計算機アーキテクチャ 125－2
（1997. 8. 20）

Fortran 階層型マクロデータフロー処理における データローカライゼーション

吉田明正[†1] 越塚健一[†2]
岡本雅巳[†3] 笠原博徳[†4]

本論文では、階層的に粗粒度並列処理を行なう階層型マクロデータフロー処理におけるデータローカライゼーション手法を提案する。本手法では、粗粒度並列処理される各階層において、ループ整合分割法を用いて処理とデータを分割し、パーシャルスタティックタスク割当を用いたダイナミックスケジューリング方式により、データ転送の多い粗粒度タスク集合を同一プロセッサに割り当て、粗粒度タスク間データ転送にローカルメモリを有効利用してデータ転送オーバヘッドを軽減する。マルチプロセッサシステム OSCAR 上で行った性能評価の結果、本データローカライゼーションを伴う階層型マクロデータフロー処理では、データローカライゼーションを用いない場合に比べて処理時間が10%～20%短縮されることが確かめられた。

Data-Localization for Fortran Hierarchical Macro-Dataflow Processing

AKIMASA YOSHIDA,[†1] KEN'ICHI KOSHIZUKA,[†2] MASAMI OKAMOTO[†3] and HIRONORI KASAHARA [†4]

This paper proposes a data-localization scheme for Fortran hierarchical macro-dataflow processing, which hierarchically exploits coarse-grain parallelism. The proposed data-localization scheme consists of three parts: (1)loop-aligned decomposition, which decomposes multiple loops having data dependences into data-localization-groups, (2)generation of dynamic scheduling routine with partial static task assignment, which assigns macrotasks inside data-localization-group to the same processor, (3)generation of data transfer code via local memory inside data-localization-group. Performance evaluations on a multiprocessor system OSCAR show that hierarchical macro-dataflow processing with data-localization can reduce execution time by 10%–20% compared with hierarchical macro-dataflow processing without data-localization.

1. はじめに

マルチプロセッサシステム上での自動並列化コンパイラを用いた並列処理では従来よりループ並列化[1)2)]が行なわれており、イリノイ大学の Polaris[3)]やスタンフォード大学の SUIF[4)]などのような先端の並列化コンパイラでは、強力なデータ依存解析とループリストラクチャリング手法を組み合わせることでさまざまな形状のループが並列化可能になっている。しかしながら、これらのループ並列化手法では、単一ループのイタレーション（ループの繰り返し単位）間の並列性しか利用することができないという問題があった。この問題点を解決するために最近では、ループやサブルーチン等の粗粒度タスクレベルの並列性を利用するマクロデータフロー処理（粗粒度並列処理）[5)]、及び、階層的に粗粒度並列処理を行う階層型マクロデータフロー処理が提案されている[6)]。

この階層型マクロデータフロー処理のように、粗粒度タスクをプロセッサクラスタ（PC）に実行時に割り当てる方式では、粗粒度タスク間で共有されるデータを集中共有メモリ（CSM）上に配置し、粗粒度タスク間のデータ授受を集中共有メモリを介して行なうのが一般的である。しかし、このような方式では、集中共有メモリを介したデータ転送オーバヘッドが大きくなってしまうという問題が生じる。この問題点を解決するためには、処理とデータを適切に分割・配置し、各プロセッサ上のローカルメモリを介してデータ授受を実現することが必要となる。

†1 東邦大学 理学部 情報科学科
Dept. of Information Sciences, Toho University
†2 NTT（株）, NTT Corporation
†3 （株）東芝, Toshiba Corporation
†4 早稲田大学 理工学部 電気電子情報工学科, Dept. of Electrical, Electronics and Computer Engineering, Waseda Univ.

