

解説



計算物理学と超並列計算機—CP-PACS 計画—

3. 超並列計算機 CP-PACS のソフトウェア†

中 田 育 男^{††} 山 下 義 行^{††} 小 柳 義 夫^{†††}

1. はじめに

一般にソフトウェアシステムの課題は、ハードウェアの持つ性能を最大限に引き出せるようにすることと、使いやすい環境を提供することであるが、CP-PACS のような分散メモリ型超並列計算機では、並列性に絡んだ課題が重要である。以下では、基本 OS とプログラミング環境について、CP-PACS での解決方法を述べ、最後に、ノードプロセッサとして採用されている擬似ベクトルプロセッサの能力を最大限に引き出す最適化コンパイラの概要を述べる。

2. 基本 OS

使いやすさと高速性を実現するために、以下の4点を OS の設計の基本方針とした。最初の2つは使いやすさのためであり、後の2つは性能のためである。

(1) 1 システムイメージで使えること：並列計算機をあたかも1つの計算機として扱うことができること。プログラムがどのプロセッサで実行されるかに依存することなく必要な OS インタフェース（ここでは UNIX インタフェース）を利用できること。これにより、並列プログラムの開発が容易になる。

(2) 標準的な OS であること：このことと(1)より、従来のプログラムが並列計算機上でほとんどそのまま利用可能となり、他機種のプログラムの移植性も高くなる。

(3) スケーラビリティが高いこと：プロセッサ数を増加させるのが容易であり、増加したらそれに応じて性能が向上すること。

(4) 軽い OS であること：ユーザの使用可能なメモリ容量を大きくするために OS 占有メモリ容量を小さく抑えることにより、OS の存在や介入による性能低下を最小限にすること。

2.1 基本 OS の概要

上記の要件を満たすために、マイクロカーネル&サーバ構成の OS とした。マイクロカーネルはカーネギー・メロン大学で開発された Mach 3.0 をベースとしている。構成を図-1 に示す。

カーネル空間上には、OS 機能のうち、特にハードウェアに依存する部分の大きいものを実装する。そのため、メモリ制御機能、タスク制御機能、プロセス間通信制御機能、入出力ドライバといった機能を持つ。ユーザに提供する UNIX の機能や、並列プログラムの実行制御機能などは、ユーザ空間と同じレベルに実装され、UNIX サーバとして機能する。

従来の UNIX システムをそのまま分散メモリ型のアーキテクチャに対応させるとすれば、socket などの重い通信機能を用いてプロセッサ間の通信を行う必要があり、しかも、各プロセッサ上には、UNIX システム全体を必要の有無にかかわらず実装する必要があり、メモリ効率もよくない。これに対し、マイクロカーネル&サーバ構成の場合、計算専用のプロセッサ上には OS の最小限の機能であるマイクロカーネルだけを実装し、必要に応じてサーバが実装されているプロセッサと通信することにより UNIX のサービスを受ければよい。これにより、計算専用のプロセッサにおいては、OS 占有メモリ容量を削除することが可能となる。また、サーバ、およびマイクロカーネルを付加的に実装することで、プロセッサ

† Software of the Massively Parallel Computer System CP-PACS by Ikuo NAKATA, Yoshiyuki YAMASHITA (Institute of Information Sciences and Electronics, University of Tsukuba) and Yoshio OYANAGI (Department of Information Science, University of Tokyo).

†† 筑波大学電子・情報工学系

††† 東京大学理学部情報科学科

数を容易に大きくすることができ、スケーラビリティを向上させることができる。図-2に両者の比較を示す。

図-3に、CP-PACSの実際のハードウェア上にマイクロカーネルおよびサーバをどのように配置しているかを示す。各計算プロセッサ (PU)

にはマイクロカーネルのみを、サーバノード (SIOU/IOU) には、各々UNIXサーバ (プロセスサーバ、ネットワークサーバ、ファイルサーバ) を実装し、各種のサービスを提供する。

2.2 並列実行機能と分割運転機能

複数のユーザが同時に利用可能であり、それぞれの使い方に応じて最適な複数プロセッサの割当てが可能で、お互いに他のユーザの影響を受けないようにすることを目標とする。そのために以下の機能をもうけた。

(1) ノード群分割機能：大規模並列計算や小規模並列によるデバッグなどの使用目的別に全プロセッサをノード群に分割する機能を持つ。分割したノード群をノードパーティションと呼ぶ。ユ

