計算機アーキテクチャ 126－13
（1997. 10. 29）

オンチップ MIMD プロセッサにおける実行時並列化機構の性能評価

玉造潤史　松本尚　平木敬

我々は、スーパースカラアーキテクチャによるプロセッサの速度向上がこれ以上求められない現在において大規模集積回路の集積度の上昇より得られる大きなハードウエア資源を活用し、かつ、現在の命令レベルでの投機実行よりもさらに大きな粒度であるループブロックの投機実行を行なうことにより高速化を行なう機構としてループブロックを実行時に並列化し実行を行なう run-time restructuring アーキテクチャを提案してきた。ループレベルの並列性は命令レベルの並列性よりも大きな資源を必要とするが、よりハイパフォーマンスを獲得することが可能である。run-time restructuring ではオンチップ MIMD アーキテクチャをベースとした並列マイクロプロセッサにおいて逐次形式で生成されたバイナリプログラムを実行時に解析し、再構成することによって、重複実行によるループレベルの投機実行を行なうことが可能である。本稿では、我々が先に提案したプログラム再構成による並列化スレッドの投機実行を行なうオンチップ MIMD マイクロプロセッサにおける性能向上を SPEC95 ベンチマークと画像アプリケーションに多く用いられている gif,jpeg,mpeg 展開ルーチンを用いて計測した。結果として浮動小数アプリケーションでは大きなの実行時並列化の効果が得られ、また、画像アプリケーションや整数アプリケーションでもプログラム中にループが存在すれば、速度向上が得られることが示された。

Performance evaluation of a run-time restructuring mechanism on On-Chip MIMD

JUNJI TAMATSUKURI , TAKASHI MATSUMOTO and KEI HIRAKI

At present, speed-up of microprocessors based on superscalar architecture hit the ceiling. We have proposed run-time restructuring architecture to utilize large hardware resources which is available by an increasing integrity of current VLSI technology. Our system speculatively exploits dynamic parallelism among loop blocks, which is a larger granularity than that of current instruction-level speculation. Loop level parallelism requires more resources than instruction level parallelism, we can also obtain higher performance. On our run-time restructuring mechanism, on-chip MIMD microprocessors dynamically analyze sequential binary executable and restructure it to execute speculatively each loop body.

In this paper, we evaluate performance improvement of our run-time restructuring on on-chip MIMD microprocessors, using SPEC95 benchmark suite and graphics application kernel which consist of gif, jpeg, and mpeg expansion routines.

1. はじめに

現在用いられているマイクロプロセッサアーキテクチャのほとんどはスーパースカラアーキテクチャである。スーパースカラアーキテクチャでは、命令の実行時に ALU、レジスタといったプロセッサ内資源の割当から並行して機能ユニットが処理可能な命令を抽出し高速化する。この並列性抽出には命令間依存解析としてのリオーダーバッファや重複したレジスタ資源を活用するためのリネーミングレジスタなどに大きなプロセッサ資源を費やしてきた。しかし、命令レベルの並列性は既に多くの研究が示すように限界がありプロセッサ内の演算機能の有効活用は、もはや限界となっている。また大きなリオーダーバッファの複雑な構造を設計する際の困難と費やしたプロセッサ内資源量にもかかわらず、クロック向上によって得られる速度向上を越えることが出来なくなっている。

このような現状を踏まえ、我々は命令レベル並列性を活かしたまま高速化を行なうアーキテクチャとしてオンチップ MIMD プロセッサ上でループレベルの並列実行を実行時に行なう並列化によって実現するアーキテクチャ (run-time restructuring) の提案[1)2)3)4)] を行なった。run-time restructuring は実行中にループを検出し、プログラムの実行において重要な高速にアクセスを行なうレジスタ上のイテレーション間依存を解析によって求め分離し、さらに遅くて大きな資源に対して割り当てられるメモリ上の依存関係はプロセッサ間にまたがる依存検出機構によって保護させることによって各要

