条件分岐の効率的実行を可能とする細粒度並列アーキテクチャ

3P−8

小松 秀昭[*1]　鈴木 英俊[*2]　深沢 良彰[*2]　門倉 敏夫[*2]

[*1] 日本ＩＢＭ東京基礎研究所
[*2] 早稲田大学理工学部

1　はじめに

これまでのノイマン型計算機では、処理速度の効率化のために、命令の実行時間の短縮や、パイプラインやベクトルプロセッサによる命令の連続的実行や、複数命令を同時に実行する並列処理のアーキテクチャ（粗結合、密結合、細粒度並列）などのさまざまな手法が考案されている。

並列処理は科学技術計算の分野においては、ベクトル・プロセッサやアレイ・プロセッサ等によってよい成績を修めている。しかし、オペレーティング・システムやコンパイラなどのシステム・プログラムや、一般的なアプリケーション・プログラムにおいては、あまり成果があがっているとはいえない。これらのプログラムは科学技術計算に比べて、対象とするデータ自身に明白な並列性（配列の操作など）が少ないことも原因の一つとして挙げられる。この解決策としてＶＬＩＷなどの細粒度並列アーキテクチャが提案され、ＶＬＩＷのためのトレース・スケジューリング[1]やパーコレーション・スケジューリング[2]などのコード・スケジュール法によって、データ自身の並列性だけでなく、プログラムが本来もっている並列性もより多く取り出すことが可能となっている。

これまで考案されてきたアーキテクチャは、ある条件によって処理を変えることを、条件分岐命令によって実現している。条件分岐命令によって実行中のコンテクストが切り替えられることは、一般的なパイプライン・アーキテクチャにおいて、パイプラインの切断による実行速度の低下を引き起こす。さらに、ＶＬＩＷなどの細粒度並列アーキテクチャにおいては、コンテクストの分断によって論理的な並列性が損なわれてしまうため、並列実行可能な命令数の減少をもたらしている。

2．本研究の目的

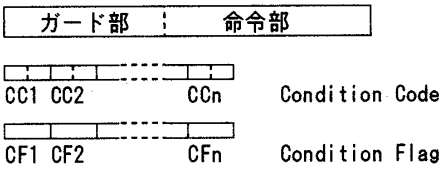
我々は、細粒度並列計算機の各命令にその命令の発火条件部分（ガード部）を付加し、実行されるかどうかを実行時に決定する機構を命令の実行部に付加するＧＩＦＴ(Guarded Instruction Architecture for Fine-grain Technique)というアーキテクチャを提案する。

条件に適合しない命令を無効化することによって、これまで条件分岐で表現されていたプログラムを条件分岐なしで記述可能となる。これによって、ＶＬＩＷなどの細粒度並列計算機で問題となっている条件分岐の存在による並列性の低下を解消し、高速な実行が可能になる。

また、このアーキテクチャをスカラ・プロセッサに適用しても、分岐命令自信の減少やそれにともなったパイプラインの乱れの減少などによる高速化が図れる。

3．ＧＩＦＴの概要

ＧＩＦＴの命令形式は、命令の実行時に満足していなければならない条件を示すガード部と、命令本体を示す命令部よりなっている。ガード部は複数の条件コードにより構成される。条件コードには、それぞれ条件レジスタに対する３通りの条件が記述される。さらに全体として、それらの条件の連言によってその命令の実行条件が記述される。



実行条件 ＝ CC1 & CC2 &・・& CCn

条件コード
・CFnレジスタが真なら、真である
・CFnレジスタが偽なら、真である
・CFnレジスタの値にかかわらず、真である

CFnレジスタが３以下の場合は、条件部は２ｎビットの長さで表現される。CFnレジスタが４以上の場合は、ガード部の長さは条件要素をエンコードすることによって２ｎビットよりも短くすることができる。

