計算機アーキテクチャ　108－7
（１９９４．１０．２８）

分散演算器・データキャッシュを持つクラスタ構成ＶＬＩＷ計算機

安倍正人　松沢伸一　岡部　公起　根元義章
東北大学大型計算機センター
〒980-77 仙台市青葉区片平 2-1-1
東北大学大型計算機センター
Tel.022-221-3380 Fax.022-262-3422

分散データキャッシュメモリと分散演算器を数百個あるいは数千個用いたＶＬＩＷ計算機システムは、例えば各プロセッサエレメントで演算した結果を後処理をすることが必要なアプリケーションのようにプロセッサ間の通信が必要な場合、同期が必要な場合およびバスのアービトレーションが必要な場合でもペナルティが小さいという利点がある。しかし、システムが複数のチップあるいはボードに分かれる場合を想定すると１つのシーケンサで協調させて動作させる必要があることから、ＭＩＭＤ型計算機の１／５ないし１／１０程度のクロック周波数で動作させる必要があり、期待した程の有効性が無い。そこで、外部から与えた比較的遅いクロックに同期させて内部で合成した高い周波数のクロックで常時動作させ、分岐命令は例えば命令の格納アドレスの下位２ビットが１１のときのみ実行されるようコンパイラがスケジュールすることにより、チップ間およびボード間のクロックの時間差を吸収し、ＭＩＭＤ型計算機で用いられているものと同じ程度のクロック周波数で動作させることのできるＶＬＩＷ計算機を提案する。そして、いくつかのリバモアベンチマークにより提案方式の有効性を示す。

VLIW Computer with Distributed Operation Units and Data Cache

Masato Abe
Shinichi Matsuzawa　Kouki Okabe　Yoshiaki Nemoto
Computer Center, Tohoku University
2-1-1 Katahira, Aoba-ku, Sendai 980-77, Japan
Tel.022-221-3380 Fax.022-262-3422

VLIW computer, which is composed of more than hundreds of operation units, has an advantage that the penalty due to the communication between operation units is small, compared to the MIMD computer. However, since VLIW computer system should be composed of many LSI chips in different PCB boads, a slower clock frequency has to be used in VLIW computer compared in MIMD computer, and the total system performance is not good. Therefore, we propose a method to make the VLIW computer operate with a faster clock as well as a MIMD computer by a compiler technique, by which a branch instruction can only be issued, for example, when the lowest two bits of the instruction address is 11. Then, the difference in the clock, which is fed to each operation unit, can be made negligeble.

The effectiveness of the proposed method is confirmed by several benchmark programs.

1　はじめに

計算速度の高速化を図るために、アーキテクチャ的にはスーパースカラ計算機やＶＬＩＷ計算機が提案され、またディレイドブランチやソフトウェアパイプライン技術といった有用なソフトウェア技術も開発されている。しかし、これらの技術が想定している計算機は複数の演算器は用意しているものの、データキャッシュメモリが１つであるため、演算器へのデータ供給能力が十分でないという問題がある。この問題を解決するために、さまざまな並列システムが研究開発されているが、ＳＩＭＤ計算機は各演算器が全て同一の動作しか行えないので融通性がない。一方、ＭＩＭＤ型計算機においては、システムが複数のチップあるいはボードに分かれた場合を想定すると、各ＰＥは独立に動作できるため、１つのシーケンサで協調させて動作させる必要が無いことから、ＳＩＭＤ計算機の５倍ないし１０倍のクロック周波数で動作させることが出来るという利点がある。しかし、例えば各プロセッサエレメントで演算した結果を後処理をすることが必要なアプリケーションのようにプロセッサ間の通信が必要な場合には同期やバスのアービトレーションの点で、有効な結果が得られないという問題がある。そこで、本論文において我々は複数のプロセッサエレメントを用いたＭＩＭＤ計算機システムに対し、分散データキャッシュメモリと分散演算器を持たせたＶＬＩＷ計算機システム（ＤＣ／ＰＵ型ＶＬＩＷ計算機: VLIW Computer with Distrbuted Caches and Processing Units）を提案する。このとき、１つのシーケンサで複数のチップあるいはボードに分かれた演算器やデータメモリをクロック周波数を下げずに協調させて動作させるために、外部から与えた比較的遅いクロックに同期させて内部で合成した高い周波数のクロックで常時動作させ、分岐命令は例えば命令の格納アドレスの下位２ビットが１１のときのみ実行されるようコンパイラがスケジュールすることにより、チップ間およびボード間のクロックの時間差を吸収する。

