2P－5

データ並列言語 Dataparallel C のマルチスレッドへの実装

黒沢 崇宏，相田 仁，齊藤 忠夫
東京大学 工学部

1 はじめに

複数のプロセッサを結合した並列計算機が開発されている．これに伴い，並列言語，及びその言語に対するコンパイラが開発されてきている．Dataparallel C は，このような並列言語のうちの一つである．

2 データ並列言語 Dataparallel C の概要

2.1 言語仕様

Dataparallel C では，逐次実行のための 1 つのフロントエンドプロセッサと並列実行のための多数のバックエンドプロセッサをプログラミングするモデルを採用している．

Dataparallel C で新たに導入された構文として，ドメイン宣言がある．ドメイン宣言は，C の構造体宣言と同じような構文で，仮想プロセッサを確保するのに用いる．

Dataparallel C では，ドメイン選択文という文が導入されている．Dataparallel C のプログラムは，ドメイン選択文の部分を並列実行することになる．

2.2 マルチプロセッサ Dataparallel C コンパイラ

ここでは，プロセッサ間で共有されるメモリが存在する並列計算機をマルチプロセッサ型並列計算機と定義する．これに対して，共有メモリを持たない並列計算機をマルチコンピュータと定義する．

Dataparallel C は，数種のマルチコンピュータと，マルチプロセッサである Sequent Symmetry に既に実装されている．

Symmetry 上に実装されている C 言語には，並列プロセス上での共有データ構造を実現する手段が提供されている．これ以外にも，並列処理用の関数群が用意されている．

次に，マルチプロセッサ Dataparallel C コンパイラの概要を説明する．マルチプロセッサ Dataparallel C コンパイラは，コード生成の際にドメイン宣言を構造体宣言に書き換える．ドメインのインスタンスは共有データとして宣言される．また，ドメイン選択文は for 文により仮想プロセッサをエミュレートするループに書き換えられる．

3 Dataparallel C のマルチスレッドへの実装

マルチスレッド環境では，全メモリ空間を共有したスレッドをプロセス内に複数つくることができる．本研究では，マルチスレッド環境として，SunOS5.3 上のマルチスレッド環境を用いることにした．

マルチスレッド環境を提供するマルチスレッドライブラリには，関数をエントリポイントとしたスレッドの生成やスレッド ID の取得，条件変数によるスレッドの同期や共有資源へのアクセスに用いる排他制御ロックなどの関数が含まれる．

3.1 マルチプロセッサ型並列計算機とマルチスレッドとの比較

マルチプロセッサ型並列計算機の環境とマルチスレッド環境を比較し，その共通点と相異点を述べる．

大きな共通点は，命令実行の最小単位の間で共有されるメモリが存在するという点である．この共通点から，Dataparallel C をマルチスレッド上に実装する場合，マルチプロセッサ Dataparallel C コンパイラを変更することで，マルチスレッド Dataparallel C コンパイラを実装することができる．

相異点は，バリア同期が実装されているかどうか，という点である．マルチプロセッサ上にはバリア同期の関数が用意されているが，マルチスレッド上にはそういった関数は用意されていない．しかし，これはマルチスレッド上でも条件変数などを用いることで実現可能である．

3.2 実装方式の提案

ドメイン選択文をどのようにマルチスレッドライブラリを用いたコードに変換するかということが問題である．これに対しては，最終フェーズであるコードを生成のフェーズで，マルチプロセッサ Dataparallel C コンパイラでは並列実行モードとなっている文のまとまりを抽出し，関数として出力するという方法を取った．これの方法より，拡張性を失なうことなくマルチスレッド上に実装するこ

"The Implementation of the Data-parallel Programming Language Dataparallel C on Multi Thread"
Takahiro Kurosawa, Hitoshi Aida and Tadao Saito
Faculty of Engineering, The University of Tokyo

とができる.

4 評価および考察

4.1 評価

Dataparallel C で書かれたベンチマークプログラムをマルチスレッドDataparallel C コンパイラによりコンパイル・実行してみて, プログラム実行時間がスレッド数に応じてどのように変化するかを調べた. ベンチマークプログラムとしては, 数値積分による π 計算, 互いに素な数の組の数, 行列の積, 推移的閉包, ガウス消去法, ガウス・ジョルダン法, 素数の数, 三角パズルの 8 問題を用いた.