ローカルメモリへのデータの分割・配置に関する研究は、現在活発に行なわれており、ユーザ指定によるデータ分割・配置のための High Performance Fortran(HPF)[7]、ループ内の作業配列をローカル化する Array Privatization 法[8]、自動データ分割・配置法[9][10][11][12]などが提案されている。しかしながら、これらの方式は適用対象が単一ループに限られている、あるいは、ユーザやコンパイラがデータをスタティックに割り当てできる場合にしか適用できないという制約がある。

一方、ダイナミックスケジューリングを用いたマクロデータフロー処理においては、複数の粗粒度タスク間での共有データをローカルメモリ経由で授受するデータローカライゼーション手法[13][14]が提案されている。しかし、この手法は、階層的に粗粒度並列処理を行う階層型マクロデータフロー処理に適用することができなかった。そこで、本論文では、ループやサブルーチン等の粗粒度タスク内部で階層的に粗粒度並列処理される場合において、データローカライゼーションを実現する方法を提案する。

本論文第 2 章では、階層型マクロデータフロー処理について概説する。第 3 章では、階層型マクロデータフロー処理におけるデータローカライゼーションについて述べる。第 4 章では、本手法をインプリメントした自動並列化コンパイラを用いて性能評価した結果について述べる。

2. 階層型マクロデータフロー処理

本章では、階層型マクロデータフロー処理の対象アーキテクチャとコンパイレーションについて述べる。

2.1 対象マルチプロセッサアーキテクチャ

階層型マクロデータフロー処理では、プロセッサ（PE）上にローカルメモリ（LM）または分散共有メモリ（DSM）を持ち、各プロセッサがインタコネクションネットワークを介して集中共有メモリ（CSM）に平等に接続されているマルチプロセッサシステムを対象とする。

階層型マクロデータフロー処理を適用する場合、全プロセッサを分割して、階層的にプロセッサクラスタ（プロセッサのグループ）を定義する。具体的には、まず、マルチプロセッサシステム全体を第 0 階層プロセッサクラスタ（PC）と定義し、第 0 階層 PC 内の PE をグループ化して第 1 階層 PC を定義する。同様に、第 n 階層 PC 内の PE をグループ化して第 $(n+1)$ 階層 PC を定義する。

2.2 階層型マクロデータフロー処理のコンパイレーション

階層型マクロデータフロー処理[6]では、ループやサブルーチン等の粗粒度タスク（マクロタスク）が、プロセッサクラスタ（PC）に割り当てられて粗粒度並列処理が行なわれる。このとき、プロセッサクラスタに割り当てられたマクロタスク内部では、ループ並列化、近細粒度並列処理[15]、あるいは、階層的にマクロデータフロー処理が適用される。以下の節では、階層型マクロデータフロー処理のコンパイレーションについて概説する。

2.2.1 階層型マクロタスク生成

階層型マクロデータフロー処理では、まず、Fortran プログラム（全体を第 0 階層マクロタスクとする）を第 1 階層マクロタスクに分割する。マクロタスクは、擬似代入文ブロック（BPA）、繰り返しブロック（RB）、あるいは、サブルーチンブロック（SB）の 3 種類から構成される[6]。

BPA は本質的には基本ブロックであるが、並列処理効率向上のために単一の基本ブロックを分割して複数の BPA を生成したり、逆に、複数の基本ブロックを融合し 1 つの BPA を生成する。RB は最外側ナチュラルループ[16]である。また、コード長を考慮し効果的にインライン展開できないサブルーチンは SB として定義する。但し、RB が Doall ループである場合は、ループインデクス範囲を分割して、複数の部分 Doall ループを生成し、分割後の部分 Doall ループを新たに RB と定義する[13]。

次に、第 1 階層マクロタスク（RB または SB）の内部に複数のサブマクロタスク（サブ BPA、サブ RB、サブ SB）を含んでいる場合には、それらのサブマクロタスクを第 2 階層マクロタスクとして定義する。

2.2.2 階層型マクロフローグラフ生成

マクロタスク生成後、マクロタスク間あるいはサブマクロタスク間のコントロールフローとデータフローを解析し、図 1のような階層型マクロフローグラフ[6]を生成する。図 1の右側は、第 1 階層（図 1 の左側）の MT3 内部で生成された第 2 階層のサブマクロフローグラフである。

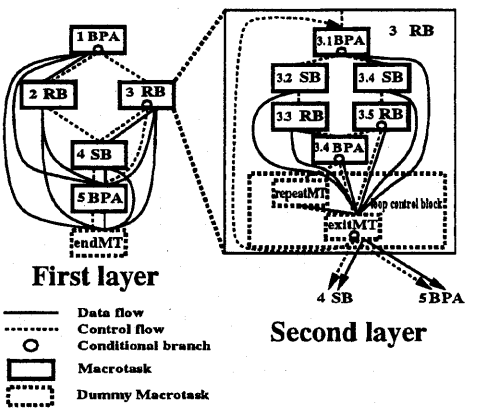

図 1 階層型マクロフローグラフ.

