非数値計算プログラムにおけるスレッドレベル並列性の限界：スレッド間メモリ曖昧性除去技術との関係

中嶋 昭夫[†]　小林 良太郎[†]
安藤 秀樹[†]　島田 俊夫[†]

マルチスレッド実行は、スーパスカラ方式の性能限界を大幅に上回ることができる技術として重要である。しかし、近年の研究では、非数値計算プログラムにおいて大きな性能向上は達成できていない。本論文では、非数値計算プログラムに内在する TLP を十分に引き出すためには、スレッド間のメモリ曖昧性除去技術にどの程度の高度さが要求されるかを調査した。SPECint95 ベンチマークを用いて評価した結果、以下のことが明らかとなった。

スレッド間メモリ依存違反は発生しないと楽観的に予測した場合、実際にはメモリ依存違反は頻繁に発生するため、TLP は厳しく制限される。高度な技術によりメモリ依存を予測した場合、各ロード命令に対し最も頻繁に発生するメモリ依存を予測できれば、利用可能な TLP はほぼ 2 倍に増加するが、依然として上限の 64%に制限される。各ロード命令に対し頻度の高い順に 2 つ、または、4 つのメモリ依存を予測できれば、それぞれ、半数、または、大部分のベンチマークプログラムにおいて、TLP は上限に達する。以上より、非数値計算プログラムに内在する TLP を十分に引き出すためには、各ロード命令に対し頻度の高い順に 2 つから 4 つのメモリ依存を予測できなければならないことが明らかとなった。

Limits of Thread-Level Parallelism in Non-Numerical Programs: Relation with Inter-Thread Memory Disambiguation Techniques

Akio Nakajima,[†] Ryotaro Kobayashi,[†] Hideki Ando[†]
and Toshio Shimada [†]

Multithreaded execution is important as a technique that can greatly improve performance beyond the bounds of superscalar processor performance. However, recent studies show that great improvements have not been achieved in non-numerical programs. This paper explores how much of sophistication is required to inter-thread memory disambiguation techniques to fully exploit thread-level parallelism in non-numerical programs. We have found the following results using SPECint95 benchmark programs.

Optimistic prediction, where no inter-thread memory violation will occur, severely limits exploitable TLP because memory dependence violations frequently occur in fact. If we assume sophistication, where a single most frequent memory dependence for each load can be predictable, exploitable TLP mostly doubles over the optimistic prediction, but it is still limited to 64% of the upper bound. If we assume two or four most frequent memory dependences for each load can be predictable, half or most benchmark programs achieve TLP of the upper bound, respectively. Consequently, we have found multiple but less than four most frequent dependences for each load must be adaptively predictable to fully exploit TLP in non-numerical programs.

1. はじめに

近年、複数のスレッドを並列に実行可能なチップマルチプロセッサ (**CMP**) の研究が盛んに行われている。マルチスレッド実行は、単一のプログラムを複数のスレッドに分割してスレッドを並列に実行することで、プログラムに内在する並列性として、ILP に加えてスレッドレベル並列性 (**TLP**) も利用することができる。単一の命令流に内在する ILP の上限は小さいため[18]、マルチプロセッサは、従来の ILP のみを並列性として利用するスーパスカラ方式を越える性能が見込まれる。CMP は、マルチプロセッサを構成する複数のプロセッサをチップ内に集積することで、プロセッサ間の通信レイテンシを小さくできる。

これまでに実用化されているマルチプロセッサは、数値計算プログラムやオンライントランザクション等

† 名古屋大学大学院工学研究科
Graduate School of Engineering, Nagoya University

の TLP を利用しやすいプログラムを対象としている。数値計算プログラムは、規則正しいループや配列の参照が多いため、コンパイラによる並列化が容易である。一方、近年の CMP[3),4),15)~17)] は、オンライントランザクションプログラム等のサーバー用途向けに実現可能となっている。オンライントランザクションプログラムは、複数のクライアントから互いに依存関係のない処理を要求されるため、並列化が容易である。

