

JAERI-Review

97-005



並列プログラミング支援環境の現状と動向

1997年3月

武宮 博・樋口健二・本間一朗・太田浩史・川崎琢治
今村俊幸・小出 洋・秋元正幸

日本原子力研究所
Japan Atomic Energy Research Institute

本レポートは、日本原子力研究所が不定期に公刊している研究報告書です。
入手の問合せは、日本原子力研究所研究情報部研究情報課（〒319-11 茨城県那珂郡東海村）あて、お申し越しください。なお、このほかに財団法人原子力弘済会資料センター（〒319-11 茨城県那珂郡東海村日本原子力研究所内）で複写による実費頒布をおこなっております。

This report is issued irregularly.

Inquiries about availability of the reports should be addressed to Research Information Division, Department of Intellectual Resources, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken 319-11, Japan.

© Japan Atomic Energy Research Institute, 1997

編集兼発行 日本原子力研究所
印 刷 株原子力資料サービス

並列プログラミング支援環境の現状と動向

日本原子力研究所計算科学技術推進センター

武宮 博・樋口 健二・本間 一朗・太田 浩史
川崎 琢治・今村 俊幸・小出 洋・秋元 正幸

(1997年2月10日受理)

本報告書は、並列プログラミングのためのソフトウェア・ツールに関する有用な情報を提供することを意図したものである。具体的には、日本原子力研究所に設置されている6社の並列計算機、即ち、富士通 VPP300／500、日電 SX-4、日立 SR2201、クレイ T94、IBM SP、インテル Paragon における並列プログラミング環境の調査及び並列言語・コンパイラ、デバッガ、性能評価ツール、統合化ツールなどの並列プログラミングのためのソフトウェア・ツールに関する研究開発の現状調査の報告である。この調査はプログラミング環境開発プロジェクトの一環として行われたものである。現在原研で開発中であるこの環境に関しても述べる。このプログラミング環境は、プログラマの途切れのない思考を支援するという概念に基づいて設計されている。

Survey on Present Status and Trend of Parallel Programming Environments

Hiroshi TAKEMIYA, Kenji HIGUCHI, Ichiro HONMA

Hirofumi OHTA, Takuji KAWASAKI, Toshiyuki IMAMURA

Hiroshi KOIDE and Masayuki AKIMOTO

Center for Promotion of Computational Science and Engineering

Japan Atomic Energy Research Institute

Nakameguro, Meguro-ku, Tokyo-to

(Received February 10, 1997)

This report intends to provide useful information on software tools for parallel programming through the survey on parallel programming environments of the following six parallel computers, Fujitsu VPP300/500, NEC SX-4, Hitachi SR2201, Cray T94, IBM SP, and Intel Paragon, all of which are installed at Japan Atomic Energy Research Institute (JAERI), moreover, the present status of R&D's on parallel softwares of parallel languages, compilers, debuggers, performance evaluation tools, and integrated tools is reported. This survey has been made as a part of our project of developing a basic software for parallel programming environment, which is designed on the concept of STA (Seamless Thinking Aid to programmers).

Keywords: Parallel Computer, Software Tools for Parallel Programming, Compilers, Debuggers, Performance Evaluation Tools, Integrated Tools, Seamless Thinking Aid

目 次

1. はじめに	1
2. 主要並列計算機における並列プログラミング環境	4
2.1 はじめに	4
2.2 調査方針	6
2.3 Fujitsu VPP300/VPP500 における並列プログラミング環境	12
2.4 NEC SX-4 における並列プログラミング環境	17
2.5 Hitach SR2201 における並列プログラミング環境	24
2.6 Cray T94 における並列プログラミング環境	33
2.7 IBM SP における並列プログラミング環境	42
2.8 Intel Paragon XP における並列プログラミング環境	49
2.9 各社並列プログラミング環境のまとめ	55
3. 並列プログラム開発支援ツールの研究動向	60
3.1 並列処理言語及びコンパイラ	60
3.2 デバッガ	71
3.3 性能評価／チューニング支援ツール	75
3.4 ツール統合型プログラミング環境	82
4. 並列プログラム開発を支援する STA 環境	92
4.1 途切れのない思考の支援	92
4.2 STA 環境	94
5. おわりに	104
謝 辞	105

Contents

1. Introduction	1
2. Parallel Programming Environment on the Existing Parallel Computers	4
2.1 Introduction	4
2.2 Survey Guidelines	6
2.3 Parallel Programming Environment on Fujitsu VPP300/VPP500	12
2.4 Parallel Programming Environment on NEC SX-4	17
2.5 Parallel Programming Environment on Hitachi SR2201	24
2.6 Parallel Programming Environment on Cray T94	33
2.7 Parallel Programming Environment on IBM SP	42
2.8 Parallel Programming Environment on Intel Paragon	49
2.9 Summary of Parallel Programming Environments	55
3. Present Status and Trend of Parallel Programming Support Tools	60
3.1 Parallel Programming Language and Compiler	60
3.2 Parallel Debugger	71
3.3 Performance Evaluation/Tuning Tools	75
3.4 Tool-integrated Programming Environments	82
4. STA Environment Supporting the Development of Parallel Programming	92
4.1 Seamless Thinking Aid	92
4.2 STA Environment	94
5. Concluding Remarks	104
Acknowledgments	105

1. はじめに

80年代後半から、種々のタイプの並列計算機が商品として出荷されている。即ち、現実世界の問題解決、主として自然科学、工学、薬学、エコロジー等の分野での問題解決に利用される計算機に対して、不断に要求されている一層の高性能、一層の低成本及び一層の生産性向上を実現するため、並列処理は今や可能な要めの技術として出現したものと見なすことができる。

一層の高性能化の観点からみれば、従来からの単一のプロセッサの高速化を押し進めるより、高度なネットワーク通信技術、高度なメモリ実装技術などを活用し、適度な性能のプロセッサを多数結合した並列処理によって大幅な高速化が達成される見込みがあること、コスト低減化の観点からも、並列計算機を構成するプロセッサを、ベクトル・プロセッサなどの高価な単一プロセッサの計算機に比較して低成本なミニ・スーパー等で構成することが可能であること、さらにパソコンやワークステーションなどにも同じく用いられ、急速な発展を遂げているVLSI(Very Large-scale Integrated Circuit)技術の活用により、並列計算機の構成プロセッサに大量生産可能な高性能マイクロ・プロセッサを用いることにより、生産性の大幅な向上が実現できることなどが技術的な背景となっている。

70年以前では、既に1968年に世界で初めての並列計算機Illiac IV (SIMD:Single Instruction Multiple Data型、分散メモリ、32台のプロセッサをリング状に接続)が米国に於いて、数値流体力学計算の高速化のために試作開発されたが、当時のスーパーコンピュータCDC7600と比較して、プログラミングの困難さ、価格性能比(数値流体計算でCDC7600の2倍から6倍程度の速度向上、試作品であることもあり高価)等で競合するものではなく、科学技術一般に普及するには至らなかった[1]。その後70年中頃に、ベクトル・プロセッサ(1976年にCray 1、その後富士通、日電、日立も出荷)が出現した。この計算機はアーキテクチャ上、 SIMD型と見なすことができる。即ち、アレイ・データに対して単純な並行処理を行うベクトル命令で処理(命令レベルの並列処理)されるため、このタイプの処理を多く含む科学技術計算では、従来の逐次型(SISD: Single Instruction Single Data)計算機に比較して極めて大きな速度向上率を達成し、現在まで広く利用されてきている。これらの利用・普及によって、自動ベクトル化コンパイラなどSIMD型のプログラミング・スタイルが一応確立していると言える。しかし、このアーキテクチャもいくつかの欠点をもっている。即ち、ベクトル命令は多段の流れ作業のようなバイ二ライン方式によって処理されるが、処理されるアレイ・データが十分バイ二ラインに供給される場合、逐次処理に対するバイ二ライン処理の速度向上率はバイ二ラインの段数(並列計算機のプロセッサ数に対応)に比例する。しかし、バイ二ラインの段数は実装上の制限から任意に増加させることができないこと[1]、また、アレイ・データに対して同一の命令処理をするように構造化することが困難なプログラム、即ちデータ・パラレル性のないプログラムでは、効率的な利用が期待できないことなどによって、スケーラビリティの実現が困難となる。

このような計算機の一層の高速化を狙って、80年代始めに、高速なネットワークで主記憶装置に数台のベクトル・プロセッサを接続した共有メモリ型のマルチプロセッサ(1982年にCray X-MP、その後日電、日立が出荷)が商品化された。このアーキテクチャでは、各ブ

ロセッサは独立したプログラムを実行することが可能であり、データ・パラレルに加え、タスク並列も可能な、いわゆるMIMD: Multiple Instruction Multiple Data型のプログラミング・スタイルが必要になった。共有メモリ型の計算機では、逐次処理プログラムを作成する時と同様に、主記憶装置内のグローバルなメモリ空間を各プロセッサに割り当て分割するため、プロセッサ間の通信に意を払うことなく比較的容易に並列プログラムを作成することが可能である。しかしながら、一般にスケーラビリティを実現することは困難である。何故なら、実装上の理由とメモリ競合を避けるため、主記憶装置に接続できるプロセッサ数は高々数十台が限度（32台程度）であり、これがボトルネックになってしまうためである[1]。

80年代後半から現在に至るまでの顕著な動向は、数千台ものプロセッサから構成され、スケーラビリティを指向した超並列計算機の出現である。現在出荷されているものは、均一なマルチプロセッサで構成され、例えばIntel Paragon、日立SR2201のようなものか、IBM SP2に代表されるようなワークステーションを疎結合したもので、いずれも物理的にメモリが各プロセッサに個別配置された分散メモリ型並列計算機のみが技術的に実現している。これらのアーキテクチャでは、安価なメモリを搭載し、安価で大量生産可能な標準的なチップによる多数のプロセッサがネットワーク結合されるため、結合されるプロセッサ数の増加に対応して、プロセッサ間通信速度が低下しないようないわゆるスケーラブルなネットワークが重要になる。このため、ハイパーキューブ、格子、クロスバー等多様なネットワーク・トポロジが実用に供されている。このアーキテクチャでは、共有メモリ型並列計算機のようなグローバル・メモリ空間がないため、データ及びプログラム・コードを分割して各プロセッサのメモリ空間に配分しなければならない。また、あるプロセッサのメモリ空間にあるデータを他のプロセッサが参照する場合には、プロセッサ間通信によってデータの転送が行われる。これはOS管理のもとにシステム・コールによって実行されるが、前述のデータ及びプログラム・コードの分割は、プログラム作成レベルで、概ね利用者に任された複雑な作業である。これは、このタイプの並列計算機の利用を困難にしている原因のひとつと言える。

以上、主としてハードウェア・アーキテクチャに注目して、並列計算機開発の歴史を概観したが、单一のベクトル計算機、共有メモリ並列ベクトル計算機、分散メモリ並列計算機と世代が新しくなるに従い、それらのハードウェア・アーキテクチャの複雑性に対応して、利用者の応用プログラムの新規作成、或いは既存逐次型応用プログラムの書き換えが困難になっている。また、ベクトル計算機あるいは種々の並列計算機を利用して、具体的な応用プログラムを実行した場合、理論的なピーク性能と実際に得られる実効性能のあいだにはかなりの開きがあることは既に経験しているところである。さらに、事前に性能向上を予測することも極めて困難になっている。即ち、性能へ大きく影響する個別ハードウェアの特徴、たとえばキャッシュ特性、複雑なメッセージ・プロトコル、ネットワーク・トポロジー特性などの内容を一般の利用者が理解するのは容易ではないからである。

このような現状のなかで、並列計算機をより容易に且つその性能を最大限に引き出すよう効率的に利用するための、並列処理ソフトウェア技術の開発も進展している。この技術の中核となるものは、与えられた並列計算機に応用プログラムを最適にマッピングする技術といえる。ここで最適マッピングとは、応用プログラムに内在するデータ及びタスク並列性を同定し、そ

の並列性に対応してデータを分割、さらにプログラム・コードを逐次処理タスクへ分割し、それらの処理が最短時間で終了するように、各プロセッサへ如何に配分し、処理するかをスケジューリングすることである[2]。即ち、与えられた並列計算機のハードウェア・アーキテクチャの特性を十分に理解し、各プロセッサの処理時間が可能な限り均等になること、さらにプロセッサ間の通信時間を可能な限り短縮することであり、このためには、計算アルゴリズムの設計時、プログラミング時、コンパイル時、そしてそれを実行する時に、並列性を容易に且つ十分に引き出すためのソフトウェア技術の開発が重要になる。

本来、このようなソフトウェア技術は、上述したような多様なハードウェア・アーキテクチャに個別的に対応したものでなく、これらを高度に抽象化した仮想的な並列計算機を想定し、一般性の高い技術であるべきであるが、逐次プログラムの自動並列化が未成熟技術であるように抽象化のレベルは低く[3], [4]、研究開発中のものも多い。このため、利用者が常に並列性を意識しつつプログラミングするなど、利用者への負担が大きく、この負担をより軽減するために上記の並列プログラム開発の各局面で、利用者を支援するため、GUI: Graphical User Interfaceなどヒューマン・インターフェイスに留意した各種のプログラム開発支援ツールから構成される並列プログラミング環境が提供されているのが現状である[3]。

そこで、本報告書は、現在の並列処理ソフトウェア技術が多くの改良すべき技術課題を内包していることを認識しつつ、既に出荷され実用に供されている各種並列計算機の並列プログラミング環境及びその研究開発動向をレビューし、問題点の把握と今後の技術開発に資する情報を得ることを意図したものである。具体的には、第2章においては、平成8年度現在、日本原子力研究所に設置されている6機種の商用並列計算機、即ち、Fujitsu VP300/500, NEC SX-4, Hitachi SR2201, Cray T94, IBM SP, Intel Paragonの各々に対して、ハードウェア及びソフトウェア構成を概観し、並列プログラム開発支援ツールとして、並列プログラミング言語及びコンパイラ、デバッガ、性能評価ツール、統合化ツールの機能、特徴等のレビューを述べる。第3章では、関連研究機関等で進めている並列プログラム開発支援ツールの研究動向を、第2章と同様な構成でレビューした。一方、第4章では、日本原子力研究所計算科学技術推進センターにおいて、現在進めている支援ツール開発のひとつについて、そのツール概念、即ち並列プログラム開発過程における開発者の途切れのない思考を支援するという概念(STA: Seamless Thinking Aid)に基づく統合化ツール開発の現状をのべる。最後に第5章では、簡単なまとめを試みる。

参考文献

- [1] Kai Hwang, "Advanced Computer Architecture, Parallelism, Scalability, Programmability," McGraw-Hill, New York, 1993.
- [2] Dan I. Moldovan, "Parallel Processing -From Applications to Systems-", Morgan Kaufmann Publishers, San Mateo, California, 1993.
- [3] Edited by K. M. Decker & R. M. Rehmann, "Programming Environments for Massively Distributed Systems," Working Conference of the IFIP WG 10.3, April 25-29, 1994, Published by Birkhauser Verlag, Basel, Germany.
- [4] Edited by Christoph W. Kesler, "Automatic Parallelization, New Approach to Code Generation, Data Distribution, and Performance Prediction," Verlag Vieweg, Wiesbaden, Germany, 1994.

2. 主要並列計算機における並列プログラミング環境

2.1 はじめに

計算機の社会への浸透に伴い、より大規模な計算処理への要望が高まっている一方、従来の単一計算機における計算能力の向上はハードウェアレベルにおける物理的制約のために限界に近づいているといわれている[1]。そのため多くの並列計算機が計算機ベンダーより提供されるようになってきた。日本原子力研究所においても、大規模計算要求に応えることを目的として、表2.1.1に示す12台の並列計算機を導入し、利用技術に関する研究開発を行っている。

大規模な計算処理を行うことを目的として開発された従来のベクトル計算機と比較した場合、これらの並列計算機は個々の計算機アーキテクチャが大きく異なることが特徴となっている。従来のベクトル計算機は全て、科学計算における同一計算の繰り返し構造に着目し、異なるデータに対し同一の演算命令を並列に実行するという命令レベルの並列性を実現するベクトル演算機能によって高速実行を実現していた。それに対し、現在の並列計算機には、タスクと呼ばれる大きな粒度(Granularity)の実行単位を各プロセッサに割り当て並列実行を行うことができるタスクレベルの並列性を実現するスカラ並列計算機や、ベクトル実行性とスカラ並列実行性を併せ持ったベクトル並列計算機などが存在する。また、各プロセッサにおけるデータ保持戦略も異なり、各プロセッサが独立な記憶域を持ち異なるプロセッサからのデータアクセスに対しては通信によってデータを送信する分散メモリ型のメモリアーキテクチャを採用する計算機、全プロセッサがアクセスできる大域的な記憶域を有する共有メモリ型のメモリアーキテクチャを採用する計算機が存在する。

このように多様なアーキテクチャを持つ並列計算機上で実行効率の高い並列プログラムを開発するためには、計算アルゴリズム等論理的な並列性の抽出もさることながら、ハードウェアアーキテクチャの特性を十分に認識することが重要である。

しかし、計算アルゴリズムによって論理的な並列性を抽出し、ハードウェアアーキテクチャを理解しただけで、実行効率の高い並列プログラムが開発できるわけではない。目的とするプログラムを作成した後、正常動作するように修正を行い、実行状況を調べて実行効率を低下させる要因を同定し、改善を施すという作業を、試行錯誤しながら繰り返し行うことによって初めて実行効率の高い並列プログラムが完成する。

上記の作業の流れは、逐次プログラム開発でも同様に発生するが、並列プログラム開発では個々の作業の困難さが増している。例えば、数値アルゴリズムをプログラミング言語で表現する(プログラムを作成す

表 2.1.1 日本原子力研究所における並列計算機

計算機名	プロセッサ数	開発元	導入場所
VPP300	16	富士通	中目黒
SR2201	64	日立	
T94	4	Cray	
SP2	50	IBM	
SX-4	6	NEC	
VPP500	42	富士通	東海研
Monte-4	4	* NEC	
Paragon	256	Intel	
Paragon	32	Intel	那珂研
VPP300	12	富士通	
Paragon	800	Intel	
Paragon	34	Intel	関西研

* 原研が開発、NECが製作

る)作業では、複数プロセスの制御を記述する必要がある。並列計算では各プロセスが独立に計算を進めるだけで全体の計算が終了することは稀である。個々のプロセスがタイミングよくデータを交換し、足並みを揃えて計算を行うことにより初めて全体の計算が効率よく終了する。このような並列プロセスの制御を記述する必要があるため、並列プログラムの記述は非常に複雑になる。

並列言語及びコンバイラは、この困難さを低減させる。並列言語は、並列処理を記述するための枠組み及び手段を提供し、コンバイラは、言語によって表現された処理を計算機上で効率的に実行できるよう最適化する機能を持つからである。

また、正常動作するように修正を加える作業(デバッグ作業)でも、並列プログラムを対象とした場合、逐次プログラムの場合と比較して作業の困難さが増している。並列プログラムでは、各プロセスが非同期的に作業を実行するため、常に同一の結果が保証されるとは限らないからである。常に同一の結果が保証されないと、正常動作を妨げる原因を究明するために何度も実行を繰り返す必要があり、作業効率が著しく低下する可能性がある。

並列デバッガは、この困難さを低減させる。並列デバッガは、多数のプロセスを制御することによって、繰り返し同一の結果を生成させることができだからである。

並列プログラムの開発を困難にする要因は、実行効率の改善を行う作業(チューンアップ作業)でも存在する。実行効率を低下させる原因是、従来の逐次プログラムのチューンアップ作業において考慮すべき要因に加えて、通信コスト、ロードインバランス等も存在する。これらの要因のうちどれが真の要因であるかを同定するためには、各サブルーチンの計算時間、通信量、通信パターン、キャッシュミスヒット率など各種データを吟味する必要がある。しかし並列プログラムでは、これらの情報を各プロセッサから収集する必要があるため、収集作業、解析作業にかかる手間が大きい。

性能解析ツールは、この困難さを低減する機能を持つ。性能解析ツールはユーザの希望するデータを収集し、わかりやすい形式でユーザに提供する。また収集データを分析し、そのプログラムが達成可能な並列実行効率に関する予測結果を提供することも可能である。

上記の例からわかるように、並列プログラム開発における困難さを低減するためには、各種並列プログラム開発支援ツールを有効に利用することも重要である。

以上の議論をまとめれば、今後の並列プログラム開発では、開発者にとって

1)対象並列計算機のアーキテクチャの理解

2)プログラム開発支援ツールの理解

がより重要となるということができる。

本章では、上記2点の理解を深めることを目的として、現在日本原子力研究所が導入している並列計算機に関して、ハードウェアアーキテクチャ上の特性を述べ、そのハードウェア特性を活かした並列プログラム開発を支援するプログラム開発支援ツールの現状についてまとめる。

2.2 調査方針

各計算機に関する具体的な記述項目は以下のとおりである。項目は大別して(1)各並列計算機のハードウェア及びソフトウェア構成を概説したシステム構成の項と(2)ソフトウェア構成の個別要素である各ツールの特徴を詳述した並列プログラム開発支援ツールの項に二分される。(1)では(i)ハードウェア構成、(ii)ソフトウェア構成について述べる。また、(2)では、(i)言語処理系、(ii)デバッガ、(iii)性能評価ツール、(iv)統合化ツールについて述べる。以下に、各項目における記述内容を示す。

(1) 各並列計算機のシステム構成

(i) ハードウェア構成

上述のように、現在の並列計算機には多様なアーキテクチャが存在する。それらは、処理特性及びメモリアーキテクチャの観点から以下のように分類される。

(イ) 処理特性の観点から

- ・スカラ並列処理

WS等で利用されているスカラプロセッサを複数結合することにより、スカラ計算を並列に実行する。

- ・ベクトル並列処理

従来のベクトルプロセッサを複数結合することにより、ベクトル計算を並列に実行する。

(ロ) メモリアーキテクチャの観点から

- ・分散メモリ型

各プロセッサが独自の記憶域を持ち、他プロセッサの所有するデータへのアクセスは、プロセッサ間で通信を行う事により実現する。

- ・共有メモリ型

大域的な記憶域を持ち、全プロセッサがその記憶域にアクセスする。

- ・共有分散メモリ型

プロセッサの持つメモリを階層化することによって、上記2種類の特徴を併せ持ったメモリアーキテクチャである。すなわち、プロセッサを幾つかのグループに分割し、グループ内に単一の共有メモリを配置することによりグループ内のプロセッサは共有メモリ型の特徴を持つ一方、他グループのプロセッサの所有するデータへのアクセスはプロセッサ間で通信により行うという分散メモリ型の特徴を持つ。

プロセッサ間でデータ通信を行う事が必要となる分散メモリ型アーキテクチャを採用した並列計算機は、プロセッサの結合形式であるネットワークトポロジーによりさらに細分される。主要なネットワークトポロジーとしては、例えばメッシュ結合、ハイパーキューブ結合、バス結合、ツリー結合などが存在する。

プログラム開発対象計算機がどのアーキテクチャを採用しているかにより、プログラムの並列化戦略は大きく異なるため、アーキテクチャの特性を理解することは重要である。

例えば、処理特性の観点からいえば、スカラ並列処理がプロセッサ間の並列性のみを対象と

するのに対し、ベクトル並列処理では、プロセッサ間の並列性に併せてさらにプロセッサ内のベクトル処理においても並列性が実現される。従って、ベクトル並列処理を効率的に行うためには、複数プロセッサ間の処理の並列性、および単一プロセッサ内のベクトル処理の並列性(いわゆるベクトル化)という2つの異なる並列性をプログラム上に表現する必要がある。

また、メモリアーキテクチャの観点からいえば、分散メモリ型の計算機では他プロセッサの所有するデータにアクセスする際、通信を行う必要があるため、自プロセッサの所有するデータへのアクセスと比較して時間的コストが高い。従って、分散メモリ型の計算機上で効率的な並列実行を実現するためには、プロセッサ間の通信を低減する必要がある。一方、共有メモリ型の計算機では、プロセッサ数が少數の場合、データアクセスに関するコストは分散メモリ型の計算機と比較して通信が発生しないため低く抑えられる。しかし、プロセッサ数が大きくなると、複数のプロセッサからの同一メモリへのアクセスが一般にプロセッサ数に比例して多くなるためプロセッサ間のアクセス競合が生じ、データアクセスがボトルネックとなる。そのため、共有メモリ型の計算機のプロセッサ数は現在数十程度に抑えられている。従って、共有メモリ型の計算機上で効率的な並列実行を実現するためには、少數プロセスによる並列実行に適するように並列実行単位の粒度を大きくする必要がある。

本項目では、原研に設置された各並列計算機において採用されている処理特性、ハードウェアアーキテクチャの特性について述べる。

(ii)ソフトウェア構成

一般に科学技術計算における並列プログラム開発作業は以下の3つのフェーズ

- (イ)プログラム設計フェーズ
 - (ロ)プログラム作成フェーズ
 - (ハ)結果・性能評価フェーズ
- から構成される。

並列プログラム開発作業は、上記の各フェーズを一度づつ実行すれば終了するものではない。各フェーズ内には作業のフィードバックループが存在し、さらにフェーズ間にもフィードバックループが存在する。作業者は必要に応じてこれらのループを繰り返すことにより作業を進めていく。並列化作業全体のワークフロー及び各フェーズにおいて使用するツール類を図2.2.1に示す。

プログラム設計フェーズは、計算対象となる現象をモデル化し、そのモデルを並列計算機上で計算するための手順を決定するフェーズである。このフェーズでは、以下の作業が存在する。

(イ)数学モデルの作成

計算対象となる現象を解析し、特徴を抽出すること(物理モデルの作成)によって方程式群で表現される数学モデルを作成する。

(ロ)計算アルゴリズムの検討

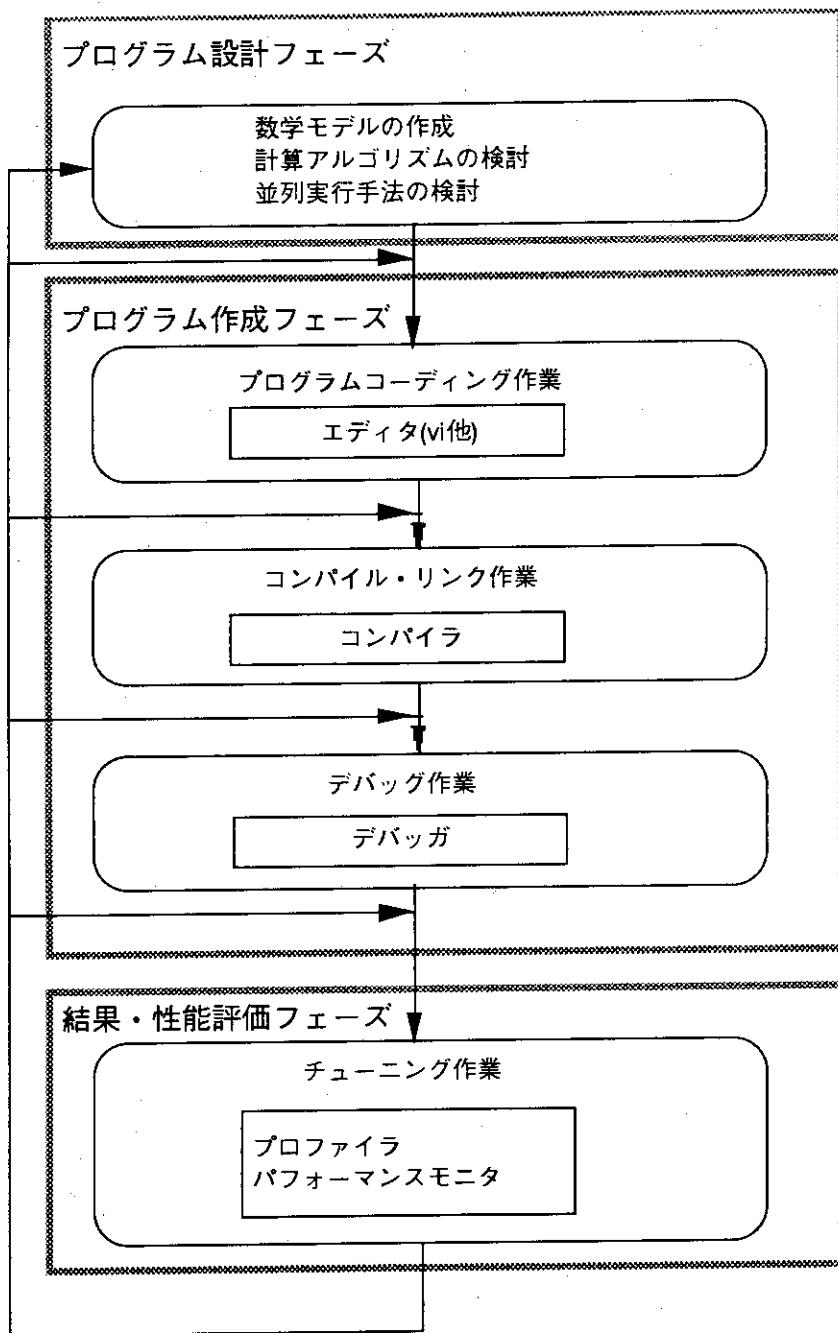


図 2.2.1 並列プログラム開発作業のワークフロー

(イ)で作成した数学モデルを計算機上に表現するための具体的な手続き表現である計算アルゴリズムを選定する。

(ハ)並列実行手法の検討

実行コストを事前評価しながら、最適な並列実行単位を決定する。並列プログラムの場合、実行コストは計算量の多寡のみならず、通信量やプロセス間のロードインバランス等にも考慮する必要がある。必要ならば、(ロ)に戻って別の計算アルゴリズムを採用する。

このフェーズの生成物である方程式群と計算アルゴリズムを指定するだけで並列プログラムを自動生成するツールも存在するが、一般的にはこのフェーズの作業は机上で行われる。

並列化の方針が明確になった後、作業者は並列化プログラムの作成、修正を行う。これがプログラム作成フェーズでの作業となる。このフェーズの作業は逐次プログラム作成時と同様である。すなわち、プログラムが動作するまでプログラムの編集、コンパイル、試走、デバッグを繰り返す。このフェーズで利用されるツールとしては、エディタ、コンパイラー、デバッガがある。

結果・性能評価フェーズでは、プログラムの正当性の確認と並列実行性能の評価を行う。プログラムの正当性の確認は、解析的にも結果が得られる現象をシミュレートし、解析結果と計算結果を比較することが一般的である。科学計算では一般に多量のデータが生成されるため、正当性の確認作業では結果を可視化し定性的な評価を行うために可視化ツールが用いられることが多い。計算結果が異なる場合は、再度プログラム作成フェーズに戻り、プログラムの修正、試走を繰り返す。

並列実行性能の評価は、プロファイラ、計算負荷・通信モニタを利用して、充分な並列実行性能が得られたかどうかを確認する。充分な並列実行性能が得られない場合は、ボトルネックとなる処理を調査し、プログラムの修正を行う。また、採用した並列実行手法や計算アルゴリズム自体が問題となる場合は、並列化方針検討フェーズまで戻って再度並列化方針を検討しなおす必要がある。

本項目では、プログラム作成フェーズ及び結果・性能評価フェーズにおいて、具体的にどのような支援ツールが各計算機に提供されているかを、プログラム開発の流れに対応させて説明する。

(2)並列プログラム開発支援ツール

(i)並列プログラミング言語及びコンパイラー

並列プログラミング言語は、処理に内在する並列性を表現するための手段である。コンパイラーは表現された並列性を認識し、実行可能なコードを生成する。従って、並列プログラミング言語及びコンパイラーは並列処理の実現において非常に重要な位置を占めている。

並列プログラミング言語は、プログラミングモデルに基づいて設計されている。プログラミングモデルとは、並列処理をどのように捉えるかという考え方の枠組みであり、並列計算機システム及びその上での並列処理を抽象化したものと捉えることができる。

これまでに様々な並列プログラミングモデルが提案されているが、科学計算においては、共有変数モデル、メッセージ・パッシング・モデル、データ・パラレル・モデル[1], [2] が代表的なモデルとして利用されている。

・共有変数モデル

共有変数モデルは共有メモリ型並列計算機を抽象化したもので、共有記憶域中の大域的なデータを複数のプロセスがアクセスすることにより並列計算が進行すると考えるモデルである。

・メッセージ・パッシング・モデル

メッセージ・パッシング・モデルは分散メモリ型並列計算機を抽象化したもので、各プロ

セスが各々局所的な記憶域を持ち、プロセス間で必要なデータを通信により授受しながら並列に計算を行うと考えるモデルである。

- ・データ・パラレル・モデル

データ・パラレル・モデルはSIMD型並列計算機を抽象化したもので、計算プロセスと制御プロセスの二種類のプロセスにより並列計算が進行すると考えるモデルである。計算プロセスは制御プロセスの制御下で互いに同期をとりながら、同一の処理を異なるデータに対し行う。制御プロセスは、計算プロセスの同期制御を行うほか、処理中の逐次部分の実行を担当する。

並列プログラミング言語は、どのプログラミングモデルに基づくかにより特徴づけることができる。

プログラミング言語を用いて作成されたプログラムを計算機が実行可能な形式に翻訳するツールがコンバイラである。従来の逐次プログラムのコンバイラは単一プロセス中の処理の依存関係を解析するだけであったが、並列プログラムのコンバイラはそれに加えて、プロセス間の処理の依存関係解析による並列性の認識を行う必要がある。並列コンバイラは並列性の認識をどの程度自動的に行うかによって特徴づけることができる。

- ・自動並列コンバイラ

コンバイラのみで並列性の抽出を行い、並列実行のためのプログラム最適化を行って、並列実行可能コードを生成する。

- ・半自動並列コンバイラ

ユーザにより挿入されたディレクティブにより並列性を認識し、自動的に並列実行のためのプログラム最適化を行って、並列実行可能コードを生成する。

- ・逐次同等コンバイラ

並列性の解析は一切行わない。また、並列実行のためのプログラム最適化も行わない。すなわち、コンバイラは逐次プログラム用コンバイラと同様の解析しか行わず、並列実行に必要な最低限の作業(並列プロセスの生成コードの挿入等)のみをおこなう。並列実行はユーザが並列実行のためのライブラリ等を利用することにより可能となる。

本項目では、並列計算機で採用されている並列プログラミング言語とプログラミングモデルとの対応付け及びコンバイラの並列性認識手法に関して述べる。

(ii) デバッグ

一般に、並列プログラムのデバッグは逐次プログラムのデバッグと比較して困難である。その原因として、以下の2つが挙げられる[3]。

(イ) 並列プロセスの制御

ユーザはデバッグを行うために、これらのプロセスに対し動作制御を行わなければならぬ。並列プログラムの実行では多数のプロセスが動作する。従って、並列プログラム用のデバッガは多数のプロセスをうまく制御できる機構を持つ必要がある。

(ロ) Probe効果の存在

並列プログラムの実行では多数のプロセスが非同期に動作する。そのためプログラムの実行順序が実行の度に変化する。Probe効果とは、デバッグ対象としているプログラムにブレークポイントを挿入したりデータを出力するという行為自体がプログラムの実行順序に影響を及ぼし、目的とするプログラムの挙動が再現できなくなる事をいう。例えば、ある変数に複数のプロセスがアクセスする際、ブレークポイントの設定の仕方によってはアクセス順序が変わることが起こりうる。従って、Probe効果のために、あるエラーの発生を再現するために何度もプログラムを実行しなければならなかったり、最悪の場合エラーが再現できなかったりする。

従って、並列デバッガは

- (イ) 多数の並列プロセスを効率的に制御するための機構を備えるべきである。
- (ロ) Probe効果を低減し、現象の再現性を高めるための機構を備えるべきである。

本項目では、並列計算機上で提供されている並列デバッガにおいて上記(イ)、(ロ)に関してどのような機構が提供されているかについて述べる。

(iii)性能評価ツール

並列計算機は多様なアーキテクチャを持ち、また並列実行性能は計算機アーキテクチャに大きく依存する。従って、一つのプログラムがどのアーキテクチャの計算機でも効率的に並列実行されることは稀であり、計算機毎に性能向上を図る必要がある。

従来の逐次プログラムの性能向上作業では、各サブルーチンの実行時間を測定し、相対的に計算時間の占める割合の大きなサブルーチンを重点的に性能向上させる手法が採られることが多かった。しかし、並列プログラムの性能向上作業では、実行時間だけではなく、プロセス間の通信コスト、プロセス間の負荷の均一化等複数のプロセス間の状態を認識し、性能向上を図る必要がある。科学計算は多数のプロセスで並列処理を行うことが一般的であるため、多数のプロセスの実行状態をユーザが容易に認識できる機構を性能評価ツールは持つ必要がある。すなわち、性能向上に関連する多数の情報の表示方法が重要である。

性能向上に直接結びつく要因は、通信回数、キャッシュヒット率、演算回数、ベクトル長などそのアーキテクチャの特徴を表すハードウェアレベルの情報が主となるが、その並列実行性能を低下させる要因がいつ、どこで発生しているかはサブルーチン、ループ等プログラムに関連づけられるソフトウェア情報に基づいて認識される。従って、性能評価においてはハードウェアレベルの情報とソフトウェアレベルの情報が密接に結びつけられる必要がある。

この問題は、ハードウェアレベルの情報をどのように収集するかというモニタリング手法と密接に関連している。モニタリング手法には、ハードウェア機構を用いたデータ収集法(ハードウェアモニタリング)とOS等ソフトウェアの機能を用いたデータ収集法(ソフトウェアモニタリング)がある。ハードウェアモニタリングはソフトウェアモニタリングと比較してデータ収集コストが低いため、収集行為自体がプログラムの実行に与える影響が低く、収集されたデータに歪みが少ないと利点を持つ。その反面、モニタ機構をハードウェアに組み込んでしまうため、モニタする情報は固定的となり、また種々のソフトウェアレベルの情報と柔軟に組み

合わせることが困難となる。ソフトウェアモニタリングは、コストが大きくなる代わりに種々の情報を柔軟にソフトウェアレベルの情報と組み合わせることが可能である。従って、モニタリング手法は性能評価ツールを特徴づける指標の一つであるといえる。

本項目では、並列計算機上で提供されている性能評価ツールに関して、どのようなモニタリング手法を採用しているか、情報をユーザが容易に認識できるようどのような機構を持っているかについて述べる。

(iv) 統合化ツール

既に述べたように、プログラム開発は幾つかのフェーズに分割される。各フェーズにおいてユーザは通常単一のツールを用いるだけでなく、幾つかのツールを組み合わせて作業を行う。例えば、プログラム作成フェーズでは、エディタを用いてプログラムを開発し、コンパイラを利用して並列コードの生成、文法エラーのチェックを行い、デバッガを用いて実行時エラーを除いていく。この作業は一度づつ行えばよいウォータフォールタイプの作業ではなく、文法エラーや実行時エラーの原因が究明されれば、再びプログラムの修正、コンパイル、デバッグという作業を行う。この時、エディタ、コンパイラ、デバッガは独立のツールではなく、作業の流れと共に互いに連携しあってユーザの作業を支援すべきである。例えば、コンパイラの発見した文法エラー個所情報がエディタに伝えられ、エディタがその個所をハイライト表示するなどしてエラー個所をユーザに認識しやすいようにすれば、ユーザの一連の作業の流れを円滑にことができる。各ツールが互いに連携し情報を交換してユーザの作業を支援できるよう一体化したシステムを構成することをツールの統合化と呼ぶ。

