1　はじめに

大規模並列計算機型スーパコンピュータは、巨大な演算速度を要求する科学技術計算に不可欠な計算資源として近年ますます普及している。このようなスーパコンピュータは、プロセッサ数の増加によりそのピーク性能を容易に向上させることが可能で、年々その記録が更新されている。しかし、これらのスーパコンピュータは、実際にアプリケーションプログラムを実行した場合の実効性能が、ピーク性能に対しきわめて低いことが知られている。これらのマシンの性能を十分引き出すためには、アプリケーションに合わせた並列化をプログラマが行なう必要がある。

ユーザが容易にシステムの性能を最大限に引き出すためには、逐次的に記述されたプログラムから並列性を抽出し、各プロセッサにスケジュールする並列化コンパイラ、スケジューラが重要であり、数多くの研究が行なわれている。現在の自動並列化コンパイラはループ並列化 (中粒度並列処理) を中心に研究が行なわれ、強力なデータ依存解析や、ループリストラクチャリング手法を用いて、様々な形状のループが並列化可能となっている。しかし、複雑なデータ依存や、ループ外への条件分岐が原因で依然並列化できないループが存在する。さらなる実効性能向上のために、ループ並列化手法以外の並列処理手法の開発が重要である。

そこで、ループ以外の部分の従来使用されていない並列性として、基本ブロック、ループ、サブルーチン間の粗粒度並列処理 [4][5] 、および、基本ブロック内部のステートメントレベルの並列処理として、近細粒度並列処理と呼ばれる並列処理が提案されている。さらに、コンパイラによるクロックレベルの厳密なコードスケジューリングにより、近細粒度並列処理において同期操作をすべて除去する無同期近細粒度並列処理 [1] も提案されている。

早稲田大学笠原研究室の OSCAR プロジェクトでは、ループ並列化に加え、粗粒度並列処理手法 (マクロデータフロー処理)、および近細粒度並列処理手法を組み合わせて処理する、マルチグレイン並列処理手法を提案し、コンパイラに実装してきた [3][2]。

このマルチグレイン並列処理は、様々な並列計算機アーキテクチャ上に実現可能だが、プロセッサ、メモリアーキテクチャ、結合方式をマルチグレイン並列処理向きに構成することにより、実効効率を最大限に引き出すことが可能になる。すなわち、従来は並列計算機を効率良く稼働させるために、並列化コンパイラ、スケジューラの研究が行なわれる傾向が強かったが、逆にマルチグレイン自動並列化コンパイラに適した並列計算機アーキテクチャを検討することが重要である。今までにも早稲田大学の OSCAR[7]、Illinois 大学の Cedar[8] などがこの考え方で設計されてきたが、これらのマシンが設計された時点からマルチグレイン並列処理の研究ははるかに進んでおり、最新の成果を生かすためには、新しいアーキテクチャが必要になる。

本報告では、マルチグレイン自動並列化コンパイラ用並列計算機アーキテクチャASCA(Advanced Scheduling oriented Computer Architecture) を提案し、その構想と全体構成について述べる。

2　マルチグレイン並列処理手法

マルチグレイン並列処理手法 [2][3] とは、基本ブロック (BB) やループ、サブルーチン間の並列処理手法である粗粒度並列処理手法 (マクロデータフロー処理手法) [4][5]、ループレベルの並列処理である中粒度並列処理手法、BB 内部の並列性を利用する近細粒度並列処理手法 [1] とを階層的に組み合わせて効率良く並列処理をする手法である。

2.1　マクロデータフロー処理 (粗粒度並列処理)

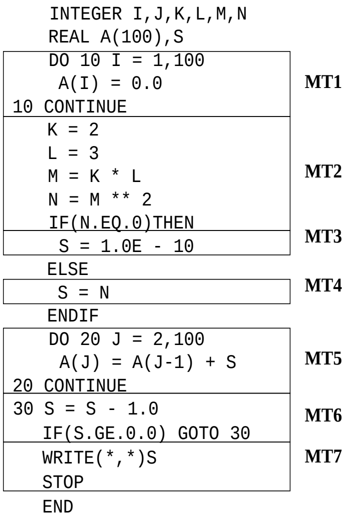
マクロデータフロー処理では、プログラムはマクロタスク (MT) に分割される。マクロタスクはブロック途中への飛び込みが無いステートメントの集合からなるブロックである。その形状から疑似代入文ブロック (BPA)、繰り返しブロック (RB)、サブルーチンブロック (SB) に分けられる (図 1)。BPA は本質的には基本ブロックであるが、並列性抽出のために単一の基本ブロックを複数に分割し複数の BPA を生成したり、逆に単一の基本ブロックの処理時間が 1 回のダイナミックスケジューリングに要する時間に比べ短い場合には、複数の基本ブロックを融合し 1 つの BPA を生成する。RB は最外側ナチュラルループである。サブルーチンブロックは、基本的にインライン展開を行なうが、コード長を考慮し効果的にインライン展開できない場合は SB として定義する。

