計算機アーキテクチャ　101－8
（１９９３．　８．　２０）

お茶の水１号の構成と評価

中里 学　　大津 金光　　戸塚 米太郎　　松本 尚　　平木 敬

東京大学理学部情報科学科

当研究室で開発中の汎用細粒度並列計算機お茶の水１号の概要を報告する。従来のバス結合型マルチプロセッサでは、低オーバーヘッドの同期・通信機構が備わっていなかったために、命令レベルの細粒度並列処理を効率よく行なうことは不可能であった。お茶の水１号では、要素プロセッサには市販の汎用高性能マイクロプロセッサを用いているが、外部の支援機構にはFPGAを用いているため、応用に適したさまざまな機構を実現することができ、細粒度並列処理の汎用テストベッドとすることができる。今回実装した機構には、1) オーバーヘッドの極めて少ない *Elastic Barrier*、2) メモリベースの同期・通信を融合したデータ駆動同期機構、3) 大規模な配列データをキャッシュ上の連続領域に効率良くフェッチする大域構造体先行フェッチ機構がある。本稿では、お茶の水１号の構造および上記の機構について述べ、細粒度並列処理における効果を考察する。

Architecture and evaluation of OCHANOMIZ-1

NAKAZATO Gaku　OOTSU Kanemitsu　TOTSUKA Yonetaro
MATSUMOTO Takashi　HIRAKI Kei

Department of Information Science
Faculty of Science
The University of Tokyo

We report the general-purpose fine-grain multiprocessor OCHANOMIZ-1. Conventional bus connected multiprocessors have no facilities to support light weight synchronization nor communication. Therefore these machines cannot handle efficiently instruction level parallelism. OCHANOMIZ-1 has commercial high performance microprocessors as its processor elements and FPGA's to support fine grain parallel processing. These FPGA's enable us to implement various type of mechanisms fitting to applications. Our current implementation of fine grain supporting mechanisms are 1) *Elastic Barrier*, which provides processors' synchronization with little overhead. 2) *Data Driven Synchronization*, which unifies producer-consumer type data transfer and synchronization. 3) *Global Structure Pre-Fetching Mechanism*, which efficiently transfers large array data (possibly with non-unit stride) into continuous cache lines. In this paper, the overall structure of OCHANOMIZ-1 and the implementation of these mechanisms are described. We also discuss the impact of these mechanisms on fine grain parallel processing.

1 はじめに

多くの大規模実用アプリケーションは容易に並列度が抽出可能な部分と抽出困難な部分を含んでいる。粗粒度で容易に並列度抽出できる部分は並列実行するプロセッサの台数さえあれば、かなりのスピードアップが可能であるが、その結果として粗粒度による並列化が困難な部分がボトルネックとなってしまう。この並列化の困難な部分に関しても高速化を達成するためには細粒度の並列性の活用が不可避である。しかしながら、複数のプロセッサが密に協調動作を行なえばプロセッサ間のデータ転送や同期のためのオーバヘッドが深刻になってくる。これらのオーバヘッドをいかに削減することができるか、いかに効率的なプロセッサ間通信・同期機構を備えたシステムを構築できるかが細粒度処理方式成功の鍵である。さらに細粒度の並列性の抽出には最適化コンパイラの支援が不可避であるが、そのためには密結合並列処理が都合がよい。これまでも効率的な密結合並列処理の実現のために様々な計算機が作成されてきた [1, 2, 3, 4]。しかしながら、これらのシステムに実装された機構は今一つ満足できないものであった。

今回我々は従来型の高性能マイクロプロセッサを要素プロセッサとして用いた汎用細粒度並列計算機システム お茶の水1号 (OCHANOMIZ–1: Omnipotent Concurrency Handling Architecture with Novel OptiMIZer–1) を作成した。お茶の水1号は低オーバヘッドな同期機構や通信機構を装備し、効率の良い細粒度並列処理が可能な密結合並列計算機システムである。細粒度並列処理支援機構として、Elastic Barrier、大域構造体先行フェッチ機構、メモリベースのデータ駆動同期機構、ページ単位でのキャッシュプロトコルの切替え機構などが実装されている。汎用大規模並列計算機システムはお茶の水1号をクラスタとして階層化することで実現される。

また細粒度の並列性の抽出には最適化コンパイラの支援が不可避である。お茶の水1号はその名の通り専用の最適化コンパイラ [5] を仮定しており、コンパイラで静的にスケジューリングされたコードが実行されることを前提としている。

以下では、2節でお茶の水1号全体のアーキテクチャについて述べる。3、4、5節ではお茶の水1号上に実現された細粒度並列処理支援機構のうち主な3つの通信・同期支援機構についてそれぞれ説明する。

