冬学期 第3回 情報輪講 資料

アウトオブオーダ実行プロセッサの機能ユニット構成

田中・小池研究室 M1 吉瀬 謙二

1995年10月30日

出典: Stephan Jourdan, Pascal Sainrat, and Daniel Litaize,
"Exploring Configurations of Functional Units in an Out-of-Order Superscalar Processor"
The 22th Annual International Symposium on Computer Architecture, pp.117-125, 1995.
他.

1. はじめに

プロセッサ構成においてスーパスカラの手法が一般的なものになっている。さらに多くの命令レベル並列度を抽出し、限られた機能ユニットを効果的に用いるためアウトオブオーダ実行をおこなうプロセッサも登場している。

アウトオブオーダ実行により向上するパフォーマンスは計算機の機能ユニット構成に大きく依存するが、コスト性能比に優れた機能ユニット構成を求める研究はほとんどおこなわれていない。

本発表ではアウトオブオーダ実行の仕組みを説明した後、機能ユニット構成を変化させたときの性能向上を求め、その結果からコスト性能比の良い機能ユニット構成を示す。ただし命令依存関係によってのみ並列度抽出が制限されると仮定している。

最後に今回得られた計算モデルについての考察をおこなう。

2. アウトオブオーダ実行

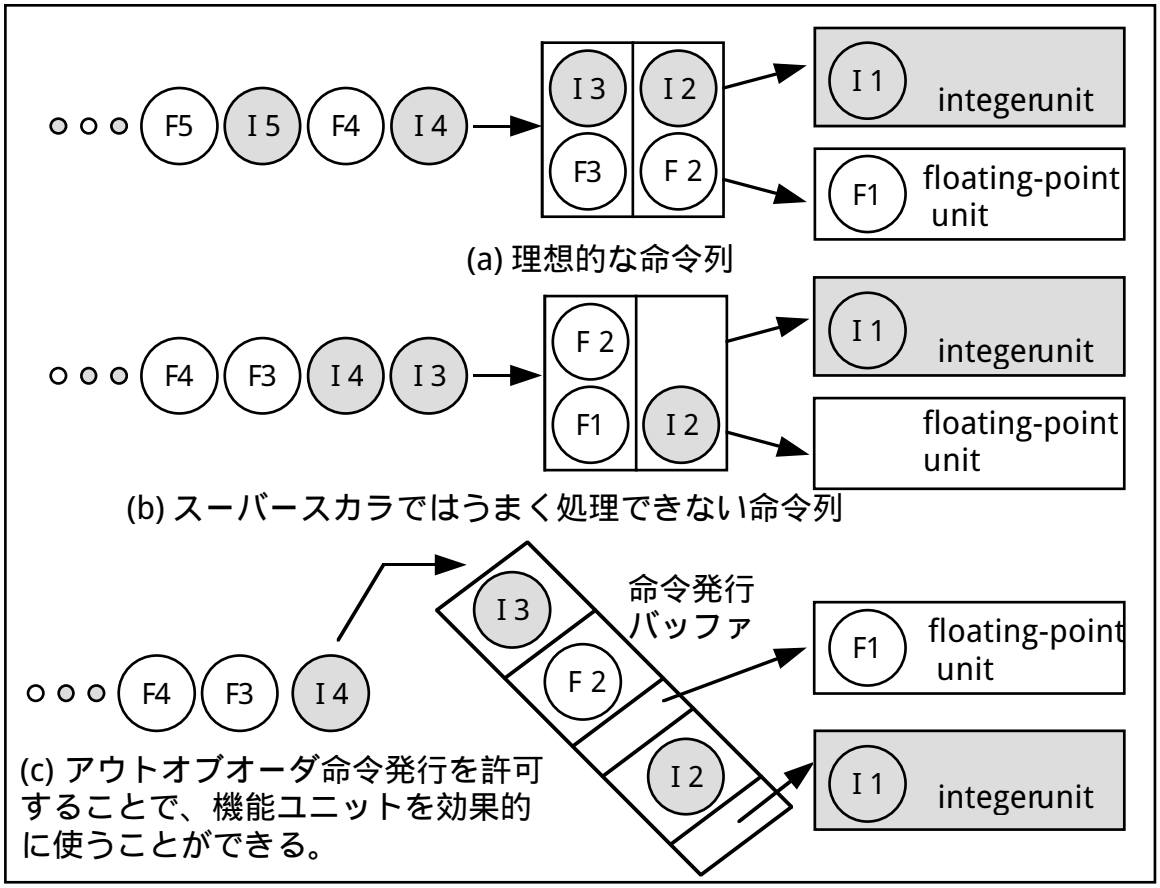


図 1　アウトオブオーダ実行で得をする理由

2.1. アウトオブオーダ実行とは？

図 1 を用いて、なぜアウトオブオーダ実行によって機能ユニットを有効に使うことができるか説明する。

機能ユニットは整数命令を処理する integerunitと不動小数点の命令を処理する floating-pointunitの 2 つを使えるとする。分岐による影響は考えない。

(a) の例はコンパイラが命令列を作成するときに 2 つの機能ユニットの命令を順番に埋め込んだ場合の実行で、本来の命令順に 2 つずつ命令発行していけば 2 つの機能ユニットを効率よく使用することができる。

(b) はコンパイラが最悪の命令列を作成したときの例で、 2 つずつ命令を発行しようとしても衝突により 2 つの機能ユニットを同時に使うことができない。実行時間もそのぶん増大してしまう。

(c) はアウトオブオーダ実行を示す。オペランドの揃った命令で機能ユニットの衝突が起こらなければ、本来の順番を変更して発行する。これより機能ユニットの使用効率を上げることができる。

