4E−5

リフレクションを導入した並列論理型言語RKL1における動的負荷分散の記述と評価*

阿隅 太志 大澤 直樹 山神 弘毅 武田 正之†
東京理科大学大学院 理工学研究科 情報科学専攻‡

1 はじめに

リフレクション[1]を導入した並列論理型言語 RKL1 [?]の有効性の検証するため，動的負荷分散メタモジュールを実装し評価する。ベースレベルのプログラムにメタモジュールを指定しRKL1コンパイラでコンパイルすることにより、簡単にメタモジュールを利用することが可能となる。

2 並列論理型言語 RKL1

並列論理型言語 RKL1 は、第五世代コンピュータプロジェクトで開発された並列論理型言語 KL1 にリフレクションを導入したプログラミング言語である。

リフレクションを導入することでベースレベルとメタレベルのコードを分割し，メタレベルのコードの再利用性を向上させている．リフレクションの導入はメタインタプリタを拡張する方式が一般的であるが，メタインタプリタによる解釈実行は効率の低下を招くために部分計算による改善が必要である．そこで，RKL1 ではメタインタプリタによる解釈実行を KL1 のボディ部に制限したため，あらかじめ部分計算によってメタレベルのコードをベースレベルのコードに反映したプログラムへコンパイルすることが可能である．

RKL1 コンパイラは，ベースレベルのコードとメタレベルのコードを独立したファイルから読み込み，メタレベルのコードをベースレベルに反映した KL1 のコードを出力する．

RKL1 ではリフレクション機構としてメタボディとメタプロセスグループという２つの概念を導入している。

メタボディはボディ部の実行の制御やカスタマイズを行うメタレベルの記述であり、ベースレベルのボディ部の変数の参照／具体化／置換、ゴールの追加／削除等が可能である。

メタボディの実行をメタプロセスと言う。メタプロセスグループはメタプロセス間におけるメタ情報の共有のために導入された概念であり、これに属するメタプロセスはグループコントローラによって統括され、メタ情報の通信を可能にする。

これらのリフレクションの指定は注釈中で行うため、リフレクションを導入していない普通の KL1 言語の処理系でベースレベルのみを実行する事も可能である。

3 メタモジュールの記述

動的負荷分散メタモジュールとして、ランダム、循環、要求駆動負荷分散メタモジュールを記述した。

ランダム負荷分散はサブタスク[1]を分散するノードを乱数により決定する負荷分散方式である。

循環負荷分散は処理を各ノードに順番に割り当てていく負荷分散方式である。各ノードに均等にタスクが分散されるが、ランダム分散同様サブタスクの粒度が不均質である場合は、処理効率が低下する。

要求駆動負荷分散は各ノードにサブタスクを割り当てているノード(サーバ)が、実行すべきサブタスクがなくなった(暇になった)ノード(クライアント)の要求により、サブタスクを順次割り当てていく負荷分散方式である。すべてのクライアントが空き時間を作ることなく計算するため、安定して台数効率を得ることができるが、クライアントの数が多いとサーバがボトルネックになる。

これらのメタモジュールの指定をベースレベルのコード内に記述することにより、簡単に負荷分散戦略を利用することができる。

4 リフレクティブ タワー

RKL1 コンパイラでは、メタレベルのメタレベルの記述が可能である。これは再帰的に記述する事が出来、リフレクティブ タワーを記述することが可能である。

*Description of dynamic load balancing on RKL1 and its Evaluation

†Taishi Asumi , Naoki Osawa , Kouki Yamagami , Masayuki Takeda
‡Department of Information Sciences , Science University of Tokyo

[1]計算対象となるベースレベルのプロセス全体をタスク、タスクを分散するために分割した計算をサブタスクと呼ぶ

今回、動的負荷分散メタモジュール上でその分散状況をリアルタイムに表示するモニタをメタメタモジュールとして記述した。このメタメタモジュールは先に定義した動的負荷分散メタモジュール内で指定することが出来る。これにより、動的負荷分散メタモジュールを変更する事なく、各々の分散状況をモニタする事が可能になる。

5　評価

ベースレベルとして N-Queens 問題 を記述し、ランダム負荷分散、循環負荷分散、要求駆動負荷分散 メタモジュールをそれぞれ指定し、RKL1 コンパイラでコンパイルし、それぞれを分散メモリ型環境、共有メモリ型環境で実行することにより、表 1に示す結果が得られた。ここでは並列環境として pvm[2] を利用した。

