計算機アーキテクチャ　147－35
ハイパフォーマンスコンピューティング　89－35
（2002．3．8）

# 分散メモリシステム上でのマクロデータフロー処理の実現

上田哲平†　本多弘樹†　吉瀬謙二†　弓場敏嗣†

並列性表現として実行開始条件用いた粗粒度処理手法では，粗粒度タスクを動的にプロセッサに割当てることによりマクロデータフロー処理を共有メモリシステム上で実現している．このマクロデータフロー処理手法を分散メモリシステム上に実現するには，タスク間での明示的なデータ通信のために，実行時に決定される変数の定義・使用関係に基づいて送信元・送信先を決定するための機能が必要である．本論文では，従来の到達する定義の概念を拡張することにより，マクロタスク間データ通信の送信元・送信先を実行時に決定する方式として「データ到達条件」を提案し，これを用いて分散メモリシステム上でマクロデータロー処理を実現するための方法を示す．

# Implementation of Macro-Data-Flow on Distributed Memory System

TEPPEI UEDA†, HIROKI HONDA†, KENJI KISE† and TOSITSUGU YUBA†

The coarse grain task parallel processing scheme using "execution start condition", which represents the parallelisms among coarse grain tasks, realizes a macro-dataflow processing by dynamic-assignment of tasks to processors on a shared memory system. An implementation of the macro-dataflow processing on a distributed memory system requires a new function to make a sender-receiver pair of the explicit data transfer between tasks based on the use-definition chain determined at run-time. This paper proposes "data reaching conditions", a method to make a sender-receiver pair of a data transfer by extending the concept of the conventional reaching definition, and its use for an implementation of macro-dataflow on a distributed memory system.

## 1. はじめに

マクロデータフロー処理 [1]は，ループや基本ブロックなどのマクロタスク(粗粒度タスク)間の並列性をコンパイル時に実行開始条件として求め，実行開始条件が成立したマクロタスクを動的に順次プロセッサに割当てることで並列実行を進めるものである．

これまでのマクロデータフロー処理では共有メモリマルチプロセッサシステムを対象としており，実行開始条件が成立したマクロタスクで使用する変数の値は共有メモリ上に存在することを前提とし，明示的なデータの通信は必要なかった．

一方，分散メモリマルチプロセッサシステム上でマクロデータフロー処理を実現するには，異なるプロセッサに割当てられたマクロタスク間のデータ授受をプロセッサ間通信により明示的に行わなければならない．このためには，どのマクロタスク間でどの変数に対してのデータの授受が必要となるかを実行時に判断しなければならないが，これまでのマクロデータフロー処理ではそのための方法がなかった．

これに対し本稿では，マクロタスク間でのデータ授受の関係をコンパイル時に「データ到達条件」として求め，実行時にこの条件を検査することによりデータ授受の送信元・送信先の関係を決定する方法を提案する(3 章)．またデータ到達条件の概念により，マクロデータフロー処理におけるプリロードを分類整理し，新たなプリロードの方法「投機的プリロード」が可能となることを示す(4 章)．さらに分散メモリシステム上でのデータ到達条件を用いたマクロデータフロー処理の実装方法を示し(5 章)，ベンチマークプログラムを用いた予備評価を行う(6 章)．

## 2. マクロデータフロー処理と実行開始条件

ここでは次章以降の記述に必要となるマクロデータフロー処理の基本的事項 [1-3]について概観する．

マクロデータフロー処理はプログラムをマクロタスクの集合としてとらえ，各マクロタスクをプロセッサへの割当て単位として並列処理するものである．

マクロタスクはプログラム中の一部分で，その部分の実行を開始する文がその部分の先頭の行であるものとし，その処理はノンプリエンプティブに行われる．

コンパイル時には，プログラムをマクロタスクに分割し，マクロタスク間の制御フローとデータ依存をマクロフローグラフで表現する．マクロフローグラフの例を図 1 に示す．方形で表すノードがマクロタスク，横の数字が

