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

2004－ARC－159 (1)
2004／7／30

プロファイルを利用した値の局所性による高速化手法

平澤将一† 平木 敬†

アプリケーションを並列に実行することは高速化に有効であるが、逐次計算プログラムを並列実行するためにはプログラマが書き直すか自動並列化コンパイラを使う必要がある。一般的な整数演算プログラムに対して自動並列化コンパイラが静的に内在する並列性を抽出することは困難であり、十分な高速化を達成することはできていない。プログラムの計算には同じ値のデータを用いて同じ計算結果を示す値の局所性がある。我々は値の局所性を利用することで動的に投機的並列性を作り出し、より多くの計算資源を活用することを提案した。異なる実行パスを同時に実行することが必要であり、グローバル変数へのアクセスを排除する必要があったため適応範囲が限定的だった。本研究においてはグローバル変数への書き込みをプログラムスライスとして分離し、書き込みの値の局所性を示しこれを利用した時の性能測定を行った。

Performance Enhancement with Profiled Data Value locality

SHOICHI HIRASAWA and KEI HIRAKI

To execute applications in parallel is an effective scheme to speed up them. When we want to execute a sequential program in parallel, we have to rewrite the program code heuristically or convert it into a parallelized one with an automatic parallelizing compiler. It is difficult to obtain enough parallelism in general non-numerical programs with automatic parallelizing compilers. There are value localities in the variables of programs. We propose to use speculative parallelism to execute such applications in parallel. Applicable functions are limited because only plain functions that do not have side effects can be executed with the method. In this study, we represent value localities in writings of global variables by profiling and devide them as a program slice from original functions and evaluate the performance.

1. はじめに

コンピュータはますます高速化の一途をたどっているが、アプリケーションの実行時に示す様々な局所性を利用することが実行の高速化に大きな役割を果たしている。メインメモリへアクセスする時に空間的な時間局所性を利用するデータキャッシュやディスクアクセスの局所性を利用するディスクキャッシュ、プロセスの利用率及びIOアクセスの局所性を利用したOSのスケジューリング、通信の混雑度の局所性を利用したネットワークなどは局所性の利用が実用化されている例である。一方、プログラムの実行時に行われる計算で用いられるデータおよび結果には値の局所性[8]が存在する。値の局所性を利用して計算に用いるデータを予測して投機実行[7]することや、計算結果を記録して再利用[1),9)]することが提案されてきた。

CPUはこれまで命令レベルの並列性を活用して高速化してきたが、スーパースカラにより得られる命令レベル並列性は限界があり、また発熱及び消費電力の問題が性能の限界を支配しつつある。CMP(Chip Multi Processor)やSMT(Simultaneous Multi Threading)によりスループット性能を向上させる方法が一般的となりつつあり、さらなる高速化を達成するためにはアプリケーションの並列実行が重要である。並列性を考慮して書かれたアプリケーションは計算資源を有効活用できるが、現存する逐次プログラムを並列実行させるためにはプログラマが発見的にプログラムのソースを書き直すか自動並列化コンパイラ[3]を使う必要がある。前者はプログラマが並列化可能な部分を発見し、逐次プログラムと同一結果が得られるようソースを書き直す困難な作業が必要である。実行の正しさはプログラマによる書き直しの正しさに依存する。後者はコンパイラが対象プログラム中の並列化可能な部分を探し出しプログラムの正しさを壊すことなく並列化を行う必要がある。暗黙的な並列性をコンパイラが静的に発見することは難しく、一般的な整数演算プログラムに対し十分な並列化を行うことは困難である。

自動並列化コンパイラによる並列化が困難なプログラムに対して、データの依存関係を超えて並列に実行させる投機的並列性[2]を用いることで利用可能な並列性を増やすことができる。データの依存関係があるため逐次に実行する必要がある処理を投機的に実行す

† 東京大学大学院情報理工学系研究科コンピュータ科学専攻

ることがプログラムに動的な並列性を導入するためである。

データ値予測[7]を用いると、データの依存関係がある処理を並列に投機実行できるためパフォーマンスの向上に寄与できる。特にプログラム中の関数やメソッドの返り値を予測することによる投機実行は、成功率を高く保つことが可能である場合には対象とする処理の実行時間が長いため大きなパフォーマンスの向上[10]をもたらすことができる。現在までに提案された投機実行方式では、投機実行の成功率を高めるためにより多くのデータを用いて詳細に予測処理を行う必要があるため計算時間がかかり、また成功率の向上には限界があった。このことは投機的状態を巻き戻すオーバヘッドとともにアプリケーションの性能向上を阻害した。

