並列グラフ簡約システムにおけるタスク割当て手法とメモリ参照局所性評価

北島宏之[†] 中泉光広[†] 沈紅[†]
小林広明[†] 中村維男[†]

関数型言語は，手続き型言語と異なり，参照透明性や検証の容易性，プログラムの高い生産性など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大きく制限されてきた．そこで本論文では，共有メモリ型マルチプロセッサシステム上で関数型言語の簡約を高速に行うための，タスク割当て手法を提案する．関数型プログラムの実行を計算機上で実現する方法として広く利用されているグラフ簡約は，一般に処理粒度が小さくなるという欠点がある．したがって，プロセッサの有効利用とメモリアクセスなどの実行時オーバヘッドの抑制を考慮したタスク割当てが重要となる．本論文で提案するタスク割当て手法は，プログラム実行時においてデータ参照の局所性を動的に考慮しながら，並列タスクの検出およびそのプロセッサへの割当てを行う．提案するタスク割当て手法をローカルメモリおよび共有メモリを持つマルチプロセッサシステムに適用し，シミュレーションによる性能評価を行った結果，大域的なメモリアクセスの抑制とプロセッサ数に比例したプログラム実行の高速化率が達成され，提案手法の有効性を確認した．

Task Scheduling Strategies and Their Locality Evaluation of Memory References on a Parallel Graph Reduction System

Hiroyuki Kitajima,[†] Mitsuhiro Nakaizumi,[†] Hong Shen,[†]
Hiroaki Kobayashi[†] and Tadao Nakamura[†]

Functional programming languages have many appealing properties such as referential transparency and high programming productivity. On the other hand, the inefficiency of their implementation on conventional computers has prevent them from wide acceptance. In this paper, we propose a task scheduling strategy for high-speed processing of functional programs on a shared-memory multiprocessor system. To reduce shared-memory accesses in parallel graph reduction, the proposed task scheduling strategy allocates tasks to processors by taking the locality of data references among the tasks into account dynamically. Software simulation experiments on a multiprocessor system with the proposed strategy show that speedups of program processing in proportion to the number of processors can be achieved by making good use of local and cluster cache memories. As a result, the effectiveness of the proposed scheduling strategy with locality consideration is revealed.

1. はじめに

関数型言語は，参照透明性，検証容易性やプログラムの高い生産性など多くの有用な特徴を持つ．特に，参照透明性や並列実行の容易性は，マルチプロセッサシステムによる関数型プログラムの高速処理を期待させる．

関数型プログラムを計算機上で実現する方法として，主にグラフ簡約[1)]が利用されている．グラフ簡約は，関数型プログラムの並列処理や遅延評価などの実現に有効な手法であるが，グラフの簡約実行時において一般に処理粒度が小さくなる傾向がある．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，細粒度による過度のタスク生成やメモリアクセスの増加などの簡約実行時におけるオーバヘッドが大きくなり，期待する処理速度の達成が困難となる．したがって，システムの処理性能を十分に引き出すためには，タスク生成の制御やキャッシュによるメモリアク

† 東北大学大学院情報科学研究科
Department of Computer and Mathematical Sciences, Graduate School of Information Sciences, Tohoku University

セス遅延の減少を考慮した効果的なタスク割当て☆が必要不可欠である．そこで本論文では，関数型プログラムにおけるデータ参照の局所性について検討し，プログラムの持つデータ参照局所性を積極的に利用することにより，共有メモリ型マルチプロセッサシステムの共有メモリアクセスを抑制するタスク割当て手法を提案する．

これまでの関数型アーキテクチャのためのタスク割当てに関する研究[2)~12)]では，負荷バランスと処理要素（processing element，以下PEとする）の有効利用について述べられてはいるが，タスク生成の制御や大域的なメモリアクセス頻度の抑制などに関して，データ参照の局所性を積極的に利用しているとはいい難い．これらの研究におけるタスク割当て手法としては，深さ優先的にタスク割当てを行うLIFO法[4)]や，深さ優先的にタスク割当てを行いつつ必要に応じて遅延的に幅優先的なタスク割当てを行うLIFO/FIFO法[10)]が主に利用されている．しかし，LIFO法は，タスクの生成待ちやメモリへのアクセス待ちによって，一般にPEの有効利用が困難となる欠点を持つ[4),6)~9)]．また，LIFO/FIFO法では，簡約グラフをルート側から切り取ることによってタスク粒度を大きくし，システムの負荷バランスが維持されるためLIFO法と比較してPEの有効利用が期待できるが，PEへのタスク配置のオーバヘッドを増加させる危険性が生じる[5),10)]．これらの問題は，グラフ簡約においてタスクの生成および処理が頻繁に繰り返される場合や，PEに対して過度の並列タスクが生成される場合などにおいて無視できなくなる．

本論文で提案するタスク割当て手法では，基本的に幅優先法を用いることによってPEの有効利用を可能とし，さらに深さ優先法との切換えを動的に行うことによって大域的なメモリアクセス頻度を抑制し，負荷バランスを調整することができる．また，簡約グラフ上の最小の可簡約項をタスクとして検出し，PEへ割り当てることによって，タスク配置のオーバヘッドをLIFO/FIFO法を用いた場合に比べ低く抑えることができると考えられる．

はじめに，2章において本論文で扱う関数型言語FLおよびその簡約グラフについて簡単に説明を行い，関数型プログラムにおけるデータ参照の局所性について検討する．次に，3章において，本論文で提案する参照の局所性を考慮した動的なタスク割当て手法について述べる．4章では，提案するタスク割当て手法を共有メモリ型マルチプロセッサシステムに実装し，システムの性能評価を行う．ここで，共有メモリとローカルメモリを持つ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[13),14)]を評価対象とする．シミュレーションによる性能評価を通じて，提案するタスク割当て手法の有効性を示す．5章はまとめである．