3.1　ＶＬＩＷに対するＧＩＦＴの効果

図１(A)のようなフローチャートを考える。これらすべての命令がお互いにその実行結果が他の命令の実行や条件フラグの状態に影響を与えないならば、これらの命令は、論理的に並列実行が可能である。ここでは、これらのすべての命令が並列に実行可能であると仮定する。

一般的に、書かれたままのプログラムでは、このような並列性が存在することはそれほど多くない。しかし、並列性を抽出するための前述のようなコード・スケジュール手法を用いると、かなりのプログラムから図１(A)のような並列性を取り出すことができる。

これらの命令をＶＬＩＷのアーキテクチャで実行するためには、図１(B)のようなコードを生成する必要がある。１命令が１サイクルで実行可能であるとすると、プログラムがＬ１に到達するまでに２サイクル必要である。また、Ｌ２、Ｌ３に到達する場合も３サイクル必要である。この値は、分岐先キャッシュなどの付加回路があり、かつ、確率的にそのキャッシュにヒットした場合である。そうでなければ、分岐によって命令パイプラインが壊されるため、その復旧に数サイクルが余分に必要となる。これは、実際のアーキテクチャによって異なるが、２から４サイクルぐらいが一般的な値である。２サイクルと仮定して、それぞれに加えると、Ｌ１、Ｌ２、Ｌ３に到達するまで、それぞれ４、５、７サイクルが必要となる。

一方、ＧＩＦＴでは、図１(C)のようなコードを生成する必要がある。この場合は、Ｌ１、Ｌ２、Ｌ３に到達するまでに１サイクルしか必要としない。また、分岐の遅れを考えると、Ｌ１が１サイクル、Ｌ２、Ｌ３が３サイクルが必要となる。

A guarded instruction architecture for fast execution of context switches
Hideaki KOMATSU+, Yoshiaki FUKAZAWA*, Yoshio KADOKURA*
+ IBM Tokyo Research Laboratory
* Science and Engineering, Waseda University

命令数では、ＧＩＦＴはＶＬＩＷに比べて２命令（分岐命令）を減少させることができる。また、並列性では、命令数を減らしたにもかかわらず、ＶＬＩＷに比べて４倍弱になる。

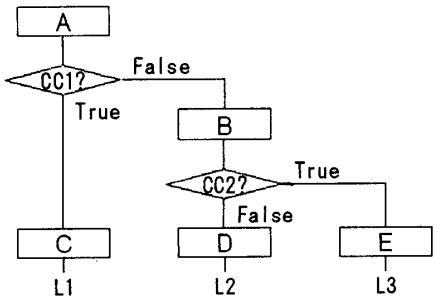


図１：並列実行可能な命令によるフローチャート

	A	Jc1 L4	Nop	Nop	Nop	Nop
	B	Jc2 L5	Nop	Nop	Nop	Nop
	D	Jp L2	Nop	Nop	Nop	Nop
L5	E	Jp L3	Nop	Nop	Nop	Nop
L4	C	Nop	Nop	Nop	Nop	Nop
L1	・・・	・・・	・・・	・・・	・・・	・・・

図２：ＶＬＩＷでの命令

※|A	F※|B	T※|C	FF|D	FT|E	FF|Jp L2	FT|Jp L3

※\|A ----- CC1とCC2の値にかかわらずA命令を実行

FT|Jp L3 -- CC1が偽でCC2が真ならL3にジャンプ

図３：ＧＩＦＴでの命令

3.2　スカラ・プロセッサに対するＧＩＦＴの効果

図４のようなＣＡＳＥ文のような一般的な多重分岐を考える。すべての分岐したコンテクスト(A,B,C)が、非常に多数の命令を実行しなければならない場合以外、スカラ・プロセッサにこのＧＩＦＴを適用することによって高速化が可能となる。