しかし、非数値計算プログラムを対象とした CMP の研究[2),10),14)] では、スーパスカラ方式を大きく越える性能向上は達成できていない。一方で、非数値計算プログラムを対象とした CMP の研究において抽出できる TLP と、Lam らが調査した TLP の上限[9)] には乖離がある。この原因は、限界研究における理想的な仮定にあるが、我々は、その中でもメモリ曖昧性除去に着目した。一般に、非数値計算プログラムでは、メモリ曖昧性除去は非常に困難なため、これをどの程度正確に行えるかは、抽出可能な TLP に大きな影響を与えると考えられる。

そこで、本論文では、非数値計算プログラムにおいて、スレッド間のメモリ曖昧性除去技術がプログラムに内在する TLP の上限にどのような影響を与えるかを調査する。

以下、2 章で関連研究を述べる。3 章でメモリ曖昧性除去技術について説明する。4 章でシミュレーションモデルを説明する。5 章で評価を行い、最後に本論文をまとめる。

2. 関 連 研 究

TLP の限界に関する研究がいくつか行われている。Lam らは、非数値計算プログラムにおいて、プログラム内の並列性を引き出すための条件を制御依存制約に注目して調査しており、制御依存解析、複数スレッドの同時実行、分岐予測による投機的実行 3 つの技術をすべて適用することが有効であることを示した[9)]。Oplinger らは、関数レベルとループレベルの TLP の限界について調査した[13)]。また、スレッド間データ依存を緩和する受信値予測の効果についても測定した。Kreaseck らは、プロファイルとコンパイラによるコード解析を行い、ループ全体あるいは関数をスレッドとして、制御およびメモリ依存がないと予測されるスレッドを実行した場合について、並列性の上限を測定した[8)]。ただし、スレッド間のオーバーラップ命令数を並列性としている。Warg らは、オブジェクト指向プログラムにおいて、ヒープ領域にアクセスのするロード命令の結果値と関数の戻り値の予測が、TLP の限界に与える影響について調査した[19)]。目次らは、Lam らのモデルにおいて、値予測の有効性と、メモリ曖昧性除去技術が TLP の上限に与える影響について調査した[11)]。この研究では、メモリ曖昧性が理想的に除去可能な場合の TLP の上限について調査されているが、メモリ曖昧性除去技術がどの程度の高度さを要求しているかは明らかとなっていない。

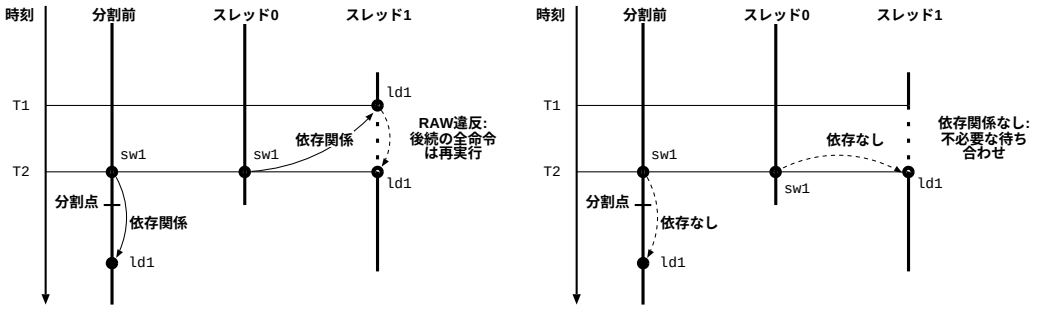
3. メモリ曖昧性除去技術

マルチスレッド実行は、プログラムの異なる部分を同時に実行することで、プログラムに内在する並列性をより多く利用できる可能性を持っている。しかし、非数値計算プログラムおいて、マルチスレッド実行によってスーパスカラ方式を上回る性能向上は達成できていない。その理由として、一般的に、非数値計算プログラムは、制御構造が複雑で、数値計算のような規則正しいループが少なく、配列やポインタの不規則な参照が多いことが挙げられる。このため、コンパイラによるメモリ曖昧性除去は非常に困難であり、TLP は非常に大きく制限される。

