システムＬＳＩ設計技術　93－3
（1999. 11. 26）

オンチップマルチプロセッサアーキテクチャSKYの評価

小川 行宏、小林 良太郎、安藤 秀樹、島田 俊夫

名古屋大学大学院工学研究科

〒464-8603 愛知県名古屋市千種区不老町
名古屋大学大学院工学研究科 電子情報学専攻 島田研究室
Tel: 052-789-3170, FAX: 052-789-3168
email: {ogawa,kobayasi,ando,shimada}@shimada.nuee.nagoya-u.ac.jp

あらまし　我々は非数値計算プログラムに対し、マルチスレッド実行により性能を向上させるオンチップ・マルチプロセッサ SKY を提案してきた。本論文では SKY に関する詳細な性能評価、解析を行なった。評価の結果、SKY は単一のスーパスカラ・プロセッサに対して大きな性能向上を達成しているが、ベンチマーク・プログラムによって性能向上率に大きな差が見られることがわかった。この原因を命令ウィンドウから発行される命令を調べることにより解析した。その結果、並列に実行可能なスレッドの有無、および、マルチスレッド実行が命令レベルの並列性に与える影響の大きさによって、性能向上率が大きく変化することがわかった。

キーワード　マルチスレッド、マルチプロセッサ、スレッドレベル並列性

Evaluation of an On-Chip Multiprocessor Architecture SKY

Yukihiro Ogawa, Ryotaro Kobayashi, Hideki Ando, Toshio Shimada

Graduate School of Engineering, Nagoya University

Shimada Laboratory, Department of Information Electronics,
School of Engineering, Nagoya University
Furo-cho, Chikusa-ku, Nagoya 4648601, Japan
Tel: +81-52-789-3170, Fax: +81-52-789-3168
email: {ogawa,kobayasi,ando,shimada}@shimada.nuee.nagoya-u.ac.jp

Abstract　We have proposed a processor architecture, called SKY, which efficiently executes multiple threads in parallel for non-numeric programs with multiple processors. This paper shows comprehensive performance evaluation results and analysis of SKY. Our evaluation results show that SKY architecture achieves good performance improvements over a single superscalar processor, but a large difference of speedups is found among benchmark programs. We analyze what causes the difference by investigating instructions issued from the instruction window. We then found that the speedup significantly changes depending on existence of threads that can be executed in parallel and interference of multithreading execution with exploiting instruction-level parallelism.

key words　multiprocessor, multithread, thread-level parallelism

1. はじめに

プロセッサの処理能力は、プログラム内に存在する並列性を利用することで向上させることができる。プログラム並列性を制約する要因は、そのプログラムに存在する依存関係である。依存関係にはデータ依存、制御依存がある。これらの依存の中で、並列性を制限する要因としては制御依存が最も大きい。

現在、商用の主なスーパスカラ・プロセッサは、この制御依存を解消する方法として**投機的実行**を用いている。これは分岐先が確定する前に分岐方向を予測し、その方向の命令を実行する技術である。しかし、現在のスーパスカラ・プロセッサが引き出している命令レベルの並列性(**ILP**: Instruction-Level Parallelism)は、Wallらが示した単一制御流において利用可能なILPの限界に近づきつつある[1]。

これに対してLamらは、投機的実行に加えて、制御依存解析と複数命令流実行を導入すれば、制御依存を大幅に緩和できるという研究結果を示した[2]。

そこで、我々はこれら3つの技術を利用したアーキテクチャ**SKY**を提案してきた[3][4][5]。SKYは、複数の独立したプロセッサを持ち、各プロセッサ間には高速なレジスタ通信機構を備えたアーキテクチャである。各プロセッサで**スレッド**を実行することにより複数命令流実行を実現する。スレッドとは、コンパイラによって分割されたプログラムの断片である。コンパイラは、プログラムの制御依存解析などを行うことによってプログラムを複数のスレッドに分割する。従来の投機的実行は各プロセッサによって行なわれる。このようにしてSKYは投機的実行、制御依存解析、複数命令流実行という3つの技術を融合し、性能向上を図っている。

我々はSKYに関して、以前の論文[3][4]において性能に関する詳細な評価は行なっていない。そこで本論文ではSKYの詳細な性能評価を行う。

評価により、SKYは単一のスーパスカラ・プロセッサに対して大きな性能向上を達成しているものの、ベンチマーク・プログラムによって性能向上率に大きな差が見られることがわかった。また、プロセッサ資源の投資に対して性能向上のスケーラビリティが存在しないことがわかった。本研究では、SKYの性能を制限する要因について解析した。具体的には、命令ウィンドウから発行される命令を調べることにより解析した。また、SKYの性能を制限する要因を緩和するモデルを示し、評価する。

