ハイパフォーマンス
コンピューティング 52−15
（1994. 7. 23）

HDPG：階層データ分割グラフ

中西 恒夫†, 城 和貴†, Constantine D. Polychronopoulos‡,
福田 晃†, 荒木 啓二郎†

†:奈良先端科学技術大学院大学 情報科学研究科
‡:Center for Supercomputing Research and Development,
University of Illinois at Urbana-Champaign

概要

本稿では、並列化コンパイラの中間表現として階層データ分割グラフ HDPG を提案する。従来のデータ分割手法は一般的にループ並列性に応じたものが多く、その分割の最適化は特定のループについて局所的になされる。そのため、複数のループを考えた場合は個々のループでの分割の間に不整合が生じがちである。HDPG は、我々がすでに提案しているデータ分割グラフ DPG をプログラムの制御構造に応じて階層化したもので、異なる粒度における機能並列性に応じた、大域的なデータ分割手法を提供する土台としての機能する。HDPG 上では、メモリ階層等のハードウェアの階層性と親和性の高いデータ分割が可能である。

HDPG : Hierarchical Data Partitioning Graph

Tsuneo Nakanishi†, Kazuki Joe†, Constantine D. Polychronopoulos‡,
Akira Fukuda†, Keijiro Araki†

†:Graduate School of Information Science,
Nara Institute of Science and Technology
8916-5, Takayama-cho, Ikoma-shi, Nara 630-01, Japan
‡:Center for Supercomputing Research and Development,
University of Illinois at Urbana-Champaign
1308 West Main St. Urbana, IL 61801, USA
$E-mail$: $\{tuneo-n, kazuki-j\}$@$is.aist-nara.ac.jp$, cdp@$csrd.uiuc.edu$,
$\{fukuda, araki\}$@$is.aist-nara.ac.jp$

ABSTRACT

In this paper we propose HDPG — Hierarchical Data Partitioning Graph — as an intermediate representation for parallelized compilers. Traditional data partitioning techniques cope with loop parallelism and tend to get an optimal solution in a loop rather than in multiple loops. HDPG, which is an extension of HTG (Hierarchical Task Graph), can provide with a framework for data partitioning coping with functional parallelism at various granulalities. Using HDPG compilers can obtain an optimal data partitioning in a given program, moreover do partition data matching with the hierarchy in architectures such as the memory hierarchy.

1 はじめに

近年のハードウェア技術の進歩に伴い、マルチプロセッサシステムには数千～数万のプロセッサを実装することが可能となってきた。メモリ上での競合回避や実装技術等を考えれば、このようなシステムは分散共有システム、もしくはメッセージパッシング型システムとして実装するのが妥当であろう。

分散共有型システムではメモリアクセスが分散される反面、データを分散メモリに適切に分散（データ分割）しなければ、リモートのメモリアクセスが頻発し、システムのパフォーマンスが著しくダウンする。一方、メッセージパッシング型システムでは、一般にプロセッサ間通信のバンド幅が小さく[6]、共有メモリ型マルチプロセッサシステムのように大量のデータをプロセッサ間で引き渡すことはできない。そのためメッセージパッシング型システムでは、より完璧に近いデータ分割が望まれる。

データ分割は、FFT，行列計算といった個々のアプリケーションごとに最適のものを求める研究がなされてきたが、既製の逐次プログラムを並列プログラムに変換する自動並列化コンパイラには、これらの手法はほとんど適用不可能である。自動並列化コンパイラ用の一般的なデータ分割手法も多々発表されているが[3][4][5]、専らループ並列性を重視したものであり、特定のループについてしか最適化を行っていなかったり（局所的な最適化）、プログラムに厳しい仮定を設けているなど、問題も多く残されている。

