計算機アーキテクチャ　128－15
ハイパフォーマンス　70－15
コンピューティング
（１９９８．３．６）

通信性能を考慮した自動並列化コンパイラのプロファイリング方式

金山 二郎　飯塚 肇
成蹊大学大学院工学研究科

概要

近年、計算速度の向上を目的とした並列プログラミングの研究開発が盛んである。これまで様々な並列計算機が登場しており、アーキテクチャ独立なプログラミング言語モデルを持つ自動並列化コンパイラの開発が切望されている。

しかし実際には、アーキテクチャ依存のプログラミング言語モデルが今だ主流である。その原因として、自動並列化コンパイラにおける、並列計算機の特性の把握と実行速度向上への反映の立ち後れが指摘されている。

本稿では、ネットワークの構成と性能をプロファイルすることにより、自動並列化コンパイラの生成する並列コードをチューニングする方式について示す。

Communication performance profiling method for autoparallelizing compiler

Jiro KANAYAMA　Hajime IIZUKA

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

abstract

In recent years, research of parallel programming become active. Until now, various parallel computers become conspicuous, thus development of autoparallelizing compiler that have architecture-independently programming language model is desired.

However, in fact, architecture-depend programming model still the main current due to delay at the start of the grasp of characteristics of parallel computer and the reflect of it.

This paper show a tuning method of parallel-code created autoparallelizing compiler by profiling structure and performance of network.

1 まえがき

近年、計算速度の向上を目的とした並列計算の研究開発が盛んであり、様々なアーキテクチャモデルの計算機が存在している。一般に、それらは独自のプログラミング方式を備えており、並列プログラムは並列計算機環境について個別に提供される傾向にある。

そのような状況において、アーキテクチャモデルに依存しない汎用的な言語モデルと、汎用化による実行効率の低下を極力抑えた自動並列化コンパイラの開発が切望されている。

自動並列化コンパイラとは独立に、通信性能を考慮してアセンブラコードをチューニングするトランスレータが存在する。ただし、これらは計算の構造を意識しないマクロ的な変換であり、本来コンパイラが行なうべき処理を後づけしたにすぎない。

本稿では、自動並列化コンパイラが各種並列計算機用にコードを生成する際に認識する計算の構造を意識しながら、通信性能のプロファイル結果を適用する方式を示す。計算の構造を意識することで、効果的なスピードアップの実現が可能となった。今回は、特に分散環境での方式の設計と実装、評価を行なった。

第2章では、我々が研究開発を行なっている汎用言語と自動並列化コンパイラ、そして、今回用いた並列計算環境について概説する。第3章では、分散計算環境において通信性能をプロファイルする方式について解説し、第4章で実装したシステムの性能評価を行なう。第5章で考察を行ない、最後に、第6章でまとめとする。

2 環境

2.1 並列言語 log *l*

並列言語 log *l* は、同期モデルに基づく並列言語である [1]。基本的な構造は、一般に PRAM アルゴリズムと呼ばれる構造化言語にならっている。PRAM アルゴリズムには特に固定した定義は存在しないが、本稿では [2] のものを採用している。

基本的には共有メモリを持つデータ並列言語として位置付けられる。HPF[3] や VPP Fortran[4] などに代表される並列 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。並列 Fortran が実際の並列計算機を仮定しているのに対し、log *l* では、プログラムの設計を実際の計算機構成から独立に行えるよう、PRAM[5, 6, 7] に基づく並列計算モデルを仮定している (図1)。

また、特に重要な構成要素として、以下のものが挙げられる。

par 構造 並列実行構造である par 構造は次のような文法を持ち、意味的には、「集合 X の各要素 p に対して、*OPERATION*(p) を実行する」。

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

メモリモデル log *l* では共有メモリついて、メモリモデルを複数用意しており、詳細なプログラミングを可能にしている。読み込みと書き込みそれぞれについて、同時アクセスが可能か否かを指定できる。また、同時書き込みについては、アクセス競合の対処法を複数用意している。

2.2 自動並列化コンパイラ log *c*

log *c* は log *l* に基づく自動並列化コンパイラであり、現在、pthread コード、PVM コードおよび逐次コードを生成することができる [8, 9]。

主な役割として、par 構造を解釈し、実際の並列計算環境に合致する並列コードに変換する。各 par 構造は、プログラムの実行順序性を損なわないようにマッピングされる [10]。

