細粒度並列計算機ＮＬＩＴＨの実現とその評価

松永幸三[1] 小松秀昭[2] 深澤良彰[1] 門倉敏夫[1]
Kozo MATSUNAGA Hideaki KOMATSU Yoshiaki FUKAZAWA Toshio KADOKURA

6X-2

[1]早稲田大学 理工学部
Waseda University

[2]日本アイ・ビー・エム㈱
IBM Japan,Ltd.

1．はじめに

システム・プログラム等の様に、陽には並列性を含んでいないプログラムにおいても、３～４命令程度の並列性を含んでいる[1]。我々は、これらを並列に実行させる計算機NLITH(N Lane Instruction THread computer)を提案している。

NLITHでは、単一のプログラムを実行するために、複数のＰＥが各々独自な命令ストリームをもつ。この各命令ストリーム間では、同期を必要とする場合のみ、同期をとる。

2．ハードウェアの構成

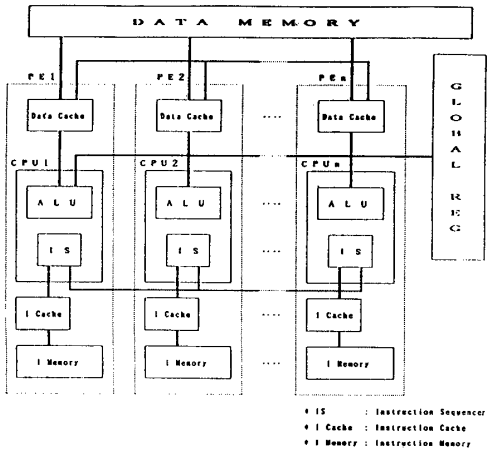


図１：NLITHのハードウェアの例

NLITHの様な密結合計算機の場合、バスのバンド幅をいかにして高めるかという問題がある。そのために、まずデータ・メモリと各PE毎のインストラクション・メモリは分離する。また、メモリ・アクセスの競合を避けるために、データ・メモリはインタリーブする。このことにより、同じバンクをアクセスしない限り、バスの競合はない。さらに、各PE毎にスヌープ・キャッシュを持たせる。バスの利用率を下げると同時に、同じアドレスに対して書込みがない限り、バスの競合は無い。この様にすることによって、インタリーブ・ファクタの少ないメモリでも、十分実用になるNLITHを作成できると考える。

現段階では、PEの個数は２～４を考えているので、レジスタにはマルチ・ポートの素子を使用するとして、グローバル・レジスタのみを考えている。

図１に現段階での、NLITHの構成を示す。

3．コード生成

NLITHにおいては、高い並列性が得られてはじめて、高速な演算が実現できる。高い並列性を抽出するために、ソース・プログラムのコンパイルは次の様に行なう。

高級言語で書かれたソース・プログラムは、コンパイラによって中間コードにコンパイルされ、最適化が施される。また、ここでは、マルチプル・ジャンプを有効に利用するために、トレース・スケジューリング[2]を利用してブランチ命令が連続するように変形する。次に、中間コードを基にし、依存性グラフを求める。オブジェクト・コードの生成は、この依存性グラフを基に行なう。

以上の手順を図２に示す。

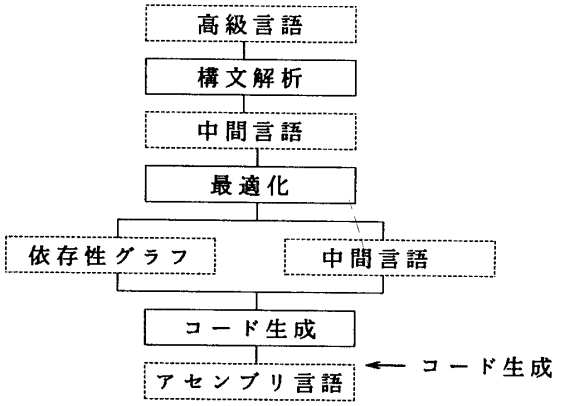
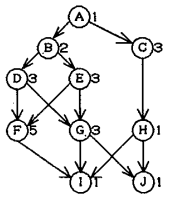