以上のような背景を鑑み、本研究ではアプリケーションに依存しない、一般的なデータ分割の枠組の提供を図る。計算機は分散共有型システムを対象とする。本研究では、自動並列化コンパイラにおける中間表現として用いられているタスクグラフの一種である、制御データ依存グラフ (CDDG: Control Data Dependency Graph) を拡張した、データ分割グラフ (DPG: Data Partitioning Graph) をすでに提案している [7]。データ分割問題は、DPG 上では節点をグルーピングする問題として取り扱われる。しかしながら DPG の提供するモデルは、現実のプログラムの並列実行とはギャップが大きい嫌いがある。一般に並列計算には階層性があり、また計算機のアーキテクチャにもメモリ階層を始めとする階層性が存在する。本稿では、こうした階層性を反映すべく、DPG を階層化した階層データ分割グラフ (HDPG: Hierarchical DPG) を提案、その定義・実例を示す。HDPG でのデータ分割は、特定のループ上で行われるものではなく、プログラム全体に対して行われる大域的なものである。そのため、機能並列性に応じたデータ分割が可能である。

第 2 章では階層タスクグラフの概要について述べる。第 3 章では DPG/HDPG の概要について触れ、第 4 章では DPG/HDPG 上でのデータ分割について検討する。

2 階層タスクグラフ

これまで、自動並列化コンパイラにおいて用いられてきたタスクグラフとして、制御フローグラフ (CFG: Control Flow Graph)，制御依存グラフ (CDG: Control Dependency Graph)，データ依存グラフ (DDG: Data Dependency Graph) 等が挙げられる。自動並列化コンパイラによる並列性の取り扱いについて考えた場合、これらのグラフは特定の粒度での並列性しか扱えないという問題点をはらんでいた。並列性はタスクレベルからインストラクションレベルまであらゆる粒度で存在し、またこれらの並列性の間には階層が存在する。そのため、タスクグラフを階層化し、こうした並列性の階層性とのギャップを埋めるアプローチがなされている [2][8]。

階層タスクグラフ (HTG: Hierarchical Task Graph) は、CFG をループや関数等の構造に基づき階層化したもので、イリノイ大の自動並列化コンパイラ Parafrase-2[1] において用いられている。例えば、図 1のプログラムの HTG は、図 2左のようになる。節点は文，もしくはループに対応するタスクを表し、枝はタスク間の制御フローを表現している。

HTG の各階層の CFG はいずれも CDG に変換され、階層制御依存グラフ (HCDG: Hierarchical CDG) が構成される。さらに HCDG にデータ依存を示す枝が追加され、階層制御データ依存グラフ (HCDDG: Hierarchical Control Data Dependency Graph) が生成される。図 1 のプログラムの HCDDG を図 2右に示す。節点は文，もしくはループに対応するタスクを表し、実線枝はタスク間のデータ依存関係を表現している。また、点線はタスク間の制御依存関係を表し、その属性として制御依存情報が付与されている。($a-b$ はタスク a で分岐方向が定まらない限り、タスク b の実行を開始できないことを意味している。)

HTG，HCDDG の定義，生成方法等については、文献 [2] を参照されたい。

```
L 1:     DO K=1, NM, LAMBDA
S 1:         T1 = K DIV M + 3
S 2:         T2 = (K + LAMBDA) DIV M + 3
S 3:         T3 = K MOD M + 5
S 4:         T4 = (K + LAMBDA) MOD M + 5 - 1
L 2:         DO I = T1, T2
S 5:             IF I <> T1
S 6:                 THEN L1 = 5
S 7:                 ELSE L1 = T3
S 8:             IF I <> T2
S 9:                 THEN L2 = N2
S10:                 ELSE L2 = T4
L 3:             DO J = L1, L2
S11:                 A(I, J) = B(I - 3, J - 5)
S12:                 B(I, J) = A(I - 2, J - 4)
                 EndDO
             EndDO
         EndDO
```

図 1: プログラム例

3 階層データ分割グラフ

本章では、データ分割グラフ (DPG: Data Partitioning Graph)/階層データ分割グラフ (HDPG: Hierarchical DPG) の概要を述べ、さらに HDPG の形式的な定義を行う。

3.1 DPG の概要

自動並列化コンパイラで用いられているタスクグラフのひとつとして、制御データ依存グラフ (CDDG: Control Data Dependency Graph) がある。DPG はこの CDDG を拡張した形式となっており、CDDG より生成される。

DPG では、複数のタスクからアクセスされる変数および配列要素（共有されている変数群）を、アクセスするタスク，アクセスの種別 (Read or Write) ならびにその時のコストによりクラス分けする。各クラスに属する変数群は DPG 上のひとつの節点として表現される。これらの節点は D–node と呼ばれ、方形の節点で表される。

