

分散メモリシステム上での OpenMP によるマクロデータフロー処理

深 川 保[†] 吉 瀬 謙 二[†]
本 多 弘 樹[†] 弓 場 敏 嗣[†]

マクロデータフロー処理では並列性を実行開始条件で表現した粗粒度タスク集合を動的にプロセッサに割当てることにより粗粒度並列処理を行う。マクロデータフロー処理手法を分散メモリシステム上で実装するには、粗粒度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。これに対して、我々は従来の到達する定義の概念を拡張することにより、粗粒度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」を提案してきた。本稿では、これを用いた分散メモリシステム上でマクロデータロー処理を OpenMP により実現するための方法を示す。

Implementation of Macro-Data-Flow using OpenMP on Distributed Memory System

TAMOTSU FUKAGAWA,[†] KENJI KISE,[†] HIROKI HONDA[†]
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. We have proposed "data reaching conditions", a method to make a sender-receiver pair of a data transfer by extending the concept of the conventional reaching definition. This paper shows the method of macro-dataflow implementation using OpenMP with the data reaching conditions 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²⁾(もしくはそれと同等のもの³⁾)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てことで並列実行を進めるものである。

マクロデータフロー処理の共有メモリマルチプロセッサシステム上での実装では、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、異なるプロセッサに割当てられたマクロタスク間での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の実装では、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、変数への定義を行うマクロタスクが複数存在する状況において、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。

これに対し我々は、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、実行時には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してきた。

本稿ではこのデータ到達条件を用いた分散メモリシステム上でのマクロデータフロー処理を OpenMP により実装する方法を示し、ベンチマークプログラムを用いた予備評価を行う。

[†] 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

2. マクロデータフロー処理¹⁻⁴⁾

2.1 マクロタスク

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。

2.2 マクロタスクの実行開始条件

マクロデータフロー処理では、マクロタスク間の並列性(先行制約)を実行開始条件で表現する。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 MT_a の終了条件は、MT_a 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 MT_b から MT_c 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 MT_i の実行開始条件は式(1)のように定義される。

(MT_i の実行確定条件)

(MT_i のデータアクセス可能条件) (1)

式(1)の MT_i の実行確定条件は、MT_i を実行することが確定するための条件で、MT_i が制御依存するマクロタスクから MT_i 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、MT_i で変数のアクセスが可能となる条件で、MT_i がデータ依存する全マクロタスク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。MT_i がデータ依存

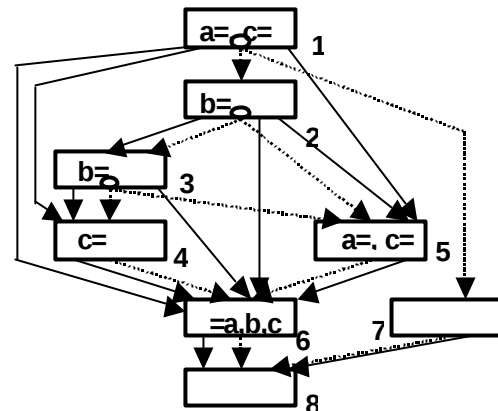


図1 マクロフローグラフの例
Fig 1 Example of macro-flow graph.

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

するマクロタスク MT_j 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。

(MT_j の終了条件)

(MT_j の非実行確定条件) (2)

式(2)の MT_j の非実行確定条件は MT_j が実行されないことが確定するための条件で、MT_j が補制御依存するマクロタスクから MT_j 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。

2.3 実行時スケジューラ

マクロデータフロー処理では、スケジューラがマクロタスク実行プロセッサに順次マクロタスクを割当ててことで処理を進める。スケジューラの実装には、集中型として特定のプロセッサで実行する方式

と各プロセッサに分散させて実行する方式がありうるが、ここでは集中型について説明する。

スケジューラのコードはコンパイル時に実行開始条件を元に生成される。

スケジューラは以下の動作を繰り返す。

- (1) 実行開始条件の検査：各マクロタスクから通達される終了および分岐方向決定の情報をもとに実行開始条件を検査し、条件が成立したマクロタスクをレディマクロタスクとする。
- (2) プロセッサへの割当て：所定の割当戦略に従ってレディマクロタスク中のどのマクロタスクをどのプロセッサに割当ててかを決定する。
- (3) 各プロセッサに実行すべきマクロタスク番号を通知する。