2章ではSKYの概要について述べる。3章ではSKYの基本性能を評価し、プロセッサの性能を制限する要因の分類を行なう。4章でプロセッサの性能を制限する要因に関する評価をする。5章でマルチスレッド実行における制約を緩和する簡易的なモデルを示し、予備的な評価を行う。6章で本研究をまとめる。

2. SKYアーキテクチャの概要

本章では、SKYアーキテクチャの概要について述べる。

SKYは、図1に示すようにリング・バスで結合された複数のスーパスカラ・プロセッサからなる。細粒度のスレッドレベル並列性(**TLP**: Thread-Level Parallelism)を利用するため、プロセッサ間でレジスタ値を直接送受信し、同期／通信のオーバヘッドを2サイクルまで減少させている。また、**命令ウィンドウ・ベース**の同期と呼ぶ新しい同期機構を導入している。この機構は、命令ウィンドウ上で受信値に対応するタグを用いてレジスタに関する同期をとる機構である。この機構により、同期により後続命令の実行がブロッキングされることはなく、プロセッサはILPを最大限利用することができる。

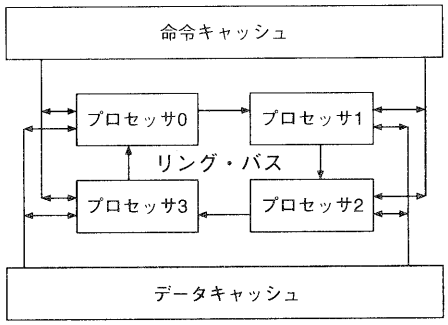

図1 SKYの構成

スレッド並列実行のためのオーバーヘッドを小さくするため、SKYでのマルチスレッド・モデルは通常のマルチスレッド・モデルと比べて次に示す制約を課している。これは、マルチスカラ・アーキテクチャ[6][7]やMUSCAT[8]と同様のモデルである。

- 各スレッドは、逐次実行における動的に連続する部分で構成される。
- 各スレッドは、逐次実行の順において自分の直後のスレッドを生成する。

図2(a)に、逐次実行命令列に対するSKYのスレッド分割を示す。同図に示すように、SKYにおける各スレッドは、動的な命令列における単一の連続した部分からなり、異なる複数の部分からは構成されない。したがって、スレッドに結合はなく、制御に関する同期は必要ない。

図2(b)に示したスレッドを並列に実行する様子を、図2(c)に示す。図2(b)において、各スレッドT_0、T_1、T_2と逐次実行の順に名前をつける。図2(c)に示すように、スレッドT_iは実行途中で、スレッドT_{i+1}を生成するという逐次生成を繰り返す。各スレッドは実行中には高々1回しか新しいスレッドを生成しない。実行のできるだけ早い時期に新しいスレッドを生成することにより、スレッドの並列実行を実現する。スレッドの生成は、`fork`命令と呼ぶ専用の命令を用い、終了は`finish`と呼ぶ専用の命令を用いて行う。以下、T_{i+1}をT_iの子スレッドと呼び、T_iをT_{i+1}の親スレッドと呼ぶ。

SKYの同期／通信機構は命令レベルで行う。通信には`send`命令と呼ぶ専用の命令を用いることで、親スレッドから子スレッドへデータを送信する。

SKY専用命令はコンパイラによって挿入される。コンパイラは、スレッドの並列実行による性能利得が大きくなるようにプログラムを分割し、`fork`、`finish`命令

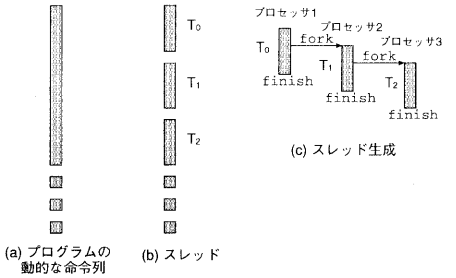

図2　SKY のマルチスレッド実行

を挿入する。また、通信すべきレジスタを求めて **send** 命令を挿入する。**fork**、**finish** 命令の挿入はプログラムの制御依存解析を行い、制御依存のないところを挿入候補とする。すべてのフォーク可能な箇所について性能利得を見積り、それが大きなものについてのみ命令を挿入する。**send** 命令の挿入はフォーク候補に **send** 命令挿入[5]アルゴリズムを用いて行う。詳細は文献3)5) を参照されたい。

3. SKY の性能を制限する要因

本章では SKY の基本モデルの評価を行ない、SKY の性能を制限する要因について検討する。まず評価環境について述べ、SKY の基本モデルの評価を行い、SKY の性能を制限する要因として IPC の損失について検討する。そして性能を制限する要因について分類する。

3.1 評価環境

ベンチマーク・プログラムとして、SPECint95 の 8 種類を使用した。NEC EWS4800/360AD(MIPSR4400) のコンパイラで最適化し、MIPS R2000[10] 用のコードを生成し、これを測定に用いた。評価はトレース駆動型シミュレータを作成し行った。トレースは pixie[11] を用いて採取した。

