttling 方式が有効であることが確認されたことを受けて，これを MPI-IO に応用した．I/O throttling 方式と段階的な MPI-IO 内部通信を組み合わせた“EARTH”を開発した．EARTH を用いることで，約 2 倍の MPI-IO 性能の向上を確認した．

上記の結果，PVAS および M-PVAS により MPI 通信の高速化や省メモリ化が実現でき，エクサにおける高スケーラブルな通信のための要素技術を開発することができたと考える．また，MPI-IO の高速化によりエクサでの I/O 処理高速化技術の開発にも成功したと考える．

c) 超軽量スレッドライブラリ

本テーマは，特にメニーコアアーキテクチャにおいて，多くのコアを有効に活用すべく，従来の Pthread に代わるようなノード内並列実行モデルの研究開発するものである．

このために，PVAS をベースに，ULP（User-Level Process）を提案している．ULP は MPI の実行モデルと相性も良く，特に負荷の不均衡が生じやすい不規則なアプリの効率的な実行と通信時間の隠蔽をともに実現しようとするものである．既に ULP の実装と基礎的な評価をおこなった．

この ULP の研究は，米 ANL との共同研究であり，日米共同研究（DOE-MEXT）の枠組みでおこなわれており，本プロジェクト終了後も ANL と共同で ULP を用いた MPI の研究開発を継続する予定である．

d) 故障レジリエンス

エクサ時代において故障レジリエンスは必須の技術であると予想されている．計画では MPI の規格に入ると思われていた ULFM（User-Level Fault Mitigation）の開発を進めると同時に京コンピュータに移植することであった．またプロジェクト後期には，新たにスペアノードの研究を開始することとなった．

ULFM の開発は米 Tennessee 大学で進められ，MPI プログラム実行中にノード故障が発生してもアプリの実行を継続可能とするものであるが，集団通信の実行中に生じた故障の対応が難しく，当初は O(P2)の処理だったものを，log オーダーにまで低減することができた．ULFM の京コンピュータへの移植は，理研が保有する京の商用版である FX10 に移植し，基本動作の確認が終了している．スペアノードの研究では，ULFM 等でユーザアプリがノード故障に耐えるとしたときに，スペアノードによる代替手法を提案し，京コンピュータ，BG/Q，TSUBAME2.5 上で評価をおこなった．

現時点で ULFM は MPI の規格として採用されるまでに至っていないが，上記の成果により，当初の計画以上の成果を残すことができたと考える．

これらの研究成果のうち，PVAS，M-PVAS に関してはオープンソースソフトウェアとして公開されている
（URL: http://www-sys-aics.riken.jp/releasedsoftware/software/pvas-1.0.0-beta.tar.gz）．

（2）顕著な成果

＜優れた基礎研究としての成果＞

1．メニーコア時代における新しいタスクモデル（PVAS）
概要：（200 字程度）
メニーコアにおいて，ノード内の並列度が高くなり，従来の並列実行モデル（プロセス並列，スレッド並列）では効率の低下が懸念される．提案する新しいタスクモデル（PVAS）では，複数のプロセスが同じ仮想アドレス空間に位置し，プロセス間通信や同期が容易になるだけでなく，スレッド並列で問題となる排他制御による性能低下を抑えることが可能になる．

2．異機種アーキテクチャにおける新しいタスクモデル（M-PVAS）
概要：（200 字程度）
加速器を持つ異機種アーキテクチャにおいて，ホスト CPU と加速器との間でのデータ転送や同期のオーバヘッドが大きな問題となることが多い．本プロジェクトにおいて提案する M-PVAS では，このような構成において同じ仮想アドレス空間を共有することで，この問題を根本的に解決する手段を提供する．

3．エクサ時代における故障ノードのスペアノードによる代替手法
概要：（200 字程度）
エクサ級のスーパーコンピュータでは，ノード数に比例する故障率により故障発生が状態化すると考えられている．これまでの研究では，故障が発生した時に，どのようにジョブの実行を継続させるかという点に注力されてきたが，本研究によりある種のアプリケーションではスペア

ノードによる故障ノードの代替が有効であることと，スペアノードによる代替時に通信速度が低下する可能性を示した．この研究はユニークであり，今後の展開が注目される．

＜科学技術イノベーションに大きく寄与する成果＞

1．メニーコア時代における新しいタスクモデル（PVAS）
概要：（200 字程度）
PVAS の研究は，技術の成熟度という観点からはまだまだ未熟であるが，その可能性は大きく，将来メニーコア時代における標準的な実行モデルとなる可能性がある．その意味で，今後も研究を進め，その優位性をアピールする必要があると考えられる．

2．MPI-IO の改良（EARTH）
概要：（200 字程度）
MPI-IO は MPI 標準に含まれるファイル I/O のための API であり，本研究において MPI-IO 実装に改良を施し，顕著な性能向上を確認している．現在，主な MPI-IO は ROMIO と呼ばれる実装がベースとなっており，我々の改良も ROMIO をベースにおこなっている．現在，EARTH の機能を ROMIO に包含させるべく，ROMIO の開発者と協議をおこなっている．また，EARTH が成熟すれば，EARTH の全ての機能を ROMIO に組み込むことも予定されている．それが実現されたならば，MPI 実装の大半に本研究の成果が反映されることになる．

3．レジリエンス基盤
概要：（200 字程度）
本プロジェクトの共同研究者である Dongarra グループ（テネシー大学）は，かねてより User-Level Fault Mitigation（ULFM）と呼ばれる，ユーザが故障に対応するプログラムを記述可能にする MPI の研究開発をおこなっている．この ULFM の研究は，次期 MPI の耐故障性に関する規格に組み込むべく，MPI の規格を決めている MPI Forum に対し働きかけをおこなっている．従い，将来の MPI にこの ULFM が組み込まれる可能性があり，その時には，多くの MPI において ULFM の機能が使えるようになり，その影響は大きいと考えられる．

§２ 研究実施体制

（1）研究チームの体制について

①「堀」グループ
研究参加者

氏名	所属	役職	参加時期
堀　敦史	理研 AICS	上級研究員	H23.4～
山本　啓二	同上	研究員	H23.4～H26.3
大野　善之	同上	リサーチアソシエイト	H23.4～H25.3
今田　俊寛	同上	同上	H23.4～H24.4
下澤　拓	東京大学理学部	D3	H23.4～H24.3
島田　明男	理研 AICS	リサーチアソシエイト	H24.4～H27.3
亀山　豊久	同上	テクニカルスタッフ	H23.4～
畑中　正行	同上	リサーチアソシエイト	H24.6～H24.8
Gerofi, Balazs	同上	研究員	H24.4～H25.3
辻田　祐一	同上	研究員	H25.4～
吉永　一美	同上	特別研究員	H25.4～

研究項目
- メニーコア OS カーネル
- 高スケーラブルな通信と I/O の実現
- 超軽量スレッドライブラリ
- 故障レジリエンス基盤

②「須藤」グループ（平成 27 年 4 月より）
研究参加者

氏名	所属	役職	参加時期
須藤　敦之	日立製作所	主任研究員	H27.4～
島田　明男	同上	研究員	H27.4～

研究項目
- メニーコア用 OS カーネル
- 超軽量スレッドライブラリ

③「並木」グループ
研究参加者

氏名	所属	役職	参加時期
並木　美太郎	東京農工大学大学院工学研究院	教授	H23.4～
佐藤　未来子	同上	特任助教	H23.4～
坂本　龍一	同上		H23.4～H27.3
小林 弘明	同上		H23.4～H25.3
高橋　昭宏	同上		H23.4～H25.3
長嶺　精彦	同上		H23.4～H25.3
深沢　豪	同上		H23.4～H26.3
田口　雄大	同上		H24.4～H24.5
嶋田　裕巳	同上		H24.4～H26.3
鈴木　健一	同上		H25.4～H27.3
塚本　潤	同上		H25.4～H27.3
和田　基	同上		H25.4～H27.3
小柴 篤史	同上	M1～2	H26.4～
郭 林瞻	同上	M1～2	H27.3～
中原 健志	同上	B4, M1	H27.3～
菊池 和美	同上	研究補助員	H23.4～

研究項目
- メニーコア用 OS カーネル
- 高スケーラブルな通信と I/O の実現

④「辻田」グループ（平成 25 年 3 月まで. 平成 25 年 4 月より堀グループに併合された）
研究参加者

氏名	所属	役職	参加時期
辻田　祐一	近畿大学工学部	准教授	H23.4～H25.3
吉永　一美	同上	博士研究員	H23.8～H25.3
六車　英峰	同上	M2	H23.4～H24.3

研究項目
・ 高スケーラブルな通信とI/O の実現

⑤「Dongarra」グループ(平成 24 年 10 月より開始)
研究参加者

氏名	所属	役職	参加時期
Jack Dongarra	テネシー大学 ICL	University Distinguished Professor	H24.10～
George Bosilca	同上	Research Assc. Prof.	H24.10～
Aurelien Bouteiller	同上	Research Scientist	H24.10～
Thomas Herault	同上	Research Scientist	H24.10～H27.3
Amina Guermouche	同上	Senior Research Assoc.	H26.4～H27.3

研究項目
・ 故障レジリエンス基盤

(2)国内外の研究者や産業界等との連携によるネットワーク形成の状況について

- 米アルゴンヌ研究所
  DOE-MEXT共同研究に関する覚書の枠組みにおいて,PVAS-ULPを用いたMPIの実装に関する研究をおこなっている.
- 米テネシー大学
  PVASを用いたOpenMPIの集団通信の実装に関する共同研究を開始した. 本研究は本CRESTプロジェクトにおける共同研究とは別である.
- 筑波大学
  佐藤（CREST統括）教授とPVASを用いたXcalableMPの実装に関する共同研究をおこなっている.
- 慶應大学
  河野（CRESTアドバイザリ）教授とPVASをベースにHPC向けOSの共同研究をおこなうこととなった.
- 独 Jülich Supercomputing Center（JSC)
  故障ノードのスペアノードによる代替の研究で,JSC所有のBG/Qの使用を許可してもらっている.

§3 研究実施内容及び成果

第2章でも述べた通り,中間評価の結果を受けて,研究テーマの研究グループへの分担を見直した結果,より一体的な研究を推進するために,研究テーマを研究グループで重複して担当することとした.この結果,本章で述べる各研究グループの成果と,研究テーマの関連が分かり辛くなっている.下表は,研究テーマと研究グループ,および研究成果の対応を表している.

研究テーマ	理研:堀G (近大:辻田G)	日立:須藤G	農工大:並木G	テネシー大: Dongarra G
メニーコア用	- PVAS	- PVAS	- M-PVAS	(なし)

OSカーネル	- M-PVAS - PVAS on McKernel	- PVAS on McKernel		
高スケーラブルな通信とI/Oの実現	- MPI (PVAS) - MPI-IO (EARTH)	- MPI (PVAS)	- MPI (M-PVAS) - MapReduce (M-PVAS)	(なし)
超軽量スレッドライブラリ	- ULP	- ULP	(なし)	(なし)
故障レジリエンス	- ULFM on 京 - Sliding Substitution	(なし)	(なし)	- ULFM - ULFM on 京

本プロジェクトで開発された主要な要素技術と，その評価結果を下表にまとめる．

要素技術	評価	成果
PVAS	MPI	- NPB で最大 28%高速 - fft2d_datatype（派生データ型）で最大 21%高速 - AlltoAll, 240 procs. で 256MB 省メモリ
	ULP	- MassiveThreads と同等のコンテキスト切替時間
M-PVAS	CPU 間呼出	- Intel 環境に比べ 100 倍近い高速化
	MPI	- マルチコア上での MPI 処理により最大約 2 倍のバンド幅
	MapReduce	- 評価中
ULFM	(改良)	- 新アルゴリズム $O(P^2) \rightarrow O(\Sigma f = 1..F(\log(P-f)))$
	(FX10 への移植)	- FX10 に移植した ULFM の動作確認
Sliding Substitution	(京, BG/Q, TSUBAME2.5)	- 提案手法の評価（比較できる他の研究が存在せず）

3．1 メニーコア用 OS カーネルと超軽量スレッドライブラリ（理研 堀グループ）

(1)研究実施内容及び成果

① メニーコア OS

エクサスケールのシステムでの採用が検討されているメニーコアアーキテクチャは，性能は低いが電力効率は高いコアを多数集約することにより，高い並列処理性能を実現するアーキテクチャとなっている．よって，システムソフトウェアもメニーコアアーキテクチャに特化したものが求められる．本プロジェクトでは，最初の 2 年でメニーコアアーキテクチャの性質解明および基礎データの収集につとめ，中間年度から，メニーコアアーキテクチャに適した新たな OS タスクモデルである Partitioned Virtual Address Space (PVAS) の研究開発を開始した．H26 年度から PVAS の有用性を検証するための応用研究を開始し，H27 年度からは，日立製作所の須藤グループに研究主体を移行し，引き続き PVAS の検証を進めている．以下に，研究内容を述べる．