† 東京大学 大学院 理学系研究科 情報科学専攻
Department of Information Science,Faculty of Science
University of Tokyo

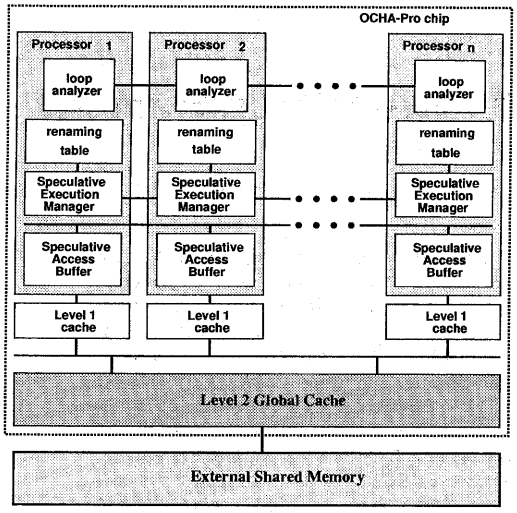

図1 Ocha-Pro の構成図

素プロセッサが独立してイテレーションを実行することを可能とするプログラムに再構成する。また各イテレーション間の並列実行は各要素プロセッサが投機的に実行することで達成される。従って、我々のアーキテクチャはループレベルの投機実行をサポートした並列プロセッサを実現する。また、このアーキテクチャでは要素プロセッサは現在のスーパースカラアーキテクチャのプロセッサを用いることで現在のプログラムの単一プロセッサでの実行速度を低下せずに、オンチップシステム特有の機能である通常のマルチチップ並列プロセッサでは実現不可能な各要素プロセッサ間結合を活用している。そのため、不可能であった実行時の並列化が実現している。

本稿では、run-time restructuring を装備したオンチップ MIMD プロセッサのテストベットである OCHA-Pro シミュレーター上で SPEC CPU INT 95 および SPEC CPU FP 95、そして重要度を増しつつあるマルチメディアアプリケーションの mpeg,jpeg,gif のデコーダ部分の実行性能をシミュレーションによって計測した。これらのアプリケーションの実行結果から速度向上と必要とする投機的メモリアクセスを行なうバッファ (Speculative Access Buffer) のハードウエア量について考察を行なった。

2. OCHA-Pro でのシミュレーション

既に提案したテストベット OCHA-Pro の構成 (図 1) の概略を述べる、この OCHA-Pro におけるパラメータをシミュレーションでは用いている。OCHA-Pro は通常の並列プロセッサとしての構成に次の機能を付加してある。

各 PE 各要素プロセッサは現在のスーパースカラマイクロプロセッサのような命令レベルの並列性を用いたマイクロプロセッサであり、プロセッサは独立して設計することができる。そのため、集積プロセス技術による性能構造を望むことができる。

Loop Analyzer Loop Analyzer は、要素プロセッサが実行時にフェッチしてきた命令流を解析し

- ループ構造を発見する。
- 再構成するプログラムを構成する。

という処理を行なっている。また、これらの解析は要素プロセッサの実行とは独立して処理されるため、要素プロセッサの性能を低下させない。

Speculative Execution Manager Speculative Execution Manager は Loop Analyzer がループを検出し再構成した後の投機実行の管理を分散して行なう。機能は次の 2 種類で Elastic Barrier[5] を用いて構成される。

(1) 要素プロセッサ内の投機実行に活用可能な資源を管理し、投機実行の開始時には割当を行ない、終了時には回収を行なう。開始時に投機実行に必要な資源が足りない時には実行を停止させる。

(2) 投機実行の成功を検証し投機実行を管理する。各イテレーションの実行を終えると各要素プロセッサに分散して装備されている Speculative Execution Manager は他の要素プロセッサに終了の同期を送り、受けとった先行するイテレーションの同期が揃った時に投機実行状態から確定状態に移す。確定状態になると Speculative Access Buffer にメモリアクセスの実行を通知する。

renaming table renaming table は、投機実行の開始時に現在の実行状態を保持するために現在のレジスタの内容を保存する。同じレジスタへの書き込みには、新たなレジスタを割り当て、レジスタの読み出しには正しいレジスタで答える。先行できるイテレーション数はこのレジスタセット数と次に述べる Speculative Access Buffer のエントリ数で決定される。

