超並列C言語NCXのMIMD計算機用コンパイラの構成

4T－10

川合 史朗，日高 宗一郎，相田 仁，齊藤 忠夫
東京大学 工学部

1 はじめに

並列C言語NCXは、データ並列の記述が容易な拡張C言語である*。我々はNCXで記述された並列プログラムをMIMD/分散メモリ型計算機上で実行するためのコンパイラの開発に取り組んでいる[2]。

NCXは言語仕様上SIMDセマンティクスを採用しているが、最低限必要な同期ポイントで同期をとりさえすれば、あとは各プロセッサが独立にコードを実行することが可能である。しかし、NCXではプロセッサ間通信が遠隔参照というimplicit/synchronousな形で記述されるのに対し、ターゲットマシン上ではexplicitに通信プリミティブを用いることになるため、プロセッサ間通信をどのようにコンパイルするかが高いパフォーマンスを得るための鍵となる。

2 NCXの並列性記述

NCXでは、並列に動作させる仮想プロセッサ(VP)の集合をフィールドとして定義する。フィールドは複数定義できるが、ある時点で有効なフィールドは一つだけである。フィールドを切り替えるための構文としてin文が用意されている。

仮想プロセッサのトポロジと物理プロセッサ(PP)の構成とを対応づける方法はNCXの仕様には含まれておらず、処理系依存となっている。

変数は必ずいずれかのフィールドに属する。他のVPの変数を参照する場合は、

variable-name@(*index*)

というシンタクスを用いる。ここで、*index*はVPを表すインデックスである。その時点でアクティブなフィールド以外のフィールドに属する変数にも自由にアクセスできる。

言語仕様の詳細については、文献[1]を参照されたい。

```
field matrix(N, N) on mesh;
double a, b, c on matrix;

   in matrix(i, j) {
      int k;
      c = 0.0;
      for (k = 0; k < N; k++)
         c += a@(i, k) * b@(k, j);
   }
```

図1: NCXによる記述例

3 コンパイラの構成

3.1 方針

一般にPPより多くのVPが要求されると考えられるので、複数のVPをひとつのPPがエミュレートすることになる。VPからPPへのマッピングを工夫すれば、遠隔参照がPP内でのメモリコピーに置き換えられ、またコストの高い全PP間での同期を使わずにすむケースも出てくる。

解析を容易にするため、各ブロックおよび関数に対して次のような属性を考える。

VP-closed 内部に次の処理を含まないもの：VP間通信、VP-closedでない関数の呼び出し、外部変数の参照、フィールドの切り替え。

PP-closed 内部に次の処理を含まないもの：PP間通信、PP-closedでない関数の呼び出し、外部変数の参照、PP間で同期を必要とするようなフィールドの切り替え。

VP-closed, PP-closedでないブロック・関数をそれぞれVP-open, PP-openと呼ぶ。

VP-closedであればPP-closedでもあり、そのような関数は同期をとる必要がなく、VPエミュレーションループの中から通常のCの関数と同じように呼び出すことができる。

VP-openかつPP-openである関数を呼び出す場合は、一旦全PPで同期をとってから関数を呼び出すことになる。また、VP-openかつPP-openであるブロックを含むループはループの終了条件判定に全PP間での通信が必要になるため、オーバヘッドの大きい処理となる。

" Compiling for Massively-Parallel C Language NCX on MIMD Computers"

Shiro Kawai, Souichirou Hidaka, Hitoshi Aida and Tadao Saito

Faculty of Engineering, The University of Tokyo

*NCXは文部省重点領域研究「超並列原理に基づく情報処理基本体系」で開発された言語である

VP-open であっても PP-closed なブロック・関数の場合は、VP エミュレーションループを切らなければならないが、コストの高い PP 間通信を必要としない。したがって、VP-open であるブロック・関数をできるだけ PP-closed にするようにマッピングを決定することが必要である。

3.2 パス構成

どの関数が VP-closed になるかは全ての関数を見ないと分からず、マッピングもそれまで決定できない。そこで、コンパイラのパスは大きく二つに分けられる (図 2)。

第 1 パス：VP レベルコンパイル

このパスでは、構文解析、意味解析および VP レベルのコンパイルが行なわれる。この段階で、モジュール内で宣言されている全てのフィールドの情報と、それらの間で VP 間通信が行なわれるかどうかという情報が抽出される。