☆ ここでのタスク割当ては，プログラムからのタスクの検出とプロセッサへの配置を含む．

2. 関数型言語FLとデータ参照の局所性

2.1 関数型言語FLおよび簡約グラフ

本論文では，関数型言語としてFL[15),16)]を使用する．FLはFP[17)]を基本とし，実用性を考慮してJ. Backusらによって開発された新しい関数型言語のひとつである．FLは，関数型言語が本来持つ参照透明性や並列実行の容易性などの特徴を持つ．このため，FLプログラムでは，その記述から並列実行可能な部分を抽出することができる．したがって，マルチプロセッサシステムによるFLプログラムの高速処理が期待できる．

基本的に，FLプログラムはひとつの関数であり，通常この関数は複数の関数から構成される．例として，**図1**に2つの行列の乗算を行うFLプログラムを示す．ここで，*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関数であり，同様に *IP* はベクトルの内積を求める関数である．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．この書換え操作を簡約と呼び，簡約される関数と値の対を可簡約項と呼ぶ．簡約は，最終結果としての正規形が得られるまで繰り返される．

FLプログラムを関数の結合と見ることにより，プログラムのグラフ表現が可能である．**図2**に，図1に示した行列乗算プログラムの簡約グラフを示す．図2において，“@”は関数の作用関係を表すためのノードであり，左側子ノードである関数が右側子ノード関数の結果に作用する．関数型プログラムの実行を実現する手法として広く用いられている並列グラフ簡約[1)]は，概念的に簡約グラフの持つ複数の可簡約項に対する並列な書換え操作からなる．この書換え操作はグラフにおける簡約であり，可簡約項の検出，簡約実行，および可簡約項の生成なども含む簡約結果のグラフへの反

```
def mmul ≡ α:(α:IP)∘(α:distl)∘distr∘[s1, trans∘s2] where
{
    def IP ≡ add∘(α:mul)∘trans
}.
```

図1 行列乗算FLプログラム
Fig. 1 Sample FL program of matrix multiplication.

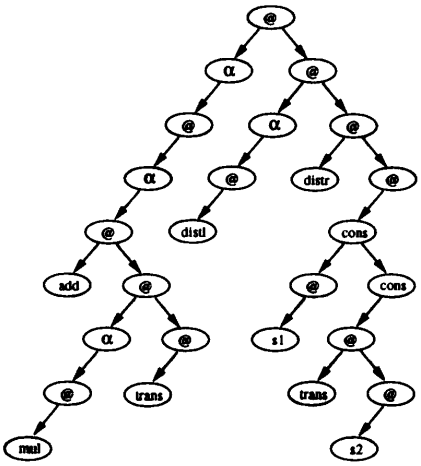

図2 行列乗算プログラムの簡約グラフ
Fig. 2 Reduction graph of the matrix multiplication program.

映から構成される．このように，簡約グラフではプログラムにおける関数の結合がノードの結合で表される．したがって，グラフの簡約実行時では，逆にノードの結合関係により関数の作用関係が一意に決定できる．

2.2 FL プログラムにおけるデータ参照の局所性

FL プログラムにおいて，関数の作用形式と簡約に使用される値から，可簡約項間の逐次性と並列性が決定される．簡約グラフにおいても同様に，グラフの各ノードの結合関係から関数の依存関係が一意に決定される．なお，値であるデータオブジェクトの関数への影響は，タスクの生成などを介してノードの結合関係として現れる．本節では，FL プログラムの高速実行を目標に，FL プログラムの持つ逐次性と並列性に基づくデータ参照局所性について検討する．ここで，計算機上における FL プログラムの実行である FL プログラムのグラフ簡約について論じるため，**図 3** に示すようなマルチプロセッサシステムを仮定する．並列グラフ簡約に必要とされる大域アドレス空間は，共有メモリ（Global Memory）により提供される．また，データ参照の局所性を利用するため，PE はローカルメモリ（Local Memory）を持つ．さらに，局所的に発生する大域的な共有メモリアクセスをより減少させるため，複数の PE をクラスタ化しクラスタキャッシュ（Cluster Cache）を設けている．

共有メモリ内の FL 簡約グラフから検出された可簡約項は，タスクとして PE に配置され，簡約が行われる．ここで，タスクは関数と簡約に使用されるデータオブジェクトから構成される．逐次実行されるタスクを順に同じ PE に配置し簡約することによって，データオブジェクトのために必要なアクセスのうち，共有メモリへのアクセスを著しく減少させることができる．

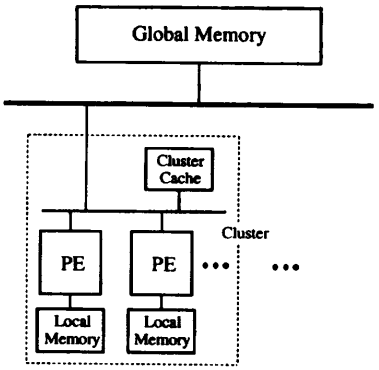

図3 マルチプロセッサシステムモデル
Fig. 3 Multiprocessor system model.