これまでの3項目(並列プログラミング言語及びコンパイラ、デバッガ、性能評価ツール)では並列プログラムにおいて利用される主要な個別のツールに関して述べてきたが、本項目では、統合化されたツールが提供されているかどうか、また提供されている場合、どの程度統合化されているかについて述べる。

2.3 Fujitsu VPP300/VPP500における並列プログラミング環境

2.3.1 ハードウェア構成

Fujitsu VPP300は、1台のフロントエンドPE (Processing Element) と、最大15台の計算用PE (論理ピーク性能最大2.2GFLOPSのベクトル計算機) をクロスバーネットワークによって接続し、1PEあたり1140MB/sec (送受信の同時実行が可能、片方向では570MB/sec) でデータ転送を行う、分散メモリ型ベクトル並列計算機である[4]。

各PEは、ベクトルユニット (乗算、加算／論理、除算、マスク、ロード、ストアの各パイプラインで構成) とスカラユニット (最大3命令の同時実行及び、メモリアクセス命令・浮動小数点演算命令・ベクトル命令の非同期実行が可能なLIW (Long Instruction Word)型RISCアーキテクチャを採用している) から構成され、最大2GBのメモリを搭載できる。

Fujitsu VPP500は、システム制御を行う2台のCP (Control Processor) と、演算処理を行うPE (論理ピーク性能最大1.6GFLOPSのベクトル計算機) を最大222台クロスバーネットワーク

表 2.3.1 Fujitsu VPP300/16, VPP500/42 ハードウェア諸元

	VPP300/16	VPP500/42
PE台数	16	42
論理ピークベクトル性能	25.6GFLOPS	67.2GFLOPS
主記憶容量	95GB	10.7GB
PE間転送速度	1140MB/sec	800MB/sec

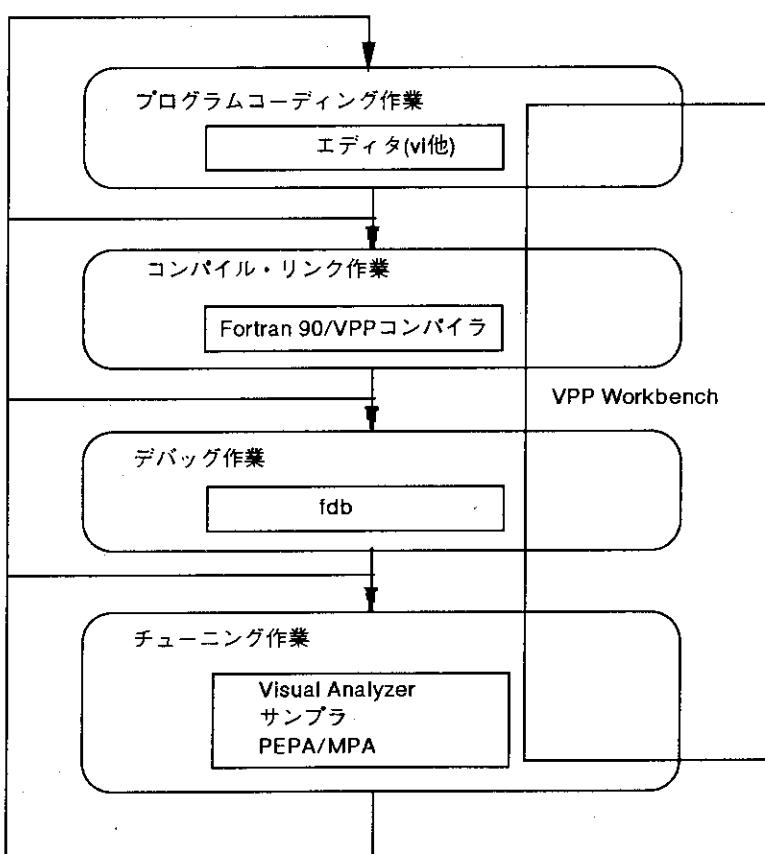


図 2.3.1 並列プログラム開発の流れと開発支援ツールの対応

によって接続し、1PEあたり800MB/sec（送受信の同時実行が可能、片方向では400MB/秒）でデータ転送を行う、分散メモリ型ベクトル並列計算機である[5]。各PEは、ベクトルユニットとスカラユニットから構成され、最大1GBのメモリを搭載できる。

日本原子力研究所に設置されたFujitsu VPP300/16及びVPP500/42のハードウェア諸元を表2.3.1に示す。

2.3.2 ソフトウェア構成

OSには、UNIX System Vリリース4を拡張したUXP/Vが搭載されており、プログラム開発を支援するツールもUXP/V上で動作する。プログラム開発作業の流れと利用するツールの対応を図2.3.1に示す。

2.3.3 並列プログラム開発支援ツール

Fujitsu VPP300及びVPP500には、各種の開発支援ツールが提供されている。以下では、最新機種であるVPP300用の並列プログラム開発支援ツールについて記述する。

2.3.3.1 並列プログラミング言語及びコンパイラ

Fujitsu VPP300にはFortranとCのコンパイラが提供されている。並列機能を持つコンパイラとしてベクトル並列コンパイラのFortran90/VPPが提供されている。また、通信ライブラリとして、FortranやCから利用できるMPI、PVM、PARAMACSが提供されている。

ベクトル並列コンパイラのFortran90/VPPは、XOCL文という並列最適化指示行（ディレクティブ）を使用することで、並列化機能を利用することができる[6]。

XOCL文の持つ機能としては、

- ・プロセッサ台数の指定
- ・バラレルリージョン（並列実行領域）の開始及び終了の指示
- ・データ分割のためのグローバル変数・配列と、分割ローカル配列の宣言
- ・処理分割（DOループのspread分割、領域のspread分割）の開始と終了の指示
- ・グローバルデータとローカルデータの間のブロック転送や、全PE上の重複ローカルデータの総和・最大値・最小値などを求めるためのデータ転送を伴う計算を行う指示
- ・同期・排他制御の指示

がある。

VPP300は分散メモリであるため、PEへのデータ分割やPE間のデータ転送の機能が必要となるが、XOCL文により実現されている。

メモリ空間は、全てのPEからアクセスできるグローバル空間と、各PE固有のメモリ空間であり、他PEからアクセスできないローカル空間の2つに分類される。この2つのメモリ空間へのデータの割り付け方として、重複ローカル・分割ローカル・グローバルの3つの形態があり、これらをデータの使用目的により使い分ける。各割り付け手法により割り付けられた配列データを、重複ローカル配列、分割ローカル配列、グローバル配列と呼ぶ。3つの配列の内、グローバル配列のみがグローバル空間におかれ、各PEよりアクセス可能である。グローバル配列を利用することにより全体を共有メモリのようにアクセスすることが可能になる。しかし、他PEからのアクセス速度は遅いため、Doループ中で参照しないというようにデータの参照位置に注意する必要がある。重複ローカル配列と分割ローカル配列は、ローカル空間におかれ。このうち、分割ローカル配列は、1つの配列を各PEで分割したもので、一般的にはグローバル配列と記憶領域を重ねて使用する。

また、各PEで実行するDOループや配列演算は、自動ベクトル化機能により高速に実行される。したがって、Fortran90/VPPを利用してプログラムを記述する場合、計算の核となるループ部分は自動ベクトル化を利用し、その外側のループや並列性のある処理をPEに分割して実行するようにXOCL文を利用することで、並列／ベクトル実行コードを生成する。

2.3.3.2 デバッガ

fdbと呼ばれる逐次プログラム用のソースレベルデバッガが、Fortran90/VP, C, C/VP向けに提供されている。並列プログラム用のデバッガは提供されていない。

fdbは、ライン入力のデバッガであり、一般のデバッガが持つ機能、即ち、プログラムの中斷・情報取得・再開機能が提供されている。さらに、ベクトル化を行った場合でも、関数入口の引数の値や、ループの出入口での変数の値のみ保証する制限下でのデバッグが可能である。この機能によりベクトル化を行った状態でデバッグを行えば、問題のサブルーチンやループを速やかに見つけ出すことが可能になる。

以上の点から考えて、VPP300におけるプログラムの開発は、プログラムのデバッグをベクトル化された状態迄に完了しておき、その後、XOCL文による並列化を行う手順となる。

並列化の作業はXOCL文による半自動並列であることから、並列化を行う前後のデータのチェック及び、異常終了時のトレース情報のチェックにより、並列デバッガなしでも、比較的容易に進められると考えられる。

2.3.3.3 性能評価ツール

Fortran, Cを対象としたアナライザ機能として、ソース解析・チューニング支援・ハードウェア情報採集の各機能が提供されている[7]。

(1) ソース解析 (Visual Analyzer)

Fortran, Cのソースプログラムの解析機能（プログラム構造、手続き／関数の呼び出しグラフ、データの参照情報）を持つ。

(2) チューニング支援 (サンプラ: ソフトウェアレベルの情報採取)

サンプラと呼ばれるプログラム実行情報を解析してチューニング情報を出力する機能をもち、タイマ割込み時に、情報をサンプラ情報ファイルに出力する。これを解析処理にかけて、チューニング情報の出力を得る。

対象となる言語は、逐次（含むベクトル）Fortran、並列Fortran、逐次（含むベクトル）Cの3言語である。

逐次Fortranでは、プログラム単位、手続き及びループ／配列式毎の負荷分布及び平均ベクトル長を求められ、チューニングが効果的な手続き／ループ／配列式を特定できる。

並列Fortranでは、プログラム単位、手続き及びループ／配列式毎の負荷分布及び平均ベクトル長と、プログラム単位及び手続き毎の並列効果情報（並列効率： $1 \leq \text{並列効率} \leq \text{プロセッサ数}$ ）及び並列化情報（並列化率： $0 \leq \text{並列化率} \leq 1$ 、並列加速度率： $0 \leq \text{並列加速度率} \leq \text{プロセッサ数}$ 、負荷バランス： $0 \leq \text{負荷バランス} \leq 1$ 、非同期転送待ち率： $0 \leq \text{非同期転送待ち率} \leq 1$ ）が求められる。並列効果情報を基にさらに並列化の実施が必要か判断でき、並列化情報により逐次実行の多い手続き、冗長実行している手続き、PE間のバランスが悪い手続き、転送終了待ちの発生する手続きを特定することができる。

逐次Cでは、関数毎の負荷分布及び平均ベクトル長が求められ、チューニングが効果的な関数を特定できる。

(3) ハードウェア情報採集 (PEPA/MPA)

環境変数の指定もしくは、サービスサブルーチンを指定して、演算情報やデータ転送に関する情報を採集できる。これらを利用することにより、MFLOPS等の性能情報や、データ転送の量、ネックとなる部分の情報を得ることができる。以下の2種類のツールがある。

(a) PEPA (PE Performance Analyzer)

PEの総平均性能、演算回数、ベクトルパイプラインの動作時間の割合等の情報を採集する。

(b) MPA (Mover Performance Analyzer)

データ転送装置であるDTU (Data Transfer Unit) 機構に関する、データ転送関連（送受信時間／送受信量／送信待ち時間等）の情報を採集する。

2.3.3.4 統合化ツール

VPP300では、UNIXワークステーションと連携してコンパイル、デバッグ、ログ表示、チューニングなどをGUIを使用して行える、VPP Workbenchという統合開発環境が提供されている[8]。以下に、機能の概要を示す。

標準のエディタとしてviが使用でき、他のエディタを指定して利用することも可能になっている。

デバッグには、MotifインターフェースでX-Window上に、ソース画面・コマンド入力行・コマンド出力領域・メニュー／コマンドボタン（fdbコマンドボタン）を配置して、fdbを使用できるようにしてある。

基本的に、ソース画面（ソースブラウザ）上で対象となるブレークポイントの設定行や、変数を選択し、fdbコマンドボタンを用いてブレークポイントの設定・解除を行い（ブレークポイントの設定・解除）、変数の値を見ることができる（変数モニタ）。又、ソース画面上には現在の停止位置を表示でき、1行毎のステップ実行や、NEXTボタンにより関数呼出しを1文として実行することもできる。変数モニタと、ブレークポイントの機能を組み合わせて、ソースブラウザで選択・登録しておいた変数の値をブレークポイント毎にみることができる。また、この画面から行単位の実行情報を収集することができ、あるブレークポイント以降の実行情報収集を行うこともできる。ただし、ソースの修正はエディタが必要となる。

チューニングの機能として、(a)サンプラ機能と、(b)ログ解析機能の2つがある。それぞれ、GUIを使用し操作性の向上を図っている。

(a) サンプラ機能

実行時の負荷の高い個所を検出するのに使用し、Fortran言語で利用が可能。一定時間毎の実行時の情報を収集し（サンプリング），後で収集結果の解析を行う。

(b) ログ解析

並列実行時の情報を収集し、時系列にグラフィック表示する。これにより、プロセッサ毎の処理時間のインバランスや、I/O待ちのネックなどの検出ができる。利用手順は、一定時間毎の実行時の情報（ログ）を収集し、後で結果の解析を行う2バス方式である。ログ

表 2.4.1 システムハードウェア緒元

CPU台数	6 (2台/ノード × 3ノード)
ノード数	3
クロック周期	2.4 nsec. (推定値)
最大ベクトル演算性能	4 GFLOPS(2 GFLOPS/CPU)
主記憶	512 MB
拡張記憶	1GB
CPU-主記憶間転送速度	32GB/SEC

表 2.4.2 SX-4 上のプログラム開発支援ツール一覧

言語処理系	FORTRAN77 及び90 C/SX及びC++
デバッガ	PDBX
性能解析ツール	ANALYZER-P/SX
エディタ	vi, emacs
統合環境	PARALLELIZER/SX

情報は、ユーザ手続き／XOCL文に対応するライブラリと時系列でのプロセッサ稼動台数累積の2種類がある。画面をクリックすることで、グラフの示している部分のソースを表示することができ、対応個所のプログラムを簡単にチェックできる。

2.4 NEC SX-4における並列プログラミング環境

2.4.1 ハードウェア構成

SX-4 [9] は、共有／分散メモリ統合型ベクトル並列計算機である。原研に設置されたシステムは、3台のSX-4/2C(ノードと呼ばれる)から構成される。SX-4/2Cは、2台のベクトル・プロセッサを持つメモリ(主記憶；512 MB, 拡張記憶；1GB)共有型の並列計算機であり、3台のSX-4/2Cは、HIPPI SWITCHで接続されている。2台のベクトル・プロセッサから構成されるSX-4/2Cの最大ベクトル演算性能は、4GFLOPS(ベクトル・プロセッサ1台の最大ベクトル演算性能は2GFLOPS)である。原研に設置されたシステムのハードウェア緒元を表2.4.1に示す。

2.4.2 ソフトウェア構成

ベクトル化あるいは並列化を支援するSX-4上のツール群を表2.4.2に示す。また、プログラム開発作業の流れと利用するツールの対応を図2.4.1に示す。

2.4.3 並列プログラム開発支援ツール

本節において、SX-4上の並列プログラミング環境(ツール)について述べる。コンパイラを除く各ツールは、複数ノードを使用した並列処理には適用できない。

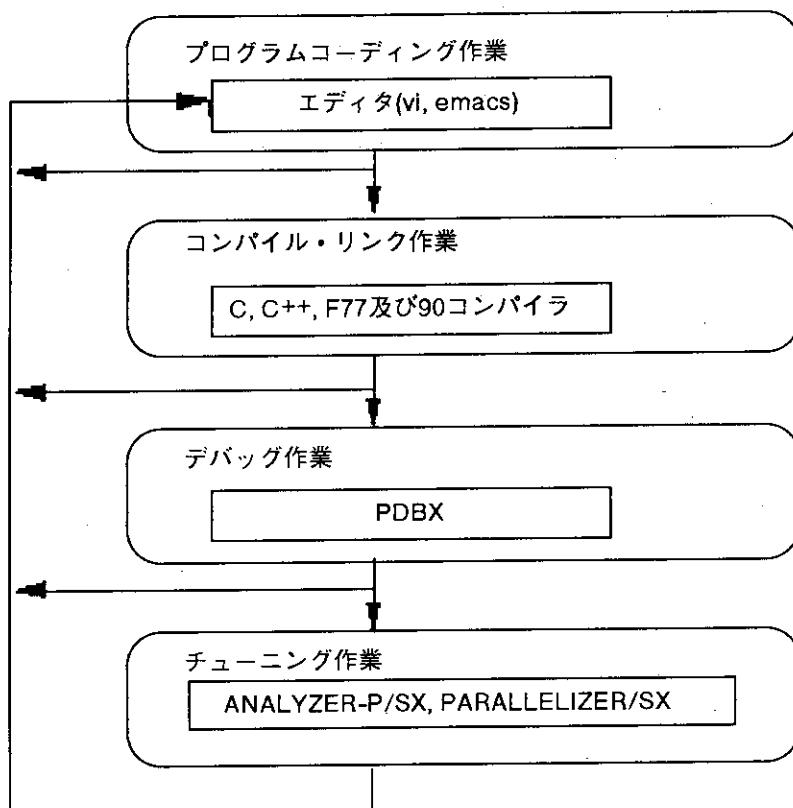


図 2.4.1 SX-4 におけるプログラム開発の流れと利用ツールの関係

2.4.3.1 並列プログラミング言語及びコンパイラ

(1) SX-4上におけるベクトル／並列プログラミング手法

前述したようにSX-4は、共有メモリ方式の並列計算機であり、ノード内で並列処理を行う場合、プロセッサ間の通信手続きを記述する必要が無いため分散メモリ方式の並列計算機と比較し、並列プログラム開発は容易である。並列処理にはマクロタスク機能またはマイクロタスク機能を用いる2つの方法がある[10]。前者がサブルーチンのような大きな計算単位の並列性に着目して並列化するのに対し、後者はDOループや文の集まりのような比較的小さな並列性に着目して並列化する。プログラムの計算コストがいくつかのDOループに集中しており、かつ、ループ長が大きい場合、マイクロタスク機能を用いた並列処理及びベクトル処理が高速化に適している。プログラムの計算コストが特定のDOループに集中していない場合は、サブルーチン単位の並列性を見いだし、マクロタスク機能を用いて並列処理を行うことで高い並列化効率を得ることができる。SX-4上での基本的なベクトル／並列化技法については次のとおり。

(a) タスクの生成と消去

タスクを生成（並列処理を開始）するための関数"PTFORK"、タスクを消去（並列処理を終了）するための関数"PTJOIN"等を、演算を制御しているサブルーチンに挿入し、マクロ・タスク機能による並列処理を実行する。

(b) グローバル・コモンとローカル・コモン

スカラ処理及びベクトル処理と比較し、並列処理では、プログラムにおけるデータの割り付

け方法に大きな違いがある。スカラ処理及びベクトル処理では、全てのデータが静的領域に割り付けられるが、並列処理は静的領域または動的領域に割り付けられる。並列処理でのデータの割り付け方法は、グローバル・コモンとローカル・コモンの2種類。すべてのプロセッサにおいて共通に使用されるデータが割り付けられるグローバル・コモンと各プロセッサにおいて個別に使用されるデータが割り付けられるローカル・コモンの混同は、当然ながら、エラーを引き起こす原因となる。

(c) 排他制御

複数のプロセスが同時に共有変数の値を変えようすると、正しい結果を得ることができない場合が発生する。このような場合は排他制御を使用して、1つのプロセスのみが共有変数を変更するようにする必要がある。例えば、排他的に処理したい部分をロック関数(PLLOCK と PLUNLOCK)で囲むことによって正しい結果を得ることができる。(ロック変数は関数 PLASGNで割り当てられる)。

(d) ベクトル／並列化指示行

*VDIRで始まる指示行を用いることでコンバイラに対し、ベクトル／並列化に必要な情報を与えることができる。例えば、*VIDR NODEPは、直後のDOループ内の配列参照に再起性の無いことを示している。

(2) 自動ベクトル／並列化機能

SX-4上のFORTRANコンバイラは、FOPP(Fortran Optimize PreProcessor)と呼ばれる自動ベクトル／並列化機能を持っている。以下、FOPPについて述べる。

(a) 機能

このツールは並列化あるいはベクトル化されたプログラムを自動生成する(より正確には、並列処理あるいはベクトル処理に最適なプログラムを自動生成する)機能を持ち、FORTRANコンバイラに組み込まれている。ユーザはコンバイラ・オプションを指定する形式で最適化機能を使用する。主な最適化については次のとおり。

(i) ベクトル化に関連した最適化

単純ループや多重ループをベクトル化に適した形式に変換する。即ち、変数・配列の定義参照関係を解析し、再帰性の無い繰り返し計算を発見し、そこにベクトル化指示行を付加する。

(ii) インライン展開

下位ルーチンを上位ルーチンに展開し、サブルーチン呼び出しのオーバヘッドを削減とともに、計算粒度を高める(ベクトル／並列処理される部分の計算量をより大きなものにする)。

(iii) 並列化に関連した最適化

ベクトル化に関連した最適化同様、変数・配列の定義参照関係を解析し、単純ループや多重ループを並列化するための指示文(*PDIR PARDO, *PDIR CRITICAL等)を挿入する。多重ループについては、基本的に内側ループをベクトル処理、外側ループを並列処理するような指示文を挿入する。

(b) 特徴

(i) 自動化による作業の効率向上

上記に「再帰性の無いことを確認し、ベクトル／並列化指示行を挿入する」と記述したが、この機能は変数／配列名あるいは明示的に記述されたインデックスから変数／配列の定義／参照関係を検出する（統語的解析による最適化）というもので、論理的に再帰性の無いことの確認は不可能である。言い換えれば、FOPPによる最適化はごく簡単なレベルである（ある水準のプログラマにとって容易な作業である）。即ち、ユーザ（プログラマ）にとって自動の意味は、単純作業の削減ということになる。大規模なコードをFOPPを用いて最適化し、ベクトル／並列処理への適応性を概観する場合などには有効と思われる。

(ii) 自動化のための戦略

FOPPはしばしば、不必要的ベクトル／並列化指示行を挿入し、性能を低下させる場合がある。ループの入替え等はある「戦略」の下に行われるが、「戦略」が失敗し、コストの大きな内側ループを外側ループに入れ換え（一般に、多重ループはコストの高いループが最内側に位置したほうがベクトル／並列処理によって性能は向上する），性能を阻害する例もあった。ループの繰り返し回数が明示的に表示されていない場合（科学技術計算においては明示的に表示されている場合のほうが多いと思われるが）にこの種の失敗が見られる。前段で述べた「統語的解析による最適化」の欠点である。本ツールはこの種の欠点を補うために、多重ループの並列化等では粒度の実行時テスト等を行っている。

2.4.3.2 デバッガ

SX-4に用意された並列プログラミング用のデバッガは、PDBX[11]である。PDBXは、dbxを並列処理用に拡張したデバッガである。dbxは、UNIX環境で動作するデバッガであり、SX-4だけでなく、SUNやHPといったワークステーションにも導入されている。

(1) 機能

dbxの主な機能については、次のとおり。

(i) ブレーク・ポイントの設定

関数の入口等予め設定したソース中の特定の場所、あるいは、ある変数の値に対する条件が成立した場合等のブレーク・ポイントでプログラムの実行を停止させ、変数の値を表示、あるいは、変更した後、プログラムをそこから実行することができる。

(ii) 実行状況の表示

指定した行や関数の実行状況を表示させることができる。

(iii) コマンドの省略形登録

頻繁に使用するコマンド（ブレーク・ポイントの設定／取消等上記機能を使用するためのコマンド）を省略形で登録することができる。

(2) 特徴

(i) コマンドが少ない

使用するコマンドの数が少ない。これは、「デバッグ作業とは、基本的に、自分の興味ある時間／場所で興味ある変数の値を探る（*）こと」という認識に起因する。即ち、本ツール

は、デバッグにおける基本作業（上記*）のみを支援するよう設計されており、（単機能のため）コマンドの数は少ない。

(ii) デバッグ情報の解析支援機能が無い

原子力コードのように大規模・複雑なプログラムの場合、ブレーク・ポイントを設定する対象やプローブする対象も大規模・複雑である。したがって、原子力コードのデバッグにおいては、単にユーザの興味ある時間／場所で興味ある変数の値を出力する機能だけでなく、出力されたデバッグ情報の解析支援機能も必要である。しかし、数値情報出力を主目的とする本ツールは、解析支援機能を持たない。

2.4.3.3 性能評価ツール

SX-4上の性能評価ツールであるANALYZER-P/SX[12]について述べる。

(1) 機能

ANALYZER-P/SXは、FORTRAN プログラムのベクトル化及び並列化に関するチューニングのための静的／動的特性を解析する性能支援向上ツールである。静的特性とは、サブルーチンの呼び出し関係、変数の参照定義関係等のプログラムの構造に関する情報（プログラムを実行しなくとも解析可能な情報）であり、動的特性とは、ある入力データを用いてプログラムを実行した際のサブルーチンやサブルーチンを構成するステートメントの実行回数やコストである。

図2.4.2に本ツールによって出力された情報の1例を示す。図2.4.2(a)は、サブルーチン毎の計算コストの一覧である。図2.4.2(b)は、あるサブルーチン内の各ループに対する計算時間、ベクトル化率等の一覧である。他社ツールと比較した場合の本ツールの特徴の一つにキャッシュ・ミス・ヒット時間の表示があげられる。キャッシュ・ミス・ヒットとはスカラ処理時に必要なデータがキャッシュ上にないことをいう。キャッシュとは、高速アクセス（主記憶と比較し、10~20倍程度の速度）が可能なメモリである。「高速」である替わりに「高価」でもあるため、そのサイズは主記憶と比較して非常に小さい。このため、データはキャッシュ上に動的に割り当てられる。当然ながら、必要なデータがキャッシュ上にない場合、主記憶上のデータを参照するため、性能は低下する。このように、キャッシュ・ミス・ヒットはスカラ処理時に起こる問題であるが、ベクトル計算機上で実行されるほとんどのジョブは、その一部（程度は別として）がスカラ処理されるため、スカラ処理性能に大きく影響するキャッシュ・ミス・ヒット情報は、性能評価において重要である。

本ツールの出力は、キャッシュ・ミス・ヒット時間を除くと、他社ツールと同様、サブルーチン等の実行回数・実行コストである。これらの情報を用いてプログラムは性能向上のためのチューニングを行う。

(2) 精度

この種のツールの評価基準の一つとして精度の問題が挙げられる。本ツールは、標準以上の精度（同じ機能を持つ他社のツールと同等以上の精度）を持っていると考える（これ以上の精度は技術的に困難と思われる）。これ以外のツールを用いて上記に示した情報を収集すること

PROGRAM UNIT SUMMARY LIST

PROG.UNIT.	ATR	FREQUENCY	EXEC COST%	VECTOR RATIO%
RACK_V	SUB	1959	45.9	94.83
CHKCEL_V	SUB	3378	29.8	99.69
ACETOT_V	SUB	1939	10.5	99.98
STARTP_V	SUB	1496	2.0	96.00
COLIDN_V	SUB	1930	2.0	91.27
HSTORY_V2	SUB	1	1.8	74.48
CHKCEL_S	SUB	647260	1.7	0.00
SOURCB_V	SUB	1496	0.8	99.51
XSEC	FUNC	3181026	0.8	0.00
ROTASZ_V	SUB	5356	0.7	99.97
TALLY_V	SUB	1832	0.5	99.85
ACEGAM_V	SUB	1930	0.4	99.83
ACECOS_FV	SUB	1930	0.3	99.97
TORUS_V	SUB	22457	0.2	92.09

図 2.4.2 (a) ANALYZER-P/SX による計算コスト出力結果 (サブルーチン単位)

PROGRAM UNIT SUMMARY LIST

PROG.UNIT	ATR.	FREQUENCY	INCLUSIVE CPU TIME(%)	EXCLUSIVE CPU TIME(%)	MOPS	MFLOPS	V.OP. RATIO	AVER. V.LEN	MEMORY LOSS	BANK CONF. (%)	CACHE MISS (%)
TRACK_V	SUB	1959	387.371(39.8)	381.087(39.1)	257.5	7.8	93.21	55.1	7.745(1)	116.984(12)	
CHKCEL_V	SUB	3378	275.946(28.3)	275.946(28.3)	271.3	8.1	97.43	57.7	10.407(1)	15.166(2)	
ACETOT_V	SUB	1939	118.208(12.1)	118.208(12.1)	292.5	6.9	98.69	60.3	7.057(1)	0.733(0)	
CHKCEL_S	SUB	647252	50.593(5.2)	47.043(4.8)	62.7	3.6	8.13	35.6	0.000(0)	6.172(1)	
COLIDN_V	SUB	1930	37.005(3.8)	25.709(2.6)	166.6	37.5	91.23	52.4	1.104(0)	12.617(1)	
HSTORY_V2	SUB	1	935.505(96.0)	19.465(2.0)	150.8	15.7	74.42	61.7	0.631(0)	6.064(1)	
XSEC	FUNC	3181026	18.516(1.9)	18.516(1.9)	69.6	36.0	0.00	0.0	0.000(0)	0.000(0)	
NEWCEL_S	SUB	1333770	55.192(5.7)	10.071(1.0)	33.0	0.0	0.00	0.0	0.000(0)	4.807(0)	
BREM	SUB	4	32.514(3.3)	8.394(0.9)	43.4	4.2	0.00	0.0	0.000(0)	4.188(0)	
STARTP_V	SUB	1496	70.787(7.3)	8.269(0.8)	265.9	6.9	91.64	59.7	0.135(0)	1.156(0)	
SURFAC_V	SUB	1770	62.684(6.4)	6.425(0.7)	61.6	0.9	73.17	58.5	0.063(0)	3.605(0)	
QPOL	FUNC	1205612	6.092(0.6)	6.092(0.6)	51.5	10.0	0.00	0.0	0.000(0)	0.248(0)	
SOURCB_V	SUB	1496	61.704(6.3)	4.504(0.5)	273.3	38.9	97.31	60.9	0.281(0)	0.804(0)	
TORUS_V	SUB	22457	6.272(0.6)	4.381(0.4)	115.9	19.2	83.42	26.7	0.199(0)	0.677(0)	
ACEGAM_V	SUB	1930	4.321(0.4)	4.115(0.4)	300.8	16.1	98.45	58.7	0.262(0)	0.067(0)	

図 2.4.2 (b) ANALYZER-P/SX による計算コスト出力結果 (ループ単位)

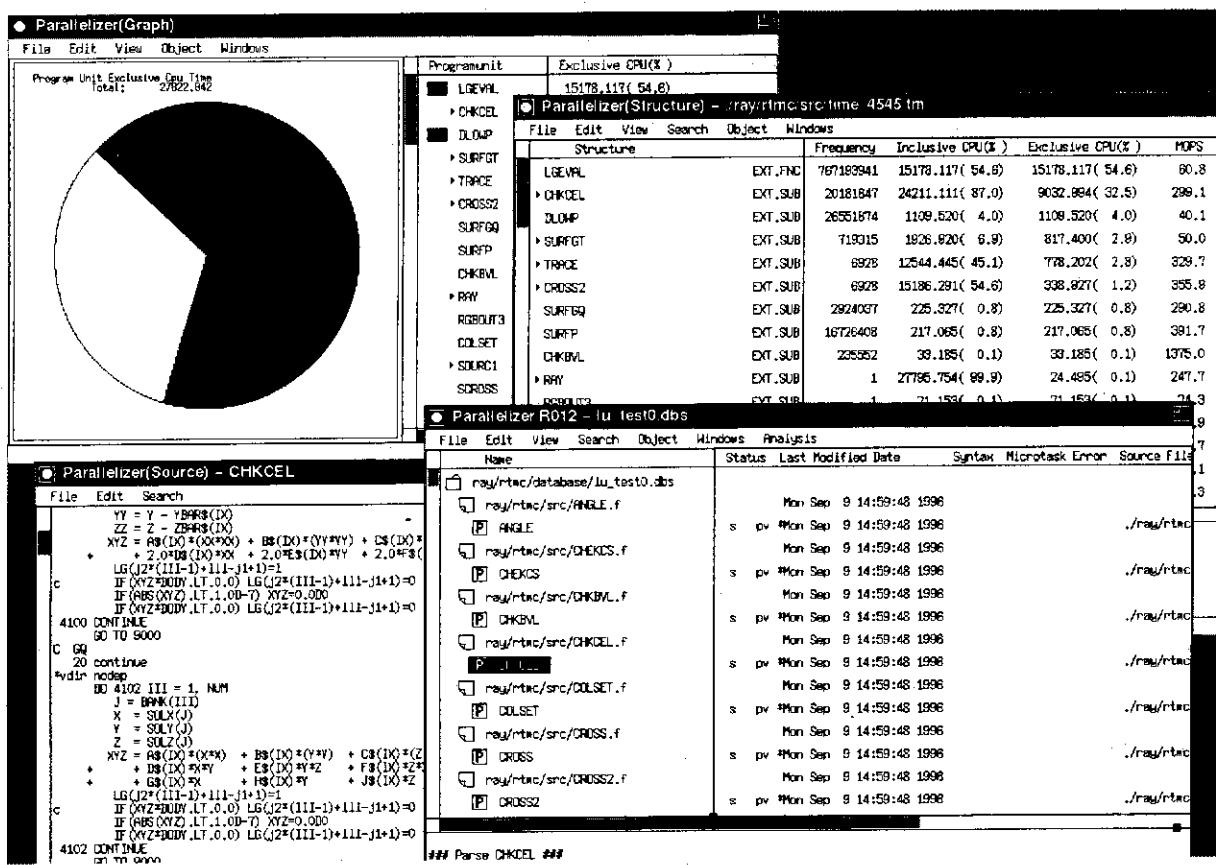


図 2.4.3 PARALLELIZER/SX 使用画面の例

はSX-4上では不可能なので正しい値は所詮把握できないが、このツールによって出力された各サブルーチンの計算コストの合計は実際に計測したコード全体の実行時間と数パーセントの誤差で一致する。この程度の誤差なら精度は十分にある、と考える。

2.4.3.4 統合化ツール

並列プログラミングのための統合化ツールであるPARALLELIZER/SX[13]について述べる。本ツールは、ANALYZER-P/SX（前節）の結果出力（プログラムの挙動解析支援）用のブラウザである。ソース修正及び構文チェックが可能（コンパイルは不可）である。プログラムのベクトル／並列化を促進し、プログラムを高速化するチューニング作業を支援する。

(1) 操作方法

本ツールは、x ウィンドウシステムを用いたマン・マシン・インターフェースを持つ。操作方法に関して次のような特徴がある。

- ・ファイル名の入力及びソース修正以外の操作はすべて、メニューバー及びマウスを用いて行う。図2.4.3に本ツールの出力画面を示す。
- ・ソースの編集エディタは、vi, emacs 等ユーザ（プログラマ）が選択できる（ウィンドウの環境変数として定義する）。

(2) 機能

(i) プログラムの構造表示

ソース表示には、3つのレベルがある。サブルーチン名、ループ番号、ステートメントである。ユーザ（プログラマ）は、必要に応じてより詳細な情報を見る。この場合、メニューバーの該当部分をクリックすることで、同じウィンドウ内にレベルに応じた情報が次々に展開されていく（expand機能）。

また、ユーザ（プログラマ）は、Trace機能を用いて、興味ある変数／配列の定義・参照箇所を知ることができる。表示については、ソース表示同様、expand機能が使用可能である。

(ii) 動的解析情報のグラフ表示

プログラムの動的解析結果については、可視化表示機能が用意されている。即ち、グラフ（円及び棒）表示とレーダー表示である。

(iii) 複数入力ファイルの比較

プログラムの挙動は入力データに依存して変化する。ユーザーがある特定の入力データを対象に性能向上のためのチューニングを進めることは稀であり、本機能を用いることで、複数の入力データに対する挙動解析を効率良く行うことが可能である。

(3) 特徴

(i) 充実したマニュアル

この種のツールとしては珍しく、マニュアルが使い易い。実際に表示されるウィンドウ及びメニューバーの図を用いて、使用方法、パラメータの意味等が明解に説明されている。

(ii) マン・マシン・インターフェースの有用性

性能評価やプログラム編集だけでなく、自動並列化ツール、また、「デバッグーコンパイル一実行」というプログラム開発に必須の繰り返し作業」にも、本ツールにおいて実現されているマン・マシン・インターフェースの実装は有効と思える。本ツール調査の折り、報告者がこのコメントを述べると、NEC側より『（当然ながら）検討中である、即ち、別ツールの開発において同様のマン・マシン・インターフェースを用いる予定』との返事があった。

2.5 Hitachi SR2201における並列プログラミング環境

2.5.1 ハードウェア構成

SR2201[14]は、分散メモリ型スカラ並列計算機である。疑似ベクトル処理機構により従来のRISCプロセッサではキャッシュに入りきらないような大規模数値計算の実行性能を向上している。プロセッサ間ネットワークに3次元クロスバネットワークを採用し、原研では64プロセッサのモデル(3次元方向のクロスバは未使用)を有している。SR2201のハードウェア諸元を表2.5.1に示す。

2.5.2 ソフトウェア構成

SR2201では並列プログラム開発のために各種ツールが提供されている。主なソフトウェア構成を表2.5.2に示す。

表 2.5.1 SR2201 ハードウエア諸元

構成	プロセッサ台数	64
	プロセッサ間転送速度	300MB/sec
	総メモリ容量	16GB
	総処理性能	19.2GFLOPS
プロセッサ	演算性能	0.3GFLOPS
	メモリ容量	256MB
	キャッシュメモリ	2次:512KB(命令), 512KB(データ)

表 2.5.2 SR2201 プログラム開発支援ツール一覧

言語処理系	FORTRAN77及び90 HPF C C++
デバッガ	ndb. xndb
性能解析ツール	ctool etool xtool
プログラム解析ツール	FORGE90
エディタ	viなど

2.5.3 並列プログラム開発支援ツール

日立 SR2201 では並列プログラムのためのコンパイラ・デバッガ・パフォーマンスマニアとメッセージ通信ライブラリをPARALLELWARE[15], [16], [17], [18]として提供している。この中にはソースコード解析ツールは含まれないが、別にFORGE90[19], [20], [21], [22]としてこれをサポートしている。図 2.5.1にPARALLELWAREの各ツールとFORGE90をプログラム開発作業の流れに対応付ける。

2.5.3.1 並列プログラミング言語及びコンパイラ

SR2201ではHI-UX/MPPと呼ばれるUNIXベースのオペレーティングシステムが搭載されている。HI-UX/MPPはMach3.0をベースとしたマイクロカーネルを採用している。複数プロセッサからなる並列計算機に対して、ファイルシステム、プロセス管理は1システムのUNIXワークステーションイメージで利用することができる。