PP へのマッピングに依存しない範囲で VP 間通信、同期点が挿入された中間表現が出力される。

中間パス：マッピングの決定

次に、宣言された全てのフィールドおよび VP 間通信の情報とから、VP をどのように PP にマップするかを決定する。但し、プログラマによるマッピングの指定があればそちらを優先する。

第 2 パス：PP レベルコンパイル

マッピングが決定したら、前半パスで得られた中間表現を walk し、PP-open/closed 属性を決定する。

PP 間通信が必要な箇所には通信ライブラリの呼び出しを挿入する。また、VP-closed なブロックに関して VP エミュレーションループを生成する。

ここで一旦、ターゲットマシン上の C 言語に落し、ターゲットマシン上の C コンパイラを起動して実行イメージを生成する。

第 1 パスについては、重点領域研究班で並行して進めている SIMD マシン用コンパイラの実装と重複する部分が多いため、共通のコードを使うことを検討している。

モジュール外の関数呼び出しについては、open/closed の属性がわからないため、VP-open であるとして処理する。ただし全てを VP-open であるとすると効率が悪いため、関数の宣言時に型修飾子 const を付加することにより、その関数が VP-closed であることをプログラマが明示的に指定できるようにする。

実行の効率を考えると、関数の種別はもっと細かくプログラマが指定できるほうが良い。#pragma や (gcc のような)__attribute__ 構文を導入することも考えられる。

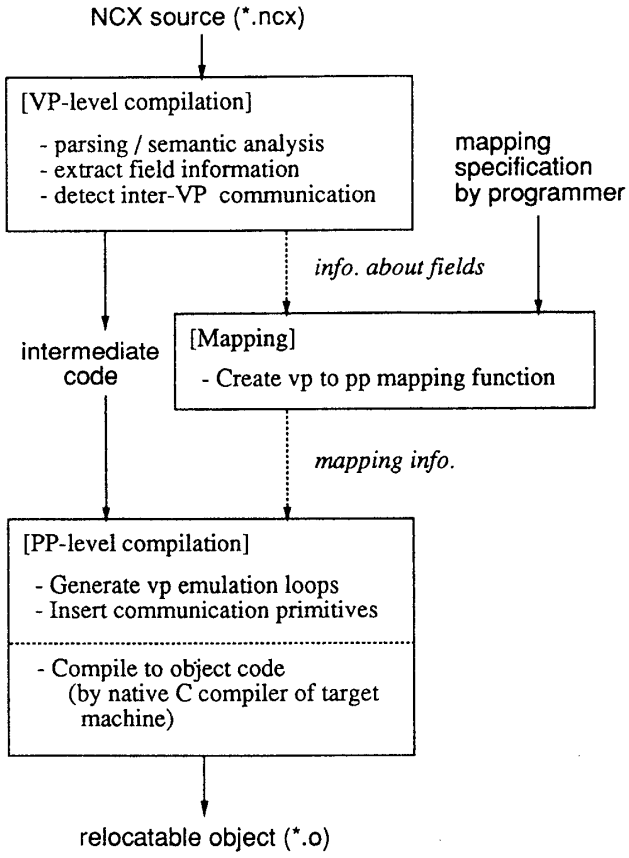

図 2: コンパイラの構成

4 拡張

複数のモジュールにソースが分割されている場合、効率のよいマッピングを行なうためには全モジュールについてフィールド情報と通信パターンを集める必要があるが、現在の構成はそのような場合を考慮していない。

マッピングはコンパイラの第 2 パスに影響するため、モジュール毎に独立して図 2の全パスを通すことは難しくなる。文献[3] のように、コンパイラドライバがモジュール間の依存関係を解析するようにする必要があるであろう。

参考文献

[1] 超並列 C 言語 NCX 仕様書 version 3, September 1993.

[2] 相田 仁, 川合 史朗, 日高 宗一郎. AP-1000 上での NCX コンパイラの実現. 文部省重点領域研究「超並列原理に基づく情報処理基本体系」第 4 回シンポジウム予稿集, pp. 2-112–2-126, March 1994.

[3] Jon Loeliger and Robert Metzger. Developing An Interprocedural Optimizing Compiler. *SIGPLAN Notices*, 29(4):41–48, April 1994.