結果として，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可能である．これが，逐次簡約タスクに対する参照局所性である．たとえば，“$f \circ g : x$”というプログラムの場合，はじめに関数“g”がデータオブジェクト“x”に作用することによって可簡約項“$g : x$”の簡約が行われ，その結果に“f”が作用する．この場合，データオブジェクトへのアクセスは，PE とローカルメモリとの間に限定できる．

一方，並列実行が可能であるタスクの場合には，それらを複数の PE で並列に簡約することによって，プログラム簡約の高速化が期待できる．ここで注意すべき点は，タスクの並列簡約がその前後にタスクの並列化，およびそれら並列簡約結果の統合をともなうことである．並列簡約タスクに対する参照局所性は，簡約の結果として並列簡約タスクの生成を行うタスク，もしくはそれら簡約結果を統合するタスクと，並列簡約タスクとの依存関係により生じる．クラスタ化を考慮しない場合，生成タスクや統合タスクと並列簡約タスク間におけるデータオブジェクトの授受は共有メモリを介して行う必要がある．一方，クラスタ化された PE では，クラスタ内の共有データへのアクセスはクラスタキャッシュ上で行えることから，共有メモリへのアクセスを抑制することが可能となる．したがって，並列簡約タスクを参照局所性を利用して効率良く PE に配置するためには，生成タスク，統合タスク，および並列簡約タスクを可能な限り同じクラスタ内の PE に配置することが必要である．

例として，“$[f, g, h] \circ k : x$”というプログラムを考える．まずはじめに，“$k : x$”がタスクとして簡約される．その結果を“x'”とすれば，次に“$[f, g, h] : x'$”が“$f : x'$”，“$g : x'$”および“$h : x'$”の 3 つの可簡約項に分割され簡約される．これらは，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．最後に，これらの並列簡約の結果が 1 つのリ

ストとして統合され，プログラムの実行結果として得られる．この場合，データオブジェクトのためのアクセスをクラスタ内に限定できる．

逐次簡約タスクおよび並列簡約タスクに対する参照局所性を考慮することによって，共有メモリへのアクセスを減少させることができ，結果として，局所的にメモリアクセスを解決できる粒度の大きなタスクを各PEへ割り当てることが可能になる．これらのデータ参照局所性は，関数の依存関係としてグラフ走査段階においてポインタの比較などにより容易に判断することが可能であり，ソフトウェアもしくはハードウェアによる実現を考慮しても大きな負担にはならないと考えられる．

3. データ参照局所性を考慮した幅優先/深さ優先タスク割当て手法

3.1 従来のタスク割当て手法

2章で述べた関数型プログラムが持つデータ参照の局所性をタスク割当て時に考慮することにより，共有メモリアクセス頻度が減少し，メモリ待ちによるPEの非稼働状態を可能な限り少なくすることができる．しかしながら，データ参照の局所性の考慮の範囲は，簡約グラフの走査順序に依存する．したがって，タスク割当てでは，PEを有効利用するようにグラフから十分な並列度を抽出し，あわせて参照局所性の考慮によりメモリアクセスを可能な限り削減するようにタスクをPEに配置することが必要とされる．

以下，タスク割当て手法の説明に抽象化した簡約グラフを用いる．抽象化した簡約グラフにおいて，ノードはタスクを，エッジはタスクどうしの依存関係を表す．説明のためタスクの走査優先度はグラフにおいて右側が高いものとし，簡単のためタスクのグラフからの検出時間やPEへの配置時間は考慮せず，さらに各タスクのPEにおける簡約時間はすべて等しいと仮定する．タスク検出では，リーフである末端ノードがタスクとしてグラフより取り除かれる．タスクの簡約結果は，親タスクの検出に反映される．システム構成として，PE数を8とする．

従来のタスク割当てでは，PEを有効に利用するための簡約の高並列化を目的として，簡約グラフを幅優先に走査し，タスクを可能な限り多く検出していた[4]．幅優先法を抽象化した簡約グラフに適用した場合の，各PEへのタスク配置を図4に示す．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番号を示し，bはそのタスクが実行されるときの簡約ステップを表す．図4より，幅優先法は可能な限り

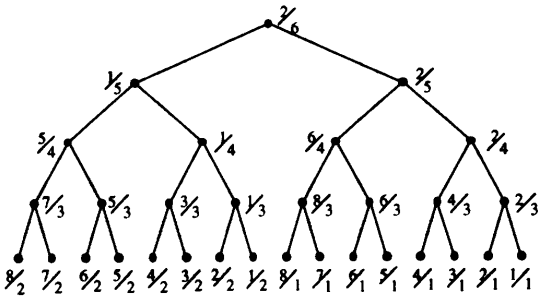

図4 幅優先タスク割当て
Fig. 4 Breadth-first task scheduling.

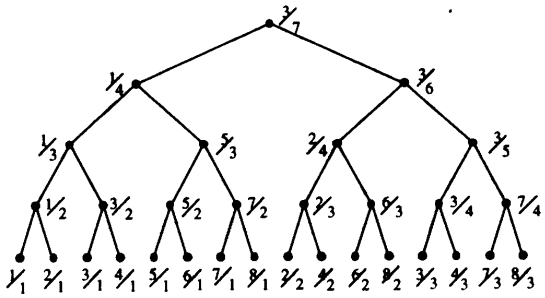

図5 深さ優先タスク割当て
Fig. 5 Depth-first task scheduling.

