3E－1

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

田井秀樹　山下義行　中田育男
筑波大学

1　はじめに

大規模で柔軟な並列計算機のアーキテクチャとして、分散メモリ型MIMD並列計算機が注目されている。このアーキテクチャでは、必然的にプログラミングは繁雑になってしまうという問題がある。一方、大規模な並列計算に対し、データ並列というプログラミングパラダイムが有効であると言われている[2]。本研究では、データ並列型言語として設計された拡張C言語であるNCX[1]を取りあげ、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。本稿では、現在進めている処理系の実装状況と、オブジェクトコードの実行効率、今後実装を行なう最適化について述べる。

2　データ並列型C言語NCX

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]。その主な特徴としては、仮想プロセッサを主体としたSIMD型と呼ばれる実行モデルを持つことが挙げられる。本稿では、NCX言語についての詳しい説明は省略する。

3　NCXコンパイラ

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。オブジェクトコードには、ランタイムライブラリの呼び出しが含まれる。ランタイムライブラリには、各種の通信パターンに応じた通信関数の他、メモリ管理ルーチンなどが含まれている。現在コンパイラの実装は、言語仕様をサブセットに限定して、[2][3]に基づいた変換を行なうものが完成している。この変換はナイーブなもので、最適化は考慮していない。そのため、処理系が生成するオブジェクトコードの実行効率は図1のようなものとなっている。

実験は、8台の実プロセッサ構成の分散メモリ型MIMD並列計算機(Pilot1†)で、**sieve**(素数の数えあげ), **gauss**(逆行列の計算), **jacobi**(ヤコビ型の計算)の三種類の問題について、各々**手書き**, **ncx**(NCXトランスレータの出力), **serial**(逐次プログラム)といったテストプログラムを作成し、実行時間の計測を行なった。

問題	serial	ncx	手書き
sieve	9.88sec	3.28sec 2.65 倍	1.35sec 7.32 倍
gauss	213sec	54.7sec 3.89 倍	22.8sec 9.34 倍
jacobi	8.42sec	6.06sec 1.39 倍	0.605sec 13.9 倍

図1: 実行時間(下段はスピードアップ率)

4　性能低下の原因とその最適化

実験結果とトランスレータの出力から、いくつかオーバーヘッドの要因となる部分が明らかになった。それらのうち本稿では、(1)変数のベクトル化による局所参照性の低下、(2)トランスレータが生成した一時変数へのメモリコピーといったものについて説明し、その解決案を述べる。

4.1 変数のベクトル化による局所参照性の低下

通常、実プロセッサと仮想プロセッサの対応は一対多になるので、各実プロセッサは、複数の仮想プロセッサの実行を担当することになる。そのためトランスレータは、仮想プロセッサ上の各変数を、実プロセッサが担当する仮想プロセッサ台数(QUOTA)分の配列へと変換する(図2)。

```
in F(i) {
  int t;
  t = a + 1;
  b = t;
}
    ↓
for (VPi = 0; VPi < QUOTA; ++VPi) {
  int t[QUOTA];
  t[VPi] = a[VPi] + 1;
  b[VPi] = t[VPi];
}
```

図2: 変数のベクトル化

変換後の**for**ループは、仮想エミュレーションループと呼ぶもので、実プロセッサが担当している仮想プロセッサの実行をエミュレートするためのループである。しかしながら、必ずしも変数のベクトル化を行なう必要はない。図2の例では、変数tをベクトル化する必要はなく、スカラー値としておいても問題はない。ベクトル化不要の条件を示す。

条件1 (変数のベクトル化不要の条件) 変数aの生存区間が一つのエミュレーションループ内に含まれるならば、そのエ

"A compiler of the data-parallel language NCX for distributed memory MIMD parallel computer"
Hideki TAI, Yoshiyuki YAMASHITA and Ikuo NAKATA
(University of Tsukuba)

† 筑波大学計算物理学研究センター

ミュレーションループに出現する変数 a は、ベクトル化する必要はない。 □

前述の実験の sieve に対して現在のトランスレータが出力するコードについて、この最適化を手で施し再実験を行なった結果、実行時間が約 75%に短縮された。

4.2 一時変数との間のメモリコピー

NCX トランスレータは、通信を必要とする式に対して次のような変換を行なう。

```
a = b@(i-1, j) + b@(i, j+1);
        ↓
NCX_GridRead(tmp1, b, 2, -1,  0);
NCX_GridRead(tmp2, b, 2,  0, +1);
for (VPi = 0; VPi < QUOTA; ++VPi)
  a[VPi] = tmp1[VPi] + tmp2[VPi];
```