一方、タスクならびにそのタスクのみがアクセスする変数および配列要素（タスクについてプライベートな変数群）は、円形の節点で表される[1]。これらの節点は C–node と呼ばれる。ある C–node cv に対応するタスクが、D–node dv の表すクラスに含まれる変数，配列要素に Read または Write アクセスする場合、それぞれ dv から cv，cv から dv に枝が張られる。この枝には、属性としてアクセスに伴うコストが付与される。DPG では、プロセッサ割当は C–node をグルーピングする問題、データ分割は D–node をグルーピングする問題として取り扱われる。

図 3に DPG の一例を示す。

[1] この定義は [7] において発表したものより、若干の改変を加えている

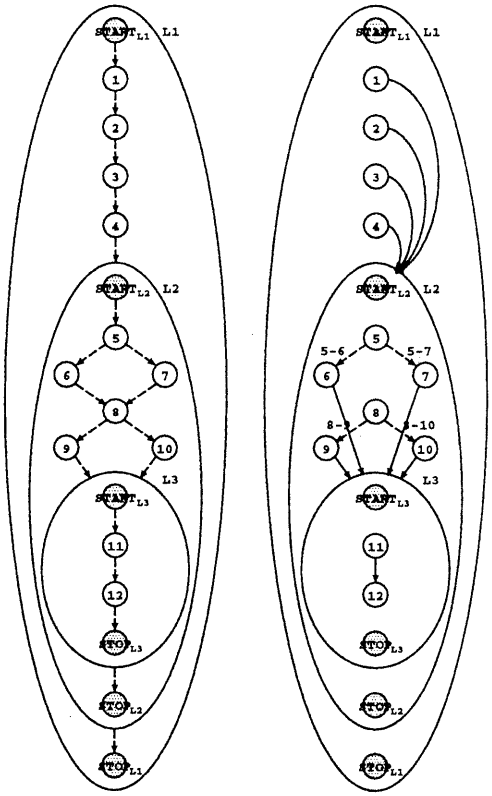

図 2: HTG（左）／HCDDG（右）

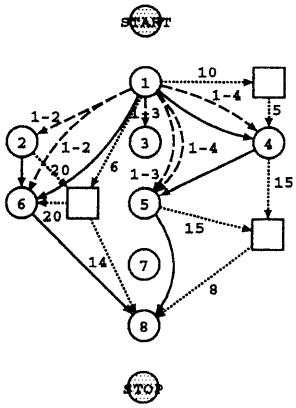

図 3: DPG

3.2 HDPG の概要

前節で述べたように、HTG 中の各節点中の CFG は CDG に変換され、さらにデータ依存枝を加えて CDDG となり、最終的には HCDDG が得られる。HDPG は、HCDDG の各節点中の CDDG を DPG に変換することにより得られる。図 1の HDPG を図 4に示す。HDPG の定義は DPG の定義とほぼ同様であるが、変数のプライベート/共有の別については異なる。HDPG では、変数のプライベート/ 共有の別は、同じ変数でも注目する階層によって変わり得る。

