計算機アーキテクチャ 147－21
ハイパフォーマンス 89－21
コンピューティング
（２００２．３．８）

ループへの効果的な投機的実行適用手法

石川 隼輔[1] 山名 早人[1,2]
[1]早稲田大学理工学部 [2]アドバンスト並列化コンパイラ研究体

概要

本稿では、ループに対して投機的実行を効果的に適用する手法について提案し、SPECcpu95ベンチマークのcompressプログラムを用いて有効性を検証した。一般的に、ループはプログラムの実行時間の大部分を占めていることから、ループ並列化によるプログラム速度向上率は高い。しかし、従来の並列化手法では、データ依存が静的に解析できない場合、データ依存が存在すると仮定し解析を進める。このため、例えば、実際には1万回に1回しかループ運搬依存(LCD)が発生しないようなループであっても並列化することができない。しかし、このようなループに対して投機的実行を適用することにより、プログラムを高速に実行することが可能となる。

本稿では、従来考慮されてこなかった投機的実行失敗時の復帰処理に必要なオーバーヘッドをパラメータとして取り入れることにより、投機的実行の効果が期待できる部分のみを選択的に投機的実行する方式を提案する。提案手法は、復帰処理オーバーヘッドの他、LCDが実行時にどの程度の確率で存在するのかを表すLCD存在確率と、投機的実行開始位置とをパラメータとし、選択的な投機的実行を実現する。本手法をcompressプログラムに適用した結果、現状では3倍の速度低下がみられた。このため、速度低下の原因を解析し、その原因を解決するための新たな投機的実行適用手法も提案する。

An Efficient Speculative Execution Scheme for Loops

Shunsuke Ishikawa[1] Hayato Yamana[1,2]
[1]School of Science and Engineering, Waseda Univ. [2]Advanced Parallelizing Compiler Project

In this paper, we propose an efficient speculative execution scheme for loops, and have confirmed the usefullness of the scheme using the compress program from SPECcpu95 benchmark. Generally, since the execution time of loops holds the large portion of the total execution time, the loop parallelization scheme improves the program performance, dramatically. However, when the data dependence cannot be analyzed statically, the conventional parallelization scheme assumes that the data dependence exists. For this reason, such a loop cannot be parallelized even if the loop carried dependence(LCD) occurs only once in 10,000 times, dynamically. However, the speculative execution scheme has been known to speedup such a loop.

In this paper, we propose the scheme to apply the speculative execution alternatively only to the portion expected to be speeduped effectively, using the overhead parameter required for the book-keeping process when the speculation fails. Such overhead has not been considered on conventional speculative execution schemes. The proposed scheme enables the alternative speculative execution using the overhead parameter for book-keeping, the LCD existence probability , and the timing of the speculative execution initiation.

As a result, in the present stage, the execution speed is fell down to one third. To solve this problem, we also propose a new speculative execution.

1. はじめに

マルチプロセッサシステムの普及とともに、プログラム実行時間の大部分を占めるループ構造の並列化技術が重要視されてきている。しかし、プログラム中に存在するループはDoAll型、DoAcross型[2]のように、従来技術で並列化可能なループだけではない。従来技術で並列化不可能なループに対しては、タスクレベルの投機的実行を適用することで並列化可能であることが知られている[1]。

タスクレベルの投機的実行適用手法として、Uhtらによる DEE(Disjoint Eager Execution)[3]や、大津らによる

SMT(Selective Multi-path Execution)[4]などが提案されている。いずれの提案手法も投機的実行失敗時の復帰処理によるオーバーヘッドを考慮せず、すべての制御依存に対して投機的実行を適用する方法をとっている。しかし、復帰処理によるオーバーヘッドは一般に大きく、適用効果の低い投機的実行は行うべきではない。

すなわち、投機的実行の成功率と成功時の効果、さらには復帰処理のオーバーヘッドを考慮した選択的な投機的実行適用手法が必要となる。本稿では、投機的実行失敗時のオーバーヘッドを考慮した、選択的な投機的実行適用手法について提案し、SPECcpu95 ベンチマーク[7]の129.compress に適用し、検討した。

2. ループの分類と投機的実行適用対象ループ

本節では、ループを 6 種類に分類し、投機的実行対象ループを明確にする。ループを「ループ制御部」と「ループ Body 部」に分けて考えた場合、ループ制御部の繰り返し回数が不定か否か、ループ Body 部はループ運搬依存(LCD)が存在するか、また存在した場合は LCD 依存距離が固定か否かによって、表 1 のようにループを分類することができる[1]。