マクロタスクを実行する各プロセッサのコードには、全てのマクロタスクの実行コードに加えて、スケジューラからのマクロタスクの割当て・実行指示を受け取るコードと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コードが含まれる。

2.4 データ到達条件

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。

マクロタスク MT_i で変数 V の使用 (MT_i 外で定義された値の使用) があり、 V への定義が複数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 MT_i 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。

データ到達条件とは、あるマクロタスク MT_i (の先頭の文) に到達する θ 変数 V への定義を持つマクロタスク集合を SiV としたとき、 MT_i (の先頭の文) での V の値が、 $MT_j \in SiV$ 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。

MT_i での V に対する $MT_j \in SiV$ のデータ到達条件は、 MT_j から MT_i へのパス上において、 MT_j 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。

(MT_j の実行確定条件)

(K 中の全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 MT_i での V の使用に関するデータ到達条件が成立するのは集合 SiV 中のただひとつのマクロタスクとなる。

図 1 の MT_6 を例にとると、 MT_6 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。

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。

2.5 スケジューラにおける通信管理

分散メモリシステム上での実装におけるスケジューラは、実行開始条件の検査とマクロタスクの割当て・実行指示に加えて、データ到達条件の検査に基づきデータ通信の必要性を判定し、必要であればプロセッサにその指示を与える。

具体的には、マクロタスク MT_i の割当てをプロセッサ P_i に決定した後に、 MT_i で使用する変数 V ごとに通信の必要性判定と通信指示のための動作を以下のように行う。

- (1) データ到達条件の検査： V に関するデータ到達条件を評価し、条件が成立したマクロタスク MT_j とそれが割当てられたプロセッサ P_j を求める。
- (2) 通信の指示： MT_j が生成した V の値が既に P_i に存在する場合 (P_i と P_j が同一の場合、もしくは P_i で既に実行されたマクロタスクで使用するために V の値が P_i に通信済みの場合) にはデータ転送の指示は行わない。そうでない場合は、 P_j に対して MT_j で定義された V の値を P_i へ送信するよう指示し、さらに P_i に対しては MT_i の実行に先立ち P_j から V の値を受信するよう指示する。

3. OpenMP を用いた実装

3.1 実装の概要

対象としたシステムは各ノードに 2 つのプロセッ

サを搭載した SMP クラスタである(表 3)．この上でデータ到達条件を用いたマクロデータフロー処理を MPI と OpenMP を用いて実装した．

ダイナミックスケジューリングには集中スケジューラ方式を採用し，ノードのうち一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 OpenMP プログラムを起動する．各ノード間の通信には MPI を用いる．また，各ノードで parallel 指示文を用いることにより，マスタースレッドはスレッドのチームを生成する．チーム内のスレッドはマスタースレッドとスレーブスレッドの 2 つである．これらのスレッドで通信と処理を個別に実行することにより，MPI の関数をマスタースレッドのみから呼び出すように実現する．

SN では，マスタースレッドが他ノードとの通信を行う通信コードを実行し，スレーブスレッドが全ノードの並列実行を管理するグローバルスケジューラコードを実行する．一方 EN では，マスタースレッドが通信とマクロタスク処理を管理するローカ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まとめて実行し，スレーブスレッドがマクロタスクの処理を行う MT 処理コードを実行する．図 2 に実装の概要図を示す．

ノード間で行われる通信は次の 4 種類である．

- (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

3.2 スケジューラノード(SN)

SN のグローバルスケジューラコードの機能を次に示す．

- (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

SN の通信コードの機能を次に示す．

- (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

スレッドの生成は，parallel 指示文により並列リージョンを定義する．この並列リージョン内で生成されたスレッドの ID を実行時ライブラリ関数によ

表3 システム構成

Table.3 System configuration.

Processor	Pentium 866[MHz] x2
Memory	1GByte
NIC	Myrinet-2000
OS	Red Hat Linux 7.1, Score Version 4.2.1
MPI	mpich1.2.0
OpenMP	RWCP Omni OpenMP Compiler(ver1.4a)

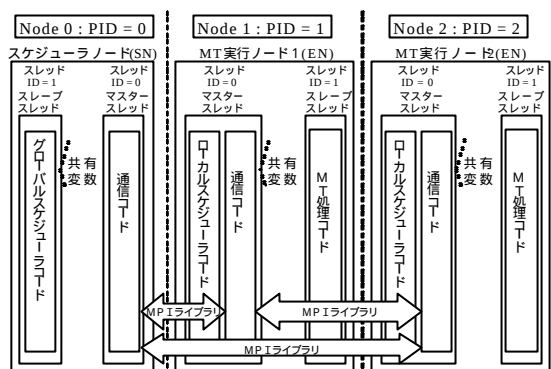


図2 分散メモリシステム上での実装例

Fig.2 Example of implementation.

