コンパイル時未定義ループ不変値を添字式に持つループの最内ループ並列化の1手法

4T－6

小笠原 武史, 小松 秀昭
日本アイ・ビー・エム（株）東京基礎研究所

1 はじめに

データ並列言語の典型的なSPMDコード生成では、ループのインデックス空間をowner computes規則に基づいて分割するコードが生成される。コンパイラによる最適化として、分割された計算に必要なデータを可能な限り外側ループあるいはループの外でプリフェッチさせることで、より多くの計算を同期的通信なしに実行するコードを生成することがあげられる[Tse93]。また同期的通信によって、pipelined computationなどSPMDコードの並列実行様式が決定される。

元のプログラムの意味を壊さずにプリフェッチを行なってプログラムをゆるい同期の下に並列に実行するためにはデータのdef-useの関係を調べることが必要である。そのためにコンパイル時のデータ依存解析が重要である。プリフェッチするデータをどのプロセッサが所有し、それら所有プロセッサが定義するどの時点の値を読み込むのかを解析する。

しかしデータ依存解析によってある配列参照についてデータ依存が複数の可能性を持つと判明した場合、その配列参照の値を定義するステートメントが1つに決まらない。そのため、データ依存のdef-use関係に基づいた単純なメッセージベクタ化ができない。

SPMDプログラムは1つのプログラムが全プロセッサで使用されるため、プロセッサごとの場合分けコードを挿入するとコードが増大し易い。詳細な場合分けによる最適化は、コードサイズを爆発させる可能性が大きい。そのためコードを大幅に増やさずに並列性を引き出す手法が現実的である。

本稿では、コードを大幅には増やさずに、複数のデータ依存可能性を伴う最内ループが持つ並列性を引き出す手法を提案する。

2 複数のデータ依存可能性を持つループの並列性

本稿では次の疑似コードが示すように、右辺に登場するループ不変なxが原因で複数のデータ依存が発生しているループを扱う。左辺のa*#IV.d+bが登場する次元と右辺のxが登場する次元は同じものとする。

$do\ \#IV.1 = 1, ub.1$
⋮
$do\ \#IV.d = 1, ub.d$
$array(.., a * \#IV.d + b, ..) = ..array(.., x, ..)..$
$enddo$
⋮
$enddo$

xが登場する次元がPn個のブロック状に分割されているとする。それ以外の次元の添字式が左辺と右辺で同じであるとして、最内ループに注目すると、array(..,x,..)の値が定義される時期は、a, b, ub.d, xの関係に応じて変わる。array(..,x,..)が最内ループで定義される場合、array(..,x..)の値を最内ループの前でプリフェッチし、owner computes ruleに基づいて分割されたインデックスセットでループを実行できる。最内ループで定義される場合、定義前の値を使用するプロセッサはプリフェッチし、定義後の値を使用するプロセッサはarray(..,x,..)を所有するプロセッサから定義後の値を受信し、分割されたインデックスセットでループを実行する。ループ実行時間は2/Pnに短縮される。

3 従来研究でのアプローチ

こうしたコンパイル時に未定義なループ不変値が1つならばindex set splitingによってデータ依存を1通りにすることができるが、ループで定義しない場合に実行しないループの無駄なコードを挿入することになる。また問題の右辺が複数登場する場合にそれら不変値の大小関係が不明なため、コンパイル時にループ変形できず対応できない。

以下では本稿が提案する最内ループパイプライン化手法について述べる。

4 最内ループパイプライン化

4.1 対象ループ

最内ループ（1重ループならそれ自身）のインデックス変数が#IV.dであるとき、次の条件を同時に満たすループを本最適化の対象とする。

A method for exploiting parallelism of innermost loop with compile-time undescidable data dependences
Takeshi Ogasawara and Hideaki Komatsu
Tokyo Research Laboratory, IBM Japan,Ltd.