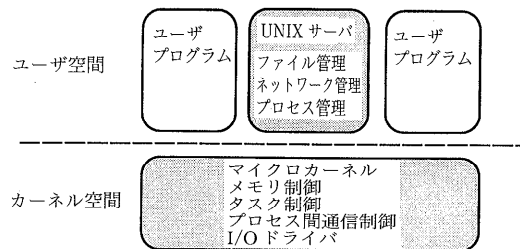


図-1 マイクロカーネル&サーバ構成

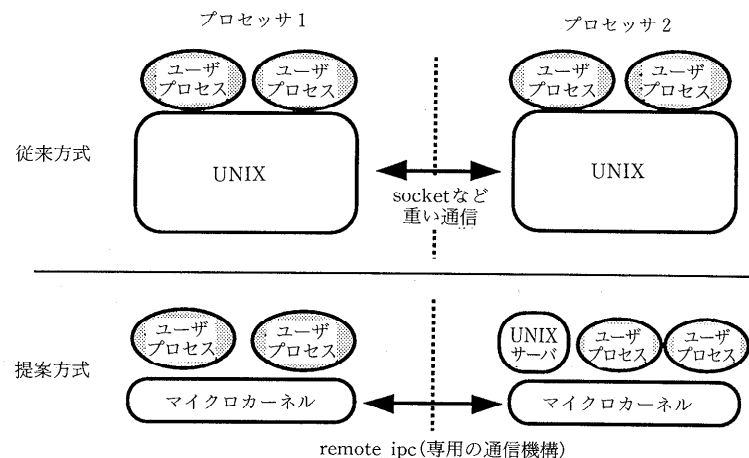


図-2 分散OSとしてのマイクロカーネル

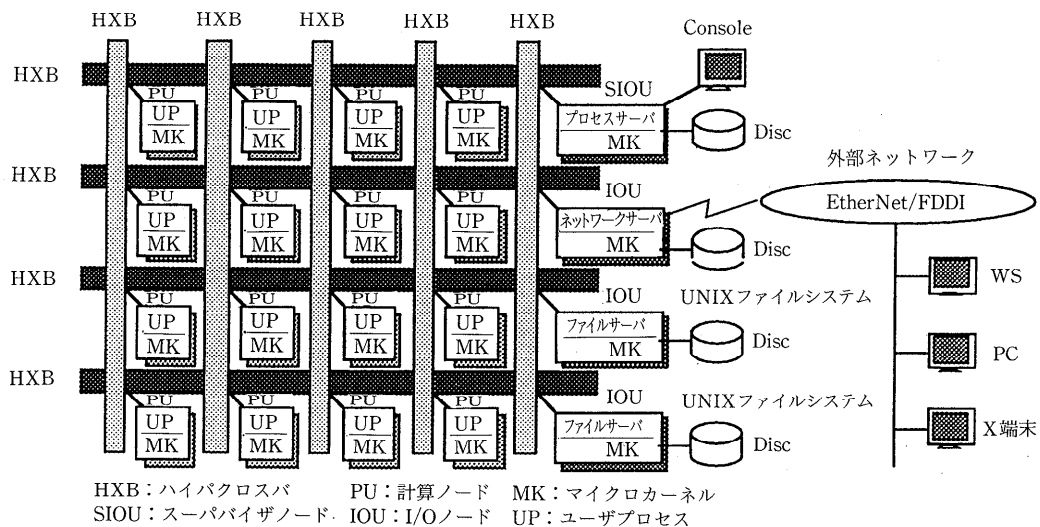


図-3 CP-PACSにおけるマイクロカーネル&サーバの実装

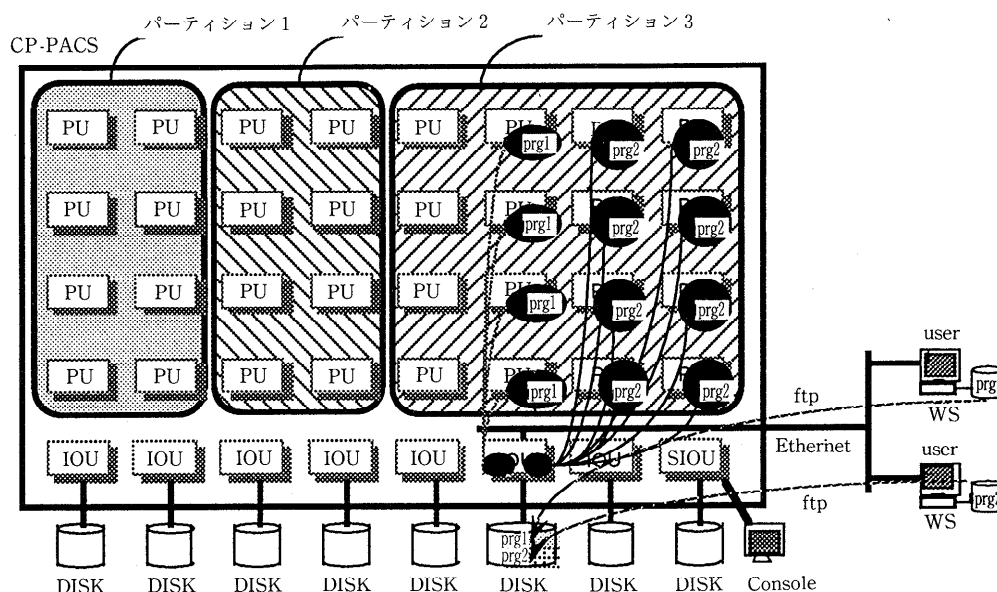


図-4 リモートワークステーションから rlogin して並列プログラムを実行する形態

ユーザプログラムはノード群を指定して、その中で実行される。ノードパーティションごとに、それを使えるユーザ（使用可能ユーザ）、1つのユーザプログラムに割り当てるノード数、同時実行されるユーザプログラム数などを制限できる。

(2) ノード割当て保護機能：ユーザプログラムはノードパーティション（ノード群）を指定してノード割当てを要求し、システムがそのときの稼働状況に応じて適当なノードを割り当てる。その際ノードの絶対アドレスを指定することはできない。このことと上記の制限により、不当なノードの使用を防止できる。この割り当てたノードの集まりをノードグループと呼ぶ。

図-4 にパーティション設定の例と、プログラム実行の様子を示す。

2.3 並列ファイル機能

CP-PACS¹⁾ではファイル入出力操作も並列実行できるようにするため、1024 台ないし 2048 台のプロセッサのうち 64 台のプロセッサがディスク装置を持つ構成になっている（ディスク装置を持つノードプロセッサを IOU と呼ぶ）。その構成を生かすファイルシステムとしては、すべての IOU 上のファイルが全体として 1 つのファイルシステムのように見える必要があり、しかも、複数のファイルへの同時アクセスが並列実行される