プログラムをマクロタスクに分割した後、マクロタスク間のコントロールフロー、データ依存を解析し、それらを表すマクロフローグラフ (MFG) を生成する [2]。さらに MFG からマクロタスク間の並列性を最大限に引き出すために、各マクロタスクの最早実行可能条件を解析する。この各マクロタスクの最早実行可能条件は、マクロタスクグラフ (MTG) で表すことができる [2]。

そして、マクロデータフロー処理では全プロセッサを分割して得られるプロセッサクラスタ (PC) に MT を割当てるが、条件分岐のような実行時まで確定できない問題に対処するため、マクロタスクを実行時に PC に割り当てるダイナミックスケジューリングを行なう。

2.2　ループ並列化 (中粒度並列処理)

PC に割り当てられたマクロタスクが Do-all 可能な RB である場合、この RB は PC 内のプロセッシングエレメン



```
      INTEGER I,J,K,L,M,N
      REAL A(100),S
      DO 10 I = 1,100
       A(I) = 0.0
 10 CONTINUE
      K = 2
      L = 3
      M = K * L
      N = M ** 2
      IF(N.EQ.0)THEN
       S = 1.0E - 10
      ELSE
       S = N
      ENDIF
      DO 20 J = 2,100
       A(J) = A(J-1) + S
 20 CONTINUE
 30 S = S - 1.0
     IF(S.GE.0.0) GOTO 30
     WRITE(*,*)S
     STOP
     END
```

図 1: MT 分割の例

ト (PE) によって中粒度すなわちイタレーションレベルで並列化される (図 2)。

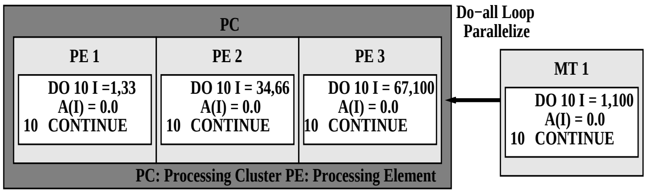


図 2: ループ並列化

2.3 近細粒度並列処理

マクロタスクが BPA やループ並列化できないループであった場合、マクロタスク内部はステートメントレベルで並列化され、PC 内部の PE によって並列処理される (図 3)。

このような近細粒度タスク集合の PC 内 PE への割り当てには、データ転送時間を考慮したヒューリスティックアルゴリズムである CP/DT/MISF 法あるいは DT/CP[13] 法を用いる。これはコンパイル時にスケジューリングを行なうスタティックスケジューリングである。近細粒度並列処理においてはこのスタティックスケジューリングをマシンクロックレベルで厳密に行なう事により、全ての同期を除去した無同期近細粒度並列処理 [1] の適用が可能である。

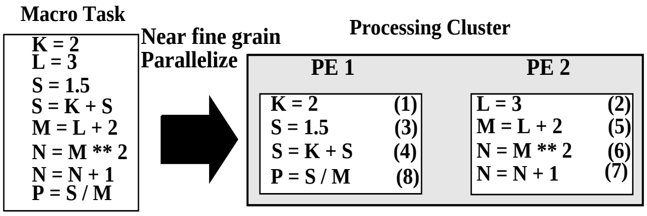


図 3: 近細粒度並列化

3 マルチグレイン並列処理向きのアーキテクチャ

ここでは、過去の試みをサーベイした後、前節で述べたマルチグレイン並列処理手法に適したアーキテクチャについて述べる。

3.1 従来の試み

3.1.1 早稲田大学 OSCAR

OSCAR [7] は、マルチグレイン並列処理を低オーバーヘッドで実現できるように 1987 年に早稲田大学で実装された。OSCAR は各々が分散共有メモリと、ローカルメモリを持つ 16 台のプロセッサエレメントと、コントロール & I/O プロセッサとが、3 つの集中型共有メモリモジュールへ、3 本のバスによって結合された分散 / 集中共有メモリ型マルチプロセッサシステムである。

