プログラミング 12−7
（1996. 11. 1）

同期モデルに基づく自動並列化コンパイラ

金山 二郎　飯塚 肇

成蹊大学大学院工学研究科情報処理専攻

概要

本稿では、同期モデルに基づいた並列プログラミング言語 log$l$ の設計と実装について述べる。

log$l$ は一個の共有メモリと共通クロックを仮定するアーキテクチャ独立な言語である。変数の集合に対する並列実行構造を持つ。また、メモリアクセスのコンフリクトについて、複数の解消法が用意されており、問題に適切なモデルを選択することで、並列プログラムを簡潔に記述することを可能にしている。プログラマは、オペレーション間の変数の並列性を記述することによってプログラムを構築する。

Autoparallelizing compiler based on a synchronous parallel computation model

Jiro KANAYAMA　Hajime IIZUKA

Department of Information Science,
Graduate School of Engineering,
Seikei University

abstract

This paper describe that design and implementation of parallel programming language log$l$ that is based on synchronous parallel computation model.

log$l$ is an architecture-independent language which assumes that the machine has one centralized shared memory. The language has a parallel execution structure for some sets of variables. In addition, log$l$ has a choice of several memory models and access conflict resolving scheme. The ability select appropriate model for each problem makes parallel programming.

1 はじめに

近年、並列計算の研究開発が盛んであり、独自のプログラミング方式を備えた様々なアーキテクチャモデルの計算機が存在している。そのような状況において、ある程度の実行効率を保ちつつ、アーキテクチャモデルに依存しない、汎用的な並列プログラム記述手法の開発が切望されている。

しかし現実には、PVM や pthread など、アーキテクチャモデルに依存した並列ライブラリを用いるのが一般的となっている。また、同じアーキテクチャモデルにも数多くの並列ライブラリ/言語が存在し、結果として、同じ問題に対する夥しい数のアプリケーションプログラムが存在することになった。この状況に対して、同期モデルに基づいた並列プログラミング言語 **log l(“localize over global” language)** を設計し、各種並列計算機に実装することによる解決を試みた。

アーキテクチャモデルが同期性を提供していない限り、並列プログラミング言語のほとんどすべてが非同期モデルを採用している。log l では、プログラミングを困難にする大きな要因であり、並列プログラマ共通の悩みである実行順序の同期の管理からユーザを開放している。

log l はオペレーションレベルの共通クロックを仮定し、同期の管理からユーザを開放している。プロセッサ、プロセスの概念はなく、変数集合に対する**並列化可能オペレーション**を定義する制御構造を持つ。また、どのオペレーションにも共通なオブジェクトである、一個の共有メモリを仮定し、オペレーション間の通信を解決している。また、共有メモリは複数のメモリモデルとアクセス競合の解消法をサポートしている。このことにより、問題に適切なモデル/解消法を選択することで、簡潔かつ詳細な並列プログラムの記述を可能にしている。

log l は操作的言語であるが、並列構造に関しては宣言的な性質を持つ。アーキテクチャ依存のハンドオプティマイズの除去と、詳細な並列性記述を許す文法が、コンパイル時の最適化を容易かつ強力にしている。

本稿では、特に分散環境に着目し、現時点で有力とされる並列ライブラリ/言語の特徴について示す。そして、それらの問題点を元に、log l の設計/実装/評価について示す。

2 分散環境における並列ライブラリ/言語

並列処理を行えるプログラミング言語は数多く存在する [1]。特に、実装の容易さから数多くのハードウェアが存在する分散環境においては、様々なアプローチの並列ライブラリ/言語が存在する。これらのソフトウェアは、主に実行効率の向上と記述の容易性から図 1 のように分類できる。