メモリ参照命令は、アクセスするメモリアドレスが動的に変化するため、実行時でないと真の依存関係にあるかどうかはわからない。マルチスレッド実行では、逐次命令順とは異なる命令順で命令は実行されるため、将来実行される先行スレッドのメモリ参照命令と依存関係が発生する可能性がある。最も保守的には、先行スレッドのストア命令のアドレスがすべて判明するまでロード命令を実行しないという方法が考えられるが、これは、先行スレッドの実行がすべて終了する制約とほとんど同等であり、スレッドを並列に実行することができなくなる。このため、スレッド間メモリ曖昧性を実行前に除去する手法が必要となる。

文献 2)，5)，6) では、スレッド間のメモリ依存違反は発生しないと楽観的に仮定してロード命令を実行する方式を提案している。これらは、予測が失敗した場合はロード命令と後続命令の再実行が必要となる。このメモリ曖昧性除去技術において性能が低下する例を、図 1(a) を用いて説明する。同図の時刻 T1, T2 はそれぞれストア命令 `sw1`、ロード命令 `ld1` のアドレスが判明する時刻である。子スレッドの `ld1` と親スレッドの `sw1` が依存関係にあるかは、時刻 T2 になってはじめて判明する。時刻 T1 で `ld1` が実行されると RAW 違反が発生するため、`ld1` と後続命令を再実行する必要がある。この違反を**スレッド間メモリ依存違反**と呼ぶ。本論文では、マルチプロセッサを構成する単一のプロセッサとして、スーパスカラ方式を仮定しており、単一スレッド内の命令はアウトオブオーダで実行される。これより、`ld1` 命令と依存関係のない命令は、`ld1` の実行よりも早い時刻で実行される。しかし、スレッド間メモリ依存違反が発生すると、`ld1` 命令と後続の命令をすべて再実行する必要がある。これは、`ld1` に依存している命令のみを選択的に再実行を行うことが困難なためである。このため、スレッド間メモリ依存違反によって利用可能な TLP は著しく制限される。

より高度な手法として、メモリ依存を正確に予測することでメモリ曖昧性を除去する手法が、文献 12) では提案されている。この手法では、メモリ依存の関係を保持するテーブルを用意し、ある静的なロード命令に対して、スレッド間でスレッド間メモリ依存違反を引き起こすとされる最近の静的なストア命令を登録する。テーブルに登録されているロード・ストア命令はメモリ依存関係にあると予測し、同期を行う。予測ミスすると TLP が低下するが、次の 2 つの場合がある。スレッド間のメモリ依存が生じないと予測ミスした場合は、前述した図 1(a) の状況が生じ、スレッド間メモリ依存違反によって TLP は著しく制限される。逆に、同図 (b) の例では、`ld1` と `sw1` には依存関係がない。しかし、`ld1` が `sw1` に依存関係があると誤って予



(a)メモリ依存違反による性能低下　　**(b)**不必要な待ち合わせによる性能低下

図 1　スレッド間のメモリ曖昧性による性能低下

測すると、ld1 は不必要に sw1 を待ち合わせることになる。

本論文では、以上で述べたメモリ曖昧性除去技術を考慮した上でモデル化し、非数値計算プログラムに内在する TLP を十分に引き出すためには、メモリ曖昧性除去技術にどの程度の高度さが要求されるかを調査した。

4. シミュレーションモデル

評価に用いたマシンモデルと、スレッド間メモリ曖昧性除去のモデルについて説明する。

4.1 マシンモデル

まず、マルチプロセッサを構成する 1 つのプロセッサの仮定について説明する。このプロセッサは、無限の命令バンド幅を持ち、分岐予測が正しい間連続して命令をフェッチできるとする。分岐予測には静的予測を用いる。予測ミスによる状態回復に要するコストはゼロとする。フェッチした命令を無限の大きさの命令ウィンドウに格納し、レジスタとメモリの両方について真の依存を満たす命令を同時に発行する。スレッド内のメモリ曖昧性は理想的に除去できるとする。レジスタは無限にあるとし、出力依存と逆依存はないとする。機能ユニットやメモリポート等は無限に存在し、資源制約はない。全ての命令の実行レイテンシを 1 サイクルとする。本論文ではこの単一スレッドのプロセッサモデルを **SP モデル**と呼ぶ。