Speculative Access Buffer 投機的に実行されるメモリアクセスの副作用を外部メモリシステムに影響を与えない機構である。メモリへの書き込みは投機実行の失敗時には正しい値を書き戻さなければならない。しかし Speculative Access Buffer は同等の機能をメモリへの書き込みを抑止して疑似的に実現する。

(1) 読み出しは実行するが読みだしたアドレスを記憶する、他のプロセッサによって書き込みがあった場合には投機の失敗を Speculative Execution Manager に通知する。

(2) 書き込みは実行せず実行が確定するまで保持

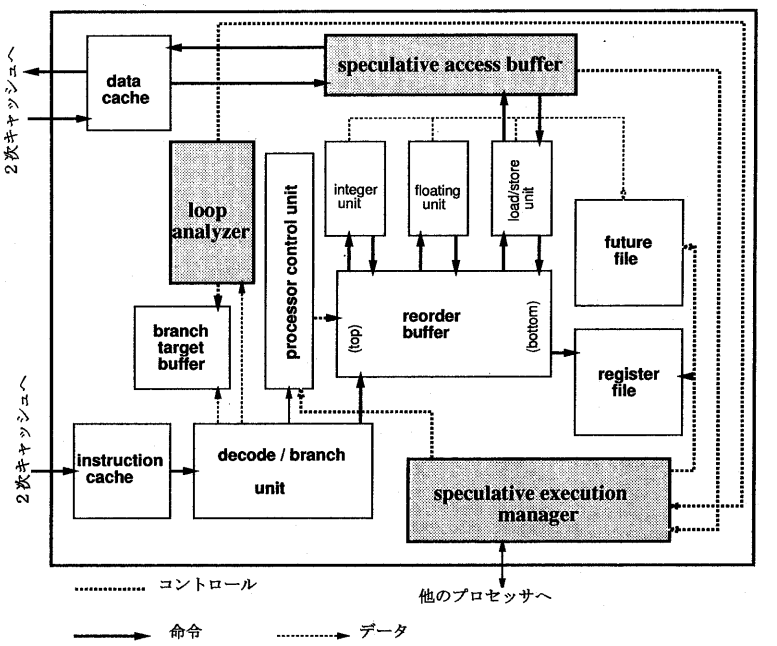

図2 各要素プロセッサ内の構成

しておく。後続のイテレーションからの読み出しの要求があった場合には、外部メモリでなく値を保持しているイテレーションの Speculative Access Buffer が答える。Speculative Execution Manager が投機実行の成功を検出すると保持していたメモリへの書き込みを実行する。

これらの機構を要素プロセッサであるスーパースカラプロセッサと (図 2) 次のように結合させている。スーパースカラアルゴリズムとしては HP PA8000 や Pentium II で採用されているリオーダーバッファ[6] を用いている。この方式では最大限に命令レベルの並列性を引き出しながらの命令発行と逐次的な命令のリタイアを行なうことが可能である。命令セットは MIPS R10000[7] と同等であり、各機能ユニットの処理に必要なクロック数も同様に設定している (表 1)。機能ユニットは整数演算ユニット 2 個、浮動小数演算ユニット 2 個、ロードストアユニット 1 個を持たせている。ロードストアユニットは Speculative Access Buffer が設けられているため発行出来る命令数は制限していない。また、ストア命令もバッファリングされるため後続のロード命令は追い越して実行される。また、投機実行の巻き戻しは命令レベル投機実行 (分岐命令など) はリオーダーバッファの後続のエントリとフューチャーファイル、リオーダーバッファの全内容を破棄しリオーダーバッファ内の処理の終了を待つ。この処理は要素プロセッサが管理する。ループレベルの巻き戻しは必要スレッドのレジスタファイルとフューチャーファイル、リオーダーバッファの全内容を破棄し、Speculative Execution Manager 内に記憶してある投機実行開始アドレスを要素プロセッサは受けとり実行を再開する。