† 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems
The University of Electro-Communications

マクロタスク番号，円形が分岐，破線で表すエッジが制御フロー，実線で表すエッジがデータ依存を表している．マクロタスクへの分割にあたっては，制御フローが非循環となるように分割する．

マクロタスクの実行開始条件とは，あるマクロタスクの実行開始が可能となるための条件で，他のマクロタスクの実行状況を項とした論理式で表現したものである．この論理式は<マクロタスク終了>と<分岐方向決定>の二種類の原子条件および論理演算子∨(論理和)と∧(論理積)で構成される．マクロタスク MTa の終了条件は，MTa が終了した際に真となる条件で，a と表記する．マクロタスク MTb から MTc への分岐による分岐方向決定条件は，この分岐が確定した際に真となる条件で，b-c と表記する．

マクロタスク MTi の実行開始条件は式(1)のように定義される．

(MTi の実行確定条件)∧(MTi のデータアクセス可能条件)　(1)

式(1)の MTi の実行確定条件は，MTi を実行することが確定するための条件で，MTi が制御依存 [4]するマクロタスクから MTi が支配するマクロタスクへの分岐による分岐方向決定条件の論理和で構成される．

式(1)の第二項は，MTi で変数のアクセスが可能となる条件で，MTi がデータ依存する全マクロタスクの集合 DDM のそれぞれのマクロタスクに対するデータアクセス可能条件の論理積で構成される．MTj∈DDM に対するデータアクセス可能条件は式(2)のように定義される．

(MTj の終了条件)　∨　(MTj の非実行確定条件)　(2)

式(2)の MTj の非実行確定条件は MTj が実行されないことが確定するための条件で，MTj が補制御依存 [2]するマクロタスクから MTj へ至らないパスへの分岐による分岐方向決定条件の論理和で構成される．

図１中の MT6 を例にとるとその実行開始条件は表 1 のとおりとなる．

並列実行に際しては，スケジューラが実行開始条件の成立したマクロタスクを各プロセッサに割当て戦略に従って順次割当てることで処理を進める．マクロタスクを実行する各プロセッサは割当てられたマクロタスクを実行しマクロタスクの終了と分岐方向決定をスケジューラへ通達する．

## 3. データ到達条件

### 3.1 マクロタスク間でのデータ授受

マクロデータフロー処理では，あるマクロタスク MTi の実行開始条件が成立した時点で，MTi のデータ依存マクロタスクのいずれもその実行が「終了している」か「実行されない」ことが確定している．よって共有メモリシステム上での実装では，たとえデータ依存するマクロタスクが異なるプロセッサで実行されたとしても，特にマクロタスク間のデータ授受を明示的に行う必要はなく， MTi で使用する変数の値は共有メモリ上に存在しているとして MTi の実行を開始できる．すなわち，同一変数に関して複数のデータ依存マクロタスクが存在しても，どのデータ依存マクロタスクとの間でデータ授受がおこなわれるのかは考慮する必要はない．

一方分散メモリシステムにおいては，データ依存関係にあるマクロタスクが異なるプロセッサで実行される際には，明示的なデータ通信が必要となる場合があり，そのためにはどのマクロタスク間でどの変数に関するデータ授受を行うかが明確でなくてはならない．