我々はプロファイリングによりアプリケーションの実行時に現れる値の局所性を利用して投機的並列性を作り出し、より多くの計算資源を有効に活用することを提案した[4]。また、そのための実行方式として過去の履歴を用いる再利用と処理のオーバヘッドを低減する投機実行を組み合わせた投機的再利用を提案した。投機的再利用においては複数の実行パスを並列に実行するため、グローバル変数へのアクセスによる副作用を排除する必要があるため適応範囲が限定的であることが問題点であった。本研究においてはプログラム中のグローバル変数への書き込み時の値の局所性をプロファイルにより調べ、プログラムスライスを作ることにより分離させた場合の実行パフォーマンスを測定した。

本論文では、第 2 章で投機的再利用の概要と問題点、第 3 章でプログラムスライスの作成および切り出し、第 4 章で実験方法及び結果、第 5 章で関連研究、第 6 章で結論を述べる。

2. 投機的再利用

本章では我々の提案しているソフトウェアによる性能向上手法である投機的再利用について述べる。

2.1 再 利 用

プログラム中の処理に用いられる値と処理の結果には値の局所性が存在する[8]。データ値再利用[1),9)]は計算で用いたデータおよび計算結果を履歴として記録し、再度同じデータによる計算が行われた場合に記録されているデータを返すことで性能を向上させる方法である。過去行われた計算の結果を用いるため正しい結果が得られるが、用いるデータを全て記録し結果の検索に用いるため処理自体に時間がかかる。

プログラム中に実行ブロック A と B があり、図 1 のように A を対象に再利用を行う場合を考える。過去同じ計算が行われていたために再利用が成功した場合 (Hit) は、記録されていた結果を用いて後続の実行ブロック B の計算をはじめることができる。一方、再利用が失敗した場合 (Miss hit) は実行ブロック A の計算を実際に行う必要がある。

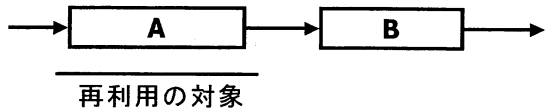

図 1 再利用を行う。成功と失敗により分岐する。

再利用の処理結果が成功であった場合と失敗であった場合で後続処理が分岐する。分岐の結果を予測することによりこれらの処理の投機実行を行うことができる。

2.2 失敗の予測による投機実行

再利用が「失敗する」と予測して投機実行を行った場合は、再利用の処理と投機実行を行う処理にデータの依存関係がないため図 2 のように自明に並列化できる。再利用の処理と実行ブロック A には依存関係がないためやり直し処理は不要である。

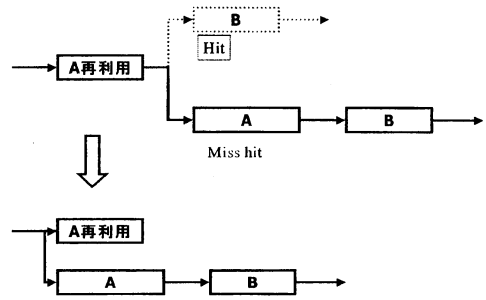

図 2 失敗の予測による投機実行。データの依存関係なし。

2.3 成功の予測による投機実行

再利用が「成功する」と予測して投機実行を行った場合は図 3 のようになる。実行ブロック B は再利用の処理結果を用いて計算を行うためデータの依存関係がある。

データ値予測により処理 B を投機的に実行した場合、データ値予測がはずれた場合のやり直し処理が必要である。

2.4 投機的再利用

再利用の「成功」と「失敗」のどちらか一方のみを予測して投機実行を行う代わりに、2 値である分岐結果の両方を予測して再利用の結果による分岐後の処理を投機実行すると図 4 のようになる。再利用に加えて投機実行を組み合わせているため、以後この実行方式を投機的再利用と呼ぶ。

