Vol.2010-OS-115 No.24
2010/8/4

Hadoop のマルチコア実行における スレッド集約スケジューリングの効果

山田 賢[†1] 日下部 茂[†2]

我々は、メモリアドレス空間を共有するスレッド間の参照の局所性に着目し、そのようなスレッドのマルチコアプロセッサ上での時分割及び空間分割実行を集約的に制御できる、コア間時間集約スケジューラを提案している。提案スケジューラはマルチコアプロセッサの各プロセッサコア間で共有するキャッシュといったメモリの階層を活用し、プログラムの実行性能を向上させることを目的としている。本稿では、MapRdeuce 型プログラミングモデルのオープンソース実装である Hadoop のアプリケーションを、マルチコアプロセッサシステムで実行する際のノード内マルチスレッド処理における提案スケジューラの効果について評価する。TaskTracker が起動する map/reduce タスクの集約実行を制御することで実行性能の向上が観測できた。

Evaluation of Inter-Core Aggregation Scheduler in Multicore Execution of Hadoop Application

Satoshi Yamada[†1] and Shigeru Kusakabe[†2]

We have proposed Inter-Core Time Aggregation Scheduler which tries to utilize the locality of references between sibling threads, which share the same memory address space, on commodity multi-core processors. Our thread scheduler controls both time and space multiplexing of the sibling threads to utilize the cache shared by multiple processing Cores on a multi-core processor in enhancing the performance of multithread programs. In this paper, we evaluate the effect of our scheduler in executing Hadoop application on a commodity multi-core processor. We observed performance improvement by controlling the execution of map/reduce tasks under a TaskTracker.

1. はじめに

本稿では、我々が提案しているスレッド間の参照の局所性を活用し性能向上を目指すコア間時間集約スケジューラ (Inter-core Aggregation Scheduler, 以下 IAS) の、MapRdeuce 型プログラミングモデルのオープンソース実装である Hadoop のマルチコアシステム上でのノード内マルチスレッド実行時の効果について評価する。

スレッドレベルでの並列処理により性能向上が可能である Simulaneous Multi-Threading や Chip Multi-Processing のようなマルチコアプロセッサが普及し、アプリケーションのマルチスレッド化も進んでいる。データマイニングや流体シミュレーションなど、これから主流となると思われるワークロードに対応した DaCapo benchmark[19] や Parsec benchmark[21] といったベンチマークスイーツでも、マルチコアプロセッサ上での実行を念頭に置き、プログラムのマルチスレッド化が取り入れられている。

プロセッサコア (以下コア) の間でキャッシュを共有するマルチコアプロセッサでは、従来の共有メモリ型マルチプロセッサと比較して、コア間で共有する情報の高速な通信が期待出来る。その一方で、コア間で共有するキャッシュの競合が発生した場合には、個々のプログラムのスループットが低下するという問題も知られている[3),8),9)]。この問題に対して、同時に実行するスレッドの組み合わせのスケジューリングによる、性能向上の試みが多く行なわれている。

この中で多く提案されているスケジューリング手法は、実行をスレッドの実行時情報のサンプリングとこの実行時情報に基づいたスケジューリングのフェーズに分割し、2 つのフェーズを繰り返す手法である[7),10),11),15)]。このような手法は理論的にはどのようなプログラムの実行時にも適用可能であるという利点がある一方で、サンプリングに伴うオーバヘッドを考慮する必要がある[16)]。また、実行時情報を得られたとしても、これを基に行なうスケジューリングの最適化問題は、2 コアの場合クラス P、3 コア以上で NP 完全問題であることが知られている[8)]。そのため、我々が想定するような多数のスレッドの同時実行時においては、スレッド情報のサンプリング、及び、スケジューリングのオーバヘッドが高く

†1 九州大学大学院システム情報科学府 (現在, ヤフー株式会社)
Graduate School of Information Science and Electrical Engineering, Kyushu University (Currently, Yahoo Japan Corporation)
†2 九州大学大学院システム情報科学研究院
Graduate School of Information Science and Electrical Engineering, Kyushu University