並列プログラミングのための言語はCおよびFORTRANで、Cに関してはANSI C, C++, FORTRANに関してはFORTRAN77, FORTRAN90, HPF(Parallel FORTRAN)をサポートする。HPFは、分散メモリー型並列計算機向け配列データのデータ分散を指定するディレクトイブ(指示文)をFORTRAN90仕様のプログラムに変換するトランスレータとして供給される。

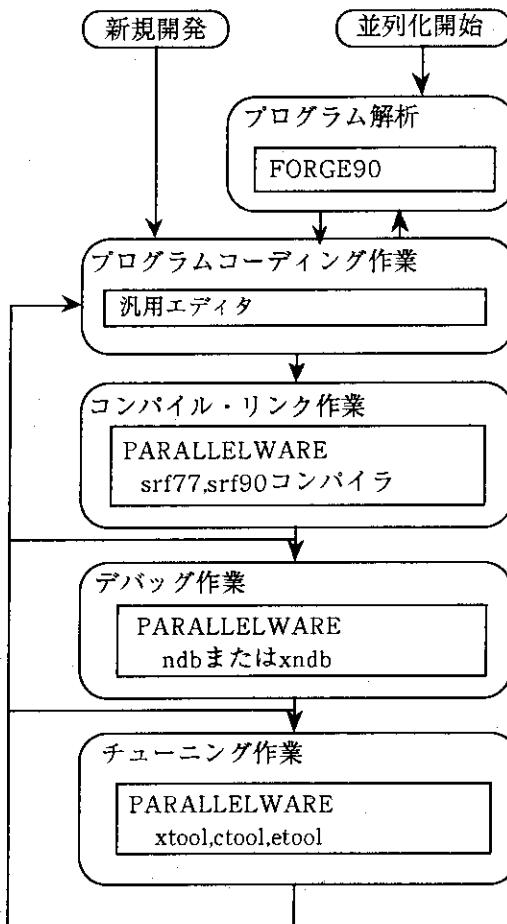


図 2.5.1 プログラム開発の流れとツールの関係

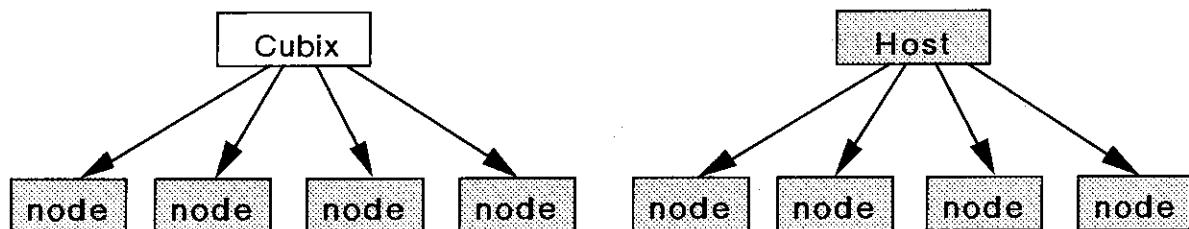


図 2.5.2 Cubix型とHost-Node型

メッセージ通信ライブラリはPVM, MPIも利用できるが、前述のPARALLELWAREでは性能評価ツールも含めたトータルな開発環境が提供されている。また、PARALLELWAREでは通信関数にPVMインターフェースを利用することも可能である。以下にPARALLELWAREを利用したプログラミングの特徴を列挙する。

(1) Host-Node と Cubix

PARALLELWAREメッセージ通信ライブラリを使用した場合にサポートされる並列プログラムのスタイルとして、並列プログラムの起動方法と入出力方法の違いにより、Cubix型とHost-Node型がある(図2.5.2参照)。

Cubix型では、Cubixというホストプログラムがあらかじめ用意されており、ユーザはノード

プログラムだけを作成すればよい(図2.5.2の網掛け部分)。ユーザのノードプログラムはコンパイルの前処理によりCBXMAINというサブルーチンに変更されコンパイルされる。ユーザはCubixというプロセスを起動し、CubixはCBXMAIN、つまり、ユーザのノードプログラムを起動する。Cubix型では各ノードプログラムに入出力処理を記述することが可能である。同一ファイルに複数のノードプログラムがアクセスする際の競合を制御するために、シングルモード、マルチモード、非同期モードの入出力モードが用意されている。ユーザが記述した入出力処理はコンパイルの前処理により各入出力モードを実現する処理に変換され、コンパイルされる。入出力モードについては後述する。

Host·Node型では、ユーザがホストプログラムとノードプログラムを陽に記述する。ユーザはホストプログラムを起動し、ノードプログラムはホストプログラムよりKXRUN関数をコールすることにより起動される。Host·Node型では同一ファイルに複数のノードプログラムがアクセスする際の競合を制御するのが難しいため、ホストプログラムに入出力を記述する。入力したデータは通信によってノードプログラムに送信される。また、ホストプログラムには、ノードプログラムの起動や入出力以外の様々な処理をユーザが自由に記述できる。

(2) 入出力

Host·Node型プログラムではファイルに対する入出力はホストプログラムで実行しなければならない、ノードプログラムはホストプログラムより通信により入力データを取得する。

Cubix型の場合、各ノードが同一のファイルに入出力するため、各ノードの実行順序が問題になる。PARALLELWAREでは3通りの入出力モードをサポートしており、入出力の一貫性を保つ助けになっている。

シングルモード

全プロセッサが同一のファイルポインタにしたがって入出力をを行う。出力時に各ノードの出力する内容が異なればエラーとなる。全ノードの入出力の完了で同期がとられる。

マルチモード

各ノードが決められた順番で入出力をを行う。順番待ちによる同期が発生する。デフォルトの入出力の順序はプロセッサ番号順であるがユーザにより変更もできる。

非同期モード

各プロセッサは勝手に入出力を行うことができる。同期はとられない。プロセッサ間の入出力の順序は保証されない。

(3) PVM インタフェース

SR2201のPARALLELWAREはPVM バージョン3.0とのインターフェースを持つ。これは、PARALLELWAREのメッセージ通信関数の代わりにPVMの関数を記述できるが、PARALLELWAREのメッセージ通信関数で実現できない機能などを中心に、一部制約事項がある。しかし、PARALLELWAREの性能評価ツールであるctool, etool, xtoolを利用できるという利点がある。

(4) 動的データ分割

プログラム実行時に、何台のプロセッサで実行するのか、プロセッサをどう割り振るのかを動的に決定することを可能とするexgridユーティリティが用意されている。プログラム中でこの機能を用いることにより、通信すべきプロセッサ、計算を実行すべき配列要素の範囲などを、実行時に得ることができる。

2.5.3.2 デバッガ

(1) 機能概要

SR2201ではndbによるラインモードでのデバッグ以外にxndbによるGUIを用いたデバッグ作業が可能である。機能的にndbとxndbで差異はないが、xndbの場合ノード毎に別のウインドウを用いることができる。いずれも、逐次型プログラム用デバッガの標準的な機能に加え、並列プログラムをデバッグするための機能を持っている。

並列プログラムのデバッグでは、複数のプログラムを制御する必要がある。ndb、xndbでは、デバッグ操作の対象ノードを指定することができる。対象ノードとして單一ノードあるいは複数のノードを指定できる。また複数のノードを集合として定義することもできる。デバッガ操作時に有効な対象範囲はプロンプトとして示される。例えば、ノード0が対象範囲である場合は「NODE 0>」、全てのノードが対象範囲である場合は「ALL>」、ノード1と3が対象範囲である場合は「1, 3 >」となる。また、コマンドの前に「on 対象範囲」という形式で指定することで、プロンプトと無関係にそのコマンドだけで有効な対象範囲を指定することもできる。

ユーザに提供されるプログラムの情報は、ソースコード、スタックの状態、実行状態、データの内容などがある。ソースコードの表示では、各ノードで動作しているプログラムを表示できると共に、ブレークポイントにより停止しているプログラムの停止位置の把握もできる。スタックの状態の表示では、プログラムの呼び出しなどにより積み上げられたシステムのスタックの内容を各ノードの最大上位8レベルまで表示する。状態の表示では各ノードのブレークポイント、内部通信レジスタ、メッセージキュー、レジスタ、セット、プロセス状態、プロセッサの利用時間などを全て表示する。データ表示では各ノードの数式、変数、配列、及び構造体を全て表示する。

ブレークポイントの設定では、各ノードに関数名、行番号、変数の変更による停止の設定がノード毎にできる。実行の再開もノード単位で制御可能であり、ノード間の実行順序を変えることによるデバッグが可能である。

xndbはXウインドウシステムに対応したデバッガであり、デバッガの機能はndbと全く変わらない。マルチウインドウの利点を生かし、ノード毎にコマンドインターフェースを設け、デバッガ操作ができる。これをxndbではノードウインドウという。xndb起動時に現れるメインウインドウは、上半分がデバッガコマンドに対応したボタンが配置された領域で、下半分がコマンド入力領域である(図2.5.3)。ノードウインドウはメインウインドウから派生する形で生成され、上部がソースリスト表示領域、中部がデバッガコマンドに対応したボタンが配置された領域で、下半分がコマンド入力領域である(図2.5.4)。メインウインドウとノードウインドウの

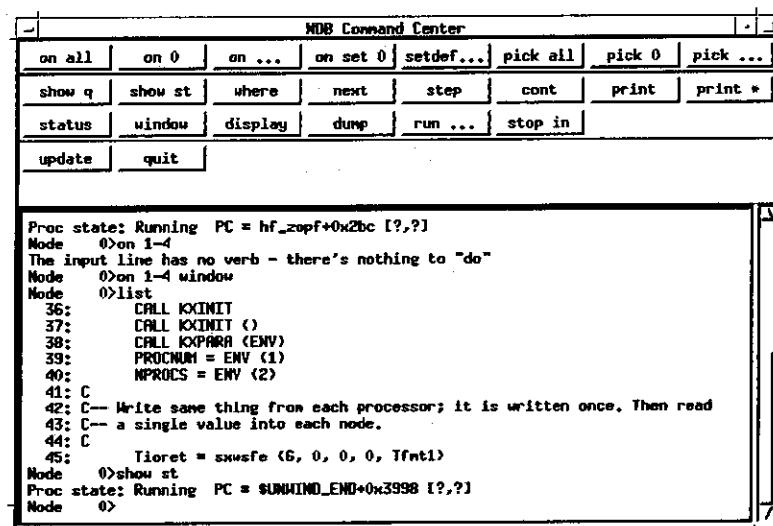


図 2.5.3 xndb のメインウインドウ

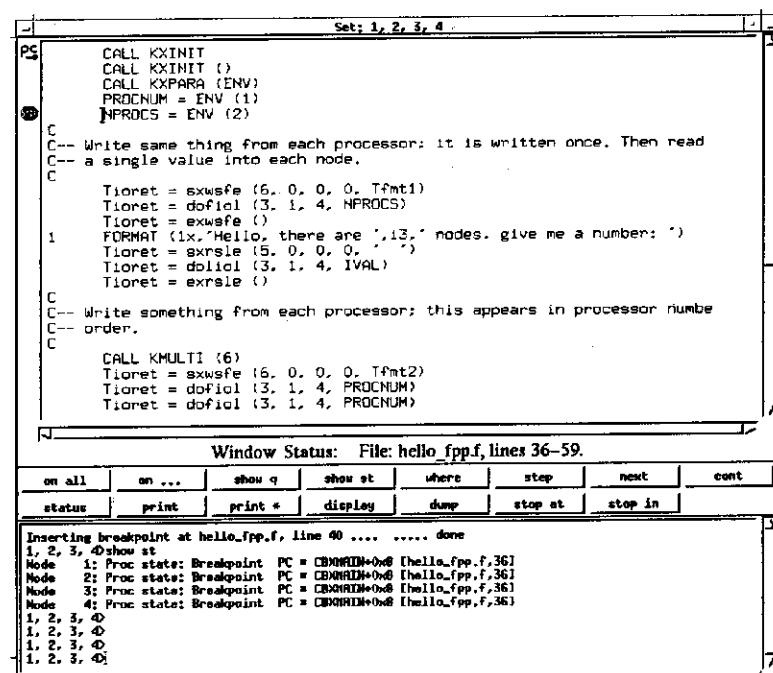


図 2.5.4 xndb のノードウインドウ

違いは、ノードウインドウは複数起動できることと、ソースリスト表示領域を持つことである。この点がxndbの大きな特徴である。

xndbのマルチウインドウシステムの下で、マウスによる対話的操作が可能である。デバッグ対象ソースプログラムのウインドウ上でのブラウズや、そのリスト上でのブレークポイントの設定が可能である。

(2) 評価

データの表示に関しては、科学技術計算ではデータ数が膨大になるため、数値による変数値表示で状態を把握するのは困難である。グラフィカルに表示できる機能を付加することにより、直観的に問題点を把握することが必要であると考える。また、データのグラフィカル表示においても、アプリケーションで扱われている物理的に意味のある表示方式がユーザにより自由に指定できることが必要である。

ブレークポイントに関しては、科学技術計算ではループの繰り返し数が大きいため、ループ中ではステップ毎にブレークポイントで停止、配列の値の確認、実行再開という手順を踏まなくてはならない場合には労力を要する。そのような労力の軽減策として、ループを停止しないまま配列要素の値をグラフィカルに表示出来る機能が必要であると考える。

xndbのGUIに関しては、対象ノード範囲の指定や関数を指定したい場合にはキーボードからテキストを入力する必要がある。これらの選択もプルダウンメニューなどによりマウスによる対話的操作が出来ることが望ましい。

2.5.3.3 性能評価ツール

PARALLELWAREには実行解析、通信解析、イベント解析のプロファイリングツールが提供されている。

(1) 実行解析 xtool

xtoolは計算時間について解析するツールであり、各ノード毎に、各関数・サブルーチン毎に、各ステップ毎に解析することが可能である。表示方式にはキャラクタ表示方式とグラフ表示方式がある。表示の種類を表 2.5.3 に示す。

(2) 通信解析 ctool

ctoolは通信時間について解析するツールで、プログラム実行における各通信関数の呼出し回数、通信時間を表示する。通信のタイミングや宛先などの情報を得ることはできない。表示の種類を表 2.5.4 に示す。

(3) イベント解析 etool

etoolはイベントとトグルの解析をする。イベントとは、システムコール、入出力、ノード間通信、およびユーザがプログラム中に追加したユーザイベントである。etoolは実行開始から、いつイベントが発生したかを解析する。トグルとは時間計測のための一種のフラグで、ユーザプログラム中でオン／オフする。etoolはトグルのオン状態の合計時間や回数などを解析する。etoolの表示形式を表 2.5.5 に示す。

表 2.5.3 xtool の表示形式一覧

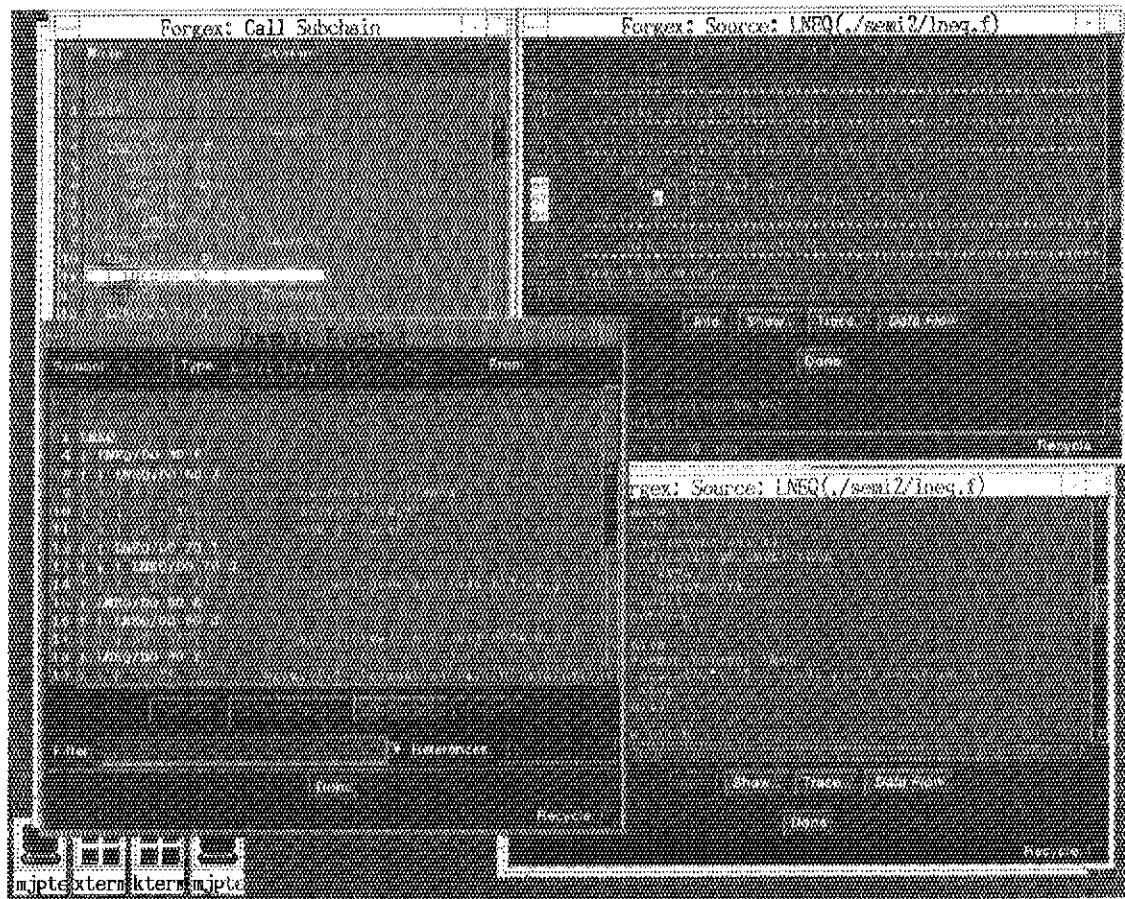
大分類	小分類	機能	表示方式
キャラクタ表示	ルーチン別リスト	サブルーチン毎のサンプル(ヒット)回数、CPU利用率	プロセッサごとにリスト表示
グラフ表示	Routin/Hits	サブルーチン対ヒット回数	プロセッサごとに棒グラフで表示
	Routin/Pct	サブルーチン対実行時間比率(%)	同上
	Routin/Calls	サブルーチン対コール回数	同上
	Hits/Node	プロセッサ対ヒット回数	サブルーチンごとに折線グラフで表示
	Pct/Node	プロセッサ対実行時間比率 (%)	同上
	Calls/Node	プロセッサ対コール回数	同上
	Source Code	ソースリストの行対応にCPU利用率の表示	CPU利用率とソースリストを並列に表示
	Assem. Code	アセンブリリストの行対応にCPU利用率の表示	CPU利用率とアセンブリリストを並列に表示
	Histograms	メモリアドレス対ヒット率	プロセッサ毎に折線グラフ表示

表 2.5.4 ctool の表示形式

大分類	小分類	機能	表示方式
キャラクタ表示	ルーチン別リスト	通信関数毎の呼び出し回数、利用時間、エラー数、閲数の戻り値別回数	プロセッサごとにリスト表示
グラフ表示	Func/Time	通信関数対利用時間	プロセッサごとに棒グラフで表示
	Func/Calls	通信関数対呼び出し回数	同上
	Time/Node	プロセッサ対利用時間	通信関数ごとに折線グラフで表示
	Calls/Node	プロセッサ対呼び出し回数	同上
	Usage/Node	プロセッサ対通信関数利用率	プロセッサ毎に、計算、ノード間通信、I/O、システムコール、描画、アイドルの時間割合を積み重ね棒グラフで表示

表 2.5.5 etool の表示形式

分類	表示方式
イベント発生グラフ	プロセッサ対イベント発生時間。プロセッサ毎に時間軸上にイベント発生状況をマーク
トグル解析表	トグル毎の合計時間、発生回数、平均値、偏差を表示 プロセッサごとにリスト表示

図 2.5.5 *forgex* の使用例

2.5.3.4 統合化ツール

SR2201のプログラム解析ツールとしてFORGE90が利用できる。また、FORGE90はプログラムの自動並列化機能も備えている。

(1) FORGExplorer forgex

*forgex*はプログラムソースコードにおける、DOループ、関数コールなどを解析し、プログラム開発者のプログラムの把握を支援する(図2.5.5)。また、変数については、クロスリファレンス情報に加え、プログラムのグローバルなデータベースを使用して、データの流れを追跡することができる。手続き引数や共通ブロックやデータの別名で対応関係がわかりにくくなっている、プログラムCALLツリーの至る所でデータの内容を調べることができる。また、ソースコードディスプレーウィンドウで変数をクリックすることによって選ぶことで、プログラムの全体にわたったその変数とその別名への全てのレファレンスを調べることができる。変数だけでなく定数やパラメーターとサブルーチン引数も調べることができる。

(2) FORGExplorer DMP,XHPF

DMP及びXHPFは逐次型のプログラムにHPFのディレクティブを自動的に挿入することにより、並列プログラムを生成する。DMPとXHPFの違いはDMPがGUIによる対話方式であるのにに対し、XHPFはコマンドによるバッチ形式で実行する点にある。

2.6 Cray T94における並列プログラミング環境

2.6.1 ハードウェア構成

Cray T94 [23] は、ベクトル演算機能を持つプロセッサ4台により構成された共有メモリ型ベクトル並列計算機である。1プロセッサあたり2本の浮動小数点演算バイラインを所有し、クロック周期2.2nsec、実効最大ベクトル性能7.2Gflops、実効最大スカラ性能1,800MIPSを実現している。また共有メモリとして1GBの記憶域を保持している。Cray T94システムハードウェアの諸元を表2.6.1に示す。

2.6.2 ソフトウェア構成

T94では並列プログラムの開発を支援するために、多数のツールが提供されている。提供されているツールを表2.6.2にまとめる。また、プログラム開発作業の流れと利用するツールの

CPU	CPU台数	4
	クロック周期	2.2nsec
	実効最大ベクトル性能	7.2Gflops
	実効最大スカラ性能	1.8GIPS
	主記憶容量	1GB
	CPU主記憶間転送速度	128GB/sec
外部メモリ	総メモリ容量	40GB
	実効転送速度	80MB/secまたは32.8MB/sec

表 2.6.1 Cray T94 システムハードウェア諸元

言語処理系	FORTRAN77 及び90		
	C		
	C++		
デバッガ		Total View	
性能解析ツール	プロファイリングツール	FLOWVIEW	
		JUMPVIEW	
		PROCVIEW	
		PROFVIEW	
	性能評価ツール	ATEXPERT	
		PERFVIEW	
		HPM	
エディタ		vi	
		emacs	
		XBROWSE	

表 2.6.2 Cray T94 システムプログラム開発支援ツール一覧

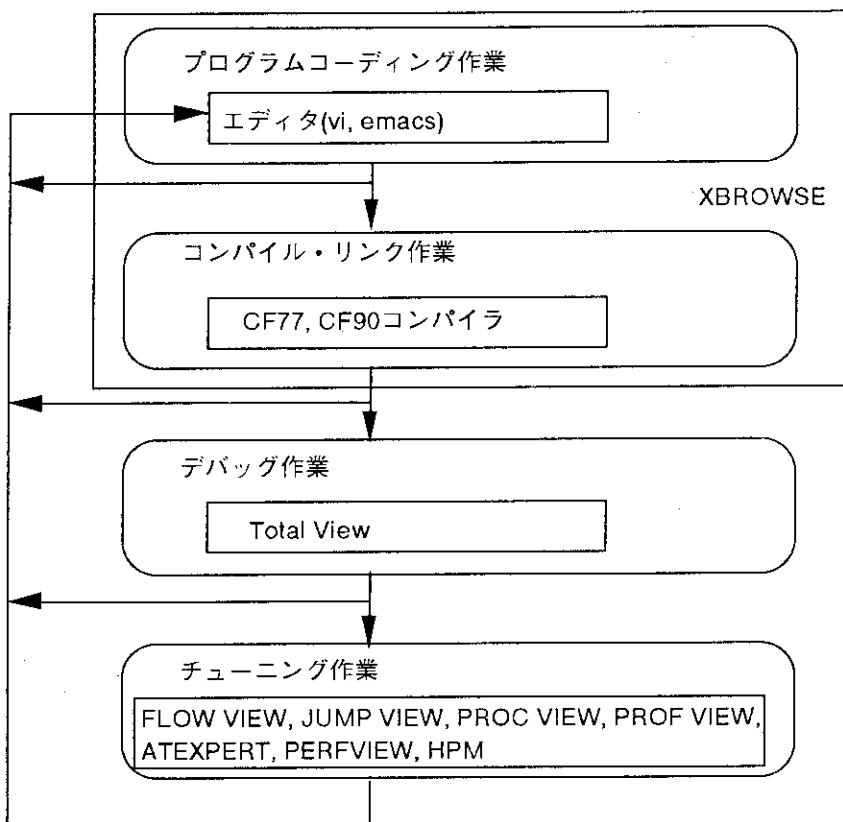


図2.6.1 Cray T-94における並列プログラム開発の流れと開発支援ツールの対応

対応を図2.6.1に示す.

2.6.3 並列プログラム開発支援ツール

2.6.3.1 並列プログラミング言語及びコンパイラ

Cray T94ではUNICOSと呼ばれるUNIXベースの並列OSにより、複数のCPU上で一つのプログラムを並列に実行するマルチタスキングを行うことができる[24].

ユーザはマルチタスキング機能を用いて、共有変数モデルに基づいた並列プログラムを開発することが可能である。共有変数モデルに基づくプログラムでは、複数のプロセスによる変数への同時アクセス制御が必要となるが、Cray T94ではシステムによる同時アクセス制御が実現されており、ユーザが意識する必要はない。

T94において並列実行を行う場合、以下の二種類の並列実行形態が存在する。

(1) 繰り返し構造に着目した並列実行

プログラムの繰り返し構造に着目してマルチタスキングを行う。異なるデータに対して同一の処理を行う形態の並列処理である。

(2) 独立なプログラムブロックに着目した並列実行

サブプログラムレベルの並列性を利用してマルチタスキングを行う。各プロセッサは各自異なる計算を行うことが一般的である。

この2つの並列実行形態を表現するために、T94ではFORTRAN, C, C++を並列ディレクティ

ディレクティブ名	意味
CMIC\$ DO ALL	本ディレクティブに続くDOループを並列実行することを示す
CMIC\$ PARALLEL 及び CMIC\$ END PARALLEL	本ディレクティブに挟まれたプログラム領域を並列実行することを示す
CMIC\$ CASE 及び CMIC\$ END CASE	上記のPARALLELディレクティブで挟まれたプログラムブロック内で用い、本ディレクティブにより分離されたプログラム領域を並列実行することを示す
CMIC\$ DO PARALLEL 及び CMIC\$ END DO	上記のPARALLELディレクティブで挟まれたプログラムブロック内で用い、本ディレクティブで挟まれたDOループを並列実行することを示す
CMIC\$ GUARD 及び CMIC\$ END GUARD	上記のPARALLELディレクティブで挟まれたプログラムブロック内で用い、本ディレクティブで挟まれたプログラム領域は臨界領域(critical section)であり、排他制御が必要なことを示す

表 2.6.3 T94 における主な並列ディレクティブ

により拡張した言語を提供している。主な並列ディレクティブ[25]を表2.6.3に示す。

T94では上記(1), (2)の2つの形態の並列実行が可能であるが、必ずしもどちらかの形態のみで並列実行を行う必要はなく、両者を混在させてプログラムを開発することも可能である。

各言語用コンパイラはユーザによって挿入された並列/ベクトル実行用ディレクティブに従って、並列/ベクトル実行コードを生成する。T94で提供されているコンパイラの大きな特徴は、自動並列化機能を持つことである[24]。この機能はオートタスキング機能と呼ばれ、ループ内のデータ依存解析を行うことにより自動的に並列/ベクトル実行可能部分を認識し、並列/ベクトル実行コードを生成する。

2.6.3.2 デバッガ

T94にはTotal View[26]という並列デバッガが実装されている。Total Viewは並列実行プロセスに関する情報の表示や制御をウィンドウを用いて実行できるブレークポイントタイプのシンボリックデバッガである(図2.6.2参照)。

Total Viewでは複数のプロセスを制御するために、コマンドを单一のプロセスだけでなく全プロセスに対して発行することが可能である。しかし、現在のところコマンド発行対象はその2種類に限られており、ユーザが希望する任意のプロセスグループに対して発行することはできない。並列プログラムのデバッグでは、幾つかのプロセスで構成されるプロセスグループに対してコマンドを発行することが頻繁に行われるため、プロセスグループへのコマンド発行機能は実現されるべきである。

Total Viewではデバッガによるプログラムの実行を任意の時点で中断し停止できるInterrupt機能が存在する。科学技術計算プログラムは、一般に時間進行や陰解法による求解のための大きなループ構造が存在する。このループ内のFORTRAN文に単純にブレークポイントを設定すると、ループが繰り返される毎に停止することになる。このループの繰り返し数は一般に非常に大きいため、単純なブレークポイント設定方式ではユーザは目的とする現象を発生させるた

```

174      flux(i,j,1) = flux(i,j,1) + flux(i,j,3) + flux(i-1,j,jg,1)
175      c
176      c
177      c-----8888-----c
178      c
179      c
180      c
181      c-----2. Calculate numerical flux-----c
182      c
183      c
184      do 200 j=1,nj
185      do 200 i=1,nx
186      dflux(i,j,1) = dflux(i,j,1) + flux(i,j,3) + flux(i-1,j,jg,1)
187      dflux(i,j,2) = dflux(i,j,2) + flux(i,j,2) + flux(i-1,j,jg,2)
188      dflux(i,j,3) = dflux(i,j,3) + flux(i,j,3) + flux(i-1,j,jg,3)
189      dflux(i,j,4) = dflux(i,j,4) * flux(i,j,4) + flux(i-1,j,jg,4)
190 200 continue
191      c

```

図 2.6.2 Total View メインウインドウ

めに多くの労力を必要とする。Total ViewのInterrupt機能を用いればブレークポイントを設定せず実行させ、目的とする現象が発生するループ回数に達した時点で実行を中断させることができある。

このような機能は一般的なプログラムではあまり有用ではないかもしれないが、科学計算のようにループ構造による繰り返し実行が特徴となっているプログラムをデバッグする際、プロセスを制御する上では有効であると考えられる。

Total Viewではブレークポイントの一種としてWatch Pointと呼ばれるものを設定可能としている。Fortranの文を対象とする一般のブレークポイントとは異なり、Watch Pointはある変数の値が変更された時点でプログラムの実行を中断するように指定することができる。すなわち、プログラムの実行に関する単純な制約を表現することができる。一般のブレークポイント設定方式では、例えば共有変数へのアクセス順序の解析は不可能である。しかし、このような制約に基づくブレークポイント設定機能を利用することにより、並列プロセスのアクセス順序の正当性を検証することができる。この機能もプロセスを制御する上で効果的である。

上記のWatch Pointは指定された変数の値が変更されたかどうかを常時監視する必要があるため、システムの負荷が大きい。すなわち、Watch Point機能を利用するにProbe効果による影響が大きくなりうる。T94ではこの負荷を低減するために、Watch Pointをハードウェアレベルで実現している。そのため、Watch Pointを設定してもProbe効果は小さく抑えられている。

2.6.3.3 性能評価ツール

T94では、性能評価ツールとして多くのツールが用意されている[27]。これらのツールは大別して、主にプログラムを構成しているサブルーチンの実行時間の占有率を示すプロファイリングツールと、主に並列処理性能を示す実行性能評価ツールの2種類に分類できる。各ツールの概要を以下に述べる。

(1) プロファイリングツール

- FLOWVIEW

プログラムの実行において、特定のサブルーチンあるいは関数の実行時間の測定結果を表示するツールである。以下の情報が採取される。

- (i) それぞれのルーチンに要した時間、全体の実行時間に占める割合、呼び出し1回あたりに要した時間

- (ii) 各ルーチンが呼び出された回数

- (iii) 1つのルーチンが呼び出したルーチンの一覧、1つのルーチンを呼び出したルーチンの一覧

- (iv) ルーチンのコールツリーグラフ

- JUMPVIEW

プログラムの任意の部分の実行性能情報を表示するツールである。FLOWVIEWがサブルーチンレベルの情報取得ツールなのに対し、JUMPVIEWはサブルーチンの中のさらに細かい部分の実行情報を採取できる。

- PROCVIEW

プログラム実行中におけるメモリ、I/Oデバイスの利用状況に関する情報を表示するツールである。プログラムが利用したメモリサイズや、いつ、どのファイルに対して何回のI/Oを行ったか、何バイトのデータI/Oがあったか、I/Oにどれだけの時間がかかったかなどの情報を採取する。

- PROFVIEW

PROFVIEWは、PROFと呼ばれるツールによって収集されたプログラム内のサブルーチンや関数の実行時間に関する情報をX-Windowやキャラクタ端末で表示するツールである。

PROFによる実行状況の収集は、後述のHPMと異なりOSレベルで行われる。

(2) 実行性能評価ツール

- ATEXPERT

並列化されたプログラムの速度向上率を予測するツールである。ツールを利用することによるオーバヘッドは10~20%程度である。

あるプロセッサ数で一度試走すれば、プロセッサの数を増減させたときに性能がどのように変化するかの予測がなされるため、試行時間の短縮が図れる。ATEXPERTの情報提供画面を図2.6.3に示す。

・ HPM (Hardware Performance Monitor)

HPMはハードウェア機構であるため、以下の特徴を持つ。

(i)個々の命令の実行回数、実行に要したクロック数まできわめて正確に測定できる。

(ii)測定に伴う負荷や誤差がほとんどない。

(iii)測定対象となるプログラムを一切変更する必要がない。すなわち、測定のために特別なライブラリを呼び出したりライブラリをリンクする必要がなく、またプログラミング言語にも依存しない。

採取される情報は以下のとおり。

(i)全CPU時間

(ii)ベクトル演算、スカラ演算、ベクトル・メモリ・リード／ライト等の実行命令数

(iii)種々のHold Issueのクロック周期数。命令実行に用いるデータの依存性や制御の依存性のために、命令を最適なサイクルで実行することができず命令実行待ちとなる状態が存在する。このような状態は一般にハザードと呼ばれる[36]が、CrayではこれをHold Issueと呼び、HPMではその解消に費やされた時間をクロック周期数単位で測定できる。

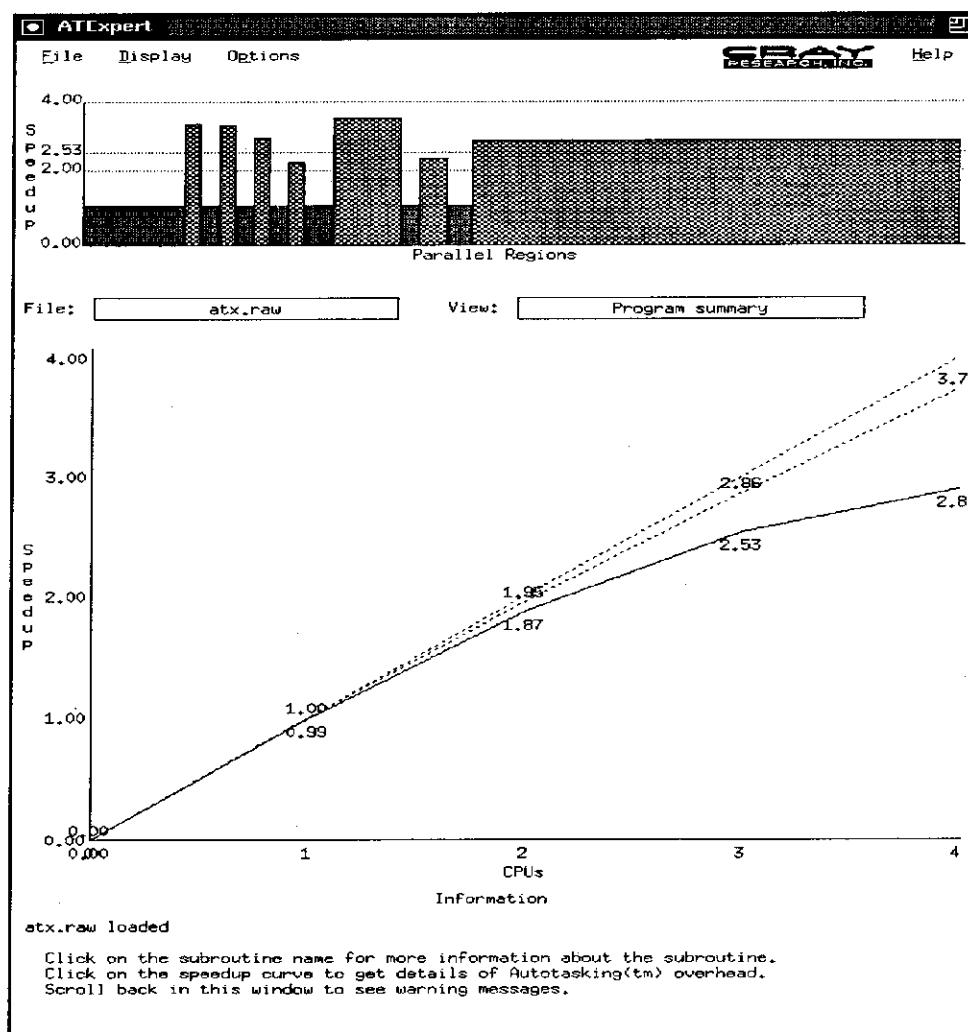


図 2.6.3 ATEXPERT の実行性能予測表示画面

(iv)命令バッファ取り出し回数

(v)メモリ参照回数

(vi)メモリコンフリクト回数

(vii)入出力ポートの参照回数

(viii)入出力ポートのコンフリクト回数

採取された情報は図2.6.4の形式でユーザに提供される。

• PERFVIEW

ハードウェアレベルの性能測定モニタリング機構(HPM)により採取された実行性能情報をもとに、プログラム中の個々のプログラム単位の性能に関する情報を表示するツールである。また、各サブルーチンの実行性能情報をもとに、チューニングの指針となるレポートの自動作成を行う。チューニングに習熟したユーザはデータから直接チューニングの指針を読みとることができるためにこのレポートの有用性は低いが、初心者にとっては有用性が高いと考えられる。

T94における性能評価ツールの特徴の一つは、ハードウェアレベルのパフォーマンスマニア(HPM)を装備しており、モニタリング環境下におけるプログラムの高速な実行を実現していることである。これは単に測定作業におけるターンアラウンドの向上だけでなく、パフォーマン