以上がアウトオブオーダ発行を用いることで実行時間が向上する理由である。

ここで重要な点は、コンパイル時に (ソフトウェアによって) 効果的なスケジューリングがおこなわれたとするとアウトオブオーダ実行の意味がなくなるということである。

2.2. アウトオブオーダ実行モデル

本実行モデルで用いる 6 つのステージと命令発行アルゴリズムを定義する。

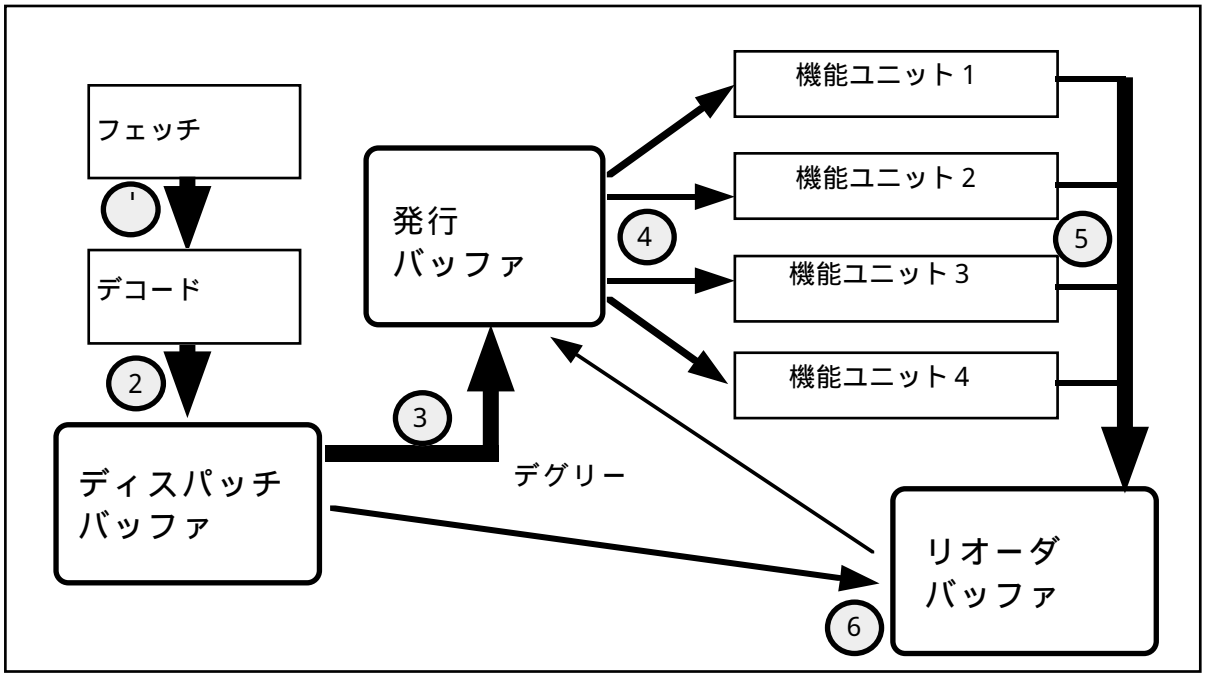


図 2　アウトオブオーダ実行モデルのステージ構成

デグリー (Degree) とは同時にディスパッチできる命令数の最大値である。図 2 ではデグリーの影響を受ける部分を太い矢印で表現している。このデグリーと機能ユニットの数によって同時に実行できる命令数の最大値が決定される。

これから考えるアウトオブオーダ実行モデルは、フェッチ、デコード、ディスパッチ、発行、実行、終了の 6 つのステージをもつ。アウトオブオーダ実行モデルにおける各ステージ構成を図 2 に示す。 1 ~ 6 の番号は各ステージに対応している。

2.2.1. ディスパッチバッファ（図 2 - 2 Time 1）

命令フェッチ・デコードがおこなわれ、ディスパッチバッファに入った命令は、ディスパッチバッファから発行バッファに送り込まれる（ディスパッチされる）。この時、同時にディスパッチされる命令数の最大値がプロセッサのデグリー (Degree)である。

ディスパッチバッファまでは本来の命令順を保っているのでディスパッチする順番はインオーダである。ディスパッチされた時のインオーダ情報はリオーダバッファに伝えられる。

2.2.2. 発行バッファ（図 2 - 2 Time 2）

発行バッファは命令の発火機構である。ハードウェアの衝突がなければ発行バッファ内のオペランドが揃った命令から発行される。

発行バッファ中で発行可能な複数命令の調停をおこなうのが命令発行アルゴリズムである。命令発行アルゴリズムの効率については様々な方法で同程度の効率が得られることが示されており[7] ,我々はもっとも古い時間に命令発行バッファに登録された命令から優先度を高く実行する: oldestfirstというアルゴリズムを採用する。

2.2.3. リオーダバッファ（図 2 - 2 Time 4）

機能ユニットで命令が実行されると、その結果は本来の命令順に整列されてリオーダバッファに格納される。

当該命令以前に割り込みが発生しないことを確認すると、当該命令の結果をレジスタファイルに書き込み当該命令をリオーダバッファのエントリから解放する。

割り込み発生の対象命令は、条件分岐、除算、メモリアクセス命令である

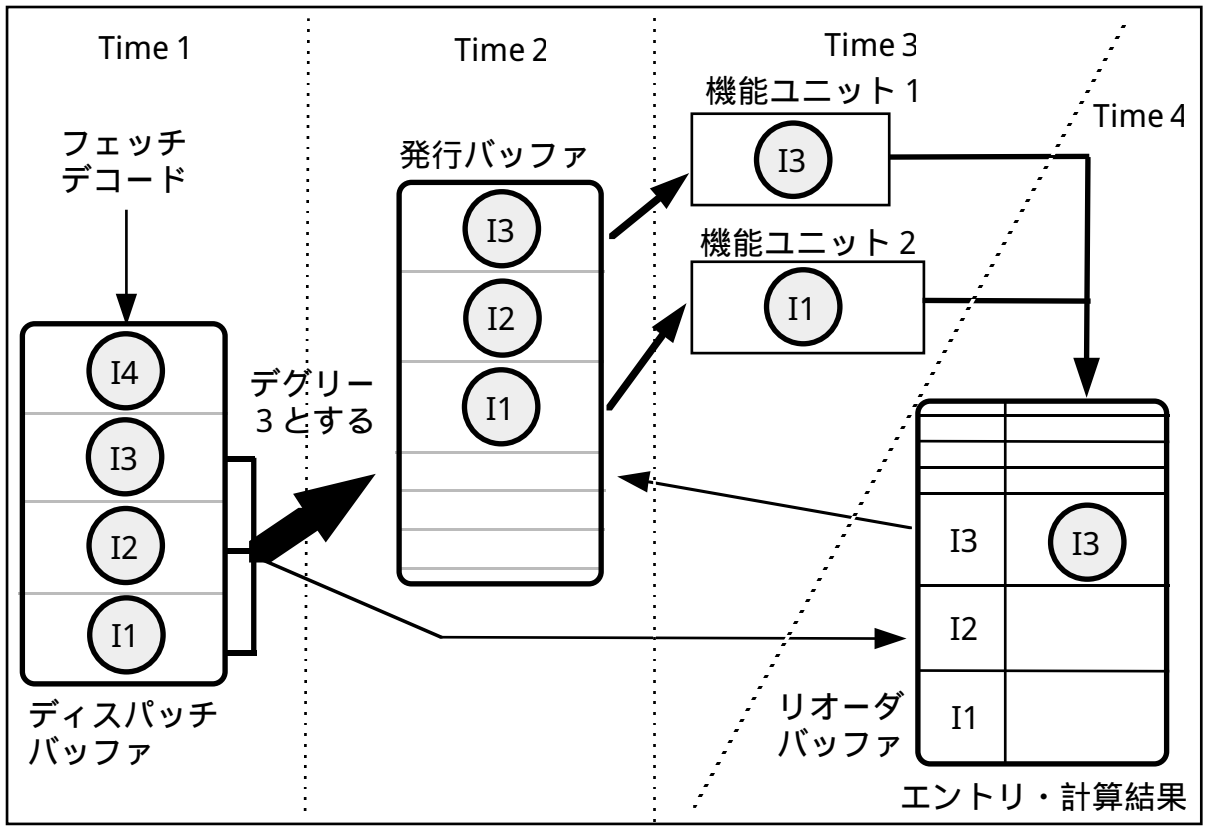


図２－２　アウトオブオーダ実行モデルにおける命令の流れ

2.3. 計算モデルでの仮定

先に定義したモデルでは以下の４つの仮定を用いている。
（１）理想的な命令フェッチ機構を考える。命令キャッシュのミスは発生しないし、
　　分岐予測の失敗は起こらない。
（２）ディスパッチされる命令数はプロセッサのデグリーとリオーダバッファが
　　あふれたときにのみ制限される。
（３）データキャッシュのミスは起こらない。
（４）命令発行バッファは一つに統合され、命令発行アルゴリズムには古いものを
　　優先する oldestfirst を採用する。