なり、現実的ではないという指摘がある[3),6),16)]。

このような方式に対し、我々は、マルチコアプロセッサ環境において、マルチスレッドプログラムを実行する場合に、共有キャッシュといったメモリ階層を活用して性能を向上させるスレッドスケジューラである IAS を提案し評価してきた[12)]。我々は、最適なスケジューリングを追求するというより、低オーバーヘッドで軽量ながら有効性の高い手法を目指している。対象としているマルチスレッドプログラムは、同一のメモリアドレス空間を共有するカーネルレベルのスレッド (以下シブリングスレッド) を複数生成し、並行・並列に実行するプログラムである。我々の手法はシブリングスレッド間での参照の局所性を利用することで、例えばキャッシュ上への共有データの集約といったメモリ階層の効果的活用と実行性能の向上を目指している。現在の実装では Linux の標準のスケジューラ (Completely Fair Scheduler, 以下 CFS) をベースに、動的管理情報のために各スレッドのメモリアドレス空間に関する拡張を加える程度で実現できている。スケジューリングの計算量も $O(1)$ で行える。これまでの研究における実測では、IAS の付加オーバヘッドが 1%程度である[18)]。

本稿では、我々が提案している IAS の、マルチコアシステム上での Hadoop アプリケーションのノード内マルチスレッド実行時の効果について評価する。本論文の構成を以下に述べる。第 2 章では、CFS と IAS について説明する。第 3 章では、Hadoop について説明する。第 4 章では Hadoop を用いた IAS の評価を行い、第 5 章でまとめを述べる。

2. コア間時間集約スケジューラ (IAS)

本章では IAS の概要について説明する。前述のように、我々は IAS を Linux の CFS をベースに実装しているが、スケジューラの詳細な実装については文献[17)] に譲る。本章では第 2.1 節で CFS について説明する。次に、第 2.2 節で IAS について述べる。

2.1 Completely Fair Scheduler (CFS) について

CFS は名前の通りスレッド間の公平な CPU 時間の割り当てを意図して実装されており、同時刻に始まった静的優先度が等しいすべてのスレッドに対して、任意の期間 *period* に CPU 時間を等しく割り当てることを目標としている。CFS ではそれぞれのスレッドが CPU を消費した時間である CPU 時間をナノ秒単位でカウントし、この CPU 時間を元に *vruntime* という優先度を計算している。CFS では *vruntime* が少ないほど割り当てられた CPU 時間が少ないことを意味する。そのため、*vruntime* が少ないスレッドの優先度が高くなり、CPU をディスパッチされるとその CPU 時間に応じた値が *vruntime* に加算される。こうしてスレッド間での CPU 時間の公平な割り当てを実現している。CFS ではデフォルトで *period* を 20ms と設定しているが、この値は負荷状況に応じて動的に変化する。CFS ではランキューはコアごとに存在し、各コア上で独立したスケジューラが実行されている。

各コアへのスレッドの割当 (以下ロードバランス) は、ユーザが `sched_setaffinity` などの API を通じてスレッドの実行コアを指定しない限り、各コア上のスケジューラの自律的な判断に基づいて行なわれる。判断の際には、静的優先度に対応してカーネル内で定義された、スレッドの重み (*weight*) を考慮する。各コアではスケジューリングの際に、コアごとのランキュー内のスレッドの重みの和を計算する。計算した結果、コア間での重みの和が偏っており、かつ、他コアから自コアにスレッドを移動させることでこれを解決出来ると判断した際に、スレッドの移動を行なう。これを全てのコアが独立に行ない、コア間での公平な重みの分配を行なう。

CFS におけるスケジューリング、及び、ロードバランスの際には、各スレッドの静的優先度、及び、*vruntime* のみ考慮され、メモリアドレス空間は考慮されない。

2.2 コア間時間集約スケジューラ (IAS)

本節では CFS に基づいた、IAS の概要について説明する。IAS は時間集約とコア間集約という 2 つの概念からなる。時間集約について第 2.2.2 節で説明する。時間集約にコア間集約の概念を追加した IAS について、第 2.2.3 節で説明する。IAS における優先度ボーナス、及び、コア間集約を行なうコアグループの設定方法について第 2.2.4 節で説明する。

