超並列計算機用NCX言語処理系の試作

1L－7

田井秀樹　松本正義　酒寄保隆　山下義行　中田育男
筑波大学

1　はじめに

データ並列言語は、複雑になりがちな並列計算機用のプログラムを、より簡潔に記述できるという特徴がある。本研究では、データ並列型C言語NCX[1] 分散メモリ型MIMD並列計算機用に最適なコードを生成する処理系の実装を目標としている。現段階では、プロセッサ内での処理の効率化をはかり、全体の性能向上を目指した。特に本稿では、処理系の生成するオブジェクトコードを、並列計算機PILOT3上で評価した結果について述べる。

1.1　超並列計算機CP-PACS

CP-PACSは、筑波大学で開発された、分散メモリ型超並列計算機である[2]。2048台のノードプロセッサは、3次元HXB(Hyper-Crossbar Network)で結合されており、ネットワークの半径が小さいという特徴を持つ。メッセージ転送には、リモートDMA機構が提供されており、OS内の通信バッファを介さずに、高速な立ち上げ、高スループットの転送を可能としている。

本稿では都合により、CP-PACSの小規模構成版ともいえる、PILOT3上での実験結果を示す。PILOT3では、最大8台のノードプロセッサが利用可能である。

2　NCXコンパイラ

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」の一環として設計された、拡張C言語である[1]。本研究では、NCXプログラムからSPMD型のCプログラムへの変換を行うトランスレータとして、NCX言語のコンパイラの実装を進めている。

処理系には、トランスレータに加えて、通信関数やメモリ管理ルーチンを含んだランタイムライブラリが必要になる。本実装では、ランタイムライブラリとして、New Humpshire大学で開発されているC*コンパイラ[3]用のものを元に、NCX用に改編している。

"A NCX compiler for massively parallel computer"
Hideki TAI, Masayoshi MATSUMOTO, Yasutaka SAKAYORI, Yoshiyuki YAMASHITA and Ikuo NAKATA
(University of Tsukuba)

2.1　最適化

本研究で実装をすすめているNCXトランスレータは、いわゆる素朴な変換をベースとして、いくつかの最適化を考えている。実行効率を得るために最も効果のある最適化には、以下のようなものがあると考えている。

1. **変数のベクトル化の抑止**[4]
   素朴な変換では、配列へ置き換える必要のある変数を、スカラー変数で代用。
2. **隣接通信における無駄なメモリコピーの削減**[4]
   オーバーラップエリアを利用したコードの生成。
3. **通信のブロック化**
   ループ内の通信をループ外に追い出す。
4. **仮想エミュレーションループの繰り返し省略**
   インアクティブな仮想プロセッサの繰り返しの省略[5]。
5. **通信関数の処理の効率化**
   通信パターンの再利用。CP-PACSのリモートDMA転送機能の利用。

データ並列言語のコンパイラに関しては、通信の最適化を扱ったものが多いが、プロセッサ内での処理に関するものは少ない。本研究での実装では、プロセッサ内での処理の高速化を考慮することで、実行性能の向上を目指している。ここで挙げたもののうち、1, 2, 4はプロセッサ内の処理の最適化である。

本稿では、1と2のPILOT3での評価結果を示す。以下最適化の内容について簡単に述べる。

1. 変数のベクトル化の抑止

仮想プロセッサ集合の実プロセッサへの分割に従い、素朴な変換では、仮想プロセッサ上の変数('a')は配列変数('vec_a')へとベクトル化される。その結果、出力プログラムには次のようなコードが良く現れる。

```
for (VPi = 0; VPi < QUOTA; VPi++) {
/* 仮想エミュレーションループ */
  vec_a[VPi] = ... ;
        :
  ... = vec_a[VPi];
}
```

仮想エミュレーションループ内で変数の定義と使用が閉じている場合には、このようなベクトル化は不要である。この最適化では、冗長な 'a' の 'vec_a' へのベクトル化を抑止し、次のようなコードを生成する。

```
for (VPi = 0; VPi < QUOTA; VPi++) {
  int tmp;
  tmp = ... ;
      :
  ... = tmp;
}
```

2. 隣接通信におけるメモリコピーの削減

NCXプログラム中では、変数名を '@(i,j)' と修飾することで他の仮想プロセッサ上の変数を参照することができる。

```
/* NCXプログラム */
a = a@(i-1) + a@(i+1);
```