だけでなく、1つの論理ファイルへのアクセスも並列実行できるようにする必要がある。それを以下のように実現している。

ファイルシステムは全体として 1 つのツリーをなすものとし、どのノードからでも、ファイルの存在するノードを意識することなくファイル名だけでアクセスできるようにするため、各ノード上のファイルシステムは全体のファイルシステムの 1 つのディレクトリとして扱うことにする。各ノードからの入出力要求はシステムコールとして発行されるため、先に述べたマイクロカーネル・サーバの機構の働きにより、サーバプロセスに伝えられる。リモートノード上のファイルシステムは、サーバ内でディレクトリとして一元管理しているため、要求にしたがって、任意のファイルにアクセスすることができる。実際のファイルアクセスは、当該ファイルが属するノードにおいて動作し、結果が要求元に返される。

1つの論理ファイルへの複数ノードからの並列アクセスを可能にするために、ユーザプログラムがそれを意識する方法と、意識する必要のない方法の 2 つをもうけた。前者は、プログラムで自分のノードプロセッサの番号やそれに近い IOU ノードの番号を知ってアクセスする方法であり、後者を実現するものとしてストライプファイルシス

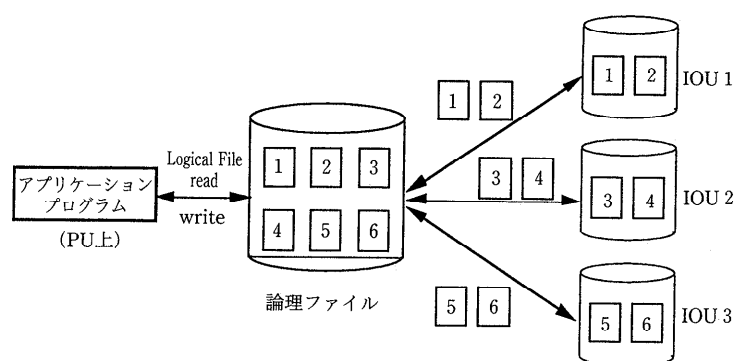


図-5 IOU 間ストライピング方式

テム (SFS) をもうけた。

SFS では、1つの論理ファイルを一定のブロックサイズ単位で分割し、複数の I/O ノード上のファイルに分散配置して、並列 I/O を実現する (図-5)。その際のファイル I/O インタフェースは従来のファイルシステムと同じで、プログラムでは特に意識する必要はない。