表1 演算ユニットの設定時間

命令	必要クロック数
外部メモリアクセス	10 clock
2 次キャッシュアクセス	3 clock
浮動小数点乗算	4 clock
浮動小数点除算	16 clock
浮動小数点 sqrt	33 clock
浮動小数点 (その他)	2 clock
整数乗算	6 clock
整数除算	7 clock
整数 (その他)	1 clock

3. ベンチマーク

シミュレータ上で SPEC 95 banchmark* と画像処理アプリケーションの再生部分**をシミュレートして実行した。速度向上の計測と共に実行時再構成法によってプログラムのどの程度の部分が並列化され投機実行されるのかを知るために並列化部分のプログラム中のサイズを測定した。また、付加ハードウエア中の Speculative

* SPEC CPU 95 benchmark version 1.0

** xanim revision 2.70.6.2 から xa_mpg.c ,xa_jpg.c xa_gif.c を用いた

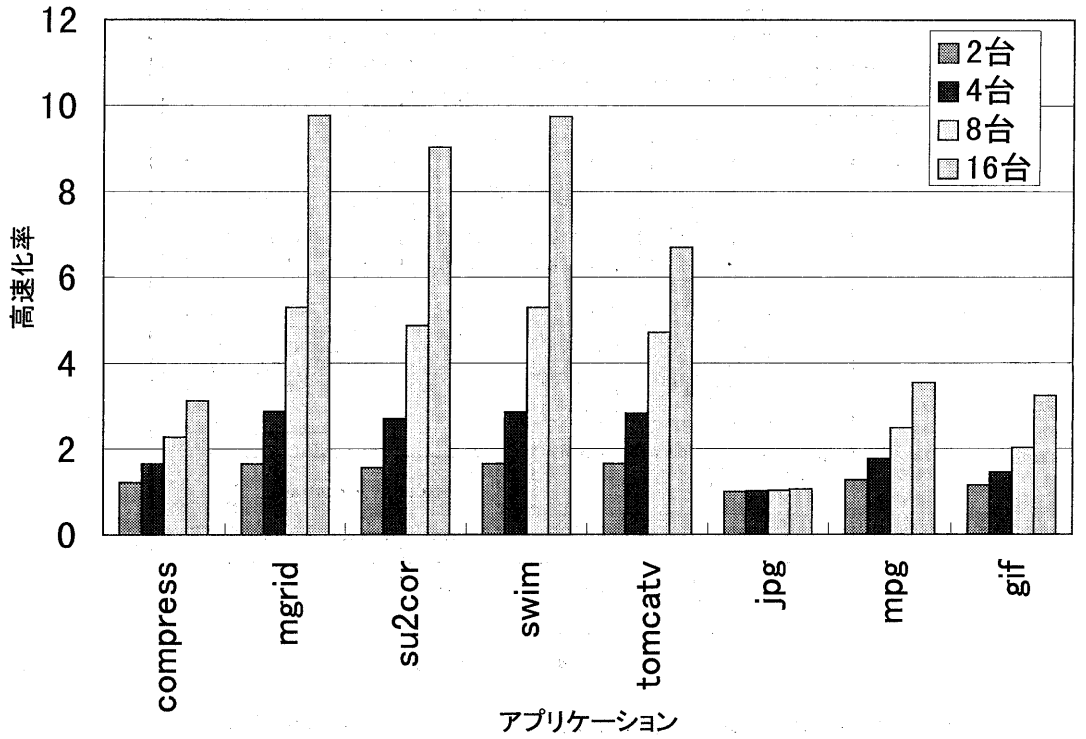

図3 ベンチマーク結果

Access Buffer のエントリ数はシミュレーションにより見積もる必要がある。そのため、ベンチマーク実行中に使用された Buffer サイズについても計測を行なった。

