Vol.2013-HPC-141 No.8
2013/9/30

次世代高性能計算機システムのための システムソフトウェア実現にむけて

住元 真司[1]　小田和 友仁[1]　宇野　俊司[1]　石川 裕[2]

概要：Exa-FLOPS クラスをターゲットにした HPC システム開発が欧米で始まっている。日本でも昨年度より Exascale をターゲットにした HPC システムの検討が始まった。本論文では、現在検討している次世代高性能計算機システムの実現に向けたシステムソフトウェアの課題とその解決のアプローチについて述べる。特に、Co-design によるアプリケーション毎の開発環境と汎用システム利用の環境をどう作るべきかについて議論する。

Towards Realizing System Software for the Next Generation High-Performance Computer System

Abstract: Developing Exa-FLOPS class HPC systems started in the world. Japan has also started the investigation for development of the exascale high performance system. This paper discusses the issues on realizing the system software of an exascale system and approach to resolving those issues, espacially how co-designed application specific OS features are provided in a general purpose OS.

1. はじめに

2011 年 11 月に「京」[1] が 10 PFLOPS の演算性能を達成した後も各国のシステムは競い合う状況が続いており、2012 年 6 月の Sequia[2](米国)、2012 年 11 月の Titan[3](米国)、そして 2013 年 6 月には Tianhe-2(中国 NUDT[4],33.86 PFLOPS) が TOP500[5] で Rank 1 を獲得し、その勢いはとまることがない。この流れは 1EFLOPS を実現する Exascale システム開発に続いており、米国 (UHPC[6])、EU(EESI[7])、中国が開発を進めている。日本でも欧米に遅れること数年、2010 年より戦略的高性能計算システム開発に関するワークショップ (SDHPC)[8]、2012 年に 文科省のもと「今後の HPCI 計画推進のあり方に関する検討ワーキンググループ」[9] において検討が進められている。

報告書 [10] では、1 EFLOPS の消費電力目標を 20MW としている。この電力対性能実現にはハードウェアの技術革新が大きな比重を占めることになる。しかし、ソフトウェアについても、消費電力を抑制しアプリケーションの実効性能を高めるため、OS カーネルからアプリケーションまでの協調設計 (Co-design) による極限までの最適化が必要である。このため、我々はこのベースとなるシステムソフトウェアの開発を進めている。[11]

高性能計算システム向けのハードウェアとアプリケーションについては、これまでも各国が競争を続けている。しかし、システムソフトウェアについては、これまでも欧米の開発機関を中心としたオープンソースを主体に構成されている。このため、Exascale システムでも、各国が協調して開発を進めるべく 2008 年より 2012 年まで International Exascale Software Project(IESP)[10], [12] が進められ、現在は、日本の文科省と米国の DoE を主体とした協調開発体制が立ち上がりつつある。

本論文では、現在検討している Exascale システム実現を視野に入れた次世代高性能計算機において、このシステムの実現に向けたシステムソフトウェアの課題とその解決のアプローチについて述べる。特に、Co-design による最適化手段を継続して確保しながら、より多くのアプリケーションが稼働するシステムソフトウェアの実現が重要な課題となるため、この環境をどう作るべきかについて議論し明らかにする。

[1] 富士通
川崎市中原区上小田中 4-1-1
[2] 東京大学/理化学研究所 AICS
東京都文京区弥生 2-11-16

2. 次世代高性能計算機の目標

現在検討している次世代高性能計算機においては、Exascale システム実現を念頭に検討された戦略的高性能計算システム開発に関するワークショップ (SDHPC)[8] が 2010 年より開催され、2012 年 3 月に「HPCI 技術ロードマップ白書」[13] と「計算科学研究ロードマップ白書」[14] が発行された。SDHPC にて設定された技術と計算科学研究のロードマップにおいては、次のことが言及されている。

「計算科学研究ロードマップ白書」では、

(1) 2018-2020 年ごろの社会ニーズと期待されうるサイエンスに基づく HPC の必要性
(2) サイエンスロードマップを実現するアプリケーションの計算手法にマッチした複数アーキテクチャの想定

と示されており、これは、きちんとサイエンスドリブンで社会に貢献できることを前提に次世代高性能計算機を作るべきという方針が示されている。

また、**「HPCI 技術ロードマップ白書」**では、

(1) 2018 年ごろまでの技術トレンドに基づく技術進展ではサイエンスロードマップ実現は困難であるため、これを越える技術を持つ計算機システムが必要
(2) 消費電力 (20-30MW)、設置面積 (2000m2) の制約を満たすシステム
(3) o(100k)-o(1M) コアの超並列性
(4) 数値演算ライブラリから計算機アーキテクチャまでの Co-design が必要