2 お茶の水1号の構成

従来型のマイクロプロセッサを結合した計算機システムによって細粒度並列処理を行う場合、データの通信・同期に対する支援がされなければオーバヘッドが大きくなり性能向上が望めない。細粒度並列処理をする小規模のプロトタイプを作成し、その上に試験的な機能を実装し有効性を検証することが重要である。汎用細粒度並列計算機お茶の水1号は従来型の高性能マイクロプロセッサを用い、低オーバヘッドでプロセッサ間の通信・同期をおこない細粒度の並列実行する。システムの全体構成図を図1に示している。要素プロセッサにはNECのVR4400MC(75MHz)[6, 7][1]を4台用いている。プロセッサは共有バスを介して互いに接続している。システム共有バスはスプリット処理を行なう同期バスであり、バスアービトレーションは分散方式で行なわれる。アドレスバスは36bit、データバスは64bitでスヌープキャッシュを支援する制御線を備えている。VR4400MCはチップ内部に2次キャッシュコントローラを内蔵しており、インバリデイト系及びアップデイト系のスヌーププロトコルをサポートしている。しかしながら、2次キャッシュの制御は全てプロセッサに握られており、スヌープのためのキャッシュの状態[2]の問い合わせ等はプロセッサを経由したものになるためにレスポンスが非常に遅い。そこでお茶の水1号ではレスポンスを高速化するために2次キャッシュのバックマップ管理を行なっている。バックマップ用として2次キャッシュと同じ高速SRAMを装備している。

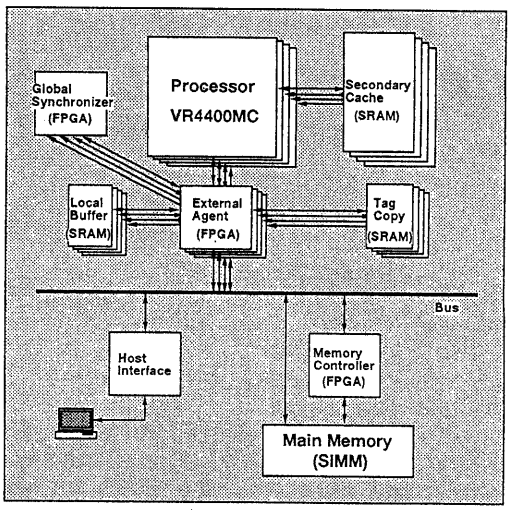

図 1: お茶の水1号の構成図

プロセッサとバスの間に外部エージェントと呼ばれる回路が位置しており、プロセッサからの要求を受けとりバスに要求を出したり、プロセッサにレスポンスを返したりする。

VR4400MCではレイテンシ隠蔽に有効なキャッシュプリフェッチが実現されていない。そこでお茶の水1号ではプリフェッチ機構を実現するための付加回路が装備されており外部エージェントにより制御されている。

主メモリとバスの間にメモリコントローラがあり、主メモリのリフレッシュをしプロセッサやホストからの要求を満たすようにメモリアクセスをする。さらにSRAMにより実現された同期ビットを管理している。プロセッサどうしが共有バスを使わずに高速に通信したり、同期を行うために大域同期機構がある。大域同期機構としてバリア型同期機構、共有レジスタファイルを実現している。ホストインタフェースを介してホストコンピュータ (PC-AT互換機) が接続されてい

[1]現在のところ供給の都合でVR4000MC(50MHz)を使っているが動いていない。

[2]invalid, clean exclusive, dirty exclusive, shared, dirty sharedがある。

る。ホストはFPGA(Field Programmable Gate Array)のコンフィギュレーション、主記憶やキャッシュに対するデータやプログラムの書き込み、システム全体のリセット、データの収集のために使用される。

外部エージェント、メモリコントローラ、大域同期機構、ホストインタフェースはXilinx社のFPGA XC4010(10,000gates相当)を用いて構成されている。このFPGAはプロセッサ内部のクロックを分周したクロック[3]で動いている。

3 大域構造体先行フェッチ機構

従来のキャッシュメモリを用いた並列計算機では参照の局所性を利用して性能を引き出すことを原則としている。それゆえに本質的に局所性のない処理を苦手としている。お茶の水1号も原則として参照の局所性を利用するが、本質的に局所性のない処理を効率良く処理するという問題に対する回答として大域構造体先行フェッチ機構(GSPF機構 : Global Structure Pre-Fetch機構)を備えている。この節ではGSPF機構について説明を行なうと共に設計時の予定性能を示す。

3.1 構成

図2は本機構の構成図である。スカラプロセッサが外部エージェントと呼ばれるシステムを介してシステム共有バスに接続されている。外部エージェントの本来の仕事はプロセッサが発行するコマンド[4]を解釈してシステム共有バスを使用してプロセッサの要求を満たすことである。外部エージェントの実現はユーザに任されているのでここに様々な機構を盛り込むことが可能となっている。本GSPF機構もこの外部エージェントを利用して実装される。