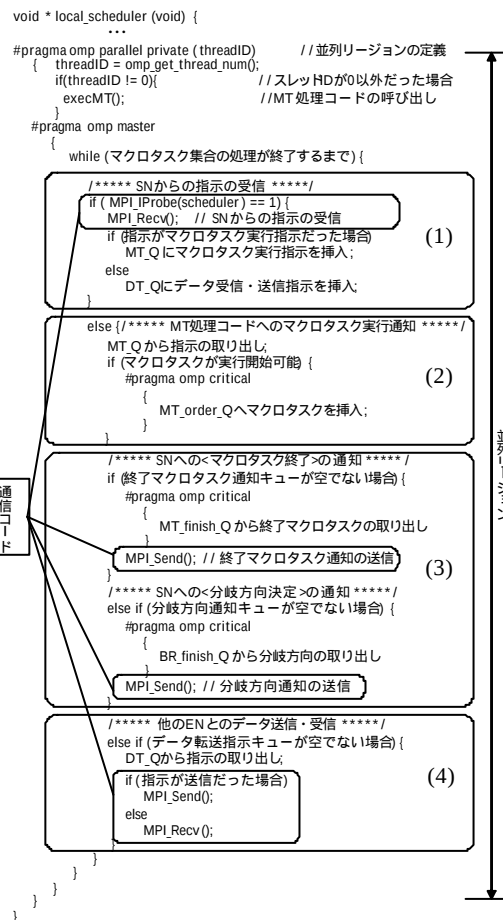


図3 ローカルスケジューラコードと通信コード

Fig.3 Local scheduler code and communication code

り取得し、その ID をもとに通信と処理を明示的に分ける．具体的には master 指示文を用いて、ID = 0 のマスタースレッドが通信コードを実行し、ID = 1 のスレーブスレッドがグローバルスケジューラコードを実行する．このとき、スレッド間の通信には共有変数を用いる．各スレッドがこの共有変数をアクセスする際は critical 指示文を用いて排他制御している．SN での共有変数は次の 4 種類のキューである．

マクロタスク実行指示キュー

グローバルスケジューラコードから通信コードへマクロタスクの実行指示を通知する際に使用．

データ転送指示キュー

グローバルスケジューラコードから通信コードへデータ転送指示を通知する際に使用．

終了マクロタスク通知キュー

通信コードからグローバルスケジューラコードへ EN から受け取った<マクロタスク終了>を通知する際に使用．

分岐方向通知キュー

通信コードからグローバルスケジューラコードへ EN から受け取った<分岐方向決定>を通知する際に使用．

3.3 マクロタスク実行ノード(EN)

EN のローカルスケジューラコードと通信コードの機能を次に示す．

- (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

EN の MT 処理コードの機能を次に示す．

- (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>の通知
- (4) (2)に伴う<分岐方向決定>の通知

EN でのスレッドの生成は SN と同様に parallel 指示文により並列リージョンを定義する．この並列リージョン内で、生成されたスレッドの ID を実行時ライブラリ関数により取得し、その ID をもとに通信と処理を明示的に分ける．具体的には master 指示文を用いて、ID = 0 のマスタースレッドがロー

```
void * execMT (void) {
    ...
    while (マクロタスク集合の処理が終了するまで) {
        ...
        if (マクロタスク実行指示キューが空でない) {
            /* **** マクロタスク実行指示の取り出し **** */
            #pragma omp critical
            {
                MT_order_Qからマクロタスク番号(MT_num)を取り出す;
            }
            /* 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 **** */
                #pragma omp critical
                {
                    BR_finish_Qへ分岐方向を挿入;
                }
                ...
            }
            .
            /* MTnの処理 */
            else if (MT_num == n) {
                ...
            }
        }
        /* **** <マクロタスク終了>の通知 **** */
        #pragma omp critical
        {
            MT_finish_Qへ終了マクロタスクを挿入;
        }
    }
}
```