表 1 に SKY の基本モデルを示す。以下、特に断らない限り、評価においてはこの基本モデルを使用する。性能比較における基準プロセッサは、SKY を構成する 1 つのプロセッサ、つまり 8 命令発行の単一スーパスカラ・プロセッサとする。表 2 に、各ベンチマーク・プログラムにおける基準プロセッサの IPC を示す。

3.2 SKY の基本性能

本節では基準プロセッサ及び SKY の性能評価を行う。

図 3 に SKY の性能の評価結果を示す。縦軸は基準プロセッサに対する性能向上率である。プロセッサ数が 2 と 4 の場合について性能を測定した。比較のため基準マシンの 2 倍の資源 (例えば、機能ユニット、命令ウィンドウのエントリ数など) を投入した 16 命令発行の単一スーパスカラ・プロセッサの性能 (図 3 の 16-issue) についても測定した。このプロセッサのハードウェア量は、2 プロセッサ構成の SKY のハードウェア量とほぼ等しい。

基準プロセッサと比較すると、2 プロセッサ構成の SKY は最大で 46.1%、幾何平均で 21.5% の性能向上率を達成している。特に、compress95、ijpeg、m88ksim では、

表1　SKY の基本モデル

(a) プロセッサ

命令発行幅	8 命令
命令フェッチ幅	8 命令
命令ウィンドウ	64 エントリ
リオーダバッファ	128 エントリ
機能ユニット	Int×8、Load/Store×4、Branch×2、FP ALU×8、Send×4
分岐予測ミスペナルティ	4 サイクル

(b) 共有リソース

命令キャッシュ	2 ポート、各ポート 8 命令、常にヒット
データキャッシュ	4 ポート、各ポート 1 つのデータアクセス、常にヒット
メモリのデータ依存	ハードウェアで暗黙的に解消
分岐予測機構	1024 エントリ 4 履歴の PAp 予測器 1024 エントリ、連想度 2 の BTB 32 レベルのリターンアドレススタック

表2　基準プロセッサの性能

ベンチマーク	IPC
compress95	2.99
gcc	2.16
go	1.92
ijpeg	3.70
li	2.70
m88ksim	2.06
perl	2.44
vortex	2.91
g-mean	2.24

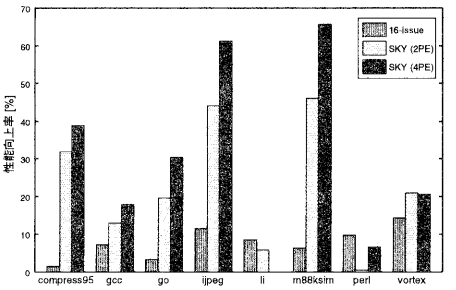

図3　基準プロセッサに対する SKY の性能

性能向上率は 30% 以上と大きい。また、2 プロセッサ構成の SKY は同程度の資源を投入したスーパスカラ・プロセッサよりも最大で 37.3%、幾何平均で 12.7% 性能向上率を達成している。

3.3 IPC 損失

SKY は大きな性能向上を達成しているものの、プログラムによって性能向上率に大きな差が見られる。ijpeg、m88ksim、compress95 では大きな性能向上を達成しているものの、逆に li と perl ではほとんど性能が向上していない。このような性能差を生じる原因を IPC の損失という点から議論する。

スーパスカラ・プロセッサにおいて 1 サイクルに同時に発行可能な最大命令数は、命令ウィンドウから機能ユ

ニットへの命令発行幅である。本論文では、この1サイクルに同時に発行可能な命令数を**理想IPC**と呼ぶ。SKYにおける理想IPCは1プロセッサあたりの理想IPCに×プロセッサ数である。例えば、基準プロセッサの理想IPCは8、2プロセッサ構成のSKYの理想IPCは16となる。また**スロット**を次のように定義する。1つのプロセッサの理想IPCが8とすると、「このプロセッサは8つの発行**スロット**がある」ということとする。

理想IPCは以下の式のように2つの項目に分解できる。

$$\text{理想 } IPC = (\text{実際の } IPC) + (IPC\text{損失})$$

実際のIPCとは1サイクルに発行された有効な命令数のことである。命令発行スロットは、実行時において全て有効な命令によって埋められるわけではなく、データ依存などの種々の要因で埋められないことがある。**IPC損失**とは命令発行ユニットの空きスロット数のことをいう。

IPC損失は何らかの要因によって引き起こされる。本研究では、このIPC損失の要因を分類し、SKYの性能について議論する。

3.4 IPC損失を引き起こす要因の分類

本節では、IPC損失を引き起こす要因の分類を行う。IPC損失を引き起こす要因は、その性質から大きく2種類に分類できる。一つはスーパスカラ・プロセッサ実行による要因であり、一つはマルチスレッド実行による要因である。

