循環パイプライン・アーキテクチャ
— pipelined MIMD の試み —

4N-4

市川 周一[1]、加藤 紀行[1]、後藤 英一[2,3,1]

1.新技術開発事業団、 2.東京大学理学部、 3.理化学研究所

はじめに

pipelined (shared resource) MIMDは、MIMD型計算機の一つのカテゴリーである (Flynn[1])。しかし、この手法を用いて実現された計算機は非常に少なく、商用機として(失敗に終わったが) HEP[2]が知られるのみである。

筆者らは現在、pipelined MIMD の一種である循環パイプライン計算機 FLATS2 を開発中である[3]。その経験に基づいて、現状の技術動向における pipelined MIMD の可能性および設計上のトレード・オフについて考察する。

基本的アイデア

一般にパイプライン型アーキテクチャにおいては、パイプラインの最大性能は、パイプラインを一段進めるのに要する時間τに反比例する。従って、τを短縮するほど高い性能が得られる(“クロックを上げる”)。一方、同じ実装技術を用いる限り、τが短くなればパイプライン1段で実行できる論理は浅くなる。従って、同じ論理機能を実現するのにも多くの段数(パイプライン・ステージ)が必要になる(つまり“パイプラインが深くなる”)。

通常パイプラインが深くなると、パイプライン内で実行中の命令同志が資源の競合を起こし(ハザード)、後続命令の実行が遅らされて実効性能が低下する。このため、ただパイプラインを細分化して深くしても、性能が向上するとは限らない。従来、この性能低下をいかにして防ぐかが、パイプライン設計上のポイントであった。

pipelined MIMD では、パイプライン内で実行される命令列をそれぞれ別の命令列(仮想的プロセッサ)に割り当てる。これによって、原則的に資源の競合を回避することができる。各命令列は異なったプロセッサのものであるから、使用する資源も異なっており、従って基本的に資源の競合は起こらない。こうして実現された計算機は、ユーザから見ると1つのハードウェアをパイプライン的に共有するMIMD型計算機である。これが、pipelined MIMD という名の由来である。

このように pipelined MIMD では、パイプラインが深くなってもハザードにともなう性能低下を防ぐことができる。そこで、従来のパイプラインに比べてτを小さく設定し、パイプラインを深く設計することによって、ハードウェアとしてのスループット(各仮想プロセッサのスループットの総和)を高めることができる。

循環パイプライン・アーキテクチャ

pipelined MIMD における各プロセッサは、同じハードウェアを時分割で共有しているため、プロセッサ外の資源を容易に共有することができる。例えば、ハードウェアとしての主記憶はプロセッサ外の資源にあたるが、これはきわめて自然に共有される。主記憶とCPU間の転送路を各プロセッサが時分割で使用することによって、特別なハードウェア(スイッチ・ネットワーク等)を付加することなく、メモリ共有型マルチプロセッサを実現することができる。

もちろん、τを短縮してCPUのスループットを上げた以上、それに合わせて主記憶の転送幅も大きくする必要がある。この辺の事情はどのようなアーキテクチャについても共通であるため、従来からキャッシュやインターリーブ等、各種の実現技法が用いられてきた。pipelined MIMD に対してもこれらの手法は適用可能ではあるが、主記憶へのアクセスをCPUと同じτでパイプライン化することによって、より自然に解決することができる[4]。

パイプライン化された主記憶は、アクセス要求を受けると処理を開始するが、その処理が継続中でもτ後には次の要求を受け付ける。従って、CPUに釣り合う転送幅を実現することが可能である。また、インターリーブ・メモリにおけるバンク・コンフリクトの問題等も生じない。

パイプライン化された主記憶を効率的に実現するには、数段にパイプライン化されたメモリ・チップが不可欠であるが、このようなチップは未だ実現されていない。こうしたパイプライン化メモリ・チップは、スーパー・コンピュータの大容量ベクター・レジスタなどにも転用可能であるため、チップ化されればたいへん有益であると考えられる。