3.1 速度向上性能

図 3は、各ベンチマークを実行した結果である。それぞれのアプリケーションはそれぞれ次のような結果を表している。

3.1.1 SPEC CINT 95

整数演算アプリケーションでは 129.compress を計測した。整数演算アプリケーションではループではなく分岐構造によって構成されているものが多く、これらについては後述する。 compress は圧縮プログラムで、制御構造の大部分はメモリに蓄えられた以前に出てきたデータに依存して決定される。そのため、構造に多くのループを持ちながらも投機が失敗してしまう場合が多くある。しかし、データの圧縮が効く場合にはプログラムも同様に同じ動作を繰り返すため、台数効果がリニアに出ないながらも高速化して実行する。

3.1.2 SPEC CFP 95

浮動小数点アプリケーションでは 101.tomcatv, 102.swim, 103.su2cor, 107.mgrid で実験を行なった。それぞれのプログラムの多くがループを多く持った構造である。また、メモリ上のデータを繰り返しアクセスしながら実行するためメモリへの投機実行の効果も大きい。そのため実行時再構成法によって大きな速度向上を得ることが出来る。特に tomcatv と su2cor は最内ループ構造が大きい。またループ構造の割合の多い swim , mgrid でも大きな並列性が得られておりグラフでは、それらの速度向上に台数効果が出ていることが分かる。

3.2 画像アプリケーション

圧縮された画像を復元する画像アプリケーションは特定のデータ分割によって区切られた部分を次々と処理する。そのため、画像データの読み込み、展開がそれぞれループによって記述される。しかし、この実験で用いた jpeg のプログラムは jpeg 分割サイズが小さく展開されるパターンが少数であるためループを用いずパターンマッチによって行なわれている。結果として、初期化部分以外の並列性は得られていない。一方、 mpeg は展開した画像を次々と処理するフィルターを分割ごとに通過させるため小さなループ構造を多数含んでいる。したがって、完全な台数効果は得られないものの速度向上を得ることができる。

