Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186–5140
Volume 2, Number 1, pages 1–3, January 2013

Power-Saving And Reliable Computer System on Heterogeneous Multi-core Architecture

Gaku Nakagawa
Department of Computer Science
Graduate School of System and Information Engineering
University of Tsukuba
Tsukuba, Ibaraki, JAPAN
gnakagaw@cs.tsukuba.ac.jp

Shuichi Oikawa
Division of Information Engineering
Faculty of Engineering, Information and Systems
University of Tsukuba
Tsukuba, Ibaraki, JAPAN

***Abstract*—Low power consumption and high reliablity are important topics for today's computer system. We advocate that heterogenepus multi-core architecture and implementing operating system in Java and C language are effective measures. In this paper, we construct an computer system model on heterogeneous multi-core architecture. To proof of concept, we are going to construct an computer system. We will explaint roadmap to construct that of system and current status.**

I. はじめに

計算機の高性能化に伴い，その消費電力は増大し続けてきた．しかしながら，社会からの省エネルギーに対する要求やスマートフォン，タブレットといった携帯情報端末の普及に伴い，高い性能を維持したまま，低消費電力を実現することが近年では求められている．

この 2 つの要求は従来は相反するものであったが，ヘテロジニアスマルチコアアーキテクチャを活用することで，これを解決しようとするアプローチが数多く検討されている．[4][3] このアプローチは研究だけではなく，実際の製品でも採用されており，nvidia 社が開発したプロセッサである Tegra 3 を例として挙げることができる．

低消費電力化だけでなく，計算機システムの信頼性を保つことにも，近年要求が高まっている．信頼性の確保のためには様々な方策が考えられるが，オペレーティングシステム (OS) の信頼性を高めることは有効な方策の一つである．OS の実装には主に C 言語が用いられることが多い．C は OS 開発のために開発された言語であり，ある程度の抽象レベルからハードウェアに近いレベルまでの柔軟なメモリ操作ができる点や，高速に動作するというメリットを持っている．しかしながら，型付けが弱く型に起因するエラーが起こりやすい，メモリリークを起こしやすいなど，信頼性確保の点からはデメリットも多い．そのため，C とそれ以外のプログラミング言語を組み合わせて OS を開発することが数多く検討されている．中でも，Java を利用したものは，C に比べて遜色ない性能を実現した例もある．[6]

以上の背景を踏まえて，この論文ではヘテロジニアスマルチコアアーキテクチャ上で実装に Java を導入した OS を動作させ，低消費電力かつ高信頼性を実現する計算機システムを提案する．また，その実現のためのロードマップと現在の状況についても提示する．

II. ヘテロジニアスマルチコアアーキテクチャによる省電力化

コンピュータシステムの動作時間のうち，CPU 待機状態である割合は非常に高い．また，何らかの処理を行っている際でも，CPU に搭載されている複数のプロセッサコアのうち，利用されているのは一部のコアで，待機状態になっているコアがあることが多い．この待機状態でも，プロセッサは一定の電力を消費するため，電力を無駄遣いしてしまう問題がある．

Mogul らの研究 [1] では，CPU 上に命令セットアーキテクチャ (ISA: Instruction Set Architecture) は共通であるが，規模の異なるプロセッサコアを搭載したマルチコアプロセッサを利用することで，計算機の電力使用効率を劇的に改善できる可能性を示した．(ASISA-CMP: Asymmetric Single-ISA Chip Multi Processor) Mogul らは現代の高性能プロセッサは OS の実行には過剰な性能を持っているとし，OS 実行を適切な規模のコアで行うことで，低消費電力が実現できるとした．ASISA-CMP では 2 つの種類のプロセッサコアを 1 つの CPU に搭載している．1 つは多機能で高い処理性能を持ち消費電力が大きい complex core で，もう一方は，complex core に比べると性能は低いが，消費電力が小さい simple core である．この ASISA-CMP では simple core が OS タスクを実行し，complex core がそのほかの高負荷処理を実行する．このようにワークロードの処理内容に応じて，処理するコアを選択することで低消費電力を実現する．また計算機が待機状態の時には complex core の動作を完全に停止し，simple core だけで動作を行うことも可能であり，より進んだ低消費電力を実現することができる．

この ASISA-CMP は，1 つの CPU に異種のプロセッサコアを混在させたマルチコアアーキテクチャはマルチコアアーキテクチャの中でもヘテロジニアスマルチコアアーキテクチャの一種である．これと対照的に，1 つの CPU に同種のプロセッサコアを複数搭載したものは，ホモジニアスマルチコアアーキテクチャと分類される．

