社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－ARC－180 (6)
2008／10／20

# ループ構造に着目したマルチグレイン・マルチレイヤ並列処理システムの提案

佐 藤 幸 紀[†1]

本稿では、5〜10 年後に必要とされる研究のシーズとしてループ構造に着目したマルチグレイン・マルチレイヤ並列処理システムを提案し、その方法論と詳細を論じる。さらに、予備実験の結果よりマルチグレイン・マルチレイヤ並列処理システムを実現する上でループ構造に着目することは有用であることを示す。

# Multigrain-multilayer parallel processing system based on detection of loop structure

YUKINORI SATO[†1]

In this paper, we propose multigrain-multilayer parallel processing system focused on detection of loop structure, and then present the methodology and details of the system. In addition, from the result of preliminary experiment, we show the effectiveness of being focused on the loop structure to realize multigrain-multilayer parallel processing system.

## 1. はじめに

歴史的にマイクロプロセッサの性能向上はアーキテクチャの面で並列処理技術が進歩してきたというよりむしろ半導体のスケーリングに伴うデバイス的な面でプロセッサ動作周波数が向上してきたことに大きく依存していた。しかしながら、近年は消費電力や信頼性の観点からこれまでのペースでプロセッサの動作周波数を向上させるのは困難となり、並列処理を核とした新しいアーキテクチャ技術の確立が求められている。このような技術として、複数のプロセッサコアを利用した並列処理、SIMD や FPGA といった強力な演算能力をもつアクセラレータを利用したプログラムの高度化並列化などが考えられている。

複数のプロセッサやアクセラレータにより性能向上を目指した場合、そのようなハードウェアを実現する技術よりも、その理論性能を引き出すための技術が重要になるであろう。現時点でも Cell プロセッサや FPGA は非常に強力な演算能力を備えているが、その能力を引き出すためには非常に高度で時間のかかるチューニングを行う必要があることが問題となっている。従って、これまでのような性能向上はデバイス主導にて達成されるという先入観を捨て、並列処理を核とした適応型の技術が性能向上を牽引するという方向へ転換する必要があろう。

しかしながら、並列処理を行うためにアプリケーションの事例毎に変化する並列化するべき対象を的確に見つけ出し、その部分を並列実行する適切な手段を見出すことは解空間の広い発見的な作業である。さらに、アプリケーションプログラムは年を追うごとにコードの規模や複雑さを増してきている。また、他人が書いたサブルーチンやライブラリを生産性向上の目的に利用する機会が増えていることに由来してプログラムの全体構成の把握や、その一部の修正することが非常に困難な作業となりつつある。このような状況の下で個々のアプリケーションにあわせた適応型の並列処理技術を実現するためには、その方法論を確立し、自動化を進めることが必須であろう。

以上のような必要性を踏まえて、本稿では 5〜10 年後に必要とされる研究のシーズとしてループ構造に着目したマルチグレイン・マルチレイヤ並列処理システムを提案する。アプリケーションからマルチグレイン・マルチレイヤ並列性を抽出するためのモデルを構築するために本研究ではループ構造に着目する。筆者らはプログラムの並列化を支援することを目的として、プログラムの実行時にループの入れ子構造や関数呼び出しとの位置関係を検出する手法を提案している[1)]。本稿では、筆者らの提案しているループ構造検出[1)] を応用し、ループ構造に着目したマルチグレイン・マルチレイヤ並列処理システムを実現することを目指す。

マルチグレインというのは細粒度から粗粒度まで多

†1 北陸先端科学技術大学院大学 情報科学センター
Center for Information Science, Japan Advanced Institute of Science and Technology

段階であるという意味であり、マルチグレイン並列処理とは並列化の対象とする部分の粒度が様々であったとしてもシームレスに並列処理が可能であることを意味する。これまでの並列化技術では粒度が異なる場合、統一的な尺度で測ることが困難であった。例えば、細粒度では命令レベル、ループレベルという尺度であり、粗粒度では関数レベル、タスクレベル、プロセスレベルという尺度と粒度毎に尺度が変化してきた。本研究においては、ループ階層構造に着目することにより多種多様な並列部分の構造を一元的に把握することを目指す。

