ハイパフォーマンス
コンピューティング 56－1
（１９９５．６．１）

FL 階層化並列簡約システムの性能評価

北島 宏之　沈 紅　片平 昌幸　小林 広明　中村 維男

東北大学大学院情報科学研究科

関数型言語は手続き型言語と違い、参照透明性やプログラムの高い生産性などの多くの有用な特徴を持つ。しかし、従来の計算機上では十分な処理速度が得られないために、その使用が制限されている。我々は、関数型言語FLで記述されたプログラムを高速に実行するために、マルチプロセッサ処理とパイプライン処理を統合した、階層化並列簡約システムを提案してきた。本論文では、システムの持つ性能を十分に引き出すために参照の局所性を考慮した動的タスク割り当て手法について考察し、シミュレーション実験を通じてシステムの性能評価を行う。その結果、提案するシステムおよびタスク割り当て手法の有効性が明らかになる。

Performance Studies of the FL Hierarchical Parallel Reduction System

Hiroyuki Kitajima　Hong Shen　Masayuki Katahira
Hiroaki Kobayashi　Tadao Nakamura

Graduate School of Information Sciences, Tohoku University
Aramaki Aza Aoba, Aoba-ku, Sendai 980-77, Japan

Functional programming languages differ from traditional imperative ones with many appealing properties such as referencial transparency and high programming productivity. However, the inefficiency of their implementation on conventional computers has prevent them from wide acceptance. We have proposed a hierarchical parallel reduction system by combining multiprocessor processing and pipeline processing in our earlier work. In this paper, we investigate the task scheduling strategy with locality consideration suitable for enhancing the system performance, and carry out software simulation experiments. The simulation results reveal the effectiveness of the proposed system with the scheduling strategy.

1 まえがき

マルチプロセッサシステムでは、プログラムから抽出される並列処理可能な部分を複数の処理要素で処理することにより、プログラムの高速処理が図られる。マルチプロセッサシステムは、並列性の抽出が容易なプログラムの高速実行にとって非常に有効であると考えられる。一方、関数型言語は、参照透明性、検証容易性、プログラムの高い生産性および並列実行の容易性など多くの有用な特徴を持つ。これらの利点から、マルチプロセッサシステム上における関数型言語の高速処理が期待できる。我々は、関数型言語FLで記述されたプログラムを高速に実行するため、共有メモリ型マルチプロセッサシステムである階層化並列簡約システム[1][2]を提案してきた。本論文では、このシステムに注目する。

関数型言語の遅延評価を実現するために、一般にグラフ簡約[3]が利用されている。グラフ簡約は、並列処理や遅延評価などに有効な手法であるが、一般に処理粒度が小さくなり、タスク生成のためのオーバヘッドが大きくなる欠点を持つ。このため、グラフ簡約をマルチプロセッサシステムへ実装した場合には、細粒度によるメモリアクセスなどの簡約実行時のオーバヘッドが大きくなり、理想的な処理速度の達成が困難となる。従って、システムの性能を十分引き出すためには、効果的なタスク割り当て手法が必要不可欠である。本論文では、FL 階層化並列簡約システムのための、参照の局所性を考慮した動的なタスク割り当て手法について検討する。

はじめに、2節において、本論文で基礎とするFL階層化並列簡約モデルについて簡単に説明を行い、そのシステムについて述べる。次に、3節において、参照の局所性を考慮した動的なタスク割り当て手法について考察する。4節では、システムの性能評価とその結果を示す。シミュレーションによるシステムの性能評価を通じて、提案されるシステムおよびタスク割り当て手法の有効性を示す。5節は結言である。

2 FL階層化並列簡約システム

2.1 階層化並列簡約モデル

関数型言語の実行を実現する手法として、並列グラフ簡約がよく知られている[3]。概念的に、並列グラフ簡約は、簡約グラフの持つ複数の可簡約項に対する並列な書き換え操作（簡約）からなる。これらの書き換え操作は、最終結果としての正規形が得られるまで繰り返される。