GSPF機構の動作を実装の際の制限などと併せて説明すると次のようになる。

1. プロセッサは用意してもらいたい構造体データに関する情報を外部エージェントに伝える。構造体としてはいろいろなものが考えられるが今回はコンスタントストライドに対象を絞ることにした。よって構造体に関する情報としてはベースアドレス、サイズ、ストライドの3つが分かれば十分である。[5]。プロセッサから外部エージェントへいかにしてこの情報を転送するかに関してはアドレス空間の一部を外部エージェントとの通信用レジスタに割当て情報のやり取りを行なうことで実現する。なおGSPF機構使用時のデータの基本サイズは1ダブルワード(64ビット)である。

2. 提供された情報をもとに外部エージェントはシステム共有バスを通じて、主メモリからデータを持ってきて一時的にローカルバッファメモリに格納しておく。要求されている構造体データを全て持ってくるまでプロセッサのデータ要求に答えられないという状況を避けなければならない。本実装ではプロセッサがデータを要求する順はGSPF機構がデータを獲得した通りであるという仮定をおき、現在何処までデータを獲得できたかという情報をカウンタで保持することにした。プロセッサが要求したデータが既にローカルバッファメモリ内に存在するかどうかの判断はこのカウンタとの比較で行ない、もしデータがローカルバッファメモリ内に存在すれば即座にデータをプロセッサに受け渡し、存在しなければデータがやってくるまでプロセッサを待たせる。

3. プロセッサからデータの要求が来たら即座にローカルバッファメモリからデータをプロセッサに渡す。外部エージェントのローカルメモリバッファはプロセッサのアドレス空間の一部に直接マップされておりデータの受渡しはそのアドレスへのread/writeで実現されている。

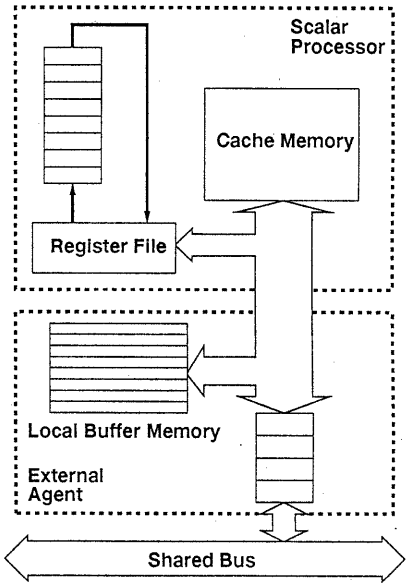

図2: GSPF機構の構成

このGSPF機構はDecoupledアーキテクチャ[8]と似ている点も存在するが、データアクセスを行なうアクセスプロセッサのための命令流がない点で異なる。さらにDecoupledアーキテクチャではプロセッサは必然的に専用プロセッサを使用することになるが、GSPF機構の場合は市販のマイクロプロセッサを使用することができるという利点を備えているまた一方でフェッチバッファ付きのプリフェッチ機構[9]とも類似しているが、フェッチバッファ付きのプリフェッチ機構は毎回プリフェッチのためのコードを必要とするのに対して、GSPF機構は一度セットアップが済めば以後プリフェッチのためのコードは必要ない点で異なる。

同じ構造体データを複数のプロセッサで利用するという場合これらを別々にデータを取りにいったのではバスの使用回数が増えてバス競合が問題となる。これはデータ転送を行なっている最中に関係するプロセッサ全てがバス上に流れているデータを取り込めばバスの使用回数を削減できる可能性がある。また同じデータを使用しなくても1回のブロック転

[3]VR4400MCでは16分周まで設定可能。

[4]データのread/write要求やスヌープキャッシュ実現のための様々な処理要求がコード化されたものである。

[5]実際には後で述べるall-read処理のためにデータを取り込ませる相手を指定するための情報も転送される。

送分のデータ中にそれぞれのプロセッサが必要とするデータが含まれる場合などは1回のブロック転送で複数のプロセッサにデータを供給できる可能性がある。

そこでGSPF機構に多少の拡張を加えて同じ構造体データを複数のプロセッサで利用する場合は通常のGSPFとは別に他のプロセッサ[6]にもデータを取り込ませるGSPFを用意した。これはMISC[10]におけるall-readプロトコルのプリフェッチ版に類似しているが、MISCにおけるall-readが直接キャッシュにデータを注入するのに対し、GSPF機構ではプロセッサからのデータの要求が来るまで外部エージェントが一時的に管理する点が異なる。