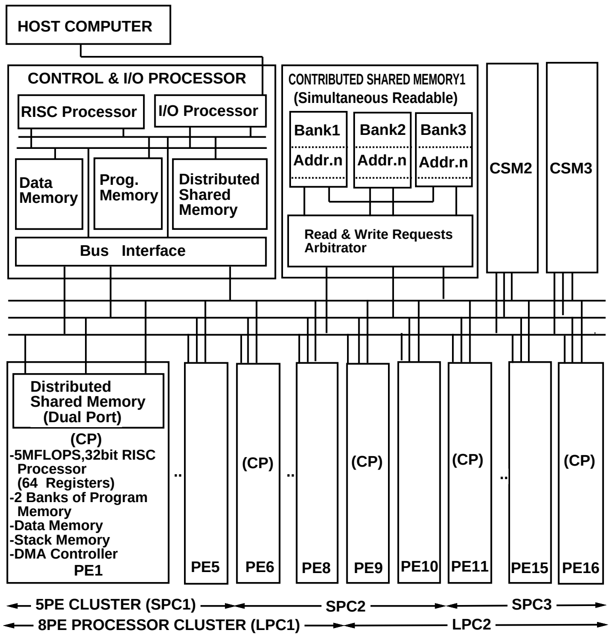


図 4: OSCAR の全体構成

PE は、最大 3MFLOPS の処理能力をもつ 1 ワード (32 ビット) 固定命令長のカスタムメイドの RISC プロセッサを持っている。

OSCAR では、スタティックスケジューリングを用いた近細粒度並列処理の効率良いインプリメントを可能とするために、以下のようなサポート機能を用意している。

- 静的な実行クロックのカスタム RISC プロセッサ
- 全 PE と同一クロックで動作するバス

これらにより、コンパイラの高精度スタティックコードスケジューラが、各 PE のバスアクセスタイミングをクロックレベルで制御可能となることで、先に述べた、基本ブロッ

ク中の全ての同期コードを削除しても正しい計算を行なえる無同期近細粒度並列処理が実現できるようになっている。

OSCARは、マルチグレイン並列処理用並列計算機アーキテクチャのプロトタイプとして、十分な機能と性能を持っており、様々なアプリケーションが実装された[11][12]が、実装時期および最初のプロトタイプであったことによる以下の問題点を持っている。

- 結合網がバスであるため、転送容量が不足する。また、このままでは、大規模なシステムの構成が困難である。
- プロセッサはカスタムRISCであるが、ほとんどパイプライン化されていない。
- 各PEがキャッシュを持っていない。

3.1.2 Illinois大学Cedar

Cedarは Illinois大学が1980年代の後半に長期間にわたって製作した並列計算機であり[8]、マクロデータフローによる並列処理を指向した点で我々のアプローチに似ている。Cedarは、Alliant社の共有メモリ型マルチプロセッサFX/8をクラスタとし、この各プロセッサを8×8のスイッチを用いた2ステージのオメガ網で共有メモリに接続した構成を持ち、32プロセッサで稼働している。FX/8は8プロセッサをバスとスイッチにより共有メモリに接続した共有メモリ型のマルチプロセッサであり、この共有メモリとクラスタ構成を用いてループレベルの並列処理を行ない、全体として粗粒度並列処理を行なう構成になっている。

Cedarの問題点は以下の通りである。

- 商用機であるFX/8をそのまま利用したため、演算の実行時間の予測、結合網の通過時間の予測、結合網自体スケジューリングが不可能で、近細粒度並列処理を行なうことが困難である。
- クラスタ構成が固定であり、柔軟性を欠く。
- グローバルネットワークのオメガ網は、ECLデバイスを用いたおおがかりなものにもかかわらず、レイテンシィが大きく、衝突時の混雑等の問題により転送時間の予測も困難である。

このため、Cedarは現在の所、32PEでデータパラレル的な並列処理を行なった評価は発表されているが、各クラスタでループレベルの並列処理を行なうマクロデータフロー的な使われ方は報告されていない。

3.1.3 慶應大学SNAIL

SNAILは慶應大学が1989年より開発したスイッチ結合型マルチプロセッサで、現在16PEで稼働中である[9]。このシステムの主な目的はSSS(Simple Serial Synchronized)型MINの性能評価であったが、SSS型MINの動作が予測可能である点を利用し、コンパイラによるスケジュールによりMINの衝突回避やメッセージコンバイニング、プリフェッチ、さらには無同期実行を行なうことを目指していた。このために、SNAILは主記憶DRAMのリフレッシュをフレームに同期させて見えなくする等クロックレベルスケジューリングを行なうための配慮がなされていた。