このような簡約処理を計算機上で実現する場合、処理を三つの部分処理に分解することができる。まず、簡約処理を行うために、簡約グラフを探索し可簡約項を検出する。次に、検出した可簡約項を処理要素へ割り当てる。処理要素では、割り当てられた可簡約項の書き換えを行い、その結果を簡約グラフへ反映させる。これらの処理を可簡約項がなくなるまで繰り返す。以上の可簡約項の検出、配置、実行の三つのステージを時間的に重複させることにより、より高い水準でのパイプライン処理が実現できる。また、パイプラインの実行ステージを複数のプロセッサで実現することにより、並列処理可能な可簡約項を同時に処理することが可能となり、並列簡約を達成することができる。この並列処理およびパイプライン処理を統合したモデルが、階層化並列簡約モデルである[1][2]。階層化並列簡約モデルにおける簡約の正当性および停止性は、$F_{\alpha'}$[4]と呼ばれる並列簡約戦略において保証されている。

例として、図1に二つの行列の乗算を行うFLプログラムを、図2に行列乗算プログラムを階層化並列簡約モデルへマッピングした場合の時間空間図を示す。基本的に、FLプログラムは一つの関数であり、この関数は通常複数の関数から構成される。FLプログラムでは、関数を値に作用させて、新しい値を得ることによって演算を進める。図1において、*def* 文節により定義された mmul が行列乗算を行うユーザ定義の関数であり、IP は二つのベクトルの内積を求める関数である。ここで、この関数（プログラム）が作用する値としてのデータオブジェクトは、<A,B>の書式を持ち、AおよびBはそれぞれi×jおよびj×k（i、jおよびkは整数）の行列でなければならない。図2におけるデータオブジェクトサイズは、i=3、j=1およびk=2である。図2に示した三つの軸はそれぞれ時間、パ

```
def  mmul ≡ α : (α: IP ) ∘ ( α: distl ) ∘ distr ∘ [ s1、trans ∘ s2 ]   where
[
    def  IP ≡ + ∘ ( α : * ) ∘ trans
].
```

図 1: 行列乗算 FL プログラム

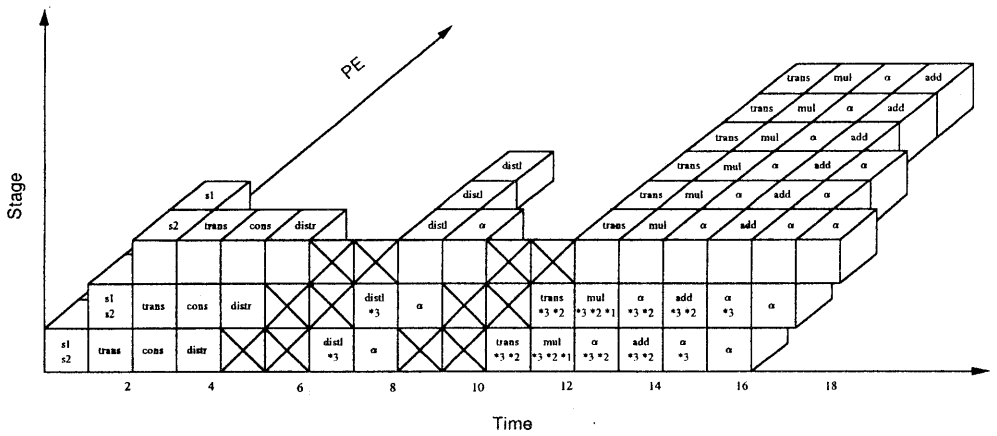

図 2: 行列乗算プログラムの時間空間図

イプラインステージ、処理要素PEである。ある時刻に処理される可簡約項の基本関数名も併せて図に示す。ここで×は、データ依存が原因で起こるパイプラインストールによる、システムの非稼働状態を表わす。