GSPF機構でall-readサポートするためにシステムバスのプロトコルに拡張がなされ、あたかも他のプロセッサから要求を出したかのように振舞うことができるようになった。ここでシステム共有バスについて簡単に触れる。システムバスには誰がバスを通じて要求を出したかを知らせる信号線があり、各プロセッサ[7]に対応した線が存在している。通常のリクエストでは要求を出す側は自分に対応した線のみをアクティブにして要求を出し、返事をする側は要求を受け取ったときにアクティブになっていた線と同じ線をアクティブにすることで返事をする。これを拡張して、all-readの要求を出す側が要求を出す際に他のプロセッサの分までアクティブにすることで一度のリクエストで複数回のリクエストを出したことにする。返事を返す側はリクエストを受け取った時と同じ信号線をアクティブにするので他のプロセッサは単に返事を受け取るだけで良い。この際、複数のプロセッサが同じリクエストを出そうとしていたわけであるが一番最初にリクエストを出せたプロセッサ以外はリクエストを出さずに返事だけを受け取るようにしなければならない。もしも受け取ったブロック中の全データを使用しない場合は、プロセッサに受け渡す時に不必要なデータをスキップすればよい。

3.2 性能見積り

本小節では本機構の予定性能を示す。同時にキャッシュメモリを使用した場合との比較も行なう。

先ず、図3にサンプルプログラムを示す。これらのプログラムに対してR3000用の最適化コンパイラ[8]を用いて最適化レベルを最大にしてコード生成を行なった結果、ループのアンロールが行なわれ1回のイテレーション中に4つの演算と4回分の演算に相当するロードストア命令が実行されるコードが出力された。これを当研究室で開発したコンパイラOP.1[5][9]のコード再配置バックエンドを用いてコードの再配置を行なったコードをもとに予定性能を算出した。サンプルプログラム1を用いてユニットストライド、ストライドが4のコンスタントストライドの性能、サンプルプログラム2を用いて積和演算でall readを使用した場合と使用しない場合の性能をそれに対応するキャッシュの場合との比較と併せて算出した。

以下に予定性能値算出の際の設計値及び若干の仮定を列挙する。命令の実行にかかるクロック数などの数値はお茶の水1号に採用されたプロセッサVR4400[7]の実際の値を使用した。

```
double a[N], b[N], c[N];
double s;
int i;

/* サンプルプログラム 1 */
for (i = 0; i < N; i++)
    a[i] = b[i] + c[i];

/* サンプルプログラム 2 */
s = 0.0;
for (i = 0; i < N; i++)
    s += a[i] * b[i];
```

図3: サンプルプログラム

	サンプル1	
	ユニットストライド	ストライド4
GSPF機構	12.5	12.5
キャッシュ	7.1	1.8
	サンプル2	
	all-read使用	all-read使用しない
GSPF機構	45.3	37.5
キャッシュ	21.4	21.4

表1: GSPF機構予定性能(MFLOPS)

- プロセッサは75MHzで動作。
- 命令ストリームによるキャッシュミスは起こらないとする。
- 1次キャッシュ、2次キャッシュのラインサイズはどちらも4ダブルワード(32バイト)であり、1次キャッシュへのアクセスには2クロック、2次キャッシュへのアクセスには6クロックを要する。
- メモリから2次キャッシュへの1ライン分のデータ転送に24クロック。
- バス上では2クロックに1データ(64ビット幅)を外部エージェントに供給でき、外部エージェントが保持する1ライン分のデータを2次キャッシュにフィルするのに6クロックを必要とする。

表1は上記設計値(及び若干の仮定)をもとにして、GSPF機構を使用した場合とキャッシュメモリを使用した場合との比較を行なったものである。キャッシュを使用した場合もGSPF機構を使用した場合も同じコードを使用しているのでどちらかと言えばキャッシュに不利に働いている可能性もありフェアな比較とは言えないかもしれないが、GSPF機構の方が単純にキャッシュのみを使用した場合よりも性能が出せる予定である。

[6]正確にはそれに附属している外部エージェントである。

[7]正確には外部エージェントであるがここではプロセッサとして話をする。

[8]現時点でR4000用の最適化コンパイラが手元になかったのでやむを得ずR3000用の最適化コンパイラを使用した。

[9]OP.1は現在、GSPF機構を活用するコードを出力できない。

4 メモリベースのデータ駆動的同期機構

細粒度並列処理ではプロセッサ間のデータ通信が頻繁に起こりうるため、データ通信とそれに伴う同期のオーバヘッドを小さく抑えることが重要である。従来、生産者消費者型の同期のために利用されてきた同期機構としてはHEPのfull/emptyビット[11]やデータ駆動計算機等で用いられてきたI-structureメモリ[12]があげられる。しかし、フォンノイマン型のプロセッサを要素プロセッサとする並列計算機ではこれまでこのような機構は用いられてこなかった。しかし、スヌープキャッシュ機構とfull/emptyビットによる同期機構を組み合わせた生産者消費者型の同期を効率良く処理する機構[10]が提案されるに至り、フォンノイマン型の共有メモリ共有バスマルチプロセッサにおいてもその性能が期待されるようになってきた。このような機構を用いると同期がデータ駆動的に行なわれるため、データ通信と同期を統合的に処理でき、データ通信とは別の同期のための特別な手段を必要としないという利点がある。