ただし、log *c* は汎用性確保を第一の目的としていたため、各並列計算環境の詳細な特徴を直接フィードバックしてこなかった。

本稿では、並列計算環境ごとに異なる通信性能を把握する方式を考案し、log *c* の附属プログラムとしての実装を行なった。そして、このプログラムの結果を logc に反映させることで、生成コードの実行速度向上を実現した。

なお、今回は、分散環境を想定した PVM コードを生成するバージョンでの方式について示す。

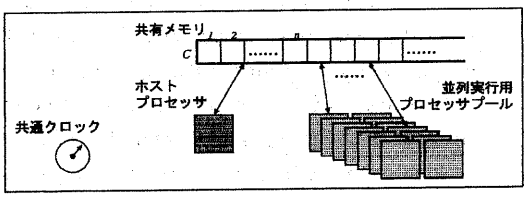

図 1: log *l* が仮定する並列計算モデル

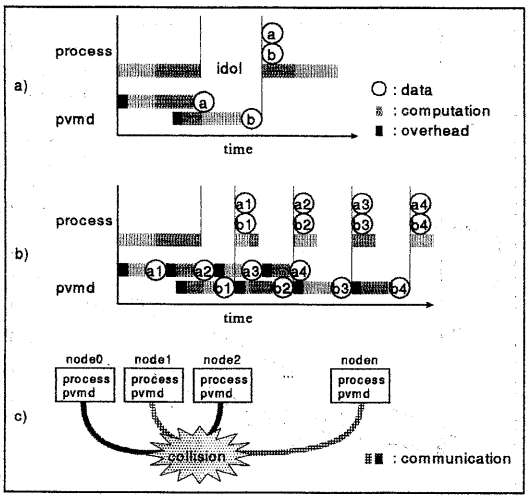

図 2: 通信における問題

2.3 クラスタ環境

今回は、イーサ接続された PC-AT 互換機 8 台を 100Base イーサ接続したクラスタ環境を扱う。各ノードのスペックは、表 1 の通りである。

CPU	Cyrix 6x86 120+GP
Memory	64M(32M 60ns EDO-RAM × 2)
Mainboard	PCI/I-P55T2P4(PBSRAM 512M)
Ether card	EtherExpressPro Model B
Ether hub	LFR12-A(100base shared)
OS	FreeBSD-2.2.2R

表 1: クラスタ環境のスペック

3 通信性能プロファイル方式

3.1 考慮する特性

今回の通信性能プロファイル方式では、主に次の 3 つの項目がボトルネックになる場合を考え、これらに対する改善を目的とする。

データ到着までの計算待ち状態 並列プロセスは一般にデータフローグラフとして表せるが、メッセージパッシングインターフェイスを用いた並列プログラムのような場合、データフローマシンで行なうように動的にプロセスを割り付けるようなことは少ない。log c の場合も同様で、むしろ動的なフェーズを極力減らすことで実行効率を確保する。するとどうしても、計算がフローするまでの待ち時間が生じる (図 2-a)。この通信を適切に分割し、待ち時間を減らす。どのような計算を行なうか不明な場合には、分割自体が不可能であるが、自動並列化コンパイラの場合は、計算についてのアルゴリズムを把握しているために、通信粒度の調整がが可能となる。

通信オーバーヘッド 通信の分割を行なうと、必然的に通信オーバーヘッドが生じ、並列計算の速度にも影響を及ぼす (図 2-b)。分割の最小値を定めるために、一回あたりの通信オーバーヘッドを計測し、これを適切な量にとどめる必要がある。すなわち、分割による副作用を抑制する。

コリジョン ネットワークの構成によっては、通信の交錯 (いわゆるコリジョン) が起こる可能性がある (図 2-c)。コリジョンに限らず、複数の通信が一度に起こった場合、通信性能が低下する可能性は少なくない。全ノードについて、通信のあらゆる状況を計測することで、通信性能が大きく低下するような通信パターンを明らかにする。コリジョンは待ち状態を生むが、改善策の違いから上記の待ち状態とは区別する。

3.2 通信部分への適用

log c は基本的に SPMD の PVM コードを生成する。PRAM アルゴリズムを SPMD のコードに変換する (コンパイルする) 際には、プロセッサへの作業の均等な分割と、それらの連動させるための手法が主な焦点となる。log c は、ループ内通信のパイプライン化を軸に、可能な限り通信回数を削減するポリシーを持つ。通信性能には関与しないので、前節の理由によって期待通りの実行効率を得られない場合がある。

