計算機アーキテクチャ 73－3
（1988. 10. 31, 11. 1）

並列処理システム－晴－の低レベルソフトウェア開発環境

神舘淳　安江俊明　萩原孝　村岡洋一

早稲田大学　理工学部

本稿ではマルチプロセッサシステム－晴－の機械語レベルプログラムを開発する際の環境について述べる。－晴－の要素プロセッサはデータアーキテクチャを採用しており、従って、その機械語はデータフローグラフそのものである。データフローグラフの概念は図を用いた表現との相性が最も良く、ユーザにとって、－晴－の機械語プログラムを文字だけを頼りに開発することは極めて困難である。今回紹介する開発環境は図的インターフェースを核としたシステムで、ユーザに明快な情報視認性と簡便な操作性とを提供している。

Lower Level Software Developing Environment of a Parallel Processing System - Harray -

(in Japanese)

Jun KOHDATE, Toshiaki YASUE, Takashi HAGIWARA, Yoich MURAOKA

School of Science and Engineering, Waseda University

3-4-1 Okubo, Shinjuku, Tokyo, 169, Japan

The purpose of this paper is to describe the software developing environment of the parallel processing system -Harray-. Each processing element of the -Harray- is constructed in dataflow architecture. Therefore the machine language of this system is a dataflow graph itself. As the idea of dataflow graphs is very congenial to graphical image, programing only with literal informations is quite difficult. The environment we introduce here provides users with well arranged informations and convenient handling.

1 はじめに

－晴－は科学技術計算の高速実行を目的としたマルチプロセッサシステムであり、当初より以下の２点を基本方針としている【1】。

・高級言語としてFORTRANの使用を前提とする。
・データフローアーキテクチャを採用する。

第一点は、処理の対象が科学技術計算であることから、これまでに蓄積されたソフトウェア資産の量を考慮したものである。また第二点に挙げたデータフローアーキテクチャは、スカラ演算に対しても並列に演算を実行することが可能な点や、問題に内在する並列性を自然に表現できる点などを評価して採用した。

このようなアプローチをとる場合、コントロールフローを多用するFORTRAN言語と、これを全く用いない純粋なデータフローマシンとでは、両者間に横たわるセマンティックギャップがきわめて大きい。そのため、これを埋める何らかの対応策が必要になる。我々はこの問題に対する解答としてＣＤフロー方式を提案し、既にその有効性を確認している【2】。ＣＤフロー方式とは、プログラムをマクロブロックと呼ぶいくつかのタスク群に分割し、マクロブロック内ではデータフロー実行をし、マクロブロック間はコントロールフローを用いて制御する方式である。

このようなデータフロー方式を用いたマルチプロセッサシステムでは、既存のツールのみを使ってのソフトウェア開発は極めて困難なものとなる。これは主に次の２つの理由による。

（１）－晴－でのプログラムデバッギングに応用できる方式が存在しない。
（２）データフローグラフによるプログラミングでは、既存の文字情報を主体としたユーザインターフェースでは不十分である。

第一の問題点については、データフロー計算機でのデバッグ方式が既にいくつか報告されている【3】【4】。しかし、これらは関数型言語の使用を前提としたものであり、非関数型言語であるFORTRANを対象とした－晴－システムでは用いることができない。さらにこれらの方式は、実行履歴を解析して行うものであるため、特定のハードウェアに強く依存し、汎用性に欠ける。そこで－晴－のプログラム検証には、異なるハードウェアに対しても応用が可能なトレース実行方式によるデバッガを開発し、これを用いることとした。

第二の問題点に対しては、ユーザインターフェースに図的表現を取り入れることで対処した。図的表現が良好なユーザインターフェース実現に有効であることは既に広く知られている。今回開発したツールの中では、エディタ及びデバッガでこれを積極的に取り入れている。

本稿では－晴－の機械語レベルでのソフトウェア開発環境を構成するツール群について述べる。まず、第２章で本開発環境の概要を述べる。続いて第３章ではアセンブラについて、第４章では図的環境をサポートするグラフィックマネジャと、その機能を用いたエディタについて説明する。第５章では前述のデバッガについて、その機能を詳しく述べる。

