Oakforest-PACS上でのIHK/McKernelの評価

Balazs Gerofi、高木 将通、石川 裕
理化学研究所計算科学研究センター
中島 研吾、塙 敏博
東京大学情報基盤センター
朴 泰祐
筑波大学大学院システム情報工学研究科

1　プロジェクトの概要と動機

2012年初頭、IHK/McKernelは東京大学情報理工学系研究科下沢拓氏の博士論文[10]の成果の一部である軽量カーネルを基に開発が始まった。オリジナル軽量カーネルは、Intel社Knights CornerのようなサーバCPUに比べて逐次性能が低くキャッシュやメモリ容量も限定的なメニーコア型コプロセッサのための軽量カーネルとして設計された。その後、本軽量カーネルはスタンドアロン型メニーコアCPU向けに拡張されると共にIHK/McKernelと名付けられた。IHKはLinuxドライバとして実装されているハードウェア資源管理と軽量カーネル管理モジュールの名前であり、McKernelは軽量カーネルの名前である。なお、IHKが提供する機能と軽量カーネルが提供する機能を特段区別しなくて良い場合にはMcKernelと呼んでいる。2014年よりMcKernel開発プロジェクトは理化学研究所 計算科学研究機構（現、計算科学研究センター）に引き継がれ、Intel社Xeon phiおよびポスト「京」スーパーコンピュータ[9]向けの軽量カーネルとして開発が続いている。IHK/McKernelの開発動機は以下のとおりである。

- 大規模科学技術計算のような大規模計算のためのスケーラブル実行環境を提供すること。
- アプリケーションに対してLinuxとのフルコンパチビリティを提供すること。すなわち、Linux上で動作しているプログラミングツールやライブラリなどのバイナリがそのまま動作すること。
- 新しいメモリ技術や電力管理機能など新しいハードウェア機能に対してシステムソフトウェアが即座に対応できるようにすること。

軽量カーネルMcKernelはLinuxカーネルと共存し、McKernelが実装していないLinuxシステムコールはLinux側に移譲することによりLinuxフルコンパチビリティを実現している。これにより、性能センシティブなOS機能や新しい機能はLinuxカーネルに直接実装保守することなく、容易に軽量カーネル側に実装することが可能となる。ユーザはLinux機能と共に軽量カーネル側で実装された機能を利用できる。

従来のHPC、ビッグデータ解析、機械学習など様々なワークロードがあるなか、軽量カーネルは全てのニーズを満たさなくても必要に応じて軽量カーネルを提供するだけで済む。IHKが有する再構成機能はノード単位でアプリケーションに応じた軽量OSカーネルを実行させることが可能である。Linuxと軽量カーネルの組み合わせによるアプリケーション最適環境を提供するアプローチの優位性については論文 [5, 6, 2, 3, 4]で既に公表している。

2018年7月時点で、McKernelはOakforest-PACS [7]上で限定ユーザに利用できるようになっている。今後、早期に一般ユーザに利用できるよう完成度を高めていくと共にポスト「京」 [9]

のシステムソフトウェアとしても成熟させていく。

2 アーキテクチャ

図1に示すようにIHK/McKernelは、Interface for Heterogeneous Kernels (IHK) [11]とMcKernelと呼ばれる軽量カーネルから構成される。