メニーコアアーキテクチャを採用するシステムでは，ノード 1 台あたりのコア数が従来よりも増加する．コア数の増加に伴い，ノード上で動作する並列プロセスの数も増加することになる．並列プロセス間の通信には，異なるノード上で動作するプロセス間の通信であるノード間通信と同一ノード上で動作するプロセス同士の通信であるノード内通信に分けられる．メニーコアを採用するシステムでは，1 ノード上で従来よりも多くの並列プロセスが動作するため，ノード内通信の発生頻度が従来のマルチコアを採用するシステムよりも増加する．よって，より高速なノード内通信が求められる．また，メニーコアアーキテクチャでは，コアあたりのメモリ量が従来のマルチコアよりも減少する傾向があるため，ノード内通信に必要なメモリ量を低減することが求められる．

メニーコアアーキテクチャにおけるノード内通信の遅延の低減とメモリ使用効率の向上を目的として，OS タスクモデル PVAS の開発を行った．PVAS は，複数のプロセスを一つのアドレス空間

で動作させるため，プロセス間のデータ送受信時に伴うコストを，従来のタスクモデルよりも低減できる．

従来のタスクモデルでは，ノード上の並列プロセスが異なるアドレス空間で動作するため，ノード内通信の際に，アドレス空間の境界越しにデータを送受信するためのコストが発生する．このコストは通信遅延の増加やメモリ消費量の増加といった形で顕在化する．代表的なノード内通信の方式として共有メモリ方式が挙げられる．この方式では，通信を行う双方のプロセスがアクセス可能な共有メモリ上の中間バッファを経由して，アドレス空間の境界越しにデータを送受信する．共有メモリを経由して送信バッファから受信バッファにデータをコピーして通信するため，メモリコピーの回数が増加し，通信遅延が増加する．また，共有メモリを各並列プロセスのアドレス空間にマッピングする必要があるため，ページテーブル（各プロセスのメモリのマッピング情報を管理する OS 内の管理テーブル）のサイズが増加し，メモリ消費量が大きくなる．

もう一つの代表的なノード内通信の方法として，OS カーネルによるデータコピーが挙げられる．この方式では，データのコピーを OS カーネルに委譲することで，アドレス空間の境界越しにデータを送受信する．OS カーネルは，ノード内の全プロセスのメモリにアクセスすることができるので，送信プロセスの送信バッファから受信プロセスの受信バッファにデータを直接コピーすることができる．よって，共有メモリによる方式よりもメモリコピーの回数が減少し，通信遅延が小さくなる．しかし，OS カーネルにデータのコピーを依頼するためのシステムコールを呼び出す必要があり，このオーバヘッドにより，送受信するデータサイズが小さい場合は共有メモリによる方式の方が，通信遅延が小さくなる．

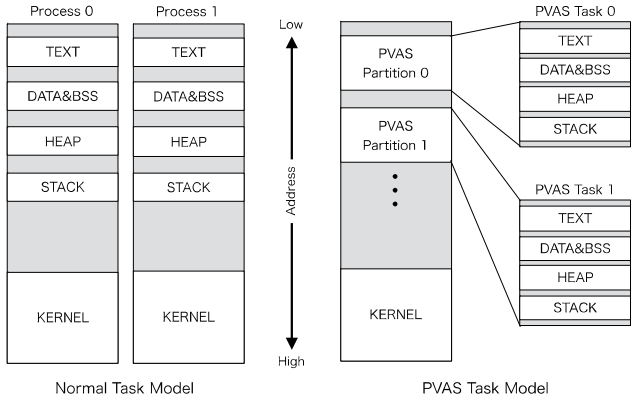


図 1　PVAS アドレス空間のレイアウト

図 1 は既存のタスクモデルと PVAS のアドレス空間レイアウトを示している．既存のタスクモデルでは並列プロセスがそれぞれ個別のアドレス空間で動作するのに対し，PVAS では一つのアドレス空間を固定サイズのセグメント（PVAS パーティション）に分割し，各並列プロセス（PVAS タスク）に割り当てる．並列プロセスが同一アドレス空間で動作するため，並列プロセスは，load/store 命令で互いのメモリにアクセスすることができる．送信プロセスの送信バッファから受信プロセスの受信バッファにデータを直接コピーすることが可能になるだけでなく，システムコールを通信時に呼び出す必要もないので，PVAS を用いるノード内通信は，これまでのノード内通信よりも通信遅延が小さくなることが期待できる．また，共有メモリをマッピングする必要もないため，ページテーブルの肥大化を回避し，カーネル内部でのメモリ消費を低減することができる．

PVAS の有用性を検証するため，PVAS を MPI のノード内通信に適用し，評価した．MPI は並列アプリケーションを実装するための通信規格であり，HPC 分野のアプリケーション開発において広く用いられている．評価をおこなうため，代表的な MPI ライブラリの一つである Open MPI のノード内通信に PVAS を適用した．図 2 は PVAS を適用した Open MPI のノード内通信を示している．並列処理を実行する MPI プロセスを PVAS タスクとして起動し，同一アドレス空間で動作させる．送信プロセスは送信バッファのアドレスを受信プロセスに通知し，受信プロセスは送信プ

ロセスの送信バッファから自身の受信バッファにデータをコピーする.

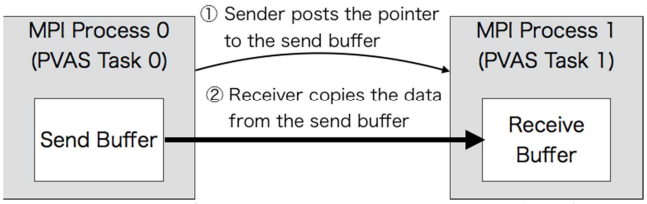


図 2　PVAS を適用した Open MPI のノード内通信

評価は，通信遅延を評価するベンチマーク，計算処理を含むアプリケーションの実行性能を評価するベンチマークを用い，既存の通信方式との比較を行った．また，通信遅延の評価において，メモリ消費量の評価も行った．

図 3 は，Intel MPI Benchmark （IMB）の ping-pong 通信を用いて，Open MPI のノード内通信の遅延を測定した結果を示している．測定はメニーコア CPU である Intel Xeon Phi を用いておこなった．”SM”は共有メモリを用いる方式の通信遅延を，”SM（KNEM）”は OS カーネルによるデータコピーを用いる方式の通信遅延を示している．”PVAS”は PVAS のノード内通信を適用した方式の通信遅延を示す．図中のグラフに示す通り，PVAS を適用した方式は全てのメッセージサイズにおいて既存の方式よりも通信遅延が小さくなった．

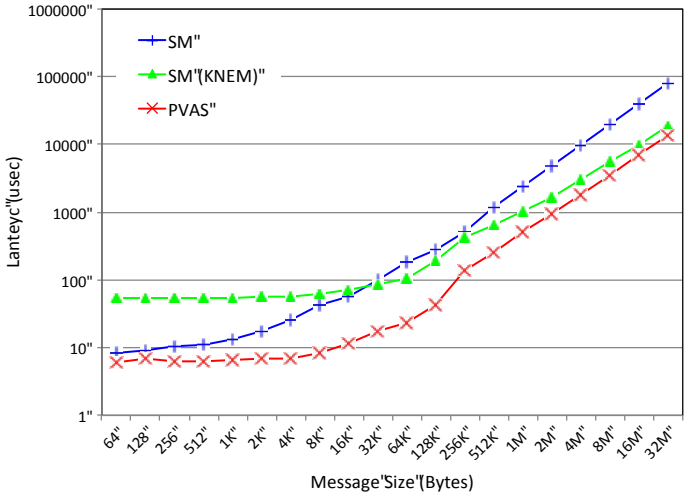


図 3　通信遅延（ping-pong 通信）

図 4 は NAS Parallel Benchmarks（NPB）の各種ベンチマークの，共有メモリを用いた方式を 1 とした時の相対実行性能を示している．NPB を用いると，通信だけではなく，計算処理を含めた MPI アプリケーションの総合的な実行性能を評価することができる．横軸にはベンチマークを，縦軸には共有メモリを用いる方式を基準とした性能の改善率（パーセント）を示した．図に示すとおり，PVAS を適用した方式では，共有メモリを用いる方式に比べ最大で約 28%実行性能を向上することができた．

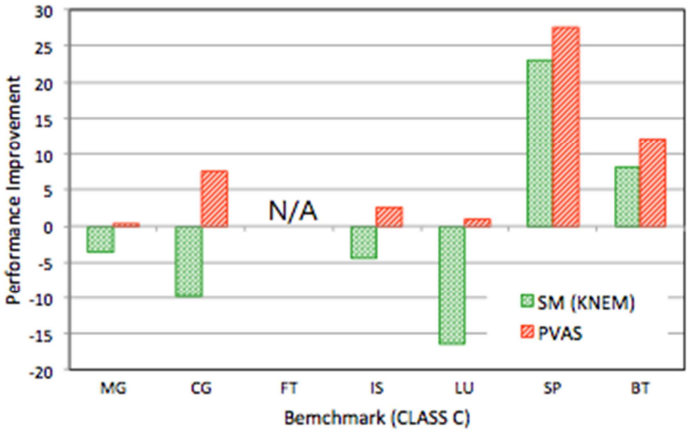


図 4　NPB の実行結果(SM を 1 とした時の相対性能［%］)

図 5 の左のグラフは，IMB の all-to-all 通信を実行したときのメモリ消費量を示している．図に示すとおり，プロセス数が 240 のとき，PVAS を適用した方式は共有メモリを用いる方式よりもメモリ消費量が約 264 MB 小さくなった．これは，共有メモリを用いる方式では，共有メモリのマッピングのためにページテーブルが肥大化しているためであると考えられる．図 5 の右のグラフは IMB の all-to-all 通信を実行したときの，システム全体のページテーブルサイズを示している．グラフに示すとおり，共有メモリを用いる方式は PVAS を適用した方式と比べ，プロセス数が 240 のときにページテーブルサイズが約 256 MB 大きくなっている．これは，MPI ライブラリのメモリ消費量の差に当てはまる．また，プロセス数の 2 乗のオーダでページテーブルサイズが増加していることがわかる．

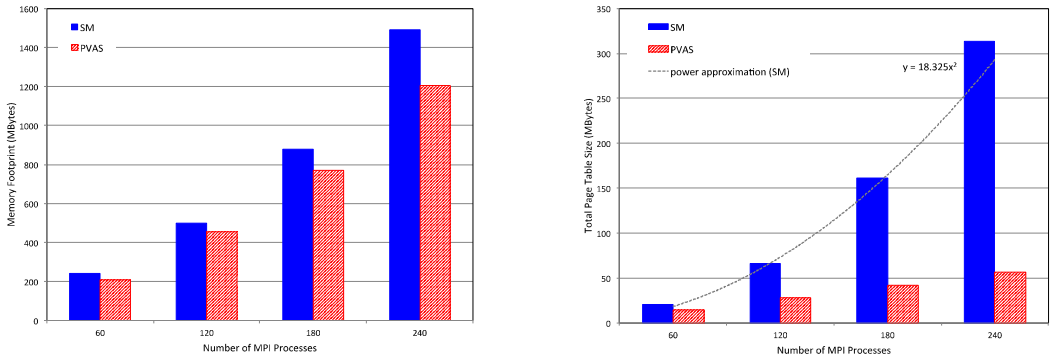


図 5　MPI ライブラリのメモリ消費量とページテーブルサイズの増加量

このように，PVAS を用いると，メニーコアアーキテクチャを用いるシステム上で高速かつメモリ消費量の少ないノード内通信を実現することができる．メニーコアアーキテクチャはエクサスケールのシステムでの採用が有力視されており，PVAS がシステムソフトウェアの側面から，エクサスケールのシステムの実現に寄与することができると考える．

② 軽量スレッド

メニーコア環境で並列アプリケーションを実行する場合，コア間のロードバランスを均等にすることが求められる．コアごとに処理する仕事量が異なると，仕事量の多いコアの実行時間に依存して，全体の実行時間が大きくなってしまう．ロードバランスを行う方法として，オーバサブスクリプションとマイグレーションを組み合わせる方法が挙げられる．ノード上で，コア数を大きく超える数の並列プロセスを動作させ，負荷分散の粒度を細かくする(オーバサブスクリプション)．そして，動的にコア間でプロセスの移動を行う(マイグレーション)ことで，均等に負荷を分散する．

オーバサブスクリプションを行うためには，コア上でタスク切り替え（コンテキストスイッチ）を行う必要がある．従来の OS では，プロセス間のコンテキストスイッチはアドレス空間の切り替えを伴うため，OS カーネルに遷移する必要があり，オーバヘッドが大きくなる．また，プロセスは OS カーネル内でのみタスクスケジューリングが可能であるため，マイグレーションをユーザレベルで制御できず，負荷分散の効率が低下するという課題がある．

そこで本プロジェクトでは，ユーザレベルでコンテキストスイッチ，タスクスケジューリングが可能なプロセスとして，User-Level Process (ULP) を提案，開発した．ULP は PVAS を拡張することで実装されている．ULP の概要を図 6 に示す．図に示すように，PVAS タスクに複数の PVAS パーティションを割り当てる．各 PVAS パーティションに ULP として実行するプログラムをロードし，PVAS タスクが実行する ULP を陽に切り替えることで，オーバサブスクリプションを実現する．同一 PVAS タスク内の ULP 群は同じアドレス空間で動作するので，ユーザレベルでコンテキストスイッチが可能となり，低オーバヘッドでオーバサブスクリプションを実現できる．