本論文では、提案手法の有効性を幾つかのリバモアベンチマークテストを用いてＭＩＭＤ計算機と比較検討した結果も報告する。

2　ＤＣ／ＰＵ型ＶＬＩＷ計算機システム構成と命令長の比較

ＭＩＭＤ計算機において、配線量およびバス競合の頻度を考慮して、しばしばクラスタ構成を取るように、本論文で提案するシステムにおいても図1に示すように、クラスタ構成を取ることにする。第４節で述べるシミュレーションにおいては、ＭＩＭＤ計算機では８個のＰＥ（プロセッサエレメント）、提案方式では８個のＯＰＣ（ Operation Unit with Cache: データキャッシュおよびプロセッシングユニットを合わせたもの）を１つのクラスタとする。（これを１次クラスタと呼ぶ）そして８個の１次クラスタを集めて６４個のＰＥあるいは６４個のＯＰＣからなる２次クラスタシステムを構成する。更に、２次クラスタを８個集めることにより、３次のクラスタを構成し、最大５１２個のＰＥあるいは５１２個のＯＰＣからなるシステムを構成する。そして、以上を繰り返すことにより、８のＮ乗（Ｎはクラスタの次数）のＰＥあるいはＯＰＣからなる主システムを構成する。また、最下位のクラスタと同じレベルにＤＭ（分散メモリ）が配置される。更に、主システムと強調して動作するキャッシュおよび分散メモリの管理のためのメモリＩ／Ｏ管理装置を最下位のクラスタと同じレベルそれぞれと最上位のクラスタと同じレベルに１つ配置する。

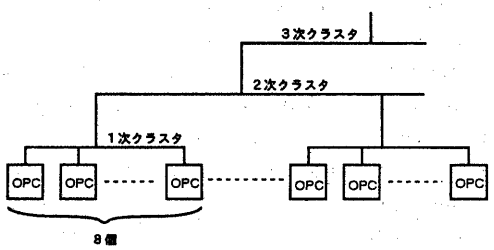

図 1: データキャッシュを持つ演算器をクラスタ構成にしたＶＬＩＷ計算機

ＯＰＣは、図2に示すようにデータキャッシュ（ＤＣ）、アドレス生成器（ＡＧ）、２つのレジスタ

ファイル（ＲＦ）、演算器（ＰＵ）で構成され、それぞれが２本の内部データバスに接続されている。ここで、内部バス＃１に出力できるのは、データキャッシュ、アドレス生成器付属のレジスタファイル、演算器の３つのうち、一度に１つのみである。一方、内部バス＃２に出力できるのは、データキャッシュ、演算器付属のレジスタファイル、演算器の３つのうち、一度に１つのみである。このようにして、各バスへの出力の制御のために必要なＶＬＩＷ命令のビット数は出力しない場合も含めて、各バスそれぞれに対し２ビットとなる。また、この外に命令キャッシュが別に設けられる。

