

専用目的コンパイラ開発用 3D-1 並列化中間言語とその処理系

田村光雄 前川仁孝 笠原博徳 成田誠之助

早稲田大学理工学部

1. はじめに

最近では、各種科学技術計算の処理時間短縮のために並列計算機の利用が一般的になっている。それに伴い、並列処理に全く興味のないユーザが、並列処理マシンを使用する機会が増えている。これらのユーザにとっては並列処理のためのプログラムのチューニング、さらには高級言語を使ったプログラミングすらわずらわしい場合も少なくない。このような状況を考慮すると並列処理マシンを使う際に、ユーザには並列処理を意識させず、ユーザが理解しやすく使いやすい方式、例えばグラフィック等によってプログラム入力が行える方式が望ましいと考えられる。そのためには、グラフィックプログラムが入力されると、自動的に効率良い並列化マシンコードを生成できる各種アプリケーション専用のコンパイラを開発することが重要である。本稿では、このような機能を持った専用目的コンパイラを各種のアプリケーションに対して開発することを容易にする並列化中間言語と、その処理系について述べる。

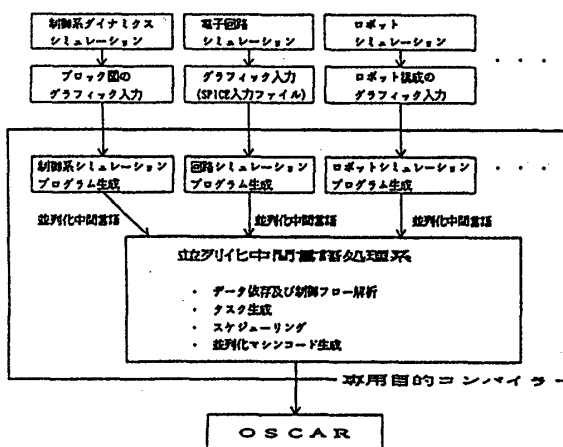


図1. 専用目的コンパイラ

2. 各種アプリケーション専用コンパイラ

図1に専用目的コンパイラの構成を示す。例えば制御系シミュレーション専用並列化コンパイラでは、まずグラフィックエディタを用いて作成された制御系のブロック図が入力されると、コンパイラが並列化中間言語（3.で詳述）で記述されたシミュレーションプログラム（制御系の動特性を数値積分等を用いて解析するプログラム）を自動的に生成する。次に並列化中間言語処理系が制御フロー解析、タスク生成、データ依存解析、

スケジューリング、並列化マシンコード生成などを行う。その詳細に関しては4.で述べる。

3. 並列化中間言語

並列化中間言語は、並列性記述等を行うために従来の逐次処理用の中間言語[3][4]を拡張した形式となっている。シミュレーションプログラムをプリプロセッサ等を用いて高級言語で記述するのではなく、中間言語で生成するのは、字句解析、構文解析等が簡略化できること、また専用目的コンパイラ製作者はコンパイラ及びアプリケーションに関する詳しい知識を持っているため中間言語レベルで質の高い並列化コードを生成できるためである。次に、本中間言語の特徴について述べる。

第1の特徴はユーザが並列処理の単位であるタスクを記述できることである。図2の基本ブロック1にタスク定義の例を示す。この中間言語では `stend` (strong task end), `tend` (task end), `:=` (代入及び task end) の記述によってその直後でタスクを区切ることを指定する。この3種のタスクバウンダリの指定のうち `tend`, `:=` によって定義されたタスクは弱いタスクバウンダリの指定であり、中間言語処理系による自動的なタスク分割あるいは融合等の最適化の対象となる。

また `stend` は強い指定を意味し、処理系はそのタスクを分割あるいは融合の対象から除外する。

ただし、現在は、従来のループ並列化等では並列処理できないようなアプリケーションを処理対象としているため、本中間言語は、近細粒度（複数命令レベル）タスクの処理を中心に設計されている。

第2の特徴は、`if then [ else ]`, `repeat until`, `while do` の3つの構造化された制御文を持つということである。これらの制御文の使用は大規模プログラムの製作を容易にし、さらに制御フローやデータ依存の解析も容易にする。

図2の例では `while do` によって基本ブロック1、`repeat until` 構造、基本ブロック3の部分の繰り返し、`repeat until` によって基本ブロック2が繰り返し実行される。図2から分かるように各基本ブロックは、制御文を含まない複数の近細粒度タスクから構成される。

第3の特徴は、図2の5行目の記述のように1つのオペコードが3個以上のオペランドを持つことができる点である。また、複数オペランドの適用範囲を広げるために、8行目のように `++` (内積) といった複合命令もサポートされている。複数オペ

