計算機アーキテクチャ 113−24
（1995. 8. 25）

RWC-1のシステム構成と基本動作

坂井修一 松岡浩司 岡本一晃 横田隆史 廣野英雄 児玉祐悦† 佐藤三久†

新情報処理開発機構 †電子技術総合研究所

超並列計算機RWC-1を開発中である。開発の目的は、(1) RWC研究プロエジェクトにおける計算基盤を与えること、(2) 近未来の汎用超並列計算機アーキテクチャの規範を示すこと、の両者にある。本稿では、RWC-1のシステム構成と基本動作について述べる。

RWC-1は、コンティニュエーション駆動型実行モデルに基づく計算機である。同モデルは、パケット処理と計算実行を自然に融合したモデルであり、効率良い超並列処理の枠組を与えている。RWC-1のプロセッサは、RICA(Reduce Interprocessor-Communication Architecture)と呼ばれるアーキテクチャを採用し、通信と計算が高度に一体化した処理を行なう。RWC-1の相互結合網は、階層化MDCE(Multi-dimensional Directed Cycles Ensemble)と呼ばれるものであり、優先度処理機構、ドレイン機構などをもつ。RWC-1の入出力系は2階層の専用相互結合網をもち、上位階層ではATMスイッチを用いたデータ交換を実現している。入出力機器としては、RAIDを基本とした大容量二次記憶システム、画像および音声インタフェース、外部ネットワーク・インタフェースをもつ。

System Design and Basic Operations of RWC-1

Shuichi SAKAI Hiroshi MATSUOKA Kazuaki OKAMOTO Takashi YOKOTA Hideo HIRONO
Yuetsu KODAMA † Mitsuhisa SATO †

Real World Computing Partnership †Electrotechnical Laboratory
Tsukuba Mitsui Building 16F, 1-6-1 Takezono, Tsukuba, Ibaraki 305, Japan

We are now developing a massively parallel computer RWC-1. The objectives are (1) to provide computational infrastructure for RWC project and (2) to pursue a general purpose stand-alone massively parallel system efficiently supporting multiple programming paradigms. This paper presents the system design and basic operations of RWC-1.

RWC-1 is based on Continuation Driven Execution Model. The model naturally integrates packet handling and instruction execution, and gives the basis for efficient massively parallel computation. RWC-1 processor is based on RICA, Reduced Interprocessor-Communication Architecture where communication and computation are tightly fused. An interconnection network for communication is a layered MDCE, Multi-Dimensional Directed Cycles Ensemble. It contains priority control mechanisms and draining mechanisms. I/O subsystem of RWC-1 contains dedicated two-level interconnection networks upper class of which uses ATM switches. Major I/O devices are mass storage system based on RAID, audio/visual interface and network interface from/to outer world communication.

1 はじめに

超並列計算機RWC-1[1]を開発中である。開発の目的は、(1) RWC研究プロエジェクトにおける計算基盤を与えること、(2) 近未来の汎用超並列計算機アーキテクチャの規範を示すこと、の両者にある。本稿では、RWC-1のシステム構成と基本動作について述べる。

最初に、RWC-1アーキテクチャの要件について触れた後、RWC-1の全体構成を述べ、プロセッサアーキテクチャ、相互結合網、入出力アーキテクチャの概要を述べる。次に、RWC-1の基本動作（＝マルチスレッド処理）を、コンティニュエーション駆動実行モデル[2]に基づいて定式化する。最後に、RWC-1開発の現状と予定に関して簡単に報告する。

2 RWC-1のシステム構成

2.1 アーキテクチャの要件

RWC-1は、新情報処理研究プロジェクト(RWCP)において計算インフラとなる計算機であり、同時に汎用超並列計算機の規範を示す役割を負っている。RWCにおける情報処理は、(1) パターンと記号の融合した処理、(2) 時間制約のある情報処理、(3) 新しい計算原理にもとづく情報処理、などであり、汎用計算機としては、(4) 数値計算、(5) 並列記号処理、なども対象となる。