しかし、SNAILは、単純なスイッチ結合である点で、マルチグレイン並列処理に適してはいなかった。さらに、規模が小さい点、ネットワーク自体のレイテンシィがプロセッサの動作速度に比べ高速であったため、スケジュールによる衝突回避、メッセージコンバイニングは数%程度の効果しか得られなかった。また、プリフェッチはまったく性能を改善しなかった。

3.2 マルチグレイン並列処理に適した構成

我々は、従来の試みの中で、OSCARアーキテクチャを基本とし、この問題点を解決するアプローチを取った。以下、メモリ構成、結合網、プロセッサについてマルチグレイン並列処理に適した構成についてまとめる。

メモリ構成: マルチグレイン並列処理では、スタティックスケジューリングと、ダイナミックスケジューリングの2つのスケジューリング手法が使われる。特に、マクロデータフロー処理手法では、実行時不確定性に対応するために、ダイナミックスケジューリングが用いられる。この方式では、実行時に各プロセッサクラスタにタスクが割り当てられるので、データの転送先をコンパイル時に予測することが困難である。これに対応するためにはCedar同様、タスク間データ転送を行なうための大域共有メモリが必要である。

一方で効率の良い近細粒度並列処理の実行には、コンパイル時にスケジューリングを済ますスタティックスケジューリングが用いられる。この場合、スケジューリングにより実行前にデータの転送先は決定してしまうので、これに従って送り先のPEのメモリに直接データを書き込めることが望ましい。ただし、転送の方向は一方向で十分で、他のPEのメモリを読み出す必要性は少ない。このため、OSCARが持っていた単方向性の転送をサポートする分散共有メモリが有効になる。

また、コンパイラによるデータローカライゼーション技術も進んでおり、これを利用するためには、通信の必要ないデータを置くためのローカルメモリが必要である。

以上により、マルチグレイン並列処理を効率良く行なうためには共有、分散共有、ローカルメモリの3種類のメモリ構成を持つことがのぞましい。

また、キャッシュはメモリ-プロセッサ間の速度差を埋めるのに必須であるが、先に述べた無同期近細粒度並列処理

においてはコンピュータの全ての動作の予測ができなくてはならない。そこで、ソフトウェアで制御可能なキャッシュメモリが必要となる。

ネットワーク: まず、各粒度によるプロセッサ間通信の違いについて述べる。

- 粒度並列処理
  先行タスクより、後続タスクへデータ（配列、配列の一部、スカラ変数）を転送する。基本的に大量のデータ転送が行なわれる。タスクの配置が動的で、データ転送先の予測が困難なので、共有メモリ経由のデータ転送になる。
- 中粒度並列処理
  Do-all の場合、誘導変数、ループの終了判定等を転送するのみで、基本的に ループ中の PE 間のデータ転送はない。Do-across の場合ループ中のデータ転送が起こるが、一度のデータ転送量は少ない。転送先は事前に決定される。
- 近細粒度並列処理
  ステートメント間の単純変数の受渡しが行なわれるので、一度の転送量は少ない。転送先はコンパイル時に決定される。

以上をまとめると、マクロデータフロー処理では以下の 2 種類のネットワークが要求されることがわかる。

- 共有メモリへの大容量転送ネットワーク
- 低レイテンシの PE 間ネットワーク

また、コンパイラによる、プリフェッチ、ポストストアが指示され、処理とデータ転送のオーバーラップも行なうことがあるので、プロセッサとは独立に転送を制御できるデータ転送ユニットが必要である。

プロセッサ: プロセッサ構成は、無同期近細粒度並列処理を行なう場合に問題となる。先に述べたように、無同期近細粒度並列処理では、コンパイラがコンパイル時にクロックレベルで厳密にコードスケジューリングを行なうことで可能となる。このため、プロセッサは、キャッシュによる挙動も含め、全ての挙動がコンパイル時に予測できることが必要となる。

同期機構: 無同期近細粒度並列処理においては、クロックを揃えるため、単純で高速なバリア同期が必要となる。これは、同じマクロタスクを近細粒度並列処理する PE が参加できるクラスタ内バリア同期である。