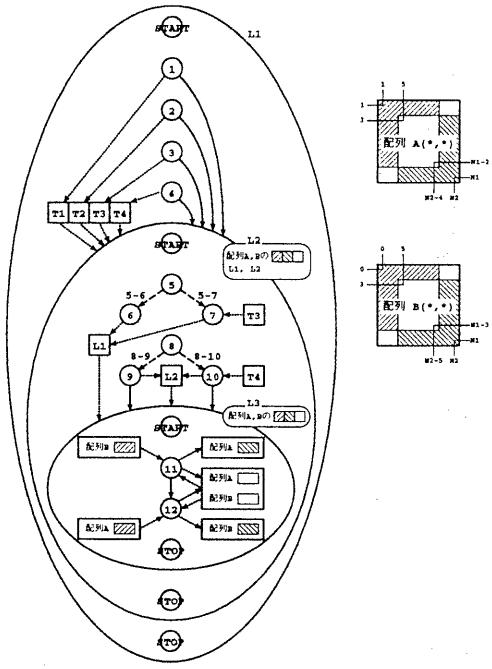

図 4: HDPG

3.3 HDPG の定義

本節では、曖昧さを排すべく HDPG の形式的な定義を行う。HDPG は HCDDG より生成される。HDPG, HCDDG はそれぞれ形式的に HDPG($\{HCV, HDV\}$, $\{HCE_c, HCE_d, HDE_r, HDE_w\}$), HCDDG($HV$, $\{HE_c, HE_d\}$) と記述されるものとする。HDPG について、HCV は C–node の集合, HDV は D–node の集合, HCE_c は制御依存を表す枝の集合, HCE_d はデータ依存を表す枝の集合, HDE_r は D–node から C–node への枝の集合（すなわち Read アクセスに該当する枝の集合）, HDE_w は C–node から D–node への枝の集合（すなわち Write アクセスに該当する枝の集合）である。HCDDG について、HV は HCDDG の節点の集合, HE_c は制御依存を表す枝の集合, HE_d はデータ依存を表す枝の集合である。HCDDG の節点, HDPG の C–node はいずれもタスクと一対一で対応している。本稿では以下、表現を簡潔にするため、HCDDG の節点もしくは HDPG の C–node と、タスクを適宜入れ換えて使うことがあるので注意されたい。

節点の親子関係

HCDDGの節点には、階層に基づく親子関係が存在する。任意のHCDDGの節点$hv \in HV$について、節点hvが内包するCFGの節点をhvの子といい、節点hvの子の集合を$C(hv)$と記す。節点hvについて、集合$D(hv) = C(hv) \cup \{hv'' : hv' = D(hv), hv'' = C(hv')\}$を定義し、その要素を$hv$の子孫という。また、$hv$が$\hat{hv}$の子である時、$\hat{hv}$を$hv$の親と呼び、$p(hv)$と記す。節点$hv$について、集合$A(hv) = \{p(hv)\} \cup \{hv'' : hv' = A(hv), hv'' = p(hv')\}$を定義し、その要素を$hv$の先祖という。例えば、図2右のHCDDGの節点$L2$について、$D(L2)$, $C(L2)$, $A(L2)$, $p(L2)$は以下のようになる。

$$
\begin{aligned}
C(L2) &= \{START_{L2}, 5, 6, 7, 8, 9, 10, L3, STOP_{L2}\} \\
D(L2) &= \{START_{L2}, 5, 6, 7, 8, 9, 10, \\
&\quad L3, START_{L3}, 11, 12, STOP_{L3}, STOP_{L2}\} \\
p(L2) &= L1 \\
A(L2) &= \{L1\}
\end{aligned}
$$

変数の共有/プライベートの別

HCDDGにおける変数の共有/プライベートの別について定義する。HCDDGにおいてある変数varが、タスク$hv \in HV$および$A(hv)$, $D(hv)$においてのみ参照され、プログラム中の他のタスクからは一切参照されない場合、その変数は$\hat{hv} = p(hv)$においてhvについてプライベートであるという。一方、varがタスク$hv \in HV$および$A(hv)$, $D(hv)$以外のタスクからも参照される場合、その変数は$\hat{hv} = p(hv)$において共有されているという。

タスク$\hat{hv} \in HV$および$D(\hat{hv})$で参照される変数の集合を$VAR_{\hat{hv}}$と記す。また$\hat{hv}$において、$hv \in C(\hat{hv})$についてプライベートな変数の集合を$PVAR_{\hat{hv}}(hv)$, 共有されている変数の集合を$SVAR_{\hat{hv}}$と記す。

HCV, HCE_c, HCE_dの定義

CDDGのHVの要素と、DPGのHCVの要素は一対一の対応関係にある。任意の$hv \in HV$に対応するHCVの要素を$g_{HV \to HCV}(hv)$と表現することにする。関数$g_{HV \to HCV}$を用いて、HV, HE_c, HE_dより次のようにHCV, HCE_c, HCE_dを改めて定義する。

$$
\begin{aligned}
HCV &= \{g_{HV \to HCV}(hv) : hv \in HV\} \\
HCE_c &= \{(hcv_1, hcv_2) : (hv_1, hv_2) \in HE_c, \\
&\quad hcv_1 = g_{HV \to HCV}(hv_1), \\
&\quad hcv_2 = g_{HV \to HCV}(hv_2)\} \\
HCE_d &= \{(hcv_1, hcv_2) : (hv_1, hv_2) \in HE_d, \\
&\quad hcv_1 = g_{HV \to HCV}(hv_1), \\
&\quad hcv_2 = g_{HV \to HCV}(hv_2)\}
\end{aligned}
$$

