# 分散メモリ型商用並列計算機上でのデータフロー言語の配列参照の効率化

稲永 健太郎，日下部 茂，雨宮 真人
E-mail:{inenaga,kusakabe,amamiya}@al.is.kyushu-u.ac.jp
九州大学大学院システム情報科学研究科知能システム学専攻
〒816 福岡県春日市春日公園6-1

## Abstract

一般に，並列処理では演算の同期や実行順序のスケジューリングなどの実行制御が問題となるが，データフロー言語による記述では，繁雑な実行制御を必要とすることの多い構造データ処理を含んだ場合でも並列実行制御を明示的に記述することなく容易に並列処理記述が可能である．しかしながら，この構造データ処理における記述の抽象度の高さにより構造データアクセスにおける実行時の実アドレス計算が非効率的となり，また頻繁な細粒度構造データ処理による過度のオーバヘッドが生じたりするなど，実行効率の低下が避けられない [1].

本稿では，分散メモリ型商用並列計算機 AP1000 を実装対象として構造データ処理の1つである配列参照アクセスにおけるオーバヘッドの削減を行ない高効率実行のための手法について述べる．

例題として画像処理アルゴリズムの1つである GNC のプログラムを用いて各手法の評価を行なった結果，適切な手法によりデータフロー言語配列参照の効率化の実現が可能であると確認できた．

## 1 はじめに

一般に，並列処理では演算の同期や実行順序のスケジューリングなどの実行制御が問題となる．そこで我々は，本質的に並列性を内在し実行制御はデータ依存則に沿って自動的になされるデータフローモデルに基づく言語の研究を行なっている．データフロー言語は，値による自動的同期や明示的な並列実行制御の記述が不要，そして抽象度の高い構造データ処理記述などの並列処理記述における魅力的な特徴を持つ．また，並列処理において高い並列性を追求するために，プログラムの評価方法として non-strict なセマンティクスを持つプログラムをデータフロー的に eager に評価すること (lenient 方式) が有効であると考えている．さまざまな non-strict 性の中でも特に構造体の non-strict 性が有用である [2]．これは，構造体全体が定義される前に要素単位で構造体へのアクセスが可能となるため，実行時に高い並列性を抽出することができるからである．このことから，並列処理ではデータフローモデルを考慮しデータ側に同期処理機構を持つ構造体が有効であると考える．

このように，同期処理機構を持つ non-strict な構造体は優れた特徴を持つものの，非データフローの汎用計算機を実装対象とした場合には，実行制御のほとんどをソフトウェアで実現する必要がありそのオーバヘッドが問題となる．次章では，特に構造体処理によるオーバヘッドに関する問題点を詳しく述べる．

## 2 細粒度構造データアクセスの問題点

1章で述べたように，非データフローの汎用計算機を実装対象とした場合には並列実行制御のための処理をソフトウェアで実現する必要がある．しかしながら，このソフトウェアによる実現，特に要素単位の構造データアクセスを含む細粒度処理を実現する場合には，頻繁な細粒度処理による過度の並列処理オーバヘッドが生じてしまう場合が

多く，実行効率の低下の一因となっている．ここで，具体的に細粒度構造データアクセスのオーバヘッドとしては次のようなものが挙げられる．

1. 構造データアクセスのための実アドレス計算

   ■ 高レベル配列処理プリミティブ (関数) の頻繁な呼び出し

   (これは抽象度の高い構造データアクセスの記述および確実な実行保証の得るために生じるものである．また配列処理プリミティブは階層化されている)

2. 頻繁な細粒度構造データアクセス

   ■ I-structure[3] を用いた構造データの実装による要素単位での細粒度構造データ処理

3. 頻繁な細粒度データ通信

   ■ 頻繁な細粒度データ通信

   - 通信プリミティブ (通信関数) の呼び出し
   - セルネットワークを用いたデータ通信

これらは，アドレス計算によるもの (1. に相当) と細粒度処理によるもの (2. および 3. に相当) に大きくわけることができる．

本稿では，構造データ処理の中でも使用頻度の高い配列参照アクセスを取り上げ，次章で配列参照アクセスの実行効率低下の要因であるアドレス計算および細粒度処理に対する効率向上のための技法を次章で述べる．