2.2.3 階層型マクロタスクグラフ生成

次に、コントロール依存とデータ依存を考慮したマクロタスク間並列性を最大限に引き出すために、各マクロタスクの最早実行可能条件[5]を解析する。最早実行可能条件は、コントロール依存とデータ依存を考慮したマクロタスク間の並列性を最大限に表している。この各マクロタスクの最早実行可能条件は、図2のような階層型マクロタスクグラフ（MTG）[6]で表すことができる。

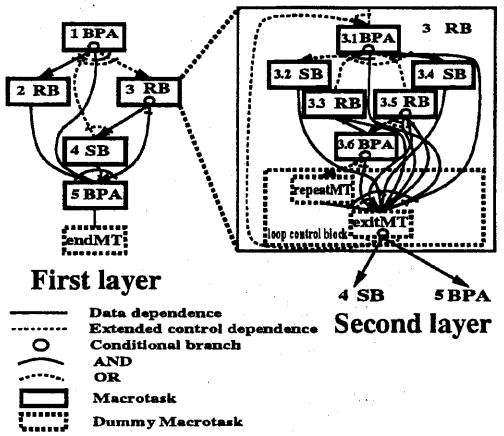

図2　階層型マクロタスクグラフ.

2.2.4 階層型ダイナミックスケジューリングルーチン生成

階層型マクロデータフロー処理では、マクロタスク間の条件分岐等の実行時不確定性に対処するために、マクロタスクを実行時にプロセッサクラスタ（PC）に割り当てるダイナミックスケジューリングルーチンを、各階層ごとにコンパイラが生成する。ダイナミックスケジューリングアルゴリズムとしては、Dynamic-CP法を採用している[6]。

3. 階層型マクロデータフロー処理におけるデータローカライゼーション

階層型マクロデータフロー処理を用いて各階層で粗粒度並列処理を行なう場合に、データ転送オーバヘッドを軽減し、効率良い並列処理を実現するためには、各階層においてデータローカライゼーションを行なうことが必要である。そこで、本手法では各階層において、(1) ループ整合分割法[13][14]を適用し、ローカル経由データ授受が行なえるように処理とデータを分割し、(2) パーシャルスタティックタスク割当を用いたダイナミックスケジューリング[14]により、多量のデータ転送を必要とするマクロタスク集合を実行時に同一プロセッサクラスタ（PC）にダイナミックスケジューリングする。また、(2) により同一PCに割り当てられるマクロタスク集合に対しては、(3) ローカルメモリデータ授受を可能とするデータ転送コードを生成する。

3.1 ループ整合分割

ループ整合分割法[13][14]は、データ依存のある複数ループ（RB）を、分割後に複数PC上での並列処理を可能とし、かつ、ローカルメモリ経由データ授受を実現できるように分割を行なうものである。以下の節では、階層型マクロデータフロー処理用のループ整合分割法について述べる。

3.1.1 ターゲットループグループ（TLG）生成

階層型マクロデータフロー処理用のループ整合分割では、まず、各階層のマクロタスクグラフ（MTG）から、ループ整合分割の対象となる部分MTG（これをターゲットループグループ（TLG）と呼ぶ）を選ぶ。例えば、図3(a) は、3つのループ（$RB1$、$RB2$、$RB3$）からなるターゲットループグループ（TLG）である。

ターゲットループグループ（TLG）は、MTG上でお互いが唯一の直接後続マクロタスク及び唯一の直接先行マクロタスクであるような配列変数に関するデータ依存が存在するループ集合（Doallループ、リダクションループ、ループキャリィドデータ依存を持ったシーケンシャルループ）である。なお、シーケンシャルループのボディ内部で階層型マクロデータフロー処理が適用される場合には、そのシーケンシャルループ自身はTLGに含めず、ボディ内部の階層においてTLGを生成する。

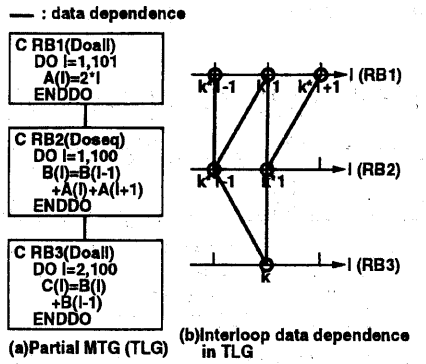

図3　ターゲットループグループ (TLG).