これらを効率的に処理する計算機アーキテクチャの要件は、(a) 高スループット、(b) 優れた応答性、(c) 不定型の問題に対する高い適応能力、(d) 極細粒度、(e) 強化されたI/O、(f) 並列OSの支援、(g) 高い信頼性、などである。要約すると、RWC-1は、「安全で応答性に優れた極細粒度超並列計算機」でなければならない。

本要件に対してわれわれは、(1) コンティニュエーション駆動実行モデルとプロセッサ・アーキテクチャRICA、(2) 大域的仮想化、(3) 優先度の支援、(4) 時分割・空間分割の支援、(5) 相互結合網MDCE、(6) 独立型の入出力系、などのアーキテクチャ要素技術・統合技術を提案・提唱し、解決をはかった。これらはすべて、実マシンRWC-1上に実装中である。

2.2 全体構成

図1にRWC-1の全体構成を示す。RWC-1は、1024個の要素プロセッサ（ＰＥ）とそれをつなぐ相互結合網、入出力用相互結合網、大容量二次記憶系、画像・音声インタフェース、および外部とのインタフェースから成る。

2.3 プロセッサアーキテクチャ

RWC-1のプロセッサは、市販のプロセッサチップを用いず、オリジナルのチップを製作し、これを用いる。これは、2.1の要件を満たす商用プロセッサチップが存在しないからである。

図2にプロセッサ・アーキテクチャを示す。プロセッサの特徴は、(1) 通信と演算が高度に融合したアーキテクチャであるRICA、(2) 超並列ＯＳの支援機能、などである。これらの詳細は、[3] [4] [5] [6]に報告した。

プロセッサは、BSB(Buffering and Scheduling Block)、EXB (EXecution Block)、POB (Packet Output Block)、MCB(Memory Control Block)およびMAINT (MAINTenance Block)から成る。特徴的なブロックは、BSBとPOBであり、それぞれパケットのバッファリングと処理の起動、パケットの生成・出力を、ハードウェア的に行なう。

2.4 相互結合網

RWC-1の相互結合網は、ATMスイッチやCaltech chipなど市販の要素スイッチを用いず、オリジナルのチップを製作してこれを用いる。これは、RWC-1で要求される通信速度や通信機能を商用チップ（を用いて構成される相互結合網）が満たさないからである。

RWC-1の相互結合網は、MDCE(Multi- dimensional Directed Cycles Ensemble)網を階層化して用いる[7]（図3）。本網は、(1) 高スループット、(2) 低遅延、(3) セルフルーティングが容易、(4) 効率良い空間分割が容易、など優れた特質をもつ。また、RWC-1の相互結合網は、SFD (Store and Forward Deadlock)防止のための仮想チャネル機構、優先度処理機構、時分割のためのドレイン機構などが実装される。

2.5 入出力アーキテクチャ

RWC-1の入出力系は、(1) 二階層の相互結合網[8][9]、(2) 大容量二次記憶系[10]、(3) 画像・音声インタフェース、および外部とのインタフェースなどから成る（図1）。

入出力用の相互結合網を二階層にしたのは、プロセス内部の局所性を利用するためと、システムの空間分割に効率的に対応するためである。下位階層の相互結合網は独自にリングバスを開発し、上位階層の相互結合網はATMスイッチなど市販ハードウェアを採用している。