# 3 配列参照オーバヘッドの削減手法

## 3.1 実アドレス計算コストの削減手法

現在生成される 実行の保証がなされている naive なコードにおいて，抽象度の高い記述がされている配列参照は，配列を指すポインタと添字のセットを引数として渡すことにより内部表現に関わらず対象要素にアクセスが可能となっている．しかしながら，添字から実アドレスに至るまでの計算を行なうのは，階層化されている配列処理プリミティブ (関数) のうち高レベルのものを用いるため実装上多くのプリミティブを呼び出してしまい非効率となる．そこで，複数の近傍要素の実アドレスの計算が階層化されている配列処理プリミティブの途中のレベルまでは同様の計算を行なうということにことに着目した実アドレス計算のコストの削減方法を提案する．

■ 近傍要素に関する推移的アドレス生成

同一局所配列[1]に存在する近傍要素を複数参照する場合，1 つの要素を参照した際に得られる情報をもとに，残りの近傍要素の実アドレスを offset 値を用いて計算する．

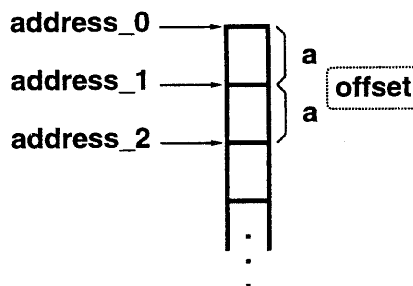


図 1: 近傍要素に関する推移的アドレス生成

この手法を用いることにより，低レベル (実アドレスに近いレベル) からの配列処理プリミティブの呼び出しが可能となり，高レベルの配列処理プリミティブの呼び出し回数を減らし実アドレス計算のコストを減らすことになる．

[1]局所配列とは，分散メモリ型並列計算機における配列の分散において PE 1 個あたりに割り当てられる配列領域を指す

## 3.2 細粒度通信を伴う配列参照コストの削減手法

通信の固定コストが大きなマシンでは，データをまとめて転送を行なう方が個別に何度も転送を行なうよりも効率的である．データフロー言語の実装においても，必要なデータのうち複数のデータがすでに定義されている場合は，Message Coalescing により通信におけるオーバヘッドの削減効果を得ることが予想される．

通信を伴う配列参照に関するこれまでの研究では，まずデータフロー言語では，配列に A-structure[3] を用いることが提案されている．しかしながら，この A-structure を用いると，

- 実行時に deadlock を起こす可能性あり
- I-structure との使い分けが困難
- 処理の逐次化を促すものであるが，場合によっては有効な並列性を損なってしまう恐れあり

などが考えられ，この構造の使用方法については慎重さを必要とする [3].

また他の言語，例えば HPF では

- Redundant Communication Elimination
- Message Coalescing
- Mesaage Aggregation

などの通信のコストを削減する手法が提案されている [4]. これらのうち今回は Message Coalescing と Redundant Communication Elimination に着目し，データフローモデルの持つ高い並列性を維持しつつこれらの手法を用いた通信オーバヘッドの削減手法を提案する．

### ■ Message Coalescing と Redundant Communication Elimination による通信オーバヘッドの削減手法

今回提案する手法は，まずデータフロー言語から生成された中間言語レベルにおいて Redundant Communication Elimination を行ない冗長な配列参照を除いた後，複数の配列参照をそれぞれ局所配列ごとに参照命令をまとめて被参照側の PE へ参照リクエストを送信する．被参照側 PE では，その参照のリクエストを受け取った時点で既に生成されている要素については一度にまとめて転送し，それ以外の要素についてはそれぞれ I-structure による読み出し処理が行なわれるというものである．