また、マクロデータフローを階層的に行なう場合、クラスタ間グループバリアが必要である。クラスタリングを柔軟に行なう場合、これに対応する柔軟なグループバリアが要求される。これはソフトウェアで行なうことも可能だが、同期のオーバーヘッドを小さくするためには、クラスタ間の柔軟なグループバリアを備えている方が望ましい。

4 ASCA の構成

前節の検討に基づき、マルチグレイン並列処理を効果的に実現するためのマルチプロセッサシステム ASCA(Advanced Sheduling oriented Computer Architecture) を開発している。

ASCA の概念図を図 5に示す。粗粒度、近細粒度並列処理を共に効率良く行なうため、各プロセッサノード (PE) は、大転送容量の粗粒度用通信ネットワークで集中共有メモリ (CSM:Centralized Shared Memory) と結合すると共に、低レイテンシィの近細粒度通信ネットワークによって結合された分散共有メモリを持つ。PE は、実行時間が完全に予測できるパイプライン型の構成を持つ CPU およびローカルメモリと、ソフトウェア制御のキャッシュを持つ。さらに、CPU と独立に交信を制御するコントローラを内蔵している。

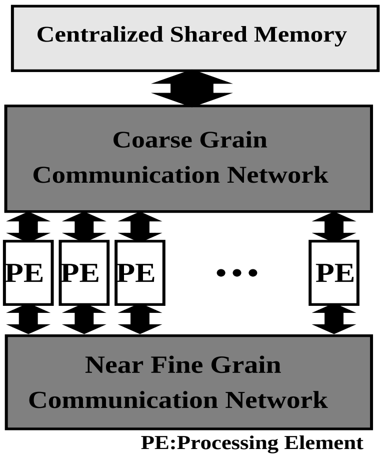


図 5: ASCA の構成

図 5はあくまで概念図にすぎない。レイテンシィの小さい近細粒度ネットワークを全 PE に張りめぐらすことは、実装上不可能であるし、粗粒度、近細粒度ネットワークを別々に持たせることは、効率が悪い。そこで、ASCA では階層型多重バスモデルと R-Clos 網を導入することにより、この問題点を解決している。

4.1 ASCA の結合網 :R-Clos 網

共有バスは最も簡単な結合方式であり、どのプロセッサがバスの所有権を得るかのみを考慮すれば結合網の混雑を防止することができるため、スケジューラからの制御が最も容易である。先に述べた OSCAR は 3 本の共有バスを持

ち、これらのバス上での転送順序のスケジューリングをクロックレベルで行ない、データ転送を含む基本ブロック内の同期コードを全て除去しても正しい計算結果の得られる無同期近細粒度並列処理を行なうことができることが確認されている [1]。すなわち、近細粒度並列処理用の結合方式としては、最も有望で実績があると考えられる。しかし、実際の共有バスは、多数の PE を結合する方法としては実装上の問題が大き過ぎる。そこで、ASCA では、コンパイラ / スケジューラから見た結合モデルと実際の結合網を分離する。コンパイラ / スケジューラ向きのモデルとしては、スケジュールし易い階層型多重バスを採用し、実際の結合網にはこれをエミュレート可能な多段結合網 R-Clos 網を用いることにした。

4.1.1 階層型多重バスモデル

階層型多重バスモデルは、階層化されたクラスタ内でバスを複数持たせ、クラスタ内の通信はその複数のバスを用い、クラスタ外へは上位バスへのリンクであるジョイントバスを用いるモデルである。各クラスタはクラスタを構成するユニット数と同数のバスをもち、各 PE に対して専用バスが一本ずつ割り当てられる。すなわち、これらのバスはアクセスする先が同じ (出線競合) でなければ衝突しない。

クラスタ間の交信には、上位階層のバスを用いることになるが、これもクラスタ内のバス同様、各クラスタに対して一本のバスが割り当てられているため、出線競合がなければ、クラスタ毎にひとつの PE が上位階層のバスを用いて転送することが可能である。

このモデルは、コンパイラ / スケジューラにとっては、クラスタ単位の出線競合が起きないことを保証すれば良いため、通信コストも考慮したスケジューリングや無同期実行のスケジューリングに対して有利である。