次に、マルチプロセッサモデルについて説明する。プロセッサ数を無限とし、無限の数のスレッドを同時に実行できるとする。通信やスレッド生成などの並列実行に関わるオーバヘッドはゼロとする。スレッド間のメモリ曖昧性除去技術については、4.2 節で説明する。

マルチプロセッサモデルにおけるスレッドの分割について説明する。我々は以前の研究において、非数値計算プログラムにおいて、より多くの並列性を引き出すためのコンパイル技術として、スレッド分割レベルの違いによる TLP の限界について調査した[7]。これより、関数やループレベルの並列化は有効ではなく、基本ブロックレベルの分割が有効であることが明らかとなった。そこで、本論文では、スレッド分割レベルを基本ブロックレベルとした。基本ブロックレベルのスレッド分割について説明する。基本ブロックの先頭をフォーク点、それを**後支配**[20] する基本ブロックの先頭を子スレッドの開始点とする。後支配の定義から、子スレッドはその親スレッドに制御依存しない。今回の測定では、関数毎に制御依存関係を解析したので、後支配する基本ブロックは関数内に限定した。なお、基本ブロック X が基本ブロック Y を後支配するとは、制御フローグラフ[1] において、Y から終点に至るいかなるパス上にも X がある場合をいう。

また、我々は以前の研究において、制御依存制約の緩和技術として、投機的スレッド生成と投機的レジスタ通信の有効性を示した[7]。まず、投機的スレッド生成について説明する。非数値計算では、分岐の出現頻度が高く、制御依存が並列性を大きく制限すると考えられる。この制約を緩和するために、スレッド生成点にある命令が命令ウィンドウに入ると、制御依存の解消を待たずに投機的にスレッドを生成すると仮定した。次に、投機的レジスタ通信について説明する。マルチスレッド実行では、子スレッドの命令がその親スレッドの命令に依存している場合、親スレッドで生成された値を子スレッドで利用できるのは、最も早くて、その値が子スレッドに**到達**[1] すると確定した時である。この制約を緩和するために、分岐予測を利用して定義の到達を予測できると仮定した。以上 2 つの投機における予測ミスのコストはゼロとした。本論文では、以上で述べたマルチプロセッサモデルを **PD モデル**と呼ぶ。

PD モデルのスレッド分割の判断について説明する。実行時に命令ウィンドウに基本ブロック (フォーク点) がフェッチされたとき、後支配するブロック (子スレッド開始点の候補) のうち、手前のブロックから、フォーク点で子スレッドを生成することにより実行終了時刻が子スレッドを生成しない場合より早く終了できるかどうかを測定し、並列性の向上が期待できるとして分割する。そうでない場合は、分割を行わない。

最後に、プログラムに内在する並列性の上限を示す **ORACLE モデル**について説明する。ORACLE モデルは、完全な分岐予測により、制御に関する制約が全く無い理想のモデルである。SP モデルと同様に、スレッド内のメモリ曖昧性は理想的に除去できるとする。これより、ORACLE モデルは真のデータ依存のみによって並列性が定まる。

いずれのモデルにおける並列性の測定においても、完全に関数を展開し、関数の呼び出しに関係する本質的でない制約を除いた。具体的には、スタックフレー

ムを確保および解放する操作、関数呼び出しに関わるコンパイラに対する種々の規約に起因する依存（たとえば、関数を越えて生存するレジスタを再使用する際のスタックへの保存と回復）を無視した。また、ループの誘導変数を削除した。これはループ伝搬依存として並列性を制約するが、コンパイラにより容易に取り除くことができるからである。

