スーパスカラと LIW の性能比較

4P－1

川崎弘太　小沢年弘　木村康則
（株）富士通研究所

1　はじめに

単一プロセッサを高性能化する上で、命令レベル並列の活用は重要な課題であり、スーパスカラ(SS)およびLIW(Long Instruction Word)は、命令レベル並列を活用する代表的なプロセッサ・アーキテクチャである。スーパスカラは、並列に実行できる命令をハードウェアにより動的に検出するが、LIWは、並列に実行できる操作をソフトウェアにより静的に検出し1つの長形式命令にまとめておくことを特徴としている。

これまで、両者の汎用プロセッサとしてのシステム性能の比較は、あまり行われていない[1]。そこで本研究では、両者の違いを上記の点と捉え、それ以外の条件を等しくした場合の両者の性能をシミュレーションによって求め、比較した。

2　ターゲット・アーキテクチャ

ターゲット・アーキテクチャとしては、両者の特徴が現れやすいように、次の項目は共通とした。両者とも、同じ機能ユニットを持ち、in-order 命令発行、ハードウェア・レジスタ・リネーミングなし、命令およびデータキャッシュ（ノンブロッキング方式）を備え、多方向分岐命令は使用しない（スーパスカラでは使われていないため）。キャッシュは、汎用プロセッサであることと現状を考慮して採用した。キャッシュの採用に伴い、言語処理系では実際のレイテンシを知ることのできないキャッシュミスに対処するため、LIWでも異なるLIW命令間ではインターロックされることとした。この結果、LIWでの操作のレイテンシの違いを利用した最適化はできないことになった。したがって、両者の違いは主に、同時に発行される命令（操作）が固定であるか可変であるかに現れることになる。

3　評価方法

本研究では、言語処理系の最適化レベルは等しいとし、スーパスカラの各命令をLIW命令の各操作とみなすことにより、次のように性能評価を行った。

同一のアセンブラコードを、それぞれの動作を行わせるため、後述のようにローカルコードスケジュールして実行コードを生成する。そしてそれらをシミュレータで動作させ、実行サイクル数を比較する。

スーパスカラのコードスケジューリングでは、データの依存関係、各操作のレイテンシ、機能ユニットの数を考慮して、各命令の実行開始可能なタイミングを求め、その順序に並べかえた。

LIWのコードスケジューリングでは、スーパスカラ同様に、各命令の実行開始可能なタイミングを求め、その順序に並べかえた。さらに実行開始可能な命令数が命令発行幅に達しない場合は、nop命令で補充した。LIWでは、分岐や合流点の前後でも、同時に発行される命令（操作の組合せ）は変わらないので、命令発行幅に達しない分はnop命令を補充する必要がある。キャッシュミスは、ストアはストアバッファを仮定し、ロードはインターロックをシミュレートするようスケジュールした。またLIW命令内の操作の依存関係およびその組合せは言語処理系で保証し、シミュレーションでは各操作のLIW命令のフィールドの位置は自由とした。

ベンチマークプログラムとして、SPECint92のうちのcompress、eqntott、espresso、gcc、xlispを用い、アセンブラコードは、gcc(version 2.4.5、-O2オプション)で生成した。命令セットアーキテクチャは、SPARC-V8を想定し、UltraSPARCを参考に、命令発行幅(4)、機能ユニットの種類と数(ALU2、LSU1)、各命令のレイテンシ、キャッシュの構成(32byteブロック、命令キャッシュ2way、データキャッシュdirect、ペナルティ10サイクル)等のパラメータを決め、機能ユニットは全て並列動作可能とした。

シミュレーションはトレースベースのシミュレータにより、分岐予測パーフェクトとして行った。

4　実験結果

表1に各プログラムのコードサイズ(単位KB)と、スーパスカラを理想的な(all hit)キャッシュ上で動作させた時のIPC (Instructions Per Cycle)を表す。

コードは、gccが一番大きく、compressが一番小さい。gccやespressoのLIW用のコードは、スーパスカラ用のコードに比べて、ほぼ倍である。

IPCはcompressが一番大きく、xlispが一番小さい。

同じキャッシュ構成のスーパスカラに対するLIWの相対性能を、キャッシュ容量を変化させて測定した結

Performance Comparison of Superscalar and LIW Processor Architecture
Hirota Kawasaki, Toshihiro Ozawa, Yasunori Kimura
FUJITSU LABORATORIES LTD.
1015, Kamikodanaka Nakahara-ku, Kawasaki 211, Japan