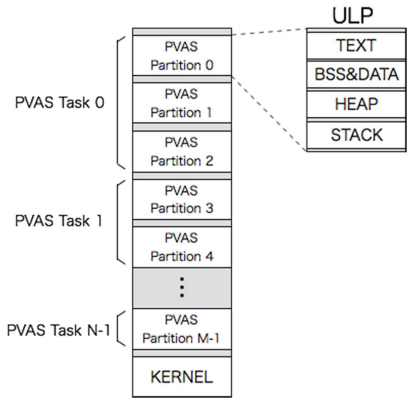


図 6　ULP の概要

また，同一ノード上の PVAS タスクはアドレス空間を共有するので，図 7 に示すように，ユーザレベルでコアをまたいだタスクスケジューリングが可能となる．ユーザレベルで ULP のマイグレーションを制御し，より効率的に負荷分散を制御できる．

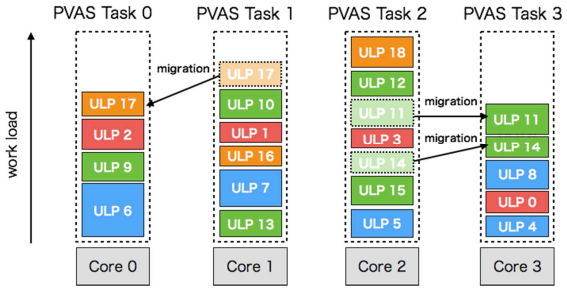


図 7　ULP のノード内マイグレーション

予備評価として，ULP のコンテキストスイッチの性能を評価した．1,000 個の並列プロセスを 1 コア上で動作させ，各並列プロセスが 1,000 回コンテキストスイッチを実行するまでの時間を測定した．並列プロセスをカーネルレベルプロセス，カーネルレベルスレッド，ユーザレベルスレッド，ULP とした場合において測定を実行した．測定にはマルチコアプロセッサである Intel Xeon を用いた．図 8 のグラフに示す通り，ULP は，ユーザレベルでコンテキストスイッチ可能なユーザレベルスレッドと，ほぼ同等の性能を実現することができた．

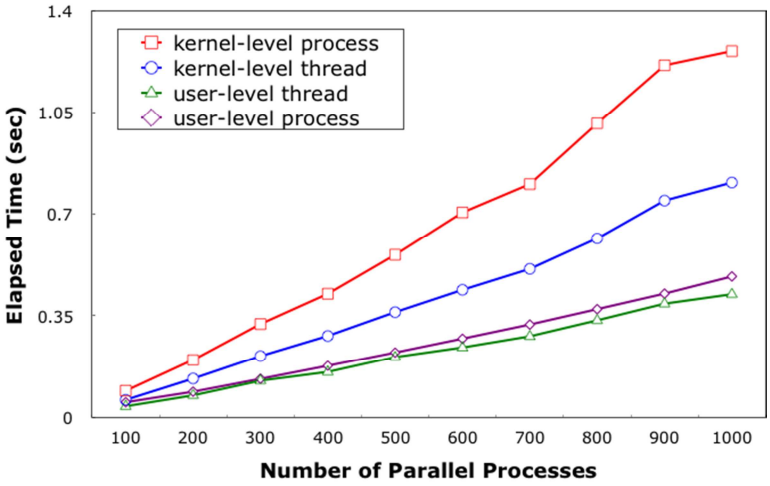


図 8　コンテキストスイッチ時間の比較

現在，ULP の特徴を利用した MPI ライブラリの開発を米アルゴンヌ国立研究所（ANL）と共同でおこなっている．MPI プロセスを ULP として起動することで，MPI アプリケーションの実行時に，効率的な負荷分散を実現することを目標としている．ANL では ULP のタスクスケジューラの実装が概ね完了しており，その評価を今後予定している．ULP の研究開発は，本プロジェクト終了後も，DOE-MEXT 共同研究における共同研究の一環として継続していく予定である．

③　スペアノードによる故障ノードの代替

ポストペタスケールにおいてはノード数が O(10)以上増大すると考えられており，それに伴うノード故障率の増大が深刻な問題になると予測されている．耐故障技術としては既に多くの先行研究があり，また，本研究プロジェクトの共同研究者（Dongarra グループ）は，User-Level Fault Mitigation と呼ばれる MPI で実現されたユーザレベルの故障レジリエンス技術の研究をおこなっている．堀グループでは，ULFM の先を見据え，アプリケーションが ULFM 等を用いて，故障耐性を実現する際に，ULFM を補完する研究を本プロジェクト後期より開始した．

ノード故障が発生した場合，動的な負荷分散をおこなっていないようなアプリケーションでは，故障ノードが使えなくなった時の対処が難しい，例えば，多くの科学技術計算で使われているステンシル計算では，静的な負荷分散であるため，ノード故障への対処が困難である．そのようなアプリケーションでは，故障ノードをスペアノードで代替する方式が有望と考えられる．しかしこの場合でも，図 9 に示すように，アプリケーションの通信トポロジーをネットワークのトポロジーに最適にマッピングしてあったとしても，スペアノードによる代替により，この最適性が失われ，新たに生じるメッセージ衝突により通信性能が低下する可能性がある．

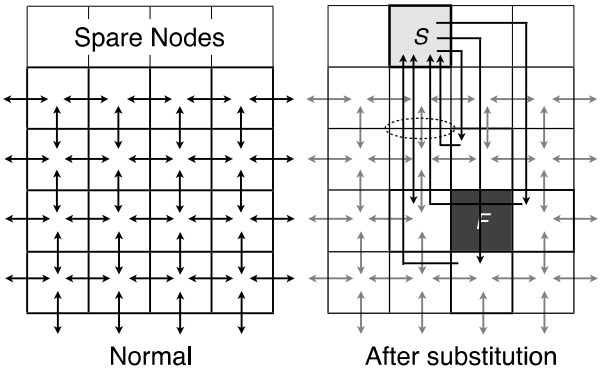


図 9　故障ノードの代替により生じる通信衝突

通信性能の低下は，故障ノードと対応するスペアノードの位置関係に依存すると考えられ，スペアノードの割り当て手法により違いが生じると考えられる．そこで，我々はスライディング方式（図 10）と呼ばれるスペアノードの代替アルゴリズムを開発し，同時にこれらのスライディング手法をハイブリッドに組み合わせる方式を提案した．

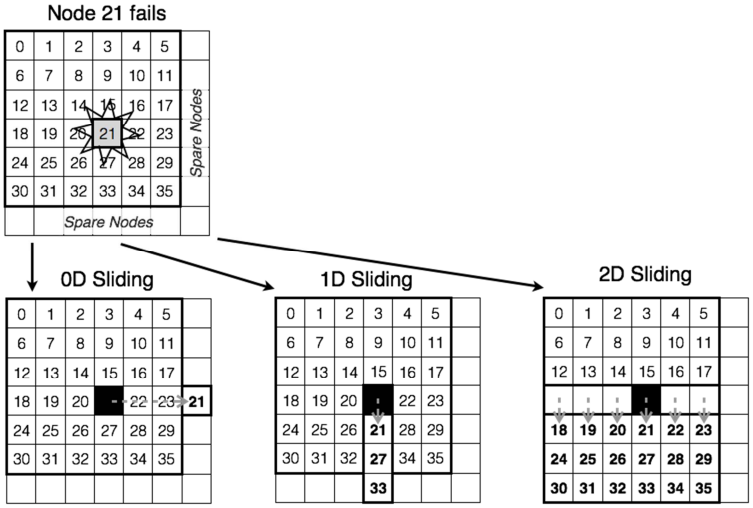


図 10　スライディング手法

提案されたハイブリッドスライディング方式は，それぞれ異なるネットワークトポロジーを持つ京コンピュータ，BG/Q（Jülich Supercomputing Center），および TSUBAME2.5（東工大）上で評価された．図 11 にその結果を示す．京は 12^3，BG/Q は 16*8*8 のノード数とトポロジーを用いて，7P ステンシル（図 11 左），Barrier と Allreduce の集団通信（図 11 右）における通信性能の低下の実測値（縦軸）を示す．これらのグラフで横軸は故障ノード数である．BG/Q では，スペアノードを確保した時点で，集団通信性能が低下するため，判例に“*”が付いている値は，スペアノードなしの場合をベースとした時の値である．京および BG/Q ともにカルテシアントポロジーを持つネットワークである．カルテシアントポロジーはステンシル通信パターンとの相性が良く，衝突が生じないようなノード割り当てが可能である．一方，Fat-tree トポロジーにおけるこのような割り当ては存在しない．このため，別途，Fat-tree トポロジーを持つ TSUBAME2.5 上でも同様の計測をおこなったが，TSUBAME2.5 では京や BG/Q で見られたような通信性能低下は見られなかった．

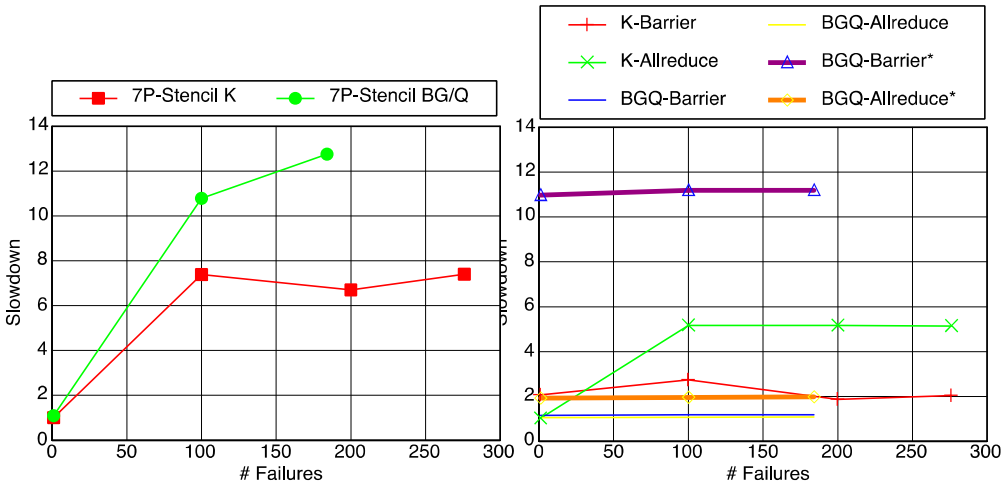


図 11　スペアノード代替による通信性能低下

将来のスーパーコンピュータがより多くのノード数を持つと考えると，故障ノードの存在が常態化し，ネットワークトポロジーの選定，集団通信の最適化などの設計時から，考慮すべきであると考えることができる．なお，本研究は，プロジェクト後期から始まったため，未だ最適な代替手法については研究の余地が残されていると考えるが，他に類をみないユニークな研究であると考える．

④　MPI-IO の最適化

本プロジェクトの当初は，MPI-IO のマルチスレッド化による高速化と，POSIX API によるファイル I/O の最適化の研究を進めていたが，後半からは MPI-IO に注力することとなった．しかしながら，POSIX API での最適化において，ファイル I/O デバイスへの I/O 要求を敢えて絞る，スロットリングという技法により性能が向上することが確認されたため，MPI-IO に対してもこの技法を適用することを研究対象とした．

集団型 MPI-IO では，そのままでは細かい I/O 要求が多数発行されるだけでなく，ファイルへの I/O の順序がバラバラになり，性能が出せないという問題があることが知られている．この問題に対処するために，Two-phase I/O という技法が有効であることが知られている．Two-phase I/O では，アグリゲータと呼ばれるプロセス群が，各 MPI プロセスから発行される I/O 要求をファイルの並びに整列させ，それから実際のファイル I/O をおこなう．しかしながら，アグリゲータを計算ノードのどこに配置すれば良いのか，といった問題がある．

我々は，スロットリングとアグリゲータの最適配置の 2 点に着目し，MPI-IO を高速化する手法を新たに開発し，これを“EARTH”と呼ぶことにした．EARTH では，以下 3 点に着目し，既存の MPI-IO に改良を施した．

- ベースとなるファイル I/O デバイスの配置，ストライピングパターンおよびネットワークトポロジーを考慮し，ファイル I/O 時に発生する可能性のある通信衝突を回避するようアグリゲータを配置する．
- 同時に，ファイル I/O への要求にスロットリングを施し，ファイル I/O への負荷を低減する．
- 上記スロットリングを適応するのであれば，アグリゲータへの集約も，スロットリングに合わせて段階的に行うことで，アグリゲータへの集約時の通信負荷も低減可能となる．

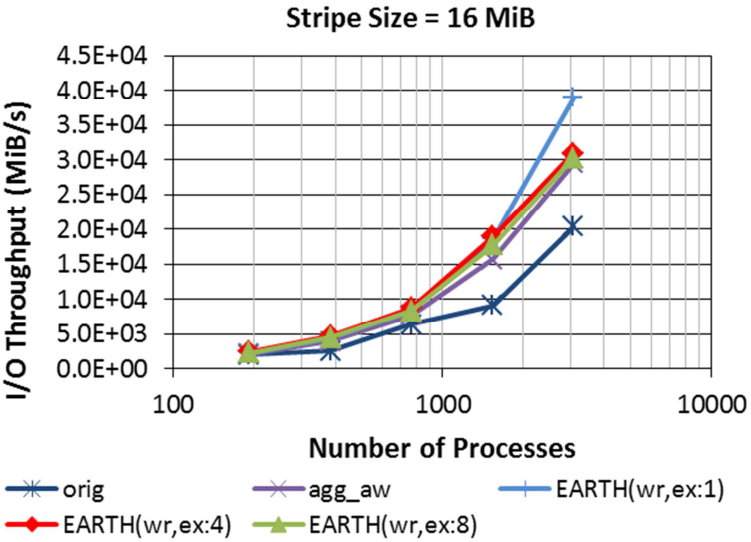


図 12　HPIO ベンチマークを用いた EARTH 性能向上

図 12 に HPIO ベンチマークを用いた際の EARTH の性能向上を，京上で計測した結果を示す．このグラフにおいて，スロットリングの程度を“ex:*n*”として示してある．このグラフから，既存実装

(図中の“*orig*”)やアグリゲータ配置のみ最適化した実装(図中の“*agg_aw*”)に比べ, EARTH の効果は特に MPI ランク数が大きいときに顕著であると考えることができる.