2.4 高速通信機能

分散メモリ型の超並列マシンによる大規模並列処理では、ネットワークを通じたノード間通信の高速化が最重要課題の1つである。さらに、その計算結果の大容量ファイルをフロント計算機システム (FCS) との間で高速に転送できる必要がある。

通常のノード間通信では、確実にデータを転送するため転送データをパケットに分割してヘッダを付加したり、受信したパケットからヘッダを削除してデータを組み立てる処理を行う。これらの処理を行うために、OS のカーネル内で転送データのバッファリングが必要となり、その結果カーネルとユーザ空間間でのデータコピーが発生し、これがノード間通信の性能を低下させる原因となっている。CP-PACS では、高速なノード間通信のためのハードウェアの機能として、直接転送先のメモリに書き込むリモート DMA が用意されている¹⁾。そこで、ソフトウェアとしては、その機能が最大限に生かされるようにライブラリを用意し、Fortran や C のプログラムから直接それら呼び出せるようにしている。また、後に述べる Parallelware の実現にもこのライブラリを活用している。

CP-PACS とフロントコンピュータシステム

(FCS) との間では、初期データ、計算結果、中間結果などのファイル転送が行われる。データ量としては、数十メガ～数十ギガバイトの複数ファイルの転送が考えられ、短時間で大容量のデータを転送するための高速ファイル転送機能が必要である。汎用プロトコル (TCP/IP) ではそれは実現できないと考えられたので、以下の特長を持つ高速ファイル転送機能 (HFTP) をもうけることにした。

(1) 複数の大容量のファイルを一括して高速に転送：複数ディスク装置に分散配置される複数ファイルを一括指定し、同時並列に転送することにより高スループットのファイル転送を実現する。

(2) 専用プロトコルによる高速データ転送：データコピーレスの専用通信プロトコルを用いることにより、汎用プロトコル (TCP/IP) では実現できていない実効性能の高いデータ転送を実現する。

(3) 標準ファイル転送プログラム (FTP) と同様のユーザインタフェース：HFTP におけるユーザインタフェースは、UNIX システムの標準となっている FTP コマンドを採用することにより、従来システムに近いイメージでの利用が可能になる。

3. プログラミング環境

プログラミング環境とは、プログラムの作成、コンパイル、デバッグ、実行、実行結果の表示、など、プログラムの開発を支援するためのソフトウェアシステムである。プログラム開発が困難であるといわれる超並列計算機用としては以下のよ

うな種々のシステムによる開発支援が必要になる。

(1) セルフ・システムまたは実機システム：目的とする並列計算機上のもの

(2) クロス・システム：ホスト計算機またはワークステーション上のソフトウェアであり、実機上で動くプログラムを生成する。

(3) シミュレーション・システム：ワークステーション上で実機での動きをシミュレートするシステム

また、メッセージ通信などを意識するのが苦手な一般ユーザでも比較的容易にプログラム開発ができることが必要であるし、マシンの性能を最大限に生かそうとするエキスパートにも使いやすい環境を提供する必要がある。以下にそれらをどのように実現しているか述べる。

3.1 プログラム言語

プログラム言語としては、アセンブラ言語、C、Fortran 90、HPF (High Performance Fortran)²⁾を備えている。一般ユーザには、通常のFortranプログラムにデータの分散配置を指定するだけで並列プログラムができるHPFのような標準的な言語が必要である。エキスパートには、CやFortranで並列を意識したプログラムを書き、それに並列に実行されるプロセス間でのメッセージ通信の命令(手続き呼び出し)を入れるようにする必要がある。

いずれの場合も、並列計算機の性能を最大限に引き出すには、コンパイラによる徹底した最適化が必要であり、それを実現すべく努力をしている。しかし、コンパイラによる最適化にも限度があるので、アセンブラ言語でのプログラミングも可能とし、そのプログラムのデバッグシステムも用意している。