この仮定より命令レベル並列度の抽出は命令依存関係によってのみ制限される。

2.4. 命令クラス

命令を以下の９つの命令クラスに分類する。レイテンシ(実行サイクル)を表2に示す。

・整数命令　(Integer)　　：　算術演算、論理演算
・シフト命令 (Shift)　　：　シフトとビット操作に関する命令
・整数 掛算命令 (integermultiply)
・整数 除算命令 (integerdivide)
・ロード/ストア命令 (load/store)
・浮動小数点 演算命令 (floating-pointarithmetic)
・浮動小数点の型変換命令 (floating-pointconvert)
・浮動小数点 掛算命令 (floating-pointmultiply)
・浮動小数点 除算命令 (floating-pointdivide)

Instructionclass	Latency	Instructionclass	Latency
Integer	1	fp arithmetic	3
Shift	1	fp convert	1
integermultiply	3	fp multiply	3
integerdivide	20	fp divide	18s/31d
load/stor	2/3		

表２　実行時間
s は単精度、d は倍精度を表す。

このレイテンシの設定はPowerPC604に基づいている。除算を除いて全ての機能ユニットはパイプライン化され互いに独立に実行を行なう。

2.5. 性能評価に用いられるベンチマークプログラム

以降の性能比較は SPEC92 の実行結果から計算されている。SPEC92 は 整数命令用の６つのプログラムから成る CINT92 と 浮動小数点用のプログラム１４個から成る CFP92 という２つのプログラムの集まりで構成される。
SPEC92 のプログラムは R4600 をベースにした SGIワークステーション上で標準のメイクファイルによりすべての最適化をおこないコンパイルした。
全てのプログラムで約 600万命令のトレースを行なった。図３は、命令の分布の様子を示している。

図 3 より、CINT92 では integer命令が41.8% でもっとも多く CFP92 ではメモリアクセスが 41.7% で最大となっている。

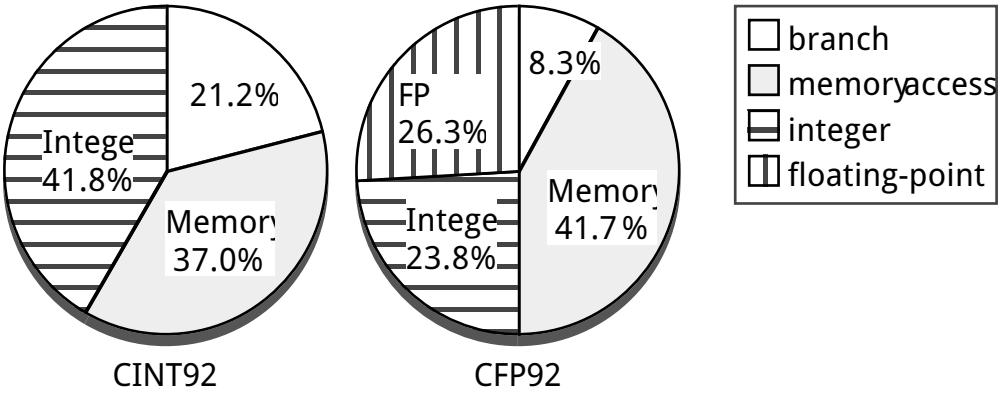


図 3　CINT92 , CFP92 における命令の分散

2.6. インオーダ発行スカラープロセッサ

比較対象のモデルとして、アウトオブオーダ実行を行なわないプロセッサモデルを作成する。このプロセッサの性能は我々が考えられる最高のパフォーマンスをもらたすRISC インオーダスカラープロセッサとで、スピードアップの比率はこのプロセッサの性能をもとに計算される。

このインオーダスカラープロセッサで、CINT92 , CFP92 を実行したときの IPC (1 サイクルあたりの平均命令終了数) はそれぞれ 0.945 と 0.75 である。

3. ルックアヘッドウインドウ (先見ウインドウ) の決定

3.1. ルックアヘッドウインドウの定義

ルックアヘッドウインドウは命令依存関係の解析をおこなうことのできる最大命令数で、発行バッファ、各機能ユニット、リオーダバッファに存在することのできる命令の最大値である。

以降の評価では、先に定義したアウトオブオーダ実行モデルを用い、コスト性能比の良いルックアヘッドウインドウのサイズを決定する。ルックアヘッドウインドウ決定に関して機能ユニットの数は無限大とするという仮定を用いる。

参考：PowerPC604のルックアヘッドウインドウは１６であるためリオーダバッファの１６エントリがうまってしまうと新たにディスパッチする事ができなくなる。

3.2. ルックアヘッドウインドウの評価 1 (CINT92)

図 4 に CINT92 のシミュレーション結果を示す。この図はデグリーが 2、4、6、8、10、∞ のそれぞれの場合について、ルックアヘッドウインドウを変化させたときの IPC (1サイクル当たりに終了した命令数の平均値) を示している。

図 4 から、ルックアヘッドウインドウが 100 (図中に線で示している。) より小さい時には、デグリーが 8 と ∞ で結果に明らかな差はみられない。デグリーが8の時にデグリーに見合ったIPCを得るにはルックアヘッドウインドウを十分大きくする必要があるが、予測された分岐命令との依存関係から、ルックアヘッドウインドウを大きくすることは難しい。 以降デグリー 8 以下の実行モデルに限定して調べていく事にする。