同様の評価を京とは全く異なるアーキテクチャを持つ TSUBAME2.5 でも評価をおこなっており, EARTH 手法の汎用性について検証を進めている最中である.

現在, EARTH の一部の機能を MPI-IO のデファクト標準である ROMIO に包含させるべく, 関係者と協議をおこなっている. また, EARTH 全体についても, 機が熟せば同様に ROMIO 実装に組み込むべく努力する予定である.

⑤ ULFM の京コンピュータへの移植

テネシー大学が開発している故障レジリエンス機能を備えた User-Level Fault Mitigation (ULFM) MPI を理研が所有する京コンピュータへの移植を進めている. ULFM を用いてプログラムを記述することで, プロセスの予期せぬ停止やノード故障が発生してもユーザプログラムの動作は可能になる. 本移植に際しては, 別途京のジョブマネージャの変更が必要である. これは既存のジョブマネージャでは, プロセスの異常停止に際し, そのジョブ全体を停止させることが仕様となっているからである. また, ジョブマネージャが検出した異常情報も ULFM で受け取る必要がある. これらの改変はジョブマネージャの開発元である富士通に H25 年度に依頼した. この改変を受け, ULFM の FX10 への移植をおこなった. 既に基本機能の動作は確認済みである. 本プロジェクト終了までに京上での評価をおこなう予定であったが, 富士通に依頼したジョブマネージャの変更が京のジョブマネージャに反映されていないことが判明し, 京による評価ができなくなった.

3. 2 メニーコア用 OS カーネル(日立 須藤グループ)

(1)研究実施内容及び成果

① メニーコア OS

(1-1) PVAS の応用研究

須藤グループは H27 年度から本プロジェクトに参加し, 理研の堀グループから引き継ぐかたちで PVAS の応用研究に取り組んでいる. PVAS の応用研究として, Open MPI のノード内通信の高速化を拡張し, Open MPI のノード内通信において, 不連続なデータの送受信を高速に実行することを可能にした. なお, 本研究において, 本プロジェクトの共同研究者でもあり, Open MPI の主要開発者でもある G. Bosilca 氏(Dongarra グループ)のアドバイスを受けている.

MPI では, 派生データ型を用いることで, メモリ上で不連続なデータを一度の通信で送受信することができる. Open MPI のノード内通信においては, 派生データ型を用いる不連続なデータの送受信に共有メモリを利用している. 図 13 に示すように, 送信プロセスの送信バッファから, 不連続なデータを共有メモリ上の中間バッファにパックする処理を行い, 共有メモリ上の中間バッファから, 受信プロセスの受信バッファに不連続なデータとしてアンパックする処理をおこなう. 共有メモリを経由してデータを送受信するため, メモリコピーの回数が多くなり, 通信遅延が増加する.

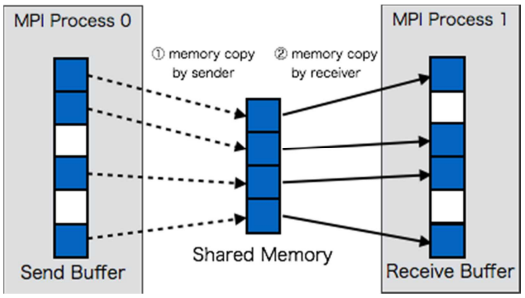


図 13 共有メモリを用いた不連続なデータの送受信

PVAS によって高速化した不連続なデータの送受信では，MPI プロセスを PVAS タスクとして起動し，送信バッファから受信バッファにデータを直接コピーする．共有メモリを経由せずにデータを送受信することで，メモリコピーの回数増加を抑制する．また，図 14 に示すように，送受信するデータを半分に分割し，送信プロセスと受信プロセスが並列にコピーすることで，通信遅延を低減する．

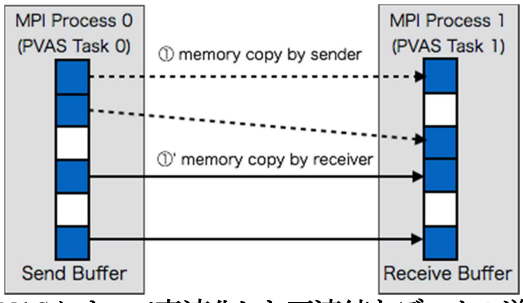


図 14　PVAS によって高速化した不連続なデータの送受信

fft2d_datatype を用いて高速化した Open MPI のノード内通信を評価した．fft2d_datatype は 2 次元フーリエ変換の計算コードであり，配列の転置に MPI の派生データ型を用いた不連続なデータの送受信を利用している．図 15 の左のグラフは，fft2d_datatype を Intel Xeon Phi 上で実行したときの実行時間を示している(240 プロセス)．横軸は配列サイズである．図 15 の右のグラフは共有メモリを用いる方式を基準とした性能の改善率を示している．グラフに示すとおり，PVAS を用いて高速化したノード内通信を用いると，不連続なデータの送受信の通信遅延を低減し，実行性能を最大で約 21%改善することができた．

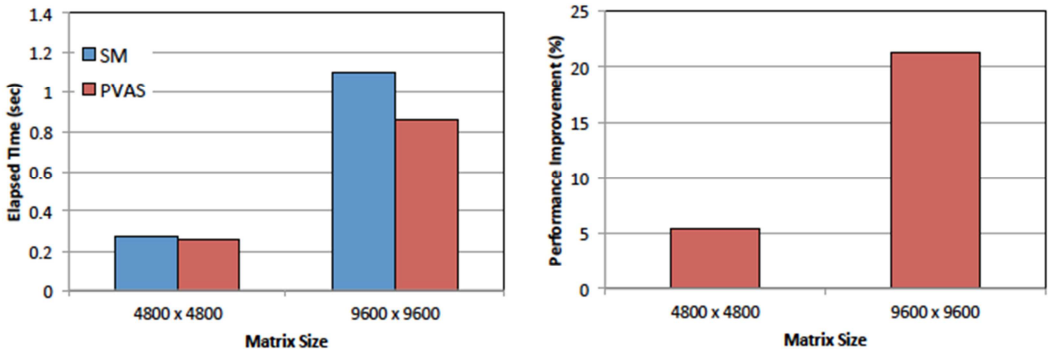


図 15　fft2d_datatype の実行結果

(1-2) PVAS の McKernel への移植

須藤グループでは，PVAS の応用研究の他に，PVAS の McKernel への移植をおこなっている．McKernel は，理研が主体となって開発しているポスト京コンピュータ向けのマイクロカーネルである．PVAS を McKernel に移植することで，本プロジェクトの研究成果を実際に運用されるシステムにフィードバックする狙いがある．理研の McKernel 開発グループと密に連携をとって PVAS の移植を推進し，概要設計が完了している．以下に，検討した設計の内容を述べる．

McKenrel は一部のシステムコールの実行を Linux 上のプロセスにオフロードする仕組みになっている．図 16 に示すように，ノード上の物理資源を分割し，Linux と McKernel を 1 つのノード上で動作させる．McKernel 上のプロセスが実行する一部のシステムコールは Linux 上で動作している mcexec プロセスにオフロードされる．McKernel 上のプロセス 1 つにつき，1 つの mcexec プロセスを Linux 上に起動する必要がある．

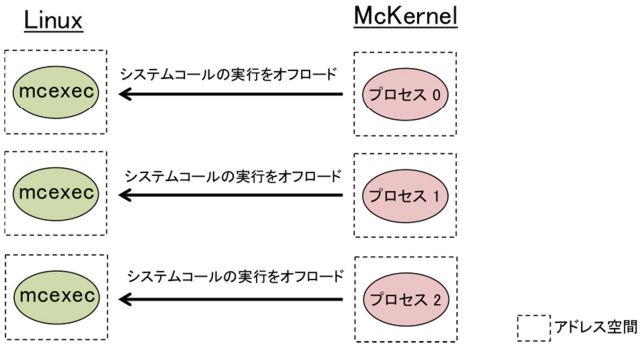


図 16　McKernel の仕組み

PVAS を McKernel 上に実装する場合，McKernel 上の PVAS タスクに対応する mcexec プロセスを Linux 上に起動する必要がある．このとき，PVAS タスクに対応する mcexec プロセス群は，対応する PVAS タスク群と同様に，アドレス空間を共有する必要がある（図 17）．アドレス空間を共有しなければ，仮想アドレスを引数にとるシステムコールを正常に実行することができなくなる．

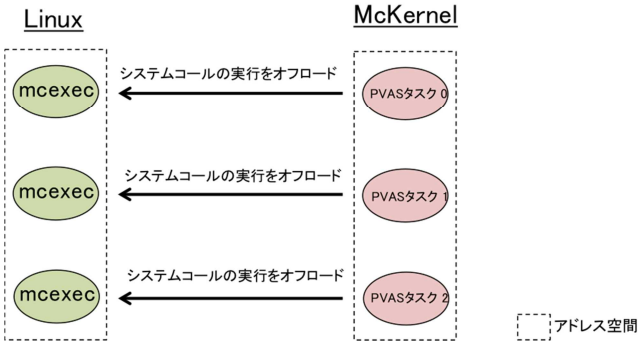


図 17　PVAS を McKernel に移植した場合

一般に，プロセスを生成するために fork システムコールが用意されている．しかし，fork で作成したプロセスは異なるアドレス空間で動作するため，fork では，McKenrel 上の PVAS タスク群に対応する mcexec プロセス群を同一アドレス空間に作成することができない．

そこで，fork ではなく，clone システムコールを用いて，アドレス空間は共有するが，その他のプロセス資源（ファイルディスクリプタテーブルやシグナルハンドラ等）は共有しない特殊なプロセスを生成し，それを mcexec プロセスとする方式を検討した．この方式では，図 18 に示すように，PVAS タスクに対応する mcexec プロセスを起動するための mcexec（manger）プロセスを用意する．親プロセスが PVAS タスクを生成したら，mcexec（manager）プロセスに，それに対応する mcexec プロセスの生成を依頼する．依頼を受けた mcexec（manager）プロセスは，clone システムコールのフラグに CLONE_VM のみをセットし，mcexec（manager）プロセスとアドレス空間以外のプロセス資源を共有しない，特殊な mcexec プロセスを生成する．こうすることで，mcexec（manager）に生成された mcexec プロセス群は，対応する PVAS タスク群と同様に，同一アドレス空間で動作することになる．アドレス空間を共有するため，仮想アドレスを引数にとるシステムコールを正常にオフロードすることができる．また，アドレス空間以外のプロセス資源は共有しないので，ファイルディスクリプタテーブル等，プロセスが個別に持っている資源にアクセスするシステムコールも正常に処理することができる．

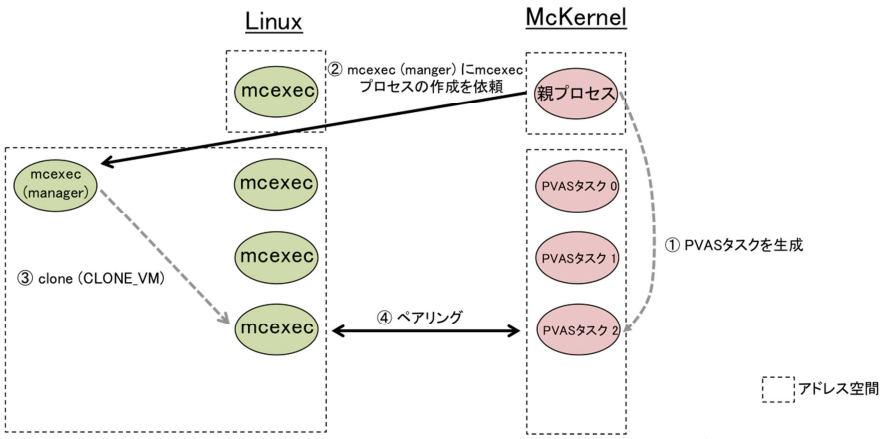


図 18　PVAS に対応する mcexec プロセスの生成

PVAS の McKernel の移植は，本年度内での完了を目標とし，実装を進めている．

3．3　高スケーラブルな通信と I/O の実現（東京農工大学　並木グループ）

(1)研究実施内容及び成果

本研究では，メニーコア CPU 上と，ホスト CPU 上にそれぞれ独立した OS カーネルがあるものとし，それぞれの CPU の特徴を活かし，総体として高性能化を達成するようなソフトウェアアーキテクチャを目標としている．このような複数 OS 構成では，個々の OS 間を適切に連携する方式が必要不可欠となる．