スーパスカラ・プロセッサ実行によって引き起こされるIPC損失の要因は以下の3つである。

- 資源競合
  他の命令により機能ユニットが使用中であり、機能ユニットを使用することができないため、命令ウィンドウから命令発行を停止することに起因するIPC損失。
- データ依存
  先行命令に対して真のデータ依存関係を持つことにより命令の発行が不可能となることに起因するIPC損失。
- 制御依存(分岐予測ミス)
  分岐予測ミスによるIPC損失。

マルチスレッド実行によって引き起こされるIPC損失の要因は以下の5つである。

- スレッド生成
  スレッドの実行開始時に十分な命令がプロセッサに供給されないことに起因するIPC損失。スレッド・サイズが小さいプログラムほどスレッド生成によるIPC損失は大きい。
- スレッド終了
  `finish`命令がリオーダ・バッファからリタイアするまでプロセッサが解放されないため、スレッドの実行終了時に十分な命令がプロセッサに供給されないことに起因するIPC損失。スレッド・サイズが小さいプログラムほどスレッド終了によるIPC損失は大きい。
- ブロッキング
  スレッド間のデータ依存が原因で命令ウィンドウへの命令供給が止まってしまうことに起因するIPC損失。SKYの命令ウィンドウ・ベースの同期機構においては発生しない。従来のfull/empty ビットを使用したレジスタ・ベースの同期機構[6][7]において発生する。
- スレッド間依存
  親スレッドの命令に対して真のデータ依存関係を持つことにより命令の発行が不可能となることに起因するIPC損失。スレッド間依存によるIPC損失が大きくなる原因として、コンパイラが何らかの理由で誤ってTLPの少ないスレッド分割を行ってしまった場合である。
- スレッド不在
  プロセッサにスレッドが存在しないため実行すべき命令が存在しないことに起因するIPC損失。スレッドが存在しない原因として、コンパイラがプログラムよりプロセッサ数に見合っただけの十分なTLPを抽出することができなかった場合である。

次章で、各要因別にIPC損失を測定し、SKYの性能を制限する要因について議論する。

4. 性能評価

本章では、SKYにおけるIPC損失について定量的に測定する。続く各節では以下に示す項目に着目して評価した。

- 2プロセッサ構成のSKYの性能
- プロセッサ台数を変化させた場合の性能
- SKYの同期機構
- 同期/通信のオーバヘッド

4.1 2プロセッサ構成のSKYの性能

図4にスレッド不在、スレッド間依存によるIPC損失を示す。これら2つの要因によるIPC損失は、コンパイラによるTLP抽出に関係がある。

図4より、スレッド間依存によるIPC損失は非常に少ない。このことよりコンパイラは、TLPの少ないスレッド分割は行っていないことがわかる。スレッド不在によるIPC損失は平均で22.1%を占め性能に大きな影響を与えていることがわかる。また、perlでは理想IPCのほぼ半分(41.4%)がスレッド不在によるIPC損失で占められている。つまり、ほとんどの時間、スレッドは2つプロセッサの一方でしか実行されていない。これがperlで性能向上がほとんどない原因である。逆に、ijpegではスレッド不在によるIPC損失が2程度と少なく、大きな性能向上を達成している原因である。

以上のように性能向上率は、コンパイラがプログラムよりTLPを抽出できたかどうかに依存していることがわかる。コンパイラがTLPを抽出できなかった理由は、3つ考えられる。1つは我々の現在のコンパイラの能力不足である。たとえば次の点で能力に不足がある。

- コンパイラは、制御等価[9]なブロックの組に着目している。これは制御依存がないブロックの組の一部であり、解析範囲が制限されている。
- 関数毎に解析を行っている。このため、フォーク点を含む関数の外に子スレッドの開始点を設けることはできない。

2つめの理由はSKYのマルチスレッド実行における制約である。特にスレッドの逐次実行の制約(1回フォークモデル)は単純化には貢献しているが、TLPの抽出には大きな制約を課している。

3つめの理由は、プログラム中に内在するTLPがそもそも十分には存在しないということである。存在しないTLPをコンパイラは引き出すことはできない。これを証明することは非常に難しいが、今後調査の必要がある。

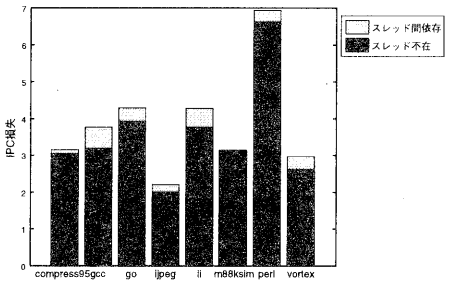

図4　コンパイラのTLP抽出に関するIPC損失

図5にスレッド生成、スレッド終了によるIPC損失、及び分岐予測ミスによるIPC損失の基準プロセッサに対する増加分を示す。これら3つの要因によるIPC損失は、マルチスレッド実行がILP利用にどの程度悪影響を与えているかに関係がある。マルチスレッド実行により分岐予測精度が低下することから、単一スーパスカラに対する分岐予測ミスによるIPC損失の増加分を追加してある。また表3に平均スレッド・サイズを示す。単位は命令である。