通信関数 NCX_GridRead は仮想プロセッサ間の隣接通信のエミュレートを行なうが、その引数は、1 つ目が代入先の変数、2 つ目が参照される変数、3 つ目が変数が属するフィールドの次元、4 つ目以降が通信の方向を表す。ところが、実プロセッサが行なう処理では、多くのローカルなメモリ間のコピーが行なわれることになる (図 3)。

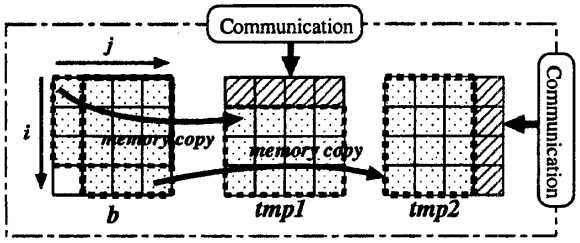

図 3: 通信文の実プロセッサによる実行の様子

このような隣接通信は、本来ならば変数 b に余分なオーバーラップエリアを持たせ、そこに他の実プロセッサから受信するデータを格納し、メモリコピーは避けるのが望ましい。以下で、そのような実行コードを生成するための方法を提案する。

一般的には、メモリコピーを行なわないために、コンパイラがあらかじめ変数にどのようなオーバーラップエリアを持たせればよいかといった事を解析しなければならない。本稿で提案する方法では、コンパイラが特殊な解析をすることなく、実行時にオーバーラップエリアの解析や割り当てを行なうことが可能である。具体的には、本節のはじめにあげた NCX プログラムに対して、トランスレータは次のようなコードを生成することになる。

```
NCX_GridRead_W(  DEFER,&tmp1,&b,2,-1, 0);
NCX_GridRead_W(NODEFER,&tmp2,&b,2, 0,+1);
for (VPi0 = 0; VPi0 < QUOTA_i; ++VPi0) {
 for (VPi1 = 0; VPi1 < QUOTA_j; ++VPi1) {
  a.data[VPi0*a.size[0] + VPi1] =
   tmp1.data[VPi0*tmp1.size[1] + VPi1] +
   tmp2.data[VPi0*tmp2.size[1] + VPi1];
 }
}
```

ただし、フィールド上の変数 a,b,tmp1,tmp2 は次のような構造体として宣言するように変換するものとする。

```
struct NCX_FieldVar_float {
 float *data; /* Array of variables on field */
 int rank;    /* Rank of field to which */
              /*       the variable belongs */
 int size[];  /* Size of each dimension */
}
```

コンパイラは、連続する NCX_GridRead_W の呼び出しのうち、最後のものだけに NODEFER フラグを指定し、それ以外には、DEFER フラグを指定するだけである。

オーバーラップエリアの実現は、通信関数 NCX_GridRead_W が引き受ける。DEFER フラグが指定されていた場合は、どのようなオーバーラップエリアが必要かといった情報を記録するだけで、実際の通信やメモリコピーは行なわない。NODEFER フラグが指定されると、それまでに記録しておいた情報をもとに、適切なオーバーラップエリアを確保し、必要な通信を行なう。通信に関しては、C*[5] における通信回数を削減する最適化 [4] を適用することができる。最終的に、tmp1,tmp2, b といった変数の各メンバを図 4 のように設定する。

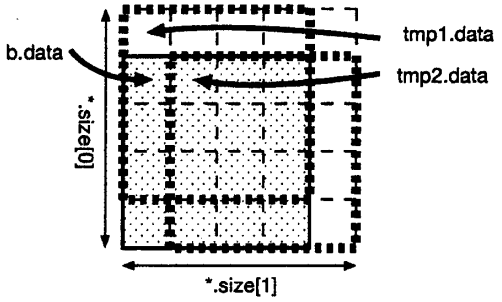

図 4: 本方法によるデータ割り付け

このように本方法では、処理系へ負担をかけることなくランタイムライブラリの工夫のみで、メモリコピーによる実行効率低下を回避することができる。

5 まとめ

現在実装を進めている NCX トランスレータが生成するオブジェクトコードはまだ十分な実行効率ではないが、その問題点を明らかにし、解決方法を示した。今後は、本稿であげた最適化を処理系へ実装するとともに、本稿で挙げたもの以外の最適化についての研究をすすめなければならない。

参考文献

[1] "超並列 C 言語 NCX 言語仕様書 Version3", 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」

[2] Philip J. Hatcher and Michael J. Quinn. *DATA-PARALLEL PROGRAMMING ON MIMD COMPUTERS*, The MIT Press.

[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] Stephen W. Chappelow, Philip J. Hatcher and James R. Mason. Optimizing Data-Parallel stencil computations in a portable framework, *Languages, compilers and Run-Time systems for scalable computers*, KLUWER ACADEMIC PUBLISHERS, 1996.

[5] J. Frankel. *A reference description of the C* language*, Technical Report TR-253, Thinking Machines Corporation, Cambridge, MA, 1991.