お茶の水1号にはこのようなデータ駆動的同期機構が導入されている。メインメモリにはワード毎にデータのfull/emptyを示す同期ビットが付加されており、同期はメモリコントローラおよび各プロセッサの外部エージェントによって行なわれる。また、お茶の水1号にはメインメモリ上にFIFOキューを構成する機構が搭載されている。このFIFOキューはメモリアドレスを識別子として構成されるものであり、複数のプロセッサが識別子であるアドレスにアクセスすることによりそのFIFOキューが共用されるものである。実際のFIFOの管理はメモリコントローラによってプロセッサとは独立に行なわれる。

以下ではお茶の水1号における同期ビットによるデータ駆動同期およびメモリ上にFIFOキューを構成する機構の実装方式および同期性能の見積もりを行なう。

4.1 同期ビット機構

お茶の水1号のメインメモリは各バンクにわかれている。各バンクのメモリにはワード毎に同期ビットが付加されている。現状では同期ビットはデータの存在を示す1ビットであるが、同期ビットの拡張により高機能メモリ[13]を実現することも可能になる。各メモリバンクにはメモリコントローラが付随されている。メモリコントローラは通常のアクセスの処理を行なうほか、同期機構の制御を外部エージェントと共に行なう。

同期ビットをキャッシュ上にも付加し、同期ビットの処理をキャッシュ上で行なうようにすれば、スヌープキャッシュプロトコルとの組合せで効率の良い同期が実現できる[10]が、お茶の水1号のように既存の内蔵キャッシュをもつプロセッサを利用したマルチプロセッサではキャッシュ上での同期ビットの管理には困難な点が多い。そこで、お茶の水1号ではメインメモリのみに同期ビットが付加されており、キャッシュ上で同期を行なうような機構にはなっていない。このため本機構はキャッシュされない領域で利用可能であり、メモリコントローラと外部エージェントによりメインメモリを介した形で同期、データ通信が実現されるようになっている。

リードリクエストがプロセッサから発行されるとそのプロセッサの外部エージェントはそのリクエストのアドレスを保持する。メモリコントローラは該当するワードの同期ビットを調べ、FULL状態であればデータのレスポンスを行なうが、EMPTY状態であれば何もしない。リードリクエストを発行したプロセッサはデータを受け取るまで待ち状態にはいる。そのプロセッサの外部エージェントはメモリコントローラからのデータのレスポンスか、あるいはターゲットアドレスへの別のプロセッサからのライトリクエストの検出を行なう。ライトの検出は外部エージェントがバス上のライトリクエストのアドレスと保持しているアドレスとの比較によっておこなう。外部エージェントはレスポンスデータが到着したときはデータを受け取りプロセッサに送り、ライトを検出したときはメモリへのライト中にメモリに書き込むデータをそのままバスから取り込んでプロセッサに送る。

プロセッサからライトリクエストが発行されるとメモリコントローラはデータのメモリへのライトとともに対応する同期ビットの値をFULL状態に書き換える。このとき上記のとおりに、先に同じアドレスへのリードリクエストによりブロックしているプロセッサがあれば、そのプロセッサの外部エージェントはデータをバスから取り込みプロセッサにデータを送る。

同期の一例を図4に示す。プロセッサBとプロセッサCはプロセッサAがアドレスXにデータを書く前にリード要求をし、ブロックされている。プロセッサBとプロセッサCの外部エージェントはバスを見張っており、プロセッサAがアドレスXにデータの書き込みをすると、外部エージェントはそのデータをバスから取り込んでプロセッサに送ることで、ブロックしていたプロセッサBとプロセッサCは再起動される。

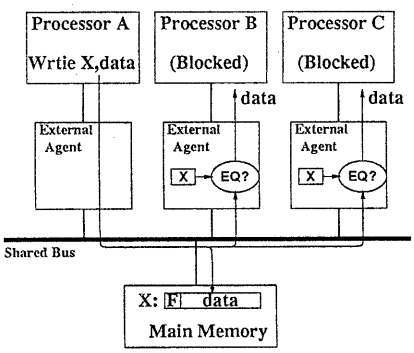

図4: 同期ビットを使用した同期の例

同期ビットはEMPTYあるいはFULLのどちらかの値をとることになるが、それぞれ0か1のどちらかに固定してはいない。アドレスの一部に同期極性ビット[10]をもたせ、極性ビットの値によってFULL/EMPTYと0/1の対応を決めている。これにより、データ通信に使い終った後のメモリの再使用が低コストに実現できる。また、アクセス手段として通常の同期とは無関係のメモリアクセスと区別する必要があるため、これもアドレスの一部のビットを使って区別している。