多くのエッジを走査しタスクを検出するため，PE利用率の向上に有効であると考えられる．しかしながら，幅優先法では，並列タスクの検出を最優先し，タスクのデータ参照局所性が有効に利用されないため，簡約に必要なデータオブジェクト移動のための共有メモリアクセスが頻繁に生じる．共有メモリアクセスの一部はクラスタキャッシュアクセスに代替できるが，幅優先法では大域的なメモリアクセス自体が増加するために，結果として期待するほどのシステム性能の向上が得られないと考えられる．たとえば，図4におけるステップ1および2で示されるタスクでは，データ参照の局所性が利用されていないために，共有メモリアクセスが必要である．また，可能な限り多くのタスクを検出するようにグラフが走査されることによって，グラフ走査途中に生成される中間ノード数の増大によるメモリ消費や，走査時間の増加などのオーバヘッドが生じる．たとえば，図4の簡約ステップ3で示されるタスクは，グラフ走査段階においてすでに中間ノードとして生成される．

中間ノードおよびタスク生成のオーバヘッドを最小に抑え，データ参照の局所性を最大限に利用するためには，簡約グラフを深さ優先に走査しタスクを検出する方法が適する[4),6)~9)]．図5は，簡約グラフに深さ優先法を適用した場合のタスク配置結果である．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度を抑制するようにタスクが簡約されるため，幅優先法に比べ参照の局所性が有効に利用される．また，中間ノードおよびタスク生成のオーバヘッドが最小に抑えられることが分かる．しかし，深さ優先法ではタスク依存性の影響が最も大きくなり，主にタスクの生成待ちによる非稼働状態のPE数が増加し，PEの有効利用が困難となる．この例では，図4のステップ5で2つのタスクが並列に簡約されているが，図5のステップ5では1つのタスクだけが簡約されている．また，深さ優先法では，局所性を高めるようにタスク走査が行われるため，タスクの並列度がPE数以下となる場合が多い．この場合，非稼働状態のPEのために新たなタスクが生成される結果，データ参照局所性の利用が困難になり，クラスタ化が行われた場合のクラスタキャッシュの有効利用は期待できない．これは，図5の簡約グラフにおけるタスク簡約のバランスの悪さに現れている．

3.2 幅優先/深さ優先タスク割当て手法

本論文では，PE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，基本的には幅優先法を用いて並列タスクの生成を行うが，タスク生成の制御およびデータ参照局所性の有効な利用のために深さ優先法との動的な切換えを行う．ここで，幅優先法と深さ優先法の切換えは，タスク検出時のシステム負荷から動的に算出される閾値によって決定される．図6に，幅優先/深さ優先法を簡約グラフに適用した場合のタスク配置結果を示す．各ノードの記号は，図4と同様である．ここでは，システム負荷としてPEの稼働状態のみを考慮し，閾値として非稼働PE数を用いている．図6では，簡約グラフを幅優先で走査し，検出されるタスク数が最初の閾値8を超える時点で深さ優先法への切換えが行われている．このように，幅優先/深さ優先法によりPEに対して必要十分な並列度を抽出する．また，図6におけるステップ1，2，3で示されるタスクのように，本手法では結果的に各PEに割り当てられる部分木に対して深さ優先法が利用されるため，PEに対するデータ参照の局所性が考慮されることが分かる．さらに，図6の簡約グラフにおけるバランスの良いタスク簡約は，本手法によってクラスタキャッシュが有効に利用されることを示している．たとえば，8PEを1〜4番および5〜8番に分け，それぞれクラスタ化が行われる場合，図6において，グラフ中央を境に左右の部分木が各クラスタ内で簡約され，クラスタキャッシュによって共有メモリアクセスを削減できる．

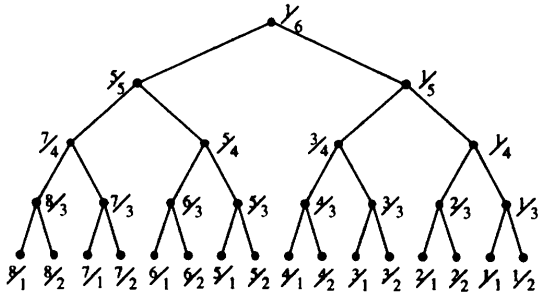

図6 幅優先/深さ優先タスク割当て
Fig. 6 Breadth-first/Depth-first task scheduling.

付録に，幅優先/深さ優先タスク割当て手法のアルゴリズムを示す．ここで，**NPn**（Node Pointer for PEn）は各処理要素**PEn**に対応し，**NPn**が指すノードは，幅優先/深さ優先法における幅優先法と深さ優先法との切換えが行われる起点ノードとなる．本アルゴリズムでは，非稼働状態のPE数（セットされていない**NPn**数）を閾値として用いている．簡約グラフにおいて，**NPn**の指すノードから**NSPn**（Node Search Pointer for PEn）によって簡約グラフが走査され，タスクが検出される．**PEn**（$n = 1, 2, ..., 8$）が非稼働状態である場合，**NPn**の指すリーフまたは**NSPn**によって発見されるリーフを非稼働状態の**PEn**に配置することにより，図6に示す結果が得られる．

LIFO/FIFO法[10]によるタスク割当てでは，PE間での大粒度タスクの転送が必要であるため，タスク配置のオーバヘッドが非常に大きくなる．一方，本手法では，基本的な関数と値の対を単位とした通常のタスク粒度を用いるため，LIFO/FIFO法に比べてタスク配置のオーバヘッドを抑制することができ，さらに共有メモリアクセスの抑制により，見かけ上大粒度のタスクのPEへの配置が可能になる．

4. 性能評価

本章では，提案したタスク割当て手法の有効性を明らかにするために，シミュレーションによる性能評価結果を示す．

4.1 シミュレーションモデル

4.1.1 FL階層化並列簡約システム

並列グラフ簡約をマルチプロセッサシステム上で実現するためにシステムに要求される機能は，簡約グラフにおけるタスクの走査・検出のためのアドレス操作やスタック操作，および適切なPEへのタスク配置のためのポインタ比較などである．これらは，比較的簡単な回路で実現可能であると考えられる．そこで本章