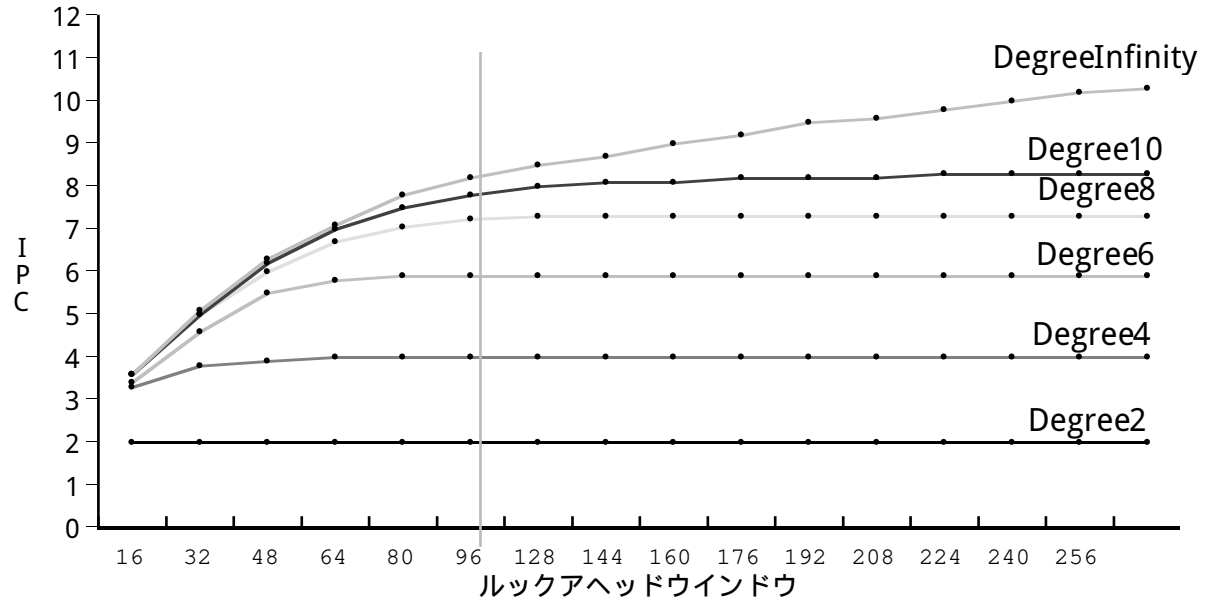


図 4　CINT92 の結果とデグリーの範囲

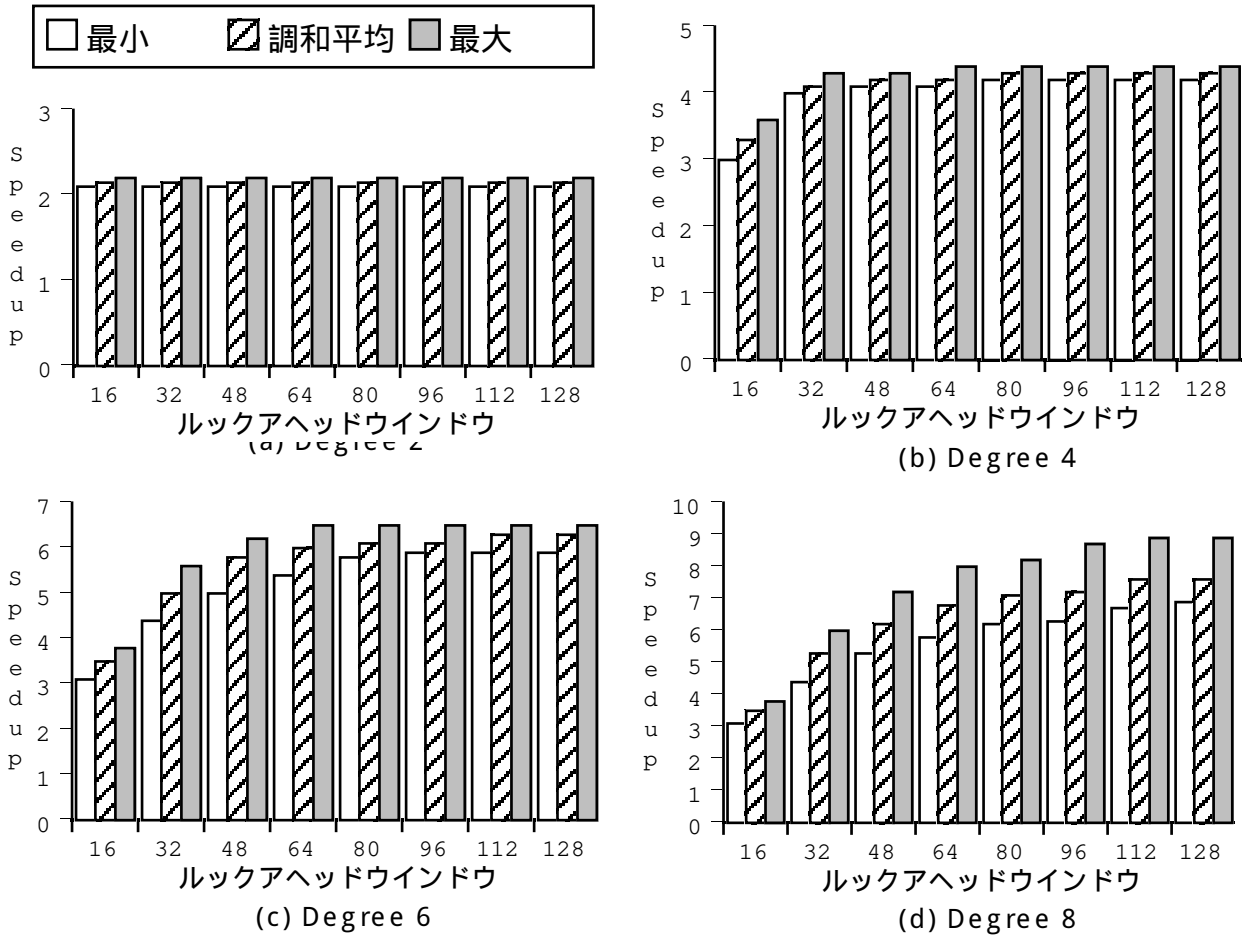


図 5　各デグリーにおけるルックアヘッドウインドウとスピードアップ

図 5 の (a) - (d) にルックアヘッドウインドウのサイズを変更した時のスピードアップを示す。ルックアヘッドウインドウの小さい所で IPC はデグリーの値と大きな差を生じる。このスピードアップの低下は命令の依存関係によるもので、デグリーの値が大きくなるほど顕著になる。

ルックアヘッドウインドウが256以上の時のシミュレーションもおこなったが、ルックアヘッドウインドウが100より大きいものはここ数年の間には実用的ではないと考えられるし、グラフが示すようにルックアヘッドウインドウを大きくすること（１００以上）で効果的なスピードアップを得ることはできない。

Degree	2	4	6	8
ルックアヘッドウインドウ	16	32	64	96
スピードアップ	2.11	4.09	5.91	7.36

表３　デグリーとルックアヘッドウインドウのトレードオフ

表３にルックアヘッドウインドウとスピードアップについて最良のトレードオフの結果をまとめる。テーブルに示したスピードアップの値はルックアヘッドウインドウが256の時の性能から 6%以内の性能低下におさまっている。