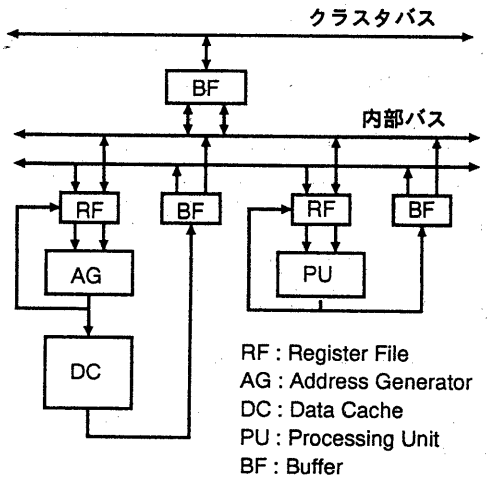

図 2: データキャッシュを持つ演算器ＯＰＣの構成

１次のクラスタは、８個のＯＰＣで構成されており、同じクラスタに属する内部バス同士はバスバッファを通して互いに接続されている。これを１次クラスタバスと呼ぶことにする。このクラスタバスに出力するＯＰＣを指定するのに３ビット、バスバッファをイネーブルするかどうかを決定するために１ビット必要とするから、８個のＯＰＣから構成される１次クラスタ１つだけからなるシステムのバス制御のために必要とされるビット数は（２＋２）＊８＋４＝３６ビットとなる。２次のクラスタは８個の１次クラスタから構成され、２本の１次クラスタバスがバスバッファを経由して互いに接続される。この時、６４個のＯＰＣから構成されるシステムのバス制御のために必要とされるビット数は３６＊８＋４＝２９２ビットとなる。最大５１２個のＯＰＣが接続される３次のクラスタにおいては、同様にして、２９２＊８＋４＝２３４０ビットが必要となる。このことから、１つのＯＰＣ当たりに必要なバス制御ビット数は約５（＝２３４０／５１２）であることが分かる。

アドレス生成器では、付属レジスタファイルの内容そのままの出力、２つのレジスタ間の加減算および１つのレジスタと値１の加減算（インクリメントとデクリメント）の５通りの演算を行うので、必要な制御ビット数は３となる。２つの付属レジスタファイルの制御には、レジスタ数をそれぞれ６４とすると、５ポートレジスタファイルなので、１つのポート当たり６ビットだから、１つのＯＰＣ当たりでは６＊５＊２＝６０ビット、更に書き込みポートはレジスタ当たり３ポートあるので、その書き込み制御に１レジスタ当たり３ビット必要だから、合計して、２つのレジスタファイルの制御に６０＋３＊２＝６６ビット必要となる。演算器の制御には、四則演算、論理演算等を考慮して１０ビットの制御ビットを割り当てる。最後に、キャッシュメモリの制御には、アドレスおよびデータの一時保持、および保持されたデータの書き込みのために合計３ビット必要とする。以上を総合すると、我々が提案するＶＬＩＷ型計算機では１つのＯＰＣ当たり約９０ビットを必要とする。ただし、多くの計算機のように命令の組み合わせを制限し、デコードによって必要な制御信号を作ることにより、必要な命令長を削減することもできる。よって、命令長の立場から見れば、ＭＩＭＤ方式の計算機と大差がないと考えられる。

なお、システム全体として、分散メモリの制御、シーケンサの制御、イミーディエイト生成器の制御のためにさらに幾ビットか必要とするが、例えば６４個のＯＰＣからなるシステムの場合、１つのＯＰＣ当たり３ビット程度となるのでさほど問題ではない。

3　分散キャッシュメモリの動作

3.1　分散メモリとキャッシュメモリ間のデータ転送

分散メモリは同期式ＤＲＡＭを用いて実現されるので、連続したアクセスは十分に高速だが、離散的なアクセスは低速である。一方、キャッシュメモリは離散的なアクセスでも十分に高速なＳＲＡＭで実現されるが容量が少ない。そこで我々は、演算を制御する主システムとは独立に動作するメモリＩ／Ｏ処理装置（ＭＩＯ）を用いて、キャッシュメモリを以下に示すように管理することにした。　ＭＩＯはＮ個のＯＰＣに分散されたキャッシュメモリを管理するが、各キャッシュメモリは論理アドレス３２ビット、１ページ当たり２５６ワード、全部で６４ページのダイレクトキャッシュ方式で管理される。