と示され、現在の「京」設置地点の利用を想定して、ハードウェアとソフトウェアを Co-design して進めるべきと示している。

さらに「HPCI 技術ロードマップ白書」では、(1) 汎用（従来型）、(2) 容量・帯域重視型、(3) 演算重視・メモリ容量削減型（帯域・演算重視型）の 3 つのアーキテクチャについてそれぞれについて既存の技術トレンドを越える改善が必要としている。

東京大学を中心に実施している文部科学省「将来の HPCI システムのあり方の調査研究」の課題では、この中の (1) 汎用（従来型）のシステムを実現する次世代高性能計算機を目標として検討を進めている。本論文では、このシステム上でのシステムソフトウェアについて論じる。

3. 次世代高性能計算機向けのシステムソフトウェアの要件

第 2 章に述べたように、次世代高性能計算機は、目標とするアプリケーション性能を設定電力以下で実行する必要がある。さらには、目標とする次世代高性能計算機は汎用（従来型）のアプローチであるため「京」の利用環境ならびに一般の HPC システムの利用環境とは大きく異ならないことが重要である。

表 1 想定ハードウェア諸元と京コンピュータ

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

本章では、次世代高性能計算機向けのシステムソフトウェアの要件を、検討中の計算機の想定ハードウェア諸元を述べた後、システムソフトウェアによる省電力化、高いアプリケーション性能実現、高い汎用性実現の 3 つの点から要件について整理する。

3.1 検討中の計算機の想定ハードウェア諸元

次世代高性能計算機の検討において、検討中の計算機の想定ハードウェア諸元 [11] を表 1 に示す。

表 1 より、ノード数は「京」と同等レベルである。このことから、システムソフトウェアに求められる基本的なスケーラビリティについては「京」で開発されたもので対応可能であることがわかる。

一方、システムに搭載される Core 数は 5,120k 個以上と 5M 個を越えることになり、「京」の 8 倍以上に増加するため、アプリケーションによっては、このプロセス数増加によるシステムソフトウェアへの影響を精査する必要がある。

3.2 システムソフトウェアによる省電力化への要件

第 2 章の議論より、システムの達成目標は、Exascale を実現する最終目標である 1 EFLOPS の消費電力目標を 20-30MW としている。しかし、この実現にはハードウェアの技術革新が大きな比重を占める。従って、システムソフトウェアにおいては、可能な限り消費電力を抑制しアプリケーションの実効性能を極限まで高めることが求められる。このために次の要件を満たす必要がある。

アプリケーションの実効効率の向上： 処理を高速化（単純化）して実行時間を短縮化する。
データ移動の削減： 大きな電力が必要なデータ移動を最小限に抑制する。
その他、間接的な処理の削減： 可能な限り、アプリケーション実行に不要な処理は削減する。

アプリケーションの実効効率の向上、同じ処理を高速化する点については、アプリケーションとシステムソフトウェアそれぞれに適応可能であるが、システムソフトウェ

アについては、処理の単純化も視野に入れるべきある。なぜなら、現在の OS カーネルを含むシステムソフトウェアスタックは、種々の抽象化、仮想化が導入されている。システムソフトウェアをアプリケーション含めて見直し、可能な限り抽象化、仮想化を排除して単純化し、実行オーバヘッドを極限まで削減することが必要である。

データ移動の削減については、特に、大きなデータ移動が伴うインターコネクトとストレージについては、データの局所性を徹底したアプリケーション実行が必要である。

その他、間接的な処理の削減については、無駄な処理の排除により不要な電力を削減することが必要になる。

3.3 高いアプリケーションの性能実現の要件

高いアプリケーション性能の実現には、次に述べるように、それぞれの単体性能を高める他、性能阻害要因の排除と省メモリ性の実現が必須となる。

個別性能の高速化： 演算性能、通信性能、I/O 性能の向上、更にこれらの処理のオーバラップ並行処理化

性能阻害要因の排除： システムソフト（OS ノイズ）や複数ジョブ実行間の外乱の排除、関数処理の実行時間バラツキの排除、

3.4 高い汎用性実現の要件

高い汎用性実現の要件としては、アプリケーションの移植性、高い運用性、省メモリ性の確保が重要になる。

アプリケーション移植性: これまでのアプリケーション資産が活用できることが重要である。このためには、移植に必要な言語、ライブラリ、開発ツール群の提供されている他、従来のハードウェア環境との性能互換性が確保されていることが望ましい。