図5よりTLPの利用が少ないperlを除いて、マルチスレッド実行によるIPC損失は1.5～2.5と非常に大きい。一般的にスレッド・サイズが小さいプログラムほど、スレッドの開始と終了によるIPC損失が大きい。スレッド生成によるIPC損失は、スレッド・サイズが小さくなるにつれ大きくなる。スレッド終了によるIPC損失は、スレッド・サイズに関係があるものの、プログラムの性質が大きく作用するものと考えられる。また、マルチスレッド実行による分岐予測精度の低下により分岐予測ミスによるIPC損失が増加する。SKYでは、分岐予測機構をプロセッサ間で共有している。静的に一つの分岐が異なるスレッドで同時に実行される場合、PAp予測器の分岐履歴表(BHT:branch history table)には、逐次実行とは異なる順で履歴が記録される。このような場合、逐次実行において存在した履歴パターンと将来の分岐方向の間の相関を、PAp予測器はとらえることができなくなり、予測を誤る確率が高まる。

図6に、マルチスレッド実行によるILP利用への悪影響によるIPC損失と性能向上の相関を示す。TLP利用の少ないperlについては、この議論の対象外なので、プロットしてない。同図より、IPC損失が多いほど性能向上率が低いという相関があることがわかる。ベンチマーク・プログラム別に見ると、図4よりliはTLPを比較

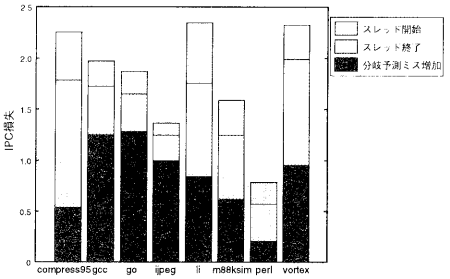

図5　マルチスレッド実行がILP利用に及ぼす悪影響

表3　スレッド平均命令数

ベンチマーク	命令数
compress95	193.3
gcc	256.0
go	263.6
ijpeg	1218.1
li	128.2
m88ksim	234.7
perl	292.9
vortex	250.6

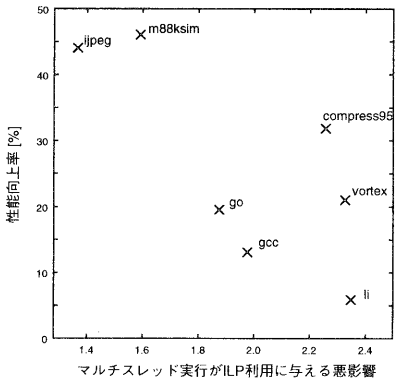

図6　IPC損失と性能向上率の相関

的多く利用しているが、マルチスレッド実行がILPの利用に及ぼす悪影響が大きく、その結果性能が向上していない。逆にgoは、liと同程度TLPを利用しているが、マルチスレッド実行がILPの利用に及ぼす悪影響が小さいので、liより高い性能向上率を達成している。また、liと同程度のマルチスレッド実行によるIPC損失量であっても、compress95、vortexはより多くのTLPを利用しているので、liより高い性能向上率を達成している。m88ksimとijpegは、TLPの利用率が大きく、かつ、マルチスレッド実行による悪影響が小さいので、他のプログラムに比べ大きな性能向上を達成している。

4.2　プロセッサ台数を変化させた場合の性能

前章の図3に示したように、4プロセッサ構成のSKYは基準マシンに対して最大65.6%、幾何平均で28.2%の

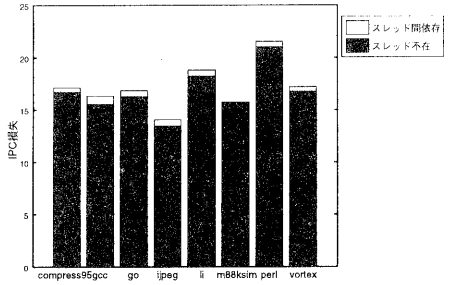

図7　スレッド不在、スレッド間依存によるIPC損失

性能向上率を達成するが、2プロセッサ構成のSKYに対して最大13.4%、幾何平均で5.5%しか性能が向上しておらず、コスト性能比が良いとはいえない。これはスレッド不在によるIPC損失が主な要因である。図7にスレッド不在、スレッド間依存によるIPC損失を示す。どのプログラムでもスレッド不在によるIPC損失が2プロセッサ分のスロットである16程度である。理想IPCに占めるスレッド不在によるIPC損失の割合は2プロセッサ構成のSKYは平均22.1%であるのに対して、4プロセッサ構成のSKYは52.2%と2プロセッサ構成のSKYに比べて4プロセッサ構成のSKYはスレッド不在によるIPC損失が大きい。2台を越えるプロセッサを効率良く利用するには、TLP抽出に関してさらなる検討が必要である。