ここでHCVの要素について、HVの場合と同様に親子関係が定義される。

$$
\begin{aligned}
C(hcv) &= \{hcv' \in HCV : \\
&\quad hv' \in C(g^{-1}_{HV \to HCV}(hcv)), \\
&\quad hcv' = g_{HV \to HCV}(hv')\} \\
D(hcv) &= \{hcv' \in HCV : \\
&\quad hv' \in D(g^{-1}_{HV \to HCV}(hcv)), \\
&\quad hcv' = g_{HV \to HCV}(hv')\} \\
p(hcv) &= g_{HV \to HCV}(p(g^{-1}_{HV \to HCV}(hcv))) \\
A(hcv) &= \{hcv' \in HCV : \\
&\quad hv' \in A(g^{-1}_{HV \to HCV}(hcv)), \\
&\quad hcv' = g_{HV \to HCV}(hv')\}
\end{aligned}
$$

さらにHCVについて、HVの場合と同様に、次のように集合$PVAR$, $SVAR$が定義される。(但し、$hcv = g_{HV \to HCV}(hv)$, $\hat{hcv} = g_{HV \to HCV}(\hat{hv})$である。)

$$
\begin{aligned}
PVAR_{\hat{hcv}}(hcv) &= PVAR_{\hat{hv}}(hv) \\
SVAR_{\hat{hcv}} &= SVAR_{\hat{hv}}
\end{aligned}
$$

$hcv \in HCV$は属性として、タスクhcvの実行コスト、ならびに$\hat{hcv} = p(hcv)$においてhcvについてプライベートな変数の群$PVAR_{\hat{hcv}}(hcv)$を持つ。定義よりHCE_c, HCE_dの要素は、HE_c, HE_dの要素と一対一対応する。HCE_c, HCE_cのいずれに属する枝も、対応するHE_c, HE_dに属する枝から、属性をそのまま引き継ぐ。

アクセスコスト

データ分割の対象となるのは、タスク間で共有される変数である。したがって、共有変数へのアクセスに伴うコストの定量化は、データ分割を行うにあたり極めて重要である。DPGならびにHDPGでは、C-nodeとD-nodeの間の枝に、C-nodeに該当するタスクがD-nodeの表す変数群に属する変数へアクセスする際の、アクセスコストが付与される。このアクセスコストは、いわゆる通信コストと混同してはならない。通信コストはハードウェアと密な関係にあり、プロセッサ割当が完了するまで決定されない量である。反面、アクセスコストはハードウェアをあまり意識しないソフトウェア的な量である。

タスク$hcv \in HCV$が共有変数$var \in SVAR_{\hat{hcv}}$(但し、$\hat{hcv} = p(hcv)$である)にReadアクセス, Writeアクセスする時のトータルのアクセスコストを、それぞれ$\omega_r(hcv, var)$, $\omega_w(hcv, var)$と記述する。タスクhcvが子を持たない場合、すなわち$C(hcv) = \phi$の場合、$\omega_r(hcv, var)$および$\omega_w(hcv, var)$は次のように定義される。

$$\omega_{r/w}(hcv, var) = n \times r \times \alpha$$

但し、nはアクセスする共有変数のサイズ, rは共有変数へのアクセスが生じる頻度である。また、αは係数で、アクセスの種別やキャッシュのヒット率などを考慮して、決定される定数である。一方、タスクhcvが子を持つ場合、すなわち$C(hcv) \neq \phi$の時は、その子孫のタスクがvarへアクセスする時のアクセスコストの合計、すなわち以下のようになる。

$$\omega_{r/w}(hcv, var) = \sum_{hcv' \in C(hcv)} \omega_{r/w}(hcv', var)$$

hcvがvarにReadアクセスしない時は$\omega_r(hcv, var) = 0$、Writeアクセスしない時は$\omega_w(hcv, var) = 0$と定義する。