高い運用性: 様々なアプリケーションの実行環境を多くのユーザが違和感なく使えることが重要である。

省メモリ性の確保： より多くのメモリ量を確保することにより、より大きな問題を実行可能にする。

4. 「京」のシステムソフトウェアの概要

「京」のシステムソフトウェアは、標準的なオープンソースをベースに「京」向けの機能拡張が施されている。本章では、第 3 章で述べた要件について、「京」のシステムソフトウェアを例に既存のシステムソフトウェアがどのように対応しているかを示す。

4.1 高いアプリケーション性能の実現

高いアプリケーション性能の実現には、演算性能、通信性能、I/O 性能の個別性能を高め、さらに、性能阻害要因を極限まで排除する必要がある。

一般に、並列プロセス数が増加するほどスケーラビリティは低下するため、高いアプリケーション性能を実現するには、並列プロセス数は少ない方が良い。このため、「京」のシステム設計においては、並列プロセス数を減らすために、スレッド並列処理とプロセス並列が共存する Hybrid Parallel 方式を推奨している。スレッド並列処理は、OpenMP と自動並列化コンパイラで対応し、プロセス並列は MPI 通信ライブラリで対応する。

4.1.1 個別性能の高速化

個別性能の高速化についての概要を述べる。詳細は各参考文献を参照されたい。

演算性能: セクタキャッシュ、SIMD 演算、loop 並列最適化をサポートしたコンパイラと最適化された算術演算ライブラリ、Large Page によるメモリアクセス性能の高速化の実現 [15]

通信性能: Tofu インタコネクトのハードウェア性能を最大限に引き出す MPI 通信（1 対 1 通信、集団通信）の実現 [16], [17]

I/O 性能: オープンソースのクラスタファイルシステムである Lustre ベースで大規模な並列 I/O 環境を実現する FEFS ファイルシステム [18] の実現、「京」全体システムで IOR ベンチマークで 1.5TB/s のファイル転送性能を達成 [19]

4.1.2 性能阻害要因の排除

「京」が持つ 80,000 ノード規模のシステムで、高いアプリケーション性能を実現するには、性能阻害要因の排除は解決必須の課題である。性能阻害要因としては、OS ノイズ (OS ジッタ)、ファイル I/O への外乱、インタコネクト通信への外乱がある。それぞれの課題と対応について述べる。

OS ノイズ: 「京」の OS は Linux ベースに開発されている。「京」ではノイズ対策の目標として 200ms 内に OS 処理を 50us 以内に抑える（ノイズ率 2.5E-04）ことを目標に詳細に OS カーネルを解析して実現している。一般的な PC クラスタのノイズ率は 0.2E-02 程度であるため、ノイズ率は 1/100 に抑えられている。この他、外乱を抑制するために次の施策を実施している。

- デーモン処理時間の削減：不要なデーモン削除、必要なデーモンについても細かくノイズ時間を精査解析し削減
- コアバインド: デーモンが重なり所定の許容時間を越えないように分散
- 同期スケジューリング：システム処理を同期実行化して、連鎖的に OS ノイズが増加するのを防止

ファイル I/O への外乱: ストレージアーキテクチャならびに、システムソフト（主としてジョブスケジューラ）の連携で実現している。

- ファイルシステム構成を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に分離し、ユーザプログラミング環境からローカルファイルシステムへのアクセスを原則禁止する。
- ジョブスケジューリング時にストレージを直下のストレージに可能な限り占有割当することにより、複数のジョブ間でのストレージアクセスへの影響を最小限に抑制する。
- ステージング処理により、グローバルファイルシステムとローカルファイルシステム間のデータ転送の外乱を最小限に抑制

インタコネクトへの外乱: 「京」のインターコネクトである Tofu インターコネクトは直接網であるため、ジョブ単位に直方体形状を与えることによりジョブ間のインターコネクト通信の外乱を抑制している。

4.2 高い汎用性の実現

本節では、高い汎用性実現について、アプリケーションの移植性、高い運用性、高い省メモリ性における「京」の取り組みについてそれぞれ述べる。

4.2.1 アプリケーションの移植性

HPC のシステムソフトウェアを考える上では、数多く配布されているオープンソースのアプリケーションやツール、ライブラリの移植性を考慮することが重要である。これらは主に PC クラスタ上で開発され、配布されている物が多いため、アプリケーション移植性を考える上で、PC クラスタと実行環境の互換性を確保することが重要である。

このため、「京」のシステムソフトウェアについても、OS はオープンソースが多く稼働している Linux をベースに、数多くのオープンソースが稼働可能なように各種ツール、ライブラリを移植している。