2 ソフトウェア開発環境の概要

図１に－晴－のソフトウェア開発環境の概要を示す。同図に示されているFORTRANコンパイラは現時点では開発途上にあり、また、－晴－の実機部分はModula-2で記述された機能レベルのソフトウェアシミュレータ、ＨＳＳＶ２（Harray System Simulater Version 2）が肩代りしている。これらのツール群はいずれもUNIXワークステーション上で稼働しており、現在の主力

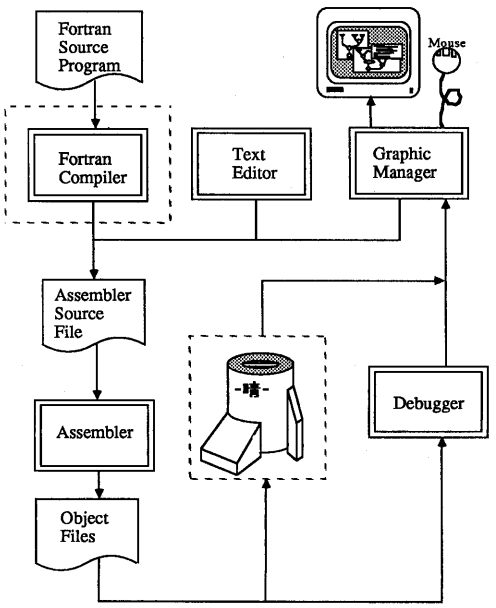

図1：－晴－の低レベルソフトウェア開発環境

開発言語は統合化アセンブラＨＩＡＳ（Harray Integrated Assembler）である。ユーザは通常のテキストエディタあるいは後述するフローグラフエディタでソースプログラムをコーディングし、それをＨＩＡＳに入力することによってオブジェクトモジュール群を得る。これらのモジュール群は直接－晴－のシミュレータ上で実行させることが可能であるが、シミュレータはハードウェアの研究を主目的としたものであるため、プログラムのデバッグ用には最低限の機能しか保持していない。また、将来－晴－の実機上でプログラムを実行させる場合には、デバッグ機能はさらに制限されたものになると予想される。そこで本開発システムでは高機能のデバッガを用意して、ユーザにデータフローグラフの論理的な誤りを発見する手段を提供している。

－晴－に限らず並列処理システムでのプログラム開発・デバッグには図的な表現方式が有効である。特にデータフローマシンはデータフローグラフという極めて図的な概念に基づいたシステムであるため、この傾向は一層顕著となる【5】【6】。この点に注目し、本開発環境ではユーザに簡便な操作性と良好な情報視認性を与えるための図的インターフェースを提供している。

3 アセンブラ

前述の通り－晴－はＣＤフロー方式を採用しているため、そのプログラムにはコントロールフローを用いてマクロブロック間の制御を行うものとデータフローを用いてマクロブロック内の制御を行うものとの２種類が必要である。そのほか、プログラムの初期データやマクロブロックの動的管理を可能にするためのテーブル等も含めると、－晴－で１つのプログラムを実行させるためには合計６つのファイルが必要になる（表１）。

```
program sum_1_10 {
        1 start 100 2;
        2 mb MB0 0 3;
        3 end;
}

assign {
        MB0 0 0;
}

gmdata {
        * nothing
}

dmdata {
        for MB0 {
                0x10010 9/l stc;
                0 2/l; * s0
                1 1/s; * x0
                11 8/r stc;
        }
}

pmprog {
        macro MB0 {
                1 cp 2/r 3/s;
                2 add 4/s;
                3 inc 5/s;
                4 cinc 6/l;
                5 cinc 7/l 8/l;
                6 tf 9/r 0/d 2/l 0/d;
                7 f 1/s;
                8 eq 7/r 6/r;
                9 ssv 10/s;
                10 fin;
        }
}
```