表 1 ループの分類[1]

<table>
<tr><td rowspan="2"></td><td rowspan="2"></td><td colspan="2">ループ制御部</td></tr>
<tr><td>繰り返し回数固定
(L1)</td><td>繰り返し回数不定
(L2)</td></tr>
<tr><td rowspan="4">ループ Body 部</td><td>LCD なし (B1)</td><td>Do All 型ループ</td><td rowspan="2">投機的実行対象
ループ
(T1)</td></tr>
<tr><td>LCD 依存距離固定
(B2)</td><td>Do Across 型
ループ</td></tr>
<tr><td>LCD 依存距離不定
(B3)</td><td colspan="2">投機的実行対象ループ (T2)</td></tr>
</table>

ループ Body 部が表 1 の条件(B1)、または(B2)を満たし、ループ制御部が条件(L1)を満たすループはそれぞれ DoAll 型、DoAcross 型ループと呼ばれ、従来技術で並列化可能なループである。

また、条件(B1)、または(B2)を満たしているが、繰り返し回数が不定(L2)なループに対しては、ループ制御部にループが繰り返す方向に ControlSpeculation を適用することで、ループを容易に投機的実行することができ、DoAll 型、DoAcross 型ループとして並列化可能である(T1)。

LCD が存在し、かつその依存距離が実行時にしかわからないループには、ループ制御部に ControlSpeculation を適用するとともに、ループ Body 部に対しても DataSpeculationを適用することで、ループを DoAcross 型ループの変形と見なすことができる(T2)。本稿では、表1の(T2)に分類されるループを対象とする。(T2)のようなループの並列化手法としては J.G.Steffan らの TLDS(Thread-Level Data Speculation)[5]や Pen-Chung Yew の Superthreaded Processor[6]などが提案されている。

3. ループへの効果的な投機的実行適用手法

表 1 の(T2)に分類されるループの並列化は、イテレーション実行中に LCD の原因となる変数の値を DataSpeculation し、次のイテレーションを投機的に起動することを再帰的に繰り返すことにより実現される。

ここで、LCD が実行時にどの程度の確率で存在するのかを表す値を **LCD 存在確率**、ループ Body 開始位置から LCD の原因となる変数の値が決定されるまでにかかる時間を **LCD ディレイ**、ループ Body 開始位置から LCD の原因となる変数の値を DataSpeculation し投機的実行を開始するまでにかかる時間を**投機的実行開始ディレイ(SE ディレイ)**と定義する。各定義と、ループへの投機的実行適用の様子を図 1 に示す。

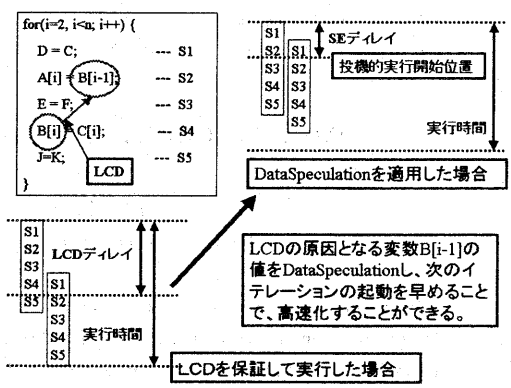

図 1 投機的実行によるループ並列化

LCD を持つループに対して、投機的実行適用を判断するためにはするには、次の 2 点を投機的実行の効果が得られるように決定しなければならない。

- DataSpeculation 適用対象 LCD
- 投機的実行開始位置

まず、ループ中に複数存在する LCD の中から、DataSpeculation を適用すべき LCD を選択する必要がある。

次に、DataSpeculation を適用すると決定された LCD について、投機的実行開始位置を選択する必要がある。従来の研究[1]では、ループ Body 開始時点で投機的実行が開始されている。しかし、投機的実行開始位置が遅れるにつれ、すでに実行されたコードから得られる情報量は多くなる。得られる情報量が多ければ、DataSpeculation の成功確率が上がる。すなわち、次イテレーションに対する投機的実行が成功する確率が

上昇する。

一方、投機的実行開始位置が遅れるにつれ、SE ディレイは大きくなり、投機的実行の効果が少なくなる。よって、投機的実行成功率と SE ディレイを考慮して、適切な位置で投機的実行を開始することが重要となる。以下、本稿で提案する投機的実行適用手法を示す。