2.2 FL階層化並列簡約システム

共有メモリ型マルチプロセッサアーキテクチャは、並列グラフ簡約に必要とされる大域的なアドレス空間を容易に提供でき、前述のような高度な並列処理モデルの実現が可能であると考えられる。この観点から我々は、FLのための階層化並列簡約モデルを実現するために、クラスタ化した共有メモリ型マルチプロセッサシステムを提案してきた[1][2]。

図3に、FL階層化並列簡約システムの基本ブロック図を示す。一般に簡約のために必要とされるデータオブジェクトの大きさは様々であり、また可簡約項の検出および配置は高速に行わなければならないために、共有メモリはその記憶内容によりグラフメモリとデータメモリに分割される。ここで、グラフメモリは簡約グラフに用いられ、コンパイラにより簡約グラフに変換されたFLプログラムが格納される。データメモリはデータオブジェクトの格納に用いる。簡約グラフより検出された可簡約項は、スケジューラにより簡約実行可能なタスクとしてタスクプールを介し、処理要素であるPEに配置される。各PEは、割り当てられたタスクに対し、簡約に必要とされるデータオブジェクトを読み込み、簡約を実行する。本システムでは、可簡約項の検出、配置および簡約実行は時間的に重複され、パイプライン化されている。

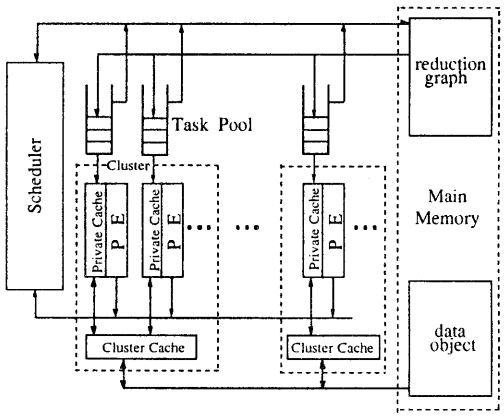

図 3: FL階層化並列簡約システムの基本ブロック図

グラフメモリに格納されている簡約グラフは、タスクプールを介し全てのPEからアクセスすることができる。また、各PEに設けられたプライベートキャッシュにより、共有メモリに対する大域的なメモリアクセス頻度を著しく低減させることが可能であり、データオブジェクトの局所的アクセスを高速に行うことができる。さらに階層化並列簡約システムでは、メモリアクセス頻度をより低減するために複数のPEをクラスタ化し、各クラスタにクラスタキャッシュが設けられている。

FL階層化並列簡約システムでは、階層化並列簡約モデルを実現するために、可簡約項（タスク）の検出および配置に対して以下の規則を加える。

(1) 簡約グラフにより、逐次実行を必要とする可簡約項は同時に検出されない。
(2) 配置ステージにおいて、検出されたタスクがタスクプール内の先のタスクと 1 対 1 の逐次実行を必要とする場合、必ずタスクプール内の同じタスクキューに送られる。
(3) 簡約ステージにおいて、同じデータオブジェクトを使用するタスクの並列処理を可能にするため、そのデータをロックせず、各 PE に対してデータのコピーを持たせる。
(4) ガーベージコレクションは、検出されるタスクおよびデータの独立性により、複雑な機構を必要とせず独立に行うことが可能である。

ここで、規則 3 により、locking / resuming 機構を利用する他の方法と比較して、システムの性能向上が可能となる。さらに、各タスクの簡約に必要とされるデータオブジェクトはタスクと 1 対 1 に対応し、データの一貫性が容易に保たれる。また、規則 3 により、規則 4 におけるガーベージコレクションの独立な実行が保証される。

上記の規則によって、結果的にデータに依存関係のない可簡約項は並列に検出され、別々の PE への配置が可能となる。逆に、データ依存関係のある可簡約項は、PE 内で逐次実行されるように配置される。例えば、"fog:x" という FL プログラムの場合には、はじめに "g:x" の簡約が行われ、その結果に "f" が作用する。一方、"[f,g,h]:x" では、"f:x"、"g:x" および "h:x" の 3 つの可簡約項に分割でき、相互にデータの依存性がないタスクとして並列に簡約が可能である。逐次実行が必要である可簡約項の場合には、階層化並列簡約モデルにおける高水準のパイプラインが利用可能となり、簡約が高速化される。一方、並列実行が可能である可簡約項の場合には、複数の PE で並列に簡約が行われることによって簡約が高速化される。