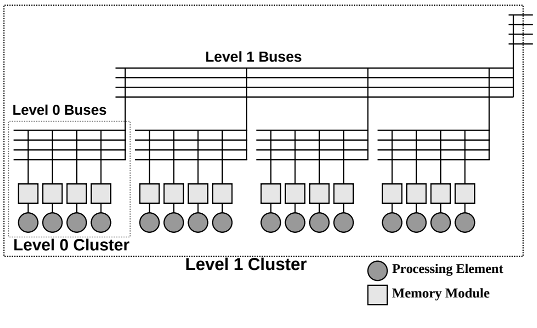


図 6: 階層型多重バス

4.1.2 R-Clos 網の構成

階層型多重バスは、コンパイラによるスケジューリングの対象としては優れた特長を持つが、そのまま実装することは困難である。そこで、階層型多重バスモデルをエミュレートできる結合網 R-Clos 網を提案した [10]。R-Clos 網は Clos 網を基本として階層構造がとれるように拡張したものである。 Clos 網は、3 ステージの多段結合網であるが、これを階層構造に拡張するために中間ステージに Exchanger を設け、これをクラスタ外通信のパスとして用いる。

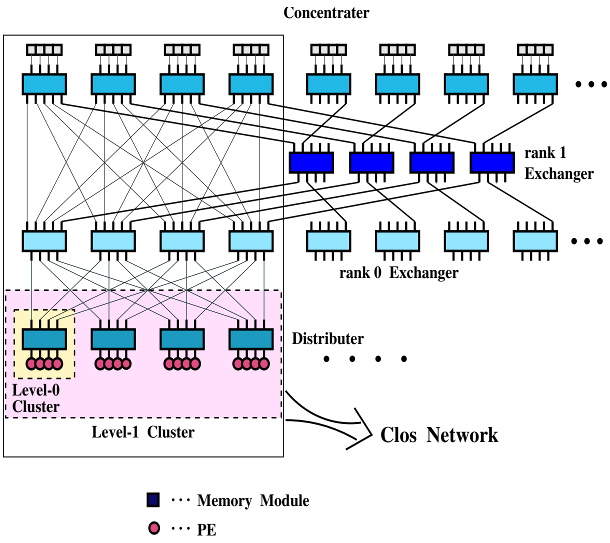


図 7: R-Clos 網

R-Clos 網はそれ自体はブロッキング網であるが、簡単な条件を考慮するだけで、階層型多重バスモデルを衝突なしにエミュレートすることができる [10]。しかも、近細粒度通信ネットワークは相手 PE の分散共有メモリに書き込みができればよい単方向通信が基本であるため、スイッチを用いた高速転送網の性能を最大限に生かすことができる。特にクラスタ内 PE 間の転送能力は強力である。

さらに、ASCA では、R-Clos 網の中間ステージを拡張することにより、前述の 2 つの性質の異なるネットワーク (粗粒度通信ネットワーク、近細粒度通信ネットワーク) を、同一ネットワークで実現する。

このためには、以下の 2 種類のパケットを用いる必要がある。

粗粒度データパケット 可変長パケットであり、近細粒度データパケットに転送を妨げられる可能性がある

近細粒度データパケット 固定長パケットであり、優先度が高い

粗粒度データパケットは、近細粒度データパケットにより転送を妨げられる可能性があるが、これは近細粒度並列処理において無同期を実現する場合、データ転送に衝突が起こってはならないため近細粒度データパケットは高い優先度が与えられていることによる。これにより、低レイテンシで、スケジューリングが適切に行なわれれば衝突を起こさない近細粒度通信が実現される。一方で、転送が中断されても再開可能な粗粒度データパケットにより、大容量転送可能な粗粒度通信ネットワークが実現される。なお、後

に述べるように R-Clos 網で構成した階層のどの位置に集中共有メモリを配置するかは、今後シミュレーションにより決定する予定である。

4.2　プロセッシングエレメント

プロセッシングエレメント (PE) は、図 8のように、CPU およびネットワークとのインタフェース、分散共有メモリ、ローカルメモリ、DTC(Data Transfer Controller) より構成される。