3.1 DataSpeculation 適用対象 LCD

並列化できないループ中には一般的に複数の LCD が存在する。プログラム中に静的に存在する LCD でも、動的情報を活用することで依存が解決される場合がある。本稿では、動的な情報として LCD 存在確率に注目する。例えば、100,000 回のループ中にたった一度しか起こらない LCD でも、静的には LCD が存在すると認識される。しかし、このような場合、一度しか起こらない LCD は存在しないものとみなして投機的に各イテレーションを並列実行し、100,000 回に一回の失敗時においてのみ復帰処理を行うほうが高速であると考えられる。

すなわち、ループ中に複数存在する LCD の LCD 存在確率を動的情報(トレース等)を用いて求め、LCD 存在確率の低い LCD を存在しないものとして処理することで、高速化が可能である。同様な理由で、DataSpeculation 成功率の高い変数(データ予測成功率の高い変数)による LCD には常に DataSpeculation を適用することで、高速化可能である。以下に、具体的な DataSpeculation 適用 LCD 決定手法を示す。

(1) すべての LCD のうち LCD 存在確率が P_{min} 以下の LCD を存在しないものとみなす。次に、変数 V によるすべての LCD において、変数 V への DataSpeculation 成功確率がすべて Q_{max} 以上の場合に限り、変数 V によるすべての LCD は存在しないものとみなす。

(2) (1)の処理が終了後、(1)の処理によって残った LCD の中に、DataSpeculation 不可能な変数による LCD が含まれている制御フローは、投機的実行適対象外となる。すべての制御フローのうち、DataSpeculation 可能な変数による LCD のみが存在する制御フローが投機的実行対象フロー F となり、F 上の LCD が DataSpeculation 適用対象 LCD と決定される。

3.2 投機的実行開始位置

3.1 により決定された投機的実行適用対象制御フロー上に存在する DataSpeculation 対象 LCD の投機的実行開始位置の決定方法について述べる。本節では、ループをモデル化し、投機的実行開始位置と、失敗時の復帰処理にかかるオーバーヘッドを考慮した、速度向上率を求める手法を示す。 ただし、本稿では、投機的実行適用対象フロー上に DataSpeculation 対象 LCD はひとつしか存在しないとする。ここで、ループ中に存在する変数 V による LCD を C_i(i=1,...,max)、C_i の存在確率を P_i、各 C_i の LCD ディレイを D_i、C_i における変数 V への DataSpeculation 成功率を Q_i、投機的実行を一回失敗した場合の復帰処理にかかるオーバーヘッドを H、投機的実行開始位置を S とする。そして、DataSpeculation 適用対象 LCD を C_k (ただし、$1 \leqq k \leqq max$)とし、C_k は変数 V による LCD とする。ただし、本モデルでは C_i は排他的な制御フロー上に存在し、いずれかの LCD が解決された時点で次のイテレーションを実行開始できるとする。ループ Body 開始位置から投機的実行開始位置 S までの SE ディレイを E_s とする。投機的実行開始位置Sまでに解決された最後の LCD を C_j とし、C_j までの LCD が解決された時点での C_k の条件付き実行確率を P(k/j)とする。P(k/j)は以下の式で表せる。

$$P(k/j) = \frac{Pk}{\sum_{i=j+1}^{max} Pi}$$

図 2 に各定義を図解する。

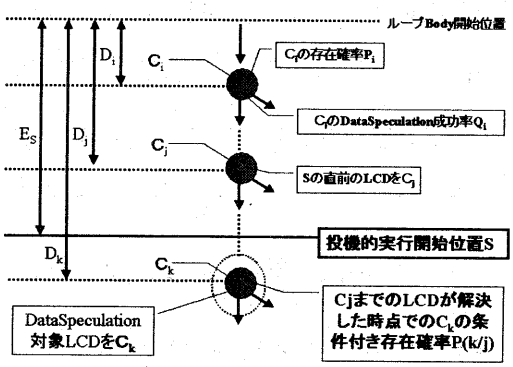

図 2 モデル化における定義

ここで、S において投機的実行を開始した場合の、データ依存を保証して実行した場合に比べた速度向上率(投機的実行を適用しない場合を基準とする)を考える。各イテレーションの実行パターンは以下の 3 パターンに分類できる。

- C_k を含まない制御フローが選択される場合 ── P1
- C_k を含む制御フローが選択されるが、DataSpeculation に失敗する場合 ── P2
- C_k を含む制御フローが選択され、かつ DataSpeculation に成功する場合 ── P3