DPG/HDPGでは変数を、アクセスするタスク, アクセスの種別とコストによってクラス分けし、各クラスについてD-nodeを生成することについては、すでに述べた。上記のアクセスコストは一般に連続値であり、したがってそのままの形で用いると、膨大な数の変数のクラス、おそらく変数の数と同数のクラスが生成され、結果無用な数のD-nodeが作られ、グラフは極めて大きなものとなる。そのため、DPGではアクセスコストを適切に離散化することにより、変数のクラス分けを行う。ここで、アクセスコストωを離散化する関数を$f(\omega)$と定義する。たとえば、$f(\omega)$は図5のような概形を持つ関数である。

HDVの定義

変数varの$\hat{hcv} \in HCV$におけるアクセスパターン$ap_{\hat{hcv}}(var)$を、次のような3つ組の集合と定義する。

$$ap_{\hat{hcv}}(var) = \{(hcv, r, w) : hcv \in C(\hat{hcv}),$$

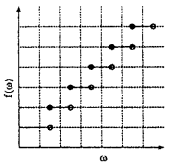

図 5: 関数 $f(\omega)$

$$r = f(\omega_r(hcv, var)),$$
$$w = f(\omega_w(hcv, var))\}$$

$\hat{hv}$において共有されている全変数 $SVAR_{\hat{hcv}}$ のアクセスパターンの集合を、$AP_{\hat{hcv}}$ と定義する。

$$AP_{\hat{hcv}} = \{ap_{\hat{hcv}}(var) : var \in SVAR_{\hat{hcv}}\}$$

$AP_{\hat{hcv}}$ は集合であるので、全く等しいアクセスパターンは区別されないことに注意されたい。$AP_{\hat{hcv}}$ の個々の要素について、ひとつの変数のクラス

$$VAR_{\hat{hcv}}(ap) = \{var \in SVAR_{\hat{hcv}} : ap_{\hat{hcv}}(var) = ap\}$$

が定義され、このクラスが$\hat{hcv}$に内包されるひとつの D–node をなす。$\hat{hcv}$内の D–node の集合を $HDV_{\hat{hcv}}$ とすれば、$AP_{\hat{hcv}}$ と $HDV_{\hat{hcv}}$ は一対一対応となる。$ap \in AP_{\hat{hcv}}$ に対応する $HDV_{\hat{hcv}}$ の要素を$g_{AP \to HDV_{\hat{hcv}}}(ap)$ と表すと、$HDV_{\hat{hcv}}$ は改めて次のように定義される。

$$HDV_{\hat{hcv}} = \{hdv = g_{AP \to HDV_{\hat{hcv}}}(ap) : ap \in AP_{\hat{hcv}}\}$$

さらに、HDVは改めて次のように定義される。

$$HDV = \bigcup_{\hat{hcv} \in HCV} HDV_{\hat{hcv}}$$

HDE_r, HDE_wの定義

$HDE_{r\,\hat{hcv}}$は次の条件を満足する (hdv, hcv) の集合である。

- $hcv \in C(\hat{hcv})$, $hdv \in HDV_{\hat{hcv}}$
- $(hcv, r, w) \in g^{-1}_{AP \to HDV\,\hat{hcv}}(hdv)$ において $r > 0$

$HDE_{w\,\hat{hcv}}$は次の条件を満足する (hcv, hdv) の集合である。

- $hcv \in C(\hat{hcv})$, $hdv \in HDV_{\hat{hcv}}$
- $(hcv, r, w) \in g^{-1}_{AP \to HDV\,\hat{hcv}}(ap)$ において $w > 0$