具体的には，次の研究を行った．

［1］　複数 OS を適切に連携し，個々の資源を仮想化し，シングルシステムイメージを提供するようなプロセス管理，メモリ管理などの資源管理の設計と実現

［2］　マルチコア，メニーコアプロセッサに対して，応用プログラムに適応した資源割当ての方式の研究．特に，本研究で得られた基盤ソフトウェアを用いて MapReduce 処理を評価し，高性能計算に対する計算基盤を検証

本質的な研究課題は，異なる特性を持つ CPU に適した仮想化と資源管理の方式を明らかにすることである．Intel Xeon のような逐次処理に性能を発揮するコアと，Intel Xeon Phi のようなプロセッサの個数は多いが個々のプロセッサの性能は低いようなハードウェア構成において，それぞれの長所を活かし，短所を補う方式が肝要となる．先行研究[1]では，オフロードエンジンの方式を提案しているが，異なる CPU 間でのメモリ共有，同期制御など性能向上においていくつかの課題があった．

具体的に本分担研究では次の技術的な課題を研究した．

［1］　マルチコア・メニーコア間での実行制御
高速なコア間同期など両プロセッサ同士の OS 連携の高速な基本機構の実現

［2］　メニーコアにおける入出力などの代行処理方式
メニーコアである Intel Xeon Phi は数値計算に特化アーキテクチャとなっており，入出力はできない構成となっていることから，メニーコア側で不可能な処理をマルチコア側で代行する必要がある．このために，OS 間の連携処理を研究した．

［3］　コア間で異なるアドレス空間の仮想化
Intel Xeon と Intel Xeon Phi では，物理的には異なるメモリであるが，相互に参照可能なメモリ構成となっている．アクセスコストの差はあるが，メモリは両プロセッサ間で共有可能となっている．これらのメモリをコア上の実行実体であるプロセスから統一的に扱う方式を研

[1] Chris J. Newburn., et, al.: Offload Compiler Runtime for the Intel Xeon Phi Coprocessor (ISC2013) および James Reinders., et, al.: An Overview of Programming for Intel Xeon processors and Intel Xeon Phi coprocessors, Intel (2012)

究した.

［4］ MapReduce 処理の方式検討

近年のビッグデータ処理などでは，大量データの処理が必要不可欠となっている．特に，MapReduce については多くの局面で利用される処理モデルであるが，マルチコア・メニーコアの連携によりプログラミングできる計算基盤を検討し，基礎評価を行った．

これらの研究成果として，次の成果を得た．

［1］ OS 間連携機構:ファイル I/O 代行機構

複数 OS 間の連携による資源管理の実現可能性を明らかにするための，メニーコアでは不可能なファイル I/O 機能の提供，コア間での同期，メモリ管理の OS 間連携機構の実現方式を提案し，性能評価により，方式の有効性を確認した．

［2］ マルチコア・メニーコア双方の OS を連携するタスクモデルと管理方式

PVAS を拡張した M-PVAS (Multiple PVAS)を提案，試作し，MPI 代行処理により方式の有効性を明らかにした．マルチコア・メニーコア上の PVAS を連携，PVAS のアドレス空間をプロセッサ間で仮想化することで，双方で共有のアドレス空間を実現した．この方式を次の二つに適用し，有効性を確認した．

・MPI の代行機構

・MapReduce への適用による実行系

以下，個々の研究成果を示す．

① OS 間連携機構:ファイル I/O 代行機構

複数 OS を持つアーキテクチャにおける資源管理を実装し，複数 OS 構成法の基本原理を明らかにするための基礎を整備した．メニーコア側において計算を並列実行に特化させる基盤により高効率な環境を提供するとともに，メニーコア側では処理を行わない入出力をマルチコア側と連携する OS 間連携機構を Linux 上に実装した．メニーコアのファイル I/O アクセスをマルチコアで代行する機能について，メニーコア側のプロセスで要求するファイル I/O をマルチコア側 OS である Linux が代行する．この方式により，入出力機構を有しないメニーコアにおいても，ファイルなどの入出力が可能となるだけでなく，並列演算性能が高いが比較的低性能なメニーコアと，逐次演算性能が高いマルチコアにおいて，その得失を考慮した資源管理が可能となる．同時に，メニーコアにおいても，仮想化された入出力装置をマルチコアと同等の API を用いて利用できる計算環境を提供することができた．

設計および実装上の留意点としては，

a) I/O アクセス代行制御の適切な経路

メニーコアとマルチコアの間で，データ転送指令，同期を含む処理内容のコマンド化が不可欠となるが，カーネル内での処理方式，ユーザランド内での処理方式を検討し，比較評価を行った．同期性能の結果から，図 19 に示す方式を採用した．

b) OS 間でのデータ転送をレスコピーで実行

性能上，もっとも重要な点は，マルチコアとメニーコア間のデータ転送の方式にある．入出力装置については，入出力の処理発行とデータの受領は，ファイル I/O のシステムコールを用いるので，応用プログラムからは明示的にその処理を捕捉できる．したがって，補足した API から適切な処理方式を選択することは比較的容易であり，どのプロセッサからデータ転送をおこなえば性能に寄与するかについて判断できる．

マルチコアとメニーコア間のデータ転送については，データのコピーは，予備データからマルチコア側 OS のファイルデータをメニーコア上のプロセスに直接転送する方式が高性能であったため，この方式を採用した．

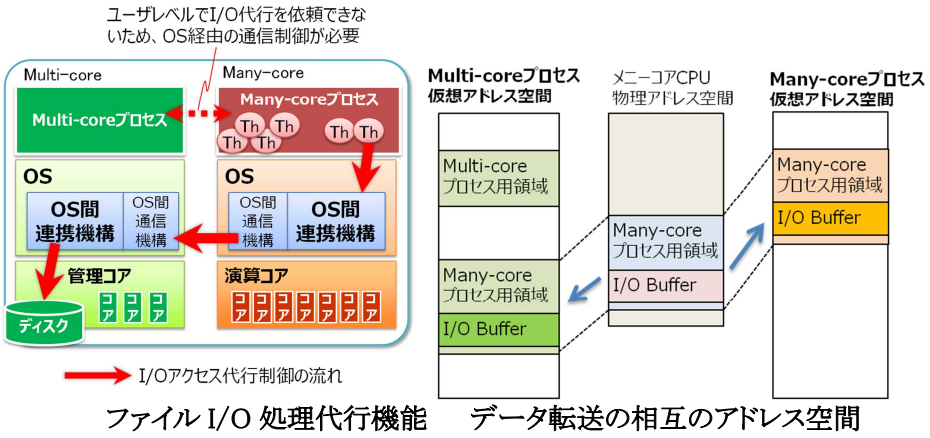


図 19　I/O アクセス代行制御の適切な経路

図 20 に本方式の評価結果を示す．メニーコア側でデータ転送するケースが indirect-access 方式，マルチコア側でデータ転送を行う方式を direct-access 方式とした．本方式により，1.7 倍程度の性能向上を達成できた．

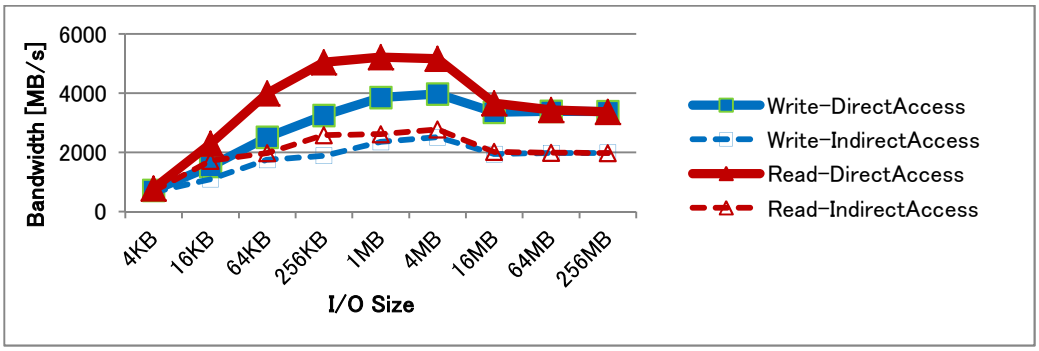


図 20　入出力処理方式の評価

② マルチコア・メニーコア双方の OS を連携するタスクモデルと管理方式

マルチコアである Intel Xeon とメニーコアである Intel Xeon Phi では，物理的には異なるメモリであるが，相互に参照可能なメモリ構成となっている．アクセスコストの差はあるが，メモリは両プロセッサ間で共有可能となっている．そこで，堀グループが提案するメニーコア側の実行形態である PVAS を，異機種環境に拡張することで，マルチコアとメニーコアを「適材適所」的に使い分けることができる．

本研究においては，次の技術的課題が重要であると考えた．

a) 本来は異種であるマルチコアとメニーコア側のアドレス空間を共有するプロセスモデルとアドレス空間モデルの設計
b) 双方の空間の一貫性管理
c) 性能を維持した上での空間管理．特にページング手法

これらの課題を解決する方式として，PVAS を拡張した M-PVAS を提案した．まず，PVAS をマルチコアとメニーコアでまたいだ上で，共有するモデルとして，マルチコアおよびメニーコア上の PVAS の空間を共有するモデルを検討し，実装した．応用プログラムからは同一の仮想アドレス空間を共有しているように見えると同時に，言語 C/C++のポインタ参照により双方のデータをアクセスできる体系になっている（図 21）．M-PVAS では，CPU をまたぐ複数の PVAS 空間を単一のアドレス空間対応付けることができる．

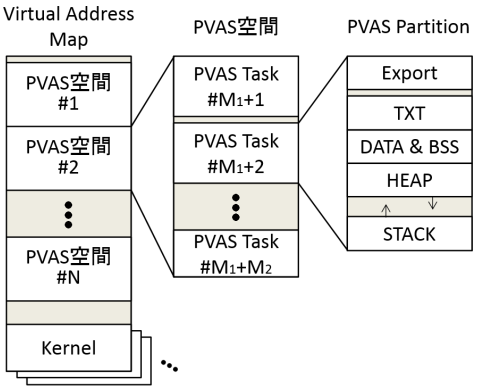


図 21　複数の PVAS アドレス空間を拡張した M-PVAS アドレス空間

本方式では，**エラー! 参照元が見つかりません。**図 22 に示すように，マルチコアとメニーコア双方に仮想アドレス空間管理のためのページテーブルを用意する．ページの内容であるアドレス空間のデータについては，データ転送を行わないが，ページテーブルの情報を相互のコアで転送することで，データ共有を行いながら，管理オーバヘッドの少ない空間共有の機構を実現した．

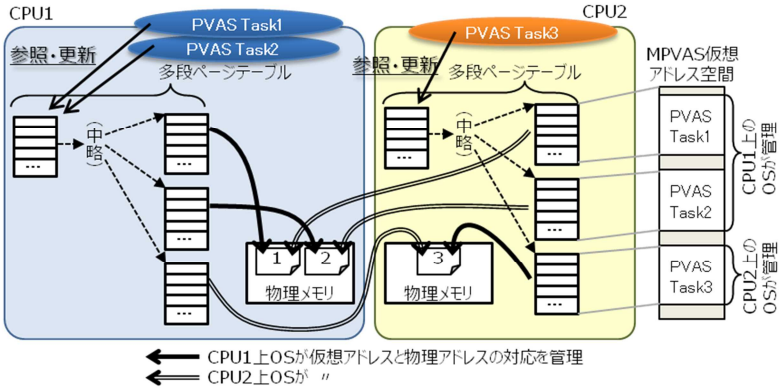


図 22　M-PVAS における M-PVAS アドレス空間管理

本方式を用いて相手側の CPU に null 処理を起動する際の時間を**エラー! 参照元が見つかりません。**図 23 に示す．この図において，横軸は引数で渡すデータのサイズであり，縦軸は処理に要した時間である．Intel オフロード方式が 800 μS 以上必要なのに対して，本方式では数 μS 程度の高い性能を得ることができた．