表 1: 受信プロセッサセットの計算

条件	定義前の値を使用するプロセッサ	定義後の値を使用するプロセッサ
$mod(x-b,\|a\|)!=0$	左辺代入するすべてのプロセッサ	
$mod(x-b,\|a\|)=0 \wedge a>0 \wedge a+b>x$	左辺代入するすべてのプロセッサ	
$mod(x-b,\|a\|)=0 \wedge a>0 \wedge x>a*ub.d+b$	左辺代入するすべてのプロセッサ	
$mod(x-b,\|a\|)=0 \wedge a>0 \wedge a+b \leq x \leq a*ub.d+b$	1 から (x-b)/a までの一部を分配されたインデックスセットとして持つプロセッサ	(x-b)/a+1 から ub.d までの一部を分配されたインデックスセットとして持つプロセッサ
$mod(x-b,\|a\|)=0 \wedge a<0 \wedge a*ub.d+b>x$	左辺代入するすべてのプロセッサ	
$mod(x-b,\|a\|)=0 \wedge a<0 \wedge x>a+b$	左辺代入するすべてのプロセッサ	
$mod(x-b,\|a\|)=0 \wedge a<0 \wedge a*ub.d+b \leq x \leq a+b$	(x-b)/a から ub.d までの一部を分配されたインデックスセットとして持つプロセッサ	1 から (x-b)/a-1 までの一部を分配されたインデックスセットとして持つプロセッサ

- 左辺の配列の m 番目の添字に a*#IV.d+b を含む (a!=0, b はループ不変数)
- 左辺の配列の m 番目の次元はプロセッサ間にブロック上に分配
- ある右辺に左辺と同じ配列の参照で m 番目の添字に x (x はループ不変変数) を含む

4.2 受信プロセッサセットの計算

owner computes rule に基づいて最内ループのインデックスセットを分割する ($LocalIndexSet.d$)。
$compute\_lis(LocalIndexSet.d, LhsSubsExp.d, ArrayDecomposition)$
次にループ定義前の値を使用するプロセッサ ($RecvPre$) と、ループ定義後の値を使用するプロセッサ ($RecvPost$) とを計算する (受信プロセッサセット $RecvProcSet$)。$LoopInvarVar$ は x のことであり、$LhsSubsExp.d$ は a*#IV.d+b のことである。
$compute\_pre\_post(LoopInvarVar, LocalIndexSet.d, LhsSubsExp.d, ReceiveProcSet)$
表 1に、受信プロセッサセットをまとめた。

4.3 受信プロセッサセットに対する通信

最内ループよりも外側の d-1 重ループのインデックス変数は固定とみなせ、また x も決まっているので、右辺に登場する左辺と同じ配列で m 次元目に x を持つ配列要素は、配列の分割情報を利用してその所有者プロセッサが求められる。所有者プロセッサを px とする。プロセッサ px は最内ループ前で先ほど求めた $RecvPre$ に array(..,x..) を送信し、$RecvPre$ は受信する。またプロセッサ px は最内ループの後で先ほど求めた $RecvPost$ に array(..,x..) を送信し、$RecvPost$ は受信する。

最内ループの外側のループインデックス変数の値を $OuterLoopIVs$、array(..,x,..) の全次元の添字式を $RhsSubsExp$ とすると、通信は、pre_send、post_send という実行時ライブラリの中で行なわれる。
$pre\_send(OuterLoopIVs, RhsSubsExp, RecvPre)$
$post\_send(OuterLoopIVs, RhsSubsExp, RecvPost)$

4.4 最内ループパイプライン化 SPMD コード

以上により、最内ループパイプライン化された疑似 SPMD コードはつぎのようになる。ただし lis.lbp, lis.ubp は compute_lis で初期された $LocalIndexSet.d$ の値で、それぞれ分配されたインデックスセットの下限、上限である。

$call\ compute\_lis(LocalIndexSet.d, LhsSubsExp.d, ArrayDecomposition)$
$call\ compute\_pre\_post(LoopInvarVar, LocalIndexSet.d, LhsSubsExp.d, ReceiveProcSet)$
$call\ pre\_send(OuterLoopIVs, RhsSubsExp, RecvPre)$
$do\ \#IV.d = lis.lb(p), lis.ub(p)$
$a(..,a*\#IV.d+b,..) = ..a(...,x,...)..$
$enddo$
$call\ post\_send(OuterLoopIVs, RhsSubsExp, RecvPost)$

5 おわりに

複数のデータ依存という状況で最内ループの並列性を活かす、最内ループパイプライン化の手法を提案した。現在 High Performance Fortran (HPF) コンパイラに実装中である。

参考文献

[Tse93] Chau-Wen Tseng. An Optimizing FORTRAN D Compiler for MIMD Distributed Memory Machines. Technical Report CRPC-TR93291-S, Rice University, Jan 93.