4.2.2 高い運用性

高い運用性を確保するために、「京」向けのシステムソフトウェアでは次の対策を行っている [20]。本節ではその概要を述べる。

システム管理機能: システム管理者に対し、シングルシステムイメージでの運用管理機能を提供する。
- システムの構成情報を管理し、構成要素である計算ノード、I/O ノードなどの稼働状況を監視
- 異常発生時にシステムの可用性を提供
- システムへのソフトウェアのインストール、ソフトウェア更新等のソフトウェア構成管理機能を提供

ジョブスケジューラ: 複数の利用者に対し、バッチジョブの実行環境を提供
- 80,000 ノード規模に対応したジョブ実行環境を提供
- システム稼働率向上のための施策：Back-fill、割り当て形状を 3 次元に回転割当するスケジューリングの採用

4.2.3 高い省メモリ性

高い省メモリ性を実現するために、「京」のシステムソフトウェアでは、システムソフトウェア全体の使用メモリ量を「京」の実メモリ量 (16GB) の 10%未満に抑えることを目標に開発された。この目標をベースに各コンポーネント毎にメモリ使用量を削減している。しかし、一般的にはメモリ使用量を減らす副作用として性能劣化する場合があるため、ケースバイケースで対処している。

ここでは、特に計算ノード数に比例してメモリ使用量が必要なファイルシステムと MPI 通信ライブラリについて述べる。

ファイルシステムの省メモリ化: ベースとしている Lustre ファイルシステムは起動時に最大構成時のメモリを確保する仕様であったため、これを改善した他、Linux のバッファキャッシュ制御、ストレージ構成の最適化により、メモリ使用量目標を満たしている [19]。

MPI 通信ライブラリの省メモリ化: メモリ使用量の多い通信バッファのメモリ量の削減は以下の 2 つの対策 [16] の他、管理メモリ向けのメモリ使用量については、開発のベースとなった Open MPI のコードを精査して必要な構造体のメモリ使用量を削減することによりメモリ使用量目標を満たしている。
- RDMA により省メモリを実現した通信プロトコル
- 高性能プロトコルと省メモリプロトコルを実装し高性能プロトコルの宛先数の設定により使用メモリ量の上限を設定

5. 次世代高性能計算機のシステムソフトウェアの目標と課題

本章では、次世代高性能計算機向けのシステムソフトウェアの目標と課題について述べる。システムソフトウェアが稼働する次世代高性能計算機の想定としては、検討中の計算機の想定ハードウェア諸元 (表 1) をベースに以下を念頭に検討する。
- 想定 Core 数は 64 以上と many core 化が一層進む、Core の動作周波数は低下
- 「京」システムと同等規模の接地面積と使用電力
- ノード数は一桁程度の増加を想定
- 単体ノード性能は 23-39 倍に比べ、搭載メモリ量は 3.2-7.9 倍と相対的にメモリ量が少ない

5.1 目標

第 3 章で述べた要件より、次世代高性能計算機向けのシステムソフトウェアの目標を次のように設定する。

高いアプリケーション性能を導き出す: 京の 100 倍の性能達成を効果的に支える

Co-design を継続実施可能: OS カーネルからアプリケー

ションまでの Co-design による最適化を妨げない

汎用システムとして利用可能: 「京」で稼働中のものを含め、多くのアプリケーションが実行可能

5.2 課題と議論

本節では、第 5.1 節で述べた目標を実現するために何が課題でどのような解決手法があるのかを議論する。

5.2.1 高いアプリケーション性能を導き出す

高いアプリケーション性能を導き出すためにシステムソフトウエア処理は、より高度にシステムアーキテクチャに適応し、低遅延処理化かつスケーラビリティを妨げないことが求められる。

低遅延処理化については、一層進む Many Core 化と消費電力を抑えるために Core の動作周波数向上は難しい。より低遅延処理の実現には、システムソフトウェア処理の簡略化に加え並列処理による遅延削減を考える必要がある。

- 簡略化： 既存の OS カーネルと基本ライブラリを含めオーバヘッドの大きな処理を調査し削減する必要がある。極限までシステムソフトウェアの低遅延化を実現した OS として、IBM の Blue Gene/L 向けに開発された CNK[21] がある。CNK は最小限度の OS 機能しか持たないため、高いアプリケーション性能を実現可能であるが、限定した機能しか提供されないため、実行できるアプリケーションに制限がある。
- 並列処理化： OS カーネルの処理毎に適用可能性を考える必要があるが、メモリコピー処理、探索処理、通信プロトコル処理などが対象。