上記、$HDE_{r\,\hat{hcv}}$, $HDE_{w\,\hat{hcv}}$を用いて、HDE_r, HDE_wは改めて次のように定義できる。

$$HDE_r = \bigcup_{\hat{hcv} \in HCV} HDE_{r\,\hat{hcv}}$$

$$HDE_w = \bigcup_{\hat{hcv} \in HCV} HDE_{w\,\hat{hcv}}$$

任意の $hde_r = (hdv, hcv) \in HDE_{r\,\hat{hcv}}$, $hde_w = (hcv, hdv) \in HDE_{w\,\hat{hcv}}$は、属性としてタスク hcvが hdv 内の変数に Read, Write する時のコストのトータル

$$\sum_{var \in VAR_{\hat{hcv}}(ap)} \omega_{r/w}(hcv, var)$$

が付与される。但し、$\hat{hcv} = p(hcv)$, $g_{AP \to HDV\,\hat{hcv}}(ap) = hdv$であるものとする。

4 DPG/HDPG 上でのデータ分割

前述したように DPG/HDPG 上でのデータ分割およびプロセッサ割当は、DPG/HDPG の節点をグルーピングする問題として取り扱われる。本章では、このうちデータ分割の枠組について主に述べることにする。計算機については、図6に示すようなクラスタ共有メモリ型計算機を仮定する。なお、実際のデータ分割アルゴリズムついては、現在検討段階である。

図 6: クラスタ共有メモリ型計算機

4.1 DPG 上でのデータ分割

DPG には C–node, D–node の 2 種類の節点が存在する。データ分割は、D–node にクラスタを割り当てることによってなされる。D–node dvにクラスタ C を割り当てた場合、dvに相当する変数群が Cのメモリ中に配置されることを意味する。一方、プロセッサ割当は、C–node にプロセッサまたはクラスタを割り当てることによってなされる。C–node が文のような細粒度のタスクを表す場合はプロセッサが、ループのような中〜粗粒度のタスクを表す場合はクラスタが割り当てられる。C–node cvにプロセッサ p を割り当てた場合、cvに相当するタスクが p によって実行され、cvについてプライベートな変数が p の属するクラスタのメモリに配置されることを意味する。同様に、C–node cvにクラスタ Cを割り当てた場合、cv の表すタスクが C内のプロセッサによって実行され、cv についてプライベートな変数が Cのメモリに配置されることを意味する。

DPG 上でデータ分割ならびにプロセッサ割当を行う際は、C–node および D–node をグルーピングし、各グループをクラスタに割り当てる。プロセッサへの割当は、各グループをクラスタに割り当てた後に、グループ内の C–node をグルーピングし、得られたグループをクラスタ内のプロセッサに割り当てることにより行う。C–node, D–node をグルーピングする際は、i) 並列性, ii) 通信コスト, iii) ネットワークトポロジといった要素が考慮される。ここでは、図7左の DPG を用いて、データ分割およびプロセッサ割当の例を示す。データ分割とプロセッサ割当は同時に行われるのが理想であるが、本稿では簡単のためにプロセッサ割当は何らかのアルゴリズムにより、すでに完了しているものとする。

C–node 群 $\{1,2,5\}$, $\{3,4\}$, $\{6,7,8\}$ を、それぞれクラスタ C_1, C_2, C_3に割り当てたものとする。D–node B, Dは、それぞれクラスタ C_2, C_3からしかアクセスされないため、それぞれ C_2, C_3に割り当てれば良い。また、D–node A にクラスタ C_1からコスト 20 でアクセスされ、C_2からコスト 40 でアクセスされる。したがって D–node$A$ はクラスタ C_2に割り当てる。D–node Cについても同様で、クラスタ C_1に割り当てると良い。データ分割は図7右のようになる。これは、D–node Cの表すアクセスパターンを有する変数、ならびに C–node1, 2, 3 に相当するタスクについてプライベートな変数は、クラスタ C_1に割り当てられることを意味している。クラスタ C_2, C_3についても同様である。リモートアクセスとなる変数アクセスは、図の太い枝に相当する変数アクセスである。その通信コストは、枝につけられたアクセスコストの関数となる。

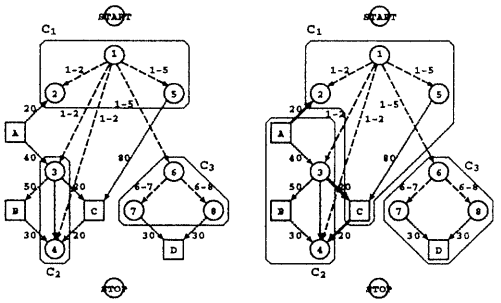

図 7: DPG 上での分割

この例では、クラスタの割当を重複がないように行ったが、必ずしもそのようにする必要はない。例えば、あるC–node cvが並列実行可能なループを表しており、その実行にp_{req}個のプロセッサが要求されているものとする。各クラスタ内のプロセッサ数$p_{cluster}$よりp_{req}が大きい場合、cvには$\lceil \frac{p_{req}}{p_{cluster}} \rceil$個のクラスタを割り当てる必要があろう。ひとつのD–nodeに複数のクラスタが重複して割り当てられた場合、D–nodeに相当する変数群は、割り当てられたクラスタのうちのひとつのメモリに配置するか、もしくはそれらのクラスタのメモリに適当に割り振られて配置される必要がある。