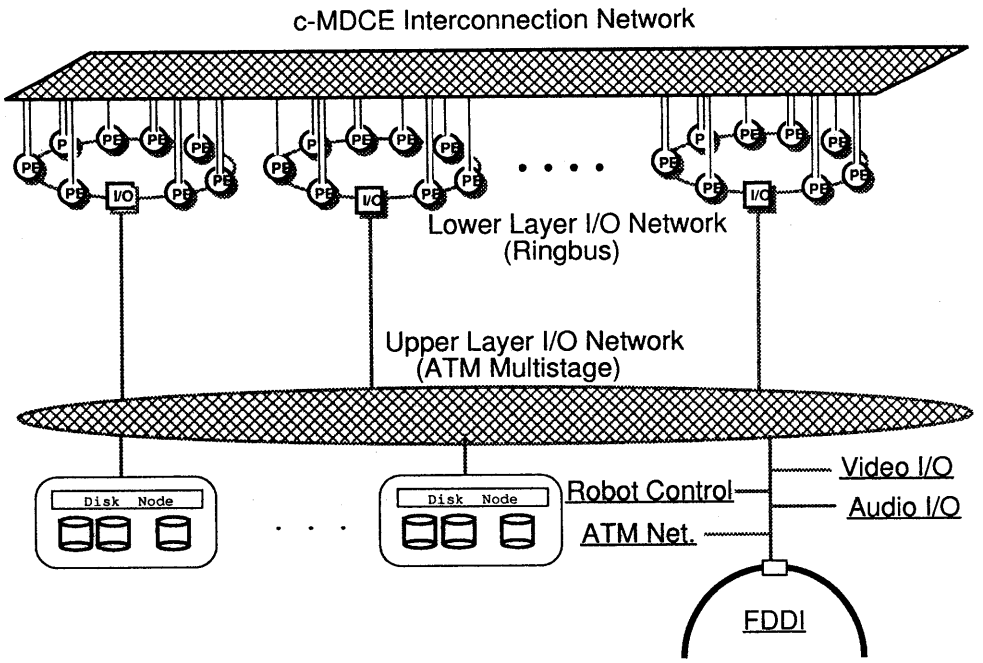
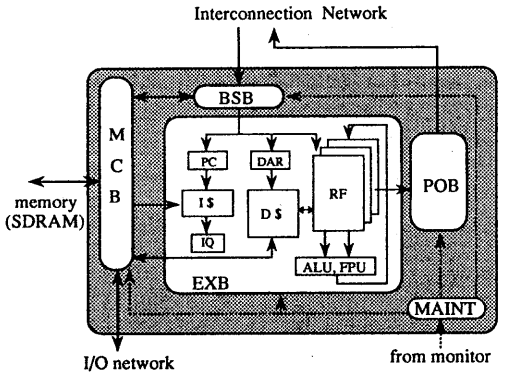


図 1: RWC-1 全体構成.



PC : Program Counter
DAR: Data Address Register
I$: Instruction Cache
IQ : Instruction Queue
D$: Data Cache
RF : Register File

図 2: RWC-1 プロセッサアーキテクチャ.

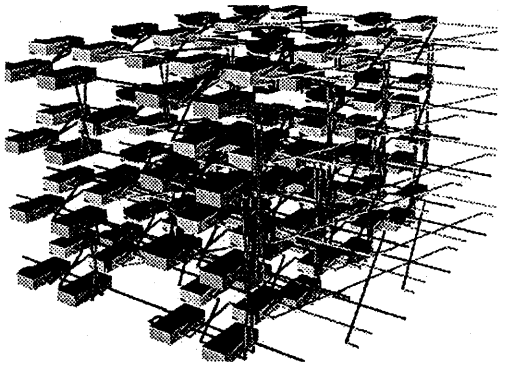


図 3: 階層型 MDCE 網.

プロセッサは、相互結合網やメモリとは独立に入出力用のインタフェースをもち、入出力と演算処理の間でメモリインタフェースを共有する形になっている。

二次記憶系においては、複数の RAID とこれを制御するディスク・ノードを 1 モジュールとし、複数のモジュールを効率良く稼働させている。画像・音声イ

ンタフェースは、それぞれRWCの応用群で用いるパターン処理を効率良く行なう処理速度・転送速度を実現する。

3 RWC-1の基本動作

本節では、RWC-1におけるスレッド処理を、コンティニュエーション駆動実行モデルに基づいて定式化する。

3.1 コンティニュエーション駆動型実行モデル

図4に、コンティニュエーション駆動型実行モデルを示す。

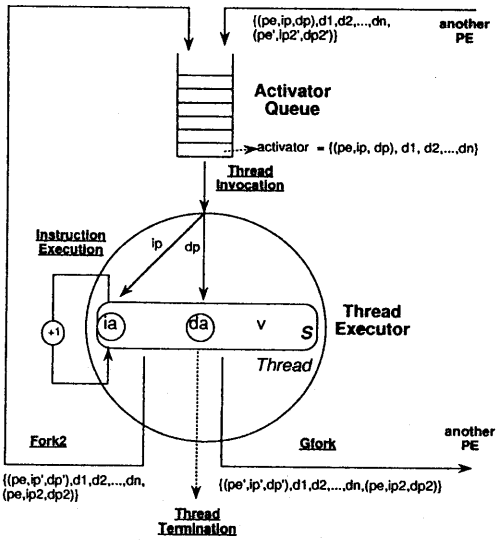


図4: コンティニュエーション駆動実行モデル.