コンパイラによる最適化としてCP-PACSで特徴的なのは、HPFコンパイラの最適化と擬似ベクトル機能を持ったスーパスカラ型プロセッサであるノードプロセッサのための最適化である。それらについては4.で述べる。

3.2 プログラム開発・デバッグ機能

プログラム開発用のツールとしては、エディタ、コンパイラ、デバッグ、性能モニタなどがある。ここでは、並列計算機で重要なものとなる並列プログラム開発環境、デバッグと性能モニタ、

および実機を使わずにそれらを可能とするシミュレータについて述べる。

並列プログラム開発環境としては、CP-PACS独自のものを開発するよりも、すでに世の中でよく使われているものを提供の方がよいと考え、米国ParaSoft社で開発されたParallelware³⁾を採用することにした(米国名はExpress⁴⁾)。さらに、パブリックドメインの標準となりつつあるPVM (Parallel Virtual Machine) インタフェースもサポートすることにした。

Parallelwareでは、ユーザプログラムに並列処理用の拡張関数呼び出しを組み込むことによって並列プログラムを可能とする。記述言語はCかFortranである。ツールとしてはグラフィカルなパフォーマンスモニタやシンボリックデバッグがある。なお、ParallelwareにはCP-PACS上のものとネットワークで結合されたワークステーションクラスタ上のものがあるので、後者でデバッグしてからCP-PACS上で実行させることができる。

その他のツールとしては、リモートカーネルデバッグ、並列モニタ、シミュレータなどがある。リモートカーネルデバッグはワークステーションなどをホストとして、CP-PACSのプログラムをデバッグするためのものであり、ホストマシン上からCP-PACS上のプログラムの実行制御をしたり、メモリやレジスタの内容の参照や更新ができる。並列モニタはCP-PACSハードウェアの稼働状況をリアルタイムでモニタするものである。シミュレータは擬似ベクトル命令などのCP-PACSハードウェア特有の機能をシミュレートし、性能チューニングのための情報を出力するものである。

4. 最適化コンパイラ

CP-PACSの最適化コンパイラにはC、Fortran 90、HPFのコンパイラがあり、最適化には各言語に依存する部分と、各言語に共通なバックエンドでの最適化がある。ここではHPFコンパイラ、およびバックエンドの最適化のうちCP-PACSの特徴である擬似ベクトル¹⁾の機能を活かす方法について述べる。

4.1 HPF コンパイラ

CP-PACS用のHPFコンパイラは、HPF言

語 (Version 1.0)²⁾の全仕様をサポートする。したがって、HPFの公式サブセットに含まれていない、配列の再分散 (REALIGN, REDISTRIBUTE) や FORALL 構造などの機能も使用できる。コンパイラの出力は、SPMD (Single Program Multiple Data-stream) 形式の Fortran 90 コードである。

並列化/最適化に関しては、主に以下のものをサポートする予定である。

(1) 通信の一括化

転送すべき複数の配列要素をまとめて1個のメッセージとすることにより、通信起動オーバーヘッドを減らす。この最適化は、連続した配列要素だけではなく、一定間隔 (ストライド) の非連続要素、さらに、異なる配列に属する要素も対象となる。

(2) グローバル演算の認識

分散配列に対する総和、最大値、最小値などの計算を行うループを認識し、グローバル演算ルーチンを用いた並列実行ループに変換する。

(3) 不規則参照ループの並列化

分散配列の添字が別の配列になっているループ、たとえば、

```
DO I=...
  A(I)=B(L(I))
ENDDO
```

のようなループでは、通信パターンが不規則になり、コンパイル時の通信一括化は困難である。CP-PACS 用 HPF は、このようなループに対して、Inspector/Executor 法⁵⁾に基づいた並列コードを生成する。これは、各ノード間で転送すべき要素をまとめたリストを実行時に作成し (Inspector フェーズ)、そのリストに従って各ノード間での一括転送を行う (Executor フェーズ) 方法である。

(4) DOACROSS ループの最適粒度調節

DOACROSS ループは、ループの繰返し間にまたがるデータフローのあるループである。たとえば次のようなループである。

```
DO I=...
  DO J=...
    A(I, J) =...A(I, J-1)+A(I-1, J) ...
  ENDDO
ENDDO
```