III. Java 言語による OS 実装

I で触れたように，C 言語はメモリやハードウェアへのアクセスしやすさ，動作速度という点ではメリットも多いが，一方で型付けの弱さやメモリリークの起こりやすさなど，信頼性確保の点からのデメリットも多い．これと対照

的に，Java はメモリへのアクセスしやすさなどは C に劣るものの，以下の点で，C に対するメリットがある．

- 強い型付けによるエラーの回避: Java の型検査は，C のそれよりも強力である．そのため，C によるプログラミングで起こりがちであった，データ型の間違いによるエラーを回避することができる．
- メモリリークの回避: ガーベッジコレクタ (GC: Garbage Collector) は参照されなくなったメモリ領域を自動的に検知し，解放することができる．C では，使用しないメモリ領域の管理はプログラマに責任があるため，メモリリークを発生させやすい．
- 拡張性の向上: オブジェクト指向の利点の一つに，その拡張性の高さがある．OS 開発に Java を導入することで，OS の信頼性を確保したまま，振る舞いを変更したり，機能を追加することが可能である．

このことから，OS を C のみで実装するのではなく，Java を組み合わせることで，より信頼性の高い OS を実現することができる可能性がある．

しかしながら，Java による OS 実装には大きな問題がある．それは実行速度である．通常，Java で記述されたプログラムは Java バイトコードに変換され，Java 仮想マシン (JVM: Java Virtual Machine) で解釈，実行される．この JVM によるオーバーヘッドは大きく，実行時にあらかじめ実ハードウェアの命令に置き換える Just-In-Time 方式などが開発されたが，C 言語での実装には及ばない．この問題点が，これまで Java による実用的な OS 実装を妨げてきた．

IV. 提案モデル

III で述べたように，Java による OS 実装には実行速度という大きな問題がある．そこで本研究の提案モデルでは，II で示した，ASISA-CMP モデルを拡張し，Java で構築した OS を特定のコアに割り当てることで，この問題点を解決し，高性能，低消費電力かつ高信頼性を確保した計算機システムを実現する．

提案するシステムのねらいは以下の 3 つである．

- OS とアプリケーション実行をコア間で分離すること: II で示した，ASISA-CMP では，規模の異なる 2 種類のコア間で，低負荷である OS 実行と高負荷であるアプリケーション実行を分離している．このモデルを利用し，本モデルでも規模のことなるコアを搭載し，OS 実行とアプリケーション実行を, 分離する．これにより，Java による OS 実行で増えるオーバーヘッドを軽減することも狙いの一つである．
- 待機状態では必要最小現のコアのみで動作すること: II で示した通り，ASISA-CMP では待機状態に電力を消費するコアの電源を切ることで，消費電力を抑制することができる．本モデルでも，この特徴を利用する．
- OS 実装に Java を利用すること: III で触れたように，OS 実装に Java を取り入れることで，計算機の信頼性向上を達成できる可能性がある．

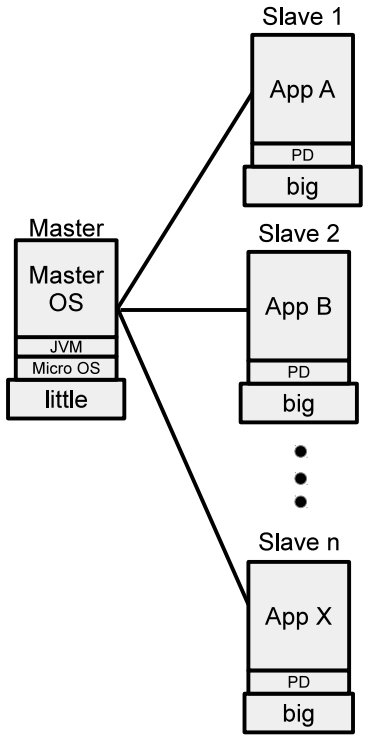

Fig. 1. 提案システム

Fig. 1 に提案モデルを示す．このモデルは，規模の異なるコアを複数搭載したヘテロジニアスマルチコアアーキテクチャで構成され，そのプロセッサ群は little core, big core に分類される．

little core は big core コアよりも性能が小さいが，かつ消費電力が小さいプロセッサコアである．このコアは ASISA-CMP モデルでの simple core に該当する．little core では Java で記述された OS である Master OS が動作する．この Master OS を動作させるために，little core では必要最初限の OS (Micro OS) と JVM が動作する．Master OS の動作には，Java の機能の一部しか用いないため，搭載する JVM は Java 仕様のサブセットのみをサポートするものとする．

big core は little core よりも強力な性能をもつが，消費電力が大きいプロセッサコアである．このコアではアプリケーションの実行を担当する．アプリケーションと Master OS のコミュニケーションのために，big core では Processor Driver (PD) が動作する．この PD はシステムコールのインタフェースや割り込みの通知などを OS の機能のうち，アプリケーションのごく近い層を担当する．

