4T－9 並列化コンパイラにおける プロセッサ間非同期通信命令を用いた通信コストの最適化

南里 豪志† 佐藤周行‡ 島崎 眞昭‡
†九州大学情報工学科 ‡九州大学大型計算機センター

1. Introduction

分散メモリ型並列計算機は並列性及び拡張性に優れた並列計算機であるが、プロセッサ間通信時間の処理時間に占める割合が大きいという問題がある。特に同期通信では、送信側と受信側のプロセッサが処理を中断してデータを転送するため、通信時間による影響が大きくなる。

これに対してカリフォルニア大学で開発されたSPMD型並列C言語split-c[1]は、split-phase assignmentというプロセッサ間非同期通信を用意しており、通信を計算と並行して行なうことが出来る。

しかしsplit-cは低水準な言語であり、通信や同期を明示的に指定しなければならない。そのため、プログラムの作成時に通信の最適化をマニュアルで行なわなければならず、大きなプログラムの最適化は困難である。

そこで本研究では、高水準言語プログラムに対する解析により通信時間の最適化を行なう自動最適化コンパイラを設計する。最適化の方針としては、通信と計算の並行性増加、及び、大量通信命令を利用した一括通信を挙げる。

2. split-phase assignment

split-cでは大域データ空間は、プロセッサ次元と局所メモリ次元の2次元構造で扱われる。この大域データ空間中のあるアドレスに対してassignmentを行なうことによりプロセッサ間通信が行なわれる。

この通信を要求フェーズとデータ転送フェーズに分けて実行するのがsplit-phase assignment命令である。送信プロセッサがこの命令を実行すると、assignmentするアドレスを所有しているプロセッサに対して送信を要求する。その後送信プロセッサは次の処理に移る。実際にデータが転送されるのは、送信側と受信側のプロセッサの空き時間である。

また、データの転送終了は明示的に知らされないため、受信側でデータを利用する前に送、受信プロセッサ間で同期をとって転送の確認をする必要がある。

このようにして通信をおこなうため、転送要求を出してから転送の確認を行なうまでの間隔が長いほど、通信と計算の並行度が上がる。また、split-cにはsplit-phase assignmentを大量の連続領域に対して一括して行なう命令が用意されており、これを用いることによって転送確認の同期回数を減らすことが出来る。

3. 自動最適化を行なうコンパイラの構築

3.1 対象言語

前節のようにsplit-cは低水準であり、通信が最適なコードをマニュアルで作成するのは困難である。そこで、高水準言語によるプログラムに対し解析を加えてsplit-phase assignmentを用いた通信最適コードを生成するコンパイラを構築する。

このコンパイラへの入力言語となる言語は、通信や同期を明示的に指示しない。しかし、並列化の方法を示すため、プログラム中で計算分割とデータ分割を指定しなければならない。

このような分割の指定方法として、High Performance Fortran (HPF)[2]の命令を利用する。HPFは、データ分割命令DISTRIBUTE、及び、計算分割命令FORALLを提供している。DISTRIBUTEはデータをブロックや行毎に仮想プロセッサに割り当てる方法を指示する。また、FORALLは配列の一部または全部に対する同時代入を行なう。

3.2 最適化方針

3.2.1 基本ブロック単位の通信

split-assignmentは、通信の開始から転送終了の確認までに実行される処理が多いほど通信と計算が並行に実行される可能性が高くなる。そのため、転送されるデータの生成直後に通信を開始し、利用される直前に確認を行なうようにすればよい。

しかし、転送確認の同期のため、頻繁にsplit-assignment命令を用いると処理速度の向上があまり見られなくなる。そこで、データを生成する基本ブロックの後にまとめて送信する。また、利用する基本ブロックの前で同期をとって確認することにより、基本ブロック単位での通信を行なう。

このためには、基本ブロックを単位とするデータの生成、利用に関する情報が必要である。任意の基本ブロッ

Communication Optimization for Parallelizing Compiler Using Asynchronous Communication
Takeshi Nanri, Hiroyuki Sato, Masaaki Shimasaki
Department of Computer Science and Communication Engineering, Kyushu Univ.
Computer Center, Kyushu Univ.