このような可簡約項（タスク）操作が可能であるのは、可簡約項のデータ依存性が簡約グラフにより一意に決定されることによる。このタスク割り当てのハードウェア上での実現には、検出されたタスクの持つデータオブジェクトへのポインタの比較が必要である。この操作は、可簡約項検出のための余分なコストとなるが、上記規則を適用しない場合のキャッシュフラッシングのオーバーヘッドと比較して十分小さいと考えられる。

3 動的タスク割り当て

3.1 幅優先/深さ優先タスク割り当て手法

有効なタスク割り当てとは、システム全体の負荷バランスを保ちつつ、システム資源の高い利用率を維持するよう、タスクを PE へ配置することである。

システム資源を有効利用するための簡約の高並列化は、簡約グラフを幅優先に走査し、簡約グラフが持つタスクを可能な限り多く検出することによって達成される。幅優先法を抽象化した簡約グラフに適用した場合の、タスクの各 PE への配置を図4に示す。ここで、各ノードがタスクを表わし、エッジはタスク同士の依存関係を表わす。また、タスクの検出優先度は、グラフにおいて右側が高いものとする。PE 数は 8 である。各グラフノードに示した記号 a/b において、a はタスクの配置された PE 番号を表わし、b はそのノードが検出されるパイプラインサイクルを表わす。図4より、幅優先法は可能な限り多くのタスクを抽出し、PE の利用率の向上に有効であると考えられる。しかし実際には、タスク生成のオーバヘッドやデータメモリアクセスによる遅延の増加を伴い、結果としてシステム性能が著しく低下する。これらの問題を解決する方法として、タスクの過度な生成を防ぐ方法、あるいはデータ参照の局所性を考慮しキャッシュのヒット率を高める方法などが考えられる。

タスク生成のオーバヘッドを最小に抑えるためには、簡約グラフを深さ優先に走査しタスクの検出に制限を与える方法が適すると考えられる。図5は、簡約グラフに深さ優先法を適用した結果である。PE 数および各グラフノードに示される記号の意味は図 4 と同様である。図 5 より、深さ優先法では並列性を抑制するようにタスクが生成されることが分かる。従って、深さ優先法を有効に利用すれば、システムの飽和が生じた場合などにおけるタスク数の制御が可能となる。しかし、この方法では、タスク間のデータ依存性の影響が最も大きくなり、主にタスク検出待ちによる処理要素の非稼働状態が増加する。結果として、システム資源の有効な利用が困難となる。