2.2.1 コア間時間集約スケジューラ (IAS) の概要

アプリケーションの実行性能の観点からは、キャッシュといったメモリ階層の活用が重要である。また、インテル社の x86 といったプロセッサでは、TLB エントリの無効化をメモリアドレス空間の切り替えを機会にハードウェアが自動的に行う。このようなプロセッサ上では、メモリ空間アドレス空間を共有するシブリングスレッドの集約実行によりメモリアドレス空間切り替え頻度を削減することで、実行性能を向上できる可能性が高い[13)]。一方で、マルチコアプロセッサにおいてはコア間でキャッシュを共有する事が一般的である事から、時分割実行によってスレッド間で共有するデータを共有キャッシュに集約出来る可能性がある。空間分割実行を行なう際にも、上述したようにキャッシュを共有するコア間で同時に実行されるスレッドの組み合わせが性能に大きく影響する。そのため、同時実行の際になるべく性能が落ちないように親和性のあるスレッドの組み合わせを選択する必要がある。

IAS は汎用のマルチコアプロセッサ上においてプロセッサコア空間と時間軸方向とでの分割実行の制御を支援する。我々はシブリングスレッドの参照するデータ領域は近接、もしくは共有していることが多いと考える。そのため、時間軸方向での分割実行を制御する事でコ

ア間で共有されたキャッシュに参照するデータを集約し、キャッシュミスの削減が出来ると考える。コア間でのデータの競合により、CPU が待たされる可能性も高くなるが、キャッシュとメモリのレイテンシの乖離が進んでいることから、キャッシュミスを削減することが性能向上により大きな影響を与えると考える[4]。但し、コア数が増加し、メモリ階層の構造が複雑化した際には、コアごとのローカルなキャッシュ間でのデータ整合性維持に伴うオーバヘッドが高くなることが予想される。そのため、プロセッサコア空間を意識した制御を同時に行うことで、データ整合性維持に伴うオーバヘッドの削減効果が期待出来る。IAS はスレッドレベル並列処理の最適なスケジューリングを保証するものではないが、マルチコアプロセッサの普及に伴うプログラムのマルチスレッド化を考慮した際に、現実的、かつ、効率的なスケジューラとして有用であると考える。我々は IAS を Linux のスレッドスケジューラである CFS をベースに実装した。IAS は Linux において各スレッドに与えられる優先度を考慮しつつ、シブリングスレッドの時空間での分割実行を $O(1)$ で実行する事が出来る。

2.2.2 時間集約について

時間集約とは、Linux で各スレッドに与えられた優先度を考慮しつつ、単一コア上でシブリングスレッドを集約実行する実行方式である。我々は時間集約を行なうスケジューラを Linux を用いて実装しており、これを時間集約スケジューラと呼ぶ。シブリングスレッドの集約実行を実現するために、時間集約スケジューラでは現在実行されているスレッドのシブリングスレッドに対して動的に優先度ボーナスを与える。第 2.1 節で述べたように、CFS では *vruntime* の値が小さい方が優先度が高い。その為、時間集約スケジューラのための優先度ボーナスは、現在実行されているスレッドのシブリングスレッドの *vruntime* を削減するように作用する。

我々はコアごとにシブリングスレッド同士をつなぐリンクを生成し、スレッドをランキューに挿入する際に、このリンクをスレッドの *vruntime* 順にソートする。リンクの起点はシブリングスレッドが共有するメモリアドレス空間に追加したメンバである。実行中のあるスレッドのシブリングスレッドを探索する際には、このスレッドのメモリアドレス空間からシブリングスレッドの起点を辿り、リンクの先頭のスレッドを参照出来る。そのため、時間集約候補のスレッド探索に費やす計算量は $O(1)$ となる。