4.2 スレッド間メモリ曖昧性除去モデル

PD モデルにおけるスレッド間メモリ曖昧性除去のモデルについてそれぞれ説明する。

blind モデルは、スレッド間メモリ依存違反は発生しないと楽観的に予測するモデルである。メモリ依存予測に失敗し、スレッド間メモリ依存違反が発生した場合は、該当命令とすべての後続命令を再実行する。このモデルは、文献 2), 5), 6) の手法に対応する。

predictable n モデルは、頻繁にスレッド間メモリ依存違反が発生する静的なロード・ストア命令について、メモリ依存予測を理想的に行い、スレッド間メモリ依存違反を回避するモデルである。あらかじめ、スレッド間のメモリ曖昧性が理想的に除去可能な PD モデルにおいてシミュレーションを行い、各静的なロード命令について、頻繁にスレッド間メモリ依存の関係となる静的なストア命令の集合とその頻度を採取する。このプロファイルを用いて、実行時に頻繁にスレッド間で依存関係にある上位 n 組に対して、スレッド間のメモリ曖昧性は理想的に除去されるとする。上位 n 組の静的なストア命令の集合に属さない静的なロード命令は、blind モデルと同様に、スレッド間メモリ依存違反は発生しないと楽観的に予測する。n が増加すると、メモリ依存予測には高度な技術が要求されることを示す。

perfect モデルは、スレッド間のメモリ曖昧性が理想的に除去できるモデルである。predictable n モデルにおいて、静的なロード命令に対応するストア命令の組み合わせが無限の場合を指す。

本論文では、blind モデル、predictable n モデルのうち n が 1, 2, 4 の場合と、perfect モデルについて測定を行い、メモリ曖昧性除去技術が TLP の限界に与える影響について評価した。

5. 評　　価

最初に評価環境について述べる。次に、スレッド間メモリ曖昧性除去技術に関する評価結果について述べる。

5.1 評 価 環 境

トレース駆動型シミュレータを作成し測定した。ベンチマークプログラムには、SPECint95 の全 8 本を用いた。トレースは SimpleScalar Tool Set Version 3.0a を修正したシミュレータを用いて採取した。命令セットは MIPS R10000 を拡張した SimpleScalar/PISA である。

表 1 に各ベンチマークにおける入力セットと、シミュレータに入力したトレースの長さ (動的命令数) を示す。各プログラムへの入力セットは、実行命令数が過大にならないようにするために、SPEC の標準入力での実行と関数の出現頻度をほぼ維持しつつ、実行命令数が 100M〜200M になるように調節した。シミュレータに入力したトレースは、プログラムの最初の 100M 命令、あるいは、それ未満で実行が終了するまでとした。ただし、ijpeg と vortex では、プログラムの初期化部分である最初から 50M 命令と 100M 命令をそれぞれスキップし、それ以降の命令を入力とした。

表 1　ベンチマークプログラム

ベンチマーク	入力セット	動的命令数
compress95	ref/bigtest.in (30K 要素に減らしたもの)	95M
gcc	ref/genoutput.i	84M
go	9×9 board, level 6, train/2stone9.in	75M
ijpeg	ref/specmun.ppm	100M
li	train/train.lsp	100M
m88ksim	train/dcrand	100M
perl	train/scrabbl.pl, scrabbl.in (3 語追加), dictionary	80M
vortex	ref/persons.1K, vortex.in	100M

表 2　各モデルの並列性と TLP の上限

ベンチマーク	SP	PD perf.	ORACLE
compress95	6.2	93.1	204.3
gcc	8.9	101.1	274.8
go	4.5	57.1	139.5
ijpeg	24.0	203.4	245.1
li	9.1	64.3	69.5
m88ksim	6.6	257.6	478.2
perl	10.6	96.7	172.7
vortex	52.6	813.4	815.7
並列性 (調和平均)	8.8	108.0	184.9
TLP(調和平均)	1.0	11.6	(17.3 倍)