STOP executed at line 184 in Fortran routine 'GCFD2D'			
CPU: 33.750s, Wallclock: 11.552s, 73.0% of 4-CPU Machine			
Memory HWM: 2564134, Stack HWM: 1565414, Stack segment expansions: 0			
CPU seconds : 33.73	CP executing : 15178858883		
Million inst/sec (MIPS) : 47.16	Instructions : 1590808184		
Avg. clock periods/inst : 9.54	CP holding issue : 13591827195		
% CP not issuing : 89.56	Inst.buf. fetches : 18159730		
Inst.buffer fetches/sec : 0.54M	F.P. ops : 19222725026		
Floating ops/sec : 569.89M	Vec F.P. ops : 19220340960		
Vector Floating ops/sec : 569.82M	actual refs : 19813812357		
CPU mem. references/sec : 587.42M	actual ref CP : 897053632354		
avg CP/mem. reference : 45.27	actual refs : 19683765792		
VEC mem. references/sec : 583.58M	actual refs : 117885141		
B/T mem. references/sec : 3.48M	actual refs : 2715		
I/O mem. references/sec : 0.00M	actual refs : 24512082		
Cache Hit Ratio : 70.30%	actual cache ops : 24512082		
Hold issue condition	% of all CPs	actual # of CPs	
Waiting on A-reg & access	: 5.82	883390048	
Waiting on S-reg & access	: 3.08	468904129	
Waiting on V-registers	: 34.85	5290100362	
Waiting on B/T-registers	: 0.15	22739017	
Waiting on Functional Units	: 33.84	5136672823	
Waiting on Shared Registers	: 2.85	432956105	
Waiting on Memory Ports	: 11.71	1778089201	
(octal) instruction type	inst./CPUusec	actual inst.	% of all insts.
(000-004) Special	: 0.28M	9582027	0.60
(005-017) Branch	: 1.88M	63410883	3.99
(02x, 030-033) A Register	: 19.06M	643024231	40.42
(034-037) B/T Memory	: 0.02M	787170	0.05
(040-043, 071-077) S Register	: 2.74M	92413600	5.81
(044-061, 0701ij6) Scalar Integer	: 3.70M	124928414	7.85
(062-070) Scalar Floating-Point	: 0.07M	2384066	0.15
(10x-13x) Scalar Memory	: 0.87M	29392491	1.85
(140-177) All Vector	: 18.53M	624885302	39.28
type of vector operation	ops/CPUusec	actual ops	
Vector Logical	: 138.59M	4674584803	
Vector Shift/Pop/LZ	: 15.98M	539097139	
Vector Integer Add	: 67.52M	2277381675	
Vector Floating Multiply	: 307.74M	10380267150	
Vector Floating Add	: 241.02M	8129698010	
Vector Floating Reciprocal	: 21.06M	710375800	
Vector Memory Read	: 387.82M	12406556684	
Vector Memory Write	: 215.75M	7277209108	
Average Vector Length for all Operations : 74.25			
hndcr1hndcr1%			

図2.6.4 HPMにより提供されるハードウェアモニタリング結果

スモニタを使用することによるProbe効果を小さくする上で重要である。

また、T94ではメモリコンフリクト発生数、ベクトル演算命令実行数といったハードウェアレベルの情報やサブルーチンレベル、ループレベル、任意のプログラム部分という形でのユーザの観点に応じたプログラムレベルの情報が豊富に提供される。特に、ATEXPERTによって提供されるプロセッサ数を変化させた時の実行性能予測情報は、プログラムの実行特性を調べるためにプロセッサ数を変更して何度もプログラムを実行するという従来の作業を軽減するという点で、非常に有用だと考えられる。

しかし、これらの情報を総合的に利用するためには、どのツールがどのような機能を持っているかを認識し、利用するツールに応じてプログラムをコンパイルし、実行するという作業を繰り返す必要がある。

一般にプログラムの性能評価作業はまず全体の性能を評価し、その後サブルーチンレベル、ループレベル、ハードウェアレベルという様々なレベルにおける詳細な情報を段階的に利用していく。この作業の過程において、どのレベルの情報を必要とするかを決定するのはユーザであるが、そのためにどのツールを利用しなければならないかは本来ユーザの行うべき判断ではない。ユーザはその作業を性能評価作業の一環として捕らえているのであるから、その支援を行うべきツールも一つの統合支援ツールとしてユーザに捉えられるべきである。

この問題を解決するためには、T94における性能評価ツール群を次のような形で統合化することが望ましい。すなわち、性能評価のためのソフトウェアアーキテクチャとして

- (i) ハードウェアレベルの情報を採取するハードウェアモニタの存在するレイヤ
- (ii) レイヤ(i)で採取された情報を種々の観点から処理を行う解析ツール群の存在するレイヤ
- (iii) 見たい情報をユーザが簡単に取得できるようにレイヤ(ii)の解析ツール群を制御するユーザインターフェースレイヤ

の3つのレイヤを考え、各レイヤのツールが連携することによりユーザに種々の観点からの情報を提供する。T94における解析ツールでは(i)のレベルあるいは(ii)のレベルのツールが単独で存在し独立に情報を提供するため、ツールを変更するたびにプログラムの再コンパイル、再実行を行う必要がある。また、ユーザは各ツールの機能を常に意識する必要がある。この独立なツール群を、必要な情報を一括して採取するツール、その情報を利用するツールに分割し、さらにユーザに各ツールを隠蔽する統合ユーザインターフェースを通して情報を提供することが、上記の問題点解決への第一歩であると考えられる。そのためには、各レイヤでどのような情報が必要かを明確化し、ツール間の連携を検討する事が必要である。

2.6.3.4 統合化ツール

T94ではXbrowse [28] というエディタを核とした種々のツール間の連携が実現されている。以下、Xbrowseに関して説明する。

Xbrowseは、Xウィンドウ対応のプログラムブラウザとしてマニュアルに位置づけられているが、ファイル管理機能、ファイル修正機能、プログラムの静的解析機能などを持ち、プログラム開発環境と見なすことが可能である。Xbrowseの画面を図2.6.5に示す。

Xbrowseの持つ主な機能は以下のとおりである。

- (1)ループ、ルーチン間のクロスリファレンス、コモンブロックに関する情報を生成する。
- (2)複数のプログラムファイルを1アプリケーションとして管理する
- (3)ファイルのツリー構造を生成する。
- (4)他のツールとの連携が可能である。

atscope--ループの依存性解析ツールであり、ループの並列化、ベクトル化に関して変数を解析し、解析情報を表示する。

cflint--ファイルの整形を行なう。

- (5)コンパイラと連携し、エラーメッセージを表示する。

ループ依存性解析作業の際、Xbrowseには並列化、ベクトル化の対象となりうるループの一覧が表示される。解析対象のループの指定を行うと、Xbrowseはatscopeと連動してプログラム内の対象ループを表示する。ループ解析作業において、ユーザはまずプログラムのループ構造に着目し、その後、対象ループ内の詳細な構造を解析する。従って、このプログラムからループ構造を抽出してユーザに提供する情報提供手法はユーザの作業手順に沿っており、有効である。

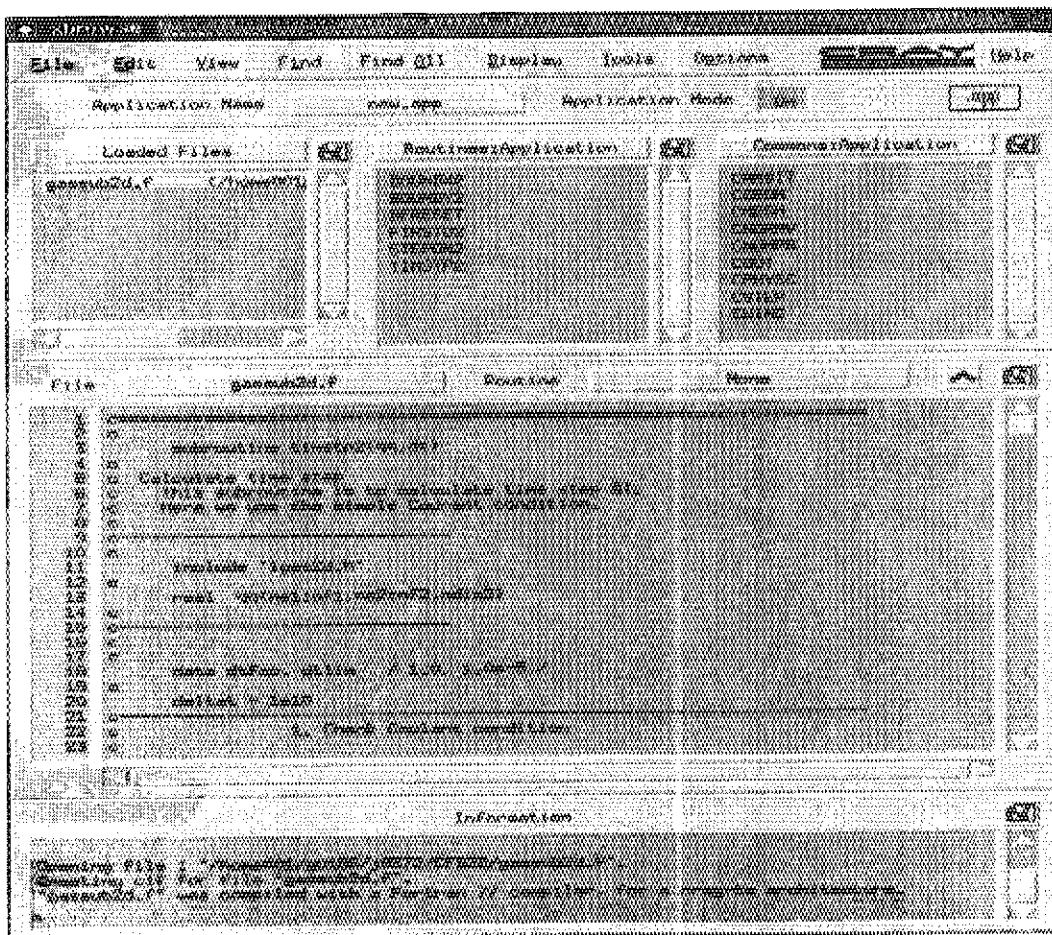


図 2.6.5 Xbrowse のメインウィンドウ

上記のように、Xbrowseではコンパイラや静的解析ツールとの連携が実現されており、実際にプログラムを実行する直前までの作業を全てこの環境内で行うことができる。しかし、プログラム開発ではプログラムの実行後もデバッグ、チューニングなどを行う必要がある。これらの作業においても必ずエディタによる修正は発生する。従って、デバッガ、チューナ等のツールとの連携も実現することが望ましい。

2.7 IBM SPにおける並列プログラミング環境

2.7.1 ハードウェア構成

IBM SPは1つのフレームに2～16台のPOWER2 RISCプロセッサを搭載したシステムで、このフレームを複数台接続することによりシステム全体で最大512プロセッサ・ノードを持つことができる多段結合型スカラ並列計算機(分散メモリ)である。各プロセッサ・ノードはハイパフォーマンス・スイッチで接続され、一組みのノード間ピーク転送能力は40MB/秒である。平成7年度に日本原子力研究所計算科学技術推進センターに導入されたIBM SPシステムのハードウェア諸元を表2.7.1に、その構成を図2.7.1に示す。このシステム構成は、50ノードからなり、2個のインタラクティブノード^{*1}と48個のバッチノード^{*2}に分けられている。

2.7.2 ソフトウェア構成

IBM SPでは並列プログラム開発のために各種ツールが提供されている。主なソフトウェア構成を表2.7.2に示す。

2.7.3 並列プログラム開発支援ツール

2.7.3.1 並列プログラミング言語及びコンパイラ

プログラミング言語としてFortran[29]、C[30]、C++がサポートされている。並列化ライブラリとしてはMPL(IBM独自の並列化ライブラリ)、PVMe(PVMのHPS(ハイパフォーマンススイッチ)対応版)、MPI[31]がある。また科学計算用ライブラリとして並列版ESSL[32]が提供されている。

コンパイラは並列化のための機能はなく、従来の逐次型計算機のコンパイラと変わりない。

2.7.3.2 デバッガ

並列プログラムのデバッグのために、行指向のpdbxとX-Windowsインターフェースを持

※1 脚注

一般ユーザがログインして作業(コンパイル・エディット・逐次実行用・並列エミュレーション実行)を行なうことができる。

※2 脚注

バッチジョブを実行するノードで、一般ユーザはログインできない。一般ユーザは、IBMロードレベラーというバッチジョブ・スケジューリング・プログラムを用いてジョブを投入する。

表 2.7.1 IBM SP ハードウェア諸元

構成	プロセッサ台数	50台
	プロセッサ間転送速度	40MB/sec
	総メモリ容量	3.2GB
	総処理性能	13.3GFLOPS
プロセッサ	演算性能	0.266GFLOPS
	メモリ容量	64MB
	キャッシュメモリ	32KB(命令) 128KB(データ)

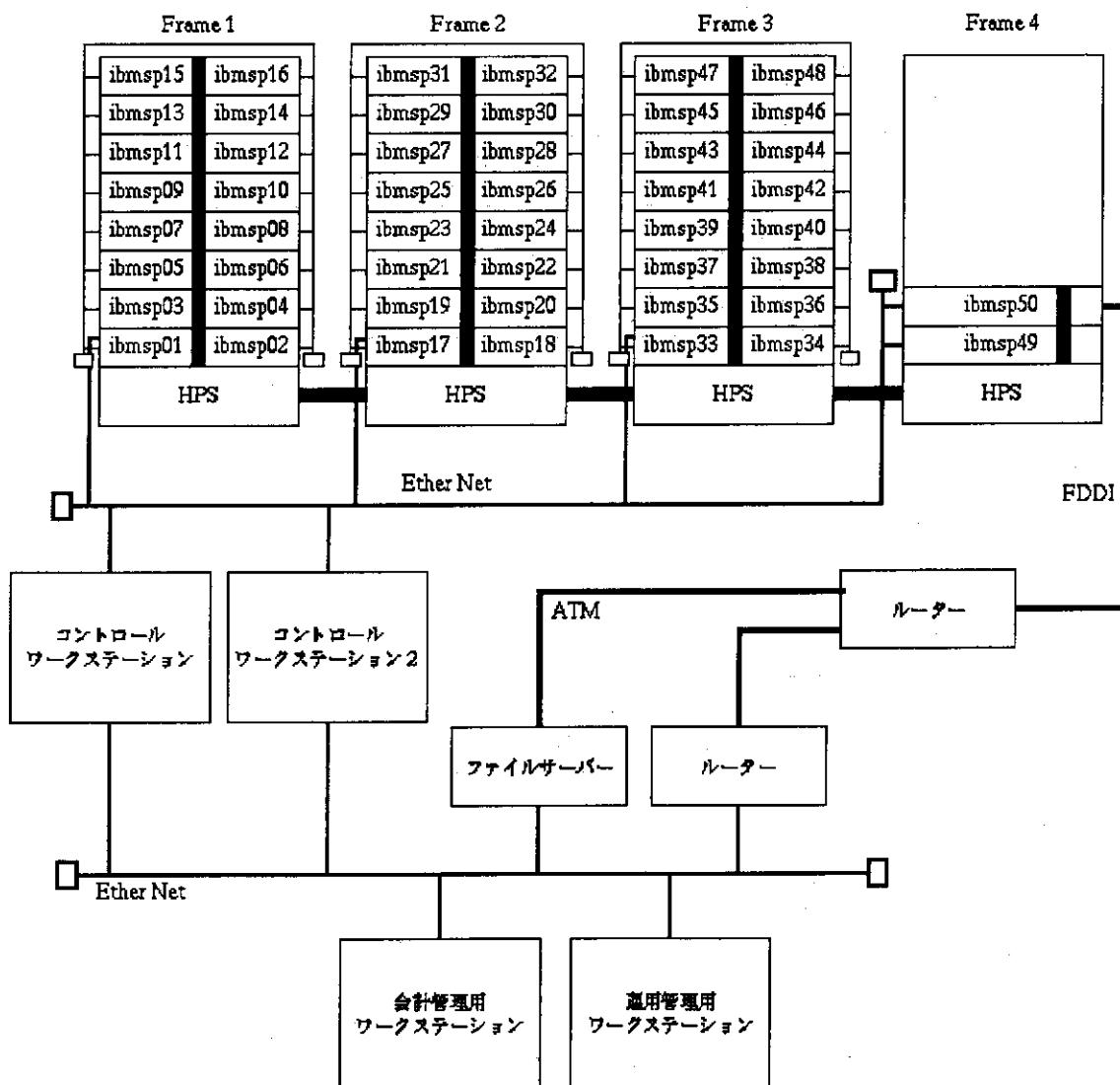


図 2.7.1 IBM SP ハードウェア・システム構成例

表 2.7.2 プログラム開発支援ツール一覧

言語	Fortran(mpif), C(mpcc), C++(mpCC)
デバッガ	並列デバッガ(pdbx, xpdbx)
性能評価ツール	プロファイラ(prof, gprof), プロセッサ・モニタ(poestat), 可視化ツール(vt)
統合化ツール	なし

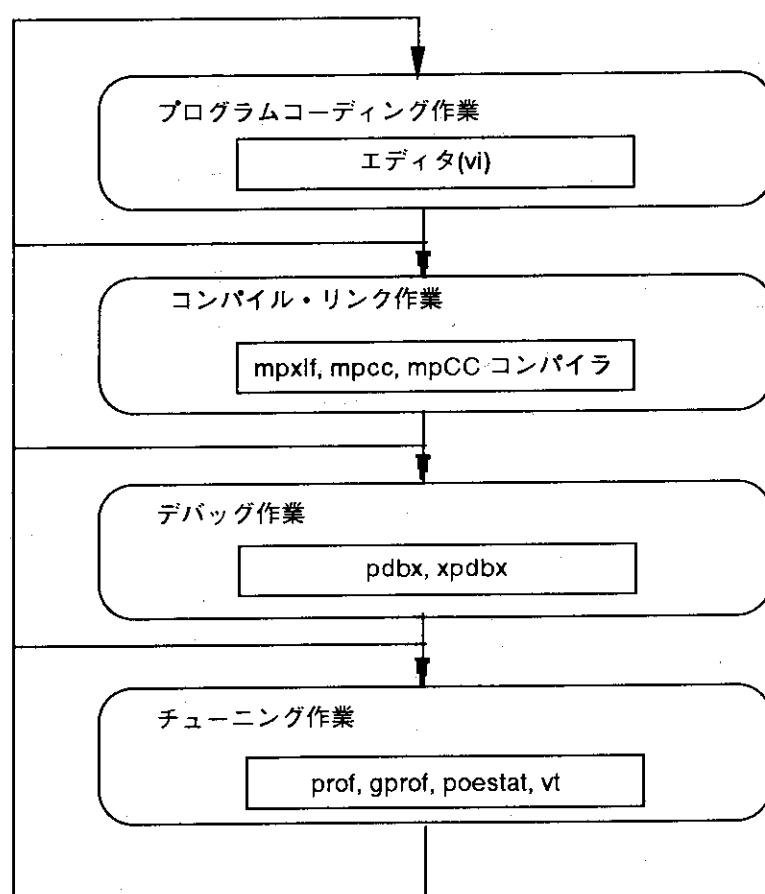


図 2.7.2 並列プログラム開発の流れと開発支援ツールの対応

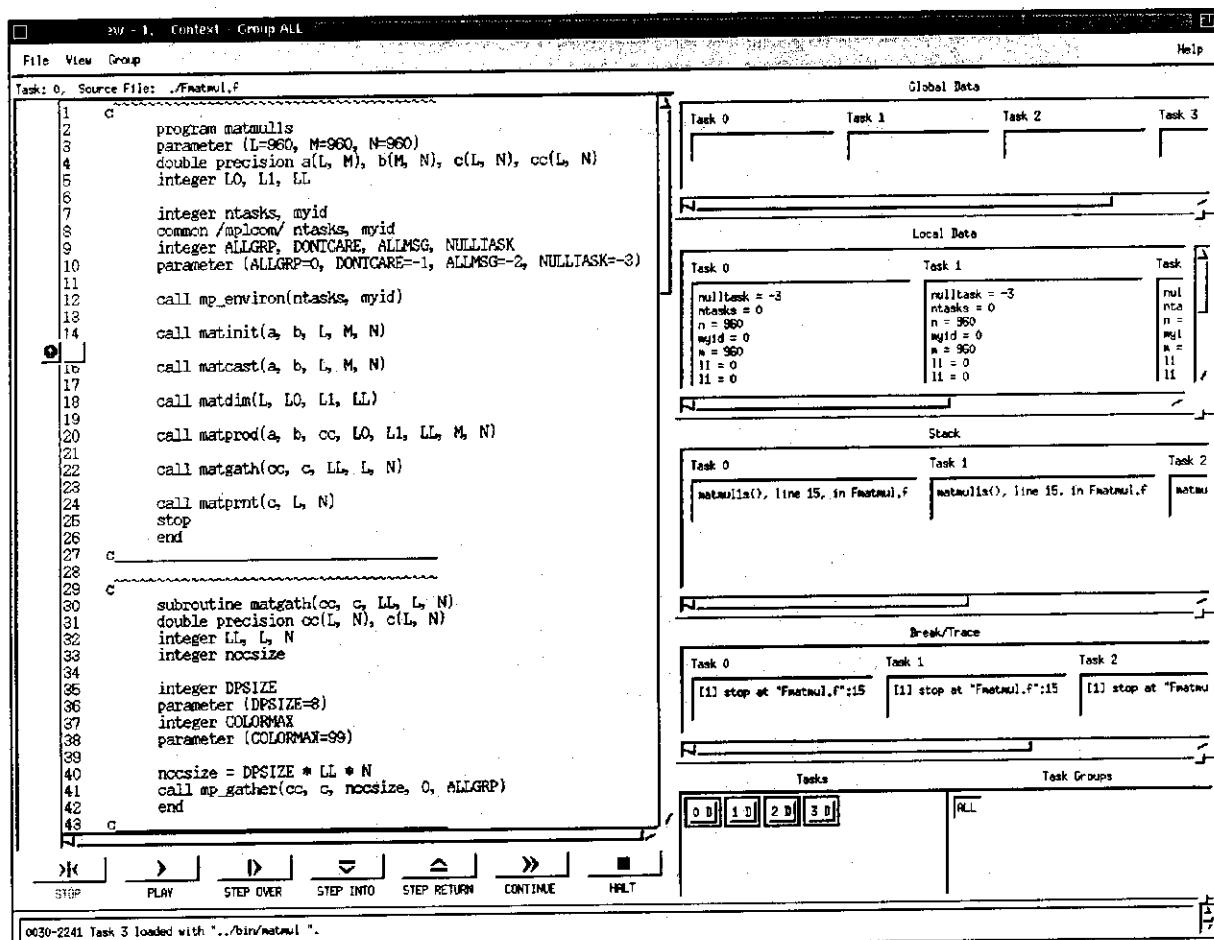


図 2.7.3 xpdbx ウィンドウ

つ xpdbx (図 2.7.3 参照)という2つの並列デバッガがサポートされている。これら2つの並列デバッガは、ブレークポイントの設定、プログラムの実行・再開、変数値の表示、プログラムのトレース等dbxに備わっているデバッグ機能を並列プログラムに対して実現している。しかしProbe効果を低減し、現象の再現性を高めるための機構は備えていない。

2.7.3.3 性能評価ツール

性能評価ツールとして

- ・プロファイラ
- ・プロセッサ・モニタ
- ・可視化ツール

がサポートされている。

(1) プロファイラ

並列プログラムのプロファイル情報のためにprofとgprofというコマンドがサポートされている。ユーザプログラムをプロファイル情報収集用のライブラリをリンクしたものを実行す

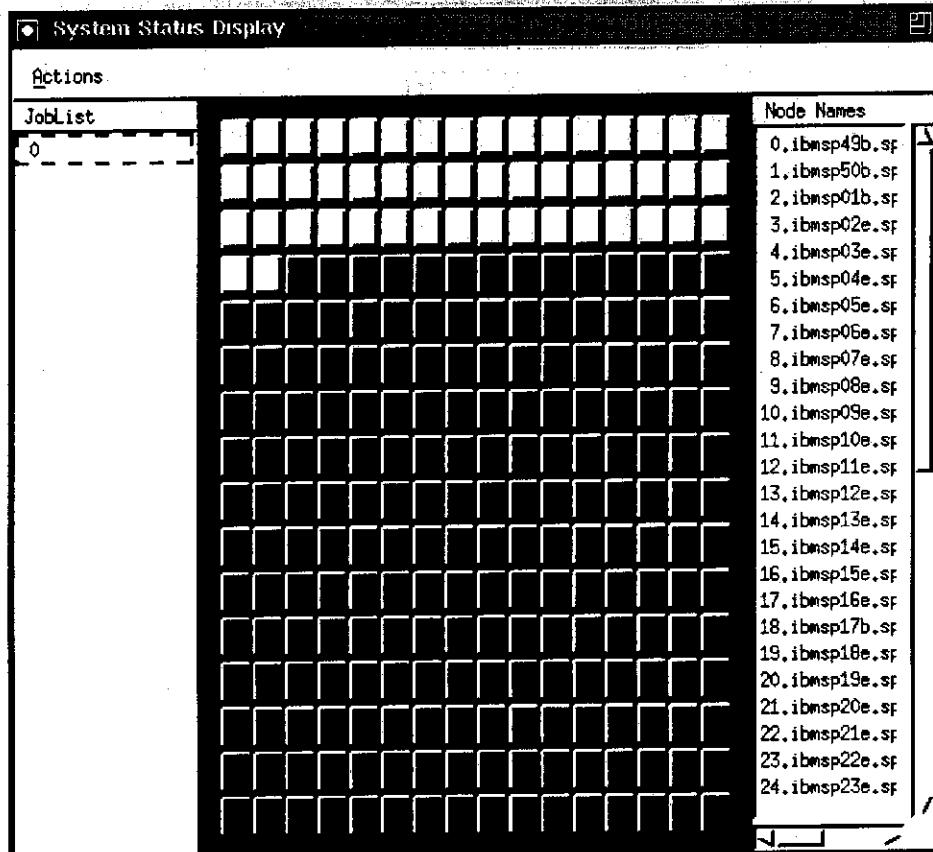


図 2.7.4 システム・ステータス・アレイ

ることによりプロファイル情報を収集する。prof, gprof はこの情報を読み込み表示する。

prof, gprof は以下のようなプロファイル情報を表示する。

- ・各サブルーチン実行時間の全実行時間に占める割合
- ・各サブルーチン実行時間の合計
- ・各サブルーチン呼び出し回数
- ・各サブルーチンの 平均実行時間

gprof はこれらに加えて

- ・サブルーチン間の呼び出し関係および実行時間

を収集・表示する。

(2) プロセッサ・モニタ

プロセッサ・ノードの使用状況を調べるプロセッサ・モニタとして、システム・ステータス・アレイ(図 2.7.4 参照)という X-Windows 分析ツールが提供されている。

システム・ステータス・アレイの各四角形は、プロセッサノードを表し、色で CPU 使用率を示す。アレイの右側にあるのは、アレイ内の各ノードの名前であり、これによってユーザはノードの名前を識別する。ノードは、左から右に、アレイの一番上の行から表示される。

(3) 可視化ツール

並列処理環境可視化ツール(VT)[33]は、ユーザプログラムおよびシステムの独自パフォーマンス特性を示す表示またはビューの集まりである。各ビューは、複雑な情報を親しみやすく理解しやすい形で表示する。たとえば、1つのビューは、棒グラフ、帯状グラフ、円グラフ、または格子を表す。多くの異なるビューがあるので、各々のユーザに適したビューおよびビジュアル化して情報を見ることができる。

VTのビューは以下に述べるように、トレースのビジュアル化(図2.7.5参照)とオンライン・パフォーマンス監視(図2.7.6参照)に使用できる。

トレースのビジュアル化

プログラムの実行中に生成された統計レコードとイベント・レコード(またはトレース・レコード)を再生する。基本的なシステムの使用法だけでなく、プログラムについての情報をビジュアル化するときにも VT が使用できる。

パフォーマンス監視

操作状況およびそれぞれのプロセッサ・ノードの動作状況を調べるためにオンライン・モニタとして VT を使用できる。パフォーマンス監視では、VT はシステム統計情報だけを表示して通信情報は表示しない。

VT のビューでビジュアル化できる項目を以下に列挙する。

- ・プロセッサ・ノード間の通信
- ・通信イベントのタイプおよび時間
- ・プロセッサ・ノードの CPU 使用率
- ・実行可能ファイルの実行に伴う並列プログラムのソースコード
- ・プログラムのタスクがプロセッサ・ノードで活動状態の実行からスワップ・アウトおよびスワップ・インされる回数
- ・ディスクからの読み込み、ディスクへの書き出し、およびディスクとプロセッサ・ノードの転送の回数
- ・プロセッサ・ノードが送受信した TCP/IP パケットの数
- ・プロセッサ・ノードが 1 ページの仮想記憶を実メモリにロードしなければならない回数
- ・プロセッサ・ノードがカーネル・サブルーチンを呼び出す回数
- ・システム活動の要約

2.7.3.4 統合化ツール

ソフトウェア構成の項で述べたような環境が提供されているが、個々のツール間の連携を実現しているツールは存在しない。

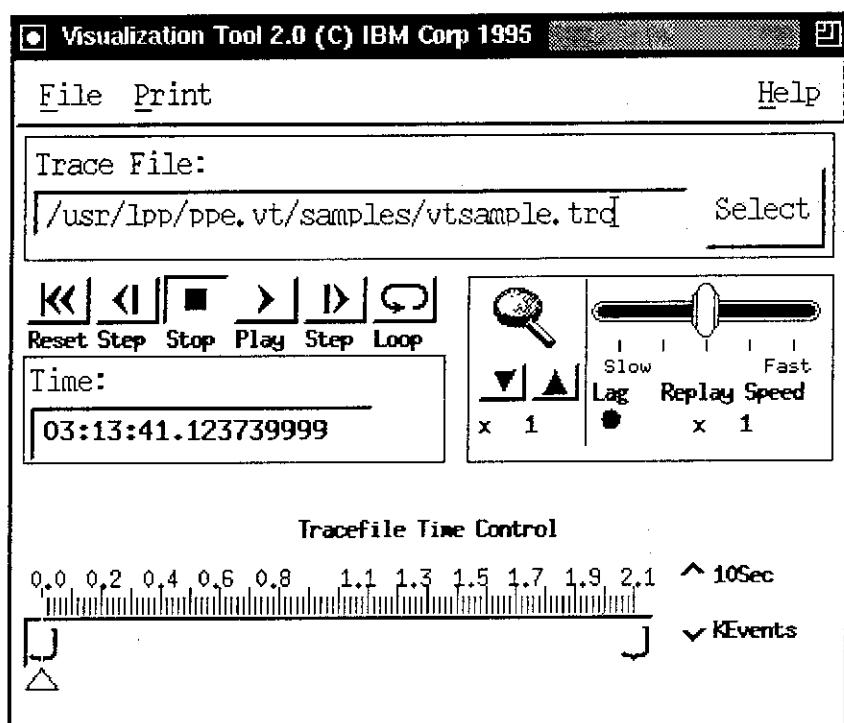


図 2.7.5 トレースのビジュアル化用 VT ウィンドウ

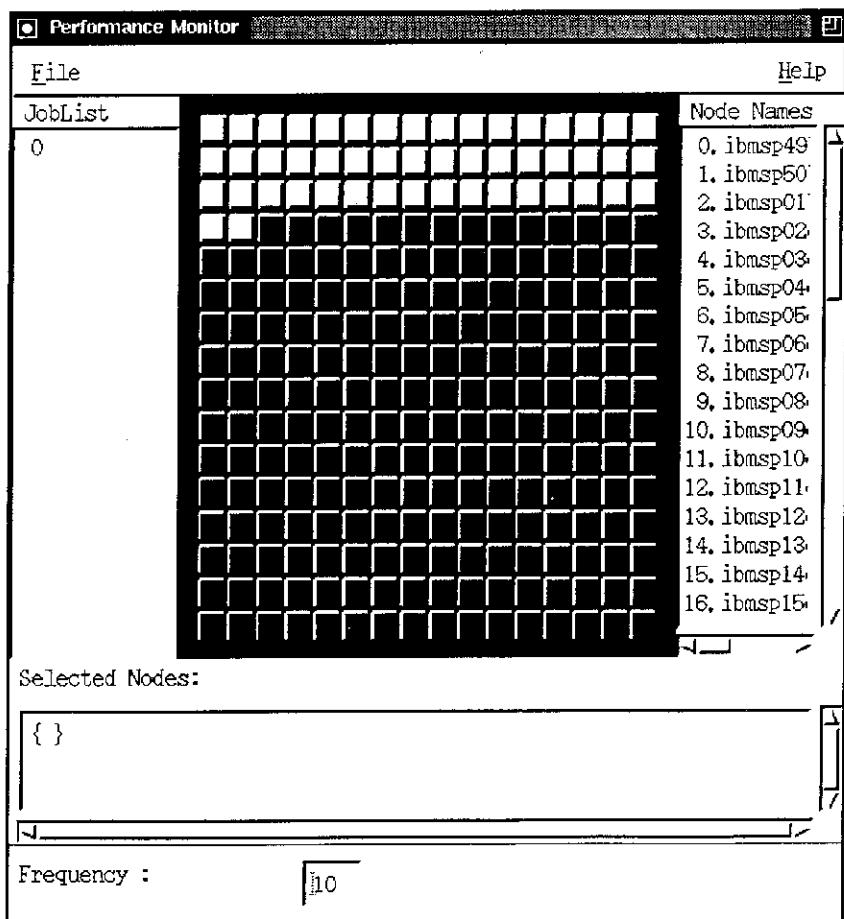


図 2.7.6 パフォーマンス監視ウィンドウ

2.8 Intel Paragon XP における並列プログラミング環境

2.8.1 ハードウェア構成

Paragon XP [34] は分散メモリ型スカラ並列計算機である。プロセッサ間ネットワークは高速に相互結合され、最大2000台までのプロセッサを結合することができる。Paragon XP/S15-256のハードウェア諸元を表 2.8.1 に示す。

2.8.2 ソフトウェア構成

Paragon XP では並列プログラム開発のために各種ツールが提供されている。主なソフトウェア構成を表 2.8.2 に示す。また、プログラム開発作業の流れとツールの対応を図 2.8.1 に示す。

2.8.3 並列プログラム開発支援ツール

Paragon XP では並列プログラムのためのコンパイラ・デバッガ・パフォーマンスマニア等の開発環境ツールが利用できる[35]。また、これらのツールを簡易に起動するためのツールとして ParAide があるが、日本原子力研究所では利用できない。

表 2.8.1 Paragon XP/S15-256 ハードウェア諸元

構 成	プロセッサ台数	256
	プロセッサ間転送速度	200 MB/sec
	総メモリ容量	8 GB
	総演算性能	19.2 GFLOPS
	プロセッサ	
プロセッサ	演算性能	75 MFLOPS
	メモリ容量	32MB

表 2.8.2 Paragon XP プログラム開発支援ツール一覧

言語処理系	Fortran77,(Fortran90)
	C, (C++)
	(Paragon HPF)
	(Validated Ada)
デバッガ	IPD, XIPD
	prof,xprof,gprof,xgprof
	ParaGraph
性能解析ツール	SVP
エディタ	viなど
統合環境	(ParAide)

() 内のツールは原研で現在使用できない。

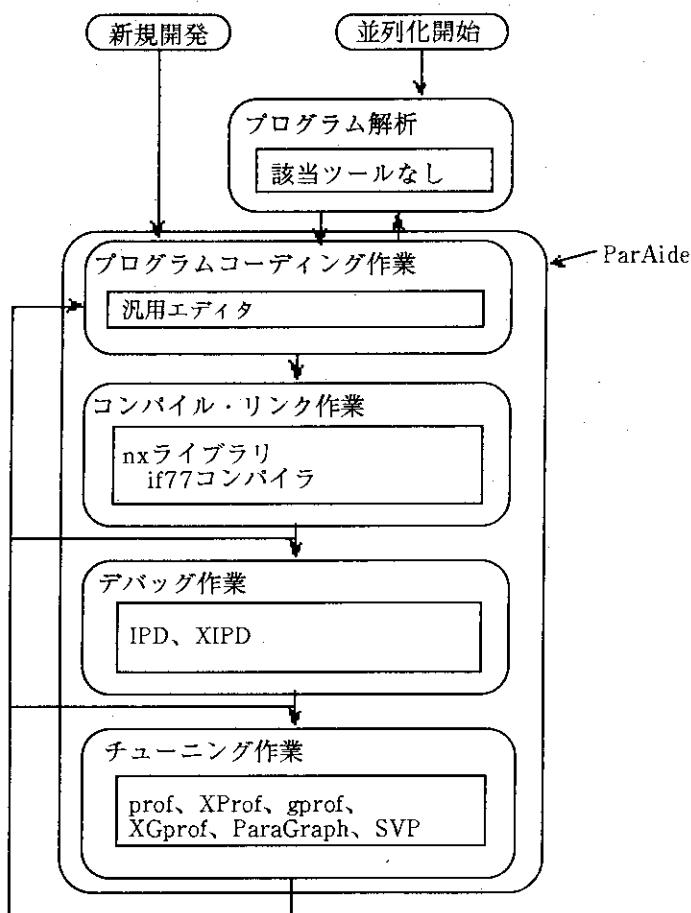


図 2.8.1 プログラム開発の流れとツールの関係

2.8.3.1 並列プログラミング言語及びコンパイラ

言語は、Fortran-77, C, C++, Validated Ada をサポートする。Fortran-77 と C 言語は相互呼出可能である。原研で使用可能なものは Fortran-77 と C である。

Fortran-77 コンパイラは、DO/ENDDO のような標準の Mil Spec の拡張がなされている。また、Paragon HPF プリプロセッサも装備されており、これを用いると、標準の HPF のデータ分割指示文を用いて、Fortran-77 と Fortran-90 プログラムを SPMD 並列プログラムに変換することができる。この並列プログラムはシステムの Fortran-77 コンパイラによってコンパイルすることができる。

通信ライブラリは NX ライブラリと呼ばれ、MIMD と SIMD プログラミングをサポートし、並列化と最適化を支援するさまざまな特徴を備えている。NX ライブラリのメッセージ通信方式には同期型(c), 非同期型(i), 非同期(h)ハンドラー型がありそれぞれの関数が c, i, h で始まる。送信(send), 受信(recv), 送信後受信(sendrecv)のほかに同期型, 非同期型には探索(probe)が用意されている。

プログラムのロード・実行時にオプションとして、実行するノード数、パーティションなどを指定できる。複数のプログラムから構成されるアプリケーションを一度に起動できる仕組み