この手法を用いることにより，次に挙げるような利点が挙げられる．

- Redundant Communication Elimination により冗長な配列参照をカットできる．
- 各配列要素では I-structure を用いることにより高い並列性を維持できる．
- A-structure のように実行時の deadlock が生じることがない．
- Message Coalescing により通信の固定コストを削減できる．

ただしこの手法では，配列要素の添字はループの index 変数の 1 次式で表されているものとする．以下，処理の概要を示す．

1. (静的解析) ソース言語レベルにおいて同一イタレーション内での同一要素への複数の参照命令が存在する (重複参照) ことが添字を解析できた喪のにおいて，これらの重複参照を中間言語レベルにおいて 1 つの参照命令に絞る．(Redundant Communication Elimination)
2. (参照側 PE) イタレーション内の配列参照のうち，同一局所配列に存在する要素への参照を局所配列ごとに分けてまとめる．
3. (参照側 PE) アドレス生成の基準となる 1 つの要素の実アドレスをと，他の要素参照のための offset 値をそれぞれ計算する．
4. (参照側 PE) 2. および 3. で生成された情報を要素要求 (request) メッセージに追加して被参照側 PE へ各局所配列ごとに送信する．

5. (被参照側 PE) 4. で送信されたメッセージを受信する.
6. (被参照側 PE) 各被参照要素が既に定義がなされている要素値はまとめて reply し，未定義な要素については，I-structure 読みだしと同様に読み出し要求は suspend され 継続スレッドは pending queue に保持される.
7. (参照側 PE) 被参照側 PE からの reply のメッセージを受信すると，継続スレッドが ready スレッドとなり やがてスケジュールされ実行される. 実行時には reply されてきたメッセージを解釈し，返されたデータごとにそれぞれの継続スレッドに対して trigger をかける.

# 4 評価

今回，例題として画像処理アルゴリズムの 1 つである GNC のプログラムを使用する. このプログラムには 4 近傍とその中心の要素の計 5 個，あるいは 8 近傍とその中心の要素の計 9 個の要素を同じイタレーションの中で参照するコードが含まれているプログラムである. 今回の評価では，分散メモリ型商用並列計算機 AP1000 を使用し，画像 (配列) の大きさとしては 64 × 64 個であり，その配列は各 PE に対してブロック分散されている. また使用した PE 台数は 4 × 4 台としている. 以下に，naive なコード，および前章で提案したそれぞれの手法を用いたコードの実行時間を示す.

A. 画像 (配列) サイズ : 64 × 64

PE 台数 : 4 × 4 台

1. 4 近傍とその中心要素の計 5 個の要素参照パターン

| | Time(sec.) | Efficiency(%) |
|---|---|---|
| naive | 4.57 | — |
| addressing | 3.99 | 12.6 |
| fetch blocking | 4.08 | 10.6 |
| addressing + fetch blocking | 4.04 | 11.4 |

2. 8 近傍とその中心要素の計 9 個の要素参照パターン

| | Time(sec.) | Efficiency(%) |
|---|---|---|
| naive | 4.57 | — |
| addressing | 4.28 | 15.5 |
| fetch blocking | 4.24 | 16.4 |
| addressing+ fetch blocking | 4.18 | 17.6 |

B. 画像 (配列) サイズ : 64 × 64

PE 台数 : 8 × 8 台

1. 4 近傍とその中心要素の計 5 個の要素参照パターン

| | Time(sec.) | Efficiency(%) |
|---|---|---|
| naive | 2.13 | — |
| addressing | 2.03 | 5.0 |
| fetch blocking | 2.05 | 3.7 |
| addressing + fetch blocking | 2.04 | 4.3 |

2. 8 近傍とその中心要素の計 9 個の要素参照パターン

| | Time(sec.) | Efficiency(%) |
|---|---|---|
| naive | 2.31 | — |
| addressing | 2.10 | 8.9 |
| fetch blocking | 2.09 | 9.5 |
| addressing+ fetch blocking | 2.07 | 10.1 |

これらの結果より，以下に挙げるようなことがわかった.