本モデルでは、各プロセッサにおいて連続して実行される命令列をスレッドと呼ぶ。スレッドの状態は、次のように表される。

$$s =< pe, ia, da, v >$$

ここで、peは、当該スレッドを実行するPE番号、iaは、現在実行中の命令番地、daは、当該スレッドの作業領域のデータ番地、vはデータ作業領域内に保持されない（レジスタなど一時記憶に保持される）内部状態である。

各スレッドは、アクティベータと呼ばれるトークンによって起動される。アクティベータは、次のように表される。

$$a = \{c, d_1, d_2, d_3, \cdots, d_n\}$$

ここで、cはコンティニュエーションであり、d_iは、データである。ここでいう「データ」は、通常の意味のデータ以外に、番地であったり、コンティニュエーションであったりしてよい。

コンティニュエーションとは、次のようなものである。

$$c = (pe, ip, dp)$$

ここで、peは当該アクティベータの行先のPE番号である。ipは命令ポインタ、dpは、データポインタであり、それぞれスレッドの先頭命令と作業領域の番地を指示している。

本モデルの抽象プロセッサは、アクティベータキュー部とスレッド実行部からなる。前者は、発火待ちのアクティベータを保持し、後者が使用可能となったところでアクティベータを送り込む。スレッド実行部の役割は、特定スレッドの命令（群）を実行すること、新しいアクティベータを生成すること、現在実行中のスレッドを終了すること、などである。

3.2 プロセッサの基本動作

3.2.1 インプリメンテーションの概略

RWC-1では、プロセッサ間でやりとりされるアクティベータは、可変長パケットとして実現され、プロセッサ内で受け渡されるアクティベータは、「レジスタ上に保持されたコンティニュエーションおよびデータ」として実現される。

アクティベータ・キューは、優先度つきのハードウェア・バッファであり、BSB(図2)として実装されている。スレッド実行部のうち、通常の命令を実行する部分はスーパスカラRISCの形態を取り、EXBがこれを実現する。新しいアクティベータを生成するのは、POBである。POBはEXB内の各演算装置（ALU, FPUなど）とは非同期に動作する。

3.2.2 スレッドの起動

スレッドは、通常、パケットによって起動される。起動は以下のような操作である。

1. ハードウェア・バッファの中から最も優先度の高いパケットが自動的に取り出され、EXBに送られる。

2. EXBでは、パケットに含まれる情報は以下のようにレジスタ群に格納される。

$$\begin{array}{rcl} pc & \leftarrow & ip; \\ dar & \leftarrow & dp; \\ register\ file & \leftarrow & d_1, d_2, \cdots, d_n; \end{array}$$

ここで、$ip, dp, d_1, d_2, \cdots, d_n$は、パケットの構成要素である。上記の3行の操作のうち、最初の2行は全体で1クロックで完了し、最後のデータ転送は、1データあたり1クロックの操作となる。さらに、後者はスレッドの実行と重畳化される。

3. スレッド実行が始まる。

以上のようにRWC-1では、パケットの受信からスレッドの起動に至るまでを完全にハードウェアが行ない、一切の命令実行（ソフトウェア）を伴わない。

3.2.3 命令の実行

スレッド内部の命令実行は、以下のようにスレッドの内部状態が変化していく過程として捉えられる。

$$\begin{array}{rl} & < pe, ia, da, v > \\ \rightarrow & < pe, ia+1, da, v > \\ \rightarrow & < pe, ia+2, da, v > \\ \rightarrow & \cdots \end{array}$$