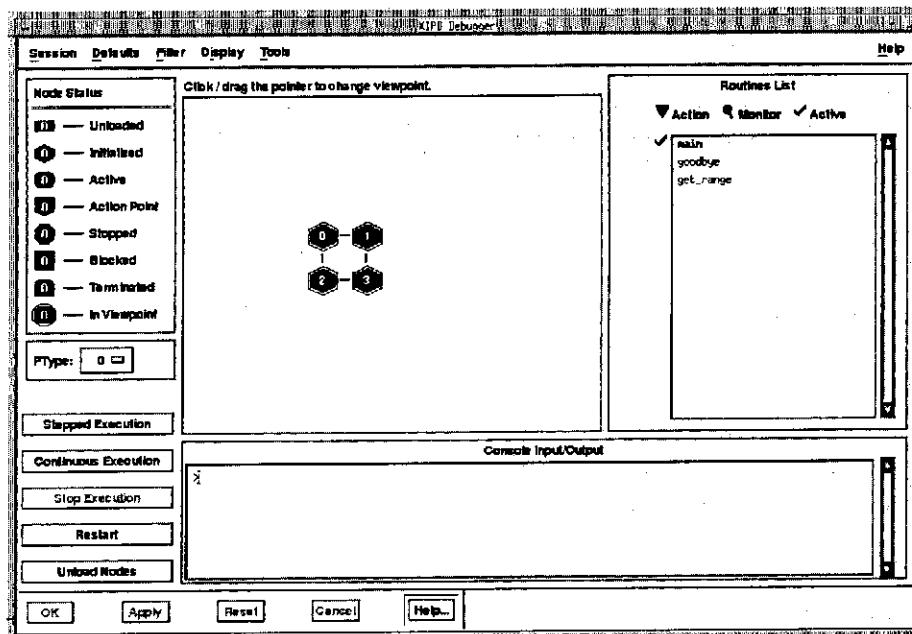


図 2.8.2 XIPD のメインウィンドウ

が、シェルインターフェースの機能としてユーザに提供されている。

2.8.3.2 デバッガ

Paragon XPでの並列プログラム用デバッガには、ラインモードのIPDとGUIによるデバッグ作業が可能であるXIPDがある。機能的にはIPDの大部分をサポートしている。しかし、サポートしていないものはあまり必要性の無いものや、GUI化によって不要となったものがあるので、実用上の問題は無いと思われる。

IPDの起動はプログラムを実行するParagon上で行うのに対して、XIPDはParagon上、フロントエンドWS上のいずれからも可能である。WS上からXIPDを起動する場合、loginセッション名などの入力が必要となる。

IPDではデバッグ操作の対象ノード及びプロセスをcontextコマンドにより指定する。また複数のノードをsetコマンドによりセットとして定義することもできる。一方、XIPDではGUI操作により、視覚的にデバッグ対象ノードを指定できる。図 2.8.2において、デバッグの対象は枠付きのプロセッサアイコンで示されている。また、各ノードの状態もその形状から把握できる。デバッグの対象の変更はプロセッサのアイコンを操作することによって可能である。

並列プログラムのロードに対してもXIPDは視覚的操作を提供している（図 2.8.3）。ユーザは、ロードするノードがメッシュ状にグラフィカルに表示された画面上で、ロードするパーティションやロードオプションを指定することが可能である（図 2.8.4）。

XIPDではNode Viewpoint上でノードをクリックすることによりそのノードが実行しているルーチン名を表示できる。さらに、ソースプログラム上で実行箇所をトレースする機能も備わっている。

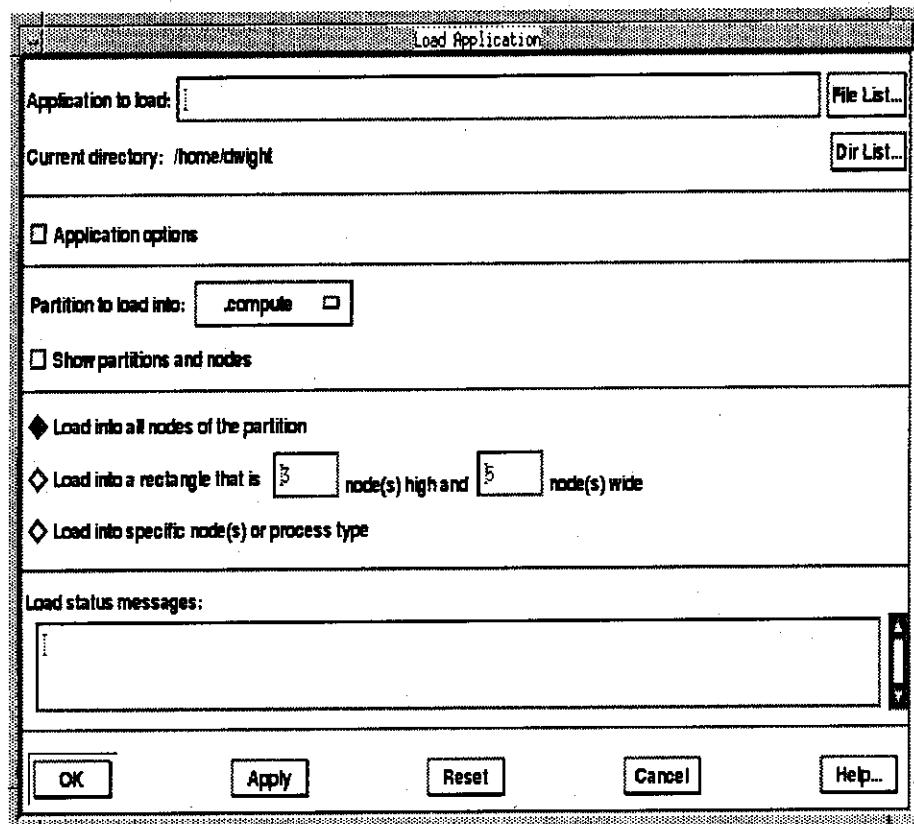


図 2.8.3 Load Application のダイアログ

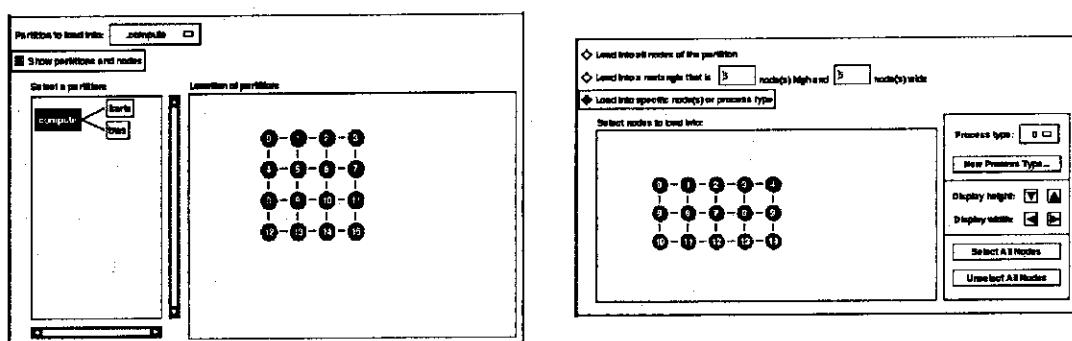


図 2.8.4 パーティション及びロードオプションの選択

プログラム実行中の状態表示には以下のものがある。

- ・送信待ちメッセージの状態表示・・・発信ノード,宛先ノード,長さ,タイプの表示
- ・受信待ちメッセージの状態表示・・・発信ノード,長さ,タイプの表示
- ・トレースバックの表示・・・ノード毎にルーチンコール階層を表示

2.8.3.3 性能評価ツール

Paragon XPでは性能評価ツールとして, prof(XProf), gprof(XGprof), ParaGrafの使用が可能である。

profでは平面的に表示する実行プロファイルしか得ることができないが, gprofでは平面的な実行プロファイル以外に, サブルーチンコールの親子関係を階層的に表示するコールグラフプロファイルを得ることができる。

さらに, ParaGrafでは, グラフィカルな表示により挙動を把握することができる。

(1)prof, xprof及びGProf, XGprof

prof,xprofではサブルーチン毎のCPU時間及び利用率, コール回数, 1コール当たりの平均CPU時間の情報を得ることができる。GProf, XGprofでは, 項目はprof, Xprofの情報と同じであるが, 表示方法においてサブルーチンコールの親子関係を含んでいる点が異なる。

プロファイルは並列に実行される各プロセス毎に作成され, 標準状態ではプログラム実行ディレクトリ下のprof, XProfでは"mon.out", GProf, XGprofでは"gmon.out"というディレクトリ下に格納される。プロファイル名は"executable_name.pid.node.ptype"という命名規則に則り作成される。executable_nameは実行プログラム名, pidはプロセスID, nodeはノード番号, ptypeはプロセスタイプである。prof, GProfを起動時にこのプロファイル名を指定しなければpid, node, ptypeとも最も小さいものを仮定する。executable_nameも指定しなければa.outが仮定される。

XProf, XGprofではプロファイル名指定の煩わしさが改善されており, 起動後にnode, ptype, pidとともに一覧表示された中からマウスにより選択が可能となっている。

(2)ParaGraph

ParaGraphはprof, XProf, gprof, XGprofに比べ, 使い勝手とプロファイル情報の種類が大幅に改善されている。利用には, Xwindowが前提となっている。

(a)Utilization Display

通信などの待ちによるIdle time, SystemタスクによるOverhead time, 入出力によるI/O time, パフォーマンスマニタの情報書き出しによるFlush time, 及びアプリケーションの計算によるBusy timeの5つのステータスについて解析する。

(b)Communication Display

以下のようなメッセージの通信状況の解析をする。

Traffic: メッセージトラフィックの時間推移

Spacetime: プロセッサの状態とメッセージの交換状況の時間推移

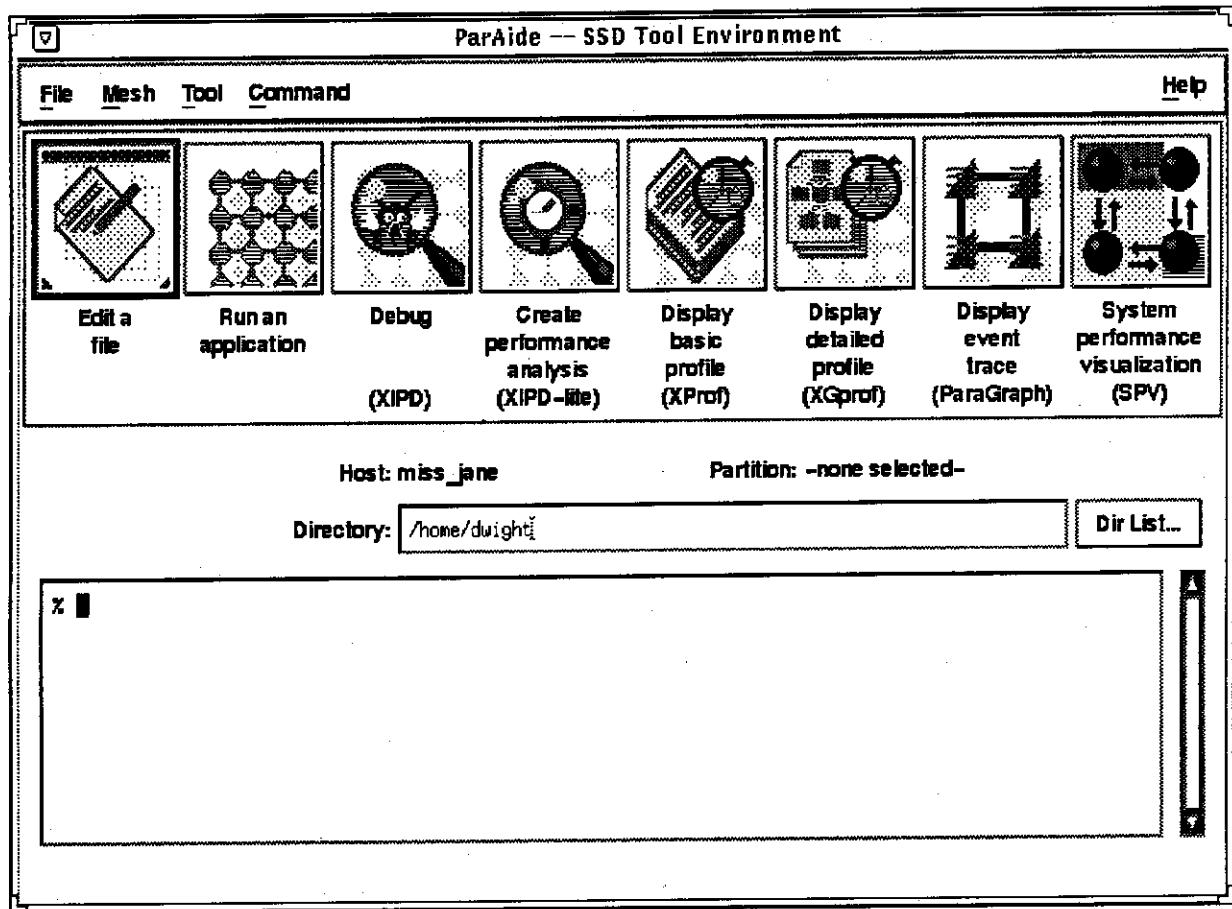


図 2.8.5 ParAide のメインウィンドウ

Queues: ある時間における各プロセッサのメッセージキューの状態

Matrix: ある時間におけるメッセージの送受信状態

Communication Mater: ある時間におけるメッセージトラフィック

Animation / Topology: ある時間におけるプロセッサの状態及びメッセージの状態

Network: ある時間におけるネットワークのトラフィックの状態

Node Info: あるノードにおけるメッセージの状態の時間推移

(c) Task Display

ユーザプログラム中に記述したtraceblockbegin(i)から traceblockend(i)までの通過状態を解析する。これをタスクという。iはタスク識別のためのタスク番号である。表示時に64カラーしかないため、それ以上の番号のものには同一色が使われる。表示内容は以下の通り。

Task Count: 全実行を通してのタスクの数を示す。

Task Gantt: 各プロセッサのタスクの時間推移

Task Status: ある時間におけるタスクの開始・終了の状態

Task Summary: 各タスクの全プロセッサ中で最初の開始から最後の終了までの時間

2.8.3.4 統合化ツール

Paragonにはツール統合環境としてParAideがある(図2.8.5)。ただし、原研では利用でき

ない。ParAideは単に種々のアプリケーションを起動するだけのツールであり、ツール間で情報をやり取りし、開発の支援をするような連携機能はない。

上段に配置された各アイコンをクリックすることにより、対応するツールが起動する。各アイコンは左より、エディタ(vi, Emacsなど指定可能)の起動、ユーザアプリケーションの起動、XIPDの起動、XIPDの起動(性能モニタ機能の設定画面として)、Xprofの起動、XGprofの起動、ParaGraphの起動、SPVの起動となる。

2.9 各社並列プログラミング環境のまとめ

表2.9.1, 2.9.2(a), (b)に、各ベンダのハードウェア（原研に設置されたものに限る）及びソフトウェア（並列プログラミング環境に限る）を示す。原研に数種類存在するVPPシリーズについては、東京に設置されたVPP300を示した。本表に示した各項目については、次のとおり。

2.9.1 ハードウェア

(1) 演算処理方式

ベクトル・プロセッサを用いた並列処理、スカラ・プロセッサを用いた並列処理のいずれかを示した。

(2) メモリ構造

NEC SX-4以外の計算機については、分散、共有のいずれかを示した。

NEC SX-4は、2.3節で述べたように、原研に設置されたものについては、分散記憶を持つ3ノードで構成され、各ノードは、主記憶を共有する2台のベクトル・プロセッサから構成される。

(3) ネットワーク結合

NEC SX-4については、HIPPI switchによる完全結合。IBM SPについては、High

表2.9.1 ハードウェアアーキテクチャの比較

	演算処理方式	メモリ構造	ネットワーク構造	メモリ容量(MB)	プロセッサ数	転送速度(GByte/sec.)	演算性能(GFLOPS)
FUJITSU VPP300	ベクトル並列	分散	クロスバー結合	512	16	18.2 1.14	2.2 35.2
NEC SX-4	ベクトル並列	分散共有	HIPPI結合	512	6	32 0.1	2.0 12.0
HITACHI SR2201	スカラ並列	分散	クロスバー結合	256	64	1.2 0.3	0.3 19.2
Cray T94	ベクトル並列	共有	なし	1000	4	128 なし	1.8 7.2
IBM SP	スカラ並列	分散	HPS結合	64	48	1.07 0.04	0.266 12.8
Intel Paragon	スカラ並列	分散	2次元メッシュ結合	32	256	不明 0.2	0.075 19.2

Performance Switch(HPS)と名付けた独自のネットワーク結合.

(4) メモリ容量

分散メモリシステムについては、1プロセッサあたりの局所記憶の容量を、共有メモリシステムについては、主記憶の容量を示した。

(5) プロセッサ数

原研に設置された並列計算機のプロセッサ数。

(6) 転送速度

上段は、CPU・主記憶間転送速度、即ち、各プロセッサが共有／分散メモリ上のデータを読み出す速度。下段は、プロセッサ間転送速度（NEC SX-4の場合、ノード間データ転送速度、Cray T94の場合、共有メモリのため該当値無し）。

(7) 演算性能

上段は、1プロセッサあたりの演算性能、下段は、システム全体の演算性能。ベクトル並列計算機の場合、ベクトル処理による理論最高性能を示した。

2.9.2 ソフトウェア

(1) 標準装備エディタ

各ベンダが提供するエディタのみを示した。ユーザがここに示したもの以外のエディタをインストールすることは基本的に可能。例えば、VPP300上にemacsをインストールすることは、UNIX環境に詳しい人間であれば、容易に実現できる。ただし、この場合、emacsは、富士通によってサポートされない。

(2) 並列処理用言語

・言語種別

を付けたものは、並列処理用に拡張されたもの。指示行を挿入することで並列処理が実現できる。Cray T94については、すべて。*の無いものについては、通信ライブラリ（コンバイラは外部関数として認識）を用いて並列処理を実現する。

・並列化方式

各計算機アーキテクチャに適した並列化方式を記載した。その他の方式についても、プログラミングは可能。

・通信ライブラリ

エディタ同様、各ベンダによって保守・支援されているものを記載した。例えば、Paragon上では、ユーザーがPVMライブラリをインストールした結果、当該プログラムは動作しているが、Intel社はサポートしていない。

・最適化内容

すべてのベンダが、逐次計算における最適化を行っている。本表では、自動ベクトル化、自動並列化機能の有無を示した。しかし、そのレベルは様々。また、コンバイラだけでなく、プリプロセッサ使用による最適化も含む。

(3) 統合されたツール群の機能

表2.9.2 (a) ソフトウェア・アーキテクチャのまとめ (1)

標準装備 エディタ	並列処理用言語					統合された ツール群の 機能
		言語種別	並列化方式	通信ライブラリ	最適化内容	
FUJITSU VPP500	vi	Fortran90*, Fortran77, C, C++	データパラレル	PARAMACS, MPI, PVM	自動ベクトル化	コンパイル デバッグ 性能評価
NEC SX-4	vi, emacs	Fortran90*, Fortran77, C, C++	共有変数	MPI	自動ベクトル化 (FOPP)	性能評価 編集
HITACHI SR2201	vi	HPF*, Fortran90, C, C++	メッセージ パッシング	PARALLELWARE, MPI, PVM	自動並列化 (HPF)	性能評価 編集
Cray T94	vi, emacs, Xbrowse	Fortran90*, Fortran77*, C*, C++*	共有変数	なし	自動ベクトル化 自動並列化	コンパイル 編集
IBM SP	vi	Fortran90, Fortran77, C, C++	メッセージ パッシング	MPI, MPL, PVMe	なし	なし
Intel Paragon	vi	Fortran77, C, C++	メッセージ パッシング	NX, MPI	なし	性能評価 編集 デバッグ

表2.9.2 (b) ソフトウェア・アーキテクチャのまとめ

	デバッガ			性能評価ツール		
	ツール名	ノード指定	通信デバッグ	ツール名	モニタリング 方式	出力モード
FUJITSU VPP300	fdb	不可	不可	Visual Analyzer	ハードウェア	post-mortem
NEC SX-4	dbx	不可	不可	ANALYZER-P/SX	ハードウェア	post-mortem
HITACHI SR2201	ndb, xndb	可	可	etool, ctool, xtool	ソフトウェア	post-mortem
Cray T94	Total View	可	—	FLOW VIEW PERF VIEW	ハードウェア ソフトウェア	post-mortem
IBM SP	pdbx, xpdbx	可	可	prof, xprof, vt	ハードウェア	real time
Intel Paragon	IPD, XIPD	可	可	prof, gprof, paragraph	ソフトウェア	post-mortem

Paragonには、ユーザーが開発（そこで編集した）プログラムを統合環境から起動できる機能も有り。

(4) デバッガ

- ・ツール名

NEC SX-4及びIBM SP上のデバッガは、サードパーティによって開発されたものをベースにしている。その名前を記載した。その他については、ベンダ固有のデバッガを記載した。

(5) 性能評価ツール

- ・ツール名

代表的ツールについて、その名前を記載した。

- ・モニタリング方式

Cray T94及びIBM SPの場合、ツールによって、解析データ収集機構が異なる。

- ・可視化機能

1例を示した。詳しくは、2.2～2.8節参照

参考文献

- [1] K. Hwang : *Advanced Computer Architecture : Parallelism Scalability Programmability*, McGraw-Hill (1993).
- [2] P. Hatcher, M. Quinn : *Data-Parallel Programming in MIMD Computers*, MIT Press (1991).
- [3] C. McDowell, D. Helmbold : *Debugging Concurrent Programs*, ACM Computing Surveys, Vol.21, No.4, pp.593-622, (1989).
- [4] in WWW Home Page : http://www.fujitsu.co.jp/hypertext/Products/Info_process/hpc/vx/index.html
- [5] in WWW Home Page : <http://guide.tokai.jaeri.go.jp/jpn/computer/vpp500/index.html>.
- [6] UXP/V Fortran90/VPP 使用手引書, J2U5-0080-01, 富士通株式会社 (1995).
- [7] UXP/V アナライザ使用手引書, J2U5-0130-01, 富士通株式会社 (1995).
- [8] VPP ワークベンチ使用手引書, J2S1-0660-01, 富士通株式会社 (1995).
- [9] SX-4 シリーズ システム概説書, GUZ 4II, 日本電気株式会社 (1995).
- [10] SUPER-UX FORTRAN90/SX プログラミングの手引き, G1AF07-2, 日本電気株式会社 (1995).
- [11] SUPER-UX DBX 利用の手引き, G1AF19-2, 日本電気株式会社 (1995).
- [12] SUPER-UX ANALYZER·P/SX 利用の手引き, G1AF15-2, 日本電気株式会社 (1995).
- [13] SUPER-UX PARALLELIZER/SX 利用の手引き, G1AF17-2, 日本電気株式会社 (1995).
- [14] in WWW Home Page : <http://www.hitachi.co.jp/Prod/comp/hpc/jpn/sr2.html>.
- [15] HI-UX/MPP PARALLELWARE ユーザーズガイド ·FORTRAN-, 6A20-3-400, (株) 日立製作所 (1996).
- [16] HI-UX/MPP PARALLELWARE リファレンス ·FORTRAN-, 6A20-3-401, (株) 日立製作所 (1996).
- [17] HI-UX/MPP PARALLELWARE ユーザーズガイド ·C-, 6A20-3-402, (株) 日立製作所 (1996).
- [18] HI-UX/MPP PARALLELWARE リファレンス ·C-, 6A20-3-403, (株) 日立製作所 (1996).
- [19] FORGE Technical Note on Parallel Program Execution, APR (1996).
- [20] FORGE HPF Parallelizer XHPF2.0 User's Guide, APR (1996).
- [21] FORGE Explorer DM Parallelizer Tool User's Guide, APR (1996).
- [22] FORGE Explorer User's Guide, APR (1996).
- [23] Software Overview for Users, SG2052 Ver.9.0, Cray Research, Inc. (1996).

- [24] CF77 コンパイルシステム Vol.4 : 並列処理, JSG-3074 Ver.5.0, Cray Research Japan Ltd. (1991) .
- [25] CF90 コマンドおよびディレクティブレファレンスマニュアル, JSR-3901 Ver. 1.0, Cray Research Japan Ltd. (1993) .
- [26] Introducing the Cray Total View Debugger, IN-2502 Ver.1.2, Cray Research, Inc. (1994) .
- [27] Tuning Guide to Parallel Vector Applications, SG-2182 Ver.1.3, Cray Research, Inc. (1994) .
- [28] Program Browser (xbrowse) , IN-2140, Cray Research, Inc. (1995) .
- [29] IBM XL FORTRAN COMPILER V.3.2 FOR AIX LANGUAGE REF., SC09-1611, 日本IBM (1994) .
- [30] C FOR AIX V.3.1.1 LANGUAGE REFERENCE , SC09-1975, 日本IBM (1994) .
- [31] PARALLEL ENVIRONMENT FOR AIX : MPL PROGRAMMING SUBROUTINE REFERENCE, GC23-3893, 日本IBM (1995) .
- [32] 青山幸也 : RS/6000 SP 並列プログラミング虎の巻 MPI 版, 私信 (1996) .
- [33] AIX 並列処理環境 操作および使用法, 日本IBM (1995) .
- [34] in WWW Home Page : <http://www.ssd.intel.com/paragon.html> .
- [35] Paragon Application Tools User's Guide, 312545-113, Intel (1994) ..
- [36] J. Hennessy, D. Patterson. 富田真治, 村上和彰, 新賽治男訳: コンピュータ・アーキテクチャ, 設計・実現・評価の定量的アプローチ, 日経BP出版センター

3. 並列プログラム開発支援ツールの研究動向

2章では、現在日本原子力研究所に導入されている並列計算機に提供されている並列プログラム開発支援ツールについて述べた。並列計算機が商用ベースで本格的に提供され始めたのはここ数年であり、十分な並列プログラム開発環境が整っているとはいえず、解決すべき問題が山積している。現在、それらの問題点を解決するための研究が活発に行われている。

本章では、並列プログラム開発支援ツールに関する研究動向をまとめる。現在の商用レベルのツールを記述した2章と研究レベルのツールを記述した本章との比較を容易にするために、構成は2章と同様にした。すなわち、最初に並列記述言語及びコンパイラ、デバッガ、性能評価ツールという個々のツールに対する研究アプローチについて述べる。最後にこれらのツールを統合し並列プログラム開発全体を一貫して支援するための統合環境構築というアプローチについて述べる。

3.1 並列処理言語及びコンパイラ

2章で述べたように、並列プログラミング言語は、並列プログラミングモデルに基づいて設計されている。並列プログラミングモデルとは、並列処理をどのように捉えるかという考え方の枠組みであり、並列計算機システム及びその上での並列処理を抽象化したものと捉えることができる。

並列プログラミングモデルは、以下のように大別することができる[1]。

(1) 共有変数モデル

共有メモリ型並列計算機を抽象化したもので、共有メモリ上の大域的なデータを複数のプロセスがアクセスしながら並列計算が進行すると考えるモデルである。

(2) メッセージ・パッシング・モデル

分散メモリ型並列計算機を抽象化したもので、各プロセスが各自局所的な記憶域を持ち、プロセス間で必要なデータを通信により授受しながら並列に計算を行うと考えるモデルである。

(3) データ・パラレル・モデル

SIMD型並列計算機を抽象化したもので、計算プロセスと制御プロセスの二種類のプロセスにより並列計算が進行すると考えるモデルである。計算プロセスは制御プロセスの制御下で互いに同期をとりながら同一の処理を異なるデータに対し行う。制御プロセスは、計算プロセスの同期制御を行うほか、処理中の逐次部分の実行を担当する。

(4) オブジェクト指向モデル

データ及び手続きを一体化したオブジェクトと呼ばれる計算主体が、動的に生成消滅し、互いにメッセージを送受信し合うことにより並列計算が進行すると考えるモデルである。

メッセージ・パッシング・モデルに類似しているが、メッセージ・パッシング・モデルは計算主体(プロセス)の動作が固定的であり、定められた動作中にデータを送受信しながら計算を進行していくのに対し、オブジェクト指向モデルでは計算主体(オブジェクト)の動作がそのオブジェクトの持つデータに対して複数用意されており、そのどれを実行するか

に関するメッセージを送受信することによって計算が進行すると考える点が大きく異なる。

(5) 関数型モデル及び論理型モデル

関数型モデルは、データに関数を作用させることにより新たなデータを生成し、更にそのデータに対して関数を作用させるということを繰り返すことにより処理が行われると考えるモデルである。ここでいう関数の大きな特徴は、副作用を持たないこと(引数の評価順序に依存せず、常に同一のデータを生成する)である。この特徴により、関数をどのような順序で作用させても結果が同一になることが保証されるため、複数の関数の並列実行を行うことが可能となる。従って、関数型モデルはデータフロー計算機に対し容易に適用することができる。

論理型モデルは、述語論理(命題論理の一種)を評価することにより処理が行われると考えるモデルである。論理型モデルには2種類の並列性が内在する[2]。

(i) AND並列性

プログラムのゴールが幾つかのサブゴールから構成され、かつ、サブゴール間の関係がAND条件になっている場合(すなわち下式のようにサブゴールB, C, Dが全てTRUEの場合にのみAというゴールがTRUEとなるという関係が成立している時)、B, C, Dの評価は並列に実行することができる。

$A \leftarrow B, C, D$

この種の並列性をAND並列性と呼ぶ。

(ii) OR並列性

プログラムのゴールが幾つかのサブゴールから構成され、かつ、サブゴール間の関係がOR条件になっている場合(すなわち下式のようにサブゴールB, Cの少なくとも一方がTRUEの場合にAというゴールがTRUEとなるという関係が成立している時)、B, Cの評価は並列に実行することができる。

$A \leftarrow B$

$A \leftarrow C$

この種の並列性をOR並列性と呼ぶ。

現在これらのモデルに基づいた言語の研究が行われている。本節では現在科学計算に用いられている代表的なモデルである共有変数モデル、メッセージ・パッシング・モデル、データ・パラレル・モデルや徐々に科学計算への利用が考え始められてきたオブジェクト指向モデルに基づく並列処理言語について述べる。NESL[3]などの関数型モデルに基づく関数型言語や、KL1[4]などの論理型モデルに基づく論理型言語については割愛する。

3.1.1 共有変数モデルに基づく並列処理言語

大域的なデータを複数のプロセスがアクセスするという純粋な共有変数モデルに基づいた言語は、主に共有メモリ型並列計算機に実装されている。このような言語の例として、CrayのFortran(CF77, CF90)が挙げられる。CrayのFortranに関する詳細は2章3節を参照されたい。

一方、各プロセスが局所的なデータを持ち、必要に応じて共有記憶域にデータを書き出した

り読み込んだりすることにより大域的なデータアクセスを行うというモデルを表現した言語としてLindaが挙げられる(正確にはLindaはそれ自体では言語とは呼べない。従来の逐次型言語に以下で説明するLindaの6つのオペレーションを追加したものと並列処理言語と捉えるべきである)。以下、Lindaについて述べる。

3.1.1.1 Linda

Linda[5] はエール大学の Dr. Gelernter によって開発された。Linda は以下に述べるようなタプル空間と呼ばれる仮想的な共有メモリモデルを設け、そのメモリモデルに対する操作を行う。共有変数へのアクセス(更新)を排他的に行うこととは、ロックをかけることにより実現できる。逆に考えれば、共有変数を一時的にアクセス空間から取り除けばよい。Linda ではこの考え方を陽に表す方法・モデルを採用している。

Linda における記憶の単位は、論理的な項の組(logical tuple(タプルと略称する))である。タプル空間はその要素であるタプルの集合であり、タプルは論理名と値フィールドのリストで構成される。タプルにはプロセスタブルとデータタブルの 2 種類がある。プロセスタブルは指定された実行ブロックを並列に実行するための能動的なデータであり、実行後データタブルに変化する。一方、データタブルは順序付けられた一連のデータの集合であり、そのままの形で生成・読み出し・消費される受動的データである。

Linda では、このタプル空間の操作を行うために、以下のような 6 種類のオペレーションが用意されている。

- out(t) ; タプル t をタプル空間に追加する。既に同じ名前のものがあれば置き換えられる。
- in(s) ; テンプレート s に一致するタプルを取り出して、タプル空間から削除する。一致するものがなければ待たされる。
- rd(s) ; テンプレート s に一致するタプルを読み出す。タプルはそのままタプル空間に残る。一致するものがなければ待たされる。
- inp(s) ; in(s) の述語バージョンである。テンプレート s に一致するタプルがなければ即座に 0 を返し、一致するものがあれば 1 を返す。
- rdp(s) ; rd(s) の述語バージョンである。テンプレート s に一致するタプルがなければ即座に 0 を返し、一致するものがあれば 1 を返す。
- eval(t) ; タプル t がタプル空間に追加された後に計算・評価されることを除いて、out(t) と同じである。

3.1.2 メッセージ・パッシング・モデルに基づく言語

メッセージ・パッシング・モデルでは、データは全て局所的(所有しているプロセスしか直接にはアクセスできない)であり、またプロセス内の動作は逐次的である。従って、单一のプロセスの動作記述には従来の逐次処理言語で十分である。このため、メッセージ・パッシング・モデルに基づく言語は、従来の逐次処理言語に通信ライブラリを付加することにより構築されることが多い。この逐次型言語に付加される通信ライブラリとして、PVM, MPIが挙げら

れる。それに対し、従来の逐次型言語にメッセージ・パッシングに関する構文拡張を行うことによりメッセージ・パッシングの概念を言語自体に内包した新しい言語の構築も試みられている。そのような例として、FORTRAN Mがある。以下、これらについて述べる。

3.1.2.1 PVM (Parallel Virtual Machine)

PVM[6]は、ネットワークに接続された異機種 UNIX 計算機群を、仮想的な单一の並列計算機として利用することを可能にするソフトウェアシステムである。PVM の開発は、Oak Ridge 国立研究所において、1989 年の夏に開始された。

PVM の下では、ユーザは逐次計算機、並列計算機およびベクトル計算機の集まりを、单一の分散メモリ型計算機(バーチャルマシン)として定義できる。PVM は、バーチャルマシンにおいて自動的にタスクを起動する機能を備え、タスク間の通信および同期機構を提供している。タスクは PVM における計算の単位であり、UNIX におけるプロセスに類似したものである。ユーザは C または Fortran を用いてアプリケーションを記述し、それに PVM ライブラリ関数を埋め込むことにより並列化を行う。PVM は、アプリケーション、マシンおよびネットワークレベルでの異機種間の利用をサポートするため、より問題に適したアーキテクチャを利用することもできる。

以下に PVM ライブラリが提供する機能を列挙する。

- ・タスク制御
 - PVM タスク識別子の取得、PVM タスクの生成・消滅等の機能
- ・情報取得
 - PVM 親タスク識別子の取得、PVM タスク状態の取得、バーチャルマシンの状態取得、バーチャルマシンの特性取得等の機能
- ・実行時設定
 - バーチャルマシンに複数のホストを追加あるいは削除する機能
- ・シグナル
 - PVM タスクにシグナルを送信する機能
- ・エラーメッセージ
 - PVM ライブラリ関数でのエラー発生時、エラー原因の情報(メッセージ)を制御する機能
- ・メッセージバッファ
 - メッセージバッファを切替えながらデータをパックし送信する機能
(アプリケーション内で複数のメッセージバッファを取り扱える)
- ・データのパック
 - 異なるデータ型をもつ複数の変数を一つのメッセージとして送受信するための機能
- ・データの送受信
 - 一対一通信、マルチキャスト機能
 - データのアンパック
 - パックされたデータをもとのプログラム言語の型に戻す機能
 - タスクグループ

PVM タスクを論理的にまとめて一つのタスクとして扱えるようにする機能

3.1.2.2 MPI (Message Passing interface)

MPI(Message Passing Interface)[7] は、40を超える団体の参加を受けて、1995年6月に最終報告第1.0版が提出され、仕様が確定した。

MPI の目標は、メッセージ通信を行うプログラムを書くための、広く一般に使われる標準を作り出すことである。

MPI が目標とした項目は以下のとおりである。

- ・ アプリケーションプログラムから呼ぶため(コンバイラや、システムの一部を実現するライブラリだけが使うのではない)のインターフェースを設計する。
- ・ 効率のよい通信(メモリ間コピーを避ける、計算と通信を並行して進める、通信用コプロセッサがあれば処理の一部をそれに任せる等)を可能にする。
- ・ 異機種環境でも使用できる処理系に備える。
- ・ C言語や Fortran言語のための使い易い呼び出し形式を可能にする。
- ・ 信頼できる通信インターフェース(ユーザは通信障害に対処する必要がない。そのような障害は下位の通信サブシステムが処理する)を想定する。
- ・ PVM, NX, Express, p4などの既存のインターフェースとそれほど違わないインターフェースを定義し、より柔軟な拡張機能を用意する。
- ・ 多くのベンダのプラットフォーム上に実装でき、その際、下位の通信およびシステムソフトウェアに大きな変更を加えずに済むインターフェースを定義する。
- ・ マルチスレッド環境への対応を考慮してインターフェースを設計する。

MPI には、専用のプロセッサ間通信ハードウェアを備えたスケーラブルな並列計算機の性能を高めることを狙いとした多くの機能がある。したがって、このようなマシンには高性能な専用 MPI 処理系が提供されることが期待される。また一方では、UNIXプロセッサ間通信プロトコルを用いる MPI 処理系は、ワークステーションクラスタやワークステーションの異機種間ネットワークの間でのポータビリティを提供する。ベンダ独自の専用 MPI 処理系及びパブリックドメインの移植性の高い MPI 処理系の開発が、既に開始されている。

MPI 標準に含まれる項目は以下のとおりである。

- ・ 一対一通信
基本的な2プロセス間通信(send, receive等)を支援する。
- ・ 集団操作
基本的な多プロセス間通信・操作(broadcast, barrier等)を支援する。
- ・ プロセスグループ
集団操作対象となるプロセスの集合を規定する。
- ・ 通信コンテキスト
タグを用いてメッセージを識別する。
- ・ プロセストポロジー

論理的通信パターンを反映したもの(例えば格子状)にプロセスグループを配置する。

- ・FORTRAN77言語とC言語の呼び出し形式
関数インターフェース、定義済定数の規約を定める。
- ・環境管理と問い合わせ
実行環境に関連するパラメータの取得・設定を可能にする。
- ・プロファイリングインターフェース
プロファイラ作成者のためのインターフェースを定める。

3.1.2.3 FORTRAN M

FORTRAN M[8]はFORTRAN77に対してメッセージ・パッシングに関する構文拡張を行うことでタスクバラレルプログラミングを支援する言語である。

FORTRAN Mでは、プログラムモジュール(サブルーチン)が独立した計算主体(プロセス)として扱われる。プロセスは、チャネルの明示的な宣言にもとづいて通信路を確保し、send/recv文によって通信を行う。プロセスの並列実行はprocessあるいはprocessdoブロックによって制御される。

プロセス間の通信、プロセスやチャネルの形成・消滅は動的に行うことが可能であるが、決定性が保証されるように文法上の制約が設けられており、デッドロックなどに陥らない安全性を持っている。

また、プロセスの割り当てに関しては仮想計算機を想定して行うのでアーキテクチャに依存しないという利点がある。

近年FORTRAN Mに対し、HPFベースでデータ・バラレル・モデルの概念を導入し、カプセル化された部分問題をHPFで記述、部分問題間の通信に現在のFORTRAN Mの構文を利用するといったハイブリッドな言語処理系の開発[9]が試みられている。