分散メモリ型並列環境[10]			sec(台数効果)
シングルノード		100.8(1.00)	
ノード数	random	cyclic	demand
1	153.2 (0.65)	142.3 (0.70)	159.4 (0.63)
2	98.4 (1.01)	95.6 (1.04)	84.1 (1.18)
3	65.8 (1.50)	61.0 (1.62)	70.7 (1.40)
4	49.5 (2.00)	57.6 (1.72)	50.4 (1.96)
5	44.3 (2.24)	41.6 (2.38)	42.9 (2.31)
6	41.1 (2.41)	40.0 (2.47)	35.8 (2.77)
7	38.1 (2.60)	33.2 (2.98)	31.0 (3.19)
共有メモリ型並列環境[11]			sec(台数効果)
シングルノード		50.52(1.00)	
ノード数	random	cyclic	demand
1	53.48 (0.94)	54.60 (0.93)	59.19 (0.85)
2	31.03 (1.63)	40.94 (1.23)	28.27 (1.79)
3	23.57 (2.14)	26.20 (1.93)	19.07 (2.65)
4	17.99 (2.90)	17.45 (2.90)	14.72 (3.43)

表 1: 12-Queens 実行結果

また、表 1において台数効果を計算しグラフ化したものを図 1に示す。台数効果とは、並列化してないプログラムの実行時間を 1.00 としたときの実行効率である。理想値としては、4 台で実行した場合は 4.00 倍の効果になる。図 1を見ると、要求駆動、循環、ランダム負荷分散の順で実行効率が得られており、台数効果がほぼ線形に得られている事がわかる。特に共有メモリ型環境での要求駆動負荷分散においてはほぼ理想的な台数効果を得る事が出来た。

[2]PVM version 3.3 : Parallel Virtual Machine system
[10]SPARCstation4 5 台,SPARCstation5 2 台
[11]SPARCstation20(4CPU)

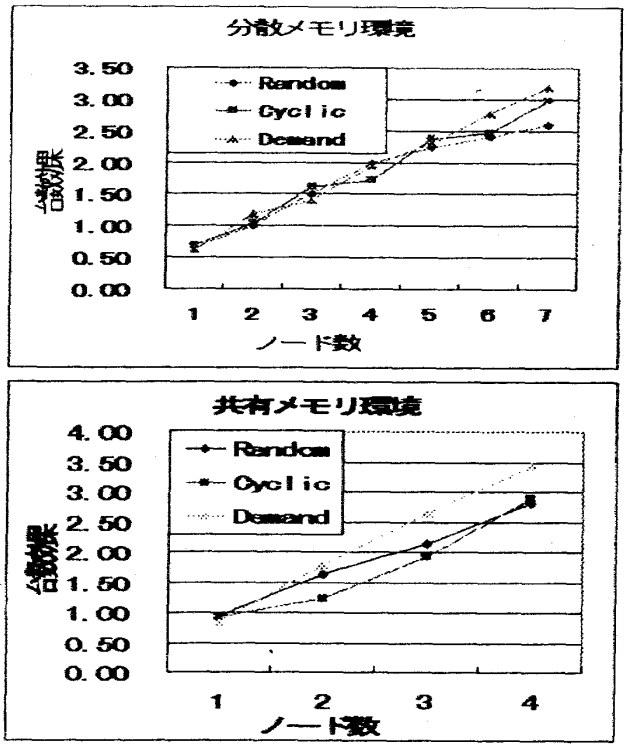

図 1: 台数効果

6　おわりに

今回、RKL1 の有効性を検証するために、いくつかのメタモジュールを記述した。RKL1 では、ベースレベルとメタレベルのプログラムを独立したモジュールとして記述でき、ベースレベル内でのメタモジュール指定の追加／変更によりベースレベルのロジックを変更せずにその実行制御のカスタマイズを行う事が出来る。また、メタモジュールによって変換されたプログラムは効率を落とさずに実行される事を確かめる事ができた。

参考文献

[1] Maes,P. *Issues In Computational Reflection*, In Meta-Level Architectures and Reflection, pp.21-35, North-holland, 1988.

[2] Takahashi,T.and Takeda,M.: *An Efficient Implementation of Reflection in KL1*,FGCS'94 Workshop on Parallel Logic Programming,Institute for New Generation Computer Technology,pp.17-26 Dec.1994.