3.1.2 TLG内でのループ間データ依存解析

次に、各階層で生成された各TLG（m 個のRBで構成されているものとする）内部において、ループ間データ依存を解析する。このループ間データ依存は、TLGの出口ノード RB_m（TLG内で m 番目のRB、標準ループと呼ぶ）のイタレーションから、TLG内の各 RB_i（TLG内で i 番目のRB、$1 \leq i \leq m-1$）のイタレーションへの直接的あるいは間接的（他RBを経由した）データ依存を意味している。

例えば、図3(a)のTLGにおいてループ間データ依存を求めると、図3(b)に示すように、RB_3のk番目のイタレーションが、RB_2のイタレーション（$k*1-1$番目、$k*1$番目）と、RB_1のイタレーション（$k*1-1$番目、$k*1$番目、$k*1+1$番目）にデータ依存していることがわかる。

3.1.3 TLG内ループの整合分割

コンパイラは、各TLG内で使用されるデータの範囲を、標準ループRB_mのインデクス範囲で表す。これをグループ変換インデクス範囲（$GCIR$）と呼ぶ。その後、$GCIR$をPC数の整数倍（n）に均等に分割し、この各分割範囲を$DGCIR^p(1 \leq p \leq n)$とする。例えば、図3(a)のTLGの場合、$GCIR$と$DGCIR^p$（この例は3PC用）は図4のようになる。

次に、$DGCIR^p(1 \leq p \leq n)$とTLG内のループ間データ依存の解析結果を用いて、各$RB_i(1 \leq i \leq m)$を分割する。具体的には、部分標準ループ（ループインデクス範囲が$DGCIR^p$のもの）がデータ依存するRB_iのイタレーション集合を、Localizable-Region（RB_i^p）として生成し、複数の部分標準ループ（ループインデクス範囲が$DGCIR^p$と$DGCIR^{p+1}$のもの）が共通にデータ依存しているRB_iのイタレーション集合を、Commonly-Accessed-Region（$RB_i^{<p,p+1>}$）として生成する。

例えば、図3(a)のTLGを、3PC用にループ整合分割法で分割すると、図4のように分割される。この場合、RB_1は、3つのLocalizable-Region（RB_1^1、RB_1^2、RB_1^3）と、2つのCommonly-Accessed-Region（$RB_1^{<1,2>}$、$RB_1^{<2,3>}$）に分割される。

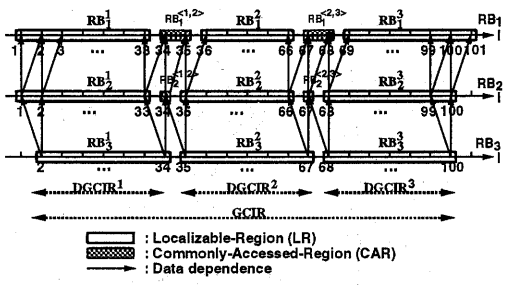

図4 ループ整合分割によるループインデクスの分割.