マルチレイヤというのはデバイスからマイクロアーキテクチャ、命令セットといったアーキテクチャ、さらにコンパイラ、OSなどのミドルウェア、そしてアプリケーションプログラムやソフトウェアといった多様なレイヤのことを意味し、マルチレイヤ並列処理とは複数のレイヤにより効果的に並列化を行うという意味である。様々な粒度を持つ並列化対象を並列化するためには、これまでのようにマイクロアーキテクチャレベルで分岐予測を行い細粒度並列性を得るということや、ソフトウェアレベルでプログラマが手動にて粗粒度並列処理を明示するというような単一のレイヤで完結する技術では不十分であり、複数のレイヤが有機的に協調することが必要であると考えられる。本研究では、ループというプログラムの構造を軸として複数のレイヤが協調しながら並列処理を行うことを目指す。

本稿の構成は以下の通りである。2節では計算機アーキテクチャ研究の方法論の現状把握と方向性について議論し、マルチグレイン・マルチレイヤ並列処理を提案する。3節ではループ階層構造と提案する並列処理の関係について述べる。4節では提案するマルチグレイン・マルチレイヤ並列処理の有効性を示すための予備実験を行う。5節は結論である。

## 2. 計算機アーキテクチャ研究の方法論

プロセッサの性能向上はHPC分野、エンタープライズ分野、組み込み分野など幅広い分野で横断的に求められてきている。このような要求に対して提供されるデバイスを利用して実現されるべき新しいアーキテクチャ技術を確立することは計算機アーキテクチャ分野の研究に共通した目標であろう。Oskinは処理性能の向上をマルチコアやメニーコアプロセッサに求める場合、将来的に大切なことはムーアの法則と同じスピードでソフトウェアから並列性を抽出する技術であると述べ、これからのアーキテクチャ研究はコンピュータが発明されて以来で最もエキサイティングな時代になるかもしれないと予測している[2)]。このような予測の背景には、アーキテクチャ研究として、どのような並列部分を対象として、並列実行をどのような手段で実現するかということに多種多様の方向性が存在し、未だにその方向性は定まっていないという現在の研究状況があると考えられる。

近年の計算機アーキテクチャ研究においてはプロセッサの振る舞いをシミュレーションして性能評価することが方法論として確立された。例えばSimpleScalarのようなシミュレータによりキャッシュの振る舞いや制御フロー、データフローの現象を容易に再現することが可能となった。それらの現象を再現しその過程を理解した結果、アウトオブオーダ命令スケジューリングなどの最適化技術や分岐予測、データプリフェッチという予測する技術が確立された。しかしながら、並列処理の対象と手段は大部分が命令レベル並列性とハードウェアによる動的並列性抽出に限定されていた。近年のアーキテクチャ研究がシステムの性能に対してファインチューニングとして数%の性能向上を提供する以上の成果が挙げられなかったのは限定された単一の対象と手段に絞って研究を重ねた結果と考えられる。

これからの計算機アーキテクチャ研究はシステムのファインチューニングの方法ではなく、根本的にシステムの性能を数倍に改善する方法を目的とすることが必要であろう。この根本的な改善を達成するためには実現のための方法論も同時に確立する必要がある。

プロセッサのシミュレーションによる性能評価という方法論において本質的な点として注意するべきことは、シミュレーションという方法論の成功ではなく、シミュレーションにより現象を再現し、さらに現象の最適化や予測につなげるという科学的なプロセスであると思われる。計算機を利用して数値実験を行う計算科学という分野においては、マルチスケール・マルチフィジックス現象の統合シミュレーションというアプローチが注目されている。これは、1つの分野の理論では対応できない複雑な現象を解析するためにミクロからマクロまでの幅広いスケールにわたり様々な物理現象の基礎方程式を解き、現象を再現しようという取り組みである。現象が再現できれば、その現象の過程を最適化し、さらには予測していくことが可能となるといわれている。

複雑な現象であっても解析するための第一歩は現象を再現し把握することである。計算機アーキテクチャの分野においては並列処理の対象と手段がそれぞれ単一に定まっているならば性能予測ができるレベルまでは達成されている。しかしながら、並列化の対象と手段が多種多様である複雑なシステムの場合、その振る舞いには解明されていないといえるであろう。複雑なシステム上で起こる複雑な現象を把握するためには、それらを正確に再現することから取り組む必要がある。計算機システム上において細粒度から粗粒度まで幅広い粒度を並列処理の対象とし並列処理を実際に行う手段がソフトウェアからハードウェアまで様々なレイヤにまたがっているという現実があるのであれば、それぞれで起こっておる現象を統一的に再現することから