投機的再利用では、プログラム処理の中で関数呼び出しを始める時に再利用を行うための履歴バッファを探索する処理が始まる。同時に、関数の実際の計算を

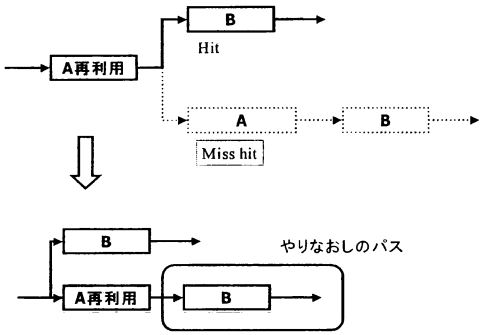

図 3　成功の予測による投機実行。データの依存関係あり。

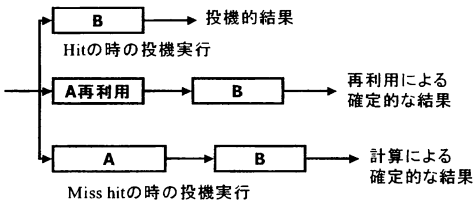

図 4　再利用の成功、失敗の両方で投機実行を行う投機的再利用

並列に開始させる。再利用の処理が終わり結果が成功であった (履歴バッファに過去行われた計算の結果があった) 場合は履歴バッファに記録されていた関数の計算結果を返し、失敗であった (履歴バッファに過去行われた計算の結果がなかった) 場合は実際に関数の計算を行っている処理の終了を待ち計算結果を返す。履歴バッファの探索が失敗に終わった場合は、並列に対象関数の計算が行われているためオーバヘッドである履歴バッファ探索処理の時間を軽減できる。

2.5　投機的再利用の具体的実現方法

提案する投機的再利用は、対象とする関数を決定して履歴バッファを用意し、対象の関数に対する呼び出しがあった時に以下の処理を行うことにより実現できる。

(1)　与えられた引数全て用いてこれをキーとし、ハッシュ表である履歴バッファの探索を始める。
(2)　与えられた引数で対象関数の実際の計算を並列に開始する。
(3)　引数を 1 つだけ用いるデータ値予測を行うことにより後続処理の投機実行を開始する。
(4)　履歴バッファがヒットし、かつデータの予測が正しかった場合は投機的に行っていた後続処理の結果を用いる。データの予測が正しくなかった場合は履歴バッファのデータを返す。
(5)　履歴バッファがミスヒットした場合は実際に計算した結果を返す。この時データを履歴バッファに入れる。

データ値予測を行って投機実行をした結果が間違っていた場合は再利用による履歴バッファ中に記録されていた値を返すことができる。

投機実行に用いるデータの予測が間違っていた場合は履歴バッファ中に記録されていた値を再利用により返すことができる。履歴バッファ中に再利用できる値がなかった場合は、対象関数の実際の計算による結果を利用することができるためプログラムの正しい計算が保証される。

2.6　投機的再利用の問題点

投機的再利用においては再利用処理、対象関数の実行、対象関数の後続処理の投機実行を並列に実行する。対象処理に副作用がない場合は実行の失敗時にロールバック処理が必要ないため並列処理以外のオーバヘッドがかからない。しかしながら対象が副作用のない関数に限定されるため従来は関数をグローバル変数へのアクセスをする関数と副作用のない純粋な関数に手作業で分ける必要があることが問題点であった。

本研究では従来手作業で行っていたグローバル変数へのアクセスを分離する処理を自動的に行うことを目的に、グローバル変数への書き込みを行うプログラムスライス[11] を分離することを提案する。グローバル変数への書き込みを行う副作用をもつ関数からこの書き込みを取り除くことにより引数と返り値のみを持つ純粋な関数ができる。一方、正しい計算を得るためには別の計算により正しくグローバル変数への書き込みを行う必要がある。グローバル変数への書き込みを取り除くことによりできた関数と重複する計算があった場合にはこの重複がオーバヘッドとなる。