図５の（１）は３つのコンテクストがどれも多数の命令を実行する場合であり、この場合の速度は通常のスカラ・プロセッサと同じである。（２）はコンテクストＢがパイプラインの分断によるペナルティに比べて短い実行時間を持つ場合である。このコードでは、コンテクストＣに到達するのに１度もパイプラインは途切れない。また、（３）はすべてのコンテクストが短い場合である。この場合では、すべての分岐命令は命令のガード部に吸収されてしまっている。コンパイラは各コンテクストの実行時間と分岐命令の実行時間とのトレードオフによって最適なコードを生成できる。

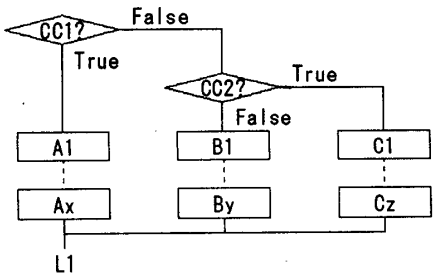


図４：多重分岐のフローチャート

T※|Jp L2 ※T|Jp L3 ※※|B1 : ※※|By ※※|Jp L1 L3:※※|C1 : ※※|Cz ※※|Jp L1 L2:※※|A1 : ※※|Ax L1:	T※|Jp L2 FF|B1 : FF|By FF|Jp L1 ※※|C1 : ※※|Cz ※※|Jp L1 L4:※※|A1 : ※※|Ax L1:	T※|A1 : T※|Ax FF|B1 : FF|By FT|C1 : FT|Cz L1:
（１）	（２）	（３）

図５：スカラ・プロセッサにＧＩＦＴを適用したコード

４．　ＧＩＦＴの評価

本アーキテクチャを評価するために、シミュレーションをおこなった結果を図６に示す 。スケジュール・アルゴリズムとして、パーコレーション・スケジューリング[2]を用いた。本アーキテクチャの目標が、シムテム・プログラムなどの並列性の少ないものであるから、対象とたプログラムは並列性少ないものを選んでおこなった。

シミュレーションは命令・データ両キャッシュとも１００％ヒットして、メモリ・アクセスには、３サイクル、その他の命令は１サイクル必要であるという仮定でおこなった。また、パイプランの分断によってジャンプ命令がは３サイクルかかる場合と１サイクルの両方についておこなった。

ＧＩＦＴ一般に、パイプラインの復旧が遅いアーキテクチャのほうがより高い速度向上率がえられる。しかし、Ｇ.Ｃ.Ｄ（ユークリッドの互除法）では、並列度の向上によりクリティカル・パスが非常に短くなったため、ジャンプのペナルティの減少が速度向上につながっている。。

ヒープ・ソートが他の２つより効果が上がっていない。これは、分岐したコンテクストが多くのメモリ・アクセスを含むなど、比較的処理時間を要するものが多かったため、条件実行命令で吸収できない部分が多かったためと考えられる。

プログラム	速度向上率	
	Jump 3(cycle)	Jump 1(cycle)
Ｇ.Ｃ.Ｄ	１.２５	１.５０
バブルソート	１.２８	１.２２
ヒープソート	１.１３	１.０８

図６：ＶＬＩＷに対するＧＩＦＴの速度向上率

まとめ

今回我々は、これまで条件分岐でしか表現できなかったプログラムを命令のガード部に吸収することによって、分岐命令の減少とパイプラインの乱れの減少、ＶＬＩＷでの命令の並列性の向上などによる高速化をめざしたアーキテクチャＧＩＦＴを提案した。現在このアーキテクチャをベースとして、ＶＬＩＷのプロセッサを設計し、パーコレーション・スケジュール[2]をもとにした、よりこのアーキテクチャに適したコード・スケジューラを考案中である。

参考文献

[1]John R.Ellis : Bulldog : A Compiler for VLIW Architectures, The MIT Press, ACM Doctoral Dissertation Award 1985

[2]Kemal Ebcioglu and Alexandru Nicolau : A global resource-constrained parallelization technique, '89 ACM SigARC International Conference on Spuercomputing