2.2.3 時間集約へのコア間集約の追加

IAS は時間集約にコア間集約を付加することで実装する。具体的には、あるコア (`master` コア) 上で時間集約されているスレッドのシブリングスレッドが異なるコア (`slave` コア) に存在する場合に、これを同時に時間集約させる。そこで、まず `master` コアが時間集約を行った際に、カーネル内のフラグ (`ia_mm`) に集約するスレッド同士が共有するメモリアドレス空間へのポインタを格納する。`master` コア上で時間集約が行われていない場合、`ia_mm` は NULL である。`ia_mm` の内容を変更できるのは `master` コアのみで `slave` コアは `ia_mm` を参照するだけである。`ia_mm` が `master` コアによってあるメモリアドレス空間にセットされたとき、`slave` コア上で動くスケジューラは `ia_mm` が指すメモリアドレス空間を共有するスレッドを、各自のコアのランキューから探す。`ia_mm` が指すメモリアドレス空間を共有するスレッドが見つかった場合、そのスレッドを次に実行するスレッドの候補と見なす。`slave` コア上のコア間集約候補のスレッドは、そのコアで一番優先度が高いスレッド、及び、時間集約候補スレッドに対して優先度ボーナスを持つ。

スレッドの各コアへの分散に関しては、第 2.1 節で示した CFS のロードバランス機能を用いる。上述したように、CFS ではロードバランスの際にメモリアドレス空間を考慮せず、また、一般的なプログラムではデフォルトで全てのコアで実行を許すように設定されている。そのため、シブリングスレッドが各コアに分散され、コア間時間集約スケジューラが活用されることが期待できる。`master` コア、`slave` コア、`ia_mm` の組はカーネル内に複数存在可能で、任意のコアの組み合わせによってコア間時間集約が出来るように実装してある。コア数が増加し、多くのマルチスレッドプログラムを同時に実行する際には、コア間集約の組み合わせを複数設定することで、単一プロセッサコア上において空間分割実行が期待出来、コア間通信のオーバヘッドを削減することが出来る。

2.2.4 優先度ボーナスと `master/slave` コアの設定

IAS の優先度ボーナス、及び、`master/slave` コアの設定は、専用の API を用いて行なう[18]。第 2.1 節で述べたように、*vruntime* はナノ秒単位で計測した各スレッドの CPU 時間から計算される値であり、ユーザが設定するには粒度が細かすぎるという問題がある。我々は、スレッドが一度 CPU を与えられた際の CPU 時間 (クォンタム時間) を基準にそれをスケールさせる形で優先度ボーナスを与えることで、容易、かつ、効果的にプログラム実行の性能を向上させることができることを確認している[18]。これにより、各スレッドによって異なるクォンタム時間の消費傾向を優先度ボーナス値に反映出来るほか、ナノ秒単位で優先度ボーナス値を考慮する必要がないという利点がある。各スレッドのクォンタム時間の計測は Linux の CFS に既に実装されている機構をそのまま使っており、付加されるオーバヘッドもほとんどない。我々が実装した IAS 専用の API を用いることで、各スレッドのクォンタム時間をスケールさせた値を優先度ボーナスとして与えることが出来る。

コア間集約を行なう `master/slave` コアについても、IAS 専用の API を使うことで任意

に設定が可能である。

3. Hadoop の概要

本稿は、Hadoop の計算ノード内部でのスレッドレベルの並列・並行処理における IAS の効果を対象としており、その観点を中心に、まず Hadoop のベースである MapReduce プログラミングフレームワークについて、次に Hadoop について簡単に紹介する。

3.1 MapReduce

MapReduce は、一つのマシンで処理できないような大量のデータを多数のマシンで構成された計算クラスタ上で効率よく分散処理するために、Google が開発したプログラミングフレームワークである[5)]。MapReduce は関数型言語の Map 関数と Reduce 関数にヒント得たとされるが、焦点はリストの再帰処理といったものではなく、大量のデータを分割したデータチャンクに対する並列分散処理を、マスタノードと多数のワーカノードによって行う点にある。MapReduce プログラミングフレームワークでの計算のジョブは、大きく分けて Map フェーズと Reduce フェーズからなり、それぞれのフェーズで多数のワーカノードによる大規模分散並列処理が可能である。