これは、通常のRISCプロセッサと同様の過程である[1]。

もしiaが、分岐命令やジャンプ命令であった場合、スレッドはその命令の実行の結果ジャンプを起こす。

$$\begin{array}{rl} & < pe, ia = \text{“}Jump(idpl)\text{”}, da, v > \\ \rightarrow & < pe, ia+idpl, da, v > \\ \rightarrow & < pe, ia+idpl+1, da, v > \\ \rightarrow & \cdots \end{array}$$

ここで、$idpl$はジャンプする相対番地である。

RWC-1では、スレッド起動のときだけ、パケットが受け取られる。スレッドの途中の命令によって、パケットが受け取られることはない。

前述のように、RWC-1はスーパスカラ動作を行なうので、上の式において個々の動作は複数命令の同時実行を含むことになる。

[1]命令によって、daおよびvが変化することがある。

3.2.4 スレッドの終了

当該スレッドは、*Break*命令の実行によって終了する。

$$\begin{array}{rl} & < pe, ia = \text{“}Break\text{”}, da, v > \\ \rightarrow & <> \end{array}$$

スレッド終了時に、BSB内にパケットがなかった場合、プロセッサはアイドル状態になる。BSB内にパケットが存在した場合、即座に新しいスレッドが起動される。この場合の手順は以下の通り。

$$\begin{array}{rl} & < pe, ia = \text{“}Break\text{”}, da, v > \\ \rightarrow & < pe, ia', da', v' > \\ \rightarrow & < pe, ia'+1, da', v' > \\ \rightarrow & \cdots \end{array}$$

上記の中で、第二行が新しいスレッドの開始を意味している。

なお、RWC-1では、優先度の高いパケットは低いスレッドの処理をpreemptすることができる。この場合も、即座に新しいスレッドが起動され、パイプライン・バブルは発生しない。

3.2.5 戻り値なしのFORK

FORKは、関数インスタンス（手続き）内部における新しいスレッドの生成である。RWC-1では、スレッドは通常、パケットによって生成されるから、FORKは、パケットの生成を意味する。本パケットは、コンティニュエーションと引数データを保持している。

パケット生成の過程は、以下の通り。

$$\begin{array}{rl} & s_k = < pe, ia = \text{“}Fork1(ia', ddpl')\text{”}, da, v > \\ \rightarrow & s_{k+1} = < pe, ia+1, da, v > \\ \rightarrow & \cdots \end{array}$$

ここで、$ddpl'$は、新しいスレッドのデータの変位である。また、$Fork1$は、実際にはMKPKT命令によって実現される。

$Fork1$の実行の直後に、次のパケットが生成される。

$$\{(pe, ia', da+ddpl'), d_1, d_2, d_3 \cdots, d_n\}$$

ここで、d_iは新しいスレッドの引数である。パケットは、すぐにハードウェア・バッファに格納される。FORKの実行後ももとのスレッドの命令列は引続き実行され、その終了後に、新しいスレッドが次のように起動される。

$\rightarrow$ $s'_1 =< pe, ia', da + ddpl', () >$
$\rightarrow$ $s'_2 =< pe, ia' + 1, da + ddpl', () >$
$\rightarrow$ $\cdots$
$\rightarrow$ $s'_m =< pe, ia' + m = \text{“}Break\text{”}, da + ddpl', () >$

ここで、() は vが空であることを示す。

もし、状態 s_{k+1}が $Break$命令を含む場合、2つのスレッドが直列に実行されることになる。この場合、$Fork1$ 命令と $Break$は、$SChain$ という一つの命令によって下のように実現される。

$$s_k =< pe, ia = \text{“}SChain(ia', ddpl')\text{”}, da, v) >$$

$SChain$ は、RWC-1 では、JUMPC(JUMP Continuation) 命令によって実現される。

3.2.6 戻り値のある FORK

新しく生成されたスレッドがもとのスレッドに値を返す必要がある場合、FORK は次のような手順となる。

$s_k =$ $< pe, ia = \text{“}Fork2(ia', ddpl', ia1, ddpl)\text{”}, da, v) >$
$\rightarrow$ $s_{k+1} =$ $< pe, ia + 1, da, v >$
$\rightarrow$ $\cdots$
$\rightarrow$ $s_{k+i} =$ $< pe, ia + i = \text{“}SChain(ia1, ddpl')\text{”}, da, v >$