本研究では、システム資源の高い利用率を維持しつつ、タスク生成の動的な制御を行う幅優先/深さ優先法を提案する。幅優先/深さ優先法では、基

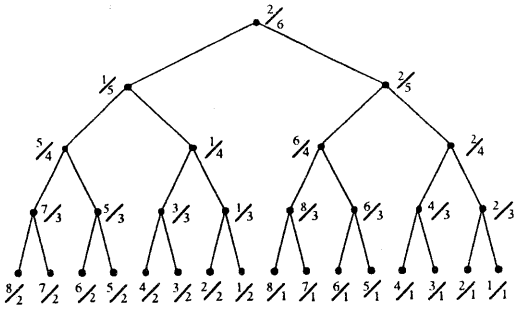

図 4:　幅優先タスク割り当て手法

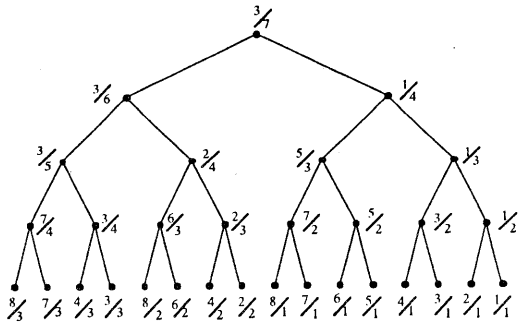

図 5:　深さ優先タスク割り当て手法

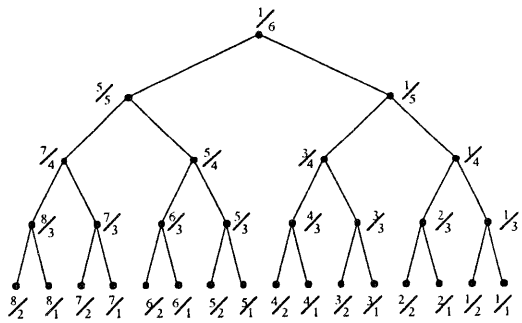

図 6:　幅優先 / 深さ優先タスク割り当て手法

本的には幅優先法を用い、タスク生成を制御するために深さ優先法との動的な切り換えを行う。ここで、幅優先法と深さ優先法の切り替えは、タスク検出時のシステム負荷およびシステムマネージャにより決定される閾値によって決定される。はじめは簡約グラフを幅優先に走査して行き、タスク検出数が閾値を超える場合に深さ優先法に切り換える。図6に、幅優先/深さ優先法を簡約グラフに適用した結果を示す。PE 数は 8 であり、各グラフノードの記号は図4と同様である。ここで、閾値は PE 数の 8 であり、システム負荷は PE の稼働率（ここでは常に8である）を用いている。図6では、簡約グラフを幅優先で走査して行き、検出されるタスク数が閾値8を超える時点で深さ優先法に切り換えが行われている。

幅優先 / 深さ優先法によって、幅優先法の利点である PE の利用率の向上と、深さ優先法の利点であるタスク数の制御が可能となる。しかし実際には、データ参照の局所性を考慮し、メモリアクセス頻度を減少させない限り、システムの有する潜在的な処理能力を引き出すことは不可能である。これまでの関数型アーキテクチャのためのタスク割り当て手法に関する研究[5-8]では、負荷バランスとシステムの有効利用について述べられてはいるが、データ参照の局所性を有効に利用しているとは言い難い。

3.2　参照の局所性を考慮したタスク割り当て

タスクの簡約に必要とされるデータは、タスクそれぞれに対して唯一である。従って、データ参照の局所性は、簡約グラフより一意に決定されるために、簡約グラフ上におけるタスク生成あるいは検出の局所性に置き換えることができる。キャッシュヒット率を向上させるためには、逐次実行の必要なタスク、つまり簡約グラフ上において 1 対 1 の親子の関係にあるタスクを、可能な限り同じPEに配置する手法が有効である。この手法を、FL 階層化並列簡約システムにおける第一のタスク割り当て戦略として導入する。この戦略は、2.2節で述べた階層化並列簡約モデル実現に必要な規則2に等しい。さらにFL 階層化並列簡約システムでは、PE の二次キャッシュとしてクラスタキャッシュが設けられている。クラスタキャッシュのヒット率を向上させるために、第二の戦略として兄弟関係にあるタスクを可能な限り同じクラスタに割り当てる。第一のタスク割り当て戦略がプライベートキャッシュに対する参照局所性の利用であり、第二の戦略がクラスタキャッシュに対する参照局所性の利用である。ここで、検出されたタスクが簡約に必要とするデータオブジェクトへのポインタを比較することにより、参照局所性を検出する。

スケジューラは、タスクの検出および配置に関して、可簡約項の並列簡約可能性を第一の要因と

して考慮する。参照の局所性は、第二の要因として考慮される。この優先順位は、並列簡約による高速実行の実現と、システム資源の有効利用のために与えられる。従って、幅優先 / 深さ優先法では、基本的に幅優先法が用いられ、その補足として深さ優先法が用いられる。