4.3 SKYの同期機構の性能

本節では、同期機構を評価する。2プロセッサ構成のSKYアーキテクチャを基本とし、以下の3つのモデルについて評価した。

Cモデル: このモデルでは、子スレッドの開始点に到達するレジスタ値が現スレッドにおいてすべて定義されるまで子スレッドを生成しない。子スレッドを生成する時は、レジスタ・ファイルの内容をすべて後続プロセッサにコピーする。レジスタ転送のバンド幅は十分にあるとし、コピーには1サイクルしかかからないと仮定した。これにより、一旦スレッドが生成されると、レジスタに関する同期は必要ない。このため、SKYを含めこれまでのレジスタ値通信を実現してきたプロセッサと異なり、個々のレジスタについての同期／通信機構が必要なく単純である。しかし、通信データの粒度を全レジスタにまで粗くしたため、同期に無駄なオーバヘッドが生じている。

Bモデル: このモデルでは、SKYと同様、個々のレジスタを通信することができるが、受信待ち命令の後続命令の実行がブロックされる。このモデルは、full/emptyビットによる既存の同期モデル[6][7]である。

Nモデル: 命令ウィンドウ・ベースの同期機構を有し、ブロッキングの生じない同期を実現する。SKYで実現されているモデルである。

図8に評価結果を示す。縦軸は基準プロセッサに対する性能向上率である。Nモデルは、他のどちらのモデルより、全てのベンチマーク・プログラムにおいて高い性能を示している。Cモデルは、平均でわずか8.7%しか性能が向上しなかった。Bモデルの性能向上率も、12.8%とあまり大きなものでない。Nモデルは Bモデルより最大で16.8%、平均で7.7%高い性能を達成している。

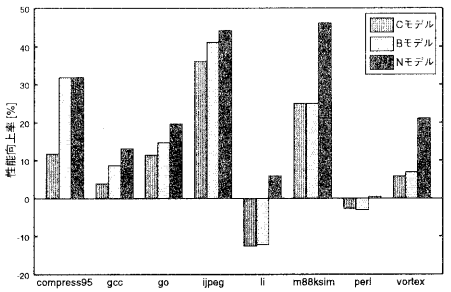

図8　同期機構の性能

図9に各同期機構におけるスレッド不在、ブロッキング、スレッド間依存によるIPC損失を示す。Cモデル、Bモデルではスレッド間のデータ依存が原因で命令ウィンドウへの命令供給が止まってしまうブロッキングにより性能が低下していることがわかる。

Cモデルでは理想IPCに対してブロッキングによるIPC損失は平均12.3%を占める。特に、liにおいて19.1%と大きく、これにより実際のIPCを低下させていることがわかる。このことより、レジスタ通信バンド幅がたとえ十分にあったとしても、全レジスタという粗い粒度での同期は性能低下の大きな要因となることがわかる。

Bモデルでは理想IPCに対してブロッキングによるIPC損失は平均7.5%を占める。特に、liでは12.0%、vortexでは14.5%を占める。

Nモデルではスレッド間依存によるIPC損失が増加しているものの、性能に影響を与えるほどではない。これらの結果より、性能改善のためには、ノンブロッキングの同期が重要であることがわかる。

スレッド不在によるIPC損失は、各モデルにおいてそれほど変化はない。各モデルにおいてIPC損失が異なるのは、通信手法の違いが実行時に無効化される`fork`命令に影響を与え、スレッド状態が変化したからである。

4.4 同期／通信のオーバヘッドの性能

同期／通信のオーバヘッドが性能に与える影響を評価する。図10に、同期／通信のオーバヘッドが2サイクルの時を基準とし、オーバヘッドが4、8、16サイクルと増加させた場合の2プロセッサ構成のSKYについての性能低下率を示す。また、図11にオーバヘッドを2、4、8、16サイクルと増加させた場合のスレッド間依存によるIPC損失を示す。

ほとんどの場合、同期／通信のオーバヘッドの増加にしたがい、スレッド間依存によるIPC損失が増加し、性能向上率が低下していくことがわかる。オーバヘッドが2サイクルと比べて、オーバヘッドが4サイクルの場合、