3.　プログラムスライスの切り出し

プログラムスライスとは、プログラム中のある変数を対象とした場合その変数の値を計算するために必要な処理である。以下の関数 function はグローバル変数 globalVal への書き込みを行うため、投機実行の対象として実行する場合にはロールバックが必要となりオーバヘッドがかかる。

```
int globalVal;
int function (int a, int b, int c) {
    int ret=0;
    int tmp=0;
    for (int i=0; i<c; i++) {
        ret = ret + a*b;
        tmp = tmp * (a + b);
    }
    globalVal = tmp;
    return ret;
}
```

そこで変数 globalVal への書き込みに注目し、globalVal の計算に用いられている処理をプログラムスラ

イスとして切り出すことにより以下の関数が得られる。

```
void global_slice (int a, int b, int c) {
    int tmp=0;
    for (int i=0; i<c; i++) {
        tmp = tmp * (a + b);
    }
    globalVal = tmp;
}
```

一方、関数 function から globalVal への書き込みを行うプログラムスライスを削除することにより以下のような副作用を持たない純粋な関数を作ることができる。

```
int plain_function  (int a, int b, int c) {
    int ret=0;
    for (int i=0; i<c; i++) {
        ret = ret + a*b;
    }
    return ret;
}
```

以上から、グローバル変数への書き込みがある関数からグローバル変数への書き込みを行うプログラムスライスを除くことができれば副作用のない純粋な関数へ変換できる。ただし、グローバル変数への書き込みを行うプログラムスライスと副作用のない純粋な関数で重複した計算が存在する場合にそれぞれを別に実行することはオーバヘッドとなる。例においてはループインデックスの i の計算が重複する計算である。グローバル変数への書き込みに値の局所性が存在すれば再利用を用いてグローバル変数への書き込みを行うプログラムスライスの実行を高速化でき、オーバヘッドを軽減することができる。

我々は Java ベンチマークプログラムの実行中にメソッドが引き起こす副作用であるグローバル変数へのデータの書き込みについて値の局所性が存在するかプロファイルにより調査した。グローバル変数への書き込みについてプログラムスライスを作成し、メソッドから分離することにより実行を行い性能を評価した。グローバル変数へのデータの書き込みに値の局所性が存在した場合はデータ値再利用を行うことにより、分離実行した場合のオーバヘッドを軽減することが可能であること示す。

4. 評価方法

我々は Java Grande Forum(JGF) ベンチマークプログラムの各プログラムの実行中に Java のクラス変数へのデータ書き込みにおいて生じる値の局所性をプロファイルにより測定した。

Java Virtual Machine Profiler Interface(JVMPI) 及び Java Virtual Machine Debug Interface(JVMDI) を利用したプロファイラを開発し、実行時に対象ベンチマークプログラムのクラス変数へのデータ書き込みの値を記録した。

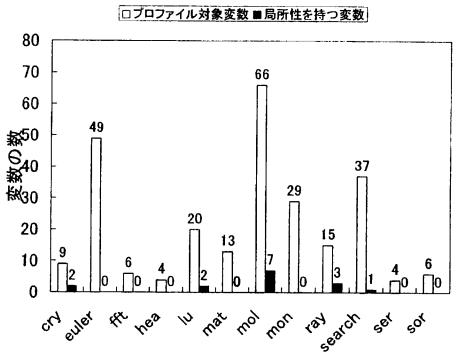

図 5 各アプリケーションプログラムのグローバル変数の数と書き込まれる値に局所性を持つ変数の数

プロファイルにより各ベンチマークプログラムで実行時にクラス変数への書き込みのデータ値にどの程度値の局所性があるか評価する。

実験に用いた計算機環境は Solaris9、Sun Fire V880(Ultra Sparc III 900MHz x 8 40GB)、Sun JDK 1.4.2_04(client VM) である。

4.1 データ値の局所性

各ベンチマークプログラムについてクラス変数の総数と、全履歴を記録していた場合に同一の値が現れる割合が 50% 以上である高い局所性を示す変数の数を図 5 に示す。それぞれのアプリケーションプログラムについて左の棒グラフはアプリケーションプログラム本体に存在するクラス変数の総数、右の棒グラフは全履歴を記録していた場合に同一の値が現れる割合が 50% 以上存在した変数の数である。cry、lu、mol、ray、search にそれぞれ高い局所性を示したクラス変数が存在し、他のアプリケーションプログラムには局所性を示したクラス変数は存在しなかったことがわかる。