ここで、$Fork2$(実際は MKPKT) は、次のパケットを生成する。

$$\{(pe, ia', da + ddpl'), d_1, d_2, d_3, \cdots, d_n, (ia1, ddpl)\}$$

ここで d_iは新しいスレッドの引数データ、$(ia1, ddpl)$ は戻り値のためのコンティニュエーションを表す。本パケットは、必要に応じてハードウェア・バッファに格納され、新しいスレッドを起動する。

もとのスレッドの終了以前に、戻り値と待ち合わせをするためのコンティニュエーションが生成され、先とは別の FORK が行なわれる。その結果、第三のスレッドが生まれ、ここで第一のスレッドが生成したデータと第二のスレッドの戻り値が一緒になる。

新しいスレッドは以下のように起動され、実行される。

$\rightarrow$ $s'_1 =$ $< pe, ia', da + ddpl', (ia1, ddpl) >$
$\rightarrow$ $s'_2 =$ $< pe, ia' + 1, da + ddpl', (ia1, ddpl) >$
$\rightarrow$ $\cdots$
$\rightarrow$ $s_{m+i} =$ $< pe, ia + i = \text{“}SChain(ia1, ddpl)\text{”}, da + ddpl', v >$

2つのパケットが $ia1$ で待ち合わせされる. この待ち合わせは、次節で述べる $JOIN$ 演算である。

3.2.7 JOIN

JOIN は、コンティニュエーション駆動型実行モデルにおけるマイクロ同期である。2個ないし3個以上のアクティベータが特定スレッドの入口で待ち合わせを行なう。

$s_1 =< pe, ia = \text{“}Join\text{”}, da, wcf >$
$\rightarrow$ $s_2 =< pe, ia + 1, da, () >$
$\rightarrow$ $\cdots$

ここで、wcfは待ち合わせフラグである。$Join$ は、到着したアクティベータの中身に従って、wcfを更新する。wcfが適切な値になったとき、Join は wcfをクリアし、s_2に分岐する。その他の場合は、Join は、引数データをメモリに格納し、いったんスレッドを終了する。

上記の操作の中には、wcfのテストアンドセット(不可分操作) が含まれる。wcfは、データ駆動計算機におけるマッチング・フラグと同様に実現することもできるし、カウンタとして実現することも可能である。

RWC-1 では、$Join$ は BSYNC 命令によって実現されている。

3.2.8 関数コール・リターン

関数コールは、(1) 新しい関数インスタンスのためのデータ作業領域の確保およびこれとコード領域との結合、(2) 引数データおよび戻り番地の引き渡し、から成る。

(1) のためには、関数を起動する前に、新しい関数インスタンスのデータ作業領域を得て、その dp を確保しておかなくてはならない。その後で、引数データの引渡しがなされる。実際には、一回のパケット転送で両者を実現することが可能である。

最初に、呼びだし側は次の命令を実行する。

$$< pe, ia = \text{“}Call(pe', Cia, Cddpl, ia', ddpl', ia2, ddpl)\text{”}, da, (v, Cdp) >$$

ここで、Cia, Cdp と $Cddpl$は、それぞれ「新しい関数インスタンスのデータ領域を確保するルーチン」の命令番地、データ番地、データの変位である[2]。ia' と $ddpl'$ は、それぞれ新しい関数インスタンスの入口スレッドの命令番地とデータの変位である。さらに、$ia2$ と $ddpl$は、それぞれ、返り値を受けとるスレッドの命令番地とデータの変位である。

$Call$命令 (MKPKT で実現) は、次のパケットを生成し、送出する。

[2] Cia, Cdp and $Cddpl$は、システム初期化のさいに決められる定数である

$\{(pe', Cia, Cdp1), d_1, d_2, d_3, \cdots, d_n, (ia', dppl')$
$(pe, ia2, dp2)\}$