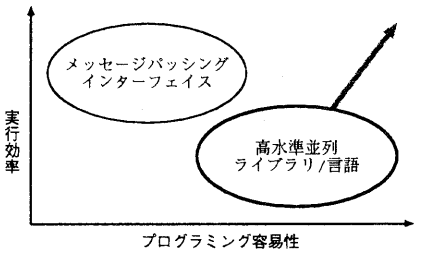

図 1: 並列処理言語の分類

並列処理の課題として最重要視されるのは、実行効率の向上である。したがって、ハードウェアを扱うためのソフトウェアは、ハードウェアの特性を最大限に利用できることが望ましいとされてきた。その点において、プロセッサ間通信を実現するハードウェアを直接的にサポートするメッセージパッシングインターフェイス **(Message Passing Interface: MPI)** は、事実上、分散環境における並列ライブラリ/言語の標準として位置付けられる。

しかし、扱う問題が複雑かつ大規模になるにつれ、プログラミングの困難さが問題視されるようになってきた。そこで、実行効率を保ちつつ、プログラマの負担をいかに軽くするかという課題について、様々なアプローチの研究が台頭してきた。

それらの研究の中で特に有力とされるアプローチが、**(ソフトウェア) 分散共有メモリ (Distributed Shared Memory: DSM)** と呼ばれる機構である。これは、各プロセッサが一個の共有メモリを利用できると仮定し、実際にはデーモンが共有メモリの一貫性保持を担うことで、通信に関するプログラマの負担を軽減しようという狙いを持つ。ただし、その設計については、未だ分類が不正確な点がある。

また、データオブジェクトが演算の際に持つ並列性に着目した、いわゆる**データ並列性**の概念に基づく研究も盛んである。これは現在、並列 Fortran というひとつの大きな勢力を形成するに到っている。

本章では、これらのアプローチの例として、PVM、TreadMark、HPF について解説する。

2.1 PVM

分散環境では、各プロセッサは、他のプロセッサへの通信機構を装備している。これを直接的に利用し、複数プロセスがデータを送受信するための関数群が MPI である。数々の実装が存在したが、現在、汎用性が高く異機種間結合をサポートしている **PVM(Parallel Virtual Machine)**[2] が主に使われている。また、MPI の標準である MPI2[3] が有力視されているが、現段階では広範な普及には到っていない。

MPI は一般にランタイムライブラリとして提供され、実際のメッセージの送受信は、各プロセッサ上で動作する専用のデーモンが担当する。

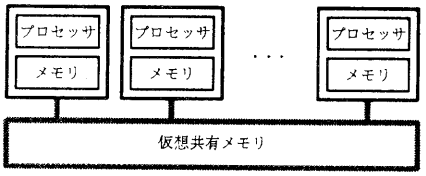

図 2: DSM

この機構が提供するのはインターフェイスのみであり、データの分割や移動などはすべてプログラマの判断で行われる。また、各プロセッサの動作は一般に非同期であることから、ユーザは必要に応じて同期を行う必要がある。

以上のように、多くのことがユーザに一任されていることから、プログラミングの困難さが欠点として挙げられる。しかし、逆に低レベルでのハンドオプティマイズが可能であることから、実行効率の向上という最も基本的な目標について、大きな可能性を提供している。また、機構自体の実装の容易さも大きな利点として挙げられる。

PVM の特徴として、メッセージパッシングインターフェイスとしては比較的早期に登場し、異機種間通信をサポートしている。また、ネットワーク結合されたワークステーションや著名な並列計算機のほとんどすべてに実装されている。

2.2 DSM

分散共有メモリシステム (DSM) は、ユーザが分散メモリを一つの共有メモリとして扱えるようなシステムである (図 2)。

このシステムでは、局所メモリを持つ数台のプロセッサが、通信機構の替わりに一つの共有メモリを持つシステムを仮定してプログラミングを行う。共有メモリの一貫性は、各プロセッサ上で動作するデーモンによりなされる。

プログラミングスタイルとして、分割を要するようなデータは共有メモリに置き、局所メモリは作業用として使う。ユーザは特に通信の束縛を受けずにプログラミングできるので、プログラミングの困難さは著しく解消される。