次に、高い局所性を示したそれぞれのクラス変数が持つ値の局所性と、値を書き込まれた回数を測定した。図 6 は各アプリケーションの中の高い局所性を示した変数についてその実行回数 (折れ線グラフ、右軸、対数グラフ) および書き込まれた値で回数が最も多い 10 個が全実行回数に占める割合 (棒グラフ、左軸) を示している。トップ 10 の値でほぼ 100% を占める変数がある一方、全履歴を記録していた場合に高い局所性を示しながら実行回数が非常に多いことから 10 個の値で占める割合が非常に小さい変数があることがわかる。

グローバル変数への書き込みをプログラムスライスとして切り出して実行した場合のパフォーマンス測定

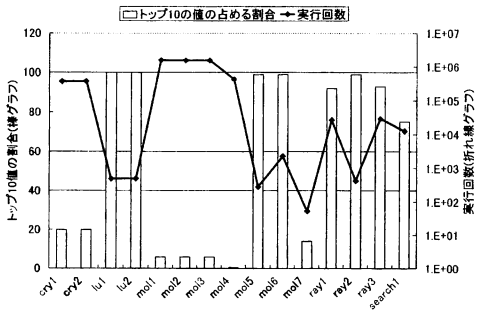

図 6 各クラス変数の実行回数 (折れ線、右軸) およびもっとも書き込まれた回数が多い 10 個の値が占める割合 (棒グラフ、左軸)

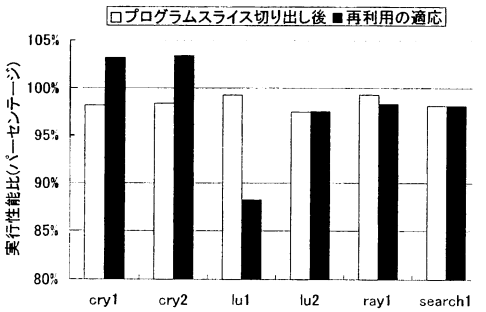

図 7 オリジナルのパフォーマンスに対する、グローバル変数への書き込みをプログラムスライスとして切り出して実行した場合の性能比、さらに再利用を適応した場合の性能比

を行った。プログラムスライスはグローバル変数への書き込みを行うメソッド内に限定し、メソッドをまたがった依存関係の解析は行っていない。また、対象は1個のメソッドのみからアクセスが行われるグローバル変数のみに限定し、複数のメソッドからアクセスされる変数は対象外とした。また、切り出したプログラムスライスに対して再利用を利用したオーバヘッドの軽減を適応した場合の実効性能比を示した。結果を図7に示す。グローバル変数への書き込みをプログラムスライスとして切り出して実行した場合にメソッド呼び出しのオーバヘッドや、共通コードの実行によるオーバヘッドにより性能が下がっていることがわかる。一方、再利用を行うことによりオーバヘッドが低減される。特にに cry1、cry2 ではオリジナルの実行より性能が向上していることがわかる。一方、lu1、search1 は性能が変化していないが、これはグローバル変数の値が他の変数ではなく自分自身の値に依存しているため再利用が行われていないことが原因である。

性能の変化が数%にとどまっている原因は切り出したプログラムスライスが小さいことや、実行回数が少ないことからプログラムの変更が総実行性能に与える影響が少ないためである。

5. 関連研究

マルチスレッド環境で暗黙的なスレッドレベル並列実行を行うことにより自動的にパフォーマンスを向上させる方法として speculative precomputation[6] がある。メモリレイテンシの問題はプロセッサの高性能化において重大な問題となっており、ポインタを多用するアプリケーションではプリフェッチが困難である。シングルスレッドアプリケーションでは有効に使われない余剰ハードウェアでアドレスの計算を行う部分をプログラムスライスとして切り出しスレッド実行し、あらかじめキャッシュミスさせておくことによりメインスレッドのキャッシュミスを隠蔽する。しかしながらプリフェッチを行うだけで、計算結果は利用しない点が本研究と異なり、計算結果を用いることでさらなる性能向上を得られる。