3.3. ルックアヘッドウインドウの評価２（ CFP92 ）

さきの CINT92 と同様の評価を CFP92 で行なう。
図６に１４個の不動小数点演算プログラムを実行したときの IPC を示す。

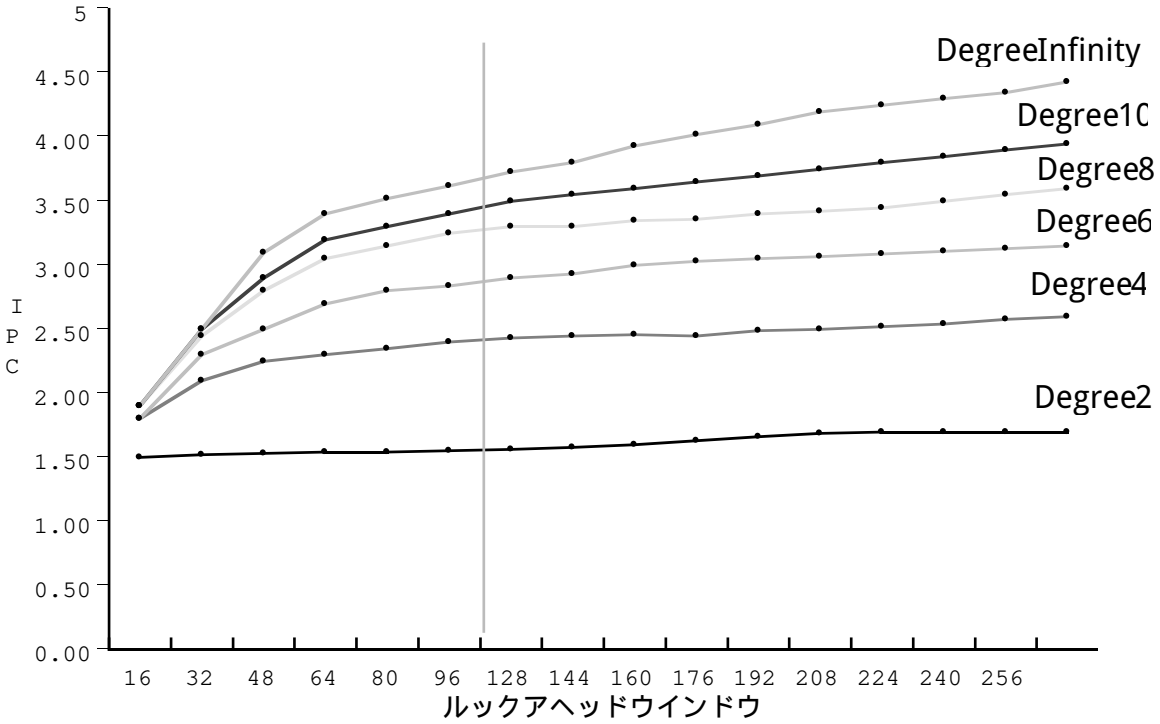


図６　CFP92 におけるルックアヘッドウインドウと IPC

それぞれのプログラムに関する結果は述べないが不動小数点プログラムには以下の特徴がある。

（１）プログラムによって、取り出しうる命令レベル並列度に大きな違いがある。
（２）ルックアヘッドウインドウが２５６以内では性能の飽和に達しない。

不動小数点命令はより長い実行サイクルが必要なため命令レベル並列度を取り出すためには十分大きなルックアヘッドウインドウが必要になる。

特徴（２）から不動小数点プログラムに関してはコスト性能比の優れた設定を決定することはできない。よって整数プログラムで得られたルックアヘッドウインドウを採用し表４にルックアヘッドウインドウが２５６の時からの性能低下をパーセント表示する。

Degree	2	4	6	8
ルックアヘッドウインドウ	16	32	64	96
性能低下（％）	16.22	23.25	19.21	15.95
スピードアップ	1.96	2.86	3.62	4.14

表４　デグリーとルックアヘッドウインドウの設定（不動小数点プログラム）

4. 整数ユニットとメモリユニットの構成

4.1. 整数ユニット（Universal Integer Unit）

一般的なプログラムでは、多くの命令は 整数とロード/ストア の命令クラスに属する。

1 サイクル整数命令とロード/ストア命令の和が全体に占める割合
- CINT92　78.6％
- CFP92　45.1％

整数ユニットの数とロード/ストアユニットの数がどれくらい必要かを評価する。
以下の 2 つのタイプのユニットを考え評価する。

- 1 サイクルで終了する全種類の整数命令を実行する整数ユニット
  （UniversalIntegerUnit）
- メモリシステムに複数アクセスポートを持った 単一のメモリユニット

このモデルでは、キャッシュミスが無いことを仮定しているためキャッシュの構成は定義しない。このことから、メモリ命令の発行に関する制限は使用可能なキャッシュポートの数という事になる。
図 7（次ページ）は 整数プログラムのシミュレーション結果で整数ユニットの数とキャッシュポートの数を評価している。

(a) はデグリーが 2 の時の様子を示す。 2 つの整数ユニットと 1 つのキャッシュポートで 95%の性能を得る事ができこの場合がコスト性能比の点でもっとも優れている。 2 つの整数ユニットと 2 つのキャッシュポート構成で、理想値に対して 99% の性能を得る事が出来る。
(b) はデグリーが 4 の時に 2 つ以上のキャッシュポートが必要となることを示している。このとき最もコスト性能比が高いのは 3 つの整数ユニットと 2 つのキャッシュポートの構成で、理想的な値に対して 92.1% の性能を得ている。
同様に デグリーが 6 の時は 4 つの整数ユニットと 3 つのキャッシュポートで 92% をカバーし、 デグリーが 8 の時は 5 つの整数ユニットと 4 つのキャッシュポートで理想的な値に対して 83% の性能比を得る。

強調すべき点としてあげられるのは、それぞれもう1つのキャッシュポートを加える事で約 5% の性能向上が得られるということである。しかしそのような改良はコスト性能比において良い方法ではない。