FL階層化並列簡約システムにおいて、データ参照の局所性を考慮することによりメモリアクセス頻度の減少が期待でき、さらに幅優先/深さ優先法を用いることによって、システムの有する潜在的な処理能力を引き出すことが可能であると考えられる。

4 シミュレーション実験および考察

4.1 シミュレーションモデル

図3に示すシステムについてシミュレータを構築し、性能評価を行う。コンパイルされたFLプログラムは、構造化されたメモリに格納される。検出された可簡約項は、タスクとして一時的にタスクプールに配置される。タスクの検出および配置は、タスクプールおよびPEの状態を監視しているスケジューラによって行われる。PEは簡約の実行を行う。

以下に、シミュレーション条件を示す。

- 性能評価の時間単位として、簡約パイプラインにおけるパイプラインサイクルを用いる。
- PE総数は16とし、4PEごとに1クラスタ化する（総クラスタ数は4）。
- 1パイプラインサイクルあたりに可能なタスクの検出数、およびグラフメモリからタスクプールへのタスクの転送数を最大16とする。
- PEでの簡約実行時間は、タスクである可簡約項の種類および使用されるデータオブジェクトの大きさによって、1ないし数パイプラインサイクルを必要とし、これらは実行時に決定される。
- キャッシュは一回の簡約に必要とされるデータオブジェクトを格納するために十分な大きさを持つものとする。データメモリのデータブロックは、キャッシュにダイレクトマップ方式で配置される。また、キャッシュアクセスおよびメモリアクセスに対するペナルティ時間を表1に示す。クラスタキャッシュ、およびデータメモリへのアクセスに衝突が生じた場合には、ラウンドロビン法によってアクセス権を与える。

以上の条件によって、FL階層化並列簡約システムのシミュレーション実験を行う。

表1: キャッシュおよびメモリアクセスペナルティ

	Private cache hit	Private cache miss	
		Cluster cache hit	Cluster cache miss (Data memory access)
Penalty	1	3	7

4.2 ベンチマークプログラム

シミュレーション実験に用いるベンチマークプログラムを表2に示す。AからEに示されるN-クイーン問題は、データの依存性と並列性を兼ね備えるプログラムであり、その書き方によって並列性と逐次性を自由に調整できる。FからJの行列乗算プログラムは、2つのベクトルの内積や行列の転置などの操作も含む、データの依存性が低く並列性の高いプログラムである。

表2に示されるプログラムのステップは、PE数無限、簡約実行時間一定、メモリアクセス遅延およびパイプラインストールなどのない理想的な条件下で、並列簡約が行われた場合のプログラム実行サイクルを表わしている。平均並列度は、プログラムの実行において生成されるタスク数をステップで割ったものである。使用するベンチマークプログラムは、タスク数が数千から数百万まで、平均並列度が数十から数万までの幅広い範囲で選んだ。従って、これらのベンチマークプログラムから得られたシミュレーション結果は、一般性を持ち、かつ信頼性を備えていると言える。

表2: ベンチマークプログラム

Program		Number of redexes	Step	Average parallelism
A	4queens	4099	236	17.369
B	5queens	19793	301	65.757
C	6queens	101693	366	277.850
D	7queens	482697	431	1119.947
E	8queens	2473187	496	4986.264
F	10x10	2562	21	122.000
G	20x20	18112	21	862.476
H	30x30	58662	21	2793.429
I	40x40	136212	21	6486.286
J	50x50	262762	21	12512.476

4.3 性能評価および考察

システム性能について比較検討するため、シミュレーション実験は3節で考察した3手法について行った。ただし、プログラムEの幅優先法による結果については、シミュレーション実験において非常に長い時間と膨大なメモリを必要とするため、結果が得られなかった場合もある。なお、ソフトウェアシミュレータはC言語で記述した。