V. ロードマップ

IV で示したモデルを実現するためには，検討すべき項目や新たに開発すべきものが多量にある．そこで，本研究では Phase 1 から Phase 3 まで段階を追って，実装を進め，段階毎にモデルの妥当性を検討する．

Phase 1 では，Java による OS タスクの性能を評価するために，Linux OS 上に検証環境を構築する．Fig. 2 に概略図を示す．この段階では Java OS は Java 実行環境 (JRE) 上で動作するアプリケーションとして動作する．この部分は，Fig. 1 の little core に相当する．big core に相当する部分は，

loader と App からなる部分である．App はアプリケーションタスクで，loader は，Processor Driver に相当する．loader は通常の Linux プロセスとして起動され，アプリケーションタスクをプロセス空間内にロードし，その領域を共有メモリ経由で Java OS からアクセスできるようにする．

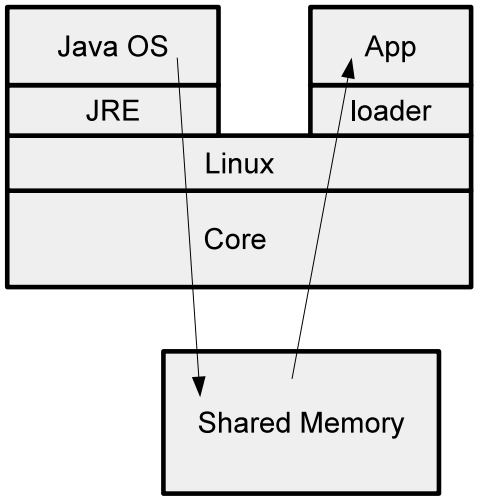

Fig. 2. Phase 1

Phase 2 では，コア間のコミュニケーション方式を検討するために，Linux の SMP カーネルを拡張した検証環境を構築する．Fig. **??** に概略図を示す．Phase 1 ではアプリケーションと OS の動作コアを明示的に分離し，Java OS と PD 間でのコミュニケーション方式を検討する．現在検討しているのは，コア間の割り込みと共有メモリによる通信である．

Phase 3 では，ヘテロジニアスマルチコアアーキテクチャを搭載した計算機環境を整備し，Phase1, 2 で得られた知見を元に，仕様を詳細化し Fig. 1 に示したモデルの実装を行う．

VI. まとめと今後の課題

本論文では，ヘテロジニアスマルチコアアーキテクチャと OS 実装に Java を導入することによる，高性能，低消費電力かつ高信頼性を確保した計算機システムのモデルを提案し，モデルの実現のためのロードマップを示した．

現段階では，ロードマップに示した Phase 1 のうち，PD に相当する loader の実装と loader と Java OS の通信機構のテスト実装が完了している．今後は実装を進め，Java による OS タスク処理の性能評価などを進め，モデルの実装を目指す予定である．

REFERENCES

[1] Mogul, J.C.; Mudigonda, J.; Binkert, N.; Ranganathan, P.; Talwar, V.; , “Using Asymmetric Single-ISA CMPs to Save Energy on Operating Systems,” Micro, IEEE , vol.28, no.3, pp.26-41, May-June 2008

[2] R. Kumar et al., “Single-ISA Heterogeneous Multi-core Architectures: The Potential for Processor Power Reduction,” Proc. IEEE/ACM Int’l Symp. Microarchitecture (MICRO 03), IEEE CS Press, 2003, pp. 81-92.

[3] Baumann, A., Barham, P., Dagand, P.-E., Harris, T., Isaacs, R., Peter, S., Roscoe, T., Schpbach, A. andd Singhania, A.: The multikernel: a new OS architecture for scalable multicore systems, Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, SOSP ’09, New York, NY, USA, ACM, pp.29—44 (2009).

[4] Nightingale, E.B., Hodson, O., McIlroy, R., Hawblitzel, C. and Hunt, G.: Helios: heterogeneous multiprocessing with satellite kernels, Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, SOSP ’09, New York, NY, USA, ACM, pp.221—234 (online), DOI:http://doi.acm.org/10.1145/1629575.1629597 (2009).

[5] J.K. Ousterhout, “Why Aren’t Operating Systems Getting Faster as Fast as Hardware?” Proc. Usenix Summer 1990 Conf.,Usenix Assoc., 1990, pp. 247-256.

[6] Golm, M., Felser, M., Wawersich, C. and Kleioder, J.: The JX Operating System, Proceedings of the General Track of the annual conference on USENIX Annual Technical Conference, Berkeley, CA, USA, USENIX Association, pp.45-58(online),available from http://dl.acm.org/citation.cfm?id=647057.713870 (2002).