では，これらの機能と 2.2 節において仮定したメモリ階層を備える FL 階層化並列簡約システム[13),14)]を評価対象とした．

図 7 に，FL 階層化並列簡約システムの基本ブロック図を示す．共有メモリ（Main Memory）は，その記憶内容によりグラフメモリ（Graph Memory）とデータメモリ（Data Memory）に分割される．ここで，グラフメモリは簡約グラフのために用いられ，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られる．グラフメモリおよびデータメモリは，通常の計算機におけるインストラクションメモリおよびデータメモリに対応する．スケジューラ（Scheduler）は，グラフメモリ，タスクプール（Task Pool）および各 PE を監視しており，タスクプールの空状態や，すでに検出されたタスクとの依存関係などによりタスクの検出を行う．本論文ではこれまで，タスクは関数とデータオブジェクトから構成されるものとしてきたが，FL 階層化並列簡約システムでは，通常の計算機構造との対応と各 PE に割り当てられる対象の意味から，プログラムにおける関数を狭義にタスクとしている．

検出されたタスクは，スケジューラにより各 PE に対応するタスクプールにいったん配置される．この際スケジューラは，検出したタスクとすでに検出し配置されたタスクとの依存性を判断し，タスクを適切なタスクプールに配置する．タスクプールは，タスク用のキューであり，PE の稼働状態によってタスクを PE に配置する．各 PE は，簡約に必要とされるデータオブジェクトをローカルメモリ（Local Memory）へ読み込み，タスクの簡約を実行する．タスクの簡約結果はスケジューラを通じて簡約グラフに反映されるため，各 PE からのグラフメモリへのアクセスは必要とされず，したがってクラスタキャッシュ（Cluster Cache）はデータメモリ側だけに設けられている．タスクの検出，配置および PE における実行は，タスク簡約の 3 つのステージとして時間的に重複されパイプライン化される．

本システムは，FL の並列簡約システムとして特化されているが，その機能は汎用の共有メモリ型マルチプロセッサシステムで容易に実現可能なものである．たとえば，本システムにおけるスケジューラは，PE と比較して様々な処理を要求されるが，それらの処理でさえ簡単な比較演算やタスク検出のための単純なアドレス計算である．したがって，本論文で提案するタスク割当て手法は一般的なシステム上に実現可能であり，本章で示す評価結果は汎用の共有メモリ型マルチプロセッサシステムにおけるグラフ簡約アルゴリズムの評価としても有効であると考えられる．

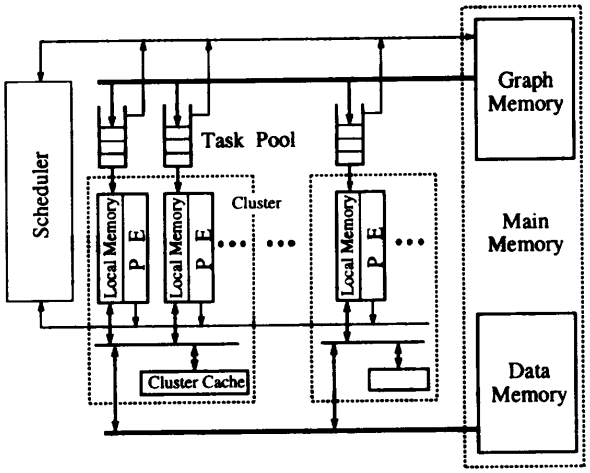

図 7 FL 階層化並列簡約システムの基本ブロック図
Fig. 7 Block diagram of the FL hierarchical parallel reduction system.

4.1.2 シミュレーション条件

図 7 に示したシステムについてシミュレータを構築し，システムの性能評価を行った．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，簡約パイプラインのパイプラインサイクルを用いる．
- PE 総数は 16 とし，16 PE を 4 PE ごとに 1 クラスタ化する（クラスタ総数は 4）．これは，システムを均等にクラスタ化することにより，タスク割当ての 3 手法におけるクラスタキャッシュの有効性を比較・検討するために設定した．
- 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，実際の演算を考慮して，タスクを構成する関数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〜数パイプラインサイクルを必要とし，これらは簡約実行時に決定されるものとする．
- ローカルメモリサイズは 4M バイト，クラスタキャッシュのサイズおよび構成は，データ参照の局所性および大域性の検討のため，2K バイト×4 クラスタとする．これらのサイズは，汎用マルチプロセッサシステムの 1 PE に備えられる一般的なそれらのサイズに比べて非常に小さい．これは，メモリアクセス競合の発生を促すことにより，シミュレーションにおいてタスク割当て手法

表1 キャッシュおよびメモリアクセス時間
Table 1 Cycle time of local memory, cluster cache and data memory accesses.