また、通常、プログラムの流れを大雑把に制御するために、バリアなどを行う関数が用意される。

ソフトウェア・ハードウェア両方の実装が開発されているが、特にソフトウェアとして、TreadMark, Midway など多くのシステムが存在する。

このシステムの評価は、プログラミングの容易さと実行効率とのトレードオフであり、そのバランスについても議論されている。

実装の特徴として、一般に、コンパイル時の最適化は行われないということが挙げられる。これは、仮想共有メモリの一貫性保持に研究の主眼が向けられていることに起因する。また、逐次アルゴリズムがほぼ無修正で動作することも大きな特徴であるが、ハンドオプティマイズを施されたプログラムが必ずしも良好な実行性能を示すとは限らない。

さらに、実行効率を考慮したコンパイル時のデータ配置・ポインタ渡しの解決など、いくつかの大きな課題を残している。

2.3 HPF

データ並列性 [4] は、もともとは SIMD 型計算機における並列動作の軸となる概念として発生したが、現在ではループのような構造におけるデータ間の並列性という概念に一般化されている。

並列 Fortran の核は、配列データのループ演算におけるデータ並列性解析に基づく演算の並列化である。プログラムは基本的にシングルスレッドであり、そこから並列性を抽出するのはコンパイラの役目である。

並列言語としては最も活発な分野であり、ポスト MPI として最も成功を収めていると言える。

HPF 言語 [5, 6] は、数ある並列 Fortran の標準化として開発された。1991 年より組織化され、現在、ようやく実用レベルの処理系が利用できるようになってきた。基本的には Fortran90 の仕様を継承しつつ、並列実行をサポートしている。ただし、そのアプローチは MPI の理念に基づいており、データ分割などはユーザが担当する。このために、データ分散、動的再配置を行う命令が用意されている。

この他、商用の並列 Fortran である VPP Fortran[7] は、データ分割を不要とする仮想的なグローバルメモリをサポートしており、本研究に最も近いアプローチをとっている。

3 設計

本章では、log l の設計について記す。log l は、基本的にはデータ並列言語として位置付けられ、VPP Fortran に似た思想のもとに設計されているが、並列実行部に大きな違いを持つ。

log l は図 3 のような構成を仮定している。

並列 Fortran は、並列実行用のプロセッサプール

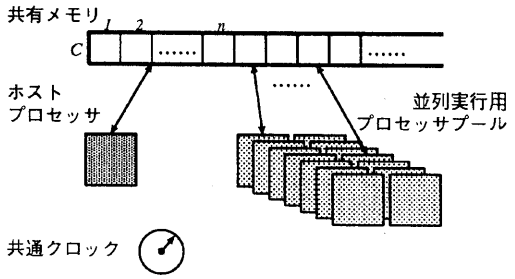

図 3: log l が仮定する構成

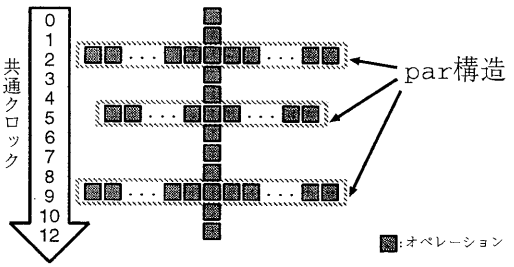

図 4: log$l$ の意味的な実行の様子

を仮定していない。すなわち、並列 Fortran は完全にシングルスレッドのプログラミングスタイルを持つ。これに対し、log$l$ では並列実行部に限りマルチスレッドのプログラミングスタイルを持つ。この並列実行部は、単位時間において完全に並列に行われることを想定している (図 4)。

このことにより、コンパイラの並列性抽出機構の負担はより軽減される。すなわち、並列 Fortran が配列演算ループを展開した状態での依存性解析を強いられていたのに対し、前方/後方の依存性解析が不要となり、依存性はアクセスコンフリクトという形に集約される。