ここで、$Cdp1 \equiv Cdp + Cddpl$であり、$dp2 \equiv da + dppl$である。$(ia', dppl')$という2つ組は、新しい関数インスタンスの入口のスレッドを起動するコンティニュエーションとなる。$(pe, ia2, dp2)$という3つ組は、戻り用のコンティニュエーションである。

本パケットは、関数起動ルーチンを起動する。関数起動ルーチンは、以下の手続きを実行する。

1. dp'によって示される新しいデータ作業領域を確保する
2. 同領域をコード領域と結合する
3. 戻り値のないFORKを実行する（以下に記述）

$$< pe', Cia' = \text{“}SChain2(ia', ddpl')\text{”}, Cdp1,$$
$$(v', dp', (pe, ia2, dp2)) >$$

ここで、$SChain2$は次のアクティベータを生成し、当該スレッドを終了する。

$$\{(pe', ia', dp1'), d_1, d_2, \cdots, d_n, (pe, ia2, dp2)\}$$

ここで、$dp1' \equiv dp' + ddpl'$である。本アクティベータは、以下のように、新しい関数インスタンスの入口のスレッドを起動する。

$$< pe', ia', dp1', (pe, ia2, dp2) >$$
$$\rightarrow \quad < pe', ia' + 1, dp1', (pe, ia2, dp2) >$$
$$\rightarrow \quad \cdots$$

新しい関数インスタンスで計算が進み、これに属するあるスレッドが値を返すとき、次のようにGChain命令が実行される。

$$< pe', ia''' = \text{“}GChain\text{”}, dp'', (rd, (pe, ia2, dp2)) >$$

ここで、rdは計算の結果得られた戻り値である。$GChain$は、次のパケットを生成し、呼びだし側にこれを送りつける[3]。

$$\{(pe, ia2, dp2), rd\}$$

呼びだし側では、新しいスレッドが本パケットによって起動され、戻り値によって新しい計算が行なわれる。

[3] $GChain$の後に本スレッドは、データ作業領域を開放し、その後に関数インスタンスが終了する

以上において、$SChain2$は$SChain$と同じ命令(JUMPCで実現)だが、$da' + ddpl'$ではなく$dp' + ddpl'$でデータ番地を生成する点、$(pe, ia2, dp2)$をアクティベータに入れている点が異なる。

本関数コールは、peとpe'の間で必要な通信が一度で済む（ハンドシェークが不要な）点が優れている。

以上がRWC-1におけるスレッド処理の基本である。RWC-1における各動作の並列化（スーパスカラ化を含む）・重畳化や付加ハードウェアによる高速化などに関しては、[5] [6]を参照されたい。

4 おわりに

超並列計算機RWC-1のシステム構成と基本動作について述べた。RWC-1は、コンティニュエーション駆動型実行モデルに基づく計算機である。同モデルによって、パケット処理と計算実行は自然に融合され、効率良い超並列の枠組が与えられる。RWC-1は同モデルに基づくプロセッサ・アーキテクチャRICAを採用し、さらに、MDCE相互結合網、独立型入出力系を採用して、RWC応用群など次世代の超並列計算に対応している。

RWC-1の開発の現状と予定を簡単に記す。PE, SU (Switching Unit: 相互結合網の要素スイッチ）は、ともに最初の版のチップが完成・到着し、現在、テストベッド上で動作試験を行なっている。図5に本テストベッド（テストベッド１）の全体を、図6にそのプロセッサ基板を示す。本テストベッドは８ＰＥシステムである。本年度は、６４ＰＥ版のテストベッド２を試作し、来年度に１０２４ＰＥよりなるRWC-1を製作する予定である。

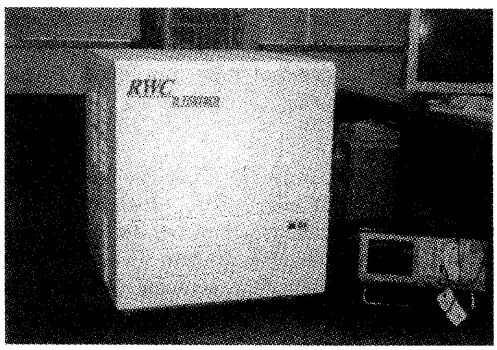


図5: テストベッド１.