3.2 パーシャルスタティックタスク割当を用いたダイナミックスケジューリング

階層型マクロデータフロー処理により粗粒度並列処理される各階層において、多量のデータ転送を必要とするマクロタスク間で、プロセッサ上のローカルメモリを介したデータ授受（データローカライゼーション）を実現するためには、それらのマクロタスク集合（以後、データローカライゼーショングループ（DLG）と呼ぶ）を、ダイナミックスケジューリング環境下で、同一PCに割り当てなければならない。これを実現するために、本手法では、パーシャルスタティックタスク割当[14]を用いたダイナミックスケジューリングルーチンを生成する。

3.2.1 データローカライゼーショングループ生成

各TLG内でループ整合分割が行なわれた後、データ転送量の多い部分RB集合（Localizable-Region(LR)）を、データローカライゼーショングループ（DLG）とする。なお、処理時間の短いCommonly-Accessed-Region(CAR)は、ダイナミックスケジューリングのオーバヘッドを軽減するため、隣接するLocalizable-Region(LR)に融合しておく。例えば、図5は、図4から生成されたデータローカライゼーショングループ（DLG）である。

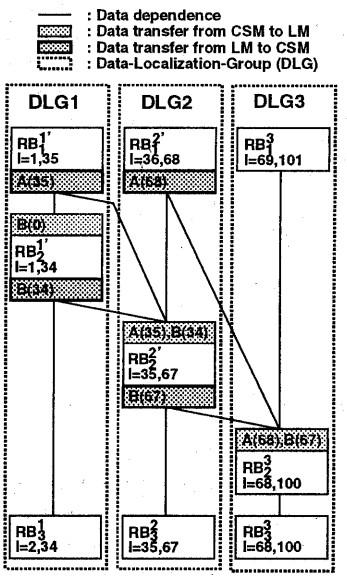

図5 データローカライゼーショングループ(DLG).

3.2.2 実行時のダイナミックスケジューリングルーチンの動作

パーシャルスタティックタスク割当を用いたダイナミックスケジューリングでは、あるデータローカライゼーショングループDLG_dの入口マクロタスク$MT_{(d,entrance)}$が実行時にPCへ割り当てられる際に、DLG_d内の後続マクロタスク$MT_{(d,i \neq entrance)}$の割り当てられるべきPC番号を指定する。その後、$MT_{(d,i \neq entrance)}$の最早実行可能条件が満たされた時には、実行待ちMTキューへ投入後、指定されたPCへ割り当てられる方式をとっている。

3.3 データローカライゼーション用データ転送コード生成

本章では、3.2.1で生成されたデータローカライゼーショングループ内のマクロタスク間で、PC内の各PE上のローカルメモリ（LM）経由でデータ授受を行う

並列マシンコードを生成する。

3.3.1 LM 経由データ授受を適用する配列の検出

ここでは、データローカライゼーショングループ内でローカルメモリ経由データ授受を適用する配列変数を検出する。具体的には、まず、データローカライゼーショングループ内部で、各配列変数 X ごとに、データローカライゼーション適用時に必要となるデータ転送を解析してそのデータ転送に要する時間 $t^X_{localize}$ を算出し、また、従来の集中共有メモリ（CSM）経由データ授受を行う場合のデータ転送時間 t^X_{CSM} を求める。次に、$t^X_{localize}$ と t^X_{CSM} を比較し、$t^X_{localize} < t^X_{CSM}$ を満たす配列変数 X に対しては、ローカルメモリ経由データ転送コードを生成する。

3.3.2 LM 経由データ転送コード生成

データローカライゼーショングループ内では、データローカライゼーションの適用される配列変数に対して、各 PE 上のローカルメモリへのロード・ストア命令を生成する。

また、データローカライゼーションの適用される配列変数のデータが、データローカライゼーショングループ外のマクロタスクと共有される場合には、CSM 経由でデータ授受を行なう転送コードを生成する。例えば、図5において、異なる DLG に属する RB（$RB_2^{1'}$ と $RB_2^{2'}$）間で共有されている配列データ（B(34)）は、CSM 経由で転送される。

4. 性能評価

本章では、データローカライゼーションを伴う階層型マクロデータフロー処理を、複数のアプリケーションプログラムを用いて、OSCAR[17]のシミュレータ（OSCAR のマシンコードをクロックレベルで厳密にシミュレート可能）上で性能評価した結果について述べる。