しかしながらあるマクロタスク MTi で変数 V の使用(MTi 外で定義された値の使用)があり，V への定義が複数

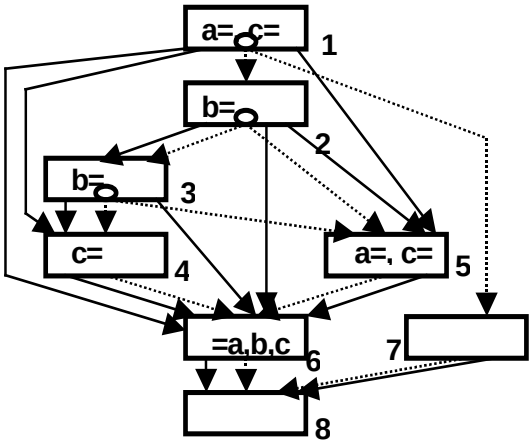


**図 1** マクロフローグラフの例
Fig 1 Example of macro-flow graph.

**表 1** 図１中の MT6 の実行開始条件
Table 1 Execution start condition for MT6 in Fig. 6.

| | | |
|---|---|---|
| 実行確定条件 | | 1-2 |
| データアクセス可能条件 | MT1 | 1 |
| | MT2 | 2∨1-7 |
| | MT3 | 3∨1-7∨2-5 |
| | MT4 | 4∨1-7∨2-5∨3-5 |
| | MT5 | 5∨1-7∨3-4 |
| 実行開始条件 | | 1-2∧1∧(2∨1-7)∧(3∨1-7∨2-5)∧(4∨1-7∨2-5∨3-5)∧(5∨1-7∨3-4) |

**表 2** 図１中 MT6 に関するデータ到達条件
Table 2 Data Reaching Conditions for MT6 in Fig. 6.

| 変数 | 定義を行うマクロタスク | データ到達条件 |
|---|---|---|
| a | MT1 | True∧(1-7∨3-4) |
| | MT5 | (2-5∨3-5)∧True |
| b | MT2 | 1-2∧(1-7∨2-5) |
| | MT3 | 2-3∧True |
| c | MT4 | 3-4∧True |
| | MT5 | (2-5∨3-5)∧True |

のマクロタスクで行われうる際，そのうちのどのマクロタスクで定義された V の値を MTi で使用すべきなのかは一般に実行時にならなければ決定できない．よって，どのマクロタスク間でどの変数に関する通信を行うかの判断も実行時に行うこととなるが，実行開始条件だけではそのための情報は不十分で，通信相手と変数を決定することができないという問題がある．

### 3.2 データ到達条件

データ到達条件とは，あるマクロタスク MTi(の先頭の文)に到達する [4]変数 V への定義を持つマクロタスク集合を{MT1...MTn}としたとき，MTi(の先頭の文)での V の値が，MTj∈{MT1...MTn}で定義した値となることが確定するための条件で，実行開始条件と同様に他のマクロタスクの実行状況を項とした論理式で表現する．

MTi での V に対する MTj∈{MT1...MTn}のデータ到達条件は， MTj から MTi へのパス上において，MTj での V への定義を kill する定義(ここでは確実な定義による kill のみを念頭に置く)を持つ全てのマクロタスクの集合を K としたとき，式(3)のように定義する．

(MTj の実行確定条件)∧
(K 中の全てのマクロタスクの非実行確定の条件) (3)

式(3)の第二項は，K の要素であるマクロタスクはひとつも実行されないことが確定する条件で，K 中の全てのマクロタスクの非実行確定条件の論理積で構成される．

並列実行中に MTi での V の使用に関するデータ到達条件が成立するのは{MT1...MTn}中のただひとつのマクロタスクとなる．

図 1 の MT6 を例にとると，MT6 で使用される各変数とこれに定義を行う各マクロタスクの組に対するデータ到達条件は表 2 のとおりとなる．

### 3.3 データ到達条件の求め方

データ到達条件は実行確定条件と非実行確定条件の求め方 [2]を利用して求めることができる．

実装においてはコンパイル時に，各マクロタスクに対してそのマクロタスクで使用する全ての変数とそのマクロタスクに到達する定義を持つマクロタスクの組に対してデータ到達条件を求める．

### 3.4 データ到達条件を用いた通信管理

分散メモリシステム上でのスケジューラは，実行開始条件の検査とマクロタスクのプロセッサへの割当て・実行指示に加えて，データ到達条件の検査により必要に応じてデータ通信の指示をプロセッサに与える．具体的にはあるマクロタスク MTi をプロセッサ Pi に割当てる際，データ到達条件の検査により，MTi で使用する変数の値の定義を行ったマクロタスクを求め，それらのマクロタスクが Pi とは異なるプロセッサに割当てられている場合には，それぞれのプロセッサにデータ通信の指示を与える．