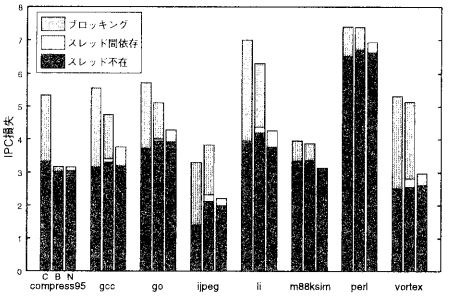

図9　同期機構のIPC損失

性能低下率は最大で1.9%であり、この程度は許容できると考えられる。オーバヘッドが8、16サイクルの場合は大きな性能低下が生じる。オーバヘッドが8サイクル以上になるとスレッド間依存によるIPC損失が増加している。オーバヘッド16サイクルでは、全体のIPCに対するスレッド間依存によるIPC損失が平均7.9%、最大で11.7%を占め、性能を低下させている原因となっている。

なおvortexでは、オーバヘッドが4サイクルの時の方が2サイクルの時より、わずかに性能が向上している。これは実行時に無効化される`fork`命令の違いによって、スレッド状態や通信するレジスタなどが変化することにより生じる。

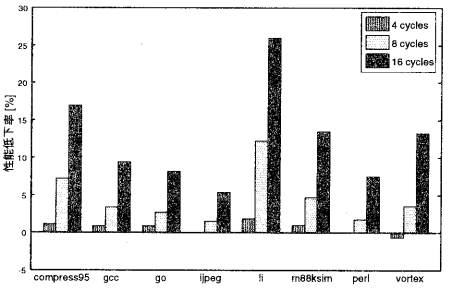

図10　同期／通信のオーバヘッドの性能

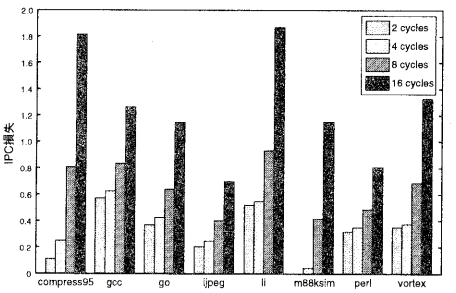

図11　同期／通信のオーバヘッド変更時のスレッド間依存によるIPC損失

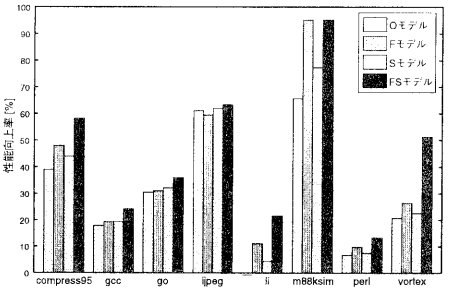

図12　性能低下を緩和するモデルの性能評価

以上より、スレッド間依存によるIPC損失が性能に大きく影響を与え、低レイテンシのレジスタ値通信が重要であることがわかる。

5. 効果的にTLPを利用するモデル

前章でSKYは大きな性能向上を達成しているものの、十分にTLPを利用していないことが分かった。本章ではより効果的にTLPを利用するための手法について、簡易的に2つのモデルを示し、評価する。

SKYにおいてTLP抽出の妨げとなっているものは制御依存による制約であり、制御依存のためにコンパイラによるTLP抽出が制限される場合である。この制約を緩和するモデルとして、投機的に`fork`命令を実行するモデル、投機的に`send`命令を実行するモデルが考えられる。

各モデルにおいて、`fork`／`send`命令はデコード直後に制御依存が解消する前に実行される。これにより制御依存によって制限されているTLPを利用し、スレッド不在／スレッド間依存によるIPC損失を減小させることができる。

5.1 性能低下を緩和するモデルの評価

本節では、性能低下を緩和するモデルの評価を行う。各モデルの効果の予備的評価を行うため、以下の仮定をした。

- 4プロセッサ構成のSKY
- 投機的に実行する`fork`命令は必ず成功する
- 投機的に実行する`send`命令は必ず成功する

以下の4つのモデルについて評価した。

Oモデル: 通常のSKYのモデル
Fモデル: 投機的に`fork`命令を実行するモデル
Sモデル: 投機的に`send`命令を実行するモデル
FSモデル: 投機的に`fork`命令と`send`命令の両方を実行するモデル

図12に各モデルの評価結果を示す。縦軸は基準プロセッサに対する性能向上率である。また、図13に各モデルのスレッド不在、スレッド間依存によるIPC損失を示す。左からO、F、S、FSモデルである。

Fモデルは、Oモデルに対して幾何平均で5.2%の向上を達成している。これは、スレッド不在によるIPC損失を緩和しているためである。しかし、スレッド間依存によるIPC損失が増加しているため性能向上率は小さ

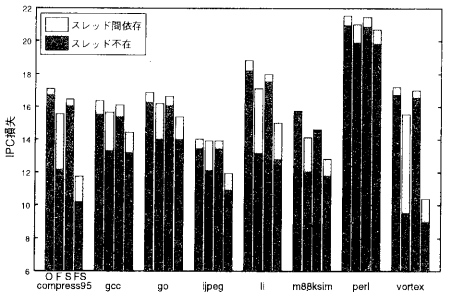

図 13　性能低下を緩和するモデルの IPC 損失

い。特に li や vortex では、スレッド不在による IPC 損失が半分近くに減少しているにも関わらず、スレッド間依存による IPC 損失が約 10 倍に増加しているため、性能向上がほとんど見られない。