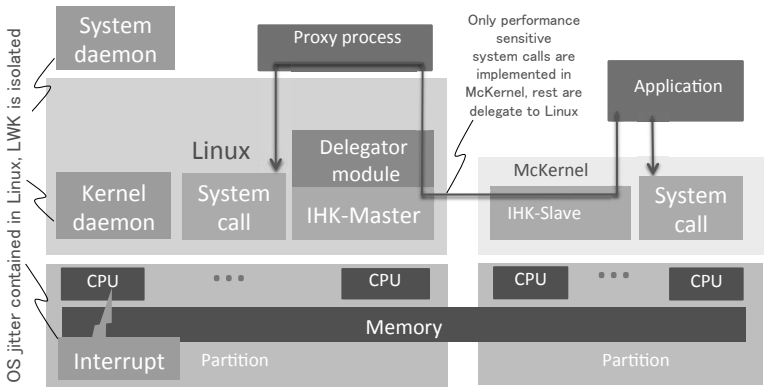

図1 IHK/McKernelの概要とシステムコール移譲機能の概略

2.1 Interface for Heterogeneous Kernels (IHK)

IHK (Interface for Heterogeneous Kernels) はLinuxカーネルに一切の変更を加えずにLinuxドライバとして実装されており大きく2つの機能が提供されている。1つは、ハードウェア資源であるCPUコアや物理メモリなどをソフトウェア的にLinuxカーネルから分離し軽量カーネルのための資源管理を行う機能である。IHKは動的に軽量カーネルのためにこれら資源を割り当て、軽量カーネルをメモリ上にロードし、ハードウェアをリブートせずに軽量カーネルを立ち上げる。ユーザはIHKを経由して軽量カーネルを停止させ異なる軽量カーネルをブートすることも出来るし、資源をLinuxカーネルに返上することも出来る。もう1つのIHK機能はInter-kernel Communication (IKC) と呼ばれている機能である。軽量カーネルはIKC機能を使ってLinuxシステムコールのLinuxへの移譲処理を実現している。

IHKはモジュール構成となっており、IHKコアドライバはIHKドライバを登録するインターフェイスを提供する。XeonやXeon Phi KNLなどSMPマシンではSMPドライバと呼ばれるドライバが使われる。IHKドライバにはこの他にPCI Expressに接続されるXeon phi KNC向けのドライバも開発されている。

2.2 McKernel

McKernelはIHK上に開発されたHPC向け軽量カーネルである。Linux ABI (Application Binary Interface)を維持しており、Linux上で開発されたアプリケーションバイナリはそのままMcKernel上で動作する。言い換えれば、McKernelで動作させるためにアプリケーションを再

コンパイルあるいは特殊なライブラリをリンクすることなく、ユーザのアプリケーションはMcKernel上で動作する。

McKernelはmmapを含む独自のメモリ管理システム、プロセス・スレッド管理、ラウンドロビン・ティックレス・スケジューラ、POSIXシグナル処理、POSIX共有メモリ、ハードウェア性能カウンタなど性能に大きく影響するシステムコールのみを実装し、それ以外のLinuxシステムコールはLinux側に処理を移譲する。McKernel上のプロセスに対応してLinux側に代理プロセス *(proxy process)* と呼ばれるプロセスが生成され、代理プロセスがシステムコールの移譲処理を担当する。代理プロセスはLinux側のプロセスであってもMcKernel上のプロセスと同じ仮想アドレス空間を含む実行環境を保持しており、アプリケーションが発行するシステムコールを透過的に処理する。代理プロセスは通常のLinuxプロセスと同様に様々な状態を保存している。例えば、McKernel上のプロセスがファイルをオープンした時のディスクリプタや内部情報は代理プロセス側に存在し、McKernel上には一切の情報は存在しない。アプリケーションがファイル操作する時に使用されるディスクリプタ番号はLinux 側で割り当てられたディスクリプタ番号が使われる。システムコール移譲処理については論文[5]を参照のこと。

2.3 統合アドレス空間

Linux側代理プロセスとMcKernel側プロセスは仮想アドレス空間を共有することにより、代理プロセスによるシステムコール移譲処理を可能としている。以下にシステムコール移譲処理の概要を示す。

1. McKernelはシステムコールを Linux側に移譲する時、システムコール番号とアーギュメントをメッセージとして組み立て、IHKが提供するIKC機能を用いて代理プロセスにシステムコール処理の要求を送信する。
2. Linux側の代理プロセスはIHK移譲カーネルモジュールに対して`ioctl()`システムコールすることで処理移譲要求を待つ。
3. McKernel側から送信されたシステムコール処理要求メッセージにより、Linux側の移譲カーネルモジュールのIHK割り込み処理ハンドラが呼び出され、待ち状態になっている代理プロセスを起床させメッセージを渡す。
4. 代理プロセスは`ioctl()`システムコールから復帰するとMcKernel側のシステムコール処理依頼に従ってシステムコールを発行する。システムコールの返値をメッセージとして組み立てIHKの移譲カーネルモジュールに対して`ioctl()`システムコールを発行してMcKernel側に結果を通知する。