一方、分散データキャッシュは、Ｗｒｉｔｅ ｉｎｖａｒｉｄａｔｅ方式[1]によりキャッシュのコヒーレンスを保つことにする。このため、図?? に示すように分散キャッシュ内のＴＬＢには、アドレスタグＡＴ、有効フラッグＶＦ、他のＯＰＣ内の分散キャッシュメモリに同じアドレスページのコピーがあるかどうかを示すコピーフラッグＣＦおよび、そのページの内容が分散メモリの第一番目のコピーであるかどうかを示すファーストフラッグＦＦから構成されている。アドレスタグＡＴは該当ページに入っているデータの分散メモリ上での論理アドレスの上位１８ビットを保持し、有効フラッグＶＦは、１ならばキャッシュメモリ中のデータが有効で、０は有効でないことを意味する。一方、８個のＯＰＣあたり１つづつと全体に共通に１つだけ設けられているメモリ入出力装置ＭＩＯの構成は図?? となっている。

4　ＯＰＣ間の同期

分岐する場合あるいは自分に所属しないレジスタファイルを参照する場合は、他のＯＰＣと同期を取る、すなわちＯＰＣ間のクロックの時間差を補正する必要がある。

4.1　分岐における同期

ここでは、同期用の外部クロックの周期は内部クロックの周期の４倍を仮定し、命令は内部クロックに同期して実行される。図3において、シーケンサ内で外部クロックに従って内部クロックが生成され、外部クロックが立ち上がってから、２つめの内部クロックの立ち下がりで、次に実行すべき４つの命令のアドレスが生成され、全ＯＰＣに転送される。各ＯＰＣでは、図3に示すように ΔT_1 あるいは ΔT_n の時間差があるが、その時間差が外部クロックの周期の半分程度であれば問題無く動作する。

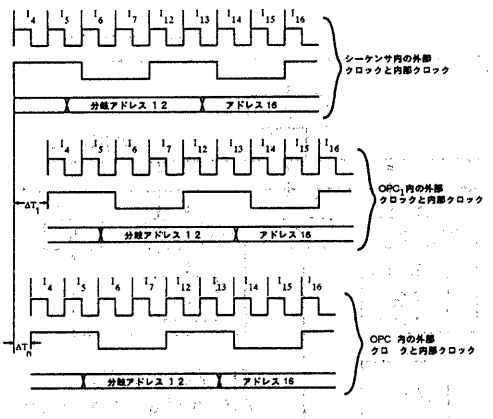

図 3:　ＯＰＣ間のクロックの時間差を補正するための分岐シーケンス

この場合、分岐命令が実行できるアドレスが限られ、ＮＯＰ命令を挿入しなければならないことも生じ、効率が低下する。しかし、全て遅いクロックで動作させるよりは高速化が期待でき、またループの場合はループアンローリングを用いることにより、効率の低下は多少抑制できる。

4.2　自分に所属しないレジスタファイルを参照する場合の同期

例えば OPC_1 のレジスタファイルから、OPC_n のレジスタファイルへデータを転送する場合を考える。この場合もＯＰＣ間のクロックの時間差を補正する必要があるので、図4に示すように、例えば OPC_1 のレジスタファイルから内部クロック

４つ分の間データを送出し、ＯＰＣ$_n$内のレジスタファイルへの書き込みはデータが揃ったころを見計らって行われる。この手法では、ＭＩＭＤ計算機におけるメッセージ交換に伴うオーバーヘッドあるいは、バス経由に伴うバスのアービトレーションと同期の問題が無く、高速にＯＰＣ間通信を行うことができる。

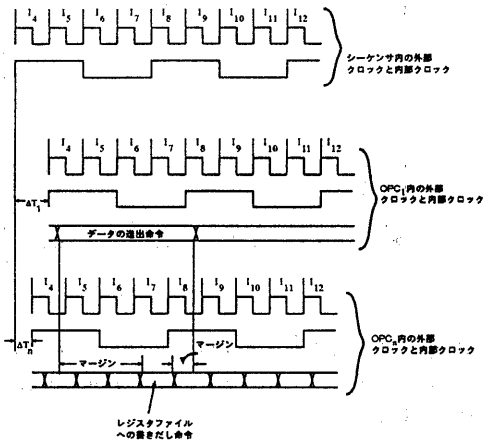

図 4: ＯＰＣ間のクロックの時間差を補正するためのＯＰＣ間通信のシーケンス

5　ベンチマークによる比較

5.1　比較対象としたＭＩＭＤ計算機の構成

ＭＩＭＤ方式では、さまざまな構成が考えられるが、ハードウエア規模を考慮して我々のと同じクラスタ構成を取ることにした。すなわち、図1においてＯＰＣの代わりにキャッシュメモリ付きのＰＥがある場合を想定した。また、キャッシュ交換の方式はどちらも同じで、ＰＥ間あるいはＯＰＣ間の通信方式だけが異なり、提案しているＤＣ／ＰＵ方式はバス経由、ＭＩＭＤ計算機はメッセージ交換とした。