このようなプログラムに関して以前の著者らのNCXコンパイラは、'a@(i-1)' と 'a@(i+1)' のコピーを作り、そこに必要な通信をしてから、代入文の実行を行なうようなコードを生成していた。著者らが参考にしたC*コンパイラ [3] の実装でも同様である。

しかし、このようなプログラムでは、'a' に相当するメモリ領域にオーバーラップエリアを持たせて、コピーを行なわないようにするのが望ましい。本研究では、通信関数の処理を遅延させることにより、コピーを1つ、もしくは全くコピーを作らないような方法を提案している [4]。

3 性能評価

実験には次の二つのNCXプログラムを用いた。

Sieve N以下の素数の数えあげ。

変数のベクトル化を抑止する最適化が適用可能。

Jacobi Jacobi(連立一次方程式の解法) 型の計算。隣接通信を多用。

隣接通信においてメモリコピーを削減する最適化が適用可能。

これらのプログラムを、PILOT3上で実行した結果を図1, 2に示す。いずれも、適用可能な最適化を行なった場合と、行なわなかった場合で計測した。グラフの縦軸は、最適化を行いプロセッサ数2台で実行した場合のスピードアップ率を2とした場合のスピードアップ率である。

いずれの最適化も、実行時間が50%〜60%に短縮され、大きな効果が得られている。特にPILOT3のように、ノードプロセッサが高速[1]なものになると、本稿で挙げたようなメモリアクセスに関する最適化の効果が、より顕著に表れているものと思われる。

[1]ピーク性能 300Mflops

図 1: Sieve

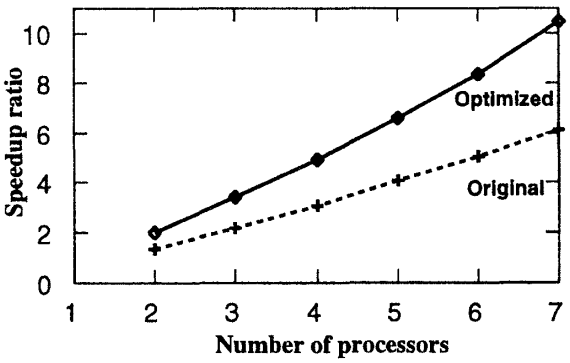
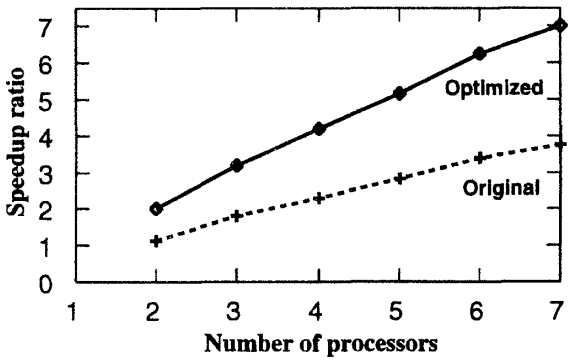


図 2: Jacobi



4 まとめ

超並列計算機においては、通信の最適化が重要であるが、全体の実行効率の向上のためには、本稿で挙げたようなプロセッサ内での処理の最適化も必要であると考え、NCXコンパイラでの最適化のケースを示した。

CP-PACSやPILOT3における実装としては、リモートDMA機構を用いた通信ライブラリを開発することが挙げられるが、これは今後の課題である。

参考文献

[1] 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」. 超並列 C 言語 NCX 言語仕様書 Version3.

[2] 中澤喜三郎, 中村宏, 朴泰祐. 超並列計算機 CP-PACS のアーキテクチャ. 情報処理, Vol. 37, No. 1, pp. 18–28, January 1996.

[3] A. Lapadula and K. Herold. A retargetable C* compiler and run-time library for mesh-connected MIMD multicomputers. Technical Report 92-15, University of New Hampshire, 1992.

[4] 田井秀樹, 山下義行, 中田育男. データ並列言語 NCX の分散メモリ MIMD 並列計算機用コンパイラ. 情報処理学会第 53 回全国大会講演論文集 (分冊 1), pp. 333–334, 1996.

[5] 貴島寿郎, 湯淺太一. Vp リストを用いたデータ並列言語のアクティビティ制御. 情報処理学会研究報告 96-PRO-10, Vol. 96, No. 107, pp. 25–30, October 1996.