図２：コンパイルの手順

Realisation and Estimation of Fine Grain Parallel Computer NLITH
Kozo MATSUNAGA, Hideaki KOMATSU, Yoshiaki FUKAZAWA, Toshio KADOKURA
Waseda University, IBM Japan

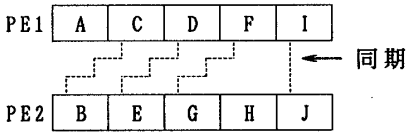
例として、図３(ａ)の様な依存グラフの各ノードを、２つのPEに割り当てる場合を考える。この問題に対する代表的な解法としてはリストスケジューリング法やＣＰ法があるが、ここではDF/IHF[3]を用いることにする。この問題の最適解は図３(ｂ)で与えられる。



(ａ)依存性グラフ

PE1	A	C	D	F			I
PE2		B		E	G	H	J

(ｂ)最適スケジュール（タイミング）



(ｃ)最適スケジュール（インストラクション）

図３：命令のスケジューリング

ここで、スケジューリングを行なう上での制約条件としては、メモリ・アクセスの競合問題が考えられる。NLITHコンパイラでは、スヌープ・キャッシュを搭載しているので、並列に実行して問題となるのは、同じアドレスに対するの書き込みだけである。したがって、自由度が高く、メモリアクセスの並列化を行ない易い。

４．システムの評価

サンプルとして、クィック・ソートその他のプログラムをコンパイルした結果、1.4～1.6程度の並列化の効率が得られた。ここで並列化の効率を、次の式で定義する。

$$\frac{\Sigma \text{ busy times}}{\text{execution time}} \times \frac{\text{scalar code size}}{\text{para code size}}$$

ただし、scalar code size はトレース・スケジューリングを行なう前のコード・サイズ、para code size はトレース・スケジューリングを行なった後のコード・サイズある。

また、フェッチしなければならないデータの量は、VLIWの場合を１とすると、PEが２個の場合は約0.8、PEが３個の場合は約0.7であった。NLITHの方が、データのフェッチ量が少ないのは、syncビットを利用することにより、タイミングを合せるための命令が必要ないためである。

４．まとめ

NLITHにおいて、すべての命令で全同期すると、VLIW計算機である。一方、PEの個数は十分にあると仮定し、各PEに１命令のみを格納し部分同期によって命令間の実行順序を制御した場合には、データフロー計算機と等価な動きを示す。したがって、NLITHは、VLIW計算機とデータフロー計算機の中間に位置すると考えられる。

部分同期を利用することによって、NLITHはデータフロー計算機に似た動きをするが、この場合、NLITHがデータフロー計算機と決定的に異なるのは、NLITHはシーケンシャルに実行が進む部分では、既存の高速化技術がそのまま使えるということである。つまり、パイプライン処理やスコアボードによる並列性も得られるということである。さらには、各PE自身がマルチALU構成になっていることですら可能である。

５．今後の課題

現段階では、グローバルなレジスタとメモリしか考えていない。各PEがローカルにメモリやレジスタを持っている場合は、どのPEに命令を割り当てるかによって、依存性グラフの形が変わってしまう。現段階で採用している、DF/IHFでは、このようなモデルの近似解を求めるのは困難である。

今後、汎用なNLITHコンパイラを作成するために、このような点も考慮に入れる必要がある。

７．参考文献

[1] H.Hagiwara, S.Tomita, S.Oyanagi, K.Shibayama : "A Dynamically Microprogramable Computer with Low-Level Parallelism" IEEE Trans. on Comp., Vol. C-29, No.7, pp.577-595(1980)

[4] John R. Ellis : "Bulldog : A Compiler for VLIW Architectures" ACM Doctoral Dissertation, The MIT Press (1986)

[6] 成田、笠原 : 「マルチプロセッサ・スケジューリング問題に関する実用的な最適解および近似アルゴリズム」電子通信学会論文誌, Vol.J67-D, No.7, pp.792-799 (Jul. 1984)