システムコールのアーギュメントには、例えば、`read()`システムコールのバッファアドレスのようにポインターが含まれる。McKernel側のプロセスがアクセスしているメモリ領域は、Linux側の代理プロセスが同じアドレスでアクセス出来る必要がある。このために、IHK/McKernelはLinuxとMcKernelが統合されたアドレス空間が利用できるようにしている。図 2に統合アドレス空間の概要を示すとともに下記に実装方法の概略を示す。

代理プロセスは`mcexec`コマンドとして実装されている。`mcexec`コマンドをポジション独立バイナリとしてコンパイルすることにより、代理プロセスである`mcexec`プロセスはMcKernel

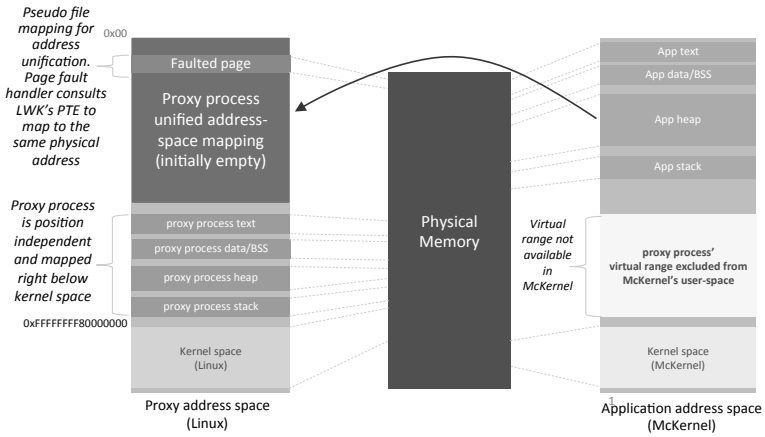

図 2　統合アドレス空間の概要

のユーザアドレス空間外に割り当てることが可能となる。図 2の左側の赤枠で記載されてい領域がこの領域となる。

McKernel側の仮想アドレス空間は、Linux側代理プロセスが疑似ファイル経由でメモリマッピングすることにより当該仮想アドレス空間をアクセスすることができる。この領域は図 2の左側の青色である。

なお、代理プロセスは疑似ファイル経由でメモリマッピングした時には物理アドレスと仮想アドレスをマッピングせず、代理プロセスが当該メモリ領域をアクセスしてページフォルトが発生した時に物理アドレスと仮想アドレスのマッピングを行っている。図 2の左側*faulted page*と記載されている箇所がページフォルトが発生している領域である。本機能により、代理プロセスがMcKernelの代わりにシステムコールを実行している場合にアプリケーションから渡されたメモリアドレスを代理プロセス側がアクセスできる。

2.4　デバイスドライバ透過性

アプリケーションがデバイスをアクセスする方法には、`read()`、`write()`、`ioctl()`などのI/Oシステムコールを使う方法とメモリマッピング機能を使ってデバイスレジスタをアクセスする方法がある。前者については、McKernelはI/Oシステムコールを実装せず、それらはLinuxカーネルに移譲する。このためにIHK/McKernelは統合アドレス空間を実現している。

メモリマッピング機能を用いた手法はInfiniband [1]のようなHPC向けネットワークデバイスでよく使われる。Infinibandでは、アプリケーション起動時にInfinibandネットワークデバイスをメモリマッピングして、アプリケーションから直接ネットワークデバイスのレジスタを読み書きして通信を実現している。