これを並列実行するときには、ループ内で通信を行いながら、パイプライン的に実行することになる。このとき、複数の繰返しをまとめて適切な粒度を選ぶことにより、性能が向上することが知られている⁶⁾。CP-PACS 用 HPF は、粒度最適化を施したコードを生成する。

4.2 擬似ベクトル処理

擬似ベクトル処理^{1),8)}は大きな配列を使った計算でのメモリレーテンシを隠すために考えられた方法であるから、配列を扱うループ (最内側のループ) にそれを利用した最適化をしなければならない。ループの目的コードに関する最適化で、命令のレーテンシを隠す方法としてよく知られているのがソフトウェア・パイプラインニング⁷⁾である。それは、ループ実行の命令列を複数のステージ (あるいはフェーズとも呼ぶ) に分割し、それらステージをパイプライン的に同時実行する方法であり、これによって命令のレーテンシを隠し、プロセッサ・リソースを効率的に利用するものである。我々も基本的にはその方式に従っている。ただし、擬似ベクトル処理には従来の方式と異なる点がいくつかある。

まず、例を示す。以下のプログラム片

```
for (int i=LOW; i<HIGH; i++) {
  tmp=c[i] * (tmp+d[i-1]);
  a[i]=tmp;
  b[i]=d[i];
}
```

は、2 命令同時発行可能なスーパースカラプロセッサでは、次のような目的コードへ変換される。ただし、命令を読みやすくするために、通常の機械語表現ではなく、代入文に模した表記を用いた。また、ここには定常的な繰返し部分 (kernel) だけ示してある。最後の命令はループのインデックスを増加し終了判定をする命令である。なお、ここでは簡単のため遅延分岐は考慮していないが、実際のコンパイラではもちろんそれも考慮している。

```
実行サイクル      | 同時に実行される命令
0:  b[i-1]:=R(30)
      | R(29):=R(28) | R(30)
1:  a[i-1]:=R(28)
2:  R(125):=c[i+47]
      | R(30):=R(31)*R(29)
```

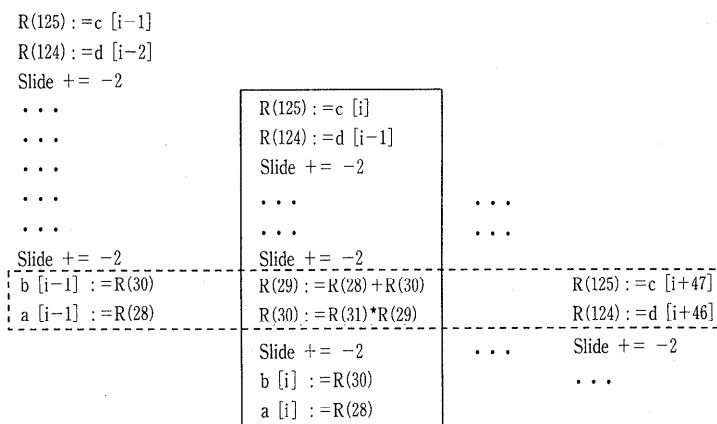


図-6 ループ1回分の実行命令と kernel

- 3: R(124) := d[i+46]
- 4: Slide += -2
- 5: CondBranch

このプログラムは以下のような考えで生成されている。まず、もとのプログラムのループ1回分の実行命令を考えてみる（図-6の実線で囲った部分）。最初に、次の2つのpreload命令で、現在のウィンドウでは見えていない遠くのレジスタにcとdをロードする（R(0)からR(31)までが見えているレジスタである）。

R(125) := c[i]

R(124) := d[i-1]

この後で、ウィンドウをスライドさせる命令「Slide += -2」を1回実行すると、上の命令の対象となっているレジスタの論理番号はR(125), R(124) から R(123), R(122) へと変化する。スライド命令を47回実行したときに、これらはR(31), R(30) となってウィンドウの中に入るので「tmp = c[i] * (tmp + d[i-1]);」の演算を実行することができる（メモリのレーテンシが相当大きくてもそれまでにはロードが完了しているはずである）。その命令が「R(29) := R(28) + R(30)」と「R(30) := R(31) * R(29)」である。ここでは、加算のレーテンシは2としている。前者の命令のR(30)はd[i-1]の値であり、後者の命令のR(30)はtmpのレジスタである。このR(30)はスライド命令を1回実行するとR(28)となる（それが前者の命令でR(28) = tmpとして使われている）。したがって「a[i] = tmp;」は次のスライド命令を実行してから「a[i] := R(28)」とす