4.2 メモリ FIFO 機構

お茶の水1号のメモリ上のもう一つの機能としてFIFOキューを構成する機能がある。FIFOキューは現在は全部で8本用意してある。このFIFOキューはメモリアドレス（の一部）を識別子として指定される。プロセッサからのアクセスは通常のメモリアクセスと同様でよく、アクセスするアドレスは識別子であるアドレスであればよい。FIFOキューの管理はメモリコントローラによって行なわれ、FIFOはキャッシュされない領域においてのみ構成される。メモリコントローラは各FIFOキューに対するポインタを持っており、FIFOの管理をリングバッファによる方式で行なっている。

プロセッサからのFIFOキューへのリードリクエストが起こると、メモリコントローラはキューにデータがあればデータのレスポンスを行ない、データがなければ何もしない。このときリードリクエストを発行したプロセッサの外部エージェントはデータのレスポンスがきたか、他のプロセッサから同じアドレスに対するライト（すなわち同じFIFOキューへのライト）が発行されたかの検出を行なう。外部エージェントはメモリからデータが返送されたときはそのデータをプロセッサに送るが、ライトの検出をしたときは再度FIFOキューへのリード要求を発行する。プロセッサからFIFOへのデータの書き込みがあると、メモリコントローラはデータを該当するFIFOに書き込むが、もしFIFOがデータで満たされていた時はそのライトをリトライさせるようにしている。

4.3 同期性能の見積もり

ここではお茶の水1号の同期ビットを使用したデータ駆動同期の性能の見積もりをソフトウェアによる同期変数を用いた同期との簡単な比較によっておこなう。例としてプロセッサAとプロセッサBの間で1ワード分のデータ通信を行なう場合（プロセッサAがデータを書き、プロセッサBがそのデータを読む）を考える。性能の比較を表2に載せる。表2の値を求めるにあたっては、お茶の水1号の設計値として2次キャッシュアクセスに6クロック、メインメモリアクセスに24クロックかかるものとし、仮定としてバスは常に空いているものとした。

状況1はプロセッサAが、プロセッサBがリードリクエストを発行する前にデータの書き込みを行なった場合である。この場合、同期ビットを用いる機構ではデータ通信のための一回のメモリアクセスのみでよいが、同期変数を用いる場合ではデータ通信以外に同期変数のためにもメインメモリを読みにいかなければならない。状況2はプロセッサBが先にリードリクエストを発行し、プロセッサAからのライトリクエストが後から起こった場合である。ただし、この場合はプロセッサAがデータのライトを行なった時点を基準にし、それからのデータ通信完了までの時間の比較を行なったものである。同期ビットを用いる機構ではプロセッサAがメモリに書き込んでいるデータをバスから直接取り込むのに対し、同期変数を用いる場合では最初に同期変数をアクセスし、その後でメモリにアクセスしにいかなければならない。このとき同期変数はキャッシュされているものとし、スヌーププロトコルとしてアップデイト型を用いたものとしている。インバリデイト型のプロトコルを用いた場合は同期変数のアクセスのためにもメインメモリへのアクセスが必要になる。

	同期ビット	同期変数
状況1	24クロック	48クロック
状況2	4クロック	30クロック

表 2: 同期機構の性能の見積もり

なお、ここでの比較はメモリへのアクセスのみに注目したものであったが、同期成立のチェックやバスのアービトレーションによる待ちなどを考慮すればさらに両者の差は広がるものと考えられる。

5 大域同期機構

共有メモリ共有バス型のマルチプロセッサシステムでは共有変数を用いてデータ通信を行うがメモリアクセスの増大やバス獲得の待ち時間などによりオーバヘッドが生じてしまう。お茶の水1号も大域構造体先行フェッチ機構、メモリベースのデータ駆動的同期機構を搭載しているが、いずれもシステム共有バスに競合が発生した時には性能が予想以上に悪くなることが考えられる。そこでお茶の水1号では各プロセッサと直接データのやりとりができるようなハードウェアをもうけて、それを利用した機構も実現した。大域同期機構(GS: Global Synchronizer)は共有バスを介さないでプロセッサ間の高速な通信・同期を支援するための機構である。図5のように1つの外部エージェントとは8bitのデータバスでつながっている。[10]これらのバスを効率的に利用して低オーバヘッドでElastic Barrier、共有レジスタファイルを実現した。

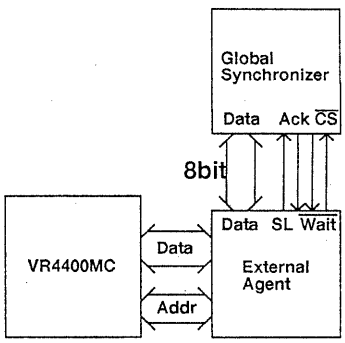

図 5: 大域同期機構(GS)