4.1 OSCAR のアーキテクチャ

性能評価に用いる OSCAR[17]は、ローカルメモリ（LM）を持つ 32 ビット RISC プロセッサ（PE）を、集中共有メモリ（CSM）に 3 本のバスで接続した分散・集中共有メモリ型マルチプロセッサシステムである。ローカルメモリ（LM）は、ローカルプログラムメモリ（LPM）、ローカルデータメモリ（LDM）、他 PE からリード・ライト可能な分散共有メモリ（DSM）の領域からなる。なお、OSCAR では、PE 上の LDM 及び DSM へのロードやストアに 1 クロックを要し、CSM 及び他 PE の DSM へのロードやストアには 4 クロックを要する。OSCAR は各 PE を集中共有メモリ（CSM）に平等結合したアーキテクチャとなっているが、複数の PE をソフトウェア的にクラスタリング（グループ化）することにより、マルチプロセッサクラスタシステムとして利用することができる。

4.2 Tomcatv プログラムを用いた性能評価

本節では、SPECfp95 ベンチマークの Tomcatv プログラムを用いて、階層型マクロデータフロー処理におけるデータローカライゼーション手法の性能評価を行なう。このプログラムは、Vectorized Mesh 生成プログラムであり、初期化部分と収束計算ループから構成されている。本手法では、処理時間の大部分を費やす収束計算ループ内のボディ部（第 2 階層）に対して階層型マクロデータフロー処理を適用する。この収束計算ループ内のボディ部（第 2 階層）は、6 個の Doall ループ（この内、3 つの Doall ループに対しては、Scalar Privatization と Loop Interchange を適用している）と、1 つのリダクションループ（総和計算ループ）、4 つの基本ブロックから構成されている。

このプログラムを OSCAR シミュレータ上で実行した結果を表1に示す。通常の階層型マクロデータフロー処理では、1PE 上での処理時間を、2PE で 1/1.96、4PE で 1/3.87、6PE で 1/5.45 に短縮している。これに対し、データローカライゼーションを伴う階層型マクロデータフロー処理を適用すると、データローカライゼーション手法を適用しない場合と比べて、2PE で 20.2%、4PE で 20.4%、6PE で 22.0%の速度向上が得られた。このことから、本手法の有効性が確認された。

なお、本性能評価において、階層型マクロデータフロー処理の場合には、集中ダイナミックスケジューラ方式を採用しているため、プログラムを計算処理する PE 以外に、ダイナミックスケジューラ用として 1PE を使用している。

表1 Tomcatv プログラムによる OSCAR 上の評価

処理方式	実行時間 [s]			
	1PE	2PE	4PE	6PE
シーケンシャル処理	10.499			
HMDF without DL		5.357	2.712	1.928
HMDF with DL		4.275	2.158	1.503

(注)HMDF：階層型マクロデータフロー処理
DL：データローカライゼーション

4.3 CG 法プログラムを用いた性能評価

次に、連立1次方程式間接求解法である CG(Conjugate Gradient)法プログラムを用いて提案手法を性能評価する。CG 法プログラムは、第 1 階層が初期化部分と収束計算ループからなり、処理時間のほとんどは収束計算ループに費やされる。そこで、本手法では、この収束計算ループのボディ部（第 2 階層）において、階層型マクロデータフロー処理を適用する。この収束計算ループのボディ部（第 2 階層）は、4 つの Doall ループ、2 つのリダクションループ、5 つの基本ブロックで構成されている。

OSCAR シミュレータ上でこのプログラムを階層型マクロデータフロー処理で実行した結果は、表2に示すように、1PE 上での処理時間を 2PE で 1/1.99、4PE で 1/3.65、6PE で 1/4.89 に短縮している。これに対

し、データローカライゼーションを伴う階層型マクロデータフロー処理では、処理時間が10%程度短縮されることが確かめられた。このことにより、階層型マクロデータフロー処理におけるデータローカライゼーション手法の有効性が確認された。

表2　CG法プログラムによるOSCAR上の評価

処理方式	実行時間[s]			
	1PE	2PE	4PE	6PE
シーケンシャル処理	155.40			
HMDF without DL		78.17	42.58	31.79
HMDF with DL		73.18	38.95	28.81

(注)HMDF：階層型マクロデータフロー処理
DL：データローカライゼーション

5. おわりに

本論文では、階層型マクロデータフロー処理により粗粒度並列処理される各階層において、粗粒度並列性を最大限に利用しつつ、粗粒度タスク間データ転送にプロセッサ上のローカルメモリを有効利用し、データ転送オーバヘッドを軽減するデータローカライゼーション手法を提案した。