5.2 評 価 結 果

表 2 に、SP, PD, ORACLE モデルの並列性の上限を各ベンチマーク毎に示す。同表の PD モデルのメモリ曖昧性除去モデルは、perfect モデルである。同表の TLP は、全ベンチマークの TLP の調和平均である。TLP は、各モデルの並列性を単一スレッドである SP モデルの並列性で割った値と定義した。SP モデルの並列性は、単一スレッド実行において利用できる ILP の上限である。PD モデルの並列性は、マルチスレッド実行において利用可能な並列性の上限である。同表より、マルチスレッド実行において制御依存制約を緩和する技術とメモリ曖昧性除去技術をすべて適用した場合、プログラムに内在する並列性の上限は平均 108.0 である。この場合の TLP は 11.6 であり、非常に大きな値であることがわかる。ORACLE モデルの並列性より、PD モデルは上限の 58%の並列性を抽出できることがわかる。

図 2 に、スレッド間メモリ曖昧性除去技術による TLP の影響を示す。横軸はベンチマークと TLP の調和平均、縦軸は TLP である。各ベンチマークに 5 本の棒グラフがある。図の凡例は 4.2 節で述べた通りである。

表 3 に、blind, predictable n モデルにおける、ス

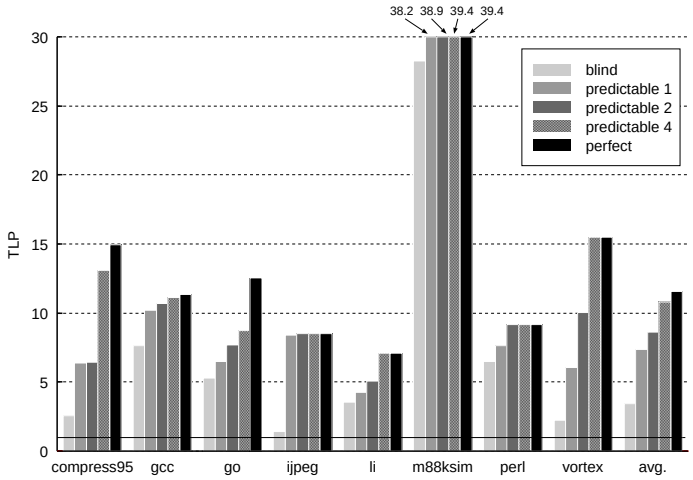


図 2 スレッド間メモリ曖昧性除去技術による TLP の影響

表 3 スレッド間メモリ依存違反が発生するスレッドの割合 [%]

ベンチマーク	blind	pred. 1	pred. 2	pred. 4
compress95	61.1	41.6	31.2	15.3
gcc	46.0	22.1	13.3	7.1
go	59.6	38.5	26.5	15.4
ijpeg	65.6	12.2	0.9	0.0
li	58.0	29.3	6.4	0.3
m88ksim	53.5	13.4	10.9	2.0
perl	44.5	25.4	10.5	4.7
vortex	66.8	44.9	34.8	17.8
算術平均	56.9	28.4	16.7	7.8

表 4 スレッド間メモリ依存違反を引き起こすロード命令の割合 [%]

ベンチマーク	blind	pred. 1	pred. 2	pred. 4
compress95	30.9	19.7	12.8	5.2
gcc	16.0	11.2	6.3	3.2
go	33.7	31.6	20.2	10.3
ijpeg	4.7	2.8	0.0	0.0
li	17.7	14.0	3.6	1.6
m88ksim	23.8	4.8	1.4	0.7
perl	16.1	12.9	3.5	1.5
vortex	8.3	7.6	4.4	2.2
算術平均	18.9	13.1	6.5	3.1

レッド間メモリ依存違反が発生するスレッドの割合を示す。スレッド間メモリ依存違反が発生するスレッド数を、総スレッド数で割った値である。この値が高いほど、TLP が低下する要因となるスレッドが頻繁に生成されることを示す。表 4 に、blind, predictable n モデルにおける、スレッド間メモリ依存違反を引き起こすロード命令の割合を示す。スレッド間メモリ依存違反が発生した動的なロード命令数を、動的な全ロード命令数で割った値である。この値が高いほど、TLP が低下する要因となる後続命令の再実行が頻繁に発生することを示す。