McKernelにおけるデバイスのメモリマッピング機能はLinuxのデバイスドライバを修正することなく以下のように実現している。詳細は論文 [2]を参照のこと。

1. アプリケーションがデバイスに対して`mmap()`システムコールを発行すると、McKernel

はLinux側のIHK移譲カーネルモジュールに要求を送る。

2. IHK移譲カーネルモジュールは代理プロセスアドレス空間にデバイスファイルをマップすると共に当該メモリ領域のページフォルトを追跡するオブジェクトを生成する。
3. Linuxからの返答時、McKernelはアプリケーションアドレス空間に対応する仮想アドレス領域を割り当てるとともに当該アドレス領域がLinux側のデバイスファイルマッピング領域であることを記録しておく。すなわち、Linux側のデバイスドライバのメモリ領域とMcKernel側の当該メモリ領域のアドレスは異なることになる。
4. アプリケーションがマッピングされたメモリ領域を初めてアクセスするとページフォルトが発生する。McKernelのページフォルトハンドラは当該領域がlinux側のデバイスドライバのメモリ領域であることを知っているので、Linux側IHKモジュールに対して物理アドレスの解決を要求する。Linux側は追跡オブジェクト経由で物理メモリアドレスを見つけてMcKernelに伝え、McKernel側で当該仮想アドレスと物理メモリのマッピングを行う。

HPC向けデバイスの多くはメモリマッピング機能を用いてI/Oシステムコールを用いずにアプリケーションレベルでI/O機能を実現することが多い。しかし、Intel社製OmniPathネットワークのようにシステムコールによって通信機能が実現されている場合がある。この場合、通信の度にLinuxカーネルに処理を移譲するために遅延が大きくなり並列性能劣化の原因となる。そこで、システムコールを移譲せずMcKernel内でデバイス処理が出来るようにMcKernelにデバイスドライバを移植するPicoDriverというフレームワークを提案している [4]。本フレームワークを用いて、OmniPathネットワークドライバの通信部分をMcKernelに移植しLinuxと同等以上の通信性能が達成されている。

2.5 Linux 疑似ファイルシステム

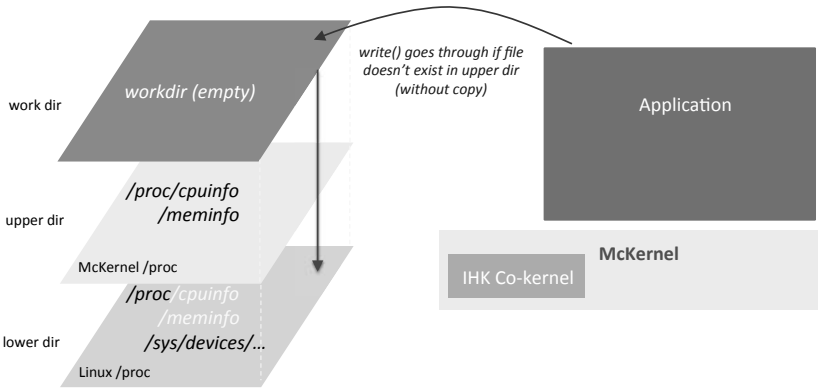

図 3 `mcoverlayfs` の概要

`/proc` and `/sys`などのLinux疑似ファイルシステムはデバッガ、プログラミングツール、ライブラリなどで参照されており、Linuxアプリケーションが透過的に動作させるためにはこれら

疑似ファイルシステムがMcKernelで実現されている必要がある。ファイルシステム操作は代理プロセス側で処理されるため、そのままでは、/proc and /sys などの疑似ファイルもLinuxが提供する疑似ファイルがアクセスされてしまう。McKernelが提供すべき疑似ファイルシステムは、Linux側の`mcoverlayfs`と呼ばれるLinuxカーネルモジュールによって実現されている。`mcoverlayfs`はLinuxのオーバーレイファイルシステムを改造して作られている。

図 3に`mcoverlayfs`の概要を示す。`/proc/meminfo`などのファイルはMcKernel固有のファイルとして処理される。本オーバレイは代理プロセスに対してのみ有効であり、他のLinuxプロセスには影響しない。本技法はDockerのcontainer engine [8] と同様である。