以上の結果を表 5 にまとめる。

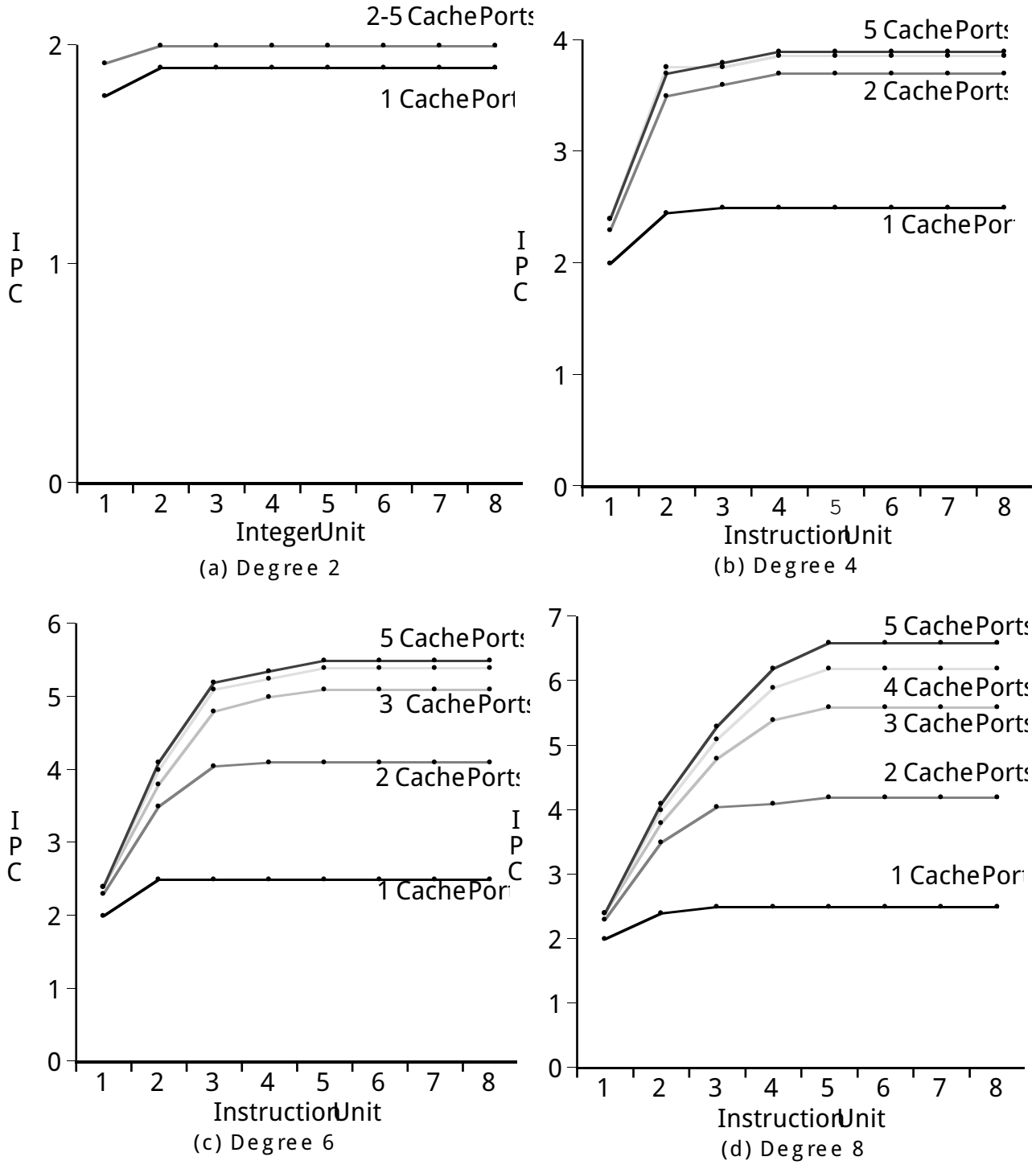


図 7　デグリーによるメモリーポートの決定

Degree	2	4	6	8
Integer/ Memory	2 / 1	3 / 2	4 / 3	5 / 4
Speedup(CINT92)	2	3.76	5.44	6.69
性能低下 (%)	5	7.9	8	9
Speedup(CFP92)	1.83	2.74	3.49	4.02
性能低下 (%)	6.7	4	3.7	3.1

表 5　整数ユニットとキャッシュポートの構成

4.2. 整数ユニットの詳細・シフトユニット

今まで、PowerPC604で取られているように1サイクルの整数命令はすべて 1 種類の整数ユニットで行なう方式 (UniversalIntegerUnits)をみてきた。ここでは、MC88110のように1サイクル整数命令を幾つかの機能ユニットでおこなう方式を検討する。

図 8 において

・AS (Arithmetic,logic andShift) は算術演算、論理演算、シフトをおこなう機能ユニット
・A (Arithmeticandlogic) は算術演算と論理演算をおこなう機能ユニット
・S (Shift) はシフトだけをおこなう機能ユニットである。

図 8 の各グラフにおいて右の構成ほど実現コストが高くなる。

シミュレーションの結果よりデグリー 2 と 4 のときは一つの AS と残りを A で構成することで、全てを AS で構成する時を100%として99.8% 以上の性能を得ている。デグリーが 6 と 8 の時は 2 つの AS ユニットを用いるのが最もコスト性能比が良い。

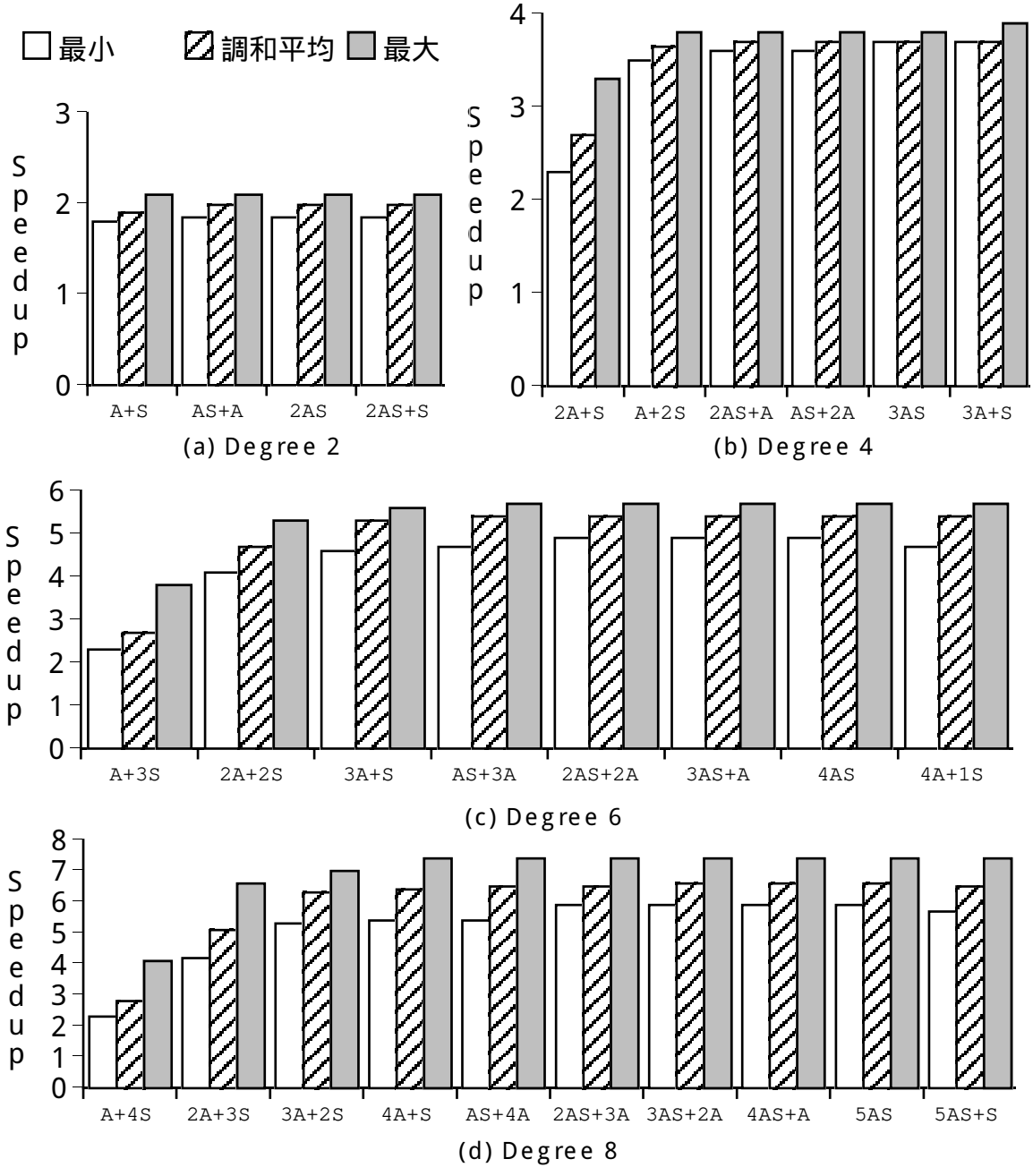


図 8　整数ユニットの構成方法

今までのシミュレーション結果を考慮し、表 6 にコスト性能比が優れた整数ユニット構成をまとめる。

Degree	lookaheadwindow	CachePort	A	AS	Multi-cycle	Speedup
2	16	1	1	1	1	2
4	32	2	2	1	1	3.75
6	64	3	2	2	1	5.44
8	96	4	3	2	1	6.69

A: Arithmetic andlogic

AS: Arithmetic logic and Shift

表 6　整数ユニット機能構成とキャッシュポート設定

5．浮動小数点演算ユニットの構成

ほとんどのスーパースカラプロセッサで、不動小数点命令はすべて専用の不動小数点ユニットを用いて実行される。今から考える不動小数点ユニットは足し算、掛算、除算等すべての命令を同一のユニットで処理するものとする。これより多くの命令レベル並列度を抽出するためには複数の不動小数点演算ユニットが必要となる。

図 8 に不動少数演算ユニットの数を変化させたときのスピードアップを示す。デグリーが 2 の時は 1 つの演算ユニットで、理想的な構成の 93% の性能を得ることができる。