3.1 共有メモリ

log$l$ が仮定している仮想共有メモリは、マルチメモリモデルをサポートしている。また、アクセスコンフリクトに関していくつかの解消法を用意し、プログラミングを補助している。問題に合わせてこれらのモデル/解消法を選ぶことにより、簡潔かつ詳細な並列プログラミングを可能にする。

サポートしているメモリモデル/ライトコンフリクト解消法は以下の通り。

EREW 排他読出/排他書込
ERCW 排他読出/同時書込
CREW 同時読出/排他書込
CRCW 同時読出/同時書込

- **COMMON** 並列にライトされる全ての値が同一でなければその値が書き込まれ、さもなくばエラーが起こり、計算停止。
- **ARBITRARY** 任意のライトが無作為に選択される。
- **PRIORITY** 高いプライオリティを持つライトが成功する。
- **COMBINING** 書き込まれる値は、並列ライトの、すべての値の線形な組み合わせ (例えば、値の総和)。

一般の計算モデルでは CREW モデルは扱われないが、これは現実にそのような並列計算機はナンセンスであるためである。これを採用した理由は、ハードウェアとしての実装形態として考えにくくとも、詳細な並列アルゴリズム設計という指針において有用であると判断したためである。

また、リードアクセスに何らかの制限を加える文法について考察中である。

なお、共有メモリに関するアイデアの多くは、PRAM (Parallel Random Access Machine)[8, 9, 10] から取り入れた。PRAM は、並列計算機の仮想モデルとして早期から導入されていた。また、メモリアクセスコンフリクトについての解消法が明確に定義されており、言語向きの明瞭な仕様を持っている。

3.2 par 構造

log$l$ における明示的な並列実行部を表す制御構造として **par 構造**がある。par 構造は、「集合 X の各要素 p に対して、*OPERATION*(p) を実行する」という意味を持つ。

$$\textbf{par}\ p \in X\ \textbf{do}\ \mathit{OPERATION}(p)$$

各要素 p は、具体的には配列の添字として機能し、配列要素はリード/ライト両方のアクセス可能性を持つ。

列は共有メモリ中に格納されており、log$l$ の並列実行部にて実行されることを仮定している。すなわち、par 構造内の各オペレーションは、基本的に順序関係を持たず、並列に展開/実行される。

代入文において、同じ配列要素が両辺に存在した場合には、一般的な操作型言語と同様、「左辺 → 右辺」という順序で評価される。基本的には左辺における演算は許さない。

OPERATION が複合文の形態であった場合、その複合文が単位クロックを消費する。

また、GOTO 文、繰り返し文の介在を許さないことで、プログラム全体における順序性を保持している。

条件文などで分岐される場合、現時点では、各分岐先で消費される実時間のバランスは考慮していない。したがって、シングルスレッドのプログラミングとはやや感覚が異なる。

なお、par 文に関するアイデアの多くは PRAM アルゴリズム [11] による。

4 実装

本章では、過去になされた log$l$ の pthread への実装 [12] と PVM への実装 [13] について要約する。

log$l$ はソースコードレベルのトランスレータであり、log$l$ コードは、log$l$ 処理系を通して pthread コード、PVM コードに変換される (図 5)。

ユーザはコマンドラインオプションとして、共有メモリのサイズやメモリモデルを指定できる (図 6)。

4.1 par 構造の並列化

並列化は par 構造に的を絞って行われる。その他の部分の実行や、実際にスレッド/プロセスを生成す

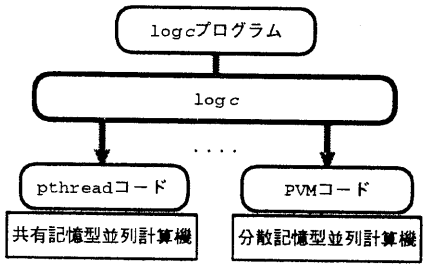

図 5: log$l$ の実装

```
logl -n360 -CRCW -COMMON sample.log smpl.c
```

図 6: コマンドラインオプションの例

る役割はホストスレッド/プロセスが担う。

4.1.1 pthread 版

ホストスレッドが、par 構造をいくつかの pthread に実行させる (ネストされていた場合、内側の par 文は逐次実行とする)。pthread 版において、par 構造は一般にループをなすが、スレッド分割を考えた場合、分割の仕方やインクリメントのオフセット値は、実行されるオペレーションに依存する。