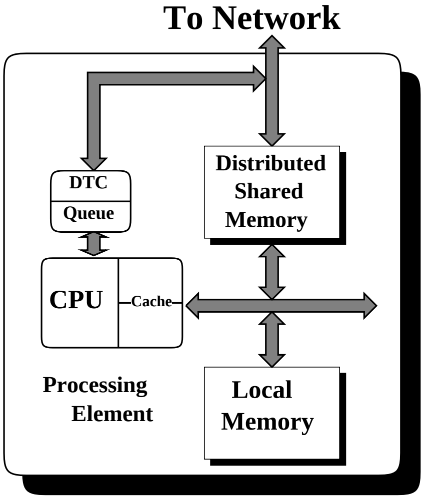


図 8: Processing Element の構成

CPU:　無同期近細粒度並列処理をサポートするためには、全ての命令の実行時間が予測可能である必要がある。実行時の処理タイミングがコンパイル時の処理計画からずれてしまうと、期待した動作を行なうことは不可能である。そこで、コンパイル時の実行計画と実行時の処理とのずれを排除し、コンパイル時の最適化の効果を最大限に引き出すために ASCA 専用のプロセッサを設計する。このプロセッサは、オペランドの値によらず命令実行時間を一定にする必要がある。このため、コンパイラが生成したコードを順番に実行し、命令発行の動的スケジューリング、Out-of-order execution などの高速化技法を行なわない構成となる。

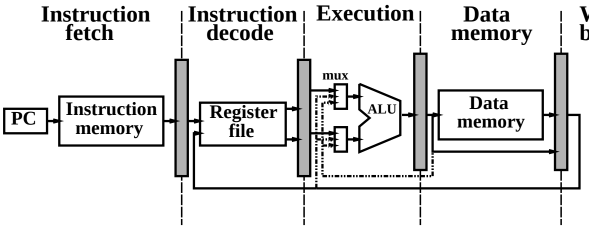


図 9: 5 段パイプライン処理

そこで、命令実行時間を乱さない高速化技法の提案が CPU 設計のポイントとなる。現在、DLX[14] をもとにした簡単な 5 段パイプライン RISC プロセッサ (図 9) を基に以下の点に関して検討している。

- 割り込みの簡略化や複雑な高速化技法をつかわないことによるクロック周期自体の短縮。
- パイプラインのステージ構成のさらなる最適化
- 浮動小数点演算装置の命令実行時間の均一化と最適化

キャッシュ構成:　キャッシュは、パイプライン化されたプロセッサに対して高速にデータを供給するために必須の技法であるが、ミスに伴い予測できない実行時間の変化を招く。このため、ASCA では、コンパイラのプリフェッチの利用に基づくソフトウェア制御可能なキャッシュをプロセッサのオンチップキャッシュおよびオンボードキャッシュの双方に導入することを検討している。

しかし、このソフトウェア制御用キャッシュは常に有効に利用できるとは限らないため、近細粒度並列処理を無同期で行なわない場合は、内蔵キャッシュは、ハードウェア制御の通常のキャッシュと利用できるようにする。さらに、分散共有メモリに対する書き込みデータなどは、一旦キャッシュにロードしてからレジスタに入れるのは、かえって性能を低下させる可能性がある。そこで、直接 PE に分散共有メモリからレジスタへのバイパスを持たせる構成を検討している。

DTC:　ASCA では、コンパイラによるプリフェッチ、ポストストアを効率良く実現するために、各 PE に CPU とは独立の転送機能を持つハードウェア DTC(Data Transfer Controller) をもつ。

CPU から発行されたプリフェッチや、ポストストア命令は、DTC との間の FIFO に蓄えられる。DTC は FIFO 中の命令にしたがい、独自に集中共有メモリからのデータの読み出し、書き込み、および他 PE の分散共有メモリへの書き込みを行なう。これにより、データ転送と処理のオーバーラップが可能になる。

4.3　同期機構

ASCA では、前述のマルチグレイン並列処理に必要なハードウェアによる同期機構を備える。

- バリア同期
- クラスタ内グループバリア
  クラスタ内で、無同期近細粒度並列処理を行なう場合にクロックレベルの同期を行なう場合や、中粒度並列処理などで利用される。

- クラスタ間グループバリア
  マクロデータフロー処理を階層的に行なう場合に使われる。これは、階層型マクロデータフロー処理におけるクラスタ間バリア同期における柔軟性程度で十分なので、階層型バスモデルにおいて、各階層バスに2本づつのバリア同期線を持たせることで対応する。

4.4 集中共有メモリ

マルチグレイン並列処理においてダイナミックスケジューリングデータを配置するなど不可欠な集中共有メモリであるが、プロセッサに階層構造を持たせた場合に、どのように配置するかが問題となる。

ASCAの場合、階層型バスモデルを用いているので、次のような配置が考えられる。

- 最上位バス(Lv.2)に配置する、全プロセッサから等距離の集中共有メモリ
- 下位のレベル(Lv.1)にそれぞれ配置する共有メモリ
- 最下位クラスタにそれぞれ共有メモリを配置
- これらの組合せ