ランドは、10行目や13行目のようなベクトル演算のような同じ演算の繰り返しの記述を簡単にし、さらに並列性の抽出も容易にする。

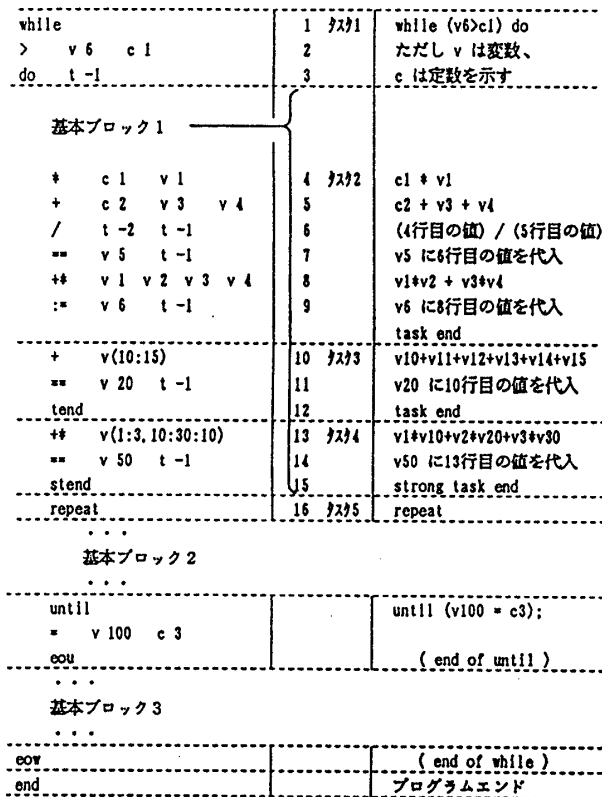


図2. 並列化中間言語プログラム例

4. 並列化中間言語処理系

次に並列化中間言語の処理系で行われる処理内容について述べる。

① 制御フロー及びデータ依存解析

中間言語プログラムの制御フローを解析し各制御構造内の基本ブロックを検出する。本並列処理手法では各基本ブロック内の近細粒度タスクを並列処理の対象とする。次に各基本ブロック内で定義されているタスク間のデータ依存を解析し、データ依存グラフを生成する。

② リストラクチャリング

タスク間の逆依存、出力依存を除去しタスク間の並列性を向上させるためにスカラリネーミング等のリストラクチャリングを行う。

③ タスク処理時間推定

タスク処理時間を推定する。データ依存グラフとタスク処理時間からなる情報であるタスクグラフは、タスク生成及びスケジューリングで使用する。

④ タスク生成

タスク生成においてはまず最初に中間言語により記述されたプログラム中での指定に基づきタスクを生成する。次に := あるいは tend で指定されたタスクの内、図3に示すように並列処理可能な多数オペランドを持つ命令が存在すると、並列実行可能なタスクに自動分割する。

また、各基本ブロックのタスクグラフ中に特定のパターンを持つ部分タスクグラフを発見すると、その部分の最小並列処理時間と逐次処理時間を推定し、逐次処理時間の方が小さければ、その部分グラフを構成するタスク集合を1タスクに融合する[5]。

⑤ スケジューリング

生成されたタスクの各プロセッサの割り当てをデータ転送を考慮したヒューリスティックなスケジューリング・アルゴリズムであるCP/DT/MISF[5]を用いて決定する。

⑥ 並列化マシンコード生成

スケジューリング結果をもとに各プロセッサごとにマシンコードを生成する。各タスクには他プロセッサ上のタスクへのデータ転送、同期のためのフラグ転送、フラグチェックのコードが付加される。また冗長な同期コードはタスク間のデータ依存及びスケジューリング結果を考慮して削除される。

タスク分割前

```
+ v(1:50)
:= v100 t-1
```

タスク分割後

```
+ v(1:10) + v(11:20) + v(21:30) + v(31:40) + v(41:50)
:= v201 t-1 := v202 t-1 := v203 t-1 := v204 t-1 := v205 t-1

+ v(201:205)
:= v100 t-1
```

図3. タスク分割

5. むすび

本稿では、各種アプリケーションの自動的な並列処理を目的とした専用目的コンパイラ開発のための並列化中間言語と、その処理系について述べた。この中間言語を用いて回路シミュレーション[6]、制御系シミュレーション用等の専用目的コンパイラをすでに開発しており、それらはOSCAR[7]上で動作している。

参考文献

- [1] 笠原：「並列処理技術」，コロナ社，(1991)
- [2] P. Antognetti, G. Massobrio：「Semiconductor Device Modeling with SPICE」，McGraw-Hill Book Company (1988)
- [3] A. V. Aho, J. D. Ullman：「コンパイラ」，培風館 (1986)
- [4] 佐々政孝：「プログラミング言語処理系」，岩波書店 (1989)
- [5] H. Kasahara et. al.：「Parallel Processing of Near Fine Grain Tasks Using Static Scheduling on OSCAR」，IEEE Computer Society (1991)
- [6] 前川等：「OSCAR上での直接法を用いた回路シミュレーションの並列処理」，情処全大, (1992-03)
- [7] 笠原等：「OSCARのアーキテクチャ」，信学論 J71-D, 8 (1988-08)