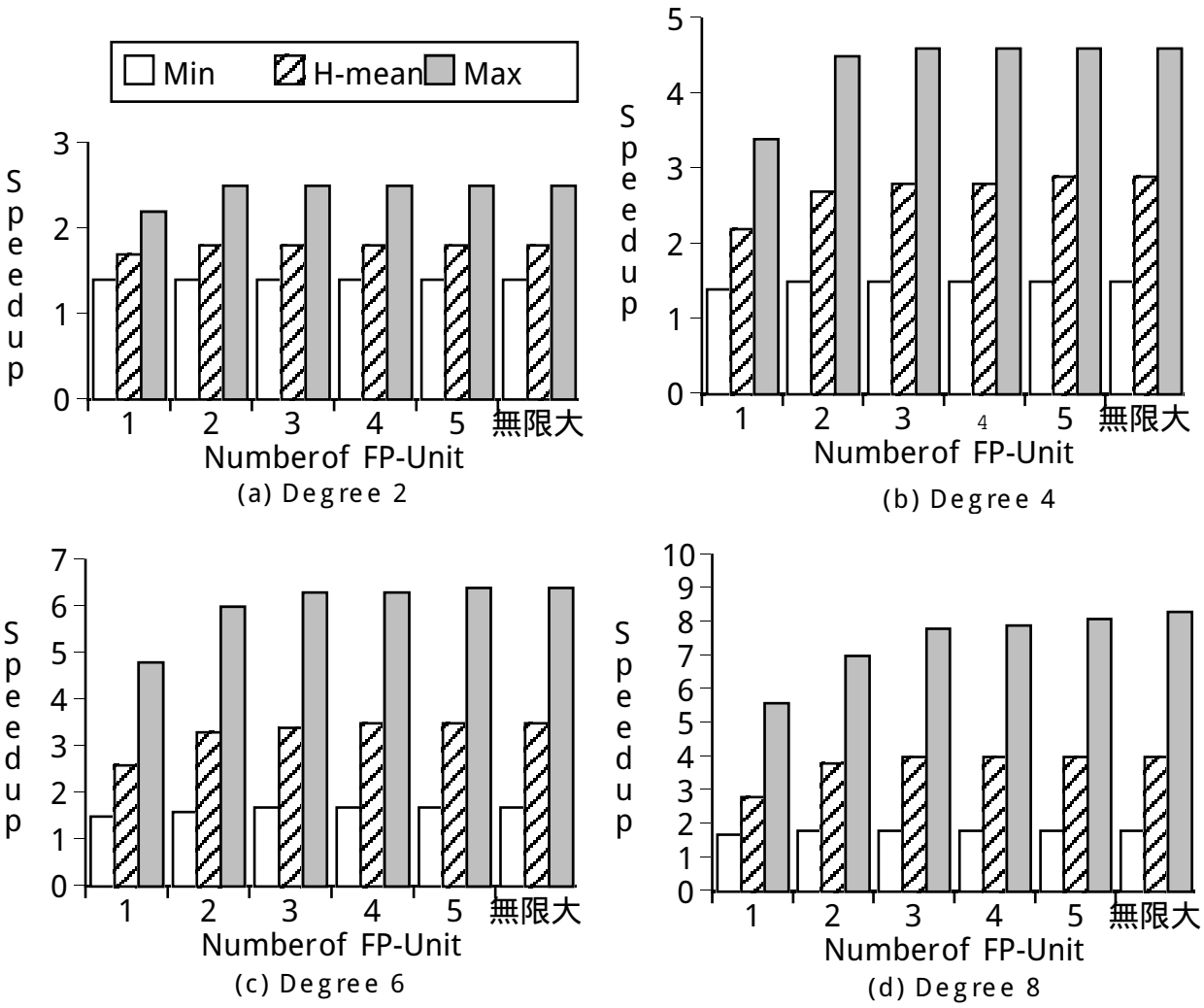


図 9　不動小数点ユニット数の決定

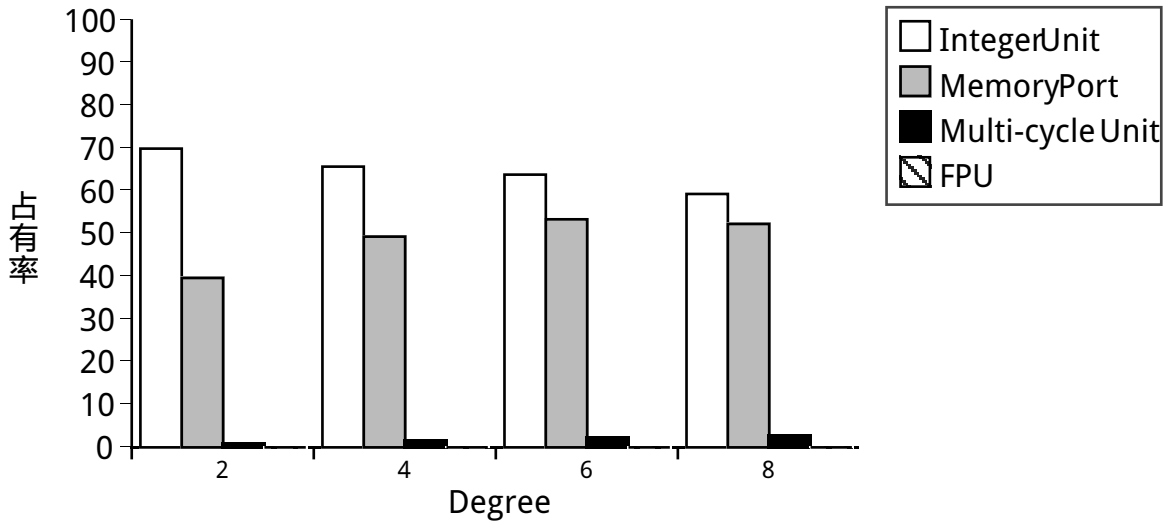
デグリーが 4, 6, 8 のときは 2 つの演算ユニットでそれぞれ 96%, 94%, 92% の性能に達し、これらの演算ユニット数がコスト性能比でもっとも優れている。
表 7 に不動小数点ユニットの数とそのときのスピードアップを示す。