3.3 プログラムの並列化率

速度向上の結果と比較のために実行時再構成法によって並列化されたプログラム中の割合を示している。プロ

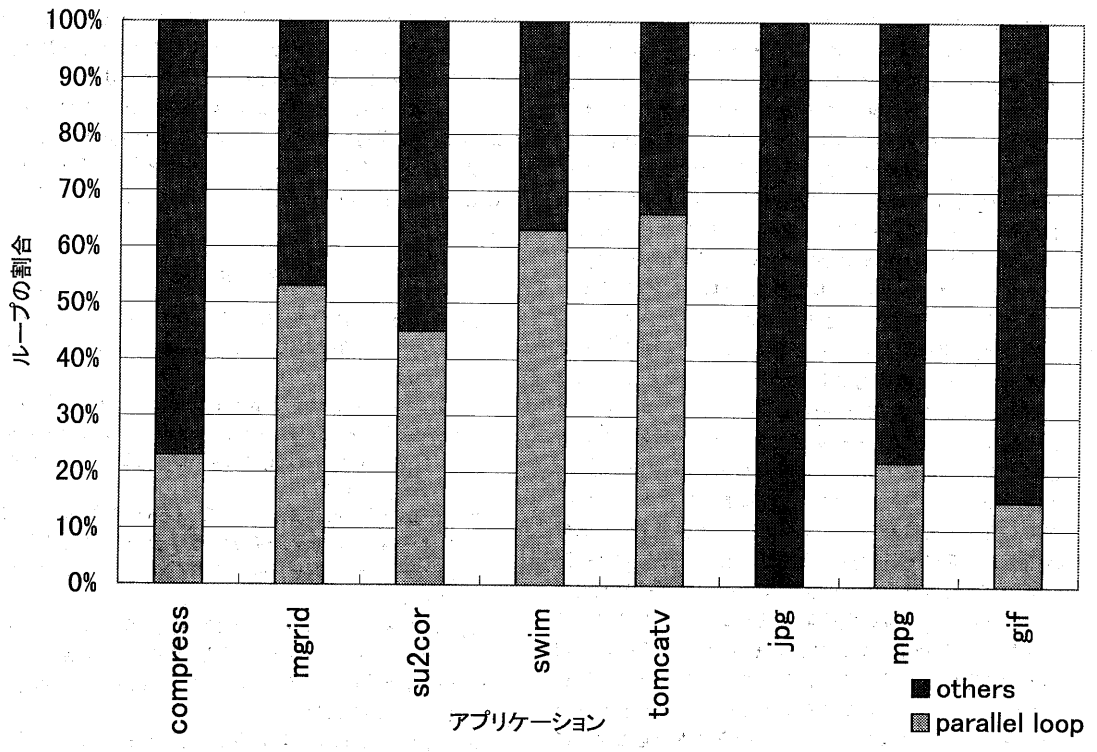

図4 プログラムの並列化率

グラム中の並列化率のため、実行されるループによって得られる速度向上はこの割合よりも十分に大きい。

特に多くの部分が並列化されるのが tomcatv , swim などの浮動小数点演算アプリケーションである。これらのプログラムは本質的にループ構造を持っているため並列化率は大きい。一方、jpeg プログラムのようにデータの特性に特化したプログラムではアプリケーションとしてはループによる並列性を用いることが可能であるのに、プログラム中にループレベルの並列性が全く現れない場合もある。しかし、平均して 20 ～ 30% はループであり、しかもそのループの実行時間は大きいため実行時再構成による並列化の効果は大きい。

3.4 Speculative Access Buffer の見積り

表 2に、ベンチマーク実行時に使用した Speculative Access Buffer のエントリ数を示した。この数値は要素プロセッサが並列実行時に使用したエントリ数の最大値とアプリケーション全体におけるエントリ数の平均値である。

この最大値と同数のエントリ数の Speculative Access Buffer が使用可能な時に、実行時再構成法による投機並列実行が可能である。また、先行投機して 1 イテレーション先に実行できるためにはさらに倍のエントリ数が必要である。そのため、最大の性能を得るには表

表2 Supeculative Access Buffer のエントリ数

ベンチマーク	必要最大数	平均エントリ数
mpeg(xanim)	9	5.52
jpeg(xanim)	1	1
tomcatv	45	10.9
swim	8	2.25
su2cor	59	13.6
mgrid	43	12.44
compress	4 2.75	

中の数値の 2,3 倍のエントリ数の Speculative Access Buffer が必要である。

tomcatv,su2cor を除いたほとんどのアプリケーションではエントリ数は 10 程度で済んでいることが分かる。また、tomcatv,su2cor では大きなループ構造が一度に多くの要求を出すため必要最大数が多いが、イテレーションのサイズが大きいため 1 オーバーラップ分あれば十分である。そのため 60 エントリから 100 エントリの Speculative Access Buffer があれば良いことが分かる。

4. 関連研究

ループレベル投機実行を用いた並列化の研究には

Stanford 大学のグループによる Hydra[8)9)] がある。Hydra ではコンパイラによって静的に並列化を行ないプロセッサ上では投機実行として動作するスレッドを生成している。また、オンチップ MIMD 自身のアーキテクチャは通常の並列プロセッサと同様に構成されている。本研究は実行時のバイナリ解析だけで並列化しているが、Hydra では静的に投機実行スレッドを生成する点が異なっている。実行時に並列化を行なう我々の方法であれば今までは不可能であった並列プロセッサで汎用マイクロプロセッサの重要な性質であるバイナリコンパチビリティを維持することができる。また、ループ以外の並列に実行可能なブロックを単位にメモリのアクセスを含めた投機実行を行なうアーキテクチャとして Wisconsin 大学の Multiscalar[10)] がある。ブロック単位に投機実行をし、メモリアクセスをも投機実行するという点は本研究と大きな類似点がある。Multiscalar では、様々な並列性を用いるために要素プロセッサは独立に実行可能ではあるが、マルチプロセッサ全体の制御構造を集中管理する機構を設けている。一方、我々は実行時にプロセッサ上でループによる並列性だけを抽出しループブロック単位の投機実行は複雑なメモリアクセスの解析を行なわず、レジスタの解析だけで行なえる並列化するというダイナミックアーキテクチャとして実現している。しかも、レジスタ上での依存となるイテレーションインデックスといった実行状態生成を重複実行により解決し各要素プロセッサが完全に独立に実行できるスレッドを生成している。そのため、全プロセッサを集中管理する機構が必要でない点が大きく異なっている。また、ループ構造は基本的に複数回処理を繰り返す場合に用いられるため投機実行したばあいに成立する可能性が大きい。その性質に基づいて作られたのが我々のアーキテクチャである。