Map フェーズでは、マスタノードが受け取った大量の入力データを、小さな単位に分割し、複数のワーカノードに配置する。各ワーカノードは、受け取ったデータから必要な情報を求める処理を行い結果を返す。Reduce フェーズでは、Map フェーズでワーカが計算し出力する結果を集計し、最終的にはマスターノードが処理結果を得る。フェーズ間でのデータのやり取りは KeyValue ペアと呼ばれる単位で行う。Map フェーズのタスクが出力した KeyValue ペアは Reduce フェーズの始めに Key ごとに集約されてから Reduce フェーズのタスクに渡される。各フェーズ内のタスクはお互いに通信することがなく、大量のマシンで効率よく並列分散処理できるように設計されている。各ワーカの Map 処理は他のワーカの Map 処理と独立で、概念的には各ワーカに分配した全てのデータを並列に処理することが可能である。Reduce フェーズでもデータの依存関係にしたがった並列処理が可能である。また、本稿では省略するが、Google では MapReduce の他に、巨大なデータを分散して格納するための大規模分散ファイルシステムやその上で動作する大規模分散データベースなども用意している。

3.2 Hadoop

Hadoop は Apache Hadoop プロジェクト[20)] で、Google の MapReduce や GFS などの分散処理技術に対するオープンソースのプロジェクトとして開発が進められている。Hadoop の MapReduce フレームワークは Java で実装されており，Hadoop での map 関数や reduce 関数も Java で記述するが、Hadoop Streaming と呼ばれるモジュールを用いることで標準入出力が扱えるすべての言語で map 関数や reduce 関数を記述することもできる。

Hadoop では、サポートするファイルシステムへのアクセスといった基本機能を Hadoop Common が提供し、その上で MapReduce エンジンが動作する。MapReduce エンジンはひとつの JobTracker を持ち、クライアント・アプリケーションはこの JobTracker に向けて map/reduce ジョブを投入する。ジョブが投入されると、JobTracker はクラスタ中の利用可能な TaskTracker に仕事を依頼し、TaskTracker が個々の map / reduce の処理を起動する。TaskTracker が一つの計算ノード内で起動する複数の map / reduce は、TaskTracker を親とするシブリングスレッドで実行され、IAS の制御対象となり得る。本実験では、この点に着目し IAS の効果を評価する。

4. 評　　価

本章では Hadoop アプリケーションの実行における IAS の効果について、Hadoop アプリケーション単独実行と DaCapo ベンチマークとの並行実行の場合について評価する。スレッド処理の負荷を高めた場合の測定も行うために、DaCapo ベンチマークの中でスレッドを用いて内部実行の並列化を行っているアプリケーションと、Hadoop と並行して実行させる。

4.1 評 価 方 法

評価環境を表 1 に示す。評価の際には Linux 2.6.24 をベースに IAS の拡張を行った実装を用い、IAS 拡張を有効／無効にして評価を行った。IAS 拡張を無効にした場合、前述のわずかなオーバヘッドを除きほぼ CFS と同等である。Hadoop のバージョンは 0.20.2 を用いた。TaskTracker が個々の map / reduce の処理を起動する。TaskTracker が一つの計算ノード内で起動する複数の map / reduce は、TaskTracker を親とするシブリングスレッドで実行され、IAS の制御対象となり得る。本実験では、この点に着目して IAS の効果を Hadoop の疑似分散モード実行で評価した。

評価対象の Hadoop のアプリケーションには、配布されている Hadoop に付属の、円周率計算 pi、 grep と sort (1GB)、streaming 実行によるパケットログ解析の独自 Haskell プログラムを用いた。測定は、time コマンドを用いた実行時間の計測と、カーネル内に実装したカウンタを用いて、IAS の時間集約、及び、コア間集約の回数のカウントを行なった。

表 1 評価環境
Table 1 Specification of our experimental platform

プロセッサ	Intel Core i7 2.67GHz 4
L2 / L3 キャッシュ	256KB x 4 / 8MB
メモリサイズ	8 GB
OS / カーネル	CentOS 5.3 / Linux 2.6.24
Hadoop	0.20.2

4.2 DaCapo ベンチマークについて

DaCapo は Java ベンチマーキングのためのベンチマークスイーツであり、SPECjvm よりも現実的かつ汎用性のあるベンチマークを目指しており、オープンソースで、自明でないメモリ負荷を持つ実世界の複数のアプリケーションの集合である[19]。以前の 2006 年版からの大きな更新となる 2009 年版では、より多くのマルチスレッドアプリケーションが含まれている。本評価では、DaCapo ベンチマークの中で、スレッドを用いて内部実行の並列化を行っているアプリケーションを Hadoop と並行して実行させることで、スレッド処理の負荷を高めた場合の測定も行った。