取り組むべきである。以上のような統一的な現象の再現を実現する1つの形態としてループに着目したマルチグレイン・マルチレイヤ並列処理を提案する。

## 3. ループ階層構造と並列性

マルチグレイン・マルチレイヤ並列処理を行うためには様々な現象を統一的に把握していく必要があり、様々の現象の間にある考え方のギャップを橋渡しする取り組みが必要である。例えば、ハードウェア上でアプリケーションプログラムのコードが実行される場合、その処理の大半がプログラムのループで占められることが知られている。すなわち、ハードウェアから見るとプログラムはループの集合とみなすことができる。一方で、一般的にプログラマは関数あるいはサブルーチンを処理の基本単位と考えて開発を進める。すなわち、ソフトフェアから見るとプログラムは関数の集合と考えられている。このように階層間での考え方にはギャップがあり、並列化や最適化の足かせとなってきた。

マルチグレイン並列処理の観点から並列処理の対象を考えると命令レベル、ループレベル、関数呼び出しレベル、スレッドレベル、タスクレベル、プロセスレベルなど多種多様な単位があり、その並列部分の大きさもそれぞれ異なる。しかしながら、プログラムの実行時間の大半がループ部分で占められているということより、各対象は何らかのループに属していると考えられる。もし、プログラムにループを使わないならばループを展開した膨大な量の逐次コードを用意するという現実的には考えにくいコードとなる。従って、コードを再利用するループ構造はプログラムにとって必要不可欠な存在と考えられる。加えて、ループにもインナーループやアウターループなどといった階層構造があり、ループの大きさも様々である。すなわち、ループは細粒度並列性から粗粒度並列性まで様々な粒度があり、マルチグレイン並列処理にも対応可能な優れた指標と考えられる。

マルチグレイン並列処理において細粒度であるインナーループの並列化技術は実用化の段階まで達しているといえる。例えば、命令レベルのスケジューリングによりループアンローリングやソフトウェアパイプライニングは実現可能であり、近年ではソフトウェアパイプライニングを支援するレジスタローテーションという機能がItaniumプロセッサにはハードウェア実装されている[3]。また、インナーループ部分をSIMD型の並列演算器やベクトルプロセッサを用いて並列化する技術も確立されている。

しかしながら、インナーループ部分だけがプログラムの実行時間の大半を占めているとは限らないため、粗粒度であるアウターループの並列化も高い次元の並列度を抽出するためには必要である[4]。アウターループの並列化は複数のプロセッサに展開されて並列処理されることが一般的であり、多くの場合プログラムを熟知した技術者が経験的に行うことに依存している。従って、粗粒度のループを検出し並列化できるかの判定を含めて自動化することが望まれている[5]。

マルチレイヤ並列処理の観点からもソフトウェアからハードウェアまで様々なレイヤにおける並列処理を最大効率にて行うためには、レイヤ間のギャップをなくし、各レイヤをまたいで見通すことが可能な共通の指標が必要となる。すなわち、マルチレイヤ並列処理環境においては上流のレイヤで得られた有効な情報が下流のレイヤで利用することを見越した情報であることが重要であり、処理の本質に迫る重要な情報をインターフェースとすることが求められる。

ループ構造はプログラマがアプリケーションを記述する段階からコンパイラがバイナリコードを生成したり、ハードウェア上で実際にコードが実行される際まで概念として共有することが可能である。また、ハードウェア上でアプリケーションプログラムのコードが実行される場合の大半の時間をループが占めるため、性能見積もりを行う観点からもよい指標といえる。

以上の理由から、本研究においてはプログラムのループ構造に着目してマルチグレイン・マルチレイヤ並列処理の実現を目指す。

## 4. 予備実験

本節では提案するマルチグレイン・マルチレイヤ並列処理システムを実現する上でループに着目することが有効であることを示すための予備実験を行う。実験には文献[1]で構築したループ構造検出法を使用した。実験を行った環境はSGI Altix350上に構築した。Altix350の構成はIntel Itanium2 CPU、Intel Compiler v8.1、RedHat Linux + SGI ProPack 3SP6である。また、利用した実行時プロファイリングツールであるPinのバージョンは2.3(17159)であり、Itanium用の構成を用いた[6]。評価実験を行うためのベンチマークプログラムとして、SPEC CPU2006ベンチマークから-Oオプションにてコンパイルした401.bzip2とデータセットとしてdryer.jpgを利用し、バイナリの実行の開始から完了までの間に出現するループの検出を行った。