スケーラビリティを妨げない処理の実現については、OS ノイズの更なる削減の他、Many Core 特有のスケーラビリティ低下を抑制する必要がある。

OS ノイズの削減： 「京」のカーネルについては、ほぼ最小レベルまで OS ノイズの削減を実施している。処理の簡略化と現在周期的に実施しているハードウェアによるタイマー処理の改良（長周期化、停止）を含めて考慮する必要がある。また、IBM Blue Gene/Q で導入された OS Core のようにシステムソフトウェア専用の Core を設定し処理実行することも考えられる。

Mnay Core 特有のスケーラビリティ低下対策： Core が増加することにより共有資源へのアクセスが増加するため共有資源の局所化が重要になる。また、メモリへのアクセス遅延も長く均一でないため、キャッシュ上の有効なデータが予期せず追い出されないようにする必要がある。

5.2.2 Co-design を継続実施可能

Exascale システムの実現には、協調設計 (Co-design) による極限までのシステム最適化が求められる [10]。ハードウェアアーキテクチャとソフトウェア間の Co-design については、ハードウェア決定時にある程度手法が決定可能であるが、ソフトウェア内での Co-design はアプリケーション毎に継続して行う必要がある。この取り組みは一過性のものでなく Exascale システム以降のシステムでも継続し実践していく必要がある。このため、想定システムが実運用に入った後もソフトウェア内での Co-design を継続的に実施していく必要があると考えている。

一方、「京」を含め既存システムは、基本的に一つの OS カーネル上のシステムソフトウェア実行が基本であるため、OS カーネルを含めた Co-design の実現は容易ではない。システムにおける Co-design では、ハードウェア構成を意識し最適な実行オブジェクトを生成するため、あらゆる箇所に修正が入ることが想定される。しかし、現状のシステムソフトウェアはアプリケーション毎にカーネルまで入れ替えられるようになっていない。実運用時に、柔軟にシステムソフトウェアを部分的に入れ替え可能なシステムソフトウェアが、今後 Co-design を継続して実施していくために必要である。

5.2.3 汎用システムとして利用可能:

高い汎用性システムを実現するためには、アプリケーションの移植性、高い運用性、高い省メモリ性の実現が重要である。これまで提供している機能をさらに改善していくことが重要である。

アプリケーションの移植性: PC クラスタとの実行環境の互換性確保、「京」で稼働しているアプリケーションの動作、これから開発され普及するソフトウェアの稼働

高い運用性: 「京」の運用ソフトウェア機能の継続した改善

高い省メモリ性: 計算ノード数に比例してメモリ使用量が必要なファイルシステムと MPI 通信ライブラリの更なる省メモリ性の改善

6. 次世代高性能計算機のシステムソフトウェアのアプローチと基本構成

第 5 章で述べた目標を満たすための課題は、相互に背反する条件をどう解決するかである。特に、汎用システム上で Co-design を実現するために、汎用的な実行環境を提供しながら、Co-design により開発されたアプリケーション専用システムソフトウェアをシステム運用中にアプリケーション実行とともに稼働させることが理想である。

一般にアプリケーション毎に適した環境を OS から提供する方法としては、provisioning による方法、仮想マシンにより実現する方法がある。

provisioning による方法は OS が起動する単位での入れ替えになる。このため入れ替え単位が大きくなる他、この手法は直接網を用いたシステムにおいては使えない。なぜなら、OS カーネルのリブート時にルータをリセットする

と通過中のパケットが廃棄され実行に影響が出るからである。

ルータをリセットすること無く、OS カーネルを入れ替えるには、仮想マシンを使う手法がある。しかし、フルセット OS カーネルの仮想マシンでの実行はオーバヘッドが多く利用できない。より、軽量に OS カーネルを動作させる仕組みが必要である。

以上のように、Co-design により開発された専用システムソフトウェアをシステム運用中にどのように稼働させることが大きな課題である。これを一つのシステムソフトウェアで実現することは容易ではない。本章では、この課題にどのようにしてアプローチするかについて延べ、現在開発中のシステムソフトウェアの基本構成について述べる。

6.1 課題解決のアプローチ

ここで、我々のターゲットが Many Core プロセッサであることを考えると、必ずしも仮想マシンを実現する必要はない。例えば、ある Core にスーパバイザ用 OS カーネルを稼働させ、他の Core を異なる OS カーネルを Co-Kernel として専用に割り当て動作させればよい。この場合、Co-Kernel を実行する Core はフルセットの OS 機能を持つ必要はなく、最小限の機能を持つことにより軽量化することが可能である。また、Co-Kernel が持たないシステムコール機能はデバイス制御を行うフルセット OS 機能を持つ OS カーネルに Delegation することで機能補完が可能である。

我々は以上の考え方による Many Core 向け Light-weight Kernel を McKernel 機構 [11], [22] と呼び、この考え方の基づく OS カーネルを次世代高性能計算機システム向けの OS カーネルとして採用することにした。