図２：ＨＩＡＳの記述例

図２はＨＩＡＳでプログラムを記述した例である。ソースファイルは５つの大きなブロックから成り、これらは表１に示した６つの実行形式ファイルのうち、ＰＭＴａｂｌｅを除いた５つのファイルの内容に相当する。ＰＭＴａｂｌｅファイルはアセンブラが計算により作成することができるので、ユーザが記述する必要はない。ＨＩＡＳはこのソースファイルからＨＳＳＶ２でのプログラム実行に必要な全ての実行形式ファイルを生成する。その際、行き先のないノードや入力

表１　－晴－の実行形式ファイル

ファイル名	内　　容
ＧＣＰｒｏｇ	マクロブロック間の制御プログラム
ＭＢＡｓｓｉｇｎ	マクロブロック／ＰＥの初期割当テーブル
ＧＭＤａｔａ	共有メモリの初期データ
ＤＭＤａｔａ	データフロープログラムの初期パケット
ＰＭＰｒｏｇ	マクロブロック内のデータフロープログラム
ＰＭＴａｂｌｅ	マクロブロック内プログラムの格納番地

アークが定義されていないノードなど、データフローグラフを不完全にする致命的な誤りが発見されると、それらはプログラマに報告される。

4 グラフィックマネジャ／エディタ

グラフィックマネジャは本システムでのユーザインターフェースの中核を担うものであり、ユーザがデータフローグラフの編集を行ったりデバッガの情報提示を図的に行う場合の処理を担当する。マネジャにはエディットモードとディスプレイモードの２つのモードが存在し、ユーザは処理の内容に応じてこれらのモードを使い分けることになる。各モードでのマネジャの動作は表２に示すようになっており、本節ではエディットモードについて詳細を述べ、ディスプレイモードについては５．７節で詳説する。

ＨＩＡＳのソースファイルは通常のテキストエディタを用いて作成することも可能であるが、データフロープログラムをコーディングする場合には（文字通り）グラフィカルな表現・操作を用いることのできるグラフィックエディタの有無により、その開発効率が大きく左右される。

－晴－のエディタは、要求される機能の大部分がグラフィックマネジャ内に含まれており、データにも共通する部分が多いので、独立したツールとはせず、マネジャのエディットモードとして実現している。

編集対象となるデータフローグラフをユーザがマネジャに指定すると、マネジャは、そのデータフローグラフの初期状態を画面に表示し（図３）、以後キーボードあるいはマウスを通じて送られて来る入力情報に基づいて内部のデータと表示画面を更新していく。ユーザはグラフィックマネジャが表示したデータフローグラフを見ながら編集作業を進め、作業の終了とともにＨＩＡＳソースファイルを得ることができる。

表２　グラフィックマネジャの動作モード

モード	動　作　内　容
エディタモード	表示中のフローグラフに対し変更を許す。モードの終了時にはＨＩＡＳのソースファイルを得ることができる。
ディスプレイモード	表示中のフローグラフに対する変更は許されない。デバッガから送られる情報を表示する際に用いられる。

5 デバッガ

5．1 基本的立場

前出の表１に示したように、－晴－のプログラムにはマクロブロック間の制御を行うＧＣＰｒｏｇと、マクロブロック内のデータ依存関係を記述したＰＭＰｒｏｇとの２種類が存在する。－晴－で採用しているマクロブロック化手法は、とかく難しいと言われているデータフロープログラムの挙動管理を容易にすることを目的としたものであるため、マクロブロック間プログラムのデバッグはさほど困難ではない。問題となるデータフロープログラムのデバッグ方式に関しては、従来よりいくつかの手法が紹介されているが【３】【４】、これらは実際にプログラムを実行させた上で履歴情報などを解析するものであり、プログラムの正当性はそれが実行されたマシンにおいてのみ保証されていた。我々はプログラムの正当性はハードウェアとは独立に検証されるべきとの立場に立っており、そのためのツールが以下に述べるＨＤ³（Harray Data Dependency Debugger）である。