**表 3** 実行開始条件の状況
Table 3 Status of execution start condition.

| | | データアクセス可能条件 | | |
|---|---|---|---|---|
| | | 全 MT 成立 | 一部 MT 成立 | 全 MT 未成立 |
| 実行確定条件 | 成立 | a | b | c |
| | 未成立 | d | e | f |

**表 4** データ到達条件とデータアクセス可能条件の状況
Table 4 Status of data reaching condition and data access condition.

| | | データアクセス可能条件 | | |
|---|---|---|---|---|
| | | 成立 (終了) | 成立 (非実行確定) | 未成立 |
| データ到達条件 | 成立 | A | n.a. | B |
| | 未成立 | C | D | E |

## 4. データ到達条件のプリロードへの応用

### 4.1 マクロデータフロー処理でのプリロード

マクロタスク処理と通信とをオーバーラップして実行できる環境では，マクロタスク MTi の実行開始以前に，他のマクロタスクの処理と並行して MTi で必要なデータの通信を行うプリロードによりデータ通信によるレイテンシを隠蔽することが望ましい．

以降の議論では，マクロタスク MTi で使用する変数 V への定義をもつマクロタスク MTj から MTi へのプリロードを行う上で次の条件を仮定する．

(1) マクロタスクの投機実行（実行開始条件が成立していないマクロタスクの実行）はしない．
(2) MTi を割当てるプロセッサが決定している．
(3) MTj が終了している．

また，スケジューラがマクロタスクをプロセッサに割当てることとプロセッサにそのマクロタスクの実行の指示を与えることは個別の動作であるとする．

### 4.2 プロセッサ割当てとプリロードの分類

ある時点での，マクロタスク MTi の実行開始条件の状況は表 3 のように場合分けできる．また，MTi で使用する変数 V への到達する定義を持つマクロタスク MTj の状況は表 4 のように場合分けできる．

表 3 と表 4 の場合分けをもとに，マクロタスク MTi のプロセッサ割当ての方法を「割当て時点に MTi が表 3 のどの状況にあるか」で分類し，それぞれの場合に MTi での V の値と MTj の関係および MTj で定義された値のプリロードの可否がどのようになるかをまとめると次のような組み合わせが考えられる．

(1) 通常割当て：a の状況での割当て．到達する定義を持つマクロタスクの状況はそれぞれ A，C のまま，

D のまま，のいずれかであることが確定しおり，以降変化しない．プリロードの対象となるのは A の状況のマクロタスクだけである．

(2) データ依存待ち割当て：b，c の状況での割当て．到達する定義を持つマクロタスクの状況は A～E のいずれかである．どのマクロタスクをプリロードの対象とするかは次の二つの方法が考えられる．

(ア) 割当て時点で A の状況のもの．もしくは割当て後 A の状況に推移したもの．

(イ) 割当て時点で A か C の状況のもの．もしくは割当て後 A か C の状況に推移したもの．

(3) 制御依存待ち割当て：d の状況での割当て．到達する定義を持つマクロタスクの状況とプリロードの対象は(1)と同じ．

(4) 制御・データ依存待ち割当て：e，f の状況での割当て．到達する定義を持つマクロタスクの状況とプリロードの対象は(2)と同じ．

上記の(2)(3)(4)では，実行開始条件が未成立のマクロタスクに対しても割当てるプロセッサを決めているもので，今後これを「プリアサイン」と呼ぶこととする．

上記(1)は従来の共有メモリシステム上でのマクロデータフロー処理でも実現されており，この場合にはプリロードの対象となるマクロタスクの状況が A であるか否かを判断する必要はない．