ればよい。

以上の命令の繰返し実行を滞りなく実行するためには、c[i]などに関してこれらの計算をすると同時に、後の繰返しのためにc[i+47]のpreload命令などを発行し、前の繰返しで計算したtmpの値をa[i-1]にpoststoreする必要がある。それら（図-6の点線で囲った部分）を、命令の同時発行の可否、発行のタイミング等を考慮の上、詰め込んだものが前記のプログラムである。

このプログラムを生成する手続きの概略は以下のようなものである。

- (1) データ依存グラフ (data dependence graph) の作成
- (2) ループ立ち上げ間隔 (loop initiation interval: II と略記される) の最小値決定
- (3) リソース予約テーブル (resource reservation table) への命令の埋め込み
- (4) レジスタ割り付け

データ依存グラフはどの値がどの演算に使われるかを示すグラフである。今の例ではループ伝搬 (loop carried) の依存もある。ループ立ち上げ間隔IIは定常的な繰返し部分のサイクル数であり、上の例では6である。その最小値はデータ依存グラフ上の演算の個数とマシンの特性から決まる。リソース予約テーブルとしては、まずIIの最小値の長さを持ったプロセッサのリソースの表（たとえば、整数演算と浮動小数点演算とロード/ストアの3種類の命令が同時に発行できるプロセッサではII行3列の表）を作り、そこにデータ依存グラフで示される命令を埋め込んでいく。全

部の命令を埋め切れればよいが、埋めきれなかった場合はIIを増やしてやり直す。最後にレジスタ割り付けを行う。

以上は一般的な手続きであるが、CP-PACSの擬似ベクトル処理を実現するコンパイラの手続きは、以下の点で通常のソフトウェアパイプライン処理を実現するコンパイラの手続きと異なる。

まず、繰返し処理の度に論理レジスタ番号が一定の刻み幅で変化していくから、複数のパイプライン・ステージで使用するレジスタ番号の間の干渉が容易に回避できるという利点がある。いわゆるレジスタの名前替え (register renaming) が実現されているのである。通常のソフトウェア・パイプラインでは、このレジスタ間の干渉のため、ループ立ち上げ間隔はそのループ処理に含まれる命令の中の最大のレーテンシ以下にはできない。ほとんどの場合そのような命令はロード命令であり、そのレーテンシは相当に大きな値であるため、問題は深刻である。その対策として一般にはループ展開 (loop unrolling および modulo variable expansion) を行い、かつ命令スケジューリングをより巧妙にするが、擬似ベクトル処理ではそのような技巧を凝らす必要がないため、ループ立ち上げ間隔の算出や命令スケジューリングのアルゴリズムがごく自然な発想に基づいて素直に実現できる。ただし、レジスタ番号が変化していくためレジスタ割当てのアルゴリズムが従来のものよりも複雑になる。

なお、if文を含んだループのソフトウェア・パイプラインは困難な課題である^{7),9),10)}が、その解決策は得ている¹¹⁾。また、多重ループのソフトウェア・パイプラインについても検討を進めており¹²⁾、これらは順次、実際のコンパイラに組み込んでいく予定である。

5. おわりに

CP-PACSのソフトウェアの設計目標と、その実現方式の概要を述べた。CP-PACSは1996年3月に稼働開始予定であり、その後は稼働実績をふまえた評価・改良を進めていく予定である。