4.2 HDPG 上でのデータ分割

本節ではHDPGにおけるデータ分割について述べる。ここではひとまず各タスクの参照するデータが静的に解析できるものと仮定する。

HDPGでデータ分割, プロセッサ／クラスタ割当を行う場合は、上層から順に下層に向けて、各層でDPGと同様の割当を行う。クラスタ割当に関しては、任意の$\hat{hcv}$にクラスタ$\{C_i\}$を割り当てた場合、$\hat{hcv}$の任意の子のタスク$hcv \in C(\hat{hcv})$、および$\hat{hcv}$に内包される任意のD–node $hdv \in HDV_{\hat{hcv}}$には、$\{C_i\}$に含まれるひとつ、もしくは複数のクラスタが割り当てられる。割当が全て完了した時点で、親子関係で葉にあたるC–node hcvの親$\hat{hcv} = p(hcv)$に内包されるD–nodeを全て調べると、各変数の配置先クラスタが得られる。この時、変数の配置先が複数存在するケースが存在し得る。例えば図4において、$L1$において$T3$, $T4$のD–nodeをクラスタC_1、$L2$をクラスタC_2に割り当てたものとする。この時、$L2$の$T3$, $T4$はC_2に割り当てられることになる。結果、T_3, T_4はC_1, C_2のいずれにも割り当てられることになり得る。データのマイグレーションを考える場合はこのような状況は許されるが、そうでない場合は変数の配置クラスタを一意に決める必要がある。

5 むすび

本稿では、データ分割を意識したタスクグラフとしてデータ分割グラフ (DPG)/ 階層データ分割グラフ (HDPG)を提案した。HDPGは、並列性の階層性や、メモリ階層をはじめとする計算機アーキテクチャの階層性との親和性が高い。データ分割, プロセッサ割当のいずれの問題も、DPG/HDPG上ではグラフの節点をグルーピングする問題として取り扱われる。

今後の予定としては、データ分割アルゴリズムの開発, ならびに自動並列化コンパイラ Parafrase–2 のフロントエンド部の実装が挙げられる。

参考文献

[1] C.D.Polychronopoulos, M.B.Girkar, M.R.Highighat, C.L.Lee, B.Leung, and D.A.Schouten: "Parafrase–2: An environment for parallelizing, partitioning, synchronizing and scheduling programs on multiprocessors", Proc. of the 1989 Int'l Conf. on Parallel Processing, St.Charles, IL, Aug. 1989. Penn State.

[2] M.B.Girkar: "Functional Parallelism Theoretical Foundations and Implementations", Ph.D Thesis No.1182, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, 1991.

[3] M.Gupta and P.Banerjee: "Demonstration of automatic data partitioning techniques for parallelizing compilers on multicomputers", IEEE Trans. on Parallel and Distributed Systems, vol.3, pp.179–193, Mar. 1992.

[4] T.S.Chen and J.P.Sheu: "Communication-Free Data Allocation Techniques for Parallelizing Compilers on Multicomputers", Proc. of Int'l Conf. on Parallel Processing, 1993.

[5] P.D.Holvland and L.M.Ni: "A Model for Automatic Data Partitioning", Proc. of Int'l Conf. on Parallel Processing, 1993.

[6] K.Hwang: "Advanced Computer Architecture", McGraw-Hill, 1993.

[7] 中西恒夫, 城和貴, 福田晃, 荒木啓二郎: "DPG: データ分割グラフ", 情処研報, 94–ARC–104–16, pp.121–128, Jan. 1994.

[8] 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", 情処学会論文誌, vol.35, no.4, pp.513–521, Apr. 1994.