(2)～(4)は (1)より積極的なプリロードで，データ到着条件を用いることにより，プリアサインを前提として可能となるものである．

なお，(2)の(イ)，(3)，(4)の(ア)と(イ)でのプリロードでは，MTi が実行されなかったり，MTj のデータ到達条件が不成立になったりして，プリロードした変数の値が不要となってしまう場合がある．このようなプリロードを「投機的プリロード」と呼ぶこととする．

## 5． 分散メモリシステム上での実装

### 5.1 実装の概要

データ到達条件を用いたマクロデータフロー処理を各ノードに 2 つのプロセッサを持つ SMP クラスタ上(表 5)に，MPI と p-thread を用いて実装した．

ノードのうちひとつをスケジューラノード(SN)，その他のノードをマクロタスク実行ノード(EN)とする．各ノードでは一つのプロセスを起動し，MPI を用いたノード間通信を行う．また各ノードに二つのスレッドを生成し処理と通信を個別のスレッドで実行することにより，これらの独立な実行を実現する．このときスレッド間の通信には共有変数を用いる．

SN では，全ノードの並列実行を統制するグローバルスケジューラコードと他ノードとの通信を行う通信コードを個別のスレッドとして実行する．EN では，通信とマクロタスク処理を統制するローカルスケジューラコードコードと他ノードとの通信を行う通信コードをまとめて一つのスレッドとして実行し，マクロタスクの処理を行う MT 処理コードをひとつのスレッドとして実行する．図 2

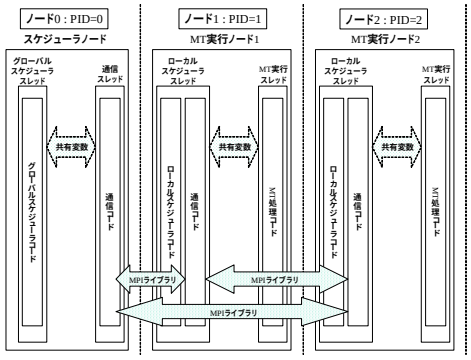


**図 2** 分散メモリシステム上での実装例

Fig. 2 Example of implementation.

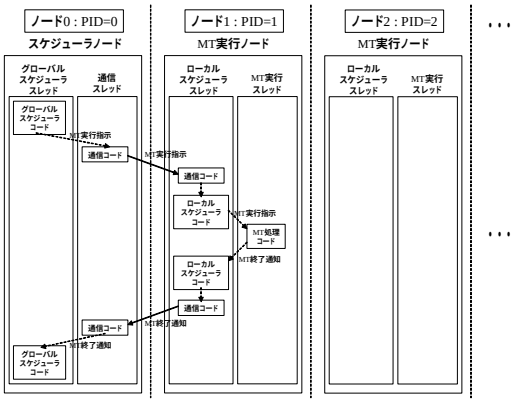


**図 3** マクロタスクの割当て実行

Fig.3 Macrotask assigning and execution.

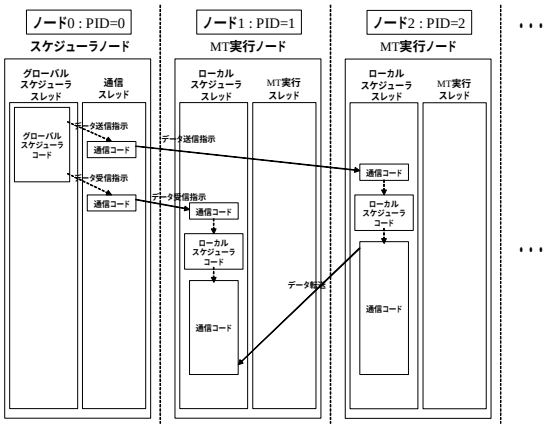


**図 4** マクロタスク間データ通信

Fig. 4 Data transfer between macrotasks..

に実装の概要図を示す.