考慮する特性のうち、計算待ち状態と通信オーバーヘッドの問題はトレードオフの関係にある。関数呼出が存在しなければ、ある部分の計算はほぼ一定の時間を消費すると考え、通信と計算の量のバランスから妥協点を見出す。この通信は 1 対 1 通信を基本としているので、ネットワークの構成よりも純粋に通信速度に依存する。

コリジョンについては、もしネットワークがコリジョンを生じないような構成であった場合には考慮する必要がないが、そうでない場合にはなんらかの対策を講じる必要がある。ここでは、1 対多以上の通信におけるなんらかの衝突すべてを扱い、遅延が生じる場合についてのみ考慮する。また、遅延が生じない 1 対多通

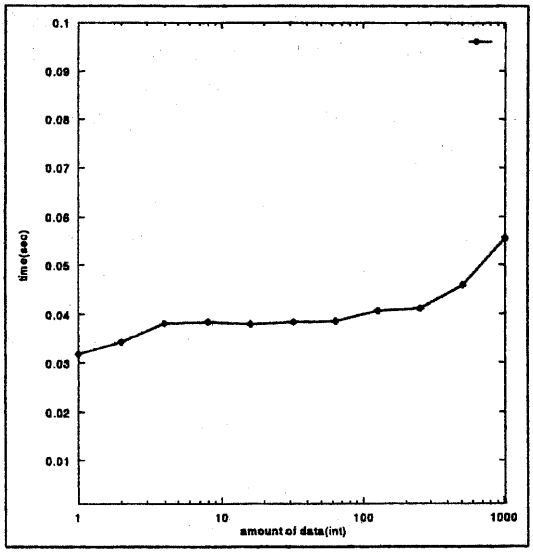

図 3: 通信性能

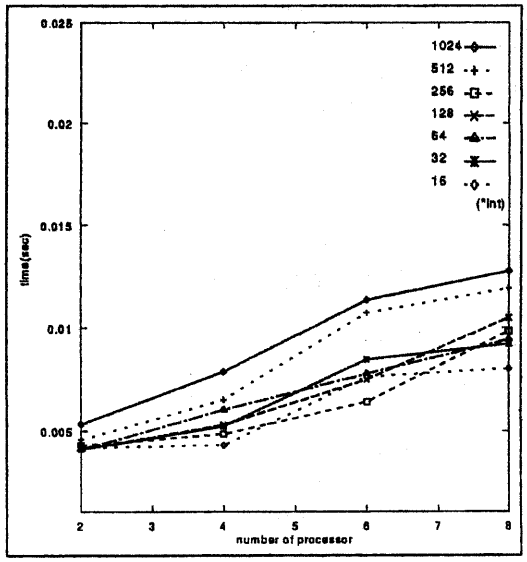

図 4: コリジョンの影響

信に関しては優先的に用いる。

4 評価

通信性能に関する評価は、通信オーバーヘッド (図 4) とコリジョン (図 4) に限られる。int の値を繰り返して通信し、繰り返し数で割った値を求めた。

通信オーバーヘッドは、1int についてのオーバーヘッドを除いた転送時間を 0 と仮定して算出した。1024int を転送しても、全体の転送時間からするとまだ通信オーバーヘッドの方が若干大きいことがわかる。

コリジョンについては、偶数台のノードそれぞれに 1 対 1 通信を行なわせることで計測した。コリジョン自体が生む遅延というよりも、スループットの上限がネットワークの構造を素直に反映する形で現れた。ほぼ全体の転送量だけの時間を消費したが、転送が終了するタイミングは全体の転送量に依存して遅れた。

また、変換例としてインテジャーソートについての計測を行なった (図 5)。入力となる数列のサイズは 2^{10} 元で、他の計測と同様、繰り返して計算した計測結果を繰り返し数で割った。

結果として、ソートすべき数列の分配を分割したことにより、PVM タスクの待ち時間が減少し、わずかながら速度向上が見られた。

5 考察

5.1 通信プロファイリング方式の考察

今回のような環境の場合、通信性能はイーサカードとハブの性能に強く依存する。通信まわりの環境はシェアードのハブであるため、コリジョンの回避が目立った効果を挙げた。通信性能を考慮したコードの改善は、本質的にはアルゴリズムを補助する役割としてしか機能しないので、劇的な性能改善に結び付くことは少ない。しかし、わずかながらも改善の余地が残されていることは事実であり、それをある程度捕捉できたと考える。