スレッド分割数は、基本的に実際の計算機のプロセッサ数とする。この par 構造が終了するまで、次の par 構造の実行はブロックされる。これにより、一時に存在するスレッドの数はプロセッサ数以下であることが保証される。また、プログラムの意味の保持を par 構造の単位に分割することが可能となる。

4.1.2 PVM 版

pthread 版と同様、ある親プロセスを仮定し、そのプロセスが必要に応じた数のプロセスを生成させる。プログラムは SPMD(Single Program Multiple Data) の形式をとる。共有メモリの初期的な入力と最終的な出力も、このプロセスが属する計算機上のハードディスク上にファイルとして配置される。

データは、基本的には実際の計算機のプロセッサ数で分割される。pthread 版と同様、ある par 構造が終了するまで、次の par 構造の実行はブロックされる。par 構造の入れ子については、最初の par 構造で生成されたプロセスの範疇で実行される。

データ分割に伴い、各プロセスで行われる処理を決定する。通信を発生させて並列プログラムとしての補完を行うが、基本的には**通信回数を最小限に抑える**形での変換がなされる。

4.2 その他の並列化戦略

4.2.1 データ分割の判断基準

現在、1) 計算機の数 2) データ量 3) データ依存性を判断基準にデータ分割を行っている。最終的には、

```
par(i: 1<=i<=n){
    par(j: Key[i]<=j<=maxkey){
        key_density[j]++;
    }
}
for(i=1;i<=n;i++){
    Rank[i] = --key_density[Key[i]];
}
```

図 7: リストランキングのプログラム

4) ネットワークの形態 5) 計算機/ネットワークの性能も加味する。

4.2.2 局所メモリ

局所メモリの存在については、各プロセッサについて適当な大きさのものが装備されていると仮定している。pthread 版においても、他のスレッドから直接操作できないメモリ領域として存在を仮定している。

4.2.3 入出力インターフェイス

入出力に関しては、現在のところディスクに固定している。これは大量データ処理を考慮したためだが、将来的には他プロセスとのリンクを可能にするか、本格的な構造化プログラム言語とすることで単体での実用性を持たせるか、考慮の余地がある。

5 評価

本章では、分散環境における評価を示す。リストランキングの log$l$ プログラムを図 7 に示す。大文字で始まる変数は共有変数を表す。

`Key[]` にはあらかじめランキングの対象とされるキーが格納され、実際のランク付けは `Rank[]` に格納される。`maxkey` は最大のキー値であり、`key_density[0..maxkey]` の各要素 `key_density[n]` には、`n` 以下であるようなキーの総数が格納される。

5.1 入出力データの分割/収集

まず、`Key[]`, `Rank[]` は `p` 個のプロセスに均等に分割配置される。分割配置は親プロセスによりなされ、`Rank[]` も親プロセスに集められてファイルとして格納される。

5.2 キーデンシティの計算

通信回数を最小限に抑えるというポリシーの下で、`key_density[]` は一旦、各プロセスに配置された `Key[]` の部分リストについてなされる。その後、隣のプロセスに伝播する形で統合され、最終的な結果がブロードキャストされる (図 8)。

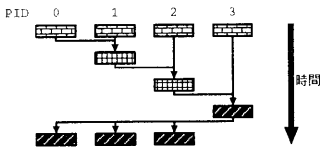

図 8: key_density[] の統合の様子

5.3 ランキングの計算

最終的に得られた key_density[] と、それを伝播する際に用いられた一時的なキーデンシティを利用して、各プロセス内の Key[] の部分リストについてランキングを計算する。