5．2 機能の概要

本節で述べるデバッガＨＤ³はマクロブロック内のデータフロープログラム検証用であり、現時点ではマ

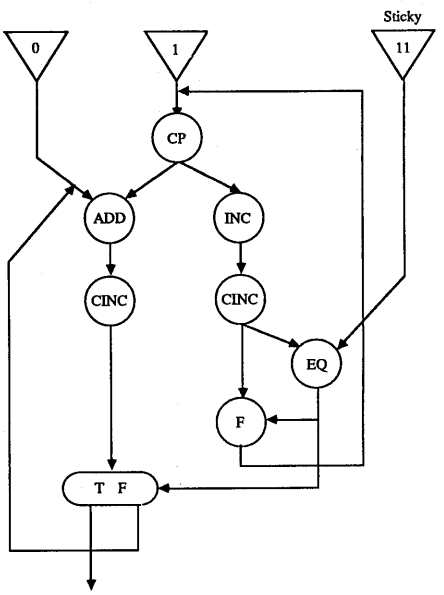

図3：初期状態のデータフローグラフ

クロブロック間プログラムはサポートしていない。HD³の主な目的はデータフローグラフに表わされたデータ依存関係をトレース実行によって解析することで、実行履歴の保存、各ノードに対するブレークポイントの設定、パケットの編集機能などを備えている。

またHD³は、フロー制御ノードへの入力を監視してTRUE／FALSEの割合を報告する機能と、プログラムの並列度に関する情報を収集する機能とを持っている。これらの機能で得られた情報は、コンパイラや、これに付随するスケジューラを作成する際に用いられる。

5.3 HD³の特徴

HD³が検証時に用いるアルゴリズムは、－晴－だけでなく、他のデータフローマシン用に書かれたプログラムに対しても応用が可能である。すなわち本デバッガは、ターゲットマシンに依存する部分を極力排除し、ハードウェアに依らない、プログラム自身の正当性を検証する仕様としてあるため、デバッガ内のデータを僅かに変更するだけで他のマシン用のプログラム†をデバッグすることも可能である。また、－晴－のマクロブロック方式に対応できるよう、一つのデータフローグラフをいくつかの部分グラフに分割して処理する考え方をとっている。この方式は複数のノードで構成される部分グラフを、あたかも高機能を持った1つのノードであるかのように扱うことを可能にするため、巨大で複雑なプログラムを階層的にデバッグすることができる。

5.4 トレース実行とブレークポイントの動作

トレース実行はHD³の最も基本的な機能であり、ハードウェア資源の量や性質（ALUの数、演算実行に要する時間等）とは無関係に、ある時点で発火可能なノードは全て発火させて次のノードに結果パケットを送出するという動作を続けていく。ブレークポイントは既存の逐次形言語用デバッガに用意されているものと同様の機能であるが、並列プログラムでは一時に多数の演算が実行されるので、必要な情報を過不足なく入手するためには、表示すべき情報の選択する際の条件指定を、細かく、かつ容易に行えるようにしなければならない。HD³でのブレークポイント設定は、「どこで（ノード指定）」、「何が起こったときに（発動条件）」、「何をするか（応答）」といった指定を表3に示す各条件の組合せによって行う。

例えば、

「ノード番号2、3またはカラー操作を行うノード（表3の1.1と1.2の条件）」
で、
「発火、または新たなパケットが到着した（同じく2.1、2.2）」
とき、
「ノード上の全パケットの情報を表示してユーザの入力を待て（3.1と3.4）」

といった設定の組合せを複数指定できるので、きわめて的確なデバッグ情報を得ることができる。

5.5 履歴管理機能

HD³はトレース実行中にトークンが流れることによって生じるデータフローグラフ内の状態変化を、常に内部に保存している。データフローグラフの実行状況を監視中に予期せぬパケットが生成されたことを発見したユーザは、必要に応じてブレークポイントを再設定し、その後この機能を活用してプログラムが停止した時点から前の状態に遡り、エラーの原因となったノードを突き止めることができる。