そのような、各機種に依存する特性を把握するために、我々は、ネットワーク構成を意識せずに、プロファイリングという方法をとった。これは、ネットワーク構成の他に内在する特性を捕捉できるということもあるが、コードの機種依存部分を生成する機能に関しても汎用性を持たせたいと考えたためである。ネットワーク構造は多岐に渡り、その構造自体をコンパイラが理解するよりも、プロファイリングの結果により判断させた方が、個別の対応も半ば機械的に行なうことができる。知らなければならない情報としては、プロセッサ数とメモリの上限のみであり、この点に関しても、満足のいく水準に達したものと考える。

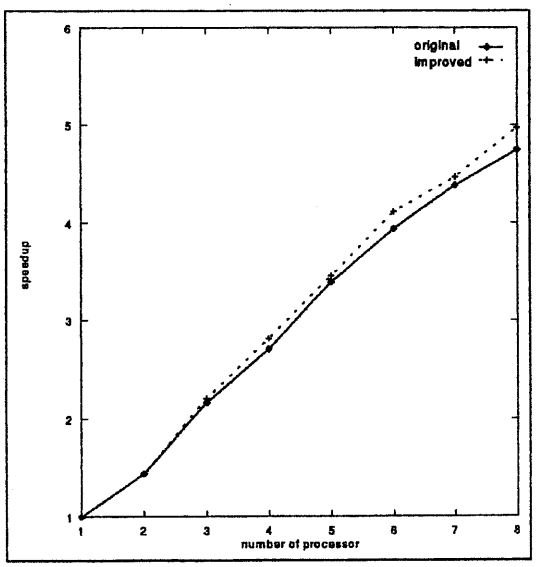

図 5: 変換例の結果

ただし、当然のことながら、計算の量自体が少ない場合には、このような配慮自体が意味をなさない。並列計算は大規模な計算が主体となってはいるが、そうでないものも実際には存在する。問題のサイズに関しての考慮は、今後の課題と言える。

5.2 扱わなかった特性

メッセージパッシングシステムの通信バッファ管理などの機構も、重要な通信性能として考えられる。例えば pvm3.3.11 の場合、通信バッファの管理は完全にシステムに委ねられている。そのため、激しく通信を繰り返すと、バッファ管理の負荷は必然的に高くなり、同じ通信でも状況によって消費する時間が異なる。しかし、通信バッファ管理をユーザに任せることは、プログラムを繁雑にし、機種依存性にも負の影響を及ぼす。ただし、バッファ管理機構の改善はメッセージパッシングシステム自体においてなすべき作業であることも事実であり、自動並列化コンパイラの範疇ではない。

また、並列計算環境に他のジョブが動作していた場合も想定しない。これは、分散 OS のような領域では問題とされ、プロセス移送などなんらかの方法でシステムの負荷を全体に散らす必要がある。しかし、ハイパフォーマンスコンピューティングという観点からは、高い負荷を持つジョブがバックグラウンドで動作している状況自体がナンセンスであり、これも自動並列化コンパイラとは別の機構で排除されているものと仮定する。

6 あとがき

本稿では、自動並列化コンパイラ log c における通信性能のプロファイリング方式についての設計と実装を行なった。また、サンプルプログラムによる評価を行なった結果、その効果が明らかになった。

自動並列化コンパイラの機種依存の対応は不可欠であり、機種依存の特性を把握し、コードに反映する方法としてプロファイルを用いることは、ネットワーク構造の直接理解よりも汎用性の点において優れている。これは、メッセージパッシングというインターフェイスを介しているために可能となっている。

今後の課題としては、pthread コードに対するプロファイリング方式の考案が挙げられる。また、PVM コードにおいても、グループ通信の性能評価などの課題が残されている。また、タスクスケジューリングとの連係がなされておらず、これについても考慮しなければならない。

参考文献

[1] 金山二郎, 飯塚肇：同期モデルに基づく自動並列化コンパイラ, 情報処理学会第 7 回プログラミング研究会 (1997).

[2] 宮野悟：並列アルゴリズム, 近代科学社 (1993).

[3] 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).

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

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

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

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

[8] 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).

[9] 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).

[10] 金山二郎, 飯塚肇：同期モデルに基づく自動並列化コンパイラにおけるタスクスケジューリング方式, 情報処理学会第14回プログラミング研究会 (1997).