P1 の場合、C_i が確定した時点で次のイテレーションを起動できる。よって、C_i の LCD ディレイである D_i が P1 が選択された時のイテレーション開始までのディレイとなる。D_i の実行時間と C_i 実行確率の積が P1 のイテレーションのディレイの合計となる。Pi によるディレイの合計を DPi とすると、以下のようになる。

表2 ループ中のLCDとその対処方法

ソース中の行No	変数名	依存距離	LCD存在確率	DataSpeculation成功確率	対処方法
484L→480L	ent	1	57.5%	予測不可能	対処不能。
496L→480L	ent	1	12.5%	予測不可能	対処不能。
504L→480L	ent	1	30.0%	100％予測可能	ひとつ前のイテレーションの変数 c の値を使用することで100％予測することが可能。
511L→(483L, 486L)	htab[i]	不定	1%未満	予測不可能	i はハッシュ変数であり、LCD の存在確率は低い。よって、依存を無視して処理。
511L→(495L, 499L)	htab[i]	不定	0.?%	予測不可能	i はハッシュ変数であり、LCD 存在確率が低い。依存を無視して処理。
514L→(483L, 486L)	htab[i]	1	0.01%未満	予測不可能	実行時に確実に依存するが、その存在確率が低い。よって、依存を無視して処理。
514L→(495L, 499L)	htab[i]	1	0.01%未満	予測不可能	実行時に確実に依存するが、その存在確率が低い。よって、依存を無視して処理。
510L→484L	codetab[i]	不定	1%未満	予測不可能	i はハッシュ変数であり、LCD の存在確率は低い。よって、依存を無視して処理。
510L→496L	codetab[i]	不定	1%未満	予測不可能	i はハッシュ変数であり、LCD の存在確率は低い。よって、依存を無視して処理。
511L→508L	free_ent	1	99.9%	100％予測可能	ひとつ前のイテレーションの free_ent の値をインクリメントすることで100％予測可能。
514L→508L	free_ent	1	0.01%未満	100％予測可能	100%予測可能であるが、存在確率が非常に低い。よって、依存を無視して処理。

$$DP1 = \sum_{i=1, i\neq k}^{max} Di \times Pi \quad \text{—— (1)}$$

同様に、P2の場合はC_kが確定した時点で次のイテレーションを起動できる。P2 の場合、実行確率は S の時点での条件付き実行確率と DataSpeculation の失敗確率の積となる。よって、DP2 は以下のようになる。

$$DP2 = Dk \times (1 - \sum_{i=1}^{j} Pi) \times P(k/j) \times (1 - Qk) \quad \text{—— (2)}$$

次に、P3について述べる。P3の場合、投機的実行開始位置 S において C_k への投機的実行が成功する。よって、C_k によるディレイはD_kではなくE_sとなる。よって、DP3は以下のようになる。

$$DP3 = Es \times (1 - \sum_{i=1}^{j} Pi) \times P(k/j) \times Qk \quad \text{—— (3)}$$

最後に、投機的実行失敗時の復帰処理によるオーバーヘッド O は以下の式で表せる。

$$O = H \times (1 - \sum_{i=1}^{j} Pi) \times (1 - P(k/j) \times Qk) \quad \text{—— (4)}$$

式(1)、式(2)、式(3)、式(4)の和が、投機的実行適用時の実行時間となる。よって、速度向上率は以下のようになる。

$$\text{速度向上率} = \frac{\sum_{i=1}^{max} Di \times Pi}{DP1 + DP2 + DP3 + O} \quad \text{—— (5)}$$

よって、式(5)の値が最大になるような S で投機的実行を開始するのが最も効果的である。

4. compressを用いた検証

3 で示した、適用手法を SPECcpu95 ベンチマークの 129.compress ベンチマークプログラムを用いて検証した。compress ベンチマークプログラムに含まれる関数compress()には while ループが含まれており、その while ループが全実行時間に占める割合は99%である。よって、while ループを並列化することでプログラム全体を高速化できる。

まず、**3.1** で示したように、ループ中に存在する LCD とその LCD 存在確立を調査する。表 2 に while ループ中に存在する LCD とその調査結果を示す

3.1 で示した手法により、DataSpeculation 適用 LCD 決定する。まず、**3.1** の P_{min} を 5%、Q_{max} を 95%とすると、表 2 からわかるよう

に、変数 ent 以外による LCD は、存在しないとみなしたほうが有効な LCD である。よって、変数 ent による LCD が含まれる制御フローのみが投機的実行対象制御フローとなる。