ノード間で行われる通信は(1)SN から EN へのマクロタスク実行指示，(2)SN から EN へのデータの送信・受信指示，(3)EN から SN へのマクロタスクの実行に伴う<マクロタスク終了>と<分岐方向決定>の通知，(4)EN 間のデータの送信・受信の 4 種類である．図 3 にマクロタスクの割当てと実行の過程を，図 4 にデータ転送の過程を示す.

### 5.2 スケジューラノード

グローバルスケジューラコードの機能を次に示す．(2)と(3)におけるマクロタスクのプロセッサへの割当ては CP/DT/MISF に基づいて決定される．またプリロードを行わない場合には(3)は実行しない.

(1) <マクロタスク終了>と<分岐方向決定>の通知に伴う実行開始条件とデータ到達条件の更新
(2) 実行開始条件が成立したマクロタスクのプロセッサへの割当て決定．マクロタスク実行指示とデータ送信・受信指示の発行
(3) プリアサイン可能なマクロタスクのプロセッサへの割当て決定．データ送信・受信指示の発行

通信コードの機能は次の通りである.

(1) EN からの<マクロタスク終了>，<分岐方向決定>の受信
(2) EN へマクロタスク実行指示とデータ送信・受信指示の送信

### 5.3 マクロタスク実行ノード

ローカルスケジューラコードの役割は，(1)使用するデータがノード内に揃ったとき MT 処理コードにマクロタスクの実行を指示，(2)通信コードがアイドル状態のときノード間通信を指示することである．これらの処理は通信コードと並列に実行する必要がないため，スレッド生成のコストを削減するためにローカルスケジューラコードと通信コードを同一のスレッドで実行する．ローカルスケジューラコードと通信コードの機能を以下に示す.

(1) SN からのマクロタスク実行指示とデータ送信・受信指示の受信
(2) データが揃ったマクロタスクがあれば，MT 処理コードに実行を指示
(3) SN へ<マクロタスク終了>，<分岐方向決定>を送信
(4) 他の EN とのデータ送信・受信

MT 処理コードの機能は次のとおりである.

(1) ローカルスケジューラコードからの指示の待ち受け
(2) マクロタスクの実行
(3) (2)に伴う<マクロタスク終了>と<分岐方向決定>の発行

図 5 にローカルスケジューラコードと通信コードの，図 6 に MT 処理コードの基本的な構成を示す.

```
void * local_scheduler (void) {

  /* スレッド間共有変数とロックの宣言 */
  extern struct queue MT_order_Q; extern pthread_mutex_t MT_order_Qlock;
  extern struct queue DT_order_Q; extern pthread_mutex_t DT_order_Qlock;
  extern struct queue MT_finish_Q; extern pthread_mutex_t MT_finish_Qlock;
  extern struct queue BR_finish_Q; extern pthread_mutex_t BR_finish_Qlock;

  extern struct queue MT_Q;       //MTキュー
  extern struct queue DT_Q;       //データ転送キュー

  while (MT集合の処理が終了するまで) {

    // スケジューラノードからの指示の受信
    if ( MPI_Probe(scheduler) == 1) {
      MPI_Recv(); // スケジューラからの指示の受信
      if (指示がマクロタスク実行指示だった場合)
        MT_Qにマクロタスク実行指示を挿入;
      else
        DT_Qにデータ受信･送信指示を挿入;
    }

    else {

      // MT処理コードへのMT実行通知
      MT_Qから指示の取り出し;
      if (マクロタスクが実行開始可能) {
        pthread_mutex_lock(&MT_order_Qlock);
        MT_order_Qへマクロタスクを挿入;
        pthread_mutex_lock(&MT_order_Qlock);
      }

      // スケジューラノードへの<マクロタスク終了>の通知
      if (終了マクロタスク通知キューが空でない場合) {
        pthread_mutex_lock(&MT_finish_Qlock);
        MT_finish_Qから終了マクロタスクの取り出し;
        pthread_mutex_lock(&MT_finish_Qlock);
        MPI_Send(); // 終了マクロタスク通知の送信
      }

      // スケジューラノードへの<分岐方向決定>の通知
      else if (分岐方向通知キューが空でない場合) {
        pthread_mutex_lock(&BR_finish_Qlock);
        BR_finish_Qから分岐方向の取り出し;
        pthread_mutex_lock(&BR_finish_Qlock);
        MPI_Send(); // 分岐方向通知の送信
      }

      // 他のマクロタスク実行ノードとのデータ送信･受信
      else if (データ転送指示キューが空でない場合) {
        DT_Qから指示の取り出し;
        if (指示が送信だった場合)
          MPI_Send();
        else
          MPI_Recv();
      }
    }
  }
}
```