はじめに、各手法の比較とシステムの有効性を検討するために、システムの性能評価における指標のひとつであるPE稼働率の計測を行った。図7に、結果を示す。図7より、各プログラムにおいてタスクが多い程、あるいは並列度が高い程、PEの高い稼働率の達成が可能であることが分かる。しかし、プログラム実行時の遅延評価のために、高い並列度が抽出できない場合（プログラムA,Bなど）や、逆に局所的に起こる過度の並列化によりメモリアクセス競合が十分に緩和できない場合など、必ずしも高い稼働率が得られるとは限らない。これらの問題の解決は、今後の課題である。また図7より、幅優先/深さ優先法の有効性が示される。特に、依存関係のあるタスクが多く生成されるN-クイーン問題において、その有効性は顕著に現われている。さらに、PE稼働率の高さは、プログラムが高速に実行されていることを暗示している。実際に、1PEの場合と比較した高速化率については、プログラムの持つ並列度が高い程PE数16に近づく結果が得られた。

次に、データ参照の局所性の利用を確認するために、各手法によるキャッシュおよびメモリアク

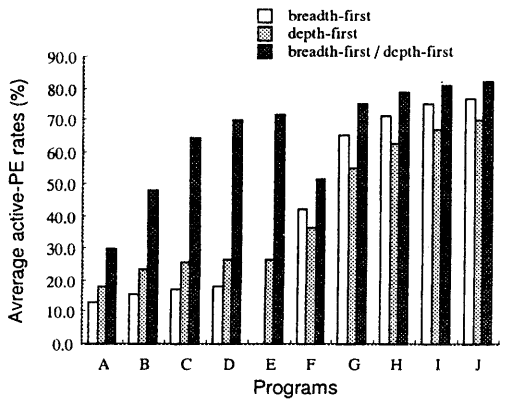

図7: PE平均稼働率

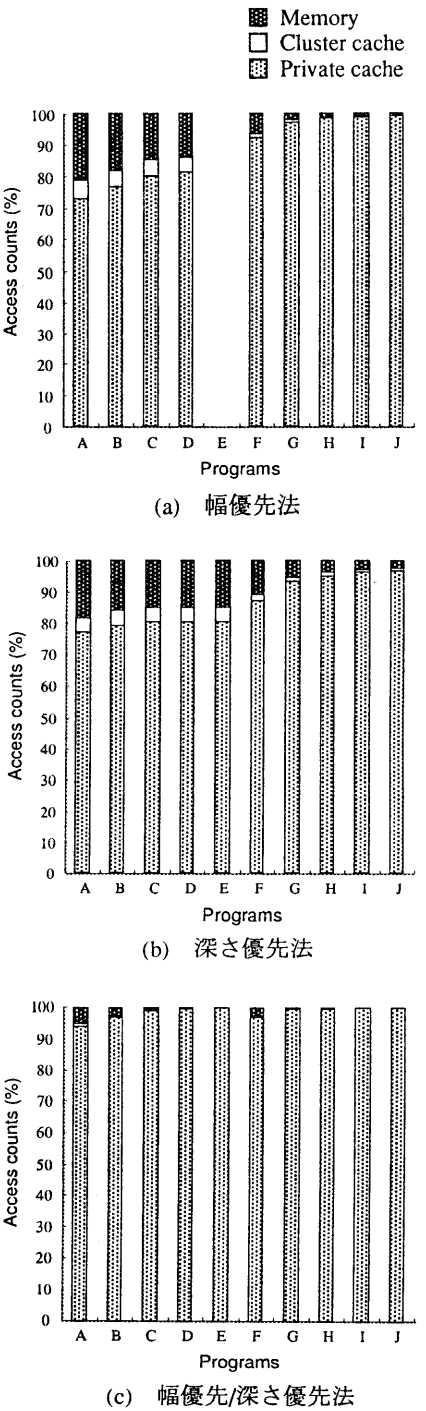

図8: キャッシュおよびメモリアクセス回数

セス回数を計測した。その結果を、図8(a)から(c)に示す。ここで、各ベンチマークプログラムの大きさの違いから、結果を総アクセス回数で正規化してある。シミュレーションを単純化するため、簡約グラフからタスクを検出する際にグラフ構造から静的にアクセス先を見積り、それぞれアクセス回数をカウントした。図8より、いずれの手法の場合もクラスタキャッシュを設けることによって、メモリアクセスを20％から30％程度減少させることが可能であることが分かる。さらに、幅優先/深さ優先法を使用することによって、参照の局所性は有効に利用されることが分かる。ここで、簡約に必要なデータオブジェクトをキャッシュあるいはメモリから読み取る場合、データオブジェクトを構成する複数セルのうちヘッダセルのアドレスのみが用いられ、これに続くデータセルはこの1回のアクセスにおいて一度に読み取られる。一方、タスクの簡約のため各PEからプライベートキャッシュ上のデータオブジェクトに対して行われるアクセスは、セル単位であっても数回もしくはそれ以上である。従って、キャッシュのヒット率を考慮した場合、図8に示したキャッシュのアクセス率より高い値が得られると考えられる。

以上より、マルチプロセッサシステムを利用する効果、およびキャッシュの有効性が示された。また、提案する幅優先/深さ優先法を用い、データの参照局所性を考慮することによって、効率のよい動的なタスク割り当てが可能であることが分かった。

5 むすび

本論文では、関数型言語FLの処理系としてFL階層化並列簡約システムに注目し、理想的なシステム性能を達成するための、グラフ簡約におけるタスク割り当て手法について検討した。

シミュレーション実験においてPEの平均稼働率を計測した結果、高い並列度を有するプログラムにおいて、また幅優先/深さ優先法を用いることにより、システム資源の有効利用が可能であり、プログラム実行の高速化が実現されることが確認された。特にN-クイーン問題などの、高並列性とタスク間の依存関係が多いプログラムにおいてその有効性は顕著となる。また、キャッシュおよびメモリアクセス回数を計測した結果、データ参照の局所性が有効に利用されることが示された。これよりキャッシュの効果が確認され、同時に幅優先/深さ優先法によりその効果は増加されることが示された。以上より、システムの有効性、およびデータの参照局所性を考慮した動的な幅優先/深さ優先タスク割り当て手法の有効性が示された。