S モデルは、O モデルに対して幾何平均で 2.5%しか向上していない。投機的に `send` 命令を実行することによりスレッド間依存による IPC 損失を緩和しているが、緩和することができるスレッド間依存による IPC 損失が小さいため、性能の向上にほとんど寄与しない。FS モデルは、O モデルと比較して幾何平均で 11.7%の性能向上を達成している。vortex では最大で 25.5%の性能向上を達成している。これは投機的な `fork` 命令実行により増加したスレッド間命令依存関係による IPC 損失を、投機的な `send` 命令実行によって緩和することができたからである。

以上より、投機的な `fork` モデル、投機的な `send` モデルは、それぞれスレッド不在、スレッド間依存による IPC 損失を緩和するが、どちらか一方のみでは十分な性能向上を達成できない。投機的な `fork`、投機的な `send` の両方を用いるモデルを使用することにより性能向上を達成できる。

6. ま と め

我々は非数値計算プログラムに対し、マルチスレッド実行により性能を向上させるオンチップ・マルチプロセッサ SKY を提案してきた。SKY は単一のスーパスカラ・プロセッサに対して、大きな性能向上を達成しているものの、プログラムによって性能向上率に大きな違いが見られる。本研究では IPC の損失という点から性能を制限する要因の分類を行い、IPC 損失と性能向上の関係について分析した。

2 プロセッサ構成の SKY において、スレッド不在による IPC 損失が幾何平均で理想 IPC の 20.1%を占め、性能を制限する要因となっていることがわかった。また、マルチスレッド実行が ILP 利用に及ぼす悪影響として、スレッド生成、スレッド終了、分岐予測ミス増加の 3 つの要因を示し、これらが性能向上と相関があることがわかった。

効果的に TLP を利用するモデルについて提案し、予備的評価を行った。投機的に `fork` 命令、`send` 命令を実行することにより、効率的に TLP を抽出し性能を向上させることができた。

謝　辞

本研究の一部は，文部省科学研究費補助金基盤研究 (C)「広域命令レベル並列によるマイクロプロセッサの高性能化に関する研究」(課題番号 1068034) 及び文部省科学研究費補助金基盤研究 (C)「分岐予測と投機的実行に関する研究」(課題番号 11680351) の支援により行った．

参 考 文 献

1) D.W.Wall, "Limit of Instruction-Level Parallelism," In *Proc. Fourth Int. Conf. on Architectural Support for Programming Language and Operating Systems*, pp.177-188, April 1991.
2) M.S.Lam and R.P.Wilson, "Limits of Control Flow on Parallelism," In *Proc. 19th Int. Symp. on Computer Architecture*, pp.46-57, June 1992.
3) 小林良太郎, 岩田充晃, 安藤秀樹, 島田俊夫, "非数値計算プログラムのスレッド間命令レベル並列を利用するプロセッサ・アーキテクチャSKY," 並列処理シンポジウム JSPP'98, pp.87-94, 1998 年 6 月.
4) R.Kobayashi, M.Iwata, Y.Ogawa, H.Ando, and T.Shimada, "An On-Chip Multiprocessor Architecture with a Non-Blocking Synchronization Mechanism," In *Proc. 25th Euromicro Conf.*, pp.432-440, September 1999.
5) 岩田充晃, 小林良太郎, 安藤秀樹, 島田俊夫, "制御等価を利用したスレッド分割技法," 情報処理学会研究報告 ARC-128, pp.127-132, 1998 年 3 月.
6) S.E.Breach, T.N.Vijaykumar, and G.S.Sohi, "The Anatomy of the Register File in a Multiscalar Processor," In *Proc. MICRO-27*, pp.181-190, December,1994.
7) G.S.Sohi, S.E.Breach, and T.N.Vijaykumar, "Multiscalar Processor," In *Proc. 22nd Int. Symp. on Computer Architecture*, pp.414-425, June 1995.
8) 鳥居淳, 近藤真己, 本村真人, 池野晃久, 小長谷明彦, 西直樹, "オンチップ制御並列プロセッサ MUSCAT の提案," 情報処理学会論文誌, Vol.39, No.6, pp.1622-1631, 1998 年 6 月.
9) M.D.Smith, M.A.Horowitz, and M.S.Lam, "Efficient Superscalar Performance Through Boosting," In *Proc. Fifth Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp.248-259, October 1992.
10) G.Kane, *MIPS RISC Architecture*, Prentice Hall,Englewood Cliffs, New Jersey, 1988.
11) M.D.Smith, "Tracing with Pixie," Technical Report CSL-TR-91-297, Stanford University, November 1991.