通信コード

図 5　ローカルスケジューラコードと通信コード

Fig.5 Local Scheduler code and Communication code

```
void * execMT (void) {

  /* スレッド間共有変数とロックの宣言 */
  extern struct queue MT_order_Q; extern pthread_mutex_t MT_order_Qlock;
  extern struct queue DT_order_Q; extern pthread_mutex_t DT_order_Qlock;
  extern struct queue MT_finish_Q; extern pthread_mutex_t MT_finish_Qlock;
  extern struct queue BR_finish_Q; extern pthread_mutex_t BR_finish_Qlock;

  while (マクロタスク集合の処理が終了するまで) {

    // マクロタスク実行指示の取り出し
    if (マクロタスク実行指示キューが空でない) {
      pthread_mutex_lock(&MT_order_Qlock);
      MT_num = remove(&MT_order_Q);
      pthread_mutex_lock(&MT_order_Qlock);

      // MT1の処理
      if (MT_num == 1) {
        ･･･
      }

      // MT2の処理
      else if (MT_num == 2) {
        ･･･
          // <分岐方向決定>の通知
          pthread_mutex_lock(&BR_finish_Qlock);
          BR_finish_Qへ分岐方向を挿入;
          pthread_mutex_lock(&BR_finish_Qlock);
        ･･･
      }

            .
            .
            .

      // MTnの処理
      else if (MT_num == n) {
        ･･･
      }

      // <終了マクロタスク>の通知
      pthread_mutex_lock(&MT_finish_Qlock);
      MT_finish_Qへ終了マクロタスクを挿入;
      pthread_mutex_lock(&MT_finish_Qlock);

    }
  }
}
```

図 6　MT 処理コード

Fig.6 Macrotask code

## 6. 評価

評価に用いたプログラムは tomcatv（SpecCFP95）と swim（SpecCFP2000）で，表 5 に示す 2 種類のシステムに対して 4.2 節での(2)-(ア)のプリロードを行わない場合と行う場合の実行時間を測定した．tomcatv，swim とも主ループの内側でマクロデータフロー処理を行い，並列コードの生成は人手で行った．また tomcatv のデータサイズは(257×257)，swim のデータサイズはシステム A では(505×505)，システム B では(808×808)とした．

それぞれの測定結果を表 6 と表 7 に示す．

システム A における tomcatv で並列化効果を得られなかった原因は，マクロタスクの粒度に対して Ethernet を介したノード間通信のオーバーヘッドが大きいためと考えられる．しかしながら swim とシステム B における tomcatv では，十分な並列処理効果が得られた．またプリロードに関しては，ノード間でのデータ転送の粒度が小さい tomcatv では性能向上を得られなかったが，swim をシステム A で実行する場合には性能向上を得られた．

## 7. おわりに

本稿では実行開始条件を用いたマクロデータフロー処理を分散メモリシステム上で実現する際に必要となる「データ到達条件」を提案した．データ到達条件を実行時に検査することにより，マクロタスクに到達する定義のうちどの定義による値を使用するかを実行時に動的に決定し，マクロタスク間のデータ授受の関係を求めることができる．またデータ到達条件を用いることにより新たに可能となる積極的なプリロード方式「投機的プリロード」を示した． さらに，データ到達条件を用いてマクロタスク間でのデータ授受を決定する方法を用いて分散メモリシステム上にマクロデータフロー処理を実現する一例を示し評価を行った．