これらの測定結果より、blind モデルでは、TLP は平均 3.4 と非常に小さいことがわかる。これは、表 3 より、すべてのベンチマークにおいて、スレッド間メモリ依存違反が発生するスレッドの割合が平均 56.9%と非常に頻繁に発生するためである。表 4 より、ijpeg, vortex では、スレッド間メモリ依存違反を引き起こすロード命令の割合はそれぞれ 4.7%、8.3%と少ないが、表 3 より、65%以上のスレッドがスレッド間メモリ依存違反を引き起こすため、利用可能な TLP は上限の 29%に制限される。これより、楽観的な予測による手法では、プログラムに内在する利用可能な TLP がほとんど失われることがわかる。

predictable 1 モデルの TLP は平均 7.4 であり、blind モデルと比較して、TLP はほぼ 2 倍に増加する。スレッド間メモリ依存違反が発生するスレッドの割合は平均 28.4%であり、blind モデルに対して大きく減少するが、利用可能な TLP は上限の 64%に制限されている。しかし、gcc, ijpeg, m88ksim は、predictable 1 モデルで perfect モデルとほぼ同等の TLP を達成している。この理由は、これらのベンチマークでは、スレッド間メモリ依存違反が発生するスレッドの割合が 25%以下にまで減少しており、また、ijpeg, m88ksim ではメモリ依存違反を引き起こすロード命令の割合も 5%以下に減少しているからである。

predictable 2 モデルの TLP は平均 8.6 であり、利用可能な TLP は上限の 74%である。predictable 1 モデルで perfect モデルとほぼ同等の TLP を達成した 3 つのベンチマークに加えて、さらに perl が perfect モデルとほぼ同等の TLP を達成している。この理由は、表 3, 4 より、スレッド間メモリ依存違反を引き起こすスレッドの割合とロード命令の割合がともに著しく減少しているからである。

predictable 4 モデルでは、predictable 1 と predictable 2 モデルで perfect モデルとほぼ同等の TLP を達成した 4 つのベンチマークに加えて、さらに compress95, li,vortex が perfect モデルとほぼ同等の TLP を達成している。predictable 4 モデルの TLP は平均 10.8 であり、利用可能な TLP は上限の 93%まで達成している。これより、predictable 4 モデルは、ほとん

どのベンチマークにおいて perfect モデルとほぼ同等の TLP を達成する。go は、スレッド間メモリ依存違反が発生するロード命令の割合が、他のベンチマークと比較して 10.3%と高い。このため、go において perfect モデルの TLP を達成するためには、predictable 4 以上の高度なメモリ曖昧性除去技術を必要とする。

6. ま と め

マルチスレッド実行は、プログラムの異なる部分を同時に実行することで、プログラムに内在する並列性をより多く利用できる可能性を持っている。しかし、近年の非数値計算プログラム向けのマルチプロセッサの研究では、スーパスカラ方式の性能を大幅に上回る性能は達成できていない。一方で、Lam らの TLP の限界に関する研究において示された、非数値計算プログラムに内在する TLP の上限と、近年のマルチプロセッサで利用可能な TLP には乖離がある。この原因は、限界研究における理想的な仮定にあるが、我々は、その中でもメモリ曖昧性除去に着目した。本論文では、非数値計算プログラムにおいて、スレッド間のメモリ曖昧性除去技術がプログラムに内在する TLP の上限にどのような影響を与えるかを調査した。その結果、以下のことが明らかとなった。

スレッド間メモリ依存違反は発生しないと楽観的に予測した場合、実際にはメモリ依存違反は頻繁に発生するため、TLP は厳しく制限される。メモリ依存予測を導入した場合、予測技術を高度なものにすると、利用可能な TLP は徐々に増加する。一般に、メモリ依存関係は動的に変化する。各ロード命令に着目した場合においてもその例外ではない。このため、メモリ依存予測には、動的な変化に順応できる高度な技術が必要とされる。我々は技術の高度さを、各ロード命令に対して予測可能なメモリ依存の数 n で表すこととした。評価結果によると、n が 1 の場合、楽観的な予測の場合と比較して TLP はほぼ 2 倍に増加するが、依然として上限の 64%に制限される。n が 2 の場合、TLP は SPECint95 ベンチマークの半数において上限に達し、平均で上限の 74%となる。さらに n が 4 の場合、ほとんどのベンチマークにおいて TLP は上限に達する。以上より、非数値計算プログラムに内在する TLP を十分に引き出すためには、各ロード命令に対し頻度の高い順に 2 つから 4 つのメモリ依存を予測できなければならないことが明らかとなった。