図4 MT処理コード

Fig.4 Macrotask execution code

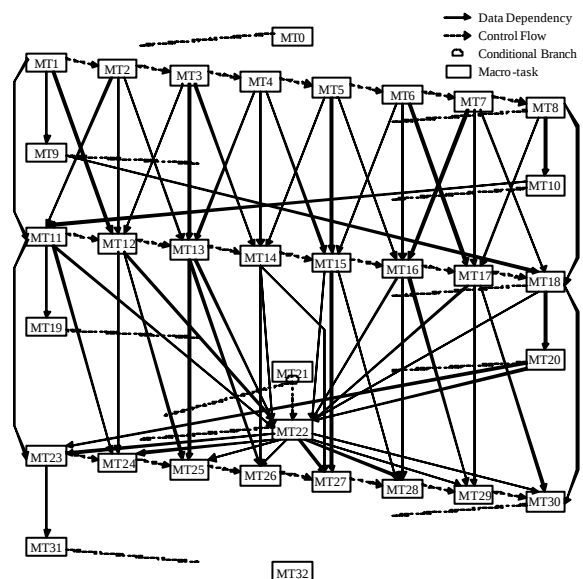


図5 swim(8 並列)のマクロフローグラフ

Fig.5 Macroflow graph of swim

カルスケジューラコードと通信コードを実行し、ID = 1 のスレーブスレッドが MT 処理コードを実行する．このとき、スレッド間の通信には共有変数を用いる．各スレッドがこの共有変数をアクセスする際は critical 指示文を用いて排他制御している．EN での共有変数は次の 3 種類のキューである．

マクロタスク実行指示キュー

ローカルスケジューラコードから MT 処理コードへマクロタスクの実行を通知する際に使用．

終了マクロタスク通知キュー

MT 処理コードからローカスケジューラコードへ<マクロタスク終了>を通知する際に使用．

分岐方向通知キュー

MT 処理コードからローカスケジューラコードへ<分岐方向決定>を通知する際に使用．

図 3 に EN におけるローカスケジューラコードと通信コードの基本的な構成を，図 4 に EN における MT 処理コードの基本的な構成を示す．

4. 評価

評価に用いたプログラムは SpecCFP95(データサイズ 505x505)の swim を C で書き直した比較的小規模のプログラムである．

並列化に際しては主ループの内側の処理をマクロタスクに分割し，マクロデータフロー処理を行うようにした．また，マクロタスク分割，並列性検出などの並列化は人手で行った．ベンチマークプログラムの主ループの内側を 8 並列で最適となるようにマクロタスク分割した場合のマクロフローグラフを図 5 に示す．マクロタスクの主要なものはループを分割または融合したものであり，マクロタスク間のデータ依存は配列変数によるものである．

このプログラムを表 3 のシステムにおいて，逐次の場合，EN 数が 2 の場合，4 の場合，および 8 の場合において並列実行し，その実行時間を測定した．測定結果を表 4 に示す．

表 4 より，2 並列の場合は逐次実行の 0.59 倍，4 並列の場合は逐次実行の 0.28 倍，8 並列の場合は逐次実行の 0.16 倍となり，ほぼリニアな結果が得られている．これにより，MPI と OpenMP を用いた実装において，ノード間でのデータ転送によるオーバーヘッドは大きくないことが分かり，十分な並列処理効果を確認することができた．

5. おわりに

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には，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．本稿では，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として OpenMP と MPI を用いる方法を示し評価を行った．現在本方式によるコードを逐次プログラムから生

表4 実行時間[sec]

Table.4 Execution time [sec].

Spec CFP95 swim		
EN 数	実行時間 [sec]	実行時間比
逐次	429	1
2	251	0.59
4	120	0.28
8	68	0.16

成するコンパイラの開発も進めている．

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．

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) 本多弘樹，上田哲平，深川保，弓場敏嗣：分散メモリシステム上でのマクロデータフロー処理のためのデータ到達条件, JSPP2002 論文集, pp.313-320 (2002).
- 5) 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).
- 6) Aho, A.V., Sethi, R. and Ullman, J.D.: Compilers – Principles, Techniques, and Tools, Addison-Wesley (1988).
- 7) Girkar, M. and Polychronopoulos, C.D.: Optimization of Data/control Conditions in Task Graphs, Proc. of the 4th Workshop on Languages and Compilers for Parallel Computing, pp.Z1-Z15, (1991).