5.2　比較条件

４節までの説明ではＯＰＣ内のキャッシュメモリの１ページ当たりのデータ数は２５６ワードとしたが、本シミュレーションでは６４、１２８、２５６、５１２ワード(１ワード＝４バイト)の４種類について調べた。また、エントリ数は６４とした。更に、キャッシュ入れ替えの際は、１内部クロック当たり８ワード読み込みが出来るものとし、最初のデータを読み込む際のオーバヘッド、すなわちバスの確保やキャッシュコヒーレンスの確保等に要する時間は内部クロック数で２０とし、後続する連続したデータの読み込みに要する時間は内部クロック数で１とした。

一方、ＭＩＭＤ計算機において、ＰＥ間でＮワードのデータを一度に交換するのに要する時間は、メッセージ交換で行うため、同じクラスタに属する場合は内部クロックで２４０＋１０Ｎ、異なるクラスタの場合は４９０＋１０Ｎとした。それに対し、提案するＤＣ／ＰＵ計算機では、同期やバスのアービトレーションの問題無く直接バスを経由してデータを交換できるので、内部クロック数で７とした。

6　リバモアベンチマークによるシミュレーション

ここでは、コンパイラが分散メモリにデータを最適に格納しており、各ＰＥあるいはＯＰＣ間でデータの競合が出来るだけ起きないようにしている。ただし、ループの回数は１０００００とした。ＫＥＲＮＥＬ　１、７は、後処理のためのＰＥ間あるいはＯＰＣ間通信の必要がないので、スピードアップレシオの結果はＤＣ／ＰＵ計算機とＭＩＭＤ計算機は全く同じである。図5および図6にその結果を示す。

一方、ＫＥＲＮＥＬ　３あるいは２４の場合は、各ＰＥあるいはＯＰＣで計算された部分積和あるいは部分最小値を集めて全体の積和あるいは最小値を求める必要がある。このためには、ＰＥあるいはＯＰＣ同士が通信し合う必要があり、ＭＩＭＤ計算機ではペナルティが大きい。この様子を図7と図8に示す。

7　むすび

ＭＩＭＤ計算機で問題となる通信によるオーバーヘッドを削減するために、ＶＬＩＷ計算機において、１つのシーケンサで多数の分散キャッシュ及び演算器をクラスタ構成にして同時に制御する計