Case	Data in Local Memory	Data not in Local Memory	
		Cluster Cache hit	Cluster Cache miss (Data in Data Memory)
Cycle	1	3	7

の違いによる結果を顕著に示すために選択した．クラスタキャッシュはライトバック方式で使用され，データメモリのデータセルはクラスタキャッシュにダイレクトマップ方式で配置される．ここで，タスク簡約に使用されるデータオブジェクトは，複数のデータセルから構成され，1データセルサイズは20バイトとする．また，ローカルメモリ，クラスタキャッシュおよびデータメモリに対するアクセス（読み出し）時間を表1のように定める．書き込みに関しても，各アクセス先に対して同様の時間を要するが，逐次タスク簡約の場合，書き込みはローカルメモリに対してのみ行う．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．

4.1.3 ベンチマークプログラム

シミュレーションに用いたベンチマークプログラムを表2に示す．A～Eは，データの依存性と並列性を兼ね備えるN-クイーン問題（n-queens）である．F～Jは，データの依存性が低く並列性の高い行列乗算プログラム（$n \times n$）である．また，表2に各ベンチマークプログラムについて実行時に簡約されるタスク数（Number of redexes）および平均並列度（Average parallelism）を示す．ここで，平均並列度は，総タスク数を，理想的な並列簡約が行われた場合のプログラム実行サイクル数で割ったものである．理想的な並列簡約状態とは，PE数に制限がなく，タスクの簡約実行時間がすべて等しく，さらにメモリアクセス遅延などのない状態を意味する．使用した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．したがって，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思われる．

4.2 性能評価結果と考察

システム性能について比較検討するため，シミュレーションは3章で考察した幅優先，深さ優先，および幅優先/深さ優先の3手法について行った．タスクの走査および検出は各手法によって行われ，逐次簡約タスクはいずれの手法においても同一のPEに配置される．また，検出された並列簡約タスクに対するデータ参照局所性についても同様に考慮し，すでに検出されたタスクとの依存性を判断して，可能な限りクラスタキャッシュを有効に利用するようタスクを配置する．

表2 ベンチマークプログラム
Table 2 Benchmark programs.

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

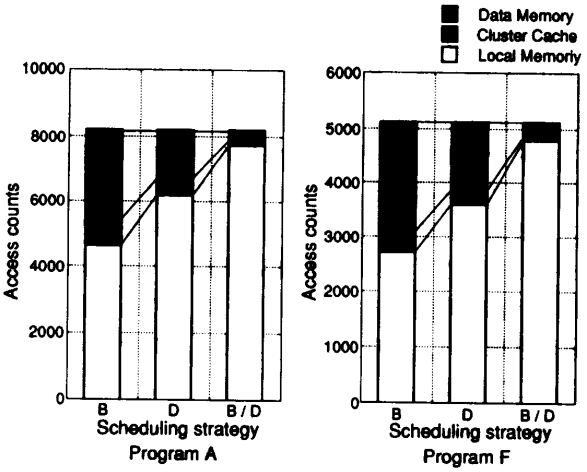

図8 メモリアクセス回数
Fig. 8 Memory access counts.

はじめに，各手法によるデータ参照局所性の利用を確認するために，プログラム実行時におけるローカルメモリ（LM: Local Memory），クラスタキャッシュ（CC: Cluster Cache）およびデータメモリ（DM: Data Memory）に対するアクセス回数（Access counts）を，プログラムAおよびFについて計測した（図8）．図8より，幅優先法（B: Breadth-first），深さ優先法（D: Depth-first），幅優先/深さ優先法（B/D: Breadth-first/Depth-first）の順に全アクセスにおけるLMへのアクセス比率が高く，幅優先/深さ優先法によるCCおよびDMへの大域的なアクセスの大幅な減少が可能であることが分かる．幅優先法では，他の手法に比べてCCに対するアクセス回数は多く，CCを有効に利用できているといえるが，大域的なアクセス自体が多く，システムの処理性能は大きく制限されると考えられる．また，深さ優先法では，幅優先法に

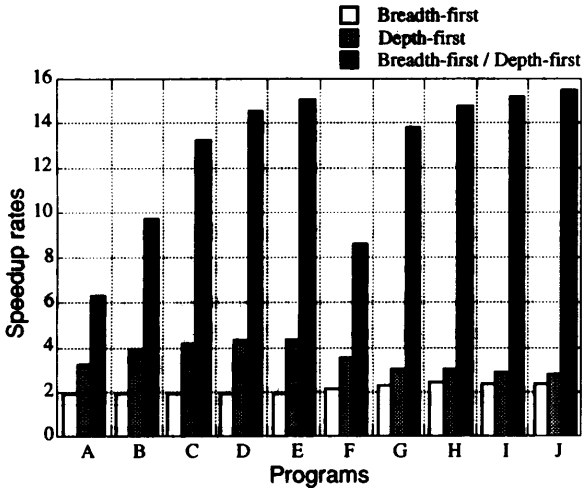

図9 1PEシステムに対するプログラム実行の高速化率
Fig. 9 Speedup rates of benchmark programs.

比べ，大域的なアクセスは少なく，またCCも有効に利用されているが，幅優先/深さ優先ほど大域的なアクセスを減少させることはできないことが分かる．計測の結果，いずれの手法においてもDMへのアクセスのうちCCによって削減されたアクセスの割合は20～30%であり，CCは各手法において同程度の有効性を示している．

次に，大域的なメモリアクセスのプログラム実行に及ぼす影響を確認するため，1 PEシステムに対するプログラム実行の高速化率（Speedup rates）を計測した（**図9**）．ここで，1 PEシステムとは1 PE - 1クラスタのシステムであり，プログラム実行の高速化率を公正に測定するため，CCサイズは8Kバイト（2Kバイト×4クラスタに相当）とした．

図9より，幅優先/深さ優先タスク割当て手法は，いずれのプログラムにおいても，他の手法に比べPE数（PEの並列度）16に近い高速化率が得られていることが分かる．幅優先法を用いた場合では，N-クイーン問題（A～E）および行列乗算（F～J）の各プログラムは十分な並列度を持つにもかかわらず，それら実行の高速化率は低く，またプログラムサイズの増大による高速化率の向上もほとんど現れていない．これは，各プログラムにおける最も小さいプログラム（AおよびF）でさえPE数に対して十分な並列度を持つが，各タスクの簡約実行にともなうデータオブジェクトのためのDMアクセスが，PEの増加にともない性能向上のボトルネックになると考えられる．