6.2 McKernel の基本構成

図 1 に次世代高性能計算機システム向け OS カーネルの基本構成を示す。Many Core の Core の一部でフル Linux を稼働させ (Host Linux)、残る Core を計算 Core として定義し、計算 Core 上で McKernel を動作させる。Host Linux と McKnernel は IHK と呼ばれる Kernel 間通信機構により結合され、McKernel の生成や制御、システムコールの Delegation 処理を実現する [22]。

アプリケーションプログラムとの API としては glibc の API を提供し、システムコールは delegation 処理により、Host Linux 上での処理が基本である。McKernel は用途毎、アプリケーション毎に必要に応じて開発されることを想定している。この場合、アプリケーション実行上遅くなるシステムコール処理については、McKernel 内で処理することにより高速化することが可能である。また、Host Linux 上では既存の運用管理系のシステムソフトウェアが稼働し、McKernel 上では一部の必須機能以外のシステムソフトウェア処理は動作させないことを想定している。

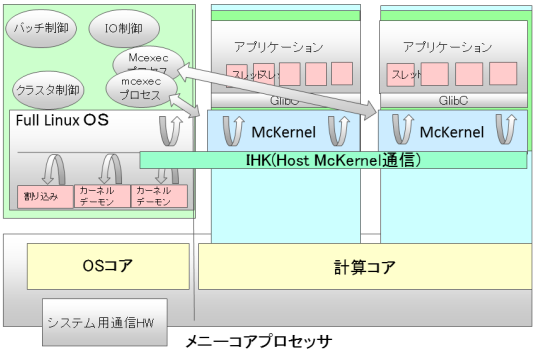

図 1 次世代高性能計算機システム向け OS カーネルの基本構成

図 1 のような構成とすることによって、標準の Linux 環境と Co-design による最適化された実行環境を同時に提供可能になる。ただし、McKernel は特権モードのコードまで最適化可能であるため、McKernel 自体も利用者が最適化可能であるとノードダウンにつながる。このため、システムの実運用においては、特権実行が必要なコードは最小限とし McKernel の本体自体は選択制にして、ユーザレベルライブラリと特権処理を実行する McKernel を含めた Co-design の実現を考える必要がある。

6.3 McKernel によるシステムソフトウェア最適化

本節では、McKernel を OS カーネルとして採用したシステムソフトウェアで、どのようなシステムソフトウェアの最適化が可能になるかについて議論する。

6.3.1 OS カーネル処理の最適化

第 3 章で述べたように現在のシステムソフトウェアには、種々の抽象化、仮想化が行われている他、バックグラウンドで様々な処理が実行されており、次のようなオーバヘッドがあり最適化の余地がある。

- バックグラウンド処理： ファイルシステムのキャッシュ、ファイル I/O のキャッシュ、プロセスメモリ空間の物理メモリ管理等の処理ががデーモン、あるいは I/O 割り込みを契機に実行される。定期的な起動によりキャッシュへの footprint も大きくなる上、定期的実行のためのメモリ管理が生じている。
- マルチユーザ対応： セキュリティ確保のためページ割り当て時には必ず 0 クリア（large page 時には大きな外乱となる）
- マルチタスク対応： メモリ割り当ての公平性確保のための Page Swapping 処理、メモリ割り当てアルゴリズムの不一致によるフラグメントの問題。

例えば、シングルユーザ・シングルプロセスを McKernel の基本とすると、割り当てられたメモリはすべて一つのプ

ロセスのために利用可能になる。こうすれば、上述した処理は行うこと自体が不要になる。また、アプリケーション毎の個別割り当てアルゴリズム採用により無駄無く高速に処理することが可能になる。リソースがすべて一つのプロセスのためのものとすると、メモリ割り当てやほとんどのシステム処理はユーザレベルで実現可能になる。システムコールを呼ぶ必要がなくなりシステムコールのオーバヘッドが解消される。

また、McKernel の導入により、管理システムソフトウェアはすべて Host Linux で実行される。これまで OS カーネルのバックグラウンドで動作していたカーネルスレッドやハードウェアによるインターバルタイマーでの処理自体不要になるため、OS ノイズの原因自体を排除することが可能である。

6.3.2 システムソフトウェア処理の最適化

McKernel との連携によりシステムソフトウェア自体の最適化は Co-design により進める予定であるため現状検討中であるが、OS カーネルとの連携により最適化可能なものは、通信機構、ファイル I/O 処理機構などがあげられる。特に、Linux OS 上でのファイル I/O 処理は Linux の仮想記憶システムと密接に連携して実装されている。この Linux の仮想記憶と連動した Paging 処理が高速なファイル I/O 処理の妨げになっている。McKernel の導入で Linux OS のボトルネックをバイパスすることが可能になり、ファイル I/O 処理の高速化が期待される。