3.1.3 データ・バラレル・モデルに基づく言語

一般に、分散メモリ型計算機上に共有変数モデルに基づく言語を実装する場合には仮想的な共有メモリ空間を構成する。しかしその場合、各プロセスのアクセスを効率よく行うことが困難であるという問題点が存在する。また、メッセージ・パッシング・モデルに基づく言語では、他プロセスの保持する変数へのアクセスを全てユーザが管理しなければならないため、ユーザの負担が大きいという問題点が存在する。一方、データ・バラレル・モデルでは、各プロセスが局所的にデータを保持するという前提にたち、かつ他プロセスの保持する変数へのアクセスは処理系が実現するという長所を持っている。さらに、科学計算では個々のプロセスが異なる処理を行うことは希であり、異なるデータに対して同一の処理を行うというSIMD型のプログラムが多いこともあり、データ・バラレル・モデルは科学計算への適合性が高い。以上の理由から、多くのデータ・バラレル・モデルに基づく並列処理言語が提案されている。

現在最も代表的なデータ・バラレル・モデルに基づく言語としては、Fortran90及びHPFが

挙げられる。これらについて以下に述べる。

3.1.3.1 Fortran90

Fortran90[10]は、Fortranの現行規格である。Fortran90の特徴は、配列演算や配列計算用組込み関数が大幅に強化されている点である。これにより、従来DOループで記述していた処理を、1つの配列演算命令で記述できる。この配列演算命令は、配列要素に独立であり、ベクトル化だけでなく、並列化もデータフロー解析なしに可能である。

また、Fortran90の後継規格としてFortran95の規格化が進んでいる。Fortran95の特徴は、次項で述べるHPFの中から並列実行構文"FOR ALL",及び属性"PURE"が言語規格として採用されている点である。"FOR ALL"及び"PURE"に関してはHPFの項で述べる。Fortran95は、96年末から97年初めに規格化される見込みである。

3.1.3.2 HPF (High Performance Fortran)

HPF[11]はFORTRAN DやVienna FORTRANの研究で得られた成果に基づいて構築された言語であり、Fortran90をベースとしている。現在、Ver.1の仕様が決定している状況で、Fortran95をベースとするVer.2が検討されている。以下にHPFの特徴を述べる。

(1) データ分割機能

配列データを分割しプロセッサに割り付けるマッピングを行うために、以下の3種類の機能／指示が定義されている。

(i) データの区分 (DISTRIBUTE)

マッピングを実現するためには、まず対象とする配列をどのようなパターンで分割するかを指定する必要がある。DISTRIBUTE指示文はこの分割パターンを指定する。分割パターンには、CYCLIC及びBLOCKが存在する。2つの分割パターンを図3.1.1に示す。

分割はあくまでも論理的なN個のプロセッサを対象として記述され、実際に使用する物理的なプロセッサの数には影響を受けない。これによってプログラムの一般性が保証される。

(ii) データの併存 (ALIGN)

次に、最終的な配列のマッピングを行うためには複数のデータ実体間のマッピングの関係を示す必要がある。ALIGN指示文はこの複数の配列のマッピングの関係を指定する。DISTRIBUTE指示文、ALIGN指示文を用いた配列の分割例を図3.1.2に示す。

(iii) テンプレート (TEMPLATE)

ALIGN指示文では実際に使用する複数の配列間のマッピングの関係を指定するだけでなく、仮想的な配列を基準としてその配列との関係を指定することもできる。TEMPLATE指示文では、この仮想的な配列の名前、大きさ、分割パターンなどを指定する。ここで指定した名前を、ALIGN指示文で使用でき、またその後で分割もできる。

これらの機能によって指定された分割に従い、HPF処理系は配列データを物理処理系（計算機）にマッピングする。

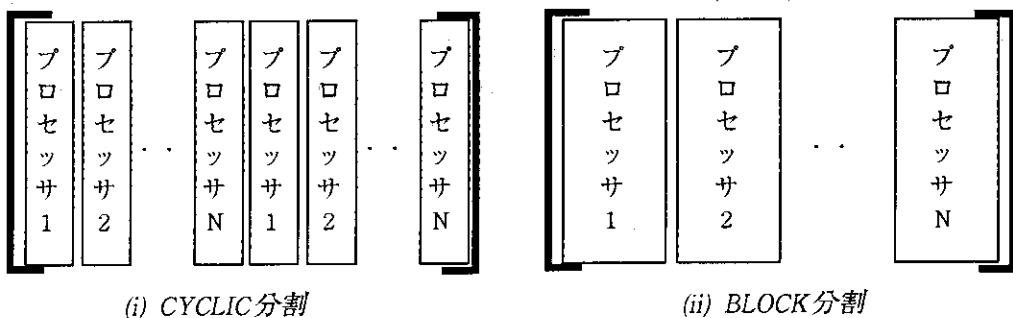


図 3.1.1 配列データの分割パターン

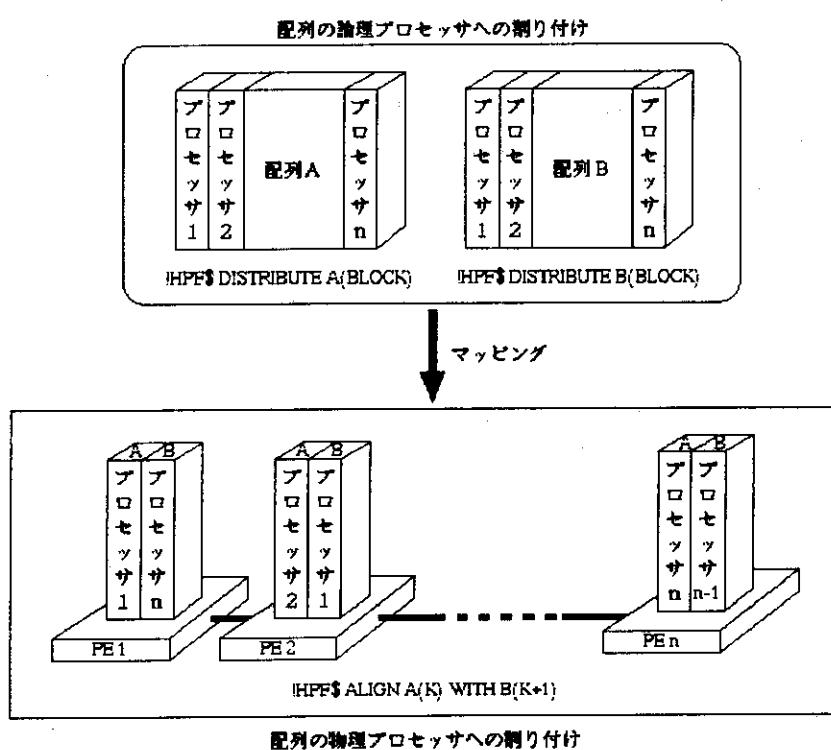


図 3.1.2 配列データのプロセッサへの割り付け

(2) データ並列実行機能

以下の構文及び指示文により、並列実行を行うループを明確に定義できる。

(i)FOR ALL構文

並列実行のためのDO構文である。ただし従来のDOループと異なり、FOR ALLループ内では、後述の純手続き、純関数のみが呼び出し可能であるという制限が存在する。これは呼び出し順序により関数や手続きの値が変化するという副作用により、計算結果が実行毎に異なるという現象が発生するのを防止するためである。

(ii)INDEPENDENT指示文

DOループや、FOR ALLの前に指定し、その中の演算や繰り返しを任意の順序で実行しても意味(結果)が変わらないことを示す。INDEPENDENT指示文で指定されたループ内では

任意の関数、手続きを呼び出すことが可能である。

FOR ALLで指定された並列実行ループ(FORALLループと呼ぶ)とINDEPENDENT指示文で指定された並列実行ループ(INDEPENDENTループ)では並列処理方式が異なる点に注意する必要がある。対象とするループを並列実行する際、両ループとも最初各プロセッサにループ実行に必要なデータが分配される。FOR ALLループの場合、ループ実行においてデータが更新されると、その命令実行後に再分配を行うため、命令単位で同期が発生する。それに対しINDEPENDENTループの場合は、ループ内の命令が全て終了するまでデータの再分配は行われず全ての命令が非同期に実行されることにより、並列実行効率は高い。しかしながら、データ間にデータ依存性が存在すると実行毎に計算結果が異なる可能性があるため、データ依存性が存在するループではFOR ALL構文を用いて並列実行を行うべきである。

(3) PURE属性の宣言

手続きあるいは関数がPURE属性を持つと宣言することで、その手続き（純関数）が副作用を生成しないことを保証する。PURE属性を持つ手続き、関数を、純手続き（純関数）と呼ぶ。純手続き（純関数）は、FORALL構文中で使用できる。

(4) 組込み関数、標準ライブラリの拡張

プロセッサ数や、マッピングを問い合わせる関数や、配列演算を行う関数を追加している。

(5) EXTRINSIC手続き(外部手続き)

HPF以外の言語(CやFortran)で記述された手続きを呼び出す機能を追加している。HPFはデータ・パラレル・モデルに基づく言語であるため、データは各プロセッサに分割されている。一方HPFプログラムから呼び出される外部手続きは逐次型言語で書かれているため、単純に外部手続きを呼び出すことはできない。そのため、外部手続きで使用されるデータの分割情報をインターフェースとして規定することにより、逐次型言語で記述された外部手続きの呼び出しを可能としている。

3.1.4 並列オブジェクト指向モデルに基づく言語

オブジェクト指向という概念は、もともとオブジェクトというものがデータと手続きをカプセル化した(データとそのデータを操作するための手続きを一体化させること)独立性の高い計算主体であり、計算がそれらの間のメッセージ・パッシングにより進行するという考え方であるため、並列処理との親和性が高い。そのためオブジェクト指向概念が普及するにつれて、その代表的な言語であるC++を並列処理に拡張しようと試みが行われてきた。まだ他のモデルに基づく言語と比較して科学計算に利用される頻度は低いが、今後の科学計算への適用が期待される。本節では、その代表的な例としてHPC++, CC++について述べる。

3.1.4.1 HPC++ (High Performance C++)

HPC++ [12] にはLevel 1とLevel 2がある。Level 1は並列ループのディレクティブやテンプレートライブラリなどからなり、Level 2では言語仕様レベルでの追加がなされる。

3.1.4.2 CC++ (Compositional C++)

CC++ [13] は C++ に並列機能を追加した言語であり、メソッド(メッセージによって起動される手続き)内の命令ブロックの並列実行(par), ループの並列実行(parfor), インスタンスの生成による並列実行(spawn)の 3 つの並列制御の構造を使用して、並列計算を表現する。

実行順序の制御には、同期変数(syncで宣言する)を参照する方法と、アトミック関数(atomicで宣言する)を使用して並列ブロック中で同じ関数が同時に実行されないようにする方法の 2 つの方法がある。

CC++ は、Solaris, AIX (on IBM RS6000 and IBM SP2), SunOS や HP/UX 上で動作する。

3.1.5 並列化コンパイラ

逐次プログラムの並列化方針は、対象とする計算機が分散メモリシステムか共有メモリシステムかによって、大きな違いがある[14]。

共有メモリシステムに対しては、ベクトル化の場合と同様な手法と変換を使うことができる。この場合の目標は、ループの異なった繰り返しをプロセッサに分散させることである。

しかし、並列化によって、スケジューリングと同期オーバヘッドが生じる。したがって、ベクトル化の場合とは異なって、1 ステートメントのみの並列化は採用せず、最大部分の並列化を目指す。

分散メモリシステムに対する並列化は、基本的に異なった方法をとる。これはプロセッサ間の通信が非常に高価だからである。逐次プログラムのデータ領域をユーザの指示に従って分割するような方法を導入する。データ領域の部分をプロセッサのローカルメモリに格納する。分割の指定はプログラムの部分動作に従って、通信を最小化するようにしなければならない。並列化コンパイラ(再構成システム)は、各プロセッサが自分のローカルデータにアクセスするだけですむことを保証しなければならない。そして、必要に応じてプロセッサ間の同期と通信を挿入しなければならない。

以下に並列化コンパイラ(再構成ツール)の事例を掲げる。

3.1.5.1 Parafrase と Parafrase-2

Parafrase [15] は逐次 Fortran プログラムを多様なアーキテクチャを対象に、並列プログラムに変換するために開発された最初の再構成ツールである。Parafrase は、D. Kuck らによつて、1970 年からイリノイ大学で開発が始められ、その後自動ベクトル化・並列化コンパイラに大きな影響を与えた。

Parafrase はソースからソースへの変換システムであり、Fortran 66 または Fortran 77 プログラムから、特定の計算機を対象とした拡張 Fortran プログラムへ変換する。このシステムは、異なるアーキテクチャ(ベクトル計算機、アレイプロセッサ、共有メモリマルチプロセッサ)を対象として、変換をすることができる。

Parafrase の入力は、逐次プログラムとバスリスト(pass list)である。バスリストはバス名と

関連するスイッチの設定を指定するもので、行うべき処理を定義する。パス名はそれぞれプログラムを読み/書きするプロセッサを識別する。パスはリストに指定された順に適用されて、それぞれソース・ソースの変換を行う。特定のアーキテクチャを指向して最適化されたバスリストを作ることができ、また新しいバスを挿入することによってシステムを拡張することもできる。スイッチと呼ばれるグローバルおよびローカル(バスに関連する)パラメータが、システムの実行を制御する。

Parafrase には 3 つの最適化が用意されている。これらの最適化は計算機への依存度が異なる。アーキテクチャ独立バス(architecture-independent pass)は、一般的な並列計算機を想定した解析と最適化を行う。中間バス(intermediate pass)では、指定されたアーキテクチャに固有の要求条件に対する処理を行う。最後に、特定の計算機に依存したバスがあり、指定された特定の計算機を狙った変換を行う。

また、Parafrase ではユーザがアサーション(assertion)またはコマンド(command)といった手段で情報を与えることができる。これらの指示はソースプログラムに埋め込まれ、(a) プログラム変数の値を指定する、(b) 文の集合の並列実行を指定する、(c) ループや逐次化を指定する、(d) 分岐やプロセッサコールなどをもつループの逐次化を指定する、ことができる。

Parafrase-2 [16] は Fortran のほかに C, Pascal などで書かれたプログラムも対象とする Parafrase の拡張版である。

3.1.5.2 PFC と PFC⁺

PFC[17](parallel Fortran converter)は、Fortran 66 または 77 を Fortran 90 に変換する、Parafraseをベースとした自動ソース・ソースペクトライザである。PFC は 1979 年以来、K. Kennedy らがライス大学で開発を行っている。

初期には、PFC はソースプログラムを通常のコンバイラが使っているような内部表現(抽象的なシンタックスツリーと付随するシンボルテーブルに基づく)に変換するものであった。最初の抽象シンタックスツリーを、PFC は次の 4 ステップ(a) プロセッサ間解析、(b) 標準変換、(c) 依存解析、(d) ベクトルコード生成、にわたって変換する。

PFC⁺ は PFC の最新版であり、共有メモリシステムのための並列コード生成アルゴリズム、プロセッサ間依存試験も組み込まれている。

3.1.5.3 SUPERB

SUPERB[18] は 1985 年から 1989 年にわたって、ボン大学の H. Zima のグループによって開発されたソース間変換の対話型再構成ツールである。Fortran 77 プログラムを SUPRENUM 計算機の並列 SUPRENUM Fortran に変換する。

SUPERB のフロントエンド(front end)は Fortran 77 プログラムを、属性付き抽象ツリー、シンボルテーブル、フローグラフおよび初期データフロー情報からなる内部表現に変換する。SUPERB の中核は、解析プログラムと変換カタログとが組み合わされたルーチンの集合を含んでいる。これらのルーチンは内部表現を処理する。解析部(analysis component)はプログラムフ

ローおよび依存解析のためのツール群を提供する。解析プログラムの主要な処理は、(a) 変換に必要な前提条件の成立の有無の確認、および正当な変換からの選択の基準の確立、(b) 変換仮定が終了した後で、後処理をするために必要な情報の用意、(c) ユーザに対するプログラムに関する詳しい情報の提供、である。分散メモリ計算機では、並列化はどうしても対話型に行わざるを得ないので、最後の項目は重要となる。

再構成過程の中でプログラムの内部表現が正規化されて、その後に行われる変換を簡単かつ効率のよいものとする。これには、DOループ正規化やIF変換が含まれる。分散メモリ計算機のアーキテクチャ(基本プロセッキングユニットにバイオペレータユニットをもつ場合)に対応して、変換は2レベルで行われる。すなわち、まず粗粒度並列性を検出し、プロセスをプロセッサに分散する。次にプロセスコードをベクトル化する。

3.2 デバッガ

3.2.1 並列デバッガの分類

McDowellとHelbold[19]は並列デバッガを分類し、それぞれの得失を詳細に論じている。彼らの分類によれば、並列デバッガは

- (1)ブレークポイントベースデバッガ
- (2)イベントベースデバッガ
- (3)静的解析デバッガ

の3種類に分類される。

ブレークポイントベースデバッガとは、逐次プログラムデバッグにおけるサイクリカルデバッギング(Cyclical Debugging)手法を並列プログラムデバッグに適用したものである。ここで、サイクリカルデバッギング手法とは、プログラム上に実行中断点(ブレークポイント)を設定し、何度もプログラムを止めて状態を検証し、その結果によって実行を継続させたり、再実行を行ったりするという動作を繰り返すことによって、徐々にプログラムエラーの発生個所を限定し、原因を追及していく方式のデバッグ手法である。

イベントベースデバッガは、デバッグ作業を

- ・通信状況などプログラムの実行に関する情報を収集するモニタリングフェーズ
- ・収集された情報に基づいて実行を再現する再演フェーズ

に分割し、デバッグを支援するものである。モニタリングで記録された情報は再演フェーズにおいて大別して3種類の形態で利用される。

(i)Browsing

収集された履歴を分析する。分析には単にテキストエディタを利用するレベルからデータベース検索システムの利用まで考えられる。

(ii)Replay

イベント履歴を用いてプログラムの再実行を行う。この形態は実行制御情報としてイベント履歴が用いられるため、再演の段階ではプローブ効果の存在しない再実行が保証される。従って、プログラムの状態を変化させないブレークポイントの設定やステップ実行が

可能である。

(iii) Simulation

並列実行を行うプロセスのうち、任意の単一のプロセスの実行環境をシミュレートするためにイベント履歴を用いる。この形態には、利用者の使いなれた逐次プロセスのデバッグスタイルをそのまま保持できるという利点がある。

静的解析デバッガは、上記2種類のデバッガと異なりプログラムの実行を行う必要がない。プログラムの実行経路を解析し、全ての可能な並列実行状態を考慮して同期エラー(デッドロック、永久待ち)やデータ利用エラー(共有変数の更新の競合)の可能性を指摘することを目的としている。

3.2.2 プローブ効果に対するアプローチ

並列プログラムデバッグ時に特有な問題の一つとして、プローブ効果の存在による再現性の欠如が挙げられる。ここで、プローブ効果とは、デバッグ対象としているプログラムにブレークポイントを挿入したりデータを出力するという行為自体がプログラムの実行に影響を及ぼし、目的とするプログラムの挙動が発生しなくなってしまう事をいう。例えば、ある共有変数に複数のプロセスがアクセスする際、ブレークポイントの設定の仕方によってはアクセス順序が変わることが起こりうる。

デバッグにおいて再現性が欠如していると、利用者は目的とするプログラムの挙動を発生させるために多大な時間を費やすことになる。これは行うべき処理を決定し実施した後、計算機から所望のデータを得るまでの広い意味でのターンアラウンドタイムが長くなることを意味している。

この1つめの問題に関して最も深刻な影響を受けるのが、ブレークポイントベースデバッガである。このタイプのデバッガは常にプログラムの実行をその場で行うため、毎回プローブ効果に悩まされる可能性がある。このタイプのデバッガは逐次型デバッガの拡張であるため構築しやすく、また利用者も従来の経験の延長線上で利用できるという特徴を持つため、2章で述べた原研に導入されている並列計算機上のデバッガを含め多数の商用ベースのシステムが構築されている。しかし、上記のような問題点が存在するため、プローブ効果が関係するエラーに対してはほとんど無力である。

イベントベースデバッガでは、モニタリングフェーズにおいて実行をモニタする作業しか行わないため、プローブ効果は最低限に抑えられている。また、一旦現象が発生すればモニタリングによって収集された情報を基に何度もその状況を再現できるという特徴を持っている。

プローブ効果の発生が通信の競合に起因することが多いことから、イベントベースのデバッガでは通信パターンの解析を支援する機能の提供に主眼がおかれてている。例えば、XabやParaGraphでは通信状況をモニタリングし、それをプロセス間通信ダイアグラムとして表示することにより、通信パターンの解析を支援している。

Xab[20]はPVMを利用した並列プログラムの通信状況をモニタリングし、グラフィカルに表示するツールである。Xabではxabと呼ばれるイベントモニタリング用ライブラリを提供して

いる。このライブラリをプログラムにリンクすることにより、PVMによる通信を制御するpvmdからモニタリングプロセスであるabmonに通信状況が渡され記録される。そのモニタリング情報は動的にグラフィカルに表示される。

Xabが動的な通信モニタであったのに対し、ParaGraph[21]は事後解析型のツールであり、モニタ結果を種々のグラフを用いて細かく解析することが可能である。ParagraphはIntel Paragon等種々の並列計算機上で稼働している。

上記のツールは単に通信パターンをグラフィカルに表示するだけの機能しか有していないが、ATEMPT[22]では通信エラーや競合状態を検知し、それを解消するために種々の機能を有している。

ATEMPTはMAD(Monitoring And Debugging)環境を構成するツールであり、並列プロセス間の通信状態の可視化機能を有している。MADはEMUと呼ばれるイベントモニタリングユーティリティ、PARASITと呼ばれるプロセス間の競合状態の検知ツール及びATEMPTから構成される。この環境により実現されている主な機能は次のものである。

(i) 単純な通信エラーの自動検知

ここで言う単純な通信エラーとは、対応するsend命令が存在しないreceive命令や、send命令とreceive命令においてメッセージ長が異なる場合等である。

(ii) 通信グラフとソースコードとの対応付け

ATEMPTでは通信グラフ上の通信パターンをクリックすることにより、その通信に対応するプログラム内の通信命令を表示することが可能である。この機能により通信グラフ上で検知されたメッセージパターンの異常をプログラムに反映させることができくなる。

(iii) 競合状態の検知

並列プログラムでは、複数のプロセスが一つのプロセスにメッセージを送る場合、通信順序が非決定的になる可能性がある。このような状態を競合状態と呼ぶ。プローブ効果の原因は、競合状態の存在である事が多い。PARASITはEMUによって収集された通信情報から競合状態を検知することが可能である。

(iv) 競合状態におけるメッセージ到着順のグラフベースの変更及びその変更順序に従ったプログラムの再実行の支援

ATEMPTでは、競合状態になっているメッセージ通信をハイライト表示することができ、利用者は画面上でそれらの通信順序を変更することができる。また、さらにPARASITと連携することにより、その通信順序にしたがったプログラムの再実行が可能である。この機能により、従来非常に困難であった競合によるエラーの修正が容易になっている。

プローブ効果による再現性の欠如という問題に対する最も有効なアプローチは、静的解析デバッガの利用である。静的解析デバッガではデバッグ作業にプログラムの実行が伴わないのでプローブ効果の影響は皆無であり、また可能な並列実行状態を全て考慮するため、どんなに発生確率の低いエラーも多数の試行を行うことなく検出することができる。問題は静的解析デバッガでは並列実行プロセスのとりうる状態の数が非常に多くなることがあり、計算の複雑さが指数関数的になってしまふことである。そのため、静的解析デバッガは現在のところプロ

トタイプレベルに留まっている。

3.2.3 デバッグ情報提供手法

並列デバッグのもう一つの問題点は、計算機内に格納されているデータをどのように利用者に提供するかである。一般に科学計算では巨大な配列を利用し自然現象のシミュレーションを行っており、並列計算では特にその傾向が顕著である。また、デバッグのための通信データなども大量になりやすい。この多量のデータを利用者に認識しやすいような形態で提供することが重要である。

この問題に関しては、2つのアプローチが存在する。一つは、通信パターンのモニタリングにおける大量の出力結果を如何に理解しやすい形で提供するかという問題に関連するもので、通信パターンの抽象化が主眼となっている。

例えば、BA (Behavioral Abstraction)では、プログラム実行の際に生じるsend, receive等の通信(イベントと呼ぶ)により構成される一連の通信パターンを、より高次のイベントとして定義することができ、それによりイベントの抽象化を行う。この抽象化されたイベントを利用してプログラムの実行を理解することができる。しかし、BAではイベントのモニタリング及び表示しか行えないので、あらかじめ利用者はどのような抽象イベントが生起するかを予期してデバッグを行わなければならない。そのため、サイクリカルデバッグスタイルに合わないという問題点が存在する。今後はイベントの抽象化機能とサイクリカルデバッグスタイルを両立させるツールの開発が望まれる。

上記の問題に関するもう一つのアプローチは、プロセス内に格納されている変数の値をいかに理解しやすい形で提供するかという問題に関連するもので、数値データの可視化に主眼がおかれていている。

例えばCM5におけるプログラミング環境Prism[23]ではプログラム中の変数値をその値に応じた色で表示することができる。また、大量の配列データの興味ある一部分のみを利用者が抽出しやすいう簡易なナビゲーションを行う機能も存在する。

しかし、Prismはプロセス内の変数値をそのまま表示するだけで、利用者の観点から自由に加工して表示する機能を有していない。このような機能の開発を目的として構築されたデバッガとして、並列プログラムデバッグ用可視化ツール[24]が存在する。このツールはデバッガとTcl/Tk[25]と呼ばれる言語用のインタプリタを連携し、デバッガで取得されたプロセス内の変数値をTcl/Tkで記述された可視化プログラムに転送することにより変数の可視化を行うものである。この機能により利用者は任意の観点からデータを解釈することが可能となる。

このツールにも利用者が任意の観点から可視化を行うためにはTcl/Tkという低レベルの言語を用いて可視化プログラムを作成しなければならず、多大な手間が必要であるという問題点が存在する。今後は如何に簡単に利用者が可視化プログラムを作成できる環境を構築するか、並列プログラムのデバッグのためにはどのような可視化手法が有効かなどの検討が必要であると考えられる。

3.3 性能評価／チューニング支援ツール

並列処理のための性能評価あるいはチューニングを支援するツールについての現状技術調査結果を述べる。3.3.1に、既存ツールの問題点、共通点等について述べる。3.3.2～3.3.7に、調査した6ツールの概要及び機能について述べる。

3.3.1 既存ツールに対する考察

(1) 問題点

並列処理の性能向上を目的として開発されてきた性能評価／チューニング支援ツールの問題点は次のとおり。

(i) 超並列計算機に対する適用性

本節で述べるすべてのツールは、各プロセッサの状態表示を行うことが可能である。しかし、超並列計算機においてプロセッサ台数が極端に大きくなった場合、すべてのプロセッサに関する状態表示を一つのウインドウ上で行うことは不可能と思われる。状態の可視化表示において拡大／縮小を行ったところで、どの程度プログラムの（大局的な）挙動解析が可能であろうか。即ち、既存ツールの超並列計算機に対する適用性については疑問を感じた。

(ii) 事後解析

既存のほとんどのツールは、ある入力データを用いた実行結果に基づく、Postmortemな（事後の）、即ち、off-line解析である。これは、on-lineで解析を行った場合、システムに大きな負荷がかかるのを避けるためである。しかしながら、性能評価に限らず、計算機における処理が実時間で行われることが望ましいことは自明であり、思考の流れが途切れるばかりでなく、解析データ・ファイルの管理などユーザにとって煩雑な作業を伴うことは、「事後処理」の問題点である。

(2) 共通点

並列処理の性能向上を目的として、また、上記（1）で述べた問題点に対するアプローチとして開発されてきた性能評価／チューニング支援ツールは、次の共通点を持っている。

(i) 可視化機能

複数プロセッサの情報を同時表示するため、数値情報の羅列は解析困難である。即ち、複数台プロセッサの状態把握には、可視化表示は不可欠である。各ツール共、可視化機能（主に、円／棒グラフ表示）を用いて、ユーザが容易に状態（プログラムの挙動）を把握できるような工夫をしている。

(ii) Xウインドウ・システムの活用

本節で述べるすべての性能評価／チューニング支援ツールは、UNIXの標準的なGUI(Graphical User Interface)であるXウインドウ・システムを用いている。また、マウス及びメニューバーを用いて操作性の向上を図っている。

(iii) on-line 解析

Postmortemな解析を行っている既存のほとんどのツール中、例外として（on-line 解析を行

う例として), 3.3.5節に述べるPATOPが挙げられる。しかしながら, PATOPが出力するジョブ情報は, 各プロセッサのアイドル/ビジー状態等ごく限られた範囲にとどまっている。

3.3.2 VISPAT (VISeualization for Performance Analysis and Tuning)

(1) 概要

VISPAT[26]は, 英国のエジンバラ大学で開発された。PUL 及びCHIMP と呼ばれる通信ライブラリを使用した場合の並列処理を前提としている。複雑・大規模な並列処理における性能評価には, 可視化機能が不可欠ではあるが, それは万能薬ではない, という認識のもと, 利用者を支援するために二つの要件に着目して設計された。一つは, チューニングにおけるボトルネックとソース・コードの関係の明確化である。通常, チューニング作業は, 既に高速化されている部分をさらに高速化することよりも, 性能を阻害している(ボトルネックとなっている)部分をソース・コードにおいて明確に特定することから始まる。即ち, 実例を通して, ボトルネック部分を検知・診断・修復する作業(狭い範囲のチューニング)を効率化するための可視化機能の実現を一つの要件としている。二つめは, チューニング作業における繰り返し作業の効率化である。チューニング作業においては, しばしば, 実行効率を左右するパラメータ(例えば, DOループの繰り返し回数)を探査する。この作業(広い範囲のチューニング)を効率化するための可視化機能の実現を二つめの要件としている。

(2) 機能

(i) データ制御及びフィルタリング

(イ) データ制御

性能評価データの可視化を制御するウインドウは, ナビゲーション・ディスプレイと呼ばれる。この操作画面を用いて, ユーザは興味のあるデータの種類, データ空間における(表示のための)尺度, 時間ステップの長さ等を指定する。図3.3.1にナビゲーション・ディスプレイ

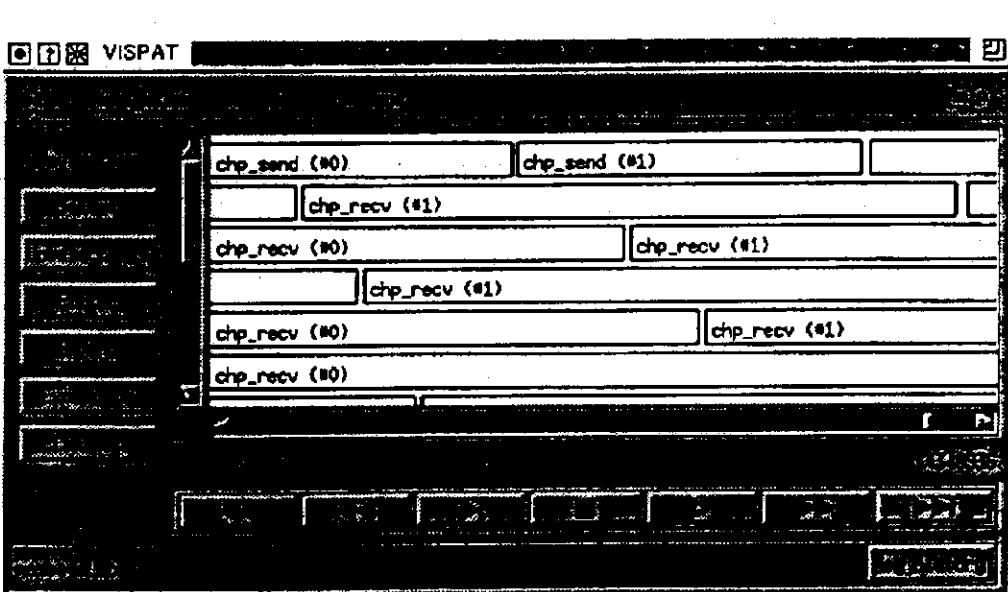


図 3.3.1 VISPAT におけるナビゲーションディスプレイ

の例を示す。

(口) フィルタリング

性能評価データを可視化する場合、ユーザは任意の位相を指定して、その位相に関する各プロセッサの状態を見ることができる。「位相」とは、この場合、計算処理及び通信処理である。後者については、さらに通信ライブラリであるPULを利用した通信及びCHIMPを利用した通信を区別している（異なる位相としている）。

(ii) データの可視化

(イ) 単相表示

ある特定の位相に注目して、各プロセッサの関係を表示するモードである。この機能により、例えば、通信の行われるタイミングを比較することができる。

(口) 動画表示

上記(イ)が、プログラム実行全体にわたって静的な表示を行うのに対し、この機能は2次元（各位相における各プロセッサの関係）イメージの動画表示を行う。

(ハ) 各プロセッサの関係表示

本システムにおいて、プログラム中の処理は機能分類されている（例えば、同期をとる、非同期に実行等）。各分類項目に対する各プロセッサの処理を表示する。

(二) 統計量表示

通信回数や通信量などの並列処理に関する統計量を表示する。量は色によって区別されており、ユーザは表示部分をクリックすることで実際の値を知ることができる。

3.3.3 MPP Apprentice Performance Tool

(1) 概要

本ツールは、T3D（分散メモリ並列計算機、クレイ社）用の性能評価ツール[27]である。数千個の規模のプロセッサから構成される超並列計算機における性能向上のためのチューニングを支援するために開発された。超並列計算機については、これまで最高性能（台数に比例した速度向上率）を引き出すことが極めて困難である、という認識のもと、超並列計算機の初心者が簡単に高い性能を引き出すことができるシステムを目指して開発された。Xウィンドウを用いて、操作性の向上を図っている。

(2) 機能

(i) 出力

ジョブの実行情報として提供されるデータは、各サブルーチン単位の実行時間、命令回数、仮想共有メモリアクセスに関するオーバーヘッド、通信時間等である。図3.3.2に本ツールの出力画面の例を示す。

(ii) 可視化機能

上記データは、各サブルーチン／関数に対して出力される。出力形式は、数値及び棒グラフ

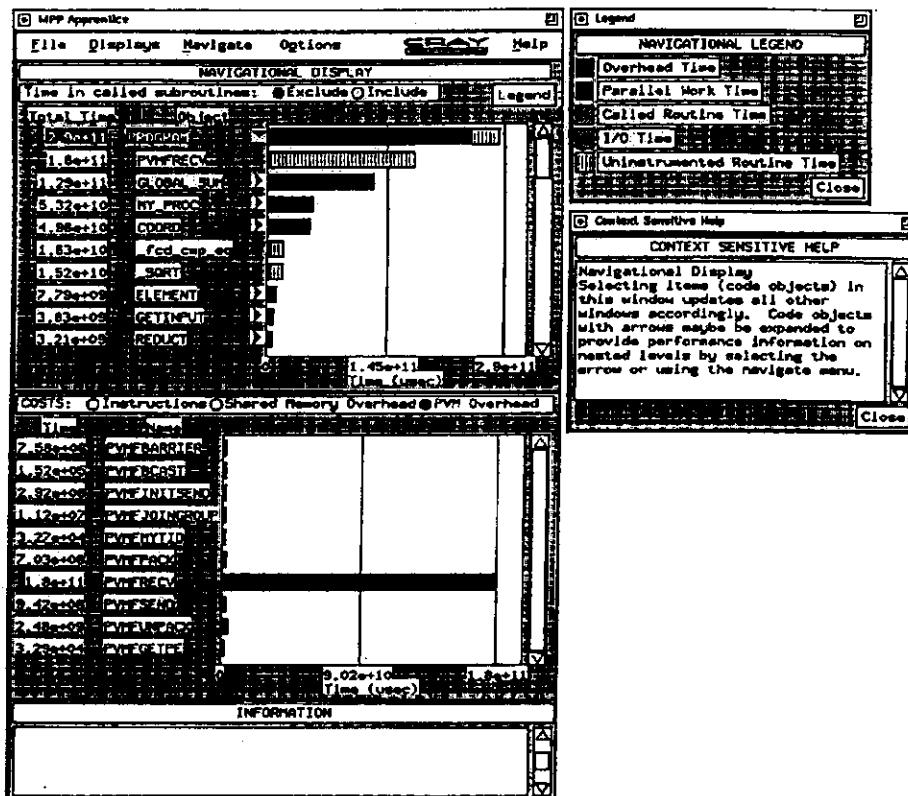


図 3.3.2 MPP Apprentice PErformance Tool 表示画面

である。

(iii) 知識ベース

本ツールには、知識ベースが組み込まれている。知識ベースは、上記データから副次的に算出される性能（例えば、FLOPS 値）を出力するとともに、性能を改善する方法をユーザに提案する。

3.3.4 ANNAI

(1) 概要

スイスの科学計算センターとNEC社の共同研究において開発された分散メモリ型並列計算機のためのプログラミング環境[28]である。稼働環境として、Cenju-2, Meiko CS-1, SUN WS（単体／クラスタ）を想定している。Xウインドウ・システムを用いている。

(2) 機能

本プログラミング環境は、i) ソース・コードに対して並列処理のための最適化を行う並列化支援ツール、ii) 静的な性能評価ツール、iii) 動的なデバッグツールによって構成される。本節では、PMA(Performance Monitor and Analyzer)と呼ばれる性能評価ツールのみを概説する。上記i), iii) 及び本ツールの詳細については、3.4節を参照されたい。