ク n におけるこれらの情報は、

outword exposed define

以降のブロックに影響するデータの define

outword exposed use

以前のブロックで define されたデータの use

を用いて以下のように定義される。

DEF(n)

n 中で outword exposed define される変数 v の集合

USE(n)

n 中で outword exposed use される変数 v の集合

次に、基本ブロックをノードとするフローグラフを作成する。ノード n で define された変数 v について、フローグラフのどのパスを通っても同じデータを用いるノード n' の集合を B_DU(n, v) と表す。

全ての変数 $v \in$ DEF(n) のうち、通信の必要があるものについて、B_DU(n, v) の同じものを一括して n の最後に送信する。また、各ノードの先頭で同期をとり、USE(n) の受け取りを確認する。これにより、基本ブロック単位の通信を実現できる。

3.2.2 データフロー解析の利用

ループ内の配列参照について前節の方法をとると、各ネストレベルの1イタレーション毎に独立した通信が行なわれる。しかし実際には、複数のイタレーションやネストレベルで一括して通信を行なう方が効率が良くなる場合が多い。そこで、ループについてはデータフロー解析を用いて最適化を行なう。

データフロー解析とは、アクセス関数が整定数及びループ変数による線形式であるプログラムに対し、データの移動に関する依存関係を調べる解析である。データフロー解析では、配列の参照、ループ境界、配列の各次元の上限についての関数を用いる。解析は、ループ変数の係数を行列として計算を加えて行ない、解析の結果はあるイタレーションで read される値について、生成する箇所を返すような関数として与えられる。

データフロー解析によって得られる情報は、逐次実行する際のイタレーションのみである。これらから通信の必要な依存関係を調べ、通信の最適化を行なうには、生成、利用するプロセッサとデータのアドレスが必要である。これには、各イタレーションとそれを実行するプロセッサの関係、及び、各データとそれを所有するプロセッサの関係についての情報を用いる。入力ソース中でこれらをそれぞれ計算分割、データ分割として指定することにより、データフロー解析を用いて最適化を行なえる。

3.3 実験

図1の LU 分解プログラムについて、split-phase assignment を用いて並列化及び通信の最適化を行なう。

```
for i1 = 0 toN do
  for i2 = i1 + 1 toN do
    X[i2][i1] = X[i2][i1]/X[i1][i1]
    for i3 = i1 + 1 toN do
      X[i2][i3] = X[i2][i3] − X[i2][i1] * X[i1][i3]
```

図 1: Sequential version of LU decomposition.

まず、N 個の仮想プロセッサを用意し、k 番目の仮想プロセッサに配列 X の k 番目の列を所有させる。この場合、i_3 ループの各イタレーションを並列に実行できる。

このプログラムを解析すると、i_3 ループの最初のイタレーションで生成されるデータが i_1 ループの次のイタレーションで $X[i_1][i_2]$ と $X[i_1][i_1]$ のアクセス に用いられることがわかる。

i_3 ループの最初のイタレーションを実行するのは i_1+1 番目の仮想プロセッサであり、送信されるデータも i_1+1 番目の列で、i_1+1 番目の仮想プロセッサが所有している。そこでこのプロセッサが i_3 ループの最初のイタレーションを計算した直後に、bulk assignment によって i_1+1 番目の列を一括して送るようにすれば、通信を効率良く行なえる。

このような最適化を行なった結果、Thinking Machines 社の CM-5(16 プロセッサ) 上で 1000 × 1000 の行列の並列 LU 分解を、18% 短縮でき、台数効果が 9.80 から 11.96 に上がった。

4. Conclusion

LU 分解を例にあげ、split-c の split-phase assignment や、bulk assignment を用いることによって通信時間を短縮できることを示した。また、基本ブロックやループに対する通信の最適化を用いて、より高級な言語から split-phase assignment による通信最適コードを生成するコンパイラを設計した。

参考文献

[1] David E.Culler *et al. Introduction to Split-C.* University of California, Berkeley, November 9, 1993

[2] High Performance Fortran Forum. *High Performance Fortran Language Specification, version 1.0* Center for Research on Parallel Computation, Rice University, Houston, TX, May 3, 1993

[3] Saman P.Amarasinghe and Monica S.Lam: "Communication Optimization and Code Generation for Distributed Memory Machines." In *Proceedings of the SIGPLAN '93 Conference on Programming Language Design and Implementation, June 1993.*