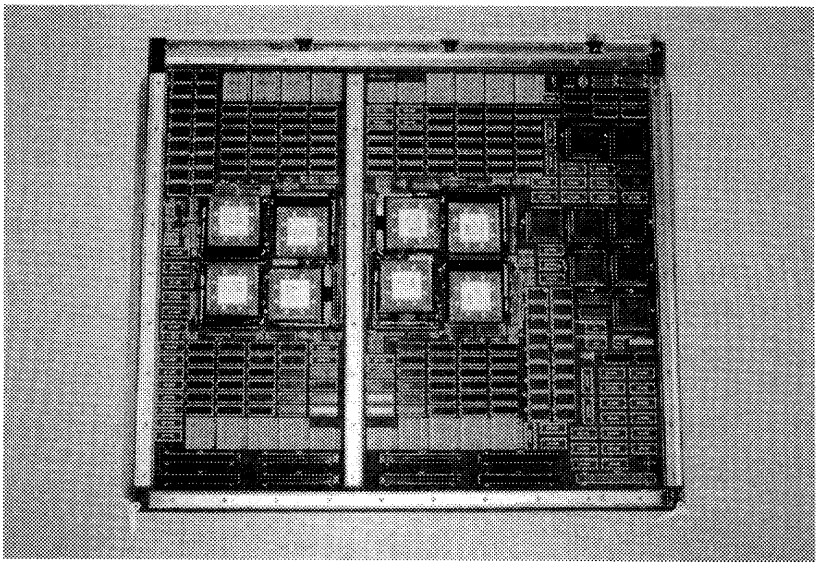


図 6: テストベッド 1 のプロセッサ基板.

謝辞

本研究を遂行するにあたりご指導・御討論いただいた RWC 島田潤一研究所長、RWC 超並列三洋研究室ならびに TRC 超並列ソフトウェア研究室の諸氏に感謝いたします。

参考文献

[1] Shuichi Sakai, Hiroshi Matsuoka, Kazuaki Okamoto, Takashi Yokota, Hideo Hirono, Yuetsu Kodama and Mitsuhisa Sato, RWC-1 Massively Parallel Architecture, High Performance Computing Conference '94, pp.33 - 38 (1994).

[2] Shuichi Sakai, Kazuaki Okamoto, Yuetsu Kodama and Mitsuhisa Sato, Reduced Interprocessor- Communication Architecture for Suporting Programming Models, Proceedings of Massively Parallel Programming Models, pp. 134 - 143 (1993).

[3] 松岡浩司、岡本一晃、廣野英雄、横田隆史、坂井修一、超並列計算機 RWC-1 用プロセッサチップの設計、CPSY95-18 (1995).

[4] 岡本一晃、松岡浩司、廣野英雄、横田隆史、坂井修一、超並列計算機 RWC-1 の命令セットアーキテクチャ、CPSY95-36 (1995).

[5] 松岡浩司、岡本一晃、廣野英雄、横田隆史、坂井修一、RWC-1 における多レベル並列処理、ARCH113-25 (1995).

[6] 岡本一晃、松岡浩司、横田隆史、廣野英雄、坂井修一、RWC-1 のマルチスレッド処理機構、ARCH113-26 (1995).

[7] 横田隆史、松岡浩司、岡本一晃、廣野英雄、坂井修一、RWC-1 の階層型 MDCE 網、ARCH113-11 (1995).

[8] 廣野英雄、松岡浩司、岡本一晃、横田隆史、坂井修一、RWC-1 の入出力リングバス、ARCH113-16 (1995).

[9] 大西一正、北村徹、大上靖弘、清水雅久、分散独立型入出力システムのための結合網の構成と評価、ARCH111-6 (1995).

[10] 大上靖弘、北村徹、大西一正、清水雅久、並列二次記憶における動的アクセス分散による負荷の平滑化、ARCH112-1 (1995).