4.3 Hadoop の実測結果と考察

表 2 に、Hadoop アプリケーションを用いた評価結果について示す。IAS の機能を使わない通常の Linux スケジューラである CFS 版での実行と、IAS 版での実行時間、前者を後者で割った速度向上比、および IAS での実行時のカーネル内の集約実行のカウンタ値を示す。CFS 版も含め、同一の実験設定でも複数試行で実測時間が異なった場合は、最も良い結果を示している。

傾向としては、表に示した Hadoop アプリケーションでは、実行時間が短くなっており IAS の効果があったといえる。特に、DaCapo ベンチマークを並行して実行し、スレッド処理の負荷が加わっている状況では効果が高いと言える結果が観測された。今回は grep のみでしか、DaCapo ベンチマークとの同時実行は行っていないが、今後はより多くの単体アプリケーションでの評価に加え、多数の組み合わせの負荷での評価を行う予定である。

また、単体ノードでの疑似分散モード実行に加え、実分散モードのクラスタ構成の計算ノードでの効果も測定する予定である。

実行時間に加えて、IAS 版では実際に集約実行を行った回数を計測したが、実際の速度向上との関係を確認するには至っていない。我々のスケジューラの効果はプログラムのスレッドの挙動によるところが大きく、マルチスレッド化されたプログラムであっても集約実行によりその性能が下がる場合があるという問題がある[1]。今後は、Hadoop アプリケーションでの効果について、より低レベルの内部イベントのカウンターや、より詳細なデータが採取できるプロファイルツールを用い、より詳細な分析を行う予定である。

表 2 Hadoop アプリケーションの実行時間と IAS スケジューラの効果
Table 2 Effect of IAS on Hadoop Application

	CFS 版 (秒)	IAS 版 (秒)	速度向上比	時間集約数	空間集約数
Hadoop 単独					
pi	45.52	42.94	1.06	46	1
grep	87.14	77.63	1.12	42	3
sort	144.91	104.33	1.39	2980	42
streaming	538.55	506.33	1.06	1117	23
Hadoop&DaCapo					
grep	117.34	77.71	1.51	76 (IAS 全体)	2 (IAS 全体)
DaCapo	111.83	87.71	1.27		

性能低下の問題は I/O の待ち時間を隠蔽するためにスレッド化したものの、扱うデータのサイズが大きいために、コア間でメモリの競合が発生しやすいプログラムなどで顕著と考えられ、IAS の適切なパラメータ設定などでこの問題を回避する必要がある。Hadoop のアプリケーションでもこのパターンに当てはまるものがあると考えられる。マルチコア環境におけるメモリ階層の活用を目指したスレッド化に関する研究が行われており、そのようなマルチスレッド化の技術が向上すれば我々のスケジューラでの効果的なパラメータ設定に有効と期待できる。例えば、マルチコア環境におけるスレッド間の適切なデータ共有をチェックするツール[2]、関連性のあるスレッドをコア間で集約する際に、共有キャッシュ活用という観点から効果的なスレッドの粒度を決定するコンパイラ[4]、といったものがある。

5. おわりに

本稿では、メモリアドレス空間を共有するスレッド間の参照の局所性に着目し、そのようなスレッドのマルチコアプロセッサ上での時分割及び空間分割実行を集約的に制御できる、コア間時間集約スケジューラの効果について、Hadoop アプリケーションに対する評価を行った。TaskTracker が起動する map/reduce の処理が、TaskTracker とメモリアドレス空間を共有するスレッドとして実行されることに着目して集約実行を制御することで実行性能の向上を観測した。