6.4 要件への対応

以上の構成とすることにより、次世代高性能計算機向けのシステムソフトウェアの要件に対応できているのかを評価する。

スケーラビリティの実現: システムソフトウェアの内、管理系のソフトウェアは Host Linux 上で動作し、原則、McKernel 上には管理系の非同期処理は不要になる。ハードウェアのインターバルタイマーも原則不要になるため、OS ノイズ原はなくなることになる。CNK や Catamount に匹敵する OS ノイズを実現可能と期待される。

Co-Design の実現: McKernel を用いてアプリケーション毎にシステムソフトウェアを構成することにより、ハードウェアからアプリケーションレベルまで最適化された環境の実現が可能である。

アプリケーション移植性: 既存アプリケーションやプログラムは Linux 上で稼働することが可能になるため、「京」や PC クラスタとのプログラム開発環境と実行環境の親和性の確保が可能である。また、Co-design の過程においても、問題になる箇所から徐々に最適化が可能なため、最適化導入も敷居が低いと考えている。

高い運用性: 既存のアプリケーションやプログラムは Linux 上で稼働することが可能になるため、様々なアプリケーションプログラムの所要要件を満たす実行環境と開発環境を提供することが可能である。

7. 関連研究

関連研究として、Light-weight Kernel 関連の研究が上げられるが、実験レベルでは多くの研究事例があるので、ここでは、高性能計算機ベンダー上のシステムで実稼働しているものの中で著名なものを取り上げる。

CNK[21]： CNK は IBM BG/L 向けに開発された Light-weight Kernel である。元々メモリ量の少ない BG/L 向けに開発されたものであるため、OS としては最小限の機能しか持たない。POSIX 準拠の GNU libc(glibc) レベルのライブラリを提供しているが、プロセス生成などいくつかの機能が制限されている。機能を最小限にしてシンプルにするために IO サブシステム、メモリ管理 (demand paging, copy on write)、プロセススケジューリングの機能は持たない。機能を限定しシンプルに設計された分、インターバルタイマーによる割り込みさえ不用であるため、OS ノイズは非常に小さい。I/O 処理は I/O ノードに Delegate 処理される。

Catamount[23]： Sandia Natioinal Laboratory により Redstorm(Cary XT-3)[24] 向けに開発された Light-weight Kernel である。Catamount は、QK(Quintessential Kernel) と PCT(Process Control Thread) から構成される。特権モードの処理は QK で実行されるが、QK 自体は資源管理機能は持たない。ユーザ設定と PCT、そして Yod と呼ばれる並列ジョブランチャーによりプログラムを実行可能としている。Catamount は独自にライブラリを作るのは大変であるため glibc を移植している。しかし、thread 機能, pipe, socket などのノード内通信, exec, fork などのプロセス関連機能、mmap 機能など多く機能が実装されていない。通信は libprotals ライブラリにより通信ライブラリ portals が使え、I/O は libsysio ライブラリにより、PVFS, Lustre が提供される。

CNL[25]： Catamount は、CNK と同じく最小限の機能を実現していたが、機能制限が多いため多くのシステムに導入するには問題があった。このため、より多くの機能を提供するため SUSE Linux Enterprise Server をベースに Cray が Linux カーネルを最適化し実装した物が CNL(Compute Node Linux) である。アプローチとしては、「京」の OS カーネルと同じである。

我々の提唱する McKernel 機構は標準の Linux 実行環境を提供しながら、アプリケーション毎に CNK や Catamount のような専用 Light-weight Kernel と同等の環境を

提供できる点が異なる。アプリケーションの移植性についても、性能を律速している機能だけをアプリケーション毎に専用に最適化することが可能な点が特徴であり、広範囲のアプリケーションに適用可能であると考えている。

8. まとめ

本論文では、現在検討している次世代高性能計算機システムの実現に向けたシステムソフトウェアの課題とその解決のアプローチについて述べた、特に、Co-design による最適化手段を継続して確保しながら、より多くのアプリケーションが稼働するシステムソフトウェアの実現が重要となるため、この環境をどう作るべきかについて議論した。

結果、Many Core システムに FULL Linux と Co-kernel が共存可能な McKernel 機構を導入することで、Co-design による最適化手段を継続して確保しながら、より多くのアプリケーションが稼働するシステムソフトウェアの実現手法として提案した。

McKernel は Many Core システムとして Intel Xeon-Phi システムでは稼働し、富士通の FX10 システム上への移植作業中である。今後、機能エンハンスを続け、様々な課題の解決に向け、開発を継続する予定である。

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」による.