今後の課題として、タスクの投機的推論や先行割り当てなど、プログラムの持つ並列度を十分に引き出すための手法の導入が挙げられる。また、キャッシュなどについてのより詳細な性能評価を行う必要がある。

参考文献

[1] H. Shen, H. Kobayashi and T. Nakamura, "Incorporating the Parallel Processing Techniques with the Demand-driven Model of Functional Programming Languages," the Proc. of 1993 IEEE Reg, 10 International Conference on Computers, Communication and Automation, pp146-149, 1993.

[2] H.Shen, H. Kitajima, H. Kobayashi and T. Nakamura, "A hierarchical parallel reduction system for the functional language FL," Proc. of High Performance Computing Conference'94, pp270-278, 1994.

[3] P. L. Wadsworth, "Semantics and paragmatics of the lambda calculus," PhD thesis, Oxford Univ., England, 1971.

[4] H. Shen, H. Kobayashi and T. Nakamura, "Developing the Lambda Calculus for FL-oriented Parallel Reductions," Proc. of Third International Conference for Young Computer Scientists, pp.649-650, 1993.

[5] R. L. Graham, "Bounds on multiprocessing timing anomalies," SIAM journal applied mathematics, Vol. 17, No. 2, pp.416-429, 1969.

[6] Koen Langendoen, "Graph reduction on shared-memory multiprocessors," PhD thesis, Amsterdam Univ., 1993.

[7] R. F. H. Hofman, K. G. Langendoen, and W. G. Vree, "Scheduling consequences of keeping parents as home," Proc. of Parallel and Distributed Systems, Taiwan, National Tsing Hwa Univ., pp580-588, 1992.

[8] P. Hudak, L. Smith, "Para-functional programming: A paradigm for programming multiprocessor systems," Proc. of 13th Principles of programming languages, pp243-254, 1986.