表 1: 各プログラムのコードサイズ (KB) と IPC

	size(SS)	size(LIW)	IPC(SS)
compress	57	74	1.62
eqntott	74	98	1.40
espresso	221	410	1.28
gcc	680	1401	1.40
xlisp	131	180	1.12

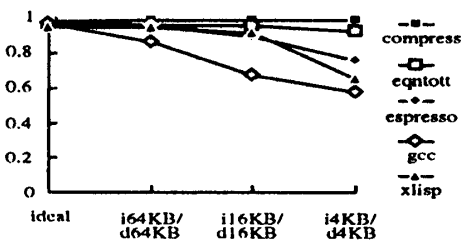

図 1: LIW の SS に対する性能

果を、図 1 に示す（命令キャッシュとデータキャッシュの容量は等しい）。

理想的なキャッシュでは、LIW の性能はスーパスカラの 94～98%である。命令／データ両キャッシュの容量が 16KB の場合は、LIW の速度はスーパスカラに対し、gcc では 85%に落ちているが、その他のプログラムでは 95%以上の性能を示している。命令／データ両キャッシュの容量が 4 KB の場合は、LIW は、compress と eqntott でスーパスカラの 9 割以上の性能を示すが、その他のプログラムでは 6 ～ 8 割の性能で、性能差が大きい。

次に、データキャッシュと命令キャッシュのどちらが性能差に影響を与えているかを調べた。

図 2 は命令キャッシュを理想的なキャッシュにして、データキャッシュの容量を変化させた場合で、データキャッシュの容量の変化は、LIW とスーパスカラの性能差にあまり影響を与えないことを示している。

図 3 はデータキャッシュを理想的なキャッシュにし

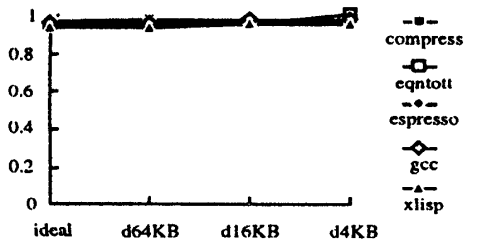

図 2: LIW の SS に対する性能 (i-cache:ideal)

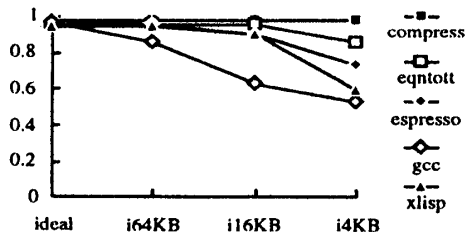

図 3: LIW の SS に対する性能 (d-cache:ideal)

て、命令キャッシュの容量を変化させた場合で、命令キャッシュが小さくなると LIW とスーパスカラの性能差が大きくなる傾向にあることを示している。これは図 1 と同じ傾向であり、データキャッシュより命令キャッシュが性能差に影響を与えていると考えられる。

プログラム（表 1 ）に注目してみると、性能差の大きいプログラムは、実行ファイルが大きくスーパスカラと LIW のコードサイズの差も大きいか、元々の IPC が小さい。LIW では多くの nop が挿入、実行されるため、LIW の性能が低下するものと考えられる。

5 まとめ

本研究では、スーパスカラおよび LIW アーキテクチャの違いを、並列に実行する命令（または操作）を、ハードウェアにより検出し動的に変えることがあるものであるか（スーパスカラ）、それともソフトウェアにより静的に検出し動的には変えないものであるか（LIW）として捉え、それ以外の条件を等しくして、両者の実行サイクルをシミュレータにより求めることにより、定量的な性能比較を試みた。

評価の結果、スーパスカラと LIW の性能差は主に命令キャッシュによって生じること、その差は、理想的なキャッシュでは数%であるが、キャッシュが小さくなると拡大し、サイズの大きいもしくは IPC の小さなプログラムではその傾向が著しいことが分かった。

今後の課題としては、命令発行幅や機能ユニット数、out-of-order 命令発行、ハードウエア・レジスタ・リネーミング、分岐予測などのプロセッサ構成の変化の影響、ベンチマークプログラムの増加、グローバルな最適化の影響などを調べることがあげられる。

参考文献

[1] 山崎 他、ハードウェア記述言語による superscalar 及び VLIW プロセッサの設計とその比較、情処全大、pp.6-7-8、1995.