ベンチマークの基準としては, 並列実行に用いるスレッドの数によりspeedup という量がどのように変化するかを調べた. speedup とは, 単一プロセッサで最も効率のいい逐次アルゴリズムを実行するのに要した時間と, 並列実行に要した時間の比である. なお, ハードウェアは, 4CPU の SPARCserver20 を用いた.

以上のベンチマークによる speedup の結果を図 1に示す. スレッド数を多くするに従い speedup が向上するものとスレッド数を多くしても speedup が向上しないか, または逆に悪化するものの 2 通りがあることがわかる.

なお, 図 1で, スレッド数が 4 を超えると speedup が減少に転じるのは, ターゲットとなる計算機に CPU が 4 個しか搭載されておらず, スレッドを 4 本より多くすると同時に実行できるスレッド数を上回るため, スレッド実行の切り換えが必要となるためである.

4.2 考察

speedup の悪化は, スレッド生成, バリア同期, 排他制御ロックなどのオーバヘッドが直接的な原因でないと考えられる. これらのオーバヘッドは小さいので, 規模の大きい問題では無視できるほどの量だからである.

speedup の悪化の原因を調べるために, 問題の規模を変えてみることでベンチマークを行った. ベンチマークの対象は, ガウス・ジョルダン法による線型方程式の解を求める問題とした. 結果を, 図 2に示す.

図 2よりわかることは, 問題の規模を大きくするにつれ, 徐々に speedup がスレッド数の増加関数となっていくことである. この原因を考えてみる.

ガウス・ジョルダン法による方程式の解を求める Dataparallel C のプログラムは, 問題の規模が大きいときは, 実プロセッサによる仮想プロセッサのエミュレーションループの回数が多くなり, CPU の命令キャッシュの効果が期待でき, また, エミュレーションループの実行に時間がかかる分だけバリア同期にかかる時間が相対的に小さくなる (粒度が粗くなる). よって, speedup が向上すると考えられる.

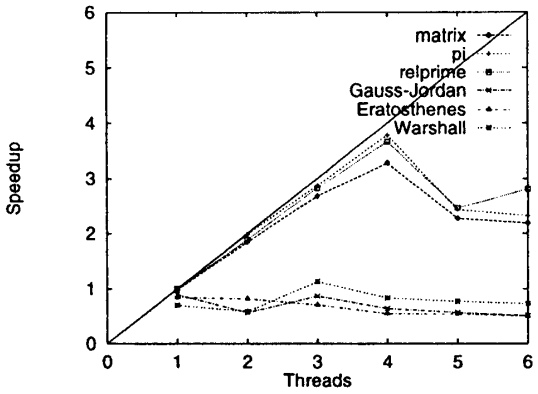

図 1: ベンチマークの結果

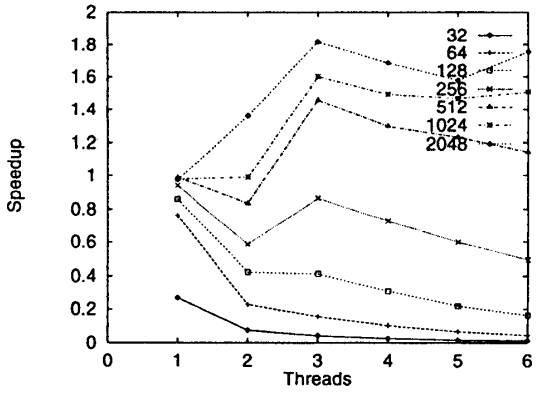

図 2: 問題の規模を変化させた場合の speedup の変化

5 結論

本研究では, マルチスレッドをターゲットとしたCのコードを生成する Dataparallel C コンパイラを作成した. 結果として, このコンパイラの生成するコードの実行効率は必ずしもよいとは言えないことがわかった. 実行効率の悪化は CPU の命令キャッシュを有効に利用できないことが原因であることが考えられたり, 問題の規模が小さい場合にはバリア同期のオーバヘッドが無視できなかったりすることが原因であると考えられる.

参考文献

[1] "Data-Parallel Programming on MIMD Computers", Philip J. Hatcher, Michael J. Quinn. The MIT Press, 1991

[2] "Programming Models for Parallel Systems," Shirley A.Williams. Wiley, 1990