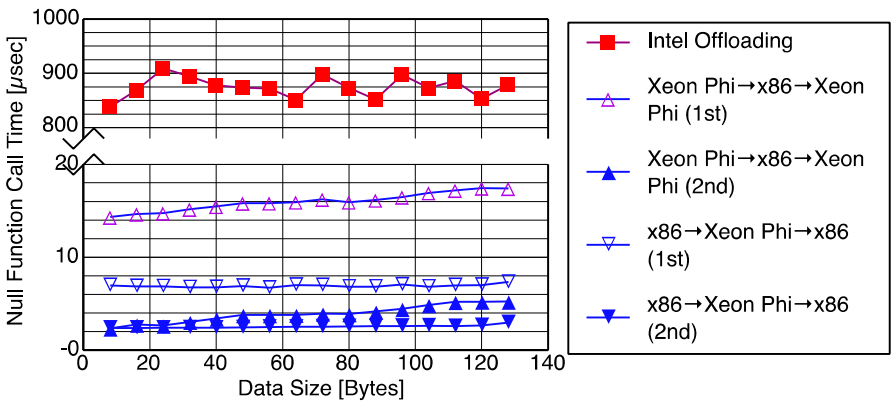


図 23　M-PVAS と Intel オフロード方式の性能比較

M-PVAS によるアクセス性能高い空間共有の方式を MPI の代行機能に適用した．**エラー! 参照元が見つかりません。**図 24 に M-PVAS を用いた MPI の I/O 代行処理の性能を示す．空間共有の機構を用いてメニーコア側からマルチコアに処理代行を依頼する．本方式により，遅延については最大 56%，帯域については最大約 2 倍の性能向上が得られた．

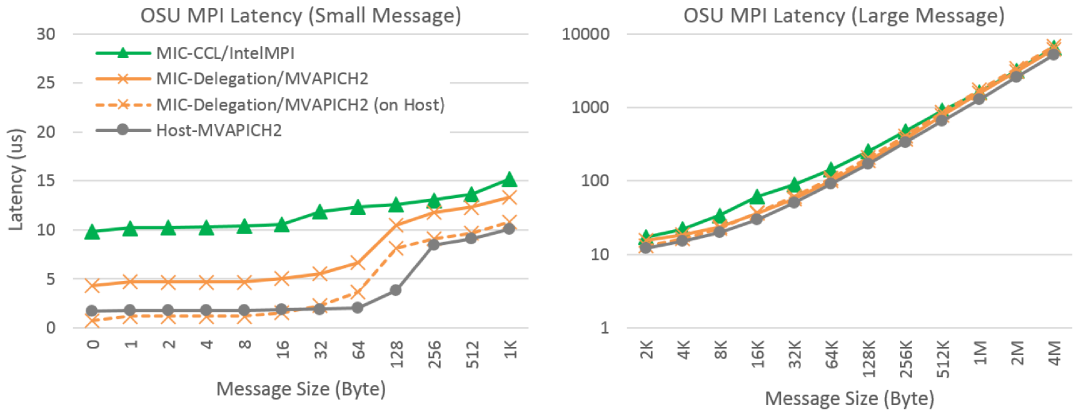
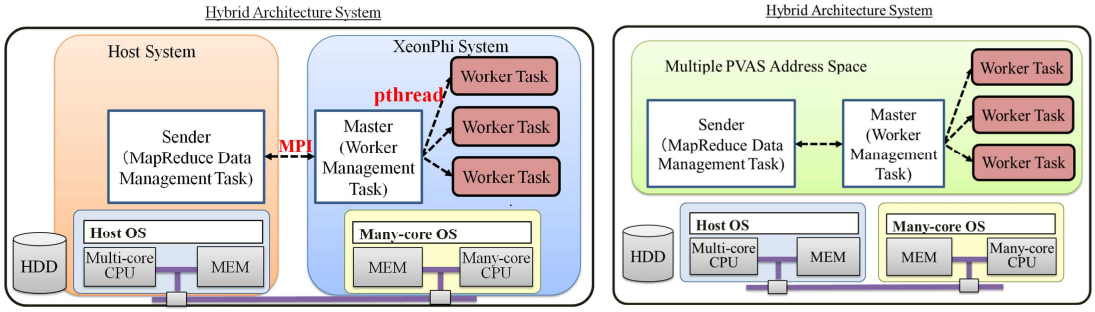


図 24 M-PVAS を用いた MPI 代行処理の性能比較

また，M-PVAS を MapReduce に適用した．MapReduce では，実行実体であるワーカの実行時制御，ワーカに渡すデータ転送などがその性能上，重要な課題となる．Intel Xeon Phi における MapReduce として，MRPhi などが知られている．従来手法では，Pthread などを用いていることから，マルチコアとメニーコア間でのデータ共有は内部で明示的に処理する必要があり，性能を得るためにはデータ転送に関する処理記述は煩雑である(図 25 左)．



XeonPhi における既存の実行手法　　M-PVAS による MapReduce

図 25 MapReduce の実装方式の違い

そこで，図 25(右)に示す M-PVAS を用いてアドレス空間共有し，データ転送を M-PVAS の機構を用いることで，性能向上ないしは処理記述の柔軟性を実現する方式を試作し，モンテカルロ法のベンチマークプログラムにより性能確認を行った．その結果 2 倍程度の性能向上が得られたが，さらに詳細な分析が今後の課題となっている．

3．4 故障レジリエンス基盤(テネシー大学 Dongarra グループ)

(1)研究実施内容及び成果

During the entire research period, UTK efforts were focused on 3 major areas. First, improving the design of the resilient concepts to make them more usable from the user perspective and more efficient from the hardware perspective. Second, completing the reference implementation of these concepts and make them available to a large audience, as a production-ready software

package what can work on any environment, from small clusters to large-scale platforms. Finally, a significant effort has been pushed in broadening the impact of this research, by creating a community of researchers interested in using these concepts to design novel fault management techniques, and maintain a high level of interactions with this community. In the remaining of this document, each one of these topics will be described in details, and all the efforts will be highlighted.

Conceptual Design

The main goal of the ULFM approach remained unchanged over this period: define a minimal set of features to notify application processes about failures, and allow the user to interrupt the normal behavior of his application and to restore communication capability after process failures. Exchanges with developers of MPI applications who envision to introduce resilience in their codes, or who have tried to do so using the ongoing ULFM specification, as well as fruitful discussion in the MPI Forum, mainly inside the Fault Tolerance Working Group and in plenary sessions, have highlighted few opportunities for improvements in the ULFM specification. Ambiguities in the proposed text have been removed, and replaced by a clearer description of the required behavior. The RMA chapter has been entirely rewritten, to account for the changes in the RMA chapter introduced in the version 3 of the MPI standard. The MPI_Comm_shrink and MPI_Comm_agree routines, which are part of the Fault Tolerance extensions, have been simplified, allowing for a more diverse usage. Non-blocking versions of some of these constructs have been also added, to allow for more flexible recovery strategies.

Implementation

In same time as improving the concepts design, we wanted to provide a decent implementation that can be used by interested parties for their own research and developments. The first part of the award period was dedicated to providing correct, but non necessary efficient and scalable, support for all concepts. In parallel we advanced on the theoretical side by developing all the necessary tools and algorithms to design and implement more scalable versions of the needed concepts.
More specifically, the last year of the project has been almost entirely focused on advancing on the performance side of the two major operations proposed for the handling of the faults (revocation and agreement), and on promulgating the extension to the MPI standard to the users communities. In same time, a significant effort has been done to improve the code quality of the available implementation, in order to provide a stable, efficient and portable implementation to the user communities.
Similarly to past years, exchanges with developers of MPI applications who envision to introduce resilience in their codes, or who have tried to do so using the ongoing ULFM specification, as well as fruitful discussion in the MPI Forum, inside the Fault Tolerance Working Group and in plenary sessions, have highlighted a few places for improvements in the ULFM specification. The main goal of the ULFM approach remains unchanged: define a minimal set of features to notify application processes about failures, and allow the user to interrupt the normal behavior of his application and to restore communication capability after process failures. Ambiguities in the proposed text have been removed, and replaced by a clearer description of the required behavior. The RMA chapter has been entirely rewritten, to account for the changes in the RMA chapter introduced in the version 3 of the MPI standard. The MPI_Comm_shrink and MPI_Comm_agree routines, which are part of the Fault Tolerance extensions, have been simplified, allowing for a more diverse usage.

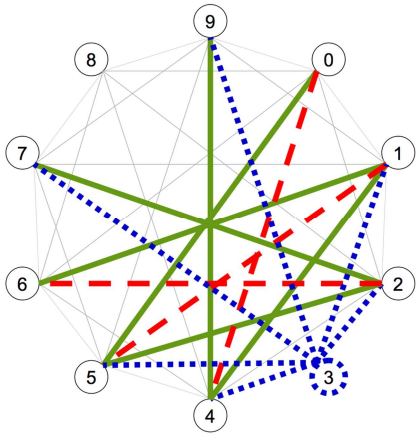


図 26　Binomial Graph Topology

On the implementation front, the MPI_Comm_revoke and MPI_Comm_agree operations have been the focus of our efforts for this year. These two routines are critical to the efficiency of a resilient code, and are by nature costly because they need to be fault-tolerant themselves. MPI_Comm_revoke is provided to the user to notify (asynchronously) all processes belonging to a communicator that an exception happened, and to stop the normal execution flow (e.g. to trigger a collective repair of the communicator). At the conceptual level, it implements a form of reliable broadcast: if one process receives such revoke notification, all processes of the communicator must receive one. Because failures happening during the revoke operation can introduce messages loss, the protocol that implements the revoke operation must be fault-tolerant, and ensure that if one notification is delivered, all notifications are delivered. The first implementation of the revocation was done at the Runtime Environment (RTE) level, over the Out-Of-Band (OOB) network. This implementation was not fault tolerant and required a large number of messages ($O(N^2)$). We redesigned the algorithms integrating research in resilient graph topologies and successfully improved not only the resilience and performance of the revocation algorithm but also the impact on the network infrastructure and the number of messages. The final algorithm that we will be delivered with ULFM is based on the Binomial Graph topology (図 26). This graph minimize the number of messages and the degree of the graph, while providing a strongly resilient topology, that can only be bipartite by the sudden disappearance of more than half of the processes. Moreover, the new algorithm is implemented at the BTL level, and takes advantage of the fast network interconnect available on most of the HPC clusters.

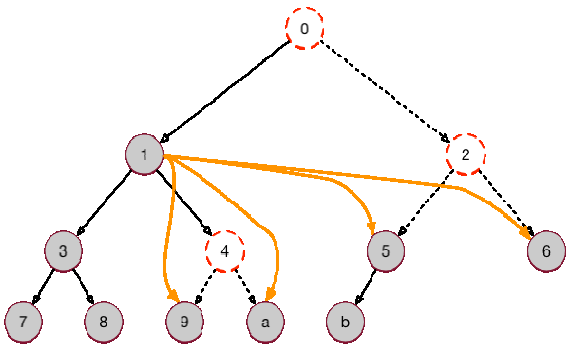


図 27　Example of Dynamic Topology

The MPI_Comm_agree routine was also relying, for parts of its services (namely the all-reduce tree maintenance) on messages at the OOB level. The overhead imposed by the OOB mechanism has been hindering the performance and scalability of the agreement. We completely reworked the agreement to use directly the fast network communication infrastructure available in Open MPI (namely the BTL). This step provided a significant boost in terms of performance of the agreement algorithms. However, this highlighted few implementation issues with quadratic search involved in the original algorithms (due mainly to the MPI semantics of translating ranks between communicators). As a result, we decided to redesign the agreement implementation from the ground up. Dynamic topologies have been introduced, topologies that remains optimal as long as no new processes disappear, and that are updated upon a successful completion of an agreement (due to the semantic of the agreement itself a consensus on the dead processes is established during the agreement, allowing for a consistent global view of the alive processes). This topology is described in the figure above (図 27). The important change is the orange lines, which represent the new connections that are established to cope with the discovery of dead processes (signaled by the red dashed line). These lines connect a process to the leftmost ancestor still alive, and count for one of the most costly, but critical, operation during the agreement. Without going in too much details, we have designed a new consensus algorithm named Early Returning Agreement, with a worst case cost of $\Sigma_{f=1..F}(\log(P-f))$ where F is the number of fault discovered during a single agreement and P is the total number of processes.

Moreover, users reports signaled recurring corner cases in the previous agreement implementations, where the agreement routine was unable to provide its service because of the presence of failures. We wrote a reference implementation of the agreement, based on an existing Early Termination Agreement protocol, which work in asynchronous all-to-all phases, ensuring a correct result in any possible failure scenario. In addition to this reference implementation, we initiated works on more efficient implementations, which will take advantage of the high resilience property of the Binomial Graph, to reduce the number of messages while still providing the insurance of safe agreement despite failures in most situations.

Broader Impact

A particular focus was on reaching out to the user communities. In this direction, we provided user support through different media: we wrote, and made accessible a set of tutorials, hands on, and manuals. All the examples and other information regarding this subject has been made available on the freely available project website (http://fault-tolerance.org). In addition, many motivating examples using the ULFM capabilities have been developed to illustrate the specification to the MPI Forum, and these examples are accessible to the users. Users feedback and requests have been integrated in the specification or implementation, and we continued to provided user support through the ULFM users mailing list during the entire period.