5.4 性能評価

生成された PVM コードの性能を、1) 逐次版, 2)TreadMark 版と比較する (図 9)。両プログラムは、いずれもキーデンシティを計算する形でのリストランキングを行う。また、TreadMark 版では key_density[] が共有配列としてデーモンにより管理される。Key[] として int 型のデータを 2^{20} 元用意し、同じソートを 10 回行わせた。

なお本稿では、ethernet で結合された Sun ワークステーション 8 台で構成された分散環境を想定した。

結果として、6 台程度から効果が見られた。TreadMark 版に比べてやや劣るのは、データ転送機構について最適化の余地があることを示している。しかし、変換された PVM プログラムが TreadMark 版のものに対してそれほどひけをとらなかったことから、性能として一定の水準を満たしていると考える。

6 おわりに

同期モデルに基づく自動並列化コンパイラ log$l$ の設計と、pthread 版、PVM 版の実装を行った。同期モデルの採用により並列プログラミングの難度を緩和するとともに、分散/共有環境において共通のプラットフォームを提供することで、汎用的な並列アプリケーションの作成を可能とした。また、性能的にも一定の水準を満たしていることを立証した。

データ分散/収集/統合アルゴリズムについて、現在では線形な方式を採用しているが、プロセッサ数が数十のオーダであった場合、性能的な欠陥となる。より汎用性の高いアルゴリズムへの変換方法を模索している。また、メモリ量が制約された条件下でのプログラム変換についても検討している。

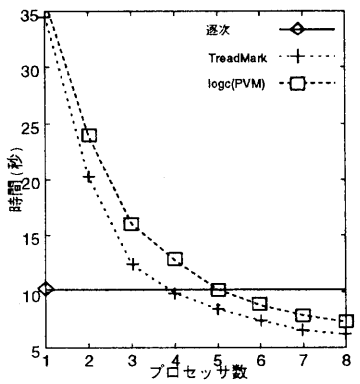

図 9: 性能の比較

参考文献

[1] 宮村勲：並列処理言語, マグロウヒル (1986).

[2] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. and Sunderam, V.: *PVM: Parallel Virtual Machine - A Users' Guide And Tutorial for Networked Parallel Computing*, The MIT Press (1994).

[3] Message Passing Interface Forum, : *MPI: A Message-Passing Interface Standard*, The MIT Press (1995).

[4] Hillis, W. D. and Steel, Jr., G. L.: Data Parallel Algorithms, *Commun. ACM*, Vol. 29, No. 12, pp. 1170–1183 (1986).

[5] 小柳義夫：ユーザからみた HPF 言語, 情報処理, Vol. 38, No. 2, pp. 86–89 (1997).

[6] 妹尾義樹：HPF 言語の現状と将来, 情報処理, Vol. 38, No. 2, pp. 90–99 (1997).

[7] 岩下英俊：HPF からみた VPP Fortran, 情報処理, Vol. 38, No. 2, pp. 114–120 (1997).

[8] Fortune, S. and Wyllie, J.: Parallelism in random access machines, in *Proc. 10th ACM Symposium on Theory of Computing*, pp. 114–118 (1978).

[9] Savitch, W. J. and Stimson, M.: Time bounded random access machines with parallel processing, *J. ACM*, Vol. 26, pp. 103–108 (1979).

[10] Goldshlager, L. M.: A universal interconnection pattern for parallel computers, *J. ACM*, Vol. 29, pp. 1073–1086 (1982).

[11] 宮野悟：並列アルゴリズム, 近代科学社 (1993).

[12] 金山二郎, 飯塚肇：PRAM プログラムから pthread プログラムへの変換, 情報処理学会第 52 回全国大会 (1996).

[13] 金山二郎, 飯塚肇：PRAM プログラムから PVM プログラムへの変換, 情報処理学会第 54 回全国大会 (1997).