(i) ソース表示

ユーザは、ソース表示のためのブラウザを常時開いておくことができる。表示レベルは、サ

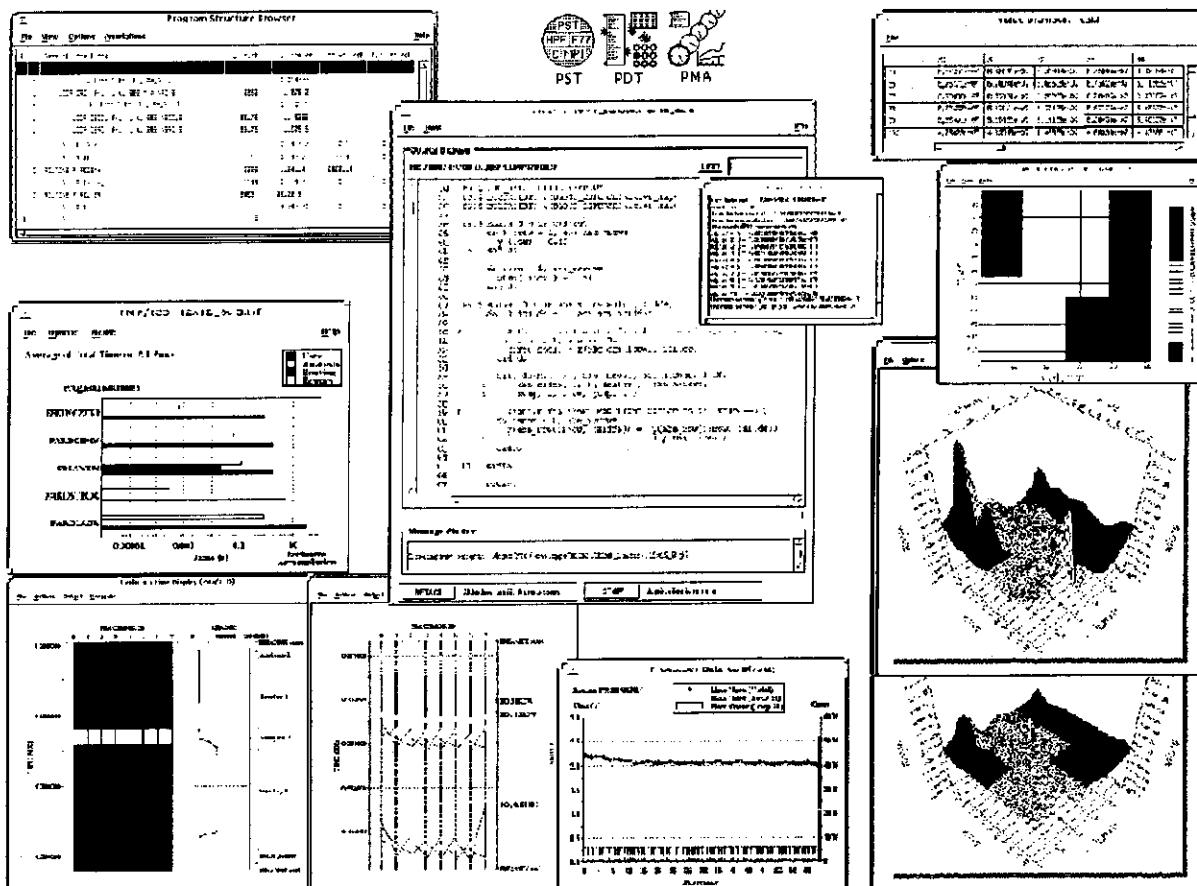


図 3.3.3 ANNAI 表示画面

ブルーチン名—ループ番号—実際のステートメント群、という3つに分かれている。各サブルーチンについて、表示レベルを個々に指定することが可能である。また、各レベルに対応する実行回数や実行時間も表示されている。

(ii) 各プロセッサの状態表示

通信処理、計算処理、同期待ち処理といった処理が色で区別された各プロセッサの状態表示が可能である。状態は、各時間ステップのタイミング・チャートとして表示され、表示内容は拡大／縮小可能である。

(iii) 実行時間表示

各プロセッサ／全プロセッサの各サブルーチンに対する実行時間が棒グラフ表示される。グラフ中の「棒」をクリックすることで、数値が表示される。

図3.3.3にANNAIの表示画面を示す。

3.3.5 PATOP (Performance Analysis TOOl for Parallel systems)

(1) 概要

PATOP[29]は、ミュンヘン工科大学において開発されたTOPSYSと呼ばれるプログラミング環境を構成する性能評価ツールである。TOPSYSは、Intel社のiPSC/2及びiPSC/860並列計算機、SUNワークステーションで構成されたクラスタ上で動作する。その特徴を以下に示す。

(i) 4つの情報出力レベル

計算時間や通信時間等のプログラムの挙動を示す情報を出力する際、ユーザが次の4つのレベルを指定する。

・システム

最も粗いレベルであり、このレベルでは、システムに対する情報を出力する（例えば、そのジョブの実行時間を出力する）。

・ノード

各プロセッサに対する情報を出力する

・オブジェクト

各サブルーチンに対する情報を出力する。

・プログラム・パート

各サブルーチンの一部に対する情報を出力する。

(ii) on-line 解析

ジョブの実行と並行してプログラム挙動解析が可能である。

(2) 機能（表示内容）

数種類の画面を用いて、アイドル／ビジー時間の表示を行う機能を持つ。アイドル／ビジー状態に主な注意を払っている理由は、i) アイドル／ビジー状態は、並列処理における大まかな傾向を把握するのにわかりやすい情報である、ii) これまで述べてきたような他のツールによって出力される詳細な（多くの種類の）情報は、次に述べる理由からon-line 解析には不適当であるためである。実行中のジョブを（任意の時間ステップにおいて）解析する場合、すべての情報を時系列データとして扱う必要がある。これは、膨大な量の解析情報をシステムが保持する必要があるため、不可能である。従って、現時点の状態把握のみが可能であるが、この場合でも、実行中のジョブはユーザの（現時点における）解析が終了するまで、実行待ちの状態となり、システムに多大の負荷がかかる。

各ノード毎の、あるいは、システム全体のアイドル時間は次のような方法によって表示される。

・指定されたある時間帯について折れ線グラフにより表示する。

・これまでの累積結果を棒グラフにより表示する。

3.3.6 NEC SX シリーズ上の性能評価ツール

(1) 概要

NEC SX-3及びSX-4（以下、本稿においてはSXシリーズと呼ぶ）上の性能評価ツールは、i) 計算時間の分布やサブルーチンの実行回数等を出力するANALYZER-P/SX [30]、及び、ii) ANALYZER-P/SXの出力結果を利用して、プログラマーにベクトル／並列化情報を教示するPARALLELIZER/SX[31]である。

(2) 機能

(i) ANALYZER-P/SX

ANALYZER-P/SX は、FORTRAN プログラムのベクトル化及び並列化に関するチューニングのための静的／動的特性を解析する性能向上支援ツールである。静的特性とは、サブルーチンの呼び出し関係、変数の参照定義関係等のプログラムの構造に関する情報（プログラムを実行しなくとも解析可能な情報）であり、動的特性とは、ある入力データを用いてプログラムを実行した際のサブルーチンやサブルーチンを構成するステートメントの実行回数やコストである。

本ツールによって出力される情報は、各サブルーチン／各ループに対する計算時間、ベクトル長、ベクトル化率、キャッシュ・ミス・ヒット時間である。

(ii) PARALLELIZER/SX

上記 (i)で述べたANALYZER-P/SX の出力及びソース・ファイルを入力とし、プログラムの挙動解析を支援するマンマシン・インターフェースである。X ウィンドウシステムを用いて、プログラムのベクトル／並列化を促進してプログラムを高速化するチューニング作業を支援する。操作方法、表示内容に関して次のような特徴がある。

(イ) 編集エディタ

ソースの編集エディタは、vi, emacs 等ユーザ（プログラマ）が選択できる（ウィンドウの環境変数として定義する）。

(ロ) ソース表示

ソース表示には、3つのレベルがある。サブルーチン名、ループ番号、実際のソース・コードである。ユーザ（プログラマ）は、必要に応じてより詳細な情報を見る。この場合、メニューバーの該当部分をクリックすることで、同じウィンドウ内にレベルに応じた情報が次々に展開されていく（expand機能）。

(ハ) 変数／配列のトレース機能

ユーザ（プログラマ）は、Trace 機能を用いて、特定の変数の定義・参照箇所を知ることができる。表示については、ソース表示同様、expand機能が使用可能である。

(二) プログラムの動的解析結果の表示

プログラムの動的解析結果については、可視化表示機能が用意されている。即ち、グラフ（円及び棒）表示とレーダー表示である。

(ホ) 複数入力ファイルの比較

プログラムの挙動は入力データに依存して変化する。ユーザーがある特定の入力データを対象に性能向上のためのチューニングを進めることは稀であり、本機能を用いることで、複数の入力データに対する挙動解析を効率良く行うことが可能である。

3.3.7 Intel Paragon 上の性能評価ツール

(1) prof/gprof

prof及びgprof は、Paragon のみならずUNIX環境では標準装備となっている実行時情報表示ツール[32]である。ただし、Paragon 上のprof/gprof は、並列処理のための拡張がなされている。

profが表示する情報は次のとおり。

- ・各サブルーチン実行時間の全実行時間に占める割合
- ・各サブルーチンの実行時間
- ・各サブルーチンの実行回数
- ・各サブルーチンを1回実行する平均時間

gprofは、上記情報に加え、サブルーチン間の関係を考慮した出力を行う。即ち、あるサブルーチンの実行時間については、そのサブルーチンから呼び出されたサブルーチンの実行時間とそのサブルーチン自身の実行時間を区別して出力する。

(2) ParaGraph

ParaGraphは、並列実行プロセスの計算負荷や通信量をモニタリングした結果をXウィンドウ上にグラフ表示する性能評価ツールである。表示するグラフの種類は次のとおり。

- ・ガントチャート
- ・棒グラフ
- ・折れ線グラフ
- ・円グラフ

図3.3.4に本ツールによる出力結果の例を示す。

3.4 ツール統合型プログラミング環境

科学技術計算プログラムの開発過程ではエディタ、コンパイラ、デバッガはもちろんのこと、コード解析ツールや性能評価ツールが利用される。マルチウィンドウシステムのもとでは、ユーザはこれらのツールをウィンドウの切り替えだけで利用することができ効率の良い開



図3.3.4 ParaGraphによるCPU利用状況出力結果

発が可能となっている。しかし、より密接に各ツールの機能を連携したツール統合型プログラミング環境を構築することで、科学技術計算プログラムの開発をより効率よく行うことを試みる研究や製品が多く開発されている。その主なものを表 3.4-1 に示す。

本節では、CAPTools 及び Annai を事例として以下で詳しく説明する。

3.4.1 CAPTools (Computer Aided Parallelisation Tools)

(1) 概要

CAPTools (Computer Aided Parallelisation Tools) は逐次プログラムのコード解析から並列プログラムの自動生成までに用いられるツールを統合化した環境である。CAPTools では、プログラム解析から並列コード生成までの各フェーズでツールが明確に解析できなかった事項に対して、ユーザがインタラクティブに情報を付加することができる。これにより、従来の自動並列化ツールより短時間で高品質の並列プログラムを得ることができる。

概念的な CAPTools の構成を図 3.4.1 に示す。キーとなる部分はユーザの並列化方針をシステムのデータベースに蓄積する部分である。並列化方針とはデータやループの分割の方法等である。従来はこれらの情報をソースコードに指示行として挿入する方式がとられていた。CAPTools ではシステムが解析を十分できない部分に対して発行している質問に対話的に回答することにより、並列化方針を指示することができる。このようにして、システムが解析した

表 3.4.1 主なツール統合型プログラミング環境

名称	研究・開発機関	概要
PED [33]	ParaScopeプロジェクト	PEDはPTOOLシステム[34]の一部。コンバイラとコード解析ツールの統合。データ依存情報をグラフ表示する。
D Editor[35]	D Systemプロジェクト	データパラレルプログラム向けに開発された言語FORTRAN D[36]あるいはHPFの開発環境。コンバイラ、コード解析ツールとエディタの統合。PEDを元にしている。プログラムのオーバービュー、依存情報、通信情報、データ配置情報、ソースコードを表示する5つの部分(Paneと呼ばれる)より構成される。
VCFS [37]	SUPERBプロジェクト	HPFに似たVienna FORTRAN[38]という言語からMPF (Message Passing FORTRAN)に変換するシステム。コンバイラ、コード解析ツールとエディタの統合。ソースコードの表示された画面上で特定のループとそこでの配列の分割方法などを指定するだけで、メッセージ通信の記述が自動的に生成される。
CAPTools [39]~[42]	Greenwich大学	コンバイラ、コード解析ツールとエディタの統合。コンバイラだけではなくユーザからの情報も加味して、データ依存情報を表示する。詳細は、以下3.4.2節参照。
FORGE90 [43]	APR	コンバイラ、コード解析ツールとエディタの統合。データ依存情報、コールグラフなどを表示する。詳細は2.5.3.4節参照。
xbrowse [44]	BNN	コンバイラ、コード解析ツールとエディタの統合。ループ構造解析、コールグラフなどを表示する。詳細は2.6.3.4節参照。
Annai [45]	CSCS-ETH/NEC共同研究	コンバイラ、コード解析ツール、性能解析ツール、デバッガの統合。並列化支援ツールPST、パフォーマンスマニタ&アナライザPMA、並列デバッグツールPDTの3つ部分から構成される。詳細は、以下3.4.2節参照。

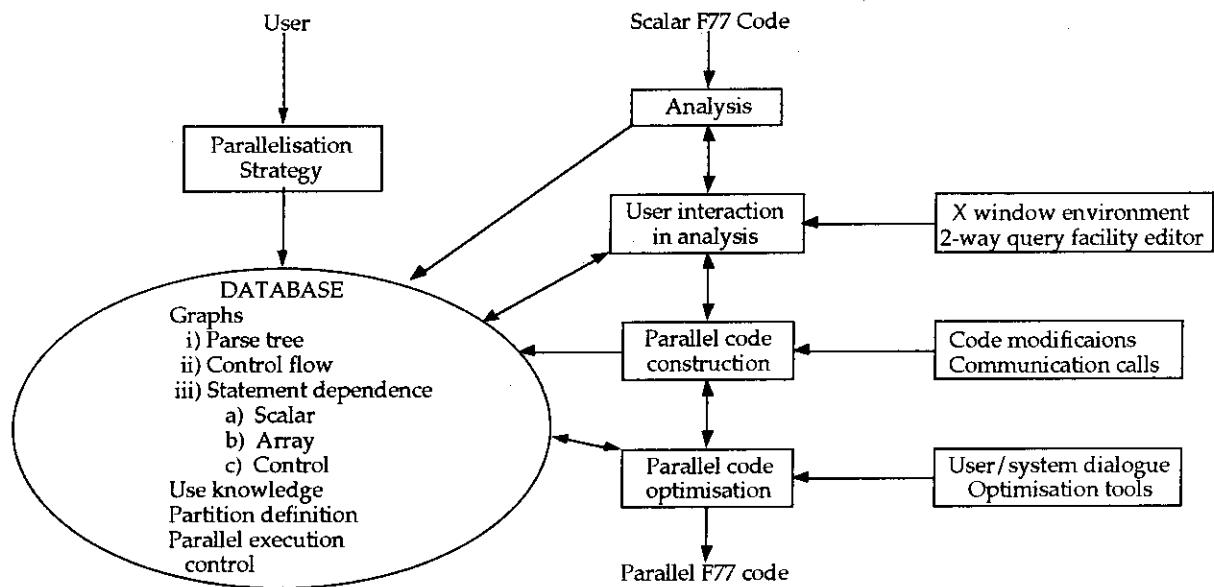


図 3.4.1 CAPTools の構造

情報とユーザから与えられた情報を統合したものをknowledge baseと呼んでいる。

(2) プラットフォーム

並列プログラムのプラットフォームとしては共有メモリ型、分散メモリ型のいずれにも対応している。通信関数は中立的なもの(個々の計算機に実装されている種々の通信関数に依存しない独立な形式であるということ)を生成し、これを特定のマシンの通信関数やPVM、MPIといった広く利用されている通信関数に変換する。現在CAPToolsが中立的なものとして生成するものは、同期型の送信(SEND)、及び受信(RECEIVE)、交換(EXCHANGE)、総和などのグループ演算(COMMUTATIVE)である。生成された並列プログラムの性能評価を実施した機種には、SUN IPXワークステーションクラスタ、HP735ワークステーションクラスタ等が挙げられている。

(3) ツール環境

CAPToolsはユーザとの対話的な進行によって、大規模な科学計算プログラムの高品質な自動並列化を目指している。そのために、以下のような機能に重点が置かれている。

- ・依存解析で得られた知識ベースを探索する機能
- ・対象の逐次プログラムの並列性を調査する機能
- ・依存解析、あるいは依存解析後に行われるデータ分割、並列プログラム生成の質を改善するために、知識ベースにユーザの情報を付加する機能。
- ・解析に必要な情報としてシステムが要求するものを調べ、それに回答できる機能
- ・可能なデータ分割オプションを探査し、明確にする機能
- ・個々の並列実行部分からデータ分割を行うのではなく、プログラム全体の並列実行部分を解析することにより、最適なデータ分割を算出する機能
- ・効果のある並列実行のコントロールマスク算出や通信処理を探査し、明確にする機能

- ・並列プログラム生成のオプションとして、ループ等の変換を行うか選択できる機能
 - ・プロファイル情報を用いて、ユーザが並列化のキーとなる部分を認知できる機能
- これらを実現する、CAPToolsのユーザインターフェースの構成を図3.4.2に示す。並列プログラム生成の過程でこれらのユーザインターフェースがどの様に関連するか主なものを挙げて紹介する。

(i) UserKnowledge ウィンドウ

CAPToolsでは変数の知識を次の情報の集合で表している。

(イ) Symbol : 変数の名称

(ロ) Defining Statement : 値を代入しているステートメント（多くはREAD文）

(ハ) Call Path : 変数を引き渡している呼び出しパス

UserKnowledge ウィンドウではこれらの情報をユーザに表示する。

(ii) KnowledgeBase ウィンドウ

依存解析ツールは各変数のとりうる範囲を明確にできない場合、ユーザに質問を要求する。KnowledgeBase ウィンドウはその質問をその変数を含む等式、不等式及びメッセージで表示し、ユーザがTrueまたはFalseのボタンで回答する機能を提供する。

(iii) DefGraph Viewer ウィンドウ

CAPToolsでは依存解析の結果をグラフィカルに表示する。ステートメントはその行番号を楕円で囲んだノードとして描かれ、行番号の昇順に縦に並べられる。依存関係はノード間を結ぶ矢印で表される。ユーザの興味対象にのみ焦点を当てるために、選択したステートメントの範囲や特定の変数に限定して解析図を表示する機能をもつ。また、Why Dependenceボタンにより依存関係の発生している理由について詳細に調査することができる。

(iv) Partitioner ウィンドウ

データの分割方法の操作はPartitioner ウィンドウを通して行う。Partitioner ウィンドウではサブルーチン毎に分割される配列とされない配列の一覧が表示され、ユーザによる操作が可

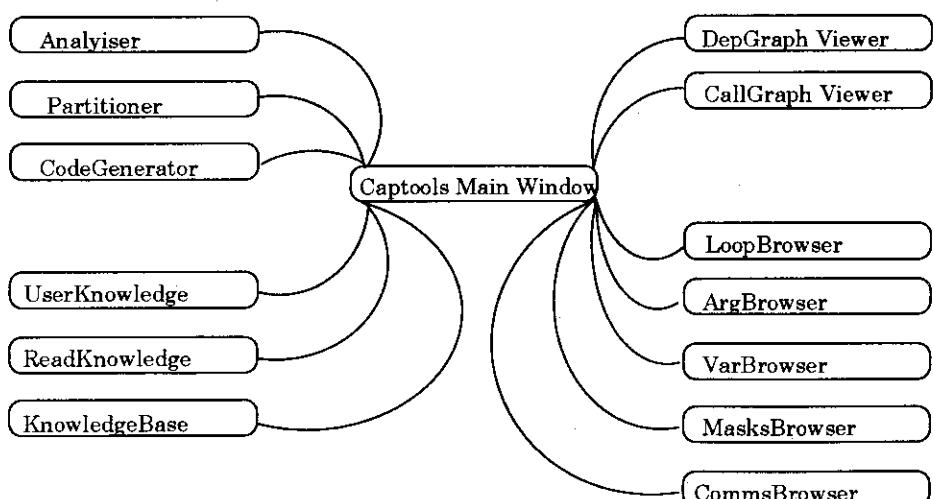


図 3.4.2 CAPToolsのユーザインターフェースの構成

能である。CAPToolsでは実行時に分割数を決定できるように、各プロセッサが担当する範囲の下限と上限を "CAP_L" 及び "CAP_H" で始まる変数で管理する。分割される配列は、そのインデックスと定義されている範囲と共に、分割範囲の管理に使用される変数名が表示される。

(v) MasksBrowser ウィンドウ

CAPToolsは分割された配列を使用するステートメントに対して、インデックスが分割範囲にあるときのみ実施する記述（通常はそのインデックスが分割範囲にあることを判定するIF文）を生成する。これをマスクと呼んでいる。例えばAという配列がインデックス3で分割され、その範囲がCAP_LAPとCAP_HAPで管理されている場合、

$$C(I) = A(IJ,K)$$

というステートメントにマスクが追加されると

$$IF (I .GH. CAP_LAP .AND. I .LH. CAP_HAP) C(I) = A(IJ,K)$$

となる。MasksBrowser ウィンドウではサブルーチン毎にこれらのマスクが表示され、ユーザによるマスクの操作が可能である。

(vi) CommsBrowser ウィンドウ

CAPToolsは分割された配列のデータ交換のための通信ステートメントを生成する。CommsBrowser ウィンドウではサブルーチン毎にこれらの通信が表示され、ユーザによる変更操作が可能である。また、Why Communicationボタンによりその通信が必要である理由について詳細に調査することができる。

3.4.2 ANNAI

(1) 概要

パラレルプロセッシングに関するCSCS-ETH(Centro Svizzero di Calcolo Scientifico・Eidgenossische Technische Hochschule Zurich)/NEC共同研究において、HPF、並列パフォーマンスマニターアナライザ、分散メモリ型並列計算機(DMPP)対応デバッガの統合ツール環境が開発されている。この環境はNEC Cenju-2をはじめ、標準MPIの動作する幾つかのプラットフォーム上に実装されている。開発段階においては、プロトタイプを順次開発し、継続的に評価し、次期プロトタイプにフィードバックするという開発方針に重点がおかれている。

また、本ツール開発における留意点は以下のとおり。

- ・統合ツール環境はデータ並列MIMD用高レベル言語でも、低レベル・メッセージ・パッシングでも使用可能なように設計し実現すること。
- ・標準化されたプログラム言語とマシンインターフェースのサポート。高レベルではHPFと可能なかぎりそのエンハンスをサポートする。低レベルではマシンインターフェースとしてMPIを使用する。
- ・今日DMPP上で困難とされている科学計算アプリケーション用の並列化ツール、デバッガ、パフォーマンスマニタ、解析ツールの利用を可能にするためにHPFの拡張を提案し実現すること。

- ・アプリケーション指向のツール設計。ツールは順次プロトタイプ化され、アプリケーション開発チームはプロトタイプの使用結果を継続的にフィードバックする。

(2) プラットフォーム

対象プラットフォームはNEC Cenju-2, Cenju-3, Meiko CS-1, IntelParagon, Cray T3D, 及びSun WS(マルチプロセッサ), Sun WSクラスタである。

通信インターフェースとして標準となりつつあるMPIを採用しており、ツール開発に必要な低レベル・メッセージ・パッシング関数等MPIのサブセットのみ使用している。

(3) ツール環境

ツール統合環境は、ユーザインタフェースを共有したPST, PDT, PMAの3つの部分から構成される。この統合環境では高レベル拡張HPFプログラムも低レベル・メッセージ・パッシング・プログラムも扱える。いずれのパラダイムでもPSTは主にコンパイラとして機能する。さらに、PSTはソースプログラムにプリプロセッサとして機能し、PDTやPMAのための情報も生成する。

PMAとPDTではユーザが抽象レベルを切り換えて情報を入手することが可能である。最も低い抽象レベルは、DMPPのハードウェアレベルの詳細情報を提供する。

最も低い抽象レベルは全マシンに実装されていると考えられる共通の通信インターフェースであるMPIを想定している。例えば、通信オーバヘッド、計算、アイドルタイムをブレークダウンした詳細情報が、プログラム実行中の任意時にとりだせる。

高抽象レベルでは、グローバルネームスペース、データ分割、あるいはユーザには单一プログラムスレッドとしてみえるデータ並列実行モードといった高レベル言語にあった形で情報が供給される。

以下、ANNAIを構成する個々のサブシステムに関して詳述する。

(i) パラレル化サポートツール(PST)

PST (Parallelization Support Tool) は超並列分散システム上の科学計算アプリケーションの並列化をサポートする。ソースプログラムに対してプリプロセッサとして機能し、PDTやPMAのための情報も生成する。

データやループはディレクティブにより、スタティックブロックやサイクリック分割や直接ユーザ定義の分割を使って、データやコントロールフローの分割が可能である。

(ii) 並列デバッグツール(PDT)

PDT (Parallel Debugging Tool) はソースレベルの対話的デバッガである。PDT及びPMAからの情報を利用する。場合に応じて、従来のソースレベルデバッガの機能とコマンドに、DMPP上での並列プログラムのデバッグのための機能が追加されている。追加されたコマンドは以下のとおりである。

- ・対象のプラットフォームへの接続と切断をするコマンド
- ・並列プログラムをロードし実行するコマンド
- ・グローバルなブレークポイントや例外処理を行うコマンド

- ・スタック、レジスタあるいはデータの検査に該当しプログラムが停止したプロセッサに切り替えるコマンド

将来のプロトタイプではシングルスレッドプログラムとして表せるPSTプログラムのソースレベルデバッグを完全にサポートすることが計画されており、さらにデータレイアウトとプログラム実行の分割具合のグラフィカルビュー（PrismやForgeでサポートされているようなビュー）を提供することによって大きな分散配列の解釈がしやすくなる。さらに、メッセージ・キャッシング・レベルでは、実行中デッドロックの防止、条件付きブレークポイント、競合条件の防止のための機能を付加する計画である。

(iii)パフォーマンスマニタ・アナライザ(PMA)

PMA（Performance Monitor and Analyzer）は、並列プログラム実行中に生成されるパフォーマンス情報を解釈し、この情報を可視化したり解析したりすることによってパフォーマンスチューニングとプログラム実行の解釈の支援をする。異なる抽象化のレベルがサポートされており、そのレベルは、通信や個々のプロセスのメモリ利用状況を解析するレベルからデータ・パラレル実行のグローバルビューのレベルまである。

PMAが outputするプログラムステートメント（及びブロック）間の関係やパフォーマンス測定値は、プログラム実行の振る舞いを理解するうえで重要である。選択された測定値と関連するコードセクションは同時表示される。即ち、ソースコードの余白に詳細な測定グラフが表示される。

インストールメントの”挿入点”，例えば通信トレースをonあるいはoffする位置はソースブラウザ上で指定する。また、適切なインストールメントのレベル（簡単な実行経過から詳細な履歴まで）は、TSA（後述）と連係しながらPMAによって設定される。この場合、TSAは、プログラムインストールメントの状態の修正や並列プログラム実行中のインストールメントの調整を行う。

(iv)ユーザインターフェース(UI)

UI（User Interface）はPDT、PMA等の構成ツールに対して一貫したユーザインターフェースを提供する。ソースコードブラウザ、PMA及びPDTからの要求に応じて、興味対象部分ソースコードの表示、実行プログラムからの標準出力、標準エラーの表示、TSAとの連係による他のツールとの制御を行う。このUIによってPDTとPMAは、統一化されたインターフェースのもとに呼び出し管理される。

UIの将来の拡張項目はPMAとUI間のフィードバックである。それは例えば、最も重要なグローバルパフォーマンスの統計を関連するソース行の横に直接表示したり、相互作用するプログラムの注釈をプログラムの呼出しグラフの中に階層表示することである。

(v)ツールサービスエージェント(TSA)

TSA（Tool Services Agent）は並列プラットフォームとPMA及びPDTとのインターフェースである。SPMDプログラムの開発に重要なグローバルなブレークポイント指定を支援する。

PDTとの相互作用はもちろんのこと、PMAも必要に応じて相互作用する。PMAはパラレル

プログラムの中に適切なインストールメント（レベルと領域）群を持ち、そして、TSAはこれを実行する。PMAによって生成される統計情報は、システムの実行時ライブラリの一部であるグローバルデータ構造の中に格納される。グローバルデータ構造はTSAを介して読まれる。

参考文献

- [1] K. Hwang : *Advanced Computer Architecture : Parallelism, Scalability, Programmability*, McGraw-Hill, Inc. (1993) .
- [2] R. Perrott : *Parallel Languages*, in *Parallel & Distributed Computing Handbook*, edited by A. Zomaya, McGraw-Hill, Inc. (1996) .
- [3] G. Blelloch : *NESL : A Nested Data-Parallel Language*, CMU-CS-95-170 (1995) .
- [4] K. Ueda, T. Chikayama : *Design of the Kernel Language for the Parallel Inference Machine*, *The Computer Journal*, Vol.33, No.6, pp.494-500, (1990) .
- [5] N. Carriero, D. Gelernter : *How to Write Parallel Programs : A Guide to the Perplexed*, *ACM Computing Surveys*, Vol. 21, No.3, pp.323-357 (1989) .
- [6] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam : *PVM3 USER'S GUIDE AND REFERENCE MANUAL* (1995) .
- [7] Message Passing Interface Forum : *MPI : A Message-Passing Interface Standard* (1995)
- [8] I. Foster, K. Chandy : *FORTRAN M : A Language for Modular Parallel Programming*, *J. of Parallel and Dist. Compt.* (1994) .
- [9] I. Foster, B. Avalani, A. Choudhary, M. Xu : *A Compilation System that Integrates High Performance Fortran and Fortran M*, Proc. 1994 Scalable high Performance Computing Conf., IEEE Comp. Sci. Press (1994) .
- [10] ISO : *Information Technology -Programming Languages- Fortran90* (1991) [ISO/IEC 1539 : 1991 and now ANSI X3.198-1992] .
- [11] High Performance Fortran Forum : *High Performance Fortran Language Specification Ver.1.0* (1993)
- [12] The HPC++ Working Group : *HPC++ White Papers*, Technical Report TR 95633, Center for Research on Parallel Computation (1995) .
- [13] P. Carlin, K. Chandy, C. Kesselman : *The Compositional CC++ Language Definition*, Technical Report Caltech-CS-TR-92-02, California Institute of Technology (1992) .
- [14] H. Zima, B. Chapman, 村岡洋一訳 : *スーパーコンパイラー, Supercompilers for Parallel and Vector Computers*, オーム社 (1995) .
- [15] D. Kuck : *A Survey of Parallel Machine Organization and Programming*, *ACM Computing Surveys*, Vol.9, pp.29-59 (1977) .
- [16] C. Polychronopoulos, M. Girkar, M. Haghight, C. Lee, B. Leung, D. Schouten : *Parafrase-2 : An Environment for Parallelizing, Partitioning, Synchronizing, and Scheduling Programs on Multiprocessors*, Proc. 1989 Int. conf. Parallel Processing, IEEE Computer Society , pp.II-39 - II-48 (1989) .
- [17] J. Allen, K. Kennedy : *Automatic Translation of FORTRAN Programs to Vector Forms*, *ACM Trans. on Programming Languages and Systems*, Vol. 9, pp.491-542 (1987) .
- [18] H. Zima, H. Bast, M. Gerndt : *SUPERB : A Tool for Semi-automatic MIMD/SIMD Parallelization*, *Parallel Computing*, Vol.6, pp.1-18 (1988) .
- [19] C. McDowell, D. Helmbold : *Debugging Concurrent Programs*, *ACM Computing Surveys*, Vol.21, No.4, pp.593-622 (1989) .
- [20] A. Beguelin, J. Dongarra, A. Geist, V. Sunderam : *Visualization and Debugging in a Heterogeneous Environment*, *Computer*, Vol.26, No.6 , pp.88-95 (1993) .

- [21] *Paragon Application Tools User's Guide*, 312545-113, Intel (1994) .
- [22] S. Grabner, D. Kranzlmuller, J. Volkert : *Debugging Parallel Programs using ATEMPT*, LNCS 919 : *High Performance Computing and Networking*, pp.235-240 (1995) .
- [23] S. Sistare, D. Allen, R. Bowker, K. Jourdenais, J. Simons, R. Title : *Data Visualization and Performance Analysis in the Prism Programming Environment*, IFIP Transactions : A, *Programming Environment for Parallel Computing*, pp.37-52 (1992) .
- [24] 小柳滋, 久保田和人, 川倉康嗣: 並列プログラムデバッグのための可視化ツール, 情報処理学会論文誌, Vol.37, No.7, pp.1299-1307 (1996) .
- [25] J. Ousterhout (西中 芳幸, 石曾根 信訳) : *Tcl & Tk ツールキット*, ADDISON-WESLEY プロフェッショナルコンピューティングシリーズ
- [26] A. Hordrouidakis, R. Procter : *The Design of a Tool for Parallel Program Performance Analysis and Tuning*, in *Programming Environments for Massively Parallel Distributed Systems*, Monte Verita, Switzerland (1994) .
- [27] W. Williams, et al. : *The MPP Apprentice Performance Tool : Delivering the Performance of the Cray T3D*, in *Programming Environments for Massively Parallel Distributed Systems*, Monte Verita, Switzerland (1994) .
- [28] C. Clemensson, et al. : *An Environment for Portable Distributed Memory Parallel Programming*, in *Programming Environments for Massively Parallel Distributed Systems*, Monte Verita, Switzerland (1994) .
- [29] T. Bemmerl, et al. : *Adapting the Portable Performance Measurement Tool PATOP to the Multi-Transputer Monitoring System DELTA-T*, in *Programming Environments for Parallel Computing*, IFIP Transactions A-11, North-Holland, 1992
- [30] SUPER-UX ANALYZER-P/SX 利用の手引き, G1AF15-2, 日本電気株式会社 (1995) .
- [31] SUPER-UX PARALLELIZER/SX 利用の手引き, G1AF17-2, 日本電気株式会社 (1995) .
- [32] Paragon Application Tools User's Guide, 312545-113, Intel (1994) .
- [33] K. Kennedy, K.S. McKinley, C.-W. Tseng : *Interactive parallel programming using the ParaScope Editor*, IEEE Trans. on Parallel and Distributed Systems Vol.2, No.3, pp.329-341 (1991) .
- [34] J.R. Allan, D. Baumgartner, K. Kennedy, A. Porterfield : *PTOOL : A semi-automatic parallel programming assistant*, in Proc. 1986 Int. Conf. on Parallel Processing, IEEE Computer Society, pp.164-170 (1986)
- [35] S. Hiranandai, K. Kennedy, C.-W. Tseng, S. Warren : *The D editor : A new interactive parallel programming tool*, in Proc. Supercomputing '94 (1994) .
- [36] S. Hiranandai, K. Kennedy, C.-W. Tseng : *Compiling Fortran D for MIMD distributed-memory machines*, Comm. ACM Vol.35, No.8, pp.66-80 (1992) .
- [37] B.M. Chapman et al. : *VIENNA FORTRAN compilation system user guide Ver.1.0*, Dept. of Statistics and Computer Science, University of Vienna, Austria (1994) .
- [38] B. Chapman, P. Mehrota, H. Zima : *Programming in Vienna FORTRAN*, Scientific Programming, Vol.1, No.1, pp.31-50 (1992) .
- [39] M. Cross, S.P. Johnson, P. Leggett, C.S. Ierotheou : *Computer aided parallelisation tools (CAPTools) - Concept overview and performance on the parallelisation of structured mesh codes*, Parallel Computing Vol.22, No.2, pp.163-195 (1996) .
- [40] S. Johnson, M. Cross, M. Everett : *Exploitation of Symbolic Information in Interprocedural Dependence Analysis*, Parallel Computing, Vol.22, No.2, pp.197-226 (1996) .
- [41] S. Johnson, C. Ierotheou, M. Cross : *Automatic Parallel Code Generation for Message Passing on Distributed Memory Systems*, Parallel Computing, Vol.22, No.2, pp.227-258 (1996) .

- [42] P. Leggett, A. Marsh, S. Jonson , M. Cross : *Integrating User Knowledge with Information from Parallelization Tools to Facilitate the Automatic Generation of Efficient Parallel FORTRAN Code*, *Parallel Computing*, Vol.22, No.2, pp.259-288 (1996) .
- [43] FORGE90 distributed memory paralleliser : *User's Guide Ver.8.0*, Placerville, (1992) .
- [44] Program Browser (xbrowse) , IN-2140, Cray Research, Inc. (1995) .
- [45] C. Clemenccon, A. Endo, J. Fritscher, A. Muler, R. Ruhl and B. J. N. Wylie : *An environment for portable distributed memory parallel programming*, in *Programming Environment for Massively Parallel Distributed Systems*, Monte Verita, Switzerland, pp.159-170 (1994) .