現在本稿で述べた実装に加え，MPI と OpenMP を用いた実装，ならびに本方式によるコードを逐次プログラムから生成するコンパイラの開発も進めている．

本稿で可能性を示したプリアサインとプリロードの方式に対しての実装とその有効性に関する研究は今後の課題である．また，マクロデータフロー処理をソフトウェア分散共有メモリ上で実装する際の一貫性制御コード生成にデータ到達条件を用いることをはじめ，データ到達条件の並列プログラム解析への応用も今後の課題としたい．

**謝辞** 本研究の一部は文部科学省科学研究費(基盤C,2,12680336)によって行われた．

**表 5** システム構成
Fig. 5 System configuration.

| | PC クラスタ A | PC クラスタ B |
|---|---|---|
| OS | Red Hat Linux 6.2<br>Kernel 2.2.19 | Red Hat Linux 7.1,<br>SCore Version 4.2.1 |
| プロセッサ | PentiumIII800[MHz] x 2 | PentiumIII866[MHz] x 2 |
| メモリ | 128M | 1GB |
| NIC | Ether-100base | Myrinet-2000 |
| MPI | mpich1.2.1 | mpich1.2.0 |

**表 6** 実行時間（tomcatv）[sec]
Fig.6 Execution time (tomcatv) [sec].

| システム | ノード数 | 実行時間 | | 実行時間比 | |
|---|---|---|---|---|---|
| | | PL なし | PL あり | PL なし | PL あり |
| A (257x257) | 1 | 100.2 | 100.2 | 1.00 | 1.00 |
| | 2 | 65.8 | 66.0 | 0.66 | 0.66 |
| | 4 | 104.0 | 103.6 | 1.04 | 1.03 |
| B (257x257) | 1 | 49.3 | 49.3 | 1.00 | 1.00 |
| | 2 | 30.2 | 30.1 | 0.61 | 0.61 |
| | 4 | 15.9 | 16.4 | 0.32 | 0.33 |

**表 7** 実行時間（swim）[sec]
Fig.7 Execution time (swim) [sec].

| システム | ノード数 | 実行時間 | | 実行時間比 | |
|---|---|---|---|---|---|
| | | PL なし | PL あり | PL なし | PL あり |
| A (505x505) | 1 | 75.8 | 75.8 | 1.00 | 1.00 |
| | 2 | 55.1 | 51.6 | 0.73 | 0.68 |
| | 4 | 41.4 | 38.5 | 0.54 | 0.52 |
| B (808x808) | 1 | 114.5 | 114.5 | 1.00 | 1.00 |
| | 2 | 61.1 | 61.3 | 0.53 | 0.54 |
| | 4 | 31.9 | 31.3 | 0.27 | 0.27 |

## 参考文献

1) 笠原博徳，小幡元樹，石坂一久：共有メモリマルチプロセッサシステム上での粗粒度タスク並列処理，情報処理学会論文誌, Vol.42, No.4, pp.910-920(2001).
2) 本多弘樹，岩田雅彦，笠原博徳，：Fortran プログラム粗粒度タスク間の並列性検出手法，電子情報通信学会論文誌, Vol.J73-D-1, No.12, pp.951-960 (1990).
3) 本多弘樹，合田憲人，岡本雅人，笠原博徳：Fortran プログラム粗粒度タスクの OSCAR における並列実行方式，電子情報通信学会論文誌，Vol.J75-D1，No.8, pp.526-535 ( 1992).
4) Ferrante, J.F., Ottenstein, K.J. and Warren, D.J.: The Program Dependence Graph and Its Use in Optimization, ACM Trans. Prog. Lang. and Sys., 9,3, pp.319-349, (1987).
5) Aho, A.V., Sethi, R. and Ullman, J.D.: Compilers – Princeiples, Techniques, and Tools, Addison-Wesley (1988).