図 3 に投機的実行対象制御フローを示す。図 3 中の Si は変数 ent の値決定文であり、Ei は制御フローを表している。以下、Si を含む LCD を LCDi とする。

図 3 からわかるように、変数 ent の値決定文は排他的な制御フロー上に存在し、変数 ent の値が決定した時点で、次のイテレーションを実行開始可能である。LCDi の内、LCD1 と LCD2 は DataSpeculation 適用不可能である(表 4-1 の 484L→480L と 496L→480L)。よって、E1,E2 は投機的実行対象外である。残った、LCD3 は DataSpeculation 適用可能であるので、E3,E4 が投機的実行対象制御フローとなり、LCD3 が DataSpeculation 適用対象 LCD と決定される。

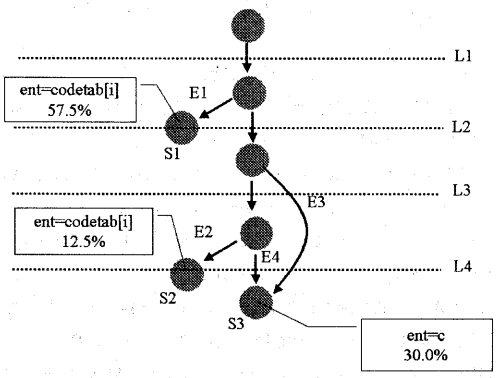

図 3 投機的実行開始位置の候補

次に、投機的実行開始位置を検討する。図 3 の Li は投機的実行開始位置の候補を示している。表 3 に各制御フローの存在確率を示す。変数 ent は図 3 の文 S1,文 S2,文 S3 の何れかひとつで決定される。ここで、文S1、もしくは文S2が実行された場合、変数 ent を Data Speculation することができないが、文S3 が実行された場合、DataSpeculation は 100%成功する。よって、E3,E4 の Path が実行された場合に変数 ent に対する DataSpeculation が成功し、結果として次のイテレーションの投機的実行が成功する

表 3 各制御フローの存在確率

制御フロー	値決定文	実行確率
E1	S1	57.5%
E2	S3	12.5%
E3	S2	10.1%
E4	S3	19.9%

以下、開始位置を L1 から L4 へと変化させた場合の DataSpeculation 成功確率を求める。

よって、投機的実行成功率は変数 ent への Data Speculation 成功率と近い値をとるが劣る。以上のように、投機的実行開始位置によって大きく DataSpeculation の成功率(正確には制御フローE3 もしくは、E4 を想定して投機的実行された次イテレーションの成功率)が変化することがわかる。しかし、Data Speculation の成功率がそのまま投機的実行の成功率となるわけではない。変数 ent 以外による LCD はすべて存在しないとしているが、実行時には確率は低いながらも存在する。変数 ent 以外による LCD が存在した場合は ent への Data Speculation が成功したとしても、投機的実行自体は失敗する。よって、投機的実行成功率は変数 ent への Data Speculation 成功率と近い値をとるが劣る。

次に、各投機的実行開始位置までの SE ディレイについて考える。本稿では 1 単位時間 1 命令に置き換えて考え、その単位を Ls とする。よって、LCD ディレイ、及び SE ディレイはループ Body 開始位置からの平均命令数となる。表 4 に開始位置における SE ディレイと、投機的実行成功率を示す。

表 4 各投機的実行開始位置におけるディレイ

実行開始位置	SE ディレイ(Ls)	投機的実行成功率
L1	3.0	約 30%
L2	4.0	約 70%
L3	12.4	約 61%
L4	14.4	約 100%

ここで、各投機的実行開始位置における速度向上率は 3.2 で示した式(5)により求めることができる。投機的実行失敗時の復帰処理にかかるオーバーヘッド H を変化させた場合の、各投機的実行開始位置における速度向上率は図 4 のようになる。

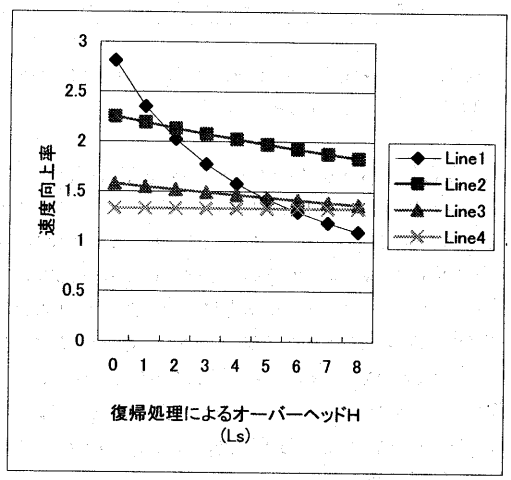

図 4 速度向上率の変化

オーバーヘッドが少ない場合には、L1 による投機的実行開始が最も効果的であり、従来研究での手法と同様であるが、オーバーヘッドの増大とともに、投機的実行開始位置を L2 にした場合のほうが効果的であることがわかる。