謝辞 本研究の一部は、文部科学省科学研究費補助金基盤研究 (C) 課題番号 15500036、文部科学省 21 世紀 COE プログラム、財団法人栢森情報科学振興財団研究助成の支援により行った。

参 考 文 献

1) A. V. Aho et al., *Compilers: Principles, Techniques, and Tools*, Addison-Wesley Publishing Company, Reading, MA, 1986.
2) H. Akkary et al., “A Dynamic Multithreading Processor,” In *Proc. 31st Int. Symp. on Microarchitecture*, pp.226-236, December 1998.
3) L. Barroso et al.,“Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing,” In *Proc. 27th Int. Symp. on Computer Architecture*, pp.12-14, June 2000.
4) J.M. Borkenhagen et al., “A multithreaded PowerPC processor for commercial servers,” IBM J. Res. Dev., vol. 44, no. 6, pp.885-898, 2000.
5) M. Franklin et al., “ARB: A Hardware Mechanism for Dynamic Reordering of Memory References,” *IEEE Trans. on Computers*, Vol.45, No.5, pp.552-571, May 1996.
6) L. Hammond et al., “Data Speculation Support for a Chip Multiprocessor,” In *Proc. Eighth Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp.58-69, October 1998.
7) 加納 正晃 他, “非数値計算プログラムにおけるスレッド・レベル並列性の限界,” 情報処理学会研究報告 2000-ARC-140, pp.55-60, 2000 年 11 月.
8) B. Kreaseck et al., “Limits of Task-based Parallelism in Irregular Applications,” In *Proc. Third Int. Symp. on High Performance Computing*, pp.43-58, October 2000.
9) M. S. Lam et al., “Limits of Control Flow on Parallelism,” In *Proc. 19th Int. Symp. on Computer Architecture*, pp.46-57, June 1992.
10) P. Marcuello et al., “Speculative multithreaded processors,” In *Proc. of the 1998 Int. Conf. on Supercomputing*, pp.13-17, July 1998.
11) 目次 勝彦 他, “スレッドレベル投機的並列処理アーキテクチャにおけるデータ依存制約緩和手法の効果,” 並列処理シンポジウム JSPP2002, pp.3-10, 2002 年 5 月.
12) A. Moshovos et al., “Dynamic Speculation and Synchronization of Data Dependence,” In *Proc. 24th Int. Symp. Computer Architecture*, pp.181-193, 1997.
13) J. T. Oplinger et al., “In Search of Speculative Thread-Level Parallelism,” In *Proc. of the 1999 Int. Conf. on Parallel Architectures and Compilation Techniques*, pp.303-313, October 1999.
14) G. S. Sohi et al., “Multiscalar Processor,” In *Proc. 22nd Int. Symp. on Computer Architecture*, pp.414-425, June 1995.
15) J. M. Tendler et al., “POWER4 System Microarchitecture,” *IBM Journal of Research and Development, Vol. 46, No. 1*, Jan 2002.
16) M. Tremblay, “MAJC: Architecture: A Synthesis of Parallelism and Scalabilitty,” *IEEE Micro,* pp.12-25, November/December 2000.
17) E. Waingold et al., “Baring It All to Software: Raw Machines,” *Computer 30 9* pp86-93, 1997.
18) D. W. Wall, “Limits of Instruction-Level Parallelism,” In *Proc. Fourth Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp.176-188, April 1991.
19) F. Warg et al., “Limits on Speculative Module-Level Parallelism in Imperative and Object-Oriented Programs on CMP Platforms,” In *Proc. Int. Confe. on Parallel Architectures and Compilation Techniques*, pp.221-230, September 2001.
20) H. Zima et al., *Supercompilers for Parallel and Vector Computers*, Addison-Wesley Publishing Company, Inc., New York, NY, 1991.