筆者たちは、このように 1)pipelined MIMD を採用し、2)CPUと主記憶を同じτでパイプライン化したアーキテクチャを、循環パイプライン・アーキテクチャ (Cyclic Pipeline Architecture; CPA) と呼んでいる。主記憶を含めた計算機システムの中を、複数の仮想プロセッサが還流しているというイメージである。循環パイプラインについては、参考文献[4]を参照されたい。

循環パイプラインの性質

ここで、改めてプロセッサの性能とは何か、考えてみる。循環パイプラインを採用した場合、パイプラインを深くしても性能の低下を避けることができるため、τを小さくしてハードウェアのスループットを上げることができる。しかし、各仮想プロセッサが循環パイプライン内を一周する時間σを考えてみると、σはτを小さくするにしたがって漸増する。これは、パイプライン化するために挿入したパイプライン・ラッチ(またはフリップ・フロップ)の遅延時間δが加わるためである。

つまり、τを小さくとると合計のスループットP_tは向上するが、各仮想プロセッサ1台あたりのスループットP_θは漸減することになる。また、パイプラインを深くすればパイプライン・ラッチの実装コストも増加するため、製作上

Cyclic Pipeline Architecture

Shuichi Ichikawa[1], Noriyuki Kato[1], and Eiichi Goto[2,3,1]

1. Reseach Development Corporation of Japan,
2. University of Tokyo, Faculty of Science,
3. The Institute of Physical and Chemical Research

不利になる。従って、τを極度に小さく、パイプラインを深くするのは好ましくない。実装技術などを勘案して適切なτを決定し、P_tとP_θを妥協させることが必要である。P_θの劣化を抑えるため、適度な先行制御を導入することが適切であると考えられる。

先程から、「循環パイプラインでは“基本的”にプロセッサ間の競合がないため、性能低下が起こらない」と述べてきた。それでは、“例外的”には性能低下が起こるのであろうか？答えはＹＥＳである。循環パイプラインにおいても、プロセッサ間で共有する資源については競合が発生し得る。各プロセッサが、そうした共有の資源をアクセスするために同期をとる際、「同期待ち」のオーバーヘッドが発生する。これが通常のパイプラインにおけるハザードに相当する。

具体的な例をあげる。各プロセッサは主記憶を共有しているが、主記憶上のデータを共有する際、データの整合性を保つためにはデータへのアクセスを相互排除的に行なう必要がある。このためには、なんらかの方法で資源をロックし、プロセッサ間の同期をとらねばならない。このような操作においては、資源のロックが他のプロセッサの性能低下を招くこともある。

技術動向との相性

キャッシュ

循環パイプラインでは、ＣＰＵのτを小さくすると同時に、主記憶へのアクセスもτ刻みで実行できるように設計する必要がある。パイプライン化メモリ・チップならば、大容量でかつτの小さいアクセス（～10ns）が安価に実現できると考えられる。しかし、チップが供給されるまでは現状の技術を代替手段として用いなければならない。

ＣＰＵの実装技術も考え合わせると、一応、現状ではτ＝10～20ns程度が現実的であると考えられる。市場には多くの高速ＳＲＡＭチップが供給されており、そのサイクル・タイムも20ns程度になりつつあるから、これらの高速ＳＲＡＭをキャッシュに用いるのが順当といえる。

マルチプロセッサ（ＭＰ）にキャッシュを用いた場合、一般にはキャッシュの同期が問題となる。しかし、循環パイプラインではキャッシュも共有されているので、MIMDとはいえ、整合性の問題は生じない。

ＬＳＩ化

循環パイプラインの利点は、τを短縮することによって生じる。従って、通常のパイプラインより小さなτで設計することになる。τが10～20nsでは、MSI/SSIの使用は論理遅延やチップ間の配線遅延から考えて不可能である。ECLのような高速デバイスを用いるか、１チップ上に集積することが必要になる。