1. A.2 のように配列参照数が多くかつ PE 間通信を伴わなずにローカルメモリに存在する要素を参照する割合が多い場合，アドレス計算コストの削減手法による効果が現われ，特に A.2 では全体の実行時間の約 15.5%の短縮となっている.

2. A.1 のように，配列参照のうち PE 間通信を伴わないローカルメモリに存在する要素を参照する割合が少ない場合，アドレス計算コストの削減を行なわずに配列参照をまとめて行なう手法を用いることにより，A.1 では全体の実行時間の約 16.4%の短縮となっている.

3. A.1 や B.1 のような参照パターンでは，参照されるブロック分散の性質上参照命令をまとめる可能性が全くなく，参照命令をまとめる手続きルーチンの処理の分だけ naive なコードの実行に比べて逆に効率が落ちている.

4. B. では A. の場合に比べて局所配列の大きさが小さくなるため，PE 間通信を伴ったリモートな配列参照の割合が高く，かつ参照命令のうち参照される要素が生成される前に参照する割合が高くなっていたため， 3. と同じ原因で効率向上の幅が A. の場合に比べて小さくなっている.

5. いずれのパターンにしても，配列参照をまとめる場合には各参照がローカルメモリに対するものか，あるいは PE 間通信を用いたものかに関わらず，配列参照を先行してまとめて行なうことにより，naive なコードのように各配列参照命令ごとに行なっていた場合に比べて実行時間の短縮が見られる.

これらのことから，ローカルメモリに存在する要素への参照アクセスはアドレス計算コストの削減手法を用いるのが効果的であり，また 局所配列 1 つあたりに含まれている要素数が多い方が両手法による効果が得られやすいという結論が得られた.

# 5 検討事項

■ イタレーションのブロック化

同一 PE 上で実行される複数のイタレーションをまとめることにより，さらに広い範囲での Message Coallescing を行なうことができアクセス単位を大きくすることが可能であると考えられる，また，Redundant Communication Elimination の適用範囲も拡張されれさらなる冗長アクセスの削減が考えられ，頻繁なスレッド切替えなどの実行制御のコストを減らすこともできる．ただし，イタレーションをまとめることによる有効な並列性が損なわれる可能性も考えられため，今後どの程度の大きさまでまとめる事ができるのかについてのさらなる検討が必要である.

# 6 おわりに

本稿では，細粒度配列参照アクセスにおけるさまざなオーバヘッドのうち，近傍要素に関する推移的アドレス生成による実アドレス計算のコストや配列参照における Message Coalescing と Redundant Communication Elimination を用いたデータ転送による細粒度配列参照のオーバヘッドの削減についての手法の提案を行なった．さらに，これらの手法を例題に用いて評価を行なった結果，全体の実行時間のうち最大約 17%の実行時間の短縮が実現され，データフロー言語の配列参照の効率化の実現が可能であることが確認できた.

# 参考文献


[1] 永井，稲永，日下部，雨宮,“AP1000 上のデータフロー細粒度並列構造データの実現”,PCW'95 Japan 予稿集,P1-B,March 1995

[2] K.E. Schauser, and S.C.Goldstein,“How Much Non-strictness do lenient Programs Require ?”,Proc. Functional Programming Languages and Computer Archtecture, 1995

[3] Seth Copen Goldstein,“The Implementation of Threaded Abstract Machine”,Department of Electrical Engineering and Computer Science Commputer Science Division, University of Calfornia,Berkley,May 1994

[4] P.Banajee,J.A.Chandy,M.Gupta,J.G.Holm, A.Lain,D.J.Palermo,S.Ramaswamy,E.Su, “The PARADIGM Compiler for Distributed -Memory Message Passing Multiprocessors”, In the First International Workshop on Parallel Processing, Bangalore,India, December 1994