3 評価

3.1 実験環境

筑波大学と東京大学によるJCAHPCが運用するOakforest-PACS (OFP)上で、CORALベンチマーク集からAMG2013、MiniFE、HPCG、 Lulesh、 Milc、Nekbone、東京大学のGeoFEMおよびGAMERAをベンチマークプログラムとして走らせた。使用した最大計算ノード数は8,192基である。これらはIntel社製OmniPathネットワークで接続されている。各計算ノードは、CPUとしてIntel社製*Knights Landing* (KNL) (68 CPUコア (4ハードウェアスレッド/コア))、16 GiBの高バンド幅メモリMCDRAMと96 GiBのDDR4メモリが搭載されている。全ての実験においてKNLのメモリモードはQuadrant Flat、すなわち、MCDRAMとDDR4メモリは異なるアドレス空間でアクセスできるように設定した。OSは272論理CPUと2つのNUMAドメインから構成される。LinuxはCentOSのLinux `3.10.0-693.11.6`版にIntelのKNLパッチが適応された版を使用し、MPI通信ライブラリは、Intel MPI Version 2018 Update 1 Build 20171011を使用した。

論文 [3]では2,048ノードまでのHPCミニアプリ実行結果として、McKernelがLinuxよりも良いスケーラビリティが得られていることを報告した。本節では、8,192ノードまでの性能評価結果を示す。

Linux上でのベンチマークプログラム実行では、OFPにおいて運用されているLinuxの設定以外にLinuxに対して適切な設定を施しスケーラビリティを向上した設定の２つのケースで計測している。以降示すグラフの凡例では、前者を`Linux`、後者を`Linux + corespec`としている。`Linux + corespec`では、MPIプロセスは`nohz_full`カーネルパラメータが設定されたコアで実行するようにした。これにより、MPIプロセスが動作しているコア上ではタイマー割り込み処理が行われずOSノイズが減少する。これ以降、`Linux`を通常Linux環境、`Linux + corespec`を最適化Linux環境と呼ぶことにする。

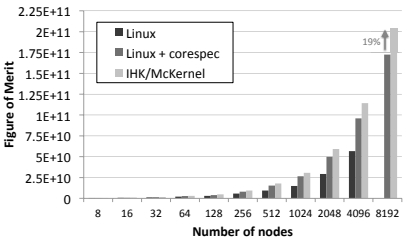

図 4　AMG2013 (CORAL benchmarks) の結果

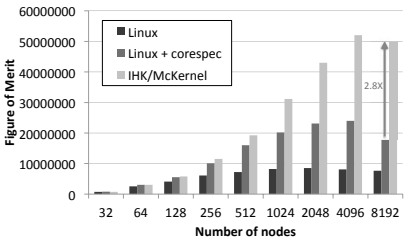

図 5　miniFE (CORAL benchmarks) の結果

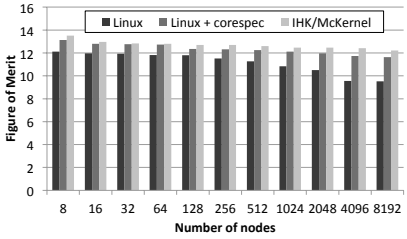

図 6　GeoFEM (U Tokyo) の結果

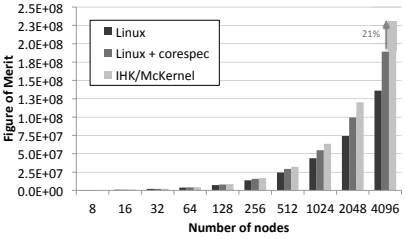

図 7　MILC (CORAL benchmarks) の結果

3.2　結果

AMG2013ミニアプリの結果を図 4に示す。本節で示すグラフのX軸はノード数、断りのない限りY軸はFOM (Figure of Merit) を表している。FOMの数字が高いほど性能が優れている。IHK/McKernelは最適化Linux環境に比べて19%の性能向上、通常Linuxに比べて2倍以上の性能向上が得られている。