参考文献

[1] Riken AICS K computer: http://www.aics.riken.jp/en/kcomputer/.
[2] LLNL Sequoia: https://asc.llnl.gov/computing_resources/sequoia/.
[3] ORNL Titan: http://www.olcf.ornl.gov/titan/.
[4] National University of Defence Technology: http://english.nudt.edu.cn/.
[5] Super Computer TOP500: http://www.top500.org/.
[6] Ubiquitous High Performance Computing (UHPC): http://www.darpa.mil/Our_Work/MTO/Programs/Ubiquitous_High_Performance_Computing_(UHPC).aspx.
[7] Europian Exascale Software Initiative(EESI): http://www.eesi-project.eu/pages/menu/homepage.php.
[8] 戦略的高性能計算システム開発に関するワークショップ (SDHPC): http://www.open-supercomputer.org/workshop/sdhpc/.
[9] 今後の HPCI 計画推進のあり方に関する検討ワーキンググループ: http://www.mext.go.jp/b_menu/shingi/chousa/shinkou/028/index.htm.
[10] IESP Roadmap: http://www.exascale.org/mediawiki/images/2/20/IESP-roadmap.pdf.
[11] 石川裕, 堀敦史, Gerofi Balazs, 高木将通, 島田明男, 清水正明, 佐伯裕治, 白沢智輝, 中村豪, 住元真司, 小田和友仁. 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. 情報処理学会研究報告 2013-OS-125(3). 情報処理学会, Apr. 2013.
[12] International Exascale Software Project(IESP): http://www.exascale.org/iesp/Main_Page.
[13] HPCI 技術ロードマップ白書: http://open-supercomputer.org/wp-content/uploads/2012/03/FutureHPCI-Report.pdf.
[14] 計算科学研究ロードマップ白書: http://open-supercomputer.org/wp-content/uploads/2012/03/science-roadmap.pdf.
[15] 雑誌 FUJITSU 2012-5 月号 (VOL.63, NO.3) スーパーコンピュータ「京」の能力を引き出すコンパイラ技術: http://img.jp.fujitsu.com/downloads/jp/jmag/vol63-3/paper10.pdf.
[16] 住元真司, 川島崇裕, 志田直之, 岡本高幸, 三浦健一, 宇野篤也, 黒川原佳, 庄司 文由, 横川三津夫. 「京」のための MPI 通信機構の設計. SACSIS 2012 - 先進的計算基盤システムシンポジウム. 情報処理学会, May 2012.
[17] 松本幸, 安達知也, 住元真司, 南里豪志, 曽我武史, 宇野篤也, 黒川原佳, 庄司文由, 横川三津夫. Mpi_allreduce の「京」上での実装と評価. 情報処理学会論文誌. コンピューティングシステム, 第 5 巻, pp. 152–162. 情報処理学会, Oct 2012.
[18] 雑誌 FUJITSU 2012-5 月号 (VOL.63, NO.3) スーパーコンピュータ「京」の高性能・高信頼ファイルシステム: http://img.jp.fujitsu.com/downloads/jp/jmag/vol63-3/paper08.pdf.
[19] Performance Evaluation of FEFS on K Computer and Fujitsu's Roadmap toward Lustre 2.x: http://www.opensfs.org/wp-content/uploads/2013/04/LUG2013-FJ-20130410-final-1.pdf.
[20] 雑誌 FUJITSU 2012-5 月号 (VOL.63, NO.3) スーパーコンピュータ「京」の運用管理ソフトウェア: http://img.jp.fujitsu.com/downloads/jp/jmag/vol63-3/paper09.pdf.
[21] Mark Giampapa, Thomas Gooding, Todd Inglett, and Robert W. Wisniewski. Experiences with a lightweight supercomputer kernel: Lessons learned from blue gene's cnk. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pp. 1–10, Washington, DC, USA, 2010. IEEE Computer Society.
[22] 佐伯裕治, 清水正明, 白沢智輝, 中村豪, 高木将通, Balazs Gerofi, 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 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.
[23] Suzanne M. Kelly and Ron Brightwell. Software architecture of the light weight kernel, catamount. In *In Cray User Group*, pp. 16–19, 2005.
[24] SNL Red Storm: http://www.cs.sandia.gov/platforms/RedStorm.html.
[25] Kurt B. Ferreira, Patrick Bridges, and Ron Brightwell. Characterizing application sensitivity to os interference using kernel-level noise injection. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pp. 19:1–19:12, Piscataway, NJ, USA, 2008. IEEE Press.