この問題は現在検討中であるが、マクロデータフロー処理の特性、およびアクセスの集中などの問題を考えると、最上位階層と、最下位にそれぞれ配置するのが良いと考える。

今後、バンク数も含め、さまざまなモデルでシミュレーションを行ない共有メモリの配置を決定していく予定である。

5 まとめ

マルチグレイン並列処理に適したマルチプロセッサシステムASCAを提案した。現在の所、集中共有メモリの位置、ソフトウェアキャッシュの構成、R-Clos網のパケット制御ハードウェア、CPUの構成等、検討中の事項や、解決しなければならない問題等が多い。現在、OSCARのネットワークをR-Closに変更したシミュレータを作成中である。これにより階層型多重バスをエミュレートする結合網としてのR-Closの評価、およびASCA全体の予備評価をおこなう。一方で、CPU、DTC、R-Closのスイッチのハードウェア設計は既に開始されており、これらの設計上のデータは、シミュレータのパラメータとして反映していく予定である。

参考文献

[1] 尾形 航, 吉田 明正, 合田 憲人, 岡本 雅巳, 笠原 博徳, "スタティックスケジューリングを用いたマルチプロセッサシステム上の無同期細粒度並列処理", 並列処理シンポジウム JSPP'93 1993

[2] 岡本 雅巳, 合田 憲人, 吉田 明正, 笠原 博徳, 成田 誠之助, "実用レベルのマルチグレイン FORTRAN コンパイラの開発", 情報処理学会研究報告, 96ARC, Oct. 1996

[3] Kasahara H, Honda H, and Narita S,"A Multi-Grain Parallelizing Compilation Scheme for OSCAR",Proc. 4th Workshop on Languages and Compilers for Parallel Computing 1991

[4] 合田 憲人, 岩崎 清, 岡本 雅巳, 笠原 博徳, 成田 誠之助, "共有メモリ型マルチプロセッサシステム上でのFortran粗粒度タスク並列処理の性能評価", 情報処理学会論文誌, Vol.37, No.3, pp.418-429, Mar. 1996

[5] 本多 弘樹, 合田 憲人, 岡本 雅巳, 笠原 博徳, "Fortranプログラム粗粒度タスクのOSCARにおける並列実行方式", 電子情報通信学会論文誌, D-1 Vol.75, No.8 pp.526-535, Aug.1992

[6] 吉田 明正, 前田 誠司, 尾形 航, 笠原 博徳, "Fortranマルチグレイン並列処理におけるデータローカライゼーション手法", 情報処理学会論文誌, Vol. 36, No. 7 Jul. 1995

[7] 笠原 博徳, 成田 誠之助, 橋本 親,"OSCAR(Optimally SCheduled Advanced multiprocessoR)のアーキテクチャ", 電子情報通信学会論文誌 D, Vol. J71-D No. 8 Aug. 1988

[8] J.Konicek, et. al., "The Organization of the Cedar System", Proc. of International Conference on Parallel Processing, vol.1 pp. I49-I56. 1991

[9] 笹原 正司, 寺田 純, 大和 純一, 塙 敏博, 天野 英晴, "SSS型MINに基づくマルチプロセッサSNAIL", 情報処理学会論文誌, Vol. 36, No. 7, pp. 1640–1651, Jul. 1995

[10] 森村 知弘,"多重バスをエミュレートする多段結合網R-Clos網", 電子情報通信学会, CPSY 97, Aug. 1997

[11] 前川 仁孝, 高井 峰生, 伊藤 泰樹, 西川 健, 笠原 博徳, "電子回路シミュレーションの階層的並列処理手法", 電子情報通信学会技術研究報告,CPSY95-22, pp.87-94, 1995

[12] 笠原 博徳, W. Premchaiswadi, 田村 光雄, 前川 仁孝, 成田 誠之助, "OSCAR上でのスパース行列直接解法の並列処理", 情報処理学会論文誌, Vol.33, No.4, Apr. 1992

[13] 笠原 博徳,"並列処理技術", コロナ社,1991

[14] JOHN L. HENNESSY, and DAVID A. PATTERSON,"COMPUTER ARCHITECTURE A QUANTITATIVE APPROACH SECOND EDITION",Morgan Kaufmann Publishers(1996)

[15] "IDT79R4600 & IDT79R4700 ORION Processor Hardware User's Manual", INTEGRATED DEVICE TECHNOLOGY 1995