図 5にMiniFEミニアプリの結果を示す。通常Linux環境では512ノードを超えるとスケーラブルではなくなる。最適化Linux環境では2,048ノードまでスケールするが、その後性能向上はほぼ認められない。一方、IHK/McKernelは4,096ノードまで順調にスケールし、8,192ノードでも性能劣化の割合は少なく、最適化Linuxに比べて2.8倍の性能向上が達成されている。Linuxにおける性能劣化の原因調査のためのマシン利用時間が確保できなかったため原因が定かではないが、メモリ管理の違い、OSノイズによる`MPI_Allreduce()` 性能劣化、が原因ではないかと推察している。

図 6は、GeoFEMの性能結果である。IHK/McKernelは6 %の性能向上が達成されている。

図 7にMILCの結果を示す。MILCでは計算ノードあたり32 MPIプロセスを割り当てた。測定時のIntel MPI通信ライブラリ環境ではMPIの初期化にスケーラビリティ問題が存在し、計算ノードあたり32 MPIプロセスでは4,096ノードまでしか実行できなかった。MILCの実行では4,096ノードまでで22%の性能向上が得られることがわかった。

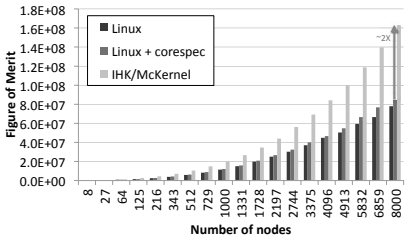

図 8　Lulesh (CORAL benchmarks) の結果

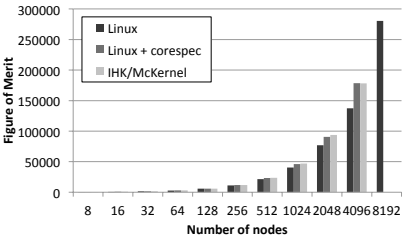

図 9　HPCG (CORAL benchmarks) の結果

図 8はLuleshベンチマーク結果である。Luleshはノード数n^3で実行される。OFPの環境では8,000ノードが最大となる。本ベンチマークでもMcKernelは最適化Linux環境よりも約2倍の性能向上が達成されている。

これ以降述べるアプリケーションベンチマークではMcKernelの優位性が大きく現れなかったが、Linux環境よりも性能が下回ることはなかった。図 9にHPCGの結果を示す。MPI初期化問題のために8,192ノードでの実行が出来なかった。4,096ノードまででIHK/McKernelおよび最適化Linuxは同等の性能を示した。

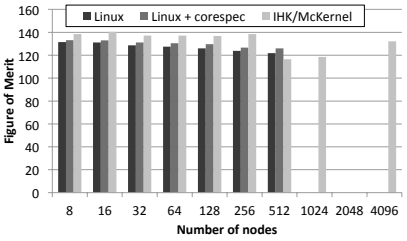

図 10　Lammps (CORAL benchmarks) の結果

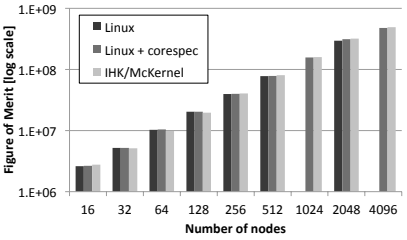

図 11　Nekbone (CORAL benchmarks) の結果

図 10にLammpsの結果を示す。McKernelはLinuxに比べて7%の性能向上が達成されている。大規模ノードではLinux環境ではLammpsは動作しなかった。なお、論文 [3]ではLammpsのMcKernel上での実行はLinux 環境よりも性能が劣っていた。この原因の 1 つは第2.4節で説明したOmniPathネットワークの問題であったが、今回の計測ではPicoDriver[4]を用いて通信性能が向上したためにLinuxに比べて7%の性能向上が達成できた。