幸い最近では大規模なセミカスタムＬＳＩが外販されるようになり、このような小さなτでも充分に実現が可能になった。また、パイプライン・ラッチのコストについても、MSI/SSIで論理を組む場合に比べて遙かに影響は少なくなる。１チップ上にある限り（たとえ多少チップのコストが上がっても）、システム全体のコストから見れば無視できる程度に小さいのである。

１チップ化した場合、ＣＰＵの入出力信号数に強い制限が課される。現状の技術ではPGAでピン数200程度が限界であるから、信号数はこれよりも更に少なくなる。しかし、この事情はどのような１チップＣＰＵでも同じである。むしろ循環パイプラインでは、入出力ピンもパイプライン的に使用するため、通常のプロセッサより事情が好転する可能性もある。

マイクロコード

マイクロコードの働きは、大きく順序制御とデコードに分けられる。デコードについては、組合せ論理をメモリのマッピングに置換するだけであるから、循環パイプラインでも全く問題ない。しかし順序制御の場合は、プロセッサの内部状態が増えることになるので、多少問題である。循環パイプラインでは仮想プロセッサの数だけ重複して内部状態を持たねばならないので、プロセッサの内部状態はなるべく単純にしたい。しかし、複雑な順序制御を行なえば内部状態が増加するのは当然である。

また、通常マイクロコード化された計算機は、マイクロコード内でしか使用できないレジスタ（スクラッチ・レジスタ）を持っている。これらのレジスタも各プロセッサの内部状態の一部であるから、プロセッサ分重複して持たねばならない。プロセッサの実現コストを下げ、またτを縮める意味からも、これらの内部状態が少ないアーキテクチャが好ましい。

こうした点を考慮して設計すれば、マイクロコード化は循環パイプラインに対しても充分に有効であると考えている。

ＲＩＳＣ

RISCにはいくつかのトピックスがある。命令を単純化してマイクロコードを廃し、制御論理を小型化／高速化して、高いクロック周波数を実現すること。また、論理の実装面積を減らした分、大容量のレジスタをオン・チップに実装し、オーバーラップ・レジスタ・ウィンドウによって高級言語のサポートも行なうこと、等々。

循環パイプラインにおいても、命令を単純化して内部状態を減らすことは有利に働く。また、制御論理の単純化はτを小さくすることに貢献する。その意味で、RISC的発想は循環パイプラインにも適している。

しかし、大容量レジスタを実装するということは、内部状態を増やすことを意味する。また、RISCのようにパイプラインを短く単純にという発想は、ＣＰＡの逆である。レジスタ指向のアーキテクチャはメモリよりレジスタ・アクセスのスループットが高いという仮定の下に成り立っているが、パイプライン化メモリ・チップを用いれば、この仮定を崩すことができる。その意味では、RISCとＣＰＡは異なった前提を持っているのである。

循環パイプラインの場合、命令を単純化する事によってプロセッサの内部状態を減らし、大容量レジスタの代わりにパイプライン・ラッチを多く実装して、より短いτを実現することが考えられる。

参考文献

[1] Flynn,M.J.:"Some Computer Organizations and Their Effectiveness," IEEE Trans on Comp, vol.C-21, pp.948-960.

[2] Jordan,H.F.:"Performance Measurements on HEP - A Pipelined MIMD Computer," Proc.of 10th Annual Int'l Symp.on Computer Architecture, pp.207-212.

[3] 後藤、市川、他：ＦＬＡＴＳ２のアーキテクチャ、第３５回全国大会、６Ｃ－４、1987.

[4] Shimizu.K,Goto.E,and Ichikawa.S:"CPC (Cyclic Pipeline Computer)-An Architecture Suited for Josephson and Pipelined-Machines," Tech.Rep 86-19, 東京大学理学部情報科学科、1987.