ULFM-based tutorials have been presented at the EuroMPI/Asia MPI Conference in Kobe in September 2014, and then a similarly shaped tutorial at the SC'14 conference in New Orleans in November 2014 and SC'15 in Austin, TX in November 2015. Presentation on the subject, including newly developed technologies have been presented at several meeting, including SIAM PP, SIAM CSE, Euro MPI, SC, ICS. As a result application developers of a significant set of scientific applications (from Sandia, Los Alamos, LLNL, Stanford and Berkeley) have reached out to us to help them integrate ULFM-based resilience to their applications. Some of these extremely encouraging results have been published at top peer-reviewed conferences. As a culmination of these research popularization efforts we held an extremely well-attended

Bird-of-Feather (BoF) session during the SC'15 conference.

3．5 公開ソフトウェア

これまでに述べてきた PVAS および M-PVAS はまとめて一つのパッケージとして公開（http://www-sys-aics.riken.jp/releasedsoftware/software/pvas-1.0.0-beta.tar.gz）されている．また，より品質を高めたバージョンの公開も準備段階にあり，準備が整い次第公開する予定である．本パッケージは，x86 アーキテクチャあるいは，Intel Xeon Phi（KNC）加速器を伴う x86 アーキテクチャのマシンで動作し，主に HPC 分野におけるシステムソフトウェア研究者を対象としている．

3．6 エクサスケールへの貢献

既に述べたように，PVAS はエクサスケールの OS カーネルとして開発が進められている McKernel への移植が進められており，これは今年度中に完了する予定となっている．

また，MPI-IO の改良である EARTH は，MPI-IO の開発元にフィードバックすることが決まっており，正式な MPI-IO として将来公開される可能性がある．この結果，エクサにおける MPI-IO の標準的な実装となる可能性がある．

テネシー大学が開発を進めている ULFM は，MPI の標準仕様となるべく努力が何年も続けられており，次期 MPI の標準仕様となる可能性がある．ここで，京コンピュータへの移植の経験を通じ，エクサへの貢献が期待できる．また理研が進めているスペアノードの研究も，エクサにおいて採用される可能性があると考える．

§4 成果発表等

(1)原著論文発表 （国内（和文）誌 2 件（＋投稿中 1 件），国際（欧文）誌 14 件（＋投稿中 1 件））

[1] Hori, Yoshinaga, Herault, Bouteiller, Bosilca, Ishikawa, “Overhead of Using Spare Nodes”, International Journal of High Performance Computing Applications（投稿中）

[2] 島田，須藤，堀，石川，河野，“メニーコアにおける派生データ型を用いた MPI ノード内通信の高速化，” In 情報処理学会論文誌，情報処理学会（投稿中）．

[3] 島田，堀，石川，“新しいタスクモデルによるメニーコア環境に適した MPI ノード内通信の実装，” In 情報処理学会論文誌，情報処理学会，volume 8, no.2, 2015.

[4] Bosilca, Bouteiller, Herault, Robert, Dongarra, “Composing Resilience Techniques: ABFT, Periodic, and Incremental Checkpointing,” International Journal of Networking and Computing, vol. 5, no. 1, pp. 2-15, January 2015.

[5] Herault, Bouteiller, Bosilca, Gamell, Teranishi, Parashar, Dongarra, “Practical Scalable Consensus for Pseudo-Synchronous Distributed Systems,” SC15, 2015.

[6] Bouteiller, Bosilca, Dongarra, “Plan B: Interruption of Ongoing MPI Operations to Support Failure Recovery,” EuroMPI 2015.

[7] Hori, Yoshinaga, Herault, Bouteiller, Bosilca, Ishikawa, “Sliding Substitution of Failed Nodes,” EuroMPI 2015.

[8] 吉永，堀，石川，“ステンシル計算におけるスペアノードを用いた故障復帰における通信性能への影響について，” In ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集, volume 2015, 2015.

[9] Bouteiller, Herault, Bosilca, Du, Dongarra, “Algorithm-based Fault Tolerance for Dense Matrix Factorizations, Multiple Failures and Accuracy,” ACM Transactions on Parallel Computing, 2014. DOI:10.1145/2686892.

[10] Shimosawa, Gerofi, Takagi, Shirasawa, Shimizu, Hori, Ishikawa, “Interface for Heterogeneous Kernels: A Framework to Enable Hybrid OS Designs Targeting High Performance Computing,” In IEEE International Conference on High Performance

Computing (HiPC), IEEE, 2014.
[11] Gerofi, Shimada, Hori, Takagi, Ishikawa, “CMCP: A Novel Page Replacement Policy for System Level Hierarchical Memory Management on Many-cores,” In Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing, ACM, 2014.
[12] Carretero, Blas, Singh, Isaila, Fahringer, Prodan, Bosilca, Lastovetsky, Symeonidou, Perez-Sanchez, Cecilia, “Optimizations to enhance sustainability of MPI applications,” EuroMPI/ASIA 2014.
[13] Bland, Bouteiller, Herault, Hursey, Bosilca, Dongarra, “An Evaluation of User-Level Failure Mitigation Support in MPI,” Computing, Springer, 2013, issn 0010-4885X, http://dx.doi.org/10.1007/s00607-013-0331-3.
[14] Sato, Fukazawa, Yoshinaga, Tsujita, Hori, Namiki, “A Hybrid Operating System for a Computing Node with Multi-Core and Many-Core Processors,” In International Journal Advanced Computer Science (IJACSci), volume 3, 2013.
[15] Bland, Bouteiller, Herault, Bosilca, Dongarra, “Post-failure recovery of MPI communication capability: Design and rationale,” International Journal of High Performance Computing Applications August 2013.
[16] Gerofi, Shimada, Hori, Ishikawa, “Partially Separated Page Tables for Efficient Operating System Assisted Hierarchical Memory Management on Heterogeneous Architectures,” In CCGRID, 2013.
[17] Hori, Kameyama, Tsujita, Namiki, Ishikawa, “An Efficient Kernel-Level Blocking MPI Implementation,” Chapter in Recent Advances in the Message Passing Interface, Springer Berlin Heidelberg, volume 7490, 2012.
[18] Yoshinaga, Tsujita, Hori, Sato, Namiki, Ishikawa, “Delegation-Based MPI Communications for a Hybrid Parallel Computer with Many-Core Architecture,” Recent Advances in the Message Passing Interface, Lecture Notes in Computer Science, Vol. 7490, Springer, Vienna, September 2012, pp. 47-56.

(2)その他の著作物(総説, 書籍など)

なし.

(3)国際学会発表及び主要な国内学会発表

① 招待講演 (国内会議 0 件, 国際会議 0 件)

なし.

② 口頭発表 (国内会議 48 件, 国際会議 10 件)

1. 発表者(所属), タイトル, 学会名, 場所, 月日

[19] 佐藤(農工大), 島田(日立), 吉永(理研), 辻田(理研), 堀(理研), 並木(農工大), “Multiple PVAS を用いた MapReduce におけるファイル I/O 処理”, 一般社団法人情報処理学会, volume 2016-HPC-153, 2016.
[20] 吉永(理研), 亀山(理研), 堀(理研), 石川(理研), “スライド手法による故障復帰方式の Tsubame2.5 を用いた性能評価”, In 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], 一般社団法人情報処理学会, volume 2016-HPC-153, 2016.
[21] 吉永(理研), 亀山(理研), 堀(理研), 石川(理研), “スライド手法によるスペアノードを利用した故障復帰方式の性能評価”, In 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], 一般社団法人情報処理学会, volume 2015, 2015.
[22] 島田(日立), 須藤(日立), 堀(理研), 石川(理研), “メニーコア環境向けタスクモデル PVAS を用いた MPI の Eager 通信高速化,” 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」第 150 回研究報告, vol.2015-HPC-150, No.41, pp.1-8 (2015-07-28).
[23] 佐藤(農工大), 島田(日立), 吉永(理研), 辻田(理研), 堀(理研), 並木(農工大), “メニーコアプロセッサにおけるPVASタスクモデルによるMapReduceアプリケーションの性能評価,” 情報処理学会「ハイパフォーマンスコンピューティング研究会」第 150 回研究報告, vol.2015-HPC-150, No.17, pp.1-5 (2015-07-28).
[24] 大川(筑波大), 堀(理研), 島田(理研), 池井(インテル), 佐藤(筑波大), “メニーコアプロセッサ向けプロセスモデルPVASのPGAS言語実行時ライブラリの設計,” In 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], 一般社団法人情報処理学会, volume 2015, 2015.
[25] Tsujita(理研), Hori(理研), Ishikawa(理研), “Locality-Aware Process Mapping for High Performance Collective MPI-IO on FEFS with Tofu Interconnect,” In Proceedings of EuroMPI/ASIA 2014 (Challenges in Data-Centric Computing), ACM, 2014.
[26] Bosilca(テネシー大), Bouteiller(テネシー大), Herault(テネシー大), Robert(テネシー大), Dongarra(テネシー大), “Assessing the Impact of ABFT and Checkpoint Composite Strategies,” IPDPSW, APDCM 2014, Phoenix, AZ, May, 2014.
[27] Tsujita(理研), Yoshinaga(理研), Hori(理研), Sato(農工大), Namiki(農工大), Ishikawa(理研), “Multithreaded Two-Phase I/O: Improving Collective MPI-IO Performance on a Lustre File System,” In Proceedings of PDP2014, Turin, February 12-14, IEEE CS, 2014.
[28] Sato(農工大), Fukazawa(農工大), Shimada(理研), Hori(理研), Ishikawa(理研), Namiki(農工大), “Design of Multiple PVAS on InfiniBand Cluster System Consisting of Many-core and Multi-core,” In Proceedings of the 21st European MPI Users’ Group Meeting, ACM, 2014.
[29] Shimada(理研), Hori(理研), Ishikawa(理研), Balaji(理研), “User-level Process towards Exascale Systems,” In 情報処理学会研究報告. 計算機アーキテクチャ研究会報告, 2014.
[30] 佐藤(農工大), 深沢(農工大), 島田(理研), 吉永(理研), 辻田(理研), 堀(理研), 石川(理研), 並木(農工大), “マルチコア・メニーコア混在型計算機における Multiple PVAS を用いた MPI Agent による MPI 通信の基礎評価,” 情報処理学会, Vol. 2014-HPC-144, No.17, 2014.
[31] 吉永(理研), 亀山(理研), 堀(理研), 石川(理研), “予備ノードを利用した故障後の実行継続手法の検討と評価,” In 情報処理学会研究報告, SIGHPC, 2014.
[32] 吉永(理研), 亀山(理研), 畑中(理研), 堀(理研), 石川(理研), “代替ノード利用手法による耐故障性実現に向けた通信性能の評価と検討,” In 情報処理学会研究報告. SIGHPC, 2014.
[33] 堀(理研), 亀山(理研), 並木(農工大), 石川(理研), “メニーコアクラスタにおけるジョブ間の性能干渉について,” In 情報処理学会研究報告. SIGHPC, 2014.
[34] 大野(NEC), 堀(理研), 石川(理研), “並列ファイルシステムに対するI/Oリクエスト調停機構の提案,” In 情報処理学会研究報告. SIGHPC, 2014.
[35] 辻田(理研), 堀(理研), 石川(理研), “FEFSにおけるストライピング処理を考慮した集団型MPI-IOの実装,” 情報処理学会研究報告. SIGHPC, 2014.
[36] 吉永(理研), 亀山(理研), 堀(理研), 石川(理研), “エクサスケールでの耐故障性実現に向けた代替ノード配置による通信性能の評価,” In 情報処理学会研究報告. SIGHPC, 一般社団法人情報処理学会, volume 2014, 2014.
[37] 辻田(理研), 堀(理研), 石川(理研), “集団型MPI-IOの高速化に向けたI/Oリクエスト発行方式の最適化,” 情報処理学会研究報告 2014-HPC-146, No. 17 2014.
[38] 島田(理研), 堀(理研), 石川(理研), “新しいタスクモデルによる MPI ノード内通信の高性能化,” 情報処理学会研究報告, Vol. 2014-HPC-145, No. 6, 2014.