一般にHの値は大きいことを考えると、本稿の提案手法が従来から手案されている手法に比べて有効である。

5. テスト実装と結果

実装には OpenMP を使用し、各イテレーションを SECTION 並列で実行する。ここで、投機的実行処理ではスレッドの生成及び、終了を頻繁に繰り返すことになるが、この処理にかかるオーバーヘッドは大きく、速度向上が得られない。そこで、SECTION 内で busy/wait により、SECTION の起動を一度に抑える。

また、4 で述べたように、変数 ent 以外による LCD は存在しないとして処理しているが、実行時には存在する可能性がある。よって、1 イテレーションごと変数 ent 以外による LCD の存在チェックを行い、もし存在した場合は投機的に起動した次のイテレーションを止める必要がある(このように、投機的実行した結果と、実際の結果の相違を確認することを以下、ループ整合性チェックと呼ぶ)。測定したプラットフォームを表 5 に示す。

表 5 測定プラットフォーム

測定マシン名	DELL PowerEdge8450
CPU	Pentium3 Xeon 733MHz× 8
メモリ	2GByte
L1 Cache	16KB(I) + 16KB(D) on chip
L2 Cache	2MB(I+D) on chip
OS	Windows 2000 Advanced Server
コンパイラ	Compaq Visual Fortran ver5.9

結果として、現状では 3 倍程度シリアル実行時に比べ速度が低下してしまうことがわかった。

6. 整合性チェックオーバーヘッド削減手法

5のテスト実装で速度低下の大きな原因は、ループの整合性チェックのオーバーヘッドであることがわかった。今回のテスト実装では変数 ent への Data Speculation は 100%成功しているが、4 で述べたように他の変数による LCD を存在しないとみなして処理しているため、実際に LCD が実行時に存在していないかどうかを 1 イテレーション実行ごとにチェックするオーバーヘッドがかかる。そこで、ループ整合性チェックによるオーバーヘッドをを解消する新たな投機的実行によるループ並列化手法を提案する。

一回のループ整合性チェックにかかるオーバーヘッドを削減することは難しいが、ループ整合性チェックの回数を減らすことで、トータルのオーバーヘッドは削減可能である。

そこで、CPU を2グループに分けて、片方はシーケンシャル実行を行い、もう片方は、従来と同様の投機的実行を行う。投機的実行部では、決められた数のイテレーションをループの整合性チェックなしで実行後、投機的実行側の独自ルーチンで一括してチェックを行うことで、チェック回数を削減できる。また、シリアル実行を常に行っているので、投機的実行失敗時のリスクを軽減できる。

7. おわりに

本稿では、ループへの効果的な投機的実行適用手法を提案し、実際に compress ベンチマークに適用した結果、現状では3倍の速度低下がみられることがわかった。今後は6の手法のインプリメントを通して、1.x 倍の速度向上を達成したい。

参考文献

[1] 山名,小池:"臨界投機実行のループへの適用",情処研報 計算機アーキテクチャ研究会, Vol139, pp.2-5 (2000.8)

[2] R.Cytron:"Doacross Beyond Vectization for Multiprocessors", Proc. of the Int. Conf. on Parallel Processing, pp.836-844 (1986)

[3] A.K, Uht. and V.Sindagi,: "Branch Effect Reduction Technique", IEEE Computer, Vol.30, No.5, pp.71-81(1997.5)

[4] 大津,古川,吉永,馬場:"投機適マルチスレッド処理の一手法", 情処研報 計算機アーキテクチャ研究会, Vol.130, pp.61-66 (1998)

[5] J.GSteffan and T.C.Mowry: "The potential for Using Thread-Level Data Speculation to Facilitate Automatic Parallelization" Proc. of HPCA-4, pp.2-13 (1998.2)

[6] J.Y.Tsai, Z.Jiang, E.Ness and P.C.Yew: "Performance Study of a Concurrent Multithread Processor" Proc. of HPCA-4, pp.24-35(1998.2)

[7] SPEC: http://www.spec.org