一方，深さ優先法を用いた場合，幅優先法に比べデータ参照の局所性が利用されており，また各プログラムがPE数に比べ十分な並列性を持つことから，並列性に関しても有効に利用できると思われる．しかしながら，深さ優先的なグラフ走査によって，逆に簡約グラフ上の局所的部分，つまり参照局所性がきわめて高い部分においてもタスクの並列化が行われ，それらのタスクが複数のPEに配置される．この結果，クラスタ間でのデータ転送の必要性からDMアクセスが増加し，速度向上率を抑制していると考えられる．このことは，F～Jの行列乗算における，プログラムサイズの増加に反する速度向上率の低下の原因と考えられる．

幅優先/深さ優先法では，各プログラムにおいてデータ参照の局所性と並列性が有効に利用でき，プログラムの平均並列度が高いほどPEの高い稼働率が達成される．しかしながら，N-クイーン問題のようなプログラムに含まれる局所的な並列度がPE数を下回ることが多いプログラムでは，プログラムの平均並列度がPE数以上であっても，結果的にプログラム実行の高速化率をPEの並列度に近づけることが困難な場合がある（プログラムA，B）．この問題の解決には，幅優先法と深さ優先法の切換えを行う閾値を動的に変化させタスクの先行割当てを行う方法など，プログラムの本来有する並列性をより多く抽出する方法の導入があげられる．一方，行列乗算プログラムのような局所的に大きな並列化が生じるプログラムの場合には，並列簡約タスクとその前後の生成タスクおよび統合タスクとの依存関係から，それら並列化が生じる前後におけるメモリアクセス競合の発生は避けられず，メモリアクセス待ちによるPEの非稼働状態が頻繁に生じる．プログラムG～Jのように，非常に多くのタスク簡約によってメモリアクセス遅延が全プログラム実行時間によって平均化され間接的に隠蔽される場合には，PE数に近いプログラム実行の高速化率が得られるが，プログラムFのようにタスクの生成が十分でない場合には，これらのメモリアクセス遅延が全プログラム実行時間に対して支配的となるため，プログラム実行の高速化は制限される．この問題の解決には，タスク生成の予測[18]を含むタスク割当て機構の実現があげられる．

シミュレーションモデルで仮定したPEでのタスクの平均簡約時間は，N-クイーン問題では約3.7サイクル，行列乗算では5.5～5.9サイクルであった．したがって，タスクの検出および配置時間はこれらによって隠蔽されるため，各タスク割当て手法およびプログラムの特徴はシミュレーション結果に直接現れているといえる．

5. おわりに

本論文では，並列グラフ簡約における動的タスク割当て手法として，幅優先/深さ優先タスク割当て手法を提案した．提案したタスク割当て手法の評価のため，関数型言語 FL の簡約グラフを用い，その処理系として FL 階層化並列簡約システムを対象にシミュレーションによる性能評価を行った．はじめに，各メモリ要素へのアクセス回数を計測した結果，データ参照の局所性を考慮した幅優先/深さ優先タスク割当て手法によって，大域的なメモリアクセスを大幅に減少できることが示された．次に，1 PE システムに対するプログラム実行の高速化率を計測した結果，提案した幅優先/深さ優先タスク割当て手法によって，PE 並列度に近いプログラム実行の高速化率が得られ，PE が有効に利用されていることが示された．データ参照局所性の効果的な利用は，各 PE がローカルメモリを持つ共有メモリ型マルチプロセッサシステムにおいて，大域的なメモリアクセスを飛躍的に減少させる．

今後の課題として，タスク生成予測などの投機的実行機構や，動的閾値を利用した先行割当て機構の導入があげられる．

謝辞 本研究を進めるにあたり，FL の資料を提供していただき，あわせて貴重なコメントをいただいた IBM Almaden Research Center の John H. Williams 博士に厚く感謝いたします．なお，本研究の一部は，平成 8 年度文部省科学研究費補助金（奨励 (A) 08780235）の援助を受けている．