OSCARシミュレータ上での性能評価により、提案したデータローカライゼーション手法は、階層的に粗粒度並列処理される粗粒度タスク間データ転送を軽減し、集中共有メモリ経由データ授受を行う階層型マクロデータフロー処理に比べて、処理時間を顕著に短縮できることが確かめられた。

今後の課題としては、階層型マクロデータフロー処理により実行される異なる階層のマクロタスク間でデータローカライゼーションを実現することがあげられる。

参考文献

1) Banerjee, U.: Loop Transformations for Restructuring Compilers, *Kluwer Academic Pub.* (1993).
2) Wolfe, M.: High Performance Compilers for Parallel Computing, *Addison-Wesley Publishing Company* (1996).
3) Blume, W., Doallo, R., Eigenmann, R., Grout, J., Hoeflinger, J., Lawrence, T., Lee, J., Padua, D., Paek, Y., Pottenger, B. and L.Rauchwerger, P. T.: Advanced Program Restructuring for High Performance Computers with Polaris, *Technical Report 1473, CSRD, University of Illinois, Urbana-Champaign* (1996).
4) Amarasinghe, S., Anderson, J., Lam, M. and Tseng, C.-W.: The SUIF Compiler for Scalable Parallel Machines, *Proc. of the seventh SIAM conference on parallel processing for scientific computing* (1995).
5) 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 (D-I), Vol. J73-D-I, No. 12, pp. 951–960 (1990).
6) 岡本雅巳, 合田憲人, 宮沢稔, 本多弘樹, 笠原博徳: OSCARマルチグレインコンパイラにおける階層型マクロデータフロー処理手法, 情報処理学会論文誌, Vol. 35, No. 4, pp. 513–521 (1994).
7) Forum, H. P. F.: High Performance Fortran Language Specification DRAFT Ver.1.0, *High Performance Fortran Forum* (1993).
8) Tu, P. and Padua, D.: Automatic Array Privatization, *6th Annual Workshop on Languages and Compilers for Parallel Computing* (1993).
9) Chen, T.-S. and Sheu, J.-P.: Communication-Free Data Allocation Techniques for Parallelizing Compilers on MUlticomputers, *IEEE trans. on parallel and distributed systems*, Vol.5, No.9 (1994).
10) Agarwal, A., Kranz, D. A. and Natarajan, V.: Automatic Partitioning of Parallel Loops and Data Arrays for Distributed Shared-Memory Multiprocessors, *IEEE Trans. on Parallel and Distributed System*, Vol. 6, No. 9, pp. 943–962 (1995).
11) Gupta, M. and Banerjee, P.: Demonstration of Automatic Data Partitioning Techniques for Parallelizing Compilers on Multicomputers, *IEEE Trans. on Parallel and Distributed System*, Vol. 3, No. 2, pp. 179–193 (1992).
12) Anderson, J. and Lam, M.: Global Optimizations for Parallelism and Locality on Scalable Parallel Machines, *Proc. of the SIGPLAN '93 Conference on Programming Language Design and Implementation*, pp. 112–125 (1993).
13) 吉田明正, 前田誠司, 尾形航, 笠原博徳: Fortran マクロデータフロー処理におけるデータローカライゼーション手法, 情報処理学会論文誌, Vol. 35, No. 9, pp. 1848–1860 (1994).
14) Yoshida, A., Koshizuka, K. and Kasahara, H.: Data-Localization for Fortran Macro-Dataflow Computation Using Partial Static Task Assignment, *Proceedings of 10th ACM International Conference on Supercomputing*, pp. 61–68 (1996).
15) Kasahara, H., Honda, H. and Narita, S.: Parallel Processing of Near Fine Grain Tasks Using Static Scheduling on OSCAR, *IEEE ACM Supercomputing'90* (1990).
16) Aho, A., Sethi, R. and Ullman, J.: Compilers (Principles, Techniques, and Tools), *Addison Wesley* (1988).
17) 笠原博徳, 成田誠之助, 橋本親: OSCARのアーキテクチャ, 電子情報通信学会論文誌 (D), Vol. J71-D, No. 8 (1988).