5.1 Elastic Barrier

バリア型同期機構は同じプロセスに属するすべてのシュレッドが同時に待ち合わせる比較的軽い同期機構でかつハードウェアとしては比較的簡単に構成できる。Elastic

[10]データパスは設計時の制約によりこれ以上増やすことはできなかった。

Barrier[14] は一般化されたバリア型同期機構で同期の発生順序が静的に決定できればオーバヘッドなしに同期をとることができるものである。お茶の水1号の目的の1つとしてこの Elastic Barrier を実装してアプリケーションに対する有効性を検証することにある。プロトタイプはプロセッサ4台と小規模構成なので同期コントローラをプロセッサ毎にもつのではなく、GS を用いて集中的な管理をすることにした。GS とプロセッサは直接データを受け渡しできないので、より高速な同期を実現するために Elastic Barrier に必要な3つのカウンタ (承認、予告、成立カウンタ) は各プロセッサの外部エージェントがもっている。外部エージェント内で処理できるものは処理してプロセッサにレスポンスを返すことにした。したがって GS の動作としては各プロセッサが出した同期信号線と同期マスクレジスタとのマスクをとり同期条件の成立を検出して Acknowledgment を返すことになる。マスクレジスタは2個あり、2組のグループが同時にバリア同期を張れるようにした。外部エージェントの働きとしては、同期情報の種類による3つのカウンタの増減、GS の SL に対する同期信号線の activate/negate の要求、プロセッサに対する継続実行の許可になる。

プロセッサの出す命令と Elastic Barrier の関係は、RREQ はプロセッサの出した特定のアドレスに対する LOAD 命令、APRV, PREQ は特定のアドレスへの STORE 命令を同期情報とみなすことにより Elastic Barrier を実現している。

FPGA によって機能を実現しているために外部エージェントと GS の中身を通常のバリア、Fuzzy Barrier[15]、FIFO キュー方式の Elastic Barrier[16] に書き換えてそれぞれの有効性を調べることができる。

5.2 マルチポート共有レジスタファイル

VR4400MC はインバリデイト系とアップデイト系のスヌープキャッシュプロトコルをサポートしているが、外部エージェントと共有バスに対する負担は大きくプロセッサをさらに効率良く動かすためには簡単にデータの通信が行える機構があると便利である。バリア型同期機構の他にプロセッサが遅延なしにアクセスできるマルチポート共有レジスタファイルを GS に持たせることにした。

共有レジスタファイルの機能の特徴としては以下の2点ある。

- 4台のプロセッサによる読み出しを常に可能にする。
- 書き込みに対しては排他制御する。

共有レジスタは 32bit で全部で 16 本ある[11]。試作機では GS と外部エージェント間のデータバスが 8bit しかないので1バイトより大きなデータに対しては複数回に分けて転送する必要がある。

実際の (外部エージェントから見た) 通信プロトコルはデータ、コントロールバスとも本数が少ないので以下のようになっている。

read $\overline{CS}$ をアサートし、データバスで read、データサイズ、レジスタ指定をする。$\overline{Wait}$ がアサートされてなければデータを読み出す。$\overline{CS}$ を戻す。

write $\overline{CS}$ をアサートし、データバスで write、データサイズ、レジスタ指定をする。$\overline{Wait}$ がアサートされてなければ書き込むデータをデータバスに送る。$\overline{CS}$ を戻す。

単純な read/write の操作だけではなく Test-and-Set などの共有変数に対する不可分命令も共有レジスタファイルを利用してオーバヘッドなしで実現できる。ある共有変数を Test できたプロセッサはクリティカル・セクションに入り、つまりデータバスにデータを受けとることができるのでそのまま処理を続けることができる。GS がハード的に共有変数を 0 から 1 に Set し、0 を読み出すのに失敗したプロセッサはコントロール信号線に 1(ロック失敗) を受けとることになる。

5.3 性能見積り

GS によって Elastic Barrier とマルチポート共有レジスタファイルを実現した時の予測性能が表3である。パラメータは3節の値に従っているとし、FPGA はプロセッサ内部のクロックを4分周したクロックで動いているとする。Elastic Barrier の APRV と PREQ については write の2クロック[12]で済むが、RREQ についてはプロセッサから外部エージェントに出てそのレスポンスがプロセッサに戻るので 12 クロックかかる。ソフトウェアによる実現は共有カウンタ、共有フラグ、各シュレッド毎のフラグで実現する場合、各プロセッサが共有カウンタを増減し最後にバリア領域に到達したプロセッサが共有フラグを反転した後 (96 クロック)、各シュレッドがフラグとの一致を確認するので 120 クロックはかかってしまう。

共有レジスタに対する read/write はデータのサイズが1バイトであるとした。共有レジスタに対する read は競合が起こらない仮定では、プロセッサから外部エージェントに出た後 GS からデータを読み出しその結果が外部エージェントを経てプロセッサに返るので 16 クロックかかる。この値は同一レジスタに対する読み出しが競合しても同じである。ソフトウェアでは同一アドレスに対する read が4台で同時に起こると最悪の場合で 24 クロックの4倍の時間がかかる。write は競合が起こらない仮定ではプロセッサの外に命令が出ればよいので2クロックで終る。しかし同一レジスタに対する書き込みが4台で起こった場合は最悪で 26 クロックかかる。共有バスを利用しての write はライトバッファーを利用すれば速い (12 クロック) が、最悪の場合は read の場合と同様 96 クロックかかる。