5. まとめ

我々は先に示したプログラム実行時再構成法におけるループ構造の投機実行の性能評価を行なった。逐次処理を扱うプログラムに関しては有効な並列性をループ構造だけから抽出するのは困難であることが分かった。そのため、実行時のバイナリトランスレーションなどによってプログラムの有する並列性を抽出するための援助が必要である。この問題については継続的な研究を行ないたい。また、SPEC CINT 95 中の 90.go(碁ゲーム),132.ijpeg(画像処理) に関しては C 処理系の問題で扱うことが出来なかったが、これらは前者は探索問題であり、後者は mpeg デコーダのように多くの並列性を含んでいる。ある種の逐次処理である compress においても速度向上が見られたことでメモリデータ依存性の少ない整数演算アプリケーションにおいても実行時再構成法は有効である。また数値演算処理である SPEC CFP 95 の各アプリケーションはメモリデータ処理であるためメモリ上のデータ間の依存とメモリアクセスの速度及びバンド幅によって性能が決定される。そのため我々が用いた投機メモリアクセス実行バッファ (Speculative Access Buffer) の有効性とそのエントリ数が十分実現可能なサイズであることが分かった。本研究ではメンターグラフィックス社とシノプシス社の University Program を用いました。両者に深く感謝します。

参考文献

1) 平木敬, 玉造潤史, 松本尚: プロセッサによる実行時ループ再構成方式, *proceedings of the HPCS'97* (1997).
2) J.Tamatsukuri, T.Matsumoto and K.Hiraki: On-Chip Parallel Architecture for Run-time Loop Restructuring, (in English) 04, University of Tokyo (1997).
3) 玉造潤史, 松本尚, 平木敬: Loop を並列実行するアーキテクチャ, 情報処理学会研究会報告計算機アーキテクチャ, Vol. 119, No. 11, pp. 61–66 (1996).
4) 玉造潤史, 松本尚, 平木敬: On Chip MIMD における大規模投機実行機構, 情報処理学会研究会報告計算機アーキテクチャ, Vol. 125, No. 24, pp. 139–144 (1997).
5) 松本尚: Elastic Barrier: 一般化されたバリア型同期機構, 情処学会論文誌, Vol. 32, No. 7, pp. 886–896 (1991).
6) Johnson, M.: *Superscalar Microprocessor Design*, Prentice Hall (1990).
7) Silicon Graphics International: *R10000 Users Manual version 1.0* (1995).
8) K.Olukotun, B.Nayfeh, L.Hammond, K.Wilson and K.Y.Chang: The Case for a Single-Chip Multiprocessor, *proceedings of the 7th International Symposium on Architectural Support for Parallel Languages and Operating Systems* (1996).
9) J.Oplinger, D.Heine, S.W.Liao, B.A.Nayfeh, M.S.Lam and K.Olukotun: Software and Hardware for Exploiting Speculative Parallelism with a Multiprocessor, Technical report, Stanford University (1997).
10) G.Sohi, S.Breach and T.Vijaykumar: Multiscalar Processors, *proceedings of the 22nd Annual International Symposium on Computer Architecture* (1995).