メソッドレベルで投機実行を行い、返り値をデータ値予測を行うことによりパフォーマンスの向上を目指す提案がなされている[5]。予測アルゴリズムには last value prediction 及び stride value prediction、parameter stride によるデータ値予測を用いているが、データ値予測は処理にかかる時間が短く失敗した場合のオーバヘッドが小さいが、成功率を上げることが困難である点が本研究で用いているデータ値再利用と異なる。Stefan ら[2] はスレッドレベルでデータ値予測による投機実行の評価を行った。対象とするスレッドは本研究で対象としたプログラム中の関数と比較して粒度が細かく、実行にかかる時間が短い。また、現実的な問題においてハードウェアによる支援を仮定している。本研究においてはソフトウェアのみによる並列実行を目指し、プログラムの実行中に現れるデータ値の局所性を利用している。

6. 結　論

投機的再利用においては複数の実行パスを並列に実行するため、グローバル変数へのアクセスによる副作用を排除する必要があり適応範囲が限定的であることが問題点であった。本研究において我々はグローバル変数への書き込みが存在する関数からグローバル変数への書き込みを行うプログラムスライスを除くことを提案した。グローバル変数への書き込みを行うプログラムスライスを取り除くことにより投機的再利用の適応が可能となるが、取り除いたグローバル変数への書き込みを行うプログラムスライスの実行がオーバヘッドになる。

プロファイル実験からプログラムの実行中にグローバル変数へ書き込まれる値には局所性が存在することが示された。高い局所性を利用してデータ値再利用を行うことにより取り除いたグローバル変数への書き込みを行うプログラムスライスの実行によるオーバヘッ

ドを軽減することが可能であることを示した。

高い局所性を示すグローバル変数への書き込みを行うプログラムスライスのみ取り除く対象とすることにより、再利用によりオーバヘッドを軽減しながら投機的再利用を適応できる副作用のない関数を得ることができることが示された。一方本実験において対象としたアプリケーションプログラムではプログラムスライスの切り出しによる総実行性能への影響は小さいため、投機的再利用による大きな性能向上が期待できる対象に対して値の局所性が高くない変数のプログラムスライスを切り出すことも可能である。以上から投機的再利用の適応範囲を広げることが可能であるためこれらの方法を適応した場合の性能評価を行っていくことが課題である。

参 考 文 献

1) Eiichi Goto and Motoaki Terashima. Mtac - mathematical tabulative automatic computing, September 1977.

2) J G.Steffan and T C.Mowry. The potential for using thread-level data speculation to facilitate automatic parallelization. In *the Fourth International Symposium on High-Performance Computer Architecture*, February 1998.

3) M. Hall, B. Murphy, S. Amarasinghe, S. Liao, and M. Lam. Overview of an interprocedural automatic parallelization system. In *Fifth International Workshop on Compilers for Parallel Computers*, pages 570–579, June 1995.

4) S.Hirasawa and K.Hiraki. Exploiting dynamic value locality in internal variables. In *Symposium on Advanced Computing Systems and Infrastructures SACSIS2004*, pages 269–275, May 2004.

5) Shiwen Hu, Ravi Bhargava, and Lizy Kurian John. The role of return value prediction in exploiting speculative method-level parallelism.

6) Steve S.W. Liao, Perry H. Wang, Hong Wang, Gerolf Hoflehner, Daniel Lavery, and John P. Shen. Post-pass binary adaptation for software-based speculative precomputation. In *Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation*, pages 117–128. ACM Press, 2002.

7) MikkoH. Lipasti and JohnPaul Shen. Exceeding the dataflow limit via value prediction. In *International Symposium on Microarchitecture*, pages 226–237, December 1996.

8) Mikko H. Lipasti, Christopher B. Wilkerson, and JohnP. Shen. Value locality and load value prediction. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, pages 138–147, October 1996.

9) Stephen E. Richardson. Caching function results: Faster arithmetic by avoiding unnecessary computation. Technical report, Sun Microsystems Laboratories SMLI TR-92-1, September 1992.

10) F. Warg and P. Stenstrom. Limits on speculative module-level parallelism in imperative and object-oriented programs on CMP platforms. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT 2001)*, pages 221–230, September 2001.

11) M Weiser. *Program slices: formal, psychological, and practical investigations of an automatic program abstraction method.* PhD thesis, University of Michigan, Ann Arbor, 1979.