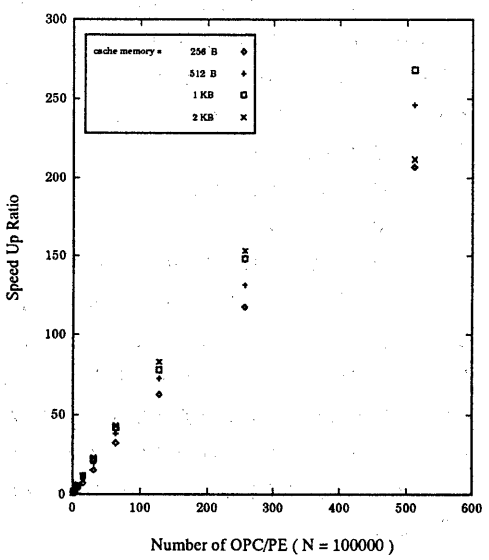

図 5: ＫＥＲＮＥＬ　１を実行した時の１個のＣＰＵに対するスピードアップレシオ (総データ数１００、０００

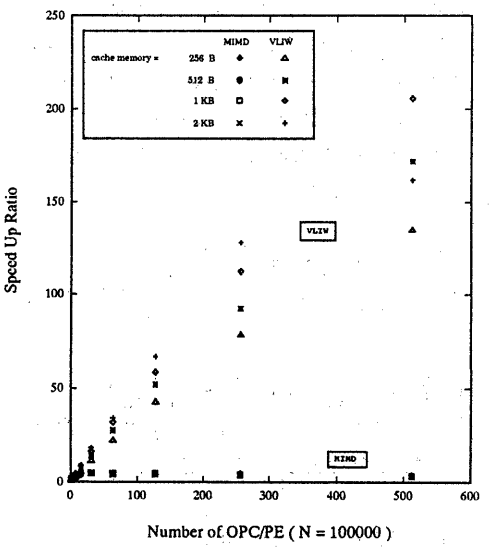

図 7: ＫＥＲＮＥＬ　３を実行した時の１個のＣＰＵに対するスピードアップレシオ (総データ数１００、０００

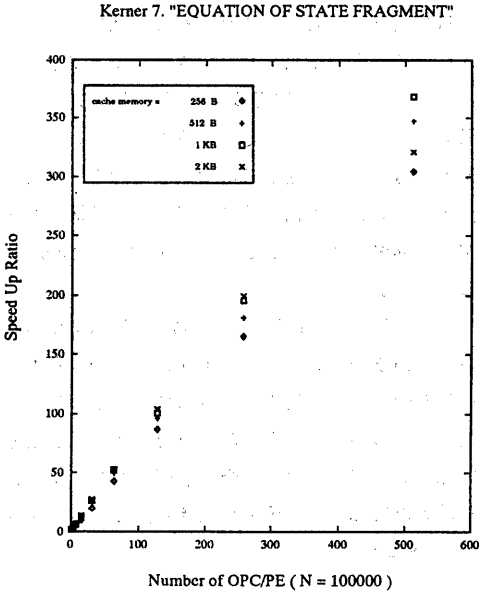

図 6: ＫＥＲＮＥＬ　７を実行した時の１個のＣＰＵに対するスピードアップレシオ (総データ数１００、０００

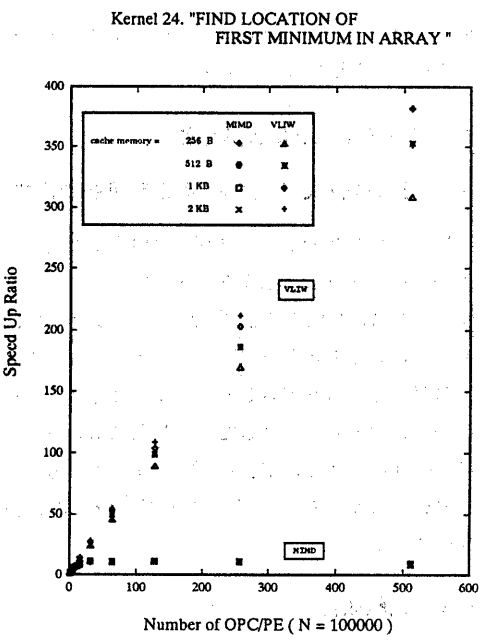

図 8: ＫＥＲＮＥＬ　２４を実行した時の１個のＣＰＵに対するスピードアップレシオ (総データ数１００、０００

算機を提案した。この際、複数チップあるいは複数ボード使用時のクロックの時間差を吸収するた

めに、分岐やＰＵ間の通信は高速な内部クロックではなく比較的低速な外部クロックに同期させることにした。その結果、各ＰＥが独立に動作できることからクロック周波数を上げることで高速化が達成できているＭＩＭＤ計算機とほとんど変わらない高速化が提案するＤＣ／ＰＵ型ＶＬＩＷ計算機で達成できるものと思われる。

リバモアベンチマークのうち並列計算機が有効な４つのプログラムについてシミュレーションを行い、提案するＤＣ／ＰＵ型ＶＬＩＷ計算機は、通信が必要なプログラムに対してＭＩＭＤ計算機より有効であることを示した。

参考文献

[1] Nitzberg B., Lo. V.,"Distributed Shared Memory: A Survey of Issues and Algorithms," IEEE Computer, August 1991, pp.52-60