4. 並列プログラム開発を支援する STA 環境

現在我々は科学技術計算における並列プログラム開発環境STA(Seamless Thinking Aid)を構築している。STAの目的は、統合化されたプログラム開発支援ツール群を提供することにより、並列プログラムの開発における開発者の途切れのない思考を支援することである。

本章では、まずプログラム開発における途切れのない思考の支援(Seamless Thinking Aid)という概念を説明し、それを実現するためのポイントを述べる。そのポイントに基づいて、現在構築を行っているSTA環境の概要を説明する。また、現在検討を行っている以下の2つの特徴、すなわち

- (1) デバッグデータ可視化機能
 - (2) エディタを中心とした統合環境の実現
- について述べる。

4.1 途切れのない思考の支援

公文[1]は、人間の行為は一般に認識、判断、決定、実施手順化、実施の5つの部分から構成されるという“行為のモデル”を提案している。並列プログラム開発における一連の作業もこの過程の繰り返しであるといえる。例えば、プログラムのデバッグ作業を考えれば、実行プログラムのある時点における変数の状況をチェックし(認識)，正誤の判断を下し(判断)，次にプログラム変数状況をチェックする時点を決定し(決定)，その実現に必要なコマンド群を整理し(実施手順化)，コマンドを発行する(実施)。そして再びこのサイクルを繰り返す。

上記の例は、並列プログラム開発の一フェーズであるデバッグ作業に対して“行為のモデル”を適用したものであるが、上記のデバッグ作業の他、コンパイル作業、チューニング作業等並列プログラム開発を構成する個々の作業フェーズも同様に表現することができる。従って、並列プログラム開発を支援するためには、各作業フェーズにおける“認識，・・・，実施”という手順の繰り返しを支援することが重要である。

しかし、各作業フェーズにおける手順の繰り返しの支援だけでは、並列プログラムの開発支援には不十分である。並列プログラム開発作業では、“認識，・・・，実施”という手順が各作業フェーズ内に限らず複数の作業フェーズにわたって繰り返されることによって、最終的に作業全体が実施される。例えば、デバッグ作業では上記の手順を繰り返すことによってプログラムエラーを探索し(フェーズ内の繰り返し)，エラーが発見されるとプログラムの変更を行い、再コンパイルすることにより再度デバッグ作業を行う(フェーズ間の繰り返し)。従って、並列プログラム開発を支援するためには、個々の作業フェーズに閉じることなく全体にわたって一連の作業の流れが円滑かつ効率的に行えるよう支援することが必要である。

この考察に基づき、我々は、“並列プログラム開発においてユーザが任意の時点で随時計算機から情報を得、判断し、それに基づいて次の行動を取るといった一連の流れを円滑かつ効率的に行えるような環境を提供すること”が並列プログラム開発の支援に重要であると考えている。そして、このような環境による並列プログラム開発の支援をすることを、途切れのない思考の支援と呼んでいる。

途切れのない思考の支援を実現するために、現在我々は以下に示す3種類のギャップの解消が重要であると考えている。

(1)処理要求から処理終了までの待ちが少ないとこと ー作業の流れのギャップの解消ー

計算機の応答時間と人間の思考時間は密接に関連しており、応答時間が長くなれば人間の思考時間も長くなることが実験により示されている。[2]

従って、この条件は上述の“行為のモデル”において、(計算機に対する操作の)実施から(その操作に対する計算機の反応の)認識までの流れを円滑にすることに対応する。

この条件を満足させるためには、例えば支援ツールの処理自体も並列化することにより高速処理を図ること(迅速なターンアラウンド)や、事後処理的(postmortem)な情報の提供ではなくリアルタイムな情報提供機能(トラッキング)を実現したり、さらにその情報により次の行為が決定されたらその時点でユーザが動的に介入できる機能(ステアリング)を実現することなどが考えられる。

(2)計算機の提供する情報とユーザの要求している情報の乖離が小さいこと ー計算機と人間の情報のギャップの解消ー

計算機の提供する情報がユーザの要求している情報と意味的、質的に異なっている場合、ユーザは計算機の提供した情報を再加工して初めてその情報を認識し、判断することが可能となる。従って、条件(2)は、人間の行為モデルにおいて、計算機から提供された情報の認識から判断、決定までに至る流れを円滑にすることに対応する。

2.1で述べたように、物理現象をモデル化し、並列計算機上で計算するためには物理モデル、数学モデル、計算アルゴリズムなど種々の表現がとられ、それらの表現間の対応付けがとられる。作業の各局面に応じて開発者は種々の表現に基づいて思考している。その際、計算機が開発者の基づいている表現に適合しない情報を提供した場合、開発者は自分自身で表現間の対応付けに基づいて提供された情報を変換し、認識、判断しなければならない。

例えば、計算結果の正誤の判断は、最終的に物理モデルに基づいて予想される結果と計算結果を比較することにより行われる。この時、計算された値をそのまま見ても、直ちに正誤を判断できることはほとんどない。プログラム設計、作成フェーズ時に行った物理モデルからプログラムへの変換に基づいて、計算空間内での値を物理空間内での物理量にマッピングすることにより初めて正誤判断可能になる。

このような情報の乖離は、並列プログラム開発における種々のフェーズで発生しうる。例えば、デバッグや実行性能評価する際、計算アルゴリズムに基づき各プロセスの動作を予想したり実行効率を評価したりする。この時ユーザは、プログラムの構造(サブルーチン構造)に基づいて予想や評価を行う。従って、ツールから提供される情報もこの構造に基づいて提供されなければならない。実行性能の評価において、並列計算機内のプロセッサの稼働状況を時間経過と共にモニタしても、その時点でプログラムのどの部分が実行されているのかがわからなければ有効な情報とはなりにくい。

情報の乖離には、もう1種類存在する。それは人間の情報処理特性と計算機の情報処理特性の違いに基づくものである。科学計算においては、非常に大量のデータを計算することが多い。計算機はこのような大量データを定量的に処理することに秀でているが、人間の場合、ある程度以上の量のデータを定量的に処理することは一般に困難であり、定性的に処理せざるを得ない。

例えば、デバッグ作業においてある配列の値を調べることを考えると、その配列の大きさが非常に巨大であった場合、その値がそのまま出力されても人間がそれから何らかの情報を得るこ

とは困難である。

従って(2)を満足させるためには、利用者がそのツールを利用する状況を検討し、その状況において利用者はどのような表現や構造に基づいて思考しているのかを分析し、その表現、構造にマッチした形態でデータを提供する必要がある。また、可視化手法などを利用して大量データの定性的認識を支援したり、データのフィルタリング機構を設けて必要なデータのみを取り扱える手段を提供したりするなどの情報提供手法の検討が必要である。

(3)開発の流れに沿って複数のツールを統合的に利用できること　—ツール間のギャップの解消—
並列プログラムの開発では作業内容に応じて種々のツールを利用する。ツールを統合化してこの流れを円滑にすることは、人間の行為モデルにおいて次に行うべき作業を決定し実施手順化した後、具体的に実施する際の流れを円滑にすることに対応する。

ここでいう統合的なツールの利用とは、以下の2点が実現されていることをいう。

- ・複数のツールの使い分けに伴って目的とする作業に本質的でない作業がユーザに発生しないように、ツール間で情報を伝達できるような機構が存在すること。

例えばプログラム開発フェーズにおいて正常実行を確認した後、性能向上作業に移る場合、現在の支援ツールでは再度性能評価のためにプログラムを実行しデータを収集しなければならない。しかし事後解析用デバッガ及び性能評価ツールでは、共に計算機内での計算実行状況をモニタリングするという作業が発生しており、通信の発生時刻、計算実行時間等共通に収集するデータも多く存在する。従って、両ツール間で採取するデータを共通化し相互利用可能にすれば不要な再実行を避けることができる。

- ・あたかも一つのツールを使い続けているような統一された操作感をユーザに提供できること。

個々のツールにより操作感(コマンド、手順)がまちまちであるとユーザはツール毎に操作を覚える必要があり、また行うべき作業とツール及びその操作手法との対応を常に意識する必要がある。これは円滑な作業の手順化を損なう。

この2点を満足させることができが(3)の実現には重要である。

4.2 STA 環境

4.2.1 STA 環境概要

現在STA環境は、Hitachi SR2201及び日本原子力研究所が開発したデスクサイド並列計算機上で構築が進められており、平成8年度末には第1版が完成する。現在のところ、SR2201及びデスクサイド並列計算機をプラットフォームとしているが、将来は任意の並列計算機上に実装する予定である。

STAは、並列プログラム開発で利用される既存の各種ツール(エディタ、コンパイラ、デバッガ、パフォーマンスマニア、可視化ツール)に新規機能を付加し、更にGUIを用いてそれらを統合することにより並列プログラム開発作業全体の支援を目指している。

STA環境で実現されている種々の機能を、その機能を利用するプログラム開発フェーズと対応させて図4.2.1に示す。

4.1で述べたように、並列プログラム開発において途切れのない思考を支援するためには、

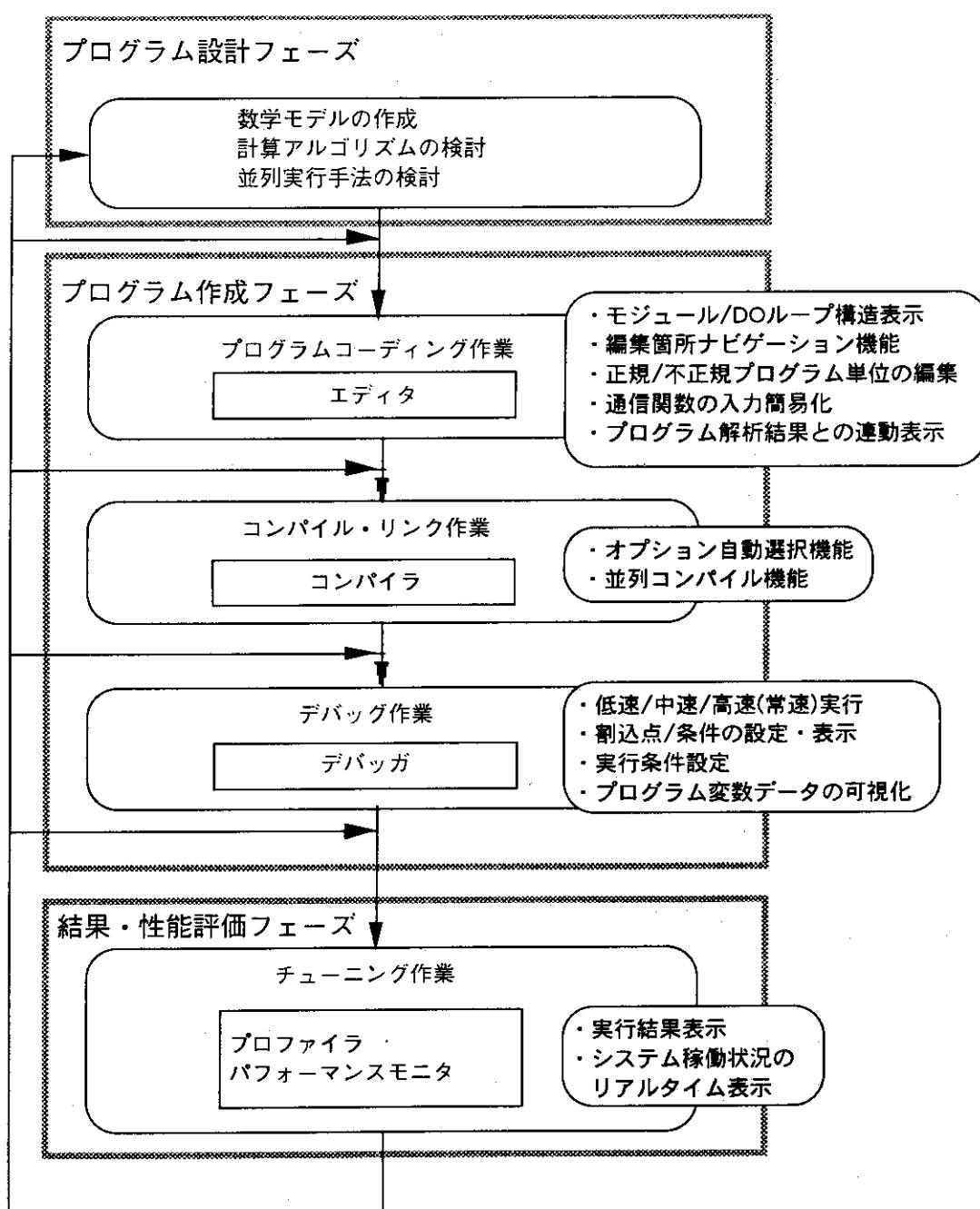


図 4.2.1 STA 環境において実現された機能

- (1) 作業の流れのギャップの解消
 - (2) 計算機とユーザの情報のギャップの解消
 - (3) ツール間のギャップの解消
- が重要である。STA 環境では、(1) を実現するために
- ・プログラムの並列コンパイル機能
 - ・デバッグ実行速度の制御機能
- を実装している。

プログラムの並列コンパイルとは、複数のファイルから構成されたプログラムを並列にコンパイルすることにより、コンパイル時の応答時間を短縮する機能である。プログラム開発初期段階では多数のエラーが存在し、何度もプログラムの修正、コンパイルを繰り返すことが多い。コンパイラからの応答が遅い場合、その都度ユーザの思考が途切れることになる。STAでは並列計算機の特長を活かしコンパイルを並列に実行することによって、応答時間を短縮し思考の途切れを低減している。

また、デバッグ実行速度の制御とは、デバッグ要求にあわせて低速、中速、高速(常速)の3種類の実行速度でデバッグを行えるよう実行速度調節を行う機能である。デバッグを行う場合、ユーザは注目する個所に到達するまではなるべく高速に実行し、注目する個所ではなるべく詳細にデータを抽出したいという要求を持っている。これは注目する個所に到達するまでに要する時間を短縮することにより思考の途切れを少なくしたいという要求である。STAではサブルーチン毎に実行速度を上記のように調節することが可能なため、既にデバッグが終了し、詳細にデバッグを行う必要のないサブルーチンには高速モードを指定することにより通常実行と同等の実行速度が実現される。また、詳細にデバッグを行う必要のあるサブルーチンには低速実行モードを指定することにより、自動的にそのサブルーチンの入り口で実行が停止され詳細なデバッグを開始することができる。中速モードを指定すると、サブルーチンの入り口で実行が停止することはないが、そのサブルーチンの実行を開始したという情報は提供される。ユーザはプログラムの実行をサブルーチン単位でモニタリングすることが可能である。この3種類の実行速度の指定を適宜行うことにより、上記のユーザ要求が満足されると考えられる。

次に、STAでは(2)を実現するために

- ・デバッグデータの可視化機能

を検討している。また、(3)を実現するために

- ・エディタを中心とした統合環境の実現

を検討している。上記2つの機能に関して4.2.2及び4.2.3で詳述する。

4.2.2 並列プログラムデバッガにおけるデータ可視化機能の実現

3章において我々は並列プログラムデバッガの問題点として、以下の2点、すなわち

(1)Probe効果の存在による再現性の欠如

(2)ユーザが認識しやすい形態での大量データの提供手法

を指摘した。これらの問題点は、途切れのない思考の実現に密接に関連している。例えば、Probe効果によってエラーの再現性が保証されなければ、デバッガという作業を行うために何度もプログラムの試走を行う必要がある。これはデバッガ作業を開始してからエラーを発見するまでの一連の作業の流れを著しく阻害する可能性を持つ。また、大量データが何も加工されずユーザに提供されると、大量データから必要な情報を認識しなければならないため、"認識、判断・・・、実施"という作業の手順を円滑に実施することが困難である。従って、並列プログラムのデバッガ作業における途切れのない思考を支援するためには、上記2つの問題を解決することが重要である。

我々はこのうち(2)の問題を解決するためのアプローチとして、現在可視化システムと連携したデバッガ(可視化デバッガ)の構築を検討している。以下、この可視化デバッガについて述べる。

4.2.2.1 データ可視化機能

一般に、作成したプログラム中に存在するエラーを修正するデバッグ作業は、プログラム開発作業の一つフェーズとして位置づけられ、その作業を支援するツールとして各計算機ベンダーからデバッガが提供されている。しかしながら、科学計算分野では他の分野と比較してデバッガの使用率が低い。これは、プログラムの実行を途中で中断し、その時点における変数値を出力することによりプログラム実行状況を推測するという一般的なデバッグスタイルが、科学計算プログラムのデバッグに対して有効でないためである。その原因として、以下の2点が挙げられる。

- (1)科学計算は連立微分方程式を数値的に解く事により、現象のメカニズムを計算機内に再現する手法であるため、ユーザは個々の変数の具体的な値が正しいかどうかを判断できない。
- (2)個々の変数の具体的な値が仮にわかっているとしても、科学計算では大規模な配列を利用して計算を行うため、配列中の変数値をそのまま出力する現在のデバッガでは数値チェックのための労力が大きい。

この問題点を解決するために必要とされるデバッガの機能に関して考察する。

一般に、科学計算では計算対象とする現象を離散化し計算を行う。その過程において、物理量は実際の物理空間から計算空間へ射影される。すなわち、物理量を格納した配列を物理空間に関する情報(例えばメッシュの配置に関する情報)に関係付けることにより計算が行われる。ユーザが理解しているのは計算対象である物理現象を支配しているメカニズム(物理法則)である。従って、プログラムが正常に動作しない場合、ユーザにとってまず必要な情報は、物理空間における物理量の分布等の傾向に関する定性的な情報であり、この情報を用いて物理法則を満たしているかどうかをチェックすることにより、ユーザはプログラムのどの部分にエラーが存在するかに関する手がかりを得る。従って、デバッガは、

- (1)物理量の格納された変数配列とメッシュ等の情報の格納された配列を利用して、物理空間における物理量に関する定性的な情報を提供する必要がある。

この定性的な情報の提供手法という観点から考えれば、画像という情報提供形態は非常に優れているといえる。実際、科学計算分野における結果解析では、この特徴を活かした可視化解析が主流となってきており、専用の可視化アプリケーションを利用したり、可視化ライブラリを利用して物理量を可視化することが従来から行われている。

従って、デバッグ作業においてもライブラリを利用して物理量を可視化することが考えられる。しかし、デバッグ作業は最初からどの部分にエラーが存在するかが明確な場合は稀であり、試行錯誤的にエラーの原因を究明していくのが一般的である。また、ある変数をチェックすることによって初めて次にチェックすべき変数が明らかになってくるというダイナミックなプロセスもある。このような特徴を持つデバッグ作業に対して、ライブラリ方式ではチェックすべき変数を変更する度にプログラムを書き直し、再コンパイルし、さらに最初から再実行しなければならない。従って、デバッガは

- (2)ユーザのチェックしたい変数が明確化された時点で、インタラクティブな指定により、可視化できる機能を持つ必要がある。

(1), (2)をまとめれば、デバッガには

ユーザがチェックしたい変数、物理空間に関する情報、利用したい可視化手法をインタラクティブに指定することにより可視化できる機能が必要である

ということになる。

4.2.2.2 システム構成

上記の機能を実現するために、我々はデバッグサブシステム、制御サブシステム、可視化サブシステムの3つのサブシステムから構成される可視化デバッグシステムを検討している。システム構成及び各サブシステム間のデータの流れを図4.2.2に示す。

各サブシステムの機能は、以下のとおりである。

(1) デバッグサブシステム

ユーザの指定した変数値を制御サブシステムに送付する。

(2) 制御サブシステム

ユーザからの要求に基づき、デバッグサブシステム、可視化サブシステムの制御を行う。

(3) 可視化サブシステム

制御サブシステムから送付されたデータを基に可視化を行う。具体的には、個々の可視化手続き(モジュールと呼ぶ)をGUIを用いてグラフィカルに連結することにより可視化を行うことのできるビジュアルプログラミング機能を有するAVS(Application Visualization System)システムを利用する。

4.2.2.3 可視化テンプレート

制御サブシステムと可視化サブシステムの連携を実現するためには、デバッグサブシステムから送付されるプログラム内の変数データと可視化サブシステムが描画のために利用する変数データとの対応付けが必要である。例えば、折れ線グラフを用いてデータの分布を調べる場合、Y軸方向の量に関する変数データとX軸方向における間隔を表す変数データが必要である。この対応付けを行う手段として、テンプレートという可視化手法を記述した一種のひな型を利用する。テンプレートは可視化手法毎に作成され、その可視化手法を実現するために必要なデータ(上記例では、Y軸方向の量に関するデータ及びX軸における変数データ)が宣言されている変数対応記述部及び描画スクリプト部(上記例では、折れ線グラフ描画プログラム)から構成される(図4.2.3.参照)。変数対応記述部には表4.2.1.に示す情報が記述される。描画スクリプト部には、具体的な可視化のための描画プロ

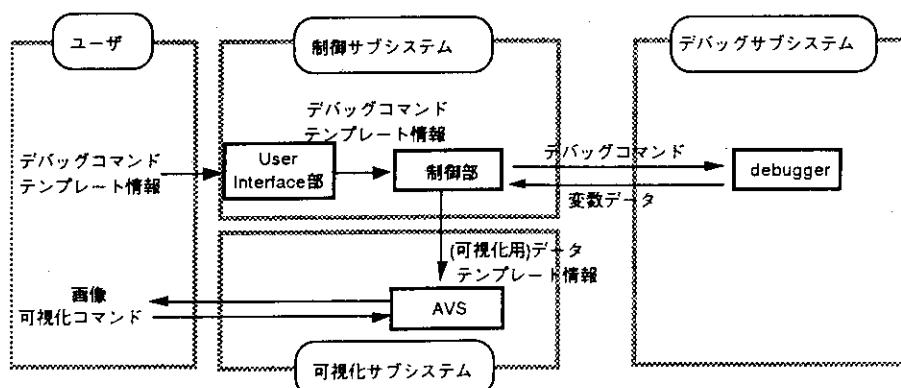


図 4.2.2 可視化デバッグシステムの構成

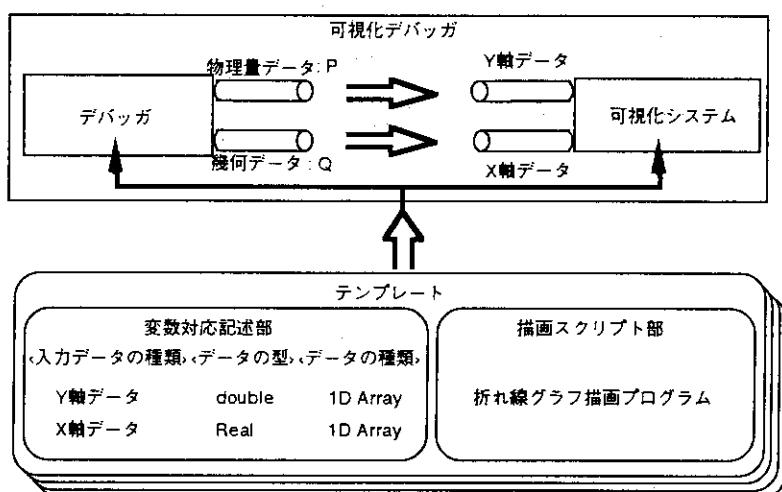


図 4.2.3 可視化テンプレートの構成

表 4.2.1 可視化テンプレートにおける変数対応記述部記述内容

変数対応部記述内容	意味
入力データの区分	入力データは描画対象となる変数データとその変数を描画する幾何空間に関するデータの2種類に区分される。
入力データの型	入力データの型を示す。Int, Real, Double, Stringのいずれかの型をとる。
入力データの種類	入力データが配列変数であるかスカラー変数であるかを示す。
入力データの次元数	入力データが配列変数である場合、配列変数の次元数を示す。
入力データの意味	入力データが利用者にとって何を意味するのかを記述する

グラムが記述される。

可視化テンプレートは以下のような形態で実現される(図4.2.4参照)。すなわち、テンプレート記述ファイルと呼ばれるファイルに、変数対応記述部の内容を記述する。描画スクリプト部はAVSのモジュールネットワークによって実現されるものとし、テンプレート記述ファイルには、モジュールネットワークに対応するCLI(Command Language Interpreter)スクリプトファイル名を記述する。このファイルを制御システムが読み込むことにより、各種メニュー画面を介して可視化する変数データの指定をユーザに促すと共に、指定されたCLIスクリプトファイルを用いてモジュールネットワークを起動する。

科学計算においてユーザの可視化対象となるデータには種々のものがあり、またそれに対応する可視化手法も様々なもののが存在する。従って、全てのデータや可視化手法に対応するテンプレートを効果的に利用できるようにするためにユーザ自身が簡単にテンプレートを定義、登録し、システムを自由に拡張、カスタマイズできる必要がある。

本システムでは、ユーザのレベルとして以下の3レベルを想定している。

(1)システムに当初から登録されているテンプレートのみを利用してデータの可視化を行う。

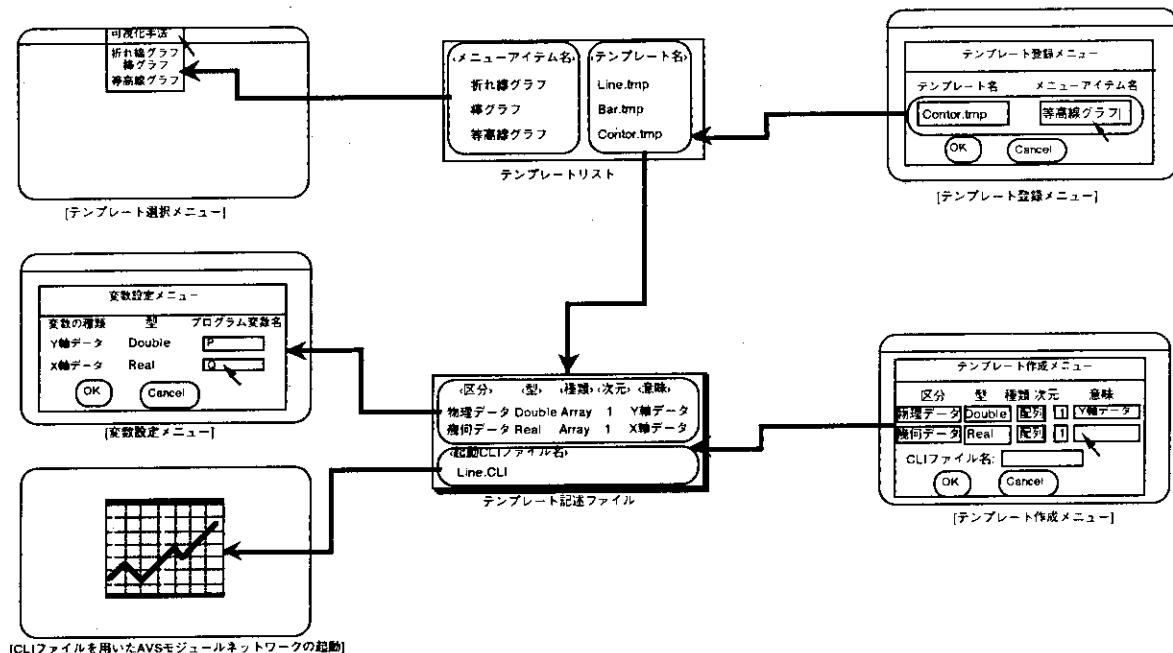


図 4.2.4 テンプレート記述ファイルに格納されるデータ内容と各種メニュー画面

- (2) AVSの既存のモジュールを利用して独自のモジュールネットワークを構成し、それをテンプレートとして登録し、独自のデータ可視化処理を行う。
- (3) ユーザ独自のモジュールを作成し、それを利用してモジュールネットワークを構成し、テンプレートとして登録、独自の可視化処理を行う。

ここで、(2), (3)のレベルのユーザを支援するためには、以下の5つの機能

- (i) テンプレート記述ファイル作成支援機能
- (ii) テンプレート記述ファイル登録機能
- (iii) CLIファイル作成支援機能
- (iv) AVSモジュールネットワーク作成支援機能
- (v) AVSモジュール作成支援機能

が必要となる。このうち、(iii), (iv), (v)はAVSシステム自体に各種支援機能が提供されている。

(i), (ii)の機能の実現を現在検討している段階である。

4.2.2.4 今後の予定

可視化デバッグシステムを用いた並列プログラムデバッグ作業において、途切れのない思考の支援を実現するためには、以下の点の実現が重要であると考えられる。

(1) Probe機能の実現

一般に、科学計算のプログラムはループによって同一処理を何度も実行することにより計算が進行する構造となっている。従来のデバッガではブレークポイントを設定することによりプログラムの実行を一時停止して変数の値を検査していたが、ループ構造を持つプログラムに対してループ内の変数の値を検査しようとすると、ループ内の処理を実行する毎にプログラムが停

止してしまう。従って、ブレークポイント方式のデバッグスタイルは有効でない。この問題に対しても、プローブポイントと呼ばれるデータチェックポイントの設定が有効であると考えられる。プログラムの実行がプローブポイントを設定した箇所に到達すると、デバッガはユーザの指定した変数のその時点における値を出力するが、停止することなく実行を継続する。このプローブポイントの設定により、ループ内における変数の値の検査に必要な作業量が軽減される。

また、プローブ機能と変数可視化機能を組み合わせることによって、ユーザは目的とする変数値の変化をダイナミックに確認することができ、簡易なトラッキング機能が実現される。トラッキング機能の実現により、ユーザは並列処理の処理の流れをプログラムの変数の変化を通して実際に理解することができる。

(2)可視化デバッグ用テンプレートの整備

代表的なテンプレートをあらかじめ整備することにより、初めて可視化デバッグ機能を利用するユーザの負担を軽減できる。そのためには、並列プログラムのデバッグにおいてどのようなデータをどのような可視化手法を用いて可視化するかを考察する必要がある。

4.2.3 エディタを中心とした統合環境の実現

4.2.3.1 目的

4.1で述べたように、我々はSTAによって、"並列プログラム開発における一連の行動の流れを円滑かつ効率的に支援することにより並列プログラム開発全体を支援する"ことを考えている。並列プログラム開発全体の流れを考えた場合、その特徴の一つとして、各作業フェーズ(コンパイル、デバッグ、チューニング等作業フェーズを指す、以下プログラム解析フェーズと呼ぶ)で作業を繰り返す場合、繰り返しの度に必ずプログラム作成修正フェーズに戻る(フィードバックと呼ぶ)ことが挙げられる。

例えば、デバッグ作業ではデバッグ作業を行ってエラーを発見した場合、必ずそのエラー個所の修正作業が発生する。その場合、プログラム解析フェーズからプログラム修正フェーズに戻る時点では、既に修正個所が判明しているはずである。しかし、現在提供されているエディタはプログラム解析フェーズで利用するツールとは独立になっており、修正すべき個所に関するいかなる情報も伝達されない。そのため、ユーザは各ツールから得られた修正すべき個所に関する情報を自分で覚え、エディタに対してその個所を指定する必要がある。これは一連の行動の円滑かつ効率的な流れを阻害する原因となる。本来、各ツールによるプログラムの解析でプログラムの訂正個所が判明したら、どのように修正すべきかに関する情報のみをユーザは覚えておけばよい。そのためには、エディタとプログラム解析フェーズで利用するツールとの連携を緊密にし、種々の情報をやり取りすることによりユーザのプログラム開発作業を支援することが重要である。その際、

(1)エディタとどのようなツールを連携させるべきか

(2)エディタとツール間でどのような情報をやり取りすべきなのか
を考察することが大きな課題となる。

現在、我々は途切れのない思考の支援を実現するために、上記2点を考察し、エディタを中心とし、そのエディタと各ツールとを連携させたシステムアーキテクチャの並列プログラム開発環境構築を検討している。本開発環境は現在必要機能の検討を開始したばかりであり、明確化されていない

いが、現時点において必要と思われる機能について述べる。

4.2.3.2 実現機能

本開発環境において実現されるべき機能は2種類に大別される。一つは上述のエディタと他プログラム開発支援ツール間の連携の強化である。もう一つはエディタ自身の編集機能の強化である。以下に具体的項目を挙げる。

(1)他プログラム開発支援ツールとの連携強化

- ・プログラム解析結果、実行結果の表示機能

既存プログラムに対し、プログラム解析結果と連動し、ソースプログラムに添って解析結果を表示する(コンパイラとの連携)。さらに例えばCPU利用率のレンジに従ったステートメントのハイライト表示、色分け表示等視覚的に情報を提供する。

- ・ループ構造表示機能

コンパイラと連携し、ループ構造の階層に従って、ソースプログラムのループ部分をハイライト表示、色分け表示する等によりループ構造を明確化する。また、特定階層を表示しない機能を提供する。また、プログラム実行結果との連動によりループの繰り返し回数にしたがった色分け表示の機能を提供する。

- ・変数・配列情報表示機能

コンパイラと連携し、ソースリスト上で指定した変数・配列をハイライト表示、色分け表示する等により、使用されている箇所を明確化する。また、プログラム解析ツールと連携し、変数の依存関係を色分け表示する等により明確化する。

- ・編集サブルーチンナビゲーション機能

プログラム解析ツールと連携して、コールツリー上のサブルーチン名をマウスなどで指定することにより、指定したルーチン編集画面に切り換える機能を提供する。また、エディタ自身の機能強化として、プログラムリスト中のサブルーチンのコール分をマウスなどで指定することにより、指定したルーチン編集画面を切り換える機能を提供する。

- ・変数・配列ナビゲーション機能

コンパイラと連携して、指定された変数・配列が使用されているソースリスト部分に編集画面を切り換える機能を提供する。画面の切り替えは、変数・配列が更新されている箇所あるいは参照されている箇所に限定する機能も提供する。さらに、更新・参照の関係に従ってナビゲートする機能も提供する。

(2)エディタ自身の編集機能強化

- ・ノード固有処理の選択表示機能

実行するノードにより処理の切り替わる部分では、IF文による処理切り替えが行われる。このノード固有処理部分に対し、ノード別にハイライト表示、色分け表示等により構造を明確化する。また、特定ノードのみを表示する機能を提供する。

- ・配列分割支援機能

配列の分割方法(ブロック分割化サイクリック分割か)や分割するインデックスなどの情報をエディタが管理し、これを用いてループを分割するための処理や定型的なメッセージ通信の処理のひな型を自動作成する機能を提供する。

・メッセージ通信ライブラリ入力支援機能

メッセージ通信ライブラリ関数の入力に際し、宛先ノードなどの指定をGUIを用いて入力可能とすることにより、入力ミスの削減と入力の簡易化を図る。

これらの機能を実現することにより、並列プログラム開発の一連の流れにおける途切れのない思考の支援が図られるものと考えられる。

参考文献

- [1] 公文俊平：情報文明論，NTT出版（1994）
- [2] J. Hennessy, D. Patterson. 富田眞治, 村上和彰, 新賽治男訳：コンピュータ・アーキテクチャ, 設計・実現・評価の定量的アプローチ, 日経BP出版センター（1990）.

5. おわりに

日本原子力研究所に設置されている並列計算機の並列プログラミング環境については、2. 9節のまとめ等から以下の点が指摘される。

- (1)個別ハードウェア・アーキテクチャに密接に対応した並列化方式が採用されている。即ち、分散メモリ・スカラ並列ではメッセージ・パッシング方式、分散メモリ・ベクトル並列ではデータ・パラレル及びメッセージ・パッシング、共有メモリ・ベクトル並列では共有変数、分散・共有メモリ・ベクトル並列では共有変数及びメッセージ・パッシングである。即ち、並列プログラミング・モデルの抽象化レベルは十分でなく、これら異なったアーキテクチャを論理的に包含する抽象度の高い仮想並列計算機モデルは未だ提案されていない。
 - (2)Cray T94及びHitachi SR2201では、ソース・コードのデータ依存解析機能（データ・パラレル・モデル）等により、逐次プログラムをそれぞれ自動ベクトル／並列化及び自動並列化するツールを提供している。今後、これらのツールの使用性を含め性能評価を体系的に実施すべきである。
 - (3)データ・パラレル性の自動検出はベクトル計算機の経験を活用できるが、比較的粒度の大きいタスク・パラレル性の自動検出は今後の課題といえる。
 - (4)ハードウェア・アーキテクチャに最適マッピングする自動並列化を目指した段階的な研究開発の手順を開発する必要があろう。
 - (5)デバッガについては、すべてブレーク・ポイント・ベースのものであり、プローブ効果をより低減するイベント・ベースのものは提供されていない。また、デバッグ情報を理解し易くするための、画像化表示機能なども具備されたものはない。
 - (6)性能評価ツールについては、ハードウェアに対して直接モニタリング方式でデータを収集し、実行終了後に解析するPostmortemなツールが殆どである。性能評価は、通信時間、演算時間、同期のオーバーヘッド時間、実行レベルでのアイドル時間などの膨大な実行データによって遂行されることから、実時間による評価は困難な面もある。分散メモリ・スカラ並列計算機に期待されているスケーラビリティ特性の評価機能は、特に重要であろう。最近は、また個別応用ソフトに対して、実行中の結果を監視しながら（トラッキング）、入力パラメータを変更して、並列実行を制御（ステアリング）するツールが開発されている。
 - (7)統合化ツールについては、機能統合を狙ったものでコンパイル、デバッグ、性能評価機能を統合したものが提供されている。ヒューマン・インターフェイスを配慮した並列プログラム開発の全過程をカバーするような統合化ツールは提供されていない。
- 並列プログラム開発支援ツールの研究開発動向としては、概略、上記の現用並列プログラミング開発支援ツールで欠如している機能等を拡充する研究開発が進展しているといえる。即ち、並列処理言語及びコンバイラに関する研究開発動向として、(1)並列プログラミングにおいて、メッセージ・パッシング関連の標準化インターフェイス、MP Iの仕様が確定し、実用化されつつあり、今後はこれを利用した並列プログラム開発が進展しポータビリティの改善が図

られること、(2)タスクパラレルを支援するFORTRAN Mの開発、(3)配列演算等を強化したデータ・パラレル化を内包した現行規格FORTRAN90の後継として、非同期並列実行による結果の再現性が保証されない現象を防止したFORTRAN 95の規格化の進展、(4)FORTRAN D及びVienna FORTRANの研究開発成果等に基づく、HPF: High Performance FORTRANのVer.1の仕様が確定し、実用化されつつあること、(5)並列化コンパイラに関しては、逐次FORTRANプログラムを多様なアーキテクチャを対象に、並列プログラムに変換するための再構成ツールの開発などが注目される。

デバッグに関する研究開発動向としては、プローブ効果による再現性の欠如を軽減したデバッグの開発及び膨大なデバッグ情報を利用者へ如何に理解し易く提供するかの手法の研究開発等が挙げられる。

性能評価／チューニング支援に関する研究動向として、データの容易な理解を狙った円／棒グラフ等の簡単な可視化表示機能、操作性向上を狙ったウインドウ・システムの活用、実行後解析が主流を成しているなか、アイドル／ビジーなど簡単なデータのオンライン表示機能を具備したツール開発が進展している。

ツール統合型プログラミング環境に関する研究開発動向として、エディタ、コンパイラ、ソース・コード解析ツール、性能評価ツール、デバッグなど並列プログラム開発のあらゆる局面での支援ツールを統合した環境の開発が進展している。

また、日本原子力研究所で開発中のツール統合プログラミング環境STAも、開発者の途切れのない思考を支援するという独自の概念の基に開発が進められていることから、早期の実用化が期待されるところである。

最後に、日本原子力研究所に設置の並列計算機の並列プログラム開発支援ツールのレビューについては、ツール類を必ずしも実際に使用して行ったものではなく、また研究開発動向のレビューについても、著者等が入手できた資料の範囲に留まったもので、果たして問題点の把握が的確で、今後の研究開発に資する資料を提供できたかどうかは、本報告書を読まれる識者の判断に委ねたい。

謝 辞

計算科学技術推進センター並列処理支援技術開発グループ・グループリーダーの相川氏には、本稿査読の際、細部にわたり貴重なる助言を頂いた。深く感謝いたします。

られること、(2)タスクパラレルを支援するFORTRAN Mの開発、(3)配列演算等を強化したデータ・パラレル化を内包した現行規格FORTRAN90の後継として、非同期並列実行による結果の再現性が保証されない現象を防止したFORTRAN 95の規格化の進展、(4)FORTRAN D及びVienna FORTRANの研究開発成果等に基づく、HPF: High Performance FORTRANのVer.1の仕様が確定し、実用化されつつあること、(5)並列化コンパイラに関しては、逐次FORTRANプログラムを多様なアーキテクチャを対象に、並列プログラムに変換するための再構成ツールの開発などが注目される。

デバッグに関する研究開発動向としては、プローブ効果による再現性の欠如を軽減したデバッグの開発及び膨大なデバッグ情報を利用者へ如何に理解し易く提供するかの手法の研究開発等が挙げられる。

性能評価／チューニング支援に関する研究動向として、データの容易な理解を狙った円／棒グラフ等の簡単な可視化表示機能、操作性向上を狙ったウインドウ・システムの活用、実行後解析が主流を成しているなか、アイドル／ビジーなど簡単なデータのオンライン表示機能を具備したツール開発が進展している。

ツール統合型プログラミング環境に関する研究開発動向として、エディタ、コンパイラ、ソース・コード解析ツール、性能評価ツール、デバッグなど並列プログラム開発のあらゆる局面での支援ツールを統合した環境の開発が進展している。

また、日本原子力研究所で開発中のツール統合プログラミング環境STAも、開発者の途切れのない思考を支援するという独自の概念の基に開発が進められていることから、早期の実用化が期待されるところである。

最後に、日本原子力研究所に設置の並列計算機の並列プログラム開発支援ツールのレビューについては、ツール類を必ずしも実際に使用して行ったものではなく、また研究開発動向のレビューについても、著者等が入手できた資料の範囲に留まったもので、果たして問題点の把握が的確で、今後の研究開発に資する資料を提供できたかどうかは、本報告書を読まれる識者の判断に委ねたい。

謝 辞

計算科学技術推進センター並列処理支援技術開発グループ・グループリーダーの相川氏には、本稿査読の際、細部にわたり貴重なる助言を頂いた。深く感謝いたします。