5.6 パケット編集機能

パケット編集機能とは、ブレークポイントに達して動作を中断したデータフローグラフから任意のパケットを取り出して、その内容を変更できるようにしたものである。この機能の付加によって、いちいちエディタに立ち帰ることなしにデバッグ作業を継続すること

表3 ブレークポイントの設定

指定項目	指定条件	
ノード指定	・ノード番号による指定	(1.1)
	・ノード番号と入力アークを指定	(1.2)
	・演算による指定	(1.3)
	・演算の種類による指定	(1.4)
発動条件	・発火したとき	(2.1)
	・新たなパケットが到着したとき	(2.2)
	・トレース実行の各ステップ毎に	(2.3)
応答	・全パケットの情報を表示	(3.1)
	・発火に関係したパケットの情報のみを表示	(3.2)
	・表示後トレース継続	(3.3)
	・表示後ユーザの入力待ち	(3.4)

† ここで言うプログラムとは動的データフローに基づいたもので、タグ情報としてカラーのみを用いるものを想定している。

が可能になる。

5.7 表示機能

ＨＤ^{3}は単独で動作することも可能だが、この場合のデバッグ情報はテキスト形式でしか表示されず、了解性には劣ると言わざるを得ない。本来このデバッガはグラフィックマネジャの併用を前提として設計されたツールであり、このときに最高の性能を発揮するようになっている。ＨＤ^{3}からの情報を表示中のグラフィックマネジャはディスプレイモードで動作しており、各ウィンドウに対するマウスからの情報はノードに対するブレークポイント設定などに用いられる。デバッガからの情報は、画面上に図式として表現されているデータフローグラフ上で、関連するノードやアークのすぐそばに表示されるため、情報視認性が格段に向上する。

図４はデバッガをクライアントとして動作中のグラフィックマネジャが表示中の画面であり、ここでは浮動小数点加算を行うノードが発火した様子と、このノードの入力アーク上でマッチングを待っているパケットの内容が示されている。

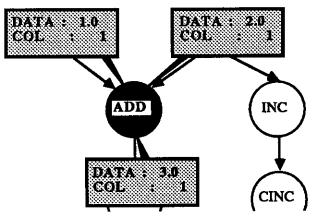

図4：ノードが発火したデータフローグラフ

6 おわりに

－晴－の低レベルソフトウェア開発を支援する開発環境について述べた。ソフトウェア危機が叫ばれている今日、プログラム開発の効率化を目指した研究が必要であることは、並列計算機あるいはデータフロー計算機の分野においても例外ではない。特にデータフローマシンのデバッグ方式に関しては、未だ標準となり得るだけの汎用性を持った方式が確立されていないのが現状であり、この方面での研究が急務である。

今後はFORTRANコンパイラの開発を進める一方、今回紹介した開発環境の評価を通じて、ユーザインターフェース、デバッグ方式などを改良していく予定である。

謝辞

本研究の遂行にあたり御討論いただいた村岡研究室諸氏に感謝します。

参考文献

【1】丸島他：”並列処理システム－晴－の実行方式”，情報研報，88-CA-69，pp.9-16（1988）

【2】Yamana, H. et al. : Syetem Architecture of Parallel Processing System -Harray-, Proc of Int. Conf. on Supercomputing, pp.76-89, (1988)

【3】高橋他：並列処理環境における関数型プログラムのデバッグ方式，情処論文誌，Vol.27，No.4，Apr. 1986

【4】島田他：”科学技術計算用データフロースーパコンピュータ SINGMA-1”，電子情報通信学会並列処理コンピュータアーキテクチャ講習会（1988.2）

【5】Davis, A.L. and Keller, R.M. : Data Flow Program Graphs, IEEE Computer, Vol.15, No.2, pp. 26-41 (1982)

【6】Nishikawa, H., Asada, K. and Terada, H. : A Decentralized Controlled Multi-Processor System, Proc. 3rd Int. Conf. on Distributed Computing Systems, pp. 639-644 (1982)