図 11にNekboneの結果を示す。McKernelとLinux環境では大きな性能差が見られなかった。

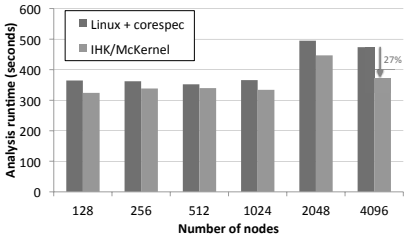

図 12　GAMERA (U Tokyo) の結果

図 12に東京大学のGAMERAアプリケーションの実行結果を示す。本グラフのY軸は実行時間である。McKernelは最適化Linux環境に比べて27%の性能向上が観測された。

4　まとめ

本稿では、HPCのための軽量カーネルIHK/McKernelの概要を紹介した後、Oakforest-PACS上でアプリケーションベンチマークを使ってIHK/McKernelとLinuxカーネルを比較した。CORALベンチマーク集のMiniFEの実行ではIHK/McKernelはLinuxの2.8倍の性能向上が実測された。今後、一般ユーザ利用に向けてIHK/McKernelの安定化を進めていく。

謝辞

本論文の一部は、文部科学省「特定先端大型研究施設運営費等補助金（次世代超高速電子計算機システムの開発・整備等）」で実施された内容に基づくものである。CORALベンチマークの実行パラメータについてIntel社mOSチームにお世話になった。東京大学地震研究所の市村強准教授および藤田航平助教にはGAMERAの利用を許諾頂き、Intel社堀越将司氏にはその実行パラメータについて助言頂いた。OFP 利用に当たっては理研 小倉崇浩氏、富士通 坂口吉生氏、佐伯敏郎氏にお世話になった。ここに感謝する。

参考文献

[1] InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.3.1, Nov 2016.

[2] B. Gerofi, M. Takagi, A. Hori, G. Nakamura, T. Shirasawa, and Y. Ishikawa. On the Scalability, Performance Isolation and Device Driver Transparency of the IHK/McKernel Hybrid Lightweight Kernel. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1041–1050, May 2016.

[3] Balazs Gerofi, Rolf Riesen, Masamichi Takagi, Taisuke Boku, Yutaka Ishikawa, and Robert W. Wisniewski. Performance and Scalability of Lightweight Multi-Kernel based Operating Systems. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2018 (to appear).

[4] Balazs Gerofi, Aram Santogidis, Dominique Martinet, and Yutaka Ishikawa. PicoDriver: Fast-path Device Drivers for Multi-kernel Operating Systems. In *2018 ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, May 2018 (to appear).

[5] Balazs Gerofi, Akio Shimada, Atsushi Hori, and Yutaka Ishikawa. Partially separated page tables for efficient operating system assisted hierarchical memory management on heterogeneous architectures. In *13th Intl. Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2013.

[6] Balazs Gerofi, Akio Shimada, Atsushi Hori, Takagi Masamichi, and Yutaka 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*, HPDC, pages 73–84, New York, NY, USA, 2014. ACM.

[7] Joint Center for Advanced HPC (JCAHPC). Basic specification of Oakforest-PACS. `http://jcahpc.jp/files/OFP-basic.pdf`, March 2017.

[8] Dirk Merkel. Docker: Lightweight Linux containers for consistent development and deployment. *Linux J.*, 2014(239), March 2014.

[9] RIKEN. The Post-K Computer Project. http://www.r-ccs.riken.jp/en/postk/project, 2017.

[10] Taku Shimosawa. Operating System Organization for Manycore Systems. http://www.ipsj.or.jp/magazine/hakase/2011/OS01.html, 2011.

[11] Taku Shimosawa, Balazs Gerofi, Masamichi Takagi, Gou Nakamura, Tomoki Shirasawa, Yuji

Saeki, Masaaki Shimizu, Atsushi Hori, and Yutaka Ishikawa. Interface for Heterogeneous Kernels: A framework to enable hybrid OS designs targeting high performance computing on manycore architectures. In *21th Intl. Conference on High Performance Computing*, HiPC, December 2014.