Degree	2	4	6	8
FP-Unit	1	2	2	2
Speedup	1.7	2.63	3.29	3.68

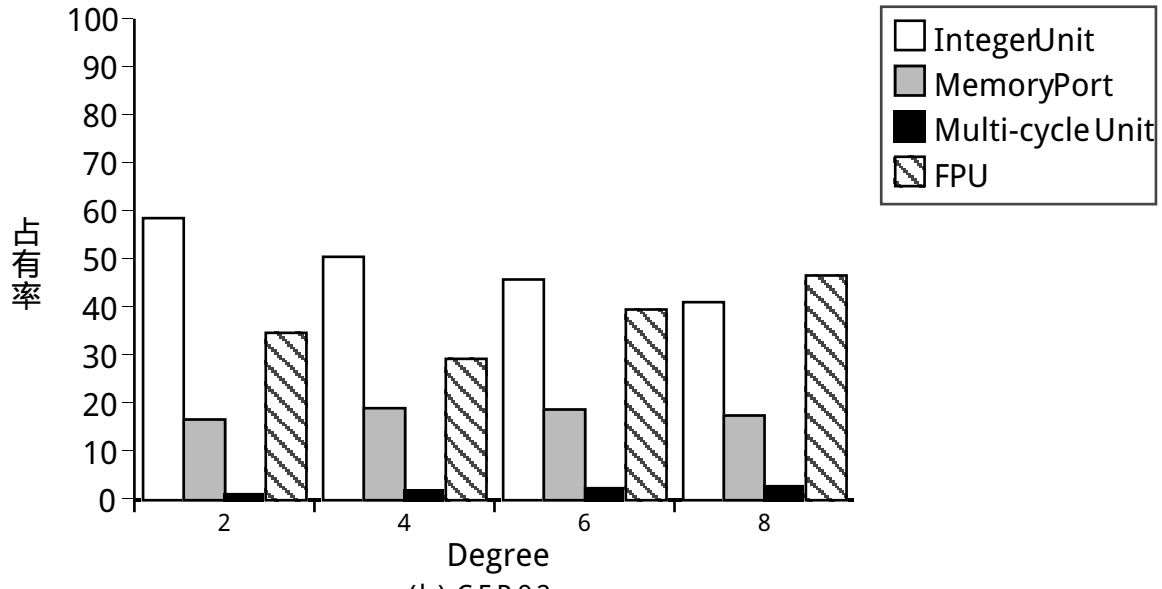
表 7　不動小数点演算ユニットとの構成

6. 機能ユニットの実行占有率

図１０では、先に定義した実行モデル構成での各機能ユニットの占有率を示す。マルチサイクル命令（整数の掛算、除算）をおこなうマルチサイクルユニットが全体を通じてほとんど使われていないことがわかる。



(a) CINT92



(b) CFP92

図１０　各ユニットの占有率

7. 結論

本発表では、コスト性能比の優れた機能ユニット構成を決定するためのシミュレーション結果を示した。ただし今回扱った実行モデルは 4 つの仮定を用いたアウトオブオーダ実行モデルであった。

今までの実験により設定された構成を表 8 にまとめる。

Degree	2	4	6	8
ルックアヘッドウインドウ	16	32	64	96
AS	1	1	2	2
A	1	2	2	3
Multi-cycle Unit	1	1	1	1
FPU	1	2	2	2
Data Cache Port	1	2 (1)	3 (1)	4 (1)
Total Number of Unit	5	7	8	9
Speedup(CINT92)	2	3.75	5.44	6.69
Speedup(CFP92)	1.7	2.63	3.29	3.68

AS:算術、論理シフト

A:算術、論理

表 8　最終的な構成

(1) は Date Cache のユニット数が 1 であることを示す。

図 1 2 に PowerPC604 の性能を 1 としたとき、今回求めた設定でどれだけスピードアップするか示す。

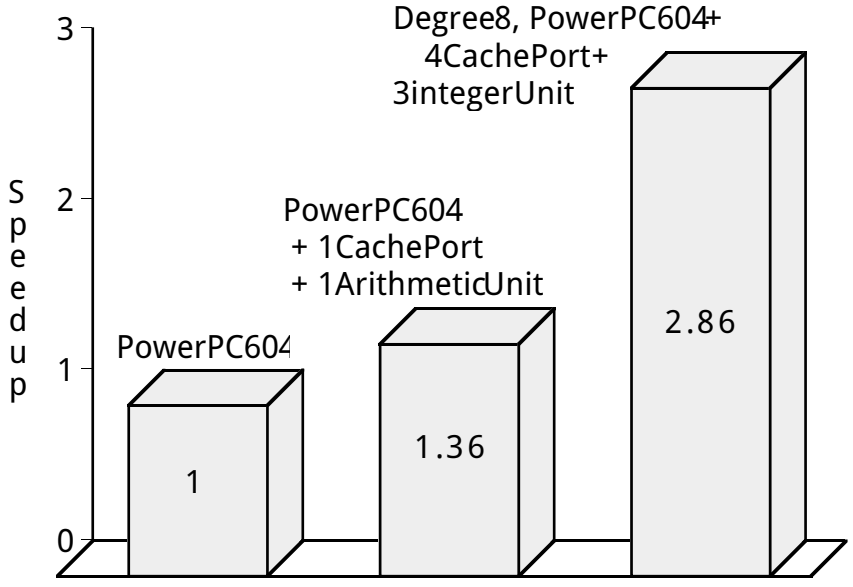


図 1 2　CINT92 を実行したときの PowerPC604 に対する相対的スピードアップ

	Degree	同時命令実行数	Single-Cycle Unit	Multi-Cycle Unit	FP Unit
PowerPC604	4	16	2	1	1
今回求めた設定	4	32	3	1	2

表 9　今回求めた設定と現実のプロセッサによる設定

8. 考察

PowerPC604の構成は今回求めた構成と異なる。シミュレーションでは以下の４つの仮定より考慮するハードウェアがシンプルになったこと、実際の製品では実現できるゲート数という制約が存在することが主な原因と考えられる。

今回用いた仮定

（１）理想的な命令フェッチ機構を考える。命令キャッシュのミスは発生しないし、分岐予測の失敗は起こらない。
（２）ディスパッチされる命令数はプロセッサのデグリーとリオーダバッファがあふれたときにのみ制限される。
（３）データキャッシュのミスは起こらない。
（４）命令発行バッファは一つに統一され、命令発行アルゴリズムには古いものを優先する oldestfirst を採用する。

現実のマイクロプロセッサ設計において、分岐命令をどう扱うか（分岐予測、投機実方式）は重要な問題である。 CINT92 においては条件分岐だけで全体の 18.7%, CFP92 では 7.3% が分岐命令であり、これらの影響を無視したデータと現実の製品を比べることには無理がある。しかし今回導き出された設定により最高で PowerPC604の約３倍のスピードアップ（４つの仮定を付け、コスト性能比で優れた設定で）が得られるということが明らかになった。

現段階で、デグリー 8 ,機能ユニット 9 という設定はハードウェア的に無理な要求であるが、ソフトウエアとハードウェアの協力によりハードウェア量の削減の道も残されている。

案外近い将来にこの設定に匹敵するプロセッサが登場するかもしれない。

参考文献

[1] David A.Patterson,John L. Hennessy,"Computer Architecture: A Quantitative Approach." Morgan Kaufmann Publishers Inc.
[2] Mike Johnson（村上和彰訳）,"Superscalar Microprocessor Design." Prentice-Hall,Inc.（日経ＢＰ出版センター）
[3] IBM, "PowerPC604 RISK Microprocessor Technical Summary", IBM Advancd Information, MPR604TSU-1, 1994
[4] R.M. Tomasulo,"An efficient Algorithm for Exploiting Multiple Arithmetic Units", IBM Jour vol.11, January 1967.
[5] Stephen Melvin, Yale Patt, "Exploiting Fine-Grained Parallelism Through a Combination of Hardware and Software Techniques" The 18th Annual International Symposium on Computer Architecture, 1991.
[6] Michael Butler, Tes-Yu Yeh, and Yale Patt "Single Instruction Stream Parallelism Is Greater than Two"
The 18th Annual International Symposium on Computer Architecture, 1991.
[7] M. Bulter and Y. Patt, "An Investigation of the Performance of Various Dynamic Scheduling techniques", Proceedings of the 25th Annual International Symposium on Microarchitecture, December 1992