これまでに、我々は IAS によるシブリングスレッドの集約効果をいくつかのベンチマー

クを用いて検証してきた[12),17)]。その結果、IAS の効果はシブリングスレッド間で共有するデータのサイズやデータへのアクセスパターン、コア間で共有するキャッシュサイズに大きく依存する事がわかっている。今後は、より多くの Hadoop アプリケーションや、コンフィグレーションに対して評価を行い、同時により詳細な内部イベントのデータを収集し、分析する予定である。アプリケーションや設定ごとに IAS の効果は異なるが、IAS はユーザによるパラメータ設定が可能であり、その設定の指針も確立することを目指す。さらに、IAS の優先度ボーナスを自律的に最適化する機構の構築を目指す。我々は IAS を実行するプラットフォーム上で、L2 キャッシュミスなどのイベント数やスループットなどを計測し、計測結果から優先度ボーナスを変更するヘルパースレッドを同時に実行し、その効果を検証することも目指す。

参 考 文 献

1) J. Anderson, et al., "Parallel Task Scheduling on Multicore Platforms," *ACM SIGBED Review*, Volume 3, Issue 1, The work-in-progress (WIP) session of the RTSS 2005, pp.1-6, 2006
2) Z. Anderson, et al., "SharC: Checking Data Sharing Strategies for Multithreaded C," *Proc. of the 2008 ACM SIGPLAN*, pp.149–158, 2008
3) D. Chandra, et al., "Predicting Inter-Thread Cache Contention on a Chip Multi-Processor Architecture," *Proc. of HPCA*, pp.340 - 351, 2005
4) S. Chen, et al., "Scheduling Threads for Constructive Cache Sharing on CMPs," *Proc. of ACM Symposium on Parallel Algorithms and Architectures*, pp.105–115, 2007
5) J. Dean, and S. Ghemawat . Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.
6) M. DeVuyst, et al., "Exploiting Unbalanced Thread Scheduling for Energy and Performance on a CMP of SMT Processors," *Proc. of International Parallel and Distributed Processing Symposium*, 2006
7) A. Fedorova, et al., "Performance of Multithreaded Chip Multiprocessors and Implications for Operating System Design," *Proc. of USENIX 2005 Annual Technical Conf*, pp.395-398, 2005
8) Y. Jiang, et al., " Analysis and Approximation of Optimal Co-Scheduling on Chip Multiprocessors ", *Proc. of PACT*, pp.220-229, 2008
9) S. Kim, et al., "Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture," *Proc. of PACT*, pp.111-122, 2004
10) S. Parekh, et al., " Thread-Sensitive Scheduling for SMT Processors ", *Technical Report, Univ. Washington*, 2000
11) A. Snavely, et al., "Symbiotic Jobscheduling with Priorities for a Simultaneous Multithreading Processor," *Proc. of International Conference on Measurement and Modeling of Computer Systems*, pp.66-76, 2002
12) S. Yamada, et al., "Development of a Thread Scheduler for Global Aggregation of Sibling Threads," *Research Reports on Information Science and Electrical Engineering of Kyushu University*, Vol. 1, No. 2, pp.69-74, 2008
13) S. Yamada, et al., "Effect of Context Aware Scheduler on TLB," *Proc. of Workshop on Multi-Threaded Architectures and Applications*, 2008
14) S. Ziemba, et al., " Analyzing the Effectiveness of Multicore Scheduling Using Performance Counters ", *Proc. of Workshop on the Interaction between Operating Systems and Computer Architecture*, 2008
15) 小川周吾, et al., "プロセスの実行時情報を用いたスケジューラによる高速化手法、" Vol.46 No.SIG 12 (ACS 11), pp.161-169, 2005
16) 近藤正章, et al., "トラクションコントロール実行：CMP 向けプロセス実行制御方式の提案" Vol.1 No.2 12 (ACS), pp.111-123, 2008
17) 山田賢, 日下部茂: CMP 上でスレッド間の参照の局所性を活用するコア間時間集約スケジューラの評価, コンピュータシステムシンポジウム (ComSys2009) 論文集, Vol. 2009, No. 13, pp. 139-148, 2009
18) 山田賢, 日下部茂: コア間時間集約スケジューラ活用のための API の実装と評価, 情報処理学会プログラミング研究会 (PRO-2009-3), 2009
19) "DaCapo benchmark suite," http://dacapobench.org/
20) Hadoop. http://hadoop.apache.org/.
21) "The PARSEC Benchmark Suite", http://parsec.cs.princeton.edu/