図1に検出された主要なループ構造を示す。図中のIDの列はループの個別番号を、InstAddrの列は昇順に並んだ命令アドレスを示す。その行にある命令が後方分岐命令の場合、B?の列に**B**と表示され、関数呼び出しの場合はIDの列にCと表示される。IDを持ち**B**の表示がない命令がループの先頭命令であり、それと同一のIDを持ち**B**の表示がある命令がそのループの最後尾の命令となる。図中の割合[%]は前回成立した後方分岐命令から今回成立した後方分岐命令まで

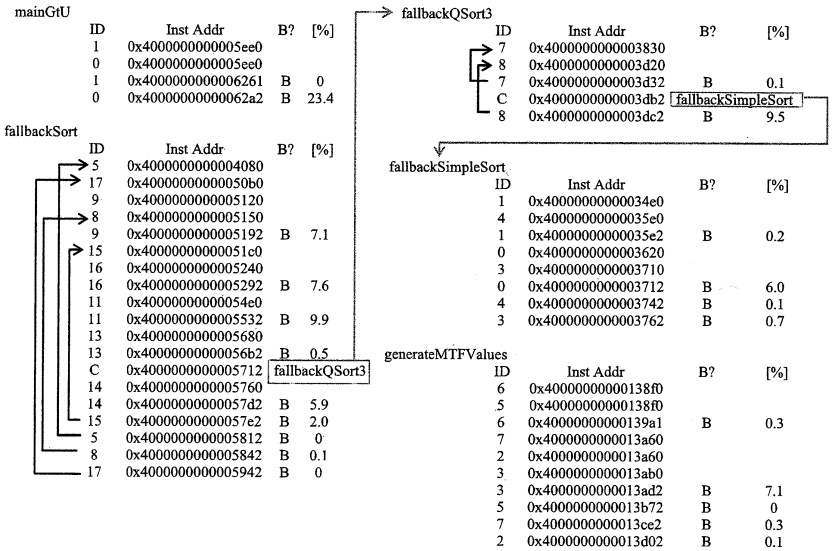


**図 1** 検出された主要なループ構造

に実行された命令数のループ毎の総和がプログラムの全実行命令中に占める割合である。図 1 においては主要なループとして各関数で検出されたループの全実行命令に占める割合の総和が 3%を超える主要な関数内の主要なループに限定して表示した。また、注目するべきアウターループと関数呼び出しに矢印を加えループ構造が見やすいようした。

結果より、今回の bzip2 の実行においては、関数 fallbackSort が階層的なループ構造を持ち、ループ内部で関数 fallbackQSort3 を呼び出していることが分かる。また、関数 fallbackQSort3 はループ内部で関数 fallbackSimpleSort を呼び出していて、関数 fallbackSimpleSort の内部にもループがあることが分かる。以上のことから、ループ階層構造に着目しても様々な粒度の並列性を検出できるといえる。

## 5. まとめと今後の課題

本稿では、ループ階層構造に着目してマルチグレイン並列処理が可能となるであろうという指針を示した。また、ループ階層構造を共通のインターフェースとしてソフトウェアからハードウェアまで様々なレイヤにおいて並列化を行うことの必要性を述べた。また、予備実験の結果からループ階層構造に着目することはマルチグレイン・マルチレイヤ並列処理に有効であることを示した。

今後の課題としては、マルチグレイン・マルチレイヤ並列処理技術を確立するために、現状のシステムの挙動を詳細に再現し理解した後に、システムを最適化や予測を行っていくことが挙げられる。

## 参 考 文 献


1) 佐藤幸紀, 鈴木健一, 中村維男. ループ並列化のためのループ階層構造を検出する実行時プロファイリング手法. 情報処理学会研究会報告 2008-HPC-117, 2008.
2) Mark Oskin. The revolution inside the box. *Commun. ACM*, Vol.51, No.7, pp. 70–78, 2008.
3) Sebastian Winkel, etal. Latency-tolerant software pipelining in a production compiler. In *Proceedings of the 6th annual international symposium on Code generation and optimization*, pp. 104–113, 2008.
4) Arun Kejariwal, etal. On the performance potential of different types of speculative thread-level parallelism. In *Proceedings of the 20th annual international conference on Supercomputing*, p.24, 2006.
5) Tipp Moseley, et al. Identifying potential parallelism via loop-centric profiling. In *Proceedings of the 4th international conference on Computing frontiers*, pp. 143–152, 2007.
6) Chi-Keung Luk, et al. Pin: building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, pp. 190–200, 2005.