謝辞 (株)日立製作所には、ソフトウェア全体の設計・開発をお願いしている。関係者に深く感謝します。

参考文献

- 1) 中澤喜三郎, 朴 泰祐, 中村 宏: 超並列計算機CP-PACSのアーキテクチャ, 情報処理, Vol. 37, No. 1, pp. 18-28 (1996).
- 2) High Performance Fortran Language Specification, High Performance Fortran Forum, Version 1.0 (May, 1993), Version 1.1 (Nov. 1994).
- 3) ParallelWare Reference Manual, Nippon Steel Corporation and Nichimen Data Systems Corporation.
- 4) <http://www.parasoft.com/express.html>
- 5) Saltz, J., Crowley, K., Mirchandaney, R. and Berryman, H.: Run-Time Scheduling and Execution of Loops on Message Passing Machines, J. of Parallel and Distributed Computing, Vol. 8, pp. 303-312 (1990).
- 6) Hiranandani, S., Kennedy, K. and Tseng, C.-W.: Evaluating Compiler Optimizations for Fortran D, J. of Parallel and Distributed Computing, Vol. 21, pp. 27-45 (1994).
- 7) Lam, M. S.: A Systolic Array Optimizing Compiler, Kluwer Academic Pub. (1989).
- 8) Nakamura, H., Imori, H., Nakazawa, K., Boku, T., Nakata, I., Yamashita, Y., Wada, H. and Inagami, Y.: A Scalar Architecture for Pseudo Vector Processing based on Slide-Window Registers, ACM Supercomputing'93, pp. 298-307 (1993).
- 9) Dehnert, J. C., Hsu, P. Y. and Bratt, J. P.: Over-lapped Loop Support in Cydra 5, Proc. 3rd ASPLOS, pp. 26-38 (1989).
- 10) Warter, N. J., Haab, G. E. and Bockhaus, J. W.: Enhanced Modulo Scheduling for Loops with Conditional Branches, Proc. IEEE Micro-25, pp. 170-179 (1992).
- 11) 山下義行, 中田育男: ループ中に条件分岐を含む場合の最適なソフトウェア・パイプラインング, 並列処理シンポジウム JSPP'94, pp. 17-24 (May, 1994).
- 12) 山下義行, 中田育男: ソフトウェア・パイプラインングにおける多重ループの最適化, 並列処理シンポジウム JSPP'95, pp. 185-192 (May, 1995).

(平成7年6月28日受付)

**中田 育男 (正会員)**

1935 年生。1958 年東京大学理学部数学教授。1960 年同大学大学院修士課程修了。1960～79 年(株)日立製作所中央研究所、同システム開発研究所勤務。1979 年 4 月より筑波大学電子・情報工学系教授・理学博士。プログラム言語、言語処理系、ソフトウェア工学などに興味を持っている。著書「コンパイラ」(産業図書)、「基礎 FORTRAN」(岩波書店)、「コンパイラ」(オーム社)。ソフトウェア科学会、電子情報通信学会、ACM、IEEE 各会員。

**山下 義行 (正会員)**

1959 年生。1982 年大阪大学理学部卒業。日立マイクロコンピュータ・エンジニアリング(株)入社。1986 年退社。1987 年筑波大学大学院博士課程電子・情報工学系入学。1989 年退学。東京大学大型計算機センター助手。1992 年、筑波大学電子・情報工学系講師。1995 年、同助教授。プログラミング言語、コンピュータ・グラフィックスの研究に従事。日本ソフトウェア科学会会員。

**小柳 義夫 (正会員)**

1943 年生。1966 年東京大学理学部物理学科卒業。1971 年同大学院理学研究科物理学専門課程修了。理学博士。同年同大助手。高エネルギー物理学研究所理論部門助手。筑波大学電子情報工学系講師、助教授、教授を経て、1991 年東京大学理学部情報科学科教授。並列処理、数値解析、計算物理学に関する研究に従事。とくに偏微分方程式の高速並列解法、最小二乗法の数値計算、乱数やモンテカルロ法に興味をもつ。物理学会、日本統計学会、応用統計学会、計算機統計学会、応用数理学会各会員。

訂 正

本誌第 36 巻 12 号 pp.1155-1165 に掲載されました解説「World-Wide Web (WWW)」の著者紹介文 (p.1165) に以下のような誤りがありました。

誤→“大庭袋圭祐”

正→“木庭袋圭祐”

本誌第 37 巻 1 号 pp.11-17 に掲載されました特集「計算物理学と超並列計算機－CP-PACS 計画－ 1. 計算物理学と CP-PACS 計画」の岩崎洋一氏および梅村雅之氏のご所属 (p.11) に以下のような誤りがありました。

誤→“++ 筑波大学物理各系および計算物理学研究センター”

正→“++ 筑波大学物理学系および計算物理学研究センター”

また、本誌第 37 巻 1 号 pp.29-37 に掲載されました特集「計算物理学と超並列計算機－CP-PACS 計画－ 3. 超並列計算機 CP-PACS のソフトウェア」の中田育男氏の著者紹介文 (p.37) に以下のような誤りがありました。

誤→“1958 年東京大学理学部数学科教授。”

正→“1958 年東京大学理学部数学科卒業。”

執筆者の方々および関係者の皆様にご迷惑をおかけしたことをお詫びして訂正いたします。