共有レジスタファイルを使用した Test&Set は Load 命令なので 16 クロックで済む。ソフトウェアでは共有バスを2回アクセスしなければならないのでこの間占有できるとすれば 48 クロック[13]はかかる。

6 まとめ

汎用細粒度並列計算機お茶の水1号の構成について説明した。お茶の水1号は従来型のプロセッサを使用した小規模のマルチプロセッサシステムであり、低オーバヘッドで細粒度

[11] その内2つは Elastic Barrier のために使用される。

[12] VR4400MC はライトバッファを持っている。

[13] これはバストランザクションにかかる時間なので実際はもっとかかる。

操作	ソフトウェア	GS
Elastic Barrier	120(96)	2-12
read	24-96	16
write	12-96	2-26
Test&Set	48	16

表 3: GS の予定性能

の通信・同期を支援する機構として大域構造体先行フェッチ機構、メモリベースのデータ駆動的同期機構、および大域同期機構を搭載している。

現在のところお茶の水1号はデバッグ中であり、近々始動を目指している。また、お茶の水1号のための上記同期機構の使用を前提とした最適化コンパイラが完成される予定である。今後の課題として、並列実行可能なさまざまなアプリケーションをお茶の水1号上で実行し、性能を測定することにより、お茶の水1号のアーキテクチャの有効性を評価していきたい。

謝辞

お茶の水1号の作成にあたりチップを供給していただいた日本電気株式会社と FPGA に関する環境を支援していただいた日本ザイリンクス社に感謝いたします。

参考文献

[1] 橋本 親 笠原博徳, "OSCAR(Optimally Scheduled Advanced Multiprocessor) のアーキテクチャ,," 電子情報学会論文誌, vol. J71-D, no. 8, pp. 1440–1445, 1988.

[2] Lenoski, D. et al., "Design of Stanford DASH Multiprocessor," *Technical Report CSL-TR-89-403, Stanford Univ.*, dec 1989.

[3] 清水ほか, "高性能マルチプロセッサ・ワークステーション TOP-1," 並列処理シンポジウム JSPP'89 論文集, pp. 155–162, feb 1989.

[4] Hill, M. D. et al., "SPUR: A VLSI Multiprocessor Workstation," *IEEE Computer*, vol. 19, no. 11, pp. 8–22, nov 1989.

[5] 稲垣 達氏, 松本 尚, 平木 敬, "細粒度並列計算機用最適化コンパイラ:OP.1," 情報処理学会プログラミング－言語・基礎・実 践－研究会報告 SWoPP'93, Aug. 1993.

[6] NEC, VR4000MC μPD30402 64 ビット・マイクロプロセッサ ユーザーズ・マニュアル ハードウェア編, Feb. 1992.

[7] NEC, VR4000 64 ビット・マイクロプロセッサ ユーザーズ・マニュアルアーキテクチャ編, Feb. 1992.

[8] Smith, J. E., "Decoupled Access/Execute Computer Architectures," in *International Symposium on Computer Architecture*, pp. 113–119, Apr. 1982.

[9] Klaiber, A. C. and H. M. Levy, "An Architecture for Software-Controlled Data Prefetching," in *International Symposium on Computer Architecture*, pp. 43–53, 1991.

[10] 松本 尚ほか, "スヌープキャッシュを用いて通信と同期を統合する機構," 電子情報通信学会コンピュータシステム研究会報告, no. CPSY90-42, pp. 25–30, July 1990.

[11] Jordan, H. F., "Performance Measurement on HEP—A Pipelined MIMD Computer," in *Proc. 10th Int. Symp. on Computer Architecture*, pp. 207–212, June 1983.

[12] Arvind and R. A. Iannucci, "Critique of Multiprocessing von Newmann Style," in *Proc. 10th Int. Symp. on Computer Architecture*, pp. 426–436, June 1983.

[13] 松本 尚, 平木 敬, "超並列計算機上の共有メモリアーキテクチャ," 電子情報通信学会コンピュータシステム研究会報告, pp. 47–55, Aug. 1992.

[14] 松本 尚, "細粒度並列実行支援機構," 情報処理学会計算機アーキテクチャ研究会報告, no. 77-12, pp. 91–98, jul 1989.

[15] Gupta, R., "The Fuzzy Barrier: A Mechanism for High Speed Synchronization of Processors," in *Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 54–63, 1989.

[16] 松本 尚, "Elastic Barrier: 一般化されたバリア型同期機構," 情報学会論文誌, vol. 32, no. 7, pp. 886–896, July 1991.