[39] 佐藤(農工大), 島田(理研), 吉永(理研), 辻田(理研), 堀(理研), 石川(理研), 並木(農工大), “Xeon Phi 搭載計算機における DMA・MMIO 併用型 CPU 間データ通信機構,” SWoPP 新潟 2014, 情処研報 システムソフトウェアとオペレーティング・システム (OS), 2014-OS-130(17), pp.1-7 (2014-07-21).
[40] Tsujita(理研), Yoshinaga(理研), Hori(理研), Sato(農工大), Namiki(農工大), Ishikawa(理研), “Improving Parallel I/O Performance Using Multithreaded Two-Phase I/O with Processor Affinity Management,” In PPAM 2013, Revised Selected Papers, Part I, Lecture Notes in Computer Science, Vol. 8384, Springer, 2014, pp. 714-723.
[41] Yoshinaga(近畿大), Tsujita(近畿大), Hori(理研), Sato(農工大), Namiki(農工大), Ishikawa(理研), “A Delegation Mechanism on Many-Core Oriented Hybrid Parallel Computers for Scalability of Communicators and Communications in MPI,” In Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, IEEE Computer Society, 2013.
[42] Shimada(理研), Gerofi(理研), Hori(理研), Ishikawa(理研), “Proposing a new task model towards many-core architecture,” In Proceedings of the First International Workshop on Many-core Embedded Systems, ACM, 2013. 深沢(農工大), 佐藤(農工大), 吉永(理研), 辻田(理研), 島田(理研), 堀(理研), 並木(農工大), “メニーコア混在型並列計算機向け大域仮想アドレス空間モデル Multiple PVAS の提案,” In 情報処理学会第 141 回 SIGHPC, 2013.
[43] 堀(理研), 島田(理研), 並木(農工大), 佐藤(農工大), 深沢(農工大), 辻田(理研), 石川(理研), “メニーコア用 Agent プログラミング環境の提案,” In IPSJ SIG Notes, Information Processing Society of Japan (IPSJ), volume 2013, 2013.
[44] Sato(農工大), Fukazawa(農工大), Nagamine(農工大), Sakamoto(農工大), Namiki(農工大), Yoshinaga(理研), Tsujita(理研), Hori(理研), Ishikawa(理研), “A design of hybrid operating system for a parallel computer with multi-core and many-core processors,” In Proceedings of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers, ACM, 2012.
[45] Shimada(理研), Gerofi(理研), Hori(理研), Ishikawa(理研), “PGAS Intra-node Communication towards Many-Core Architecture,” In PGAS 2012: 6th Conference on Partitioned Global Address Space Programing Model, 2012.
[46] Ohno(理研), Hori(理研), Ishikawa(理研), “File Composition Technique to Improve the Performance of Accessing a Number of Small Files,”In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, volume I, 2012.
[47] 島田(理研), ゲローフィ(理研), 堀(理研), 石川(理研), “メニーコア OS 向け新プロセスモデルの提案,” In IPSJ SIG Notes, 情報処理学会, 2012.
[48] 深沢(農工大), 長嶺(農工大), 坂本(農工大), 佐藤(農工大), 吉永(近畿大), 辻田(近畿大), 堀(理研), 石川(理研), 並木(農工大), “A design of I/O Library on the Light-weight OS for a Multi/Many-core Parallel Computer,” IPSJ SIGHPC, volume 2012, 2012.
[49] 深沢(農工大), 佐藤(農工大), 長嶺(農工大), 坂本(農工大), 吉永(近畿大), 辻田(近畿大), 堀(理研), 石川(理研), 並木(農工大), “Remote Resource Management on a Hybrid Operating System for a Multi/Many-core Parallel Computer,” In IPSJ SIGHPC, 2012.
[50] 吉永(近畿大), 六車(近畿大), 辻田(近畿大), 堀(理研), 並木(農工大), 佐藤(農工大), 下沢(東大), 石川(理研), “メニーコア混在型並列計算機における委託機構を用いた MPI 通信基盤,” In 情報処理学会研究報告. SIGHPC, 2012.
[51] 今田(富士通), 山中(富士通), 堀(理研), 石川(理研), “ハードウェアプリフェッチ機構の活用によるステンシル計算の性能改善手法,” In 情報処理学会研究報告.

SIGHPC, 2012.

[52] 山本(理研), “並列ファイルシステムのためのメタデータ管理手法,” 第一回・若手女性研究者向け並列コンピュータ＆ストレージ利用発表会, 2012.2.27.

[53] 大野(理研), “大規模並列ストレージ向けファイル集約ライブラリの開発と評価,” 第一回・若手女性研究者向け並列コンピュータ＆ストレージ利用発表会, 2012.2.28.

[54] 深沢(農工大), 長嶺(農工大), 佐藤(農工大), 並木(農工大), “マルチコア・メニーコア混在型計算機における資源管理コア向け OS 間連携機構の試作,” 情報処理学会第 74 回全国大会講演論文集, 5J-1, 2012.03.08.

[55] 長嶺(農工大), 坂本(農工大), 佐藤(農工大), 下沢(農工大), 吉永(近畿大), 辻田(近畿大), 堀(理研), 石川(理研), 並木, “メニーコア混在型並列計算機におけるスレッド管理方式,” In 情報処理学会研究報告. SIGHPC, 2011.

[56] 澤田(東大), 辻田(近畿大), 並木(農工大), 堀(理研), 石川(理研), “OS 開発のためのメニーコアハードウェアシミュレータの設計と実装,” In 情報処理学会研究報告. SIGHPC, 2011.

[57] 大野(理研), 堀(理研), 石川(理研), “並列ジョブのファイル I/O をひとつのファイルに集約する方式の提案と予備評価,” In 情報処理学会研究報告. SIGHPC, 2011.

[58] 堀(理研), 亀山(理研), 並木(農工大), 辻田(近畿大), 石川(理研), “ハードウェア同期機構を用いた省電力 MPI の実装と評価,” In 情報処理学会研究報告.SIGHPC, 2011.

[59] 堀(理研), 山本(理研), 大野(理研), 今田(理研), 亀山(理研), 石川(理研), “ハードウェア同期機構を用いた超軽量スレッドライブラリ,” In 情報処理学会研究報告. SIGHPC, 2011.

[60] 吉永(近畿大), 六車(近畿大), 辻田(近畿大), 堀(理研), 並木(農工大), 佐藤(農工大), 下沢(東大), 澤田(東大), 石川(理研), “メニーコア混在型並列計算機における MPI 通信基盤の提案,” In 情報処理学会研究報告. SIGHPC, 2011.

[61] 六車(近畿大), 吉永(近畿大), 辻田(近畿大), 堀(理研), 並木(農工大), 石川(理研), “パイプライン型 Two-Phase I/O の Lustre における性能評価,” In 情報処理学会研究報告. SIGHPC, 2011.

[62] 六車(近畿大), 辻田(近畿大), 堀(理研), 並木(農工大), “Two-Phase I/O の高速化に関する一検討,” In 情報処理学会研究報告, SIGHPC, 2011.

[63] 佐藤(農工大), 辻田, 堀(理研), 並木(農工大), “マルチコア・メニーコア混在型並列計算機向け OS の構想,” In 情報処理学会研究報告. SIGOS, 2011.

[64] 下沢(東大), 石川(理研), 堀(理研), 並木(農工大), 辻田(近畿大), “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,” In 情報処理学会研究報告. SIGOS, 2011.

[65] 下沢(東大), 堀(理研), 石川(理研), “メニーコア環境におけるキャッシュアウェア・オペレーティングシステムに向けて,” In 情報処理学会研究報告. SIGOS, 2011.

[66] 長嶺(農工大), 坂本(農工大), 佐藤(農工大), 下沢(東大), 吉永(近畿大), 辻田(近畿大), 堀(理研), 石川(理研), 並木(農工大), “メニーコア混在型並列計算機におけるスレッド管理方式,” Hokke2011, 情報処理学会研究報告 Vol.2011-HPC-132, 2011.11.28.

③ ポスター発表　　(国内会議 2 件, 国際会議 12 件)

1. 発表者(所属), タイトル, 学会名, 場所, 月日

[67] Sato(農工大), Yoshinaga(理研), Tsujita(理研), Hori(理研), Namiki(農工大), “Memory Management for Memory-based Inter-Task Communication on the Hybrid Operating System Node,” APsys 2015.

[68] Shimada(理研), Hori(理研), Ishikawa(理研), “Eliminating Costs for Crossing Process

Boundary from MPI Intra-node Communication," In Proceedings of the 21st European MPI Users' Group Meeting, ACM, 2014.

[69] Shimada(理研), Hori(理研), Ishikawa(理研), "Eliminating Costs for Crossing Process Boundary from MPI Intra-node Communication," EuroMPI/ASIA, 2014, Kyoto, Japan.

[70] Sato(農工大), Fukazawa(農工大), Shimada(理研), Yoshinaga(理研), Tsujita(理研), Hori(理研), Namiki(農工大), "Multiple PVAS: Parallel Task Model for the Hybrid Architecture Consisting of Many-Core & Multi-Core," HPC in Asia Poster Session, Leipzig, Germany, June 22-26, 2014.

[71] 佐藤(農工大), 深沢(農工大), 島田(理研), 吉永(理研), 辻田(理研), 堀(理研), 並木(農工大), "メニーコア混在型並列計算機向け大域仮想アドレス空間モデル「Multiple PVAS」のためのメモリ管理方式," 情報処理学会コンピュータシステムシンポジウム(ComSys 2013), ポスター・デモセッション, 芝浦工業大学, 2013-12-04.

[72] 山本(理研), 大野(理研), 堀(理研), 石川(理研), "並列ファイルシステムのためのハッシュを基にしたメタデータ管理," In ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集, volume 2012, 2012.

[73] Hori(理研), Shimada(理研), Ishikawa(理研), "Partitioned Virtual Address Space (PVAS): A New Process Model for Many-Core Operating Systems," HPC in Asia Workshop in ISC'12, Hamburg, June, 2012.

[74] Yoshinaga(近畿大), Tsujita(近畿大), Hori(理研), Sato(農工大), Namiki(農工大), Ishikawa(理研), "Delegating MPI Communications from Many-Core CPUs to Multi-Core CPUs for High Throughput," HPC in Asia Workshop in ISC'12, Hamburg, June, 2012.

[75] Hori(理研), Ishikawa(理研), "Kernel-Level Blocking MPI," VECPAR'12, Kobe, July 2012.

[76] Ohno(理研), Hori(理研), Ishikawa(理研), "File Composition Technique for Improving Access Performance of a Number of Small Files," VECPAR'12, Kobe, July 2012.

[77] Gerofi(理研), Hori(理研), Ishikawa(理研), "clone_n(): Parallel Thread Creation for Upcoming Many-Core Architectures," IEEE CLUSTER'12, Shanghai, Sept., 2012.

[78] Gerofi(理研), Shimada(理研), Hori(理研), Ishikawa(理研), "Towards Operating System Assisted Hierarchical Memory Management for Heterogeneous Architectures," SC12, Salt Lake City, Nov., 2012.

[79] Yamamoto(理研), Hori(理研), Ishikawa(理研), "Distributed Metadata Management for Exascale Parallel File System," SC12, Salt Lake City, Nov., 2012.

[80] Hori(理研), Ishikawa(理研), "Poster: MINT: a fast and green synchronization technique," In Proceedings of the 2011 companion on High Performance Computing Networking, Storage and Analysis Companion (SC11), ACM, 2011.

(4)知財出願

なし.

(5)受賞・報道等

① 受賞

[1] Sato, Fukazawa, Shimada, Yoshinaga, Tsujita, Hori, Namiki, "HPC in Asia Best Poster Award, Multiple PVAS: Parallel Task Model for the Hybrid Architecture Consisting of Many-Core & Multi-Core," HPC in Asia Poster Session, Leipzig, Germany, June, 2014.

[2] 深沢, 佐藤, 吉永, 辻田, 島田, 堀, 並木, 情報処理学会第 76 回全国大会 学生奨励賞, 深沢:マルチコア・メニーコア混在型計算機による高性能計算向けアプリケーション実行基盤「Multiple PVAS」の性能評価 2014.

[3] 島田, グローフィ, 堀, 石川, 2013 年度コンピュータサイエンス領域奨励, 島田:メニーコア OS 向け新プロセスモデルの提案, 2013.

② マスコミ(新聞・TV等)報道
なし.

④ その他
なし.

(6)成果展開事例
特になし.

①実用化に向けての展開

- 研究開発された PVAS および M-PVAS に関しては, H25 年 11 月より, 理研のホームページよりダウンロード可能となっている
(URL: http://www-sys-aics.riken.jp/releasedsoftware/software/pvas-1.0.0-beta.tar.gz)

②社会還元的な展開活動

- 研究成果は, 理研のホームページに公開されている(URL: http://www-sys-aics.riken.jp).

§5 研究期間中の活動

5. 1 主なワークショップ, シンポジウム, アウトリーチ等の活動

年月日	名称	場所	参加人数	概要
H24 年 5 月 24 日	1st Joint CREST WS Toudai and RIKEN	理研 AICS 6F 講堂	25 人	東大中島 CREST チームとの合同 WS
H25 年 10 月 28 日	中島チームとの合同 WS	東大	23 人	両チームの研究内容の発表
H26 年 3 月 14 日	日米システムソフト共同研究について	米アルゴンヌ国立研究所	17 人	日本側と米国側で共同研究の可能性について検討
H26 年 11 月 19 日	FT Open Discussion	SC14 PCCC ブース	5 人	FT に関する最新研究動向の討論
H27 年 10 月 29-30 日	LENS 国際ワークショップ	秋葉原コンベンションホール	2 日延べ 100 人	南里チーム, 朴チームとの合同国際 WS
H27 年 11 月 17 日	FT Open Discussion	SC15 PCCC ブース	5 人	FT に関する最新研究動向の討論

§6 最後に

本プロジェクトの公募の直前に理化学研究所計算科学機構(AICS)が開所し, 理研の堀チームのほとんどのメンバーが, 本プロジェクトで初めて顔をあわせることとなった. そのため, 本プロジェクトにおける研究はほとんどゼロの状態から始めたこともあり, 研究成果として主要な学会に十分な数を出せるレベルにまで至らなかった. 一方, 本研究により将来に向けた論文のネタとしては十分に提供できたと考える. そのため, 今後の研究の展開に期待したい.