参考文献

1) Wadsworth, C.P.: Semantics and Pragmatics of the Lambda Calculus, Ph.D. Thesis, Oxford Univ., England (1971).
2) Graham, R.L.: Bounds on Multiprocessing Timing Anomalies, *SIAM J. Appl. Math.*, Vol. 17, No.2, pp.416–429 (1969).
3) Hudak, P. and Smith, L.: Para-Functional Programming: A Paradigm for Programming Multiprocessor Systems, *Proc. 13th ACM Symp. Principles of Programming Languages*, pp.243–254 (1986).
4) Ruggiero, C.A. and Sargeant, J.: Control of Parallelism in the Manchester Data Flow Machine, *Proc. 3rd Functional Programming Languages and Computer Architecture, LNCS*, Vol.274, pp.1–15 (1987).
5) Watson, I., Woods, V., Watson, P., Banach, R., Greenberg, M. and Sargeant, J.: Flagship: A Parallel Architecture for Declarative Programming, *Proc. 15th IEEE/ACM Symp. Computer Architecture*, pp.124–130 (1988).
6) Augustsson, L. and Johnsson, T.: Parallel Graph Reduction with the ⟨ν, G⟩ Machine, *Proc. 4th ACM Conf. Functional Programming Languages and Computer Architecture*, pp.202–213 (1989).
7) George, L.: An Abstract Machine for Parallel Graph Reduction, *Proc. 4th ACM Conf. Functional Programming Languages and Computer Architecture*, pp.214–229 (1989).
8) Hammond, K. and Jones, S.L.P.: Profiling Strategies on the GRIP Parallel Reducer, Internal Report 10, Department of Computer Sciences, Glasgow Univ., Scotland (1991).
9) Maranget, L.: GAML: A Parallel Implementation of Lazy ML, *Proc. 5th Functional Programming Languages and Computer Architecture, LNCS*, Vol.523, pp.102–123 (1991).
10) Mohr, E., Kranz, D.A. and Halstead.Jr., R.H.: Lazy Task Creation: A Technique for Increasing the Granularity of Parallel Programs, *IEEE Trans. Parallel and Distributed Systems*, Vol.2, No.3, pp.264–280 (1991).
11) Hofman, R.F.H., Langendoen, K.G. and Vree, W.G.: Scheduling Consequences of Keeping Parents at Home, *Proc. Int. Conf. Parallel and Distributed Systems*, pp.580–588 (1992).
12) Langendoen, K.: Graph Reduction on Shared-Memory Multiprocessors, Ph.D. Thesis, Amsterdam Univ., Holland (1993).
13) Shen, H., Kobayashi, H. and Nakamura, T.: Incorporating the Parallel Processing Techniques with the Demand-Driven Model of Functional Programming Languages, *Proc. IEEE Int. Conf. Computers, Communication and Automation*, pp.146–149 (1993).
14) Shen, H., Kitajima, H., Kobayashi, H. and Nakamura, T.: A Hierarchical Parallel Reduction System for the Functional Language FL, *Proc. High Performance Computing Conference '94*, pp.270–278 (1994).
15) Backus, J., Williams, J.H. and Wimmers, E.L.: FL Language Manual (Preliminary Version), Research Report, IBM Almaden Research Center (1986).
16) Backus, J., Williams, J.H., Wimmers, E.L., Lucas, P. and Aiken, A.: FL Language Manual, Parts 1 and 2, Research Report, IBM Almaden Research Center (1989).
17) Backus, J.: Can Programming be Liberated from the Von Neumann Style? A Functional Style and Its Algebra of Programs, *Comm. ACM*, Vol.21, No.8, pp.613–641 (1978).

18) Murthy, P.V.R. and Rajaraman, V.: Implementation of Speculative Parallelism in Functional Languages, *IEEE Trans. Parallel and Distributed Systems*, Vol.5, No.11, pp.1197–1205 (1994).

付録 幅優先/深さ優先タスク割当て手法（セットされていないNPn数を閾値とする）

```
/* *GNSP : Global_Node_Search_Pointer */
/* *NSPn : Node_Search_Pointer_for_PEn */
/* *NPn  : Node_Pointer_for_PEn */

閾値 = セットされていないNPn数;

/* 幅優先/深さ優先法 */
while( !( Root_Node の簡約終了 ) ){

  /* －幅優先法の適用－
     大域的なタスク走査とタスク検出準備 */
  GNSP が, Root_Node から幅優先的にグラフ走査;
  if( GNSP が現在走査中のノード数  <  閾値 ){
    if( 現在走査中のノードがリーフ ){
      NPn をリーフであるノードにセット;
      閾値--;
    }
  }else{ /* 深さ優先法適用のための準備 */
    while( 閾値 > 0 ){
      GNSP が走査中のノードに, NPn をセット;
      閾値--;
    }
  }

  /* －深さ優先法の適用－
     各PEに対する深さ優先的な
     グラフ走査と, タスクの検出および配置 */
  for( PEn について ){
    if( ! ( PEn が稼働 ) && ( NPn がセット) ){
      NSPn により, NPn の指すノードから深さ優先
        的にグラフを走査し, リーフを発見;
      リーフを検出し, 対応するPEnに配置;
    }
  }

  if( NPn の指すノードが検出済み ) 閾値++;
}
```

(平成7年12月25日受付)
(平成8年 9 月12日採録)

北島 宏之（学生会員）

昭和45年生．平成5年東北大学工学部機械工学科卒業．平成7年同大学大学院情報科学研究科情報基礎科学専攻博士課程前期修了．現在，同博士課程後期に在学中．並列計算機アーキテクチャ，関数型言語，コンピュータグラフィックスの研究に従事．IEEE会員．

中泉 光広

昭和48年生．平成8年東北大学工学部機械知能工学科卒業．現在，同大学大学院情報科学研究科情報基礎科学専攻博士課程前期に在学中．並列計算機アーキテクチャ，関数型言語の研究に従事．

沈 紅

昭和40年生．昭和61年重慶大学工学部計算機科学系卒業．平成元年同大学大学院工学研究科修士課程修了．平成3年東北大学大学院工学研究科博士課程修了．工学博士．現在，同大学院情報科学研究科助手．並列計算機アーキテクチャ，関数型言語の研究に従事．電子情報通信学会会員．

小林 広明（正会員）

昭和36年生．昭和58年東北大学工学部通信工学科卒業．昭和63年同大学大学院工学研究科情報工学専攻博士課程修了．工学博士．現在，同大学院情報科学研究科助教授．平成7年スタンフォード大学客員助教授併任．並列計算機アーキテクチャ，コンピュータグラフィックスの研究に従事．

中村 維男

昭和19年生．昭和47年東北大学大学院工学研究科博士課程修了．工学博士．同年同大学工学部助手．昭和53年同大学助教授．昭和63年同大学教授，現在に至る．平成6年よりスタンフォード大学客員教授併任．研究分野は，計算機アーキテクチャ．