2H-07

LLVM IRコードにおける並列化指示文の挿入方式についての検討

神宮 健吾 † 大津 金光 †† 大川 猛 †† 横田 隆史 ††
† 宇都宮大学工学部情報工学科 †† 宇都宮大学大学院工学研究科情報システム科学専攻

1 はじめに

近年，マルチコアプロセッサの登場により，プログラムの並列化による実行能力の向上が行われるようになった．しかしながらプログラムの並列化は非常に困難であるため，この困難な部分を全て自動で処理する自動並列化について研究が進められてきた．それらの多くは高水準言語で記述されたソースコードを対象に並列化を行うが，この場合ソースコードが参照できないプログラムを並列化できないという問題がある．

これを解決するために本プロジェクトでは，プログラムのバイナリコードを対象に解析，最適化，並列化を行うマルチコアプロセッサ向けバイナリ変換システムについて研究している．本システムでは，異なる機械語で記述されたバイナリコードであっても同様に実行速度向上を達成できるように，LLVMを用いて機械語によらない解析，最適化，並列化処理を実装する．

LLVMとは，モジュール化された再利用可能なコンパイラおよびツールチェーン技術の集まりであり，任意のソースコードをLLVM IR (Intermediate Representation) へ変換して分析，最適化するため，ソースコードやターゲットマシンから独立した分析，最適化が実現できるという特徴を持つ．

上述の理由から本システムでは，LLVM IRコードを対象に並列化を行う必要があるが，LLVM IRコードを並列化するための手法はあまり整備されておらず，並列化処理の開発が非常に困難であるという問題がある．

そこで本研究では，より簡単にLLVM IRを並列化する新たな手法として，指示文によるLLVM IRコードの並列化を実現する．

これに関する研究として，ソースコードに記述された指示文をLLVM IRで表す手法がある [1]．この研究はLLVM IRで指示文を表すという目的を持つが，本研究は，実行速度向上のための新たな指示文をLLVM IRに実装するという目的を持つ．このため関連研究と本研究で実装する並列化手法では，生成されるLLVM IRコードが大きく異なる．

2 提案手法

本研究では，LLVM IRにおける並列化指示文と，その指示文に基づき並列化を行う並列化処理を新しく実装する．これにより，並列化指示文の挿入による，並列LLVM IRコードの生成を可能にする．

A Study on Insertion Method of Parallelization Directive in LLVM IR Code

†Kengo Jingu, ††Ootsu Kanemitsu, ††Ohkawa Takeshi, ††Yokota Takashi,
Department of Information Science, Faculty of Engineering, Utsunomiya University (†)
Department of Information Systems Science, Graduate School of Engineering, Utsunomiya University (††)

```
1 def int_directive : Intrinsic<[],[llvm_metadata_ty],
    [IntrArgMemOnly],"llvm.directive">;
```

図 1: 追加した組込み関数の定義

2.1 指示文の定義

LLVMではLLVM IRとして命令，組込み関数，型を追加することができる．指示文をLLVM IRで用いる為には，このいずれかとして定義する必要がある．

組込み関数とは，本体を持たない関数のことで，組込み関数で指示文を表す場合は組込み関数の名前，型，引数，プロパティなどを定義する．

一方，命令や型で指示文を表す場合は追加するLLVM IRの名前や，追加するLLVM IRのレクサーやパーサなどを定義する．

命令や型で表す方法は，組込み関数で表す方法に比べ多くのことを定義する必要がある．また，組込み関数であれば，関数の引数に受け取る値を用いて，各指示文が識別可能である．そのため本研究では，組込み関数により指示文を表すことにする．

以上の検討を基に作成した組込み関数の定義を図1に示す．図1では”Intrinsic<...>”でこの組込み関数の返値，引数，プロパティ，名前を定義している．この組込み関数では引数にllvm_metadata_tyを受け取る．

llvm_metadata_tyとは，メタデータを表す型であり，メタデータとは，コードに関する追加情報を命令に添付するために用いられるデータである．メタデータは複数の数字や文字列を扱うことができる．本研究では各指示文の識別にこのメタデータを用いる．メタデータを用いることで，指示文の識別やオプションの指定を1つのメタデータで行うような実装が可能となる．

2.2 指示文に基づく並列化処理

指示文に基づく並列化処理の実装として，#pragma omp parallel forという，ループの並列化を行うOpenMP指示文と対応する並列化処理を実装する．並列化処理の実装にあたり，並列実行ではLLVM IRコードが，どのように記述されているかを調査した．

図2のC++ソースコードから生成される並列LLVM IRコードを調査した結果，#pragma omp parallel forから生成される並列LLVM IRコードでは，OpenMP Runtime Libraryの関数__kmpc_fork_callと，__kmpc_for_static_init_4を呼び出し，並列実行しているということが分かった．

__kmpc_fork_call関数はフォークとジョインを行う．この関数は引数に関数を受け取りこれを実行するスレッドを生成して，その関数を並列に実行する．そのため，ループを並列実行するためには，それを関数に変換する必要がある．

__kmpc_for_static_init_4関数は，ループの各反復を

現在のスレッドが実行するために，必要となるループの上限，下限などの値を計算する．この計算では，__kmpc_for_static_init_4 関数が引数に受け取るループの上限，下限，インクリメントなどを用いて行われ，計算結果は引数のポインタへ返される．

LLVM では，__kmpc_for_static_init_4 関数に与えるループの下限を常に 0，インクリメントを常に 1 として，ループの上限だけを計算により求めるている．この計算で求まるループの上限は，元のループの反復回数と同じ値となる．本研究でもこれにならいループの下限を常に 0，インクリメントを常に 1 とし，ループの上限だけを計算で求めるように実装する．

以上をまとめると，並列化処理では，(1) ループを関数に変換し，(2)__kmpc_for_static_init_4 関数に渡すループの上限を計算する，という処理が必要となる．

3 動作確認

提案手法の動作を確認するために，並列化指示文の記述された LLVM IR コードを用意し，そのコードに対して並列化処理を実行した．得られた LLVM IR コードが，並列実行可能であれば並列処理が正常に動作したもの判断する．

並列化処理の対象には，図 2 の逐次 LLVM IR コードを使用した．逐次 LLVM IR コードの生成には，最適化オプションを”-O0”とし，さらに，OpenMP による並列化を行わないようオプションを指定した．これにより，実装した並列化処理以外による LLVM IR コードへの変更が加えられないようにする．

変換後の LLVM IR コードは図 3 のようになる．図 3 をみると行 3 に並列化指示文があり，その直後にフォークとジョインを行う__kmpc_fork_call が呼び出されている．この関数呼び出しの引数には，ループが変換されて生成される関数.omp_outlined. が与えられてる．

関数.omp_outlined. では，行 19 で__kmpc_for_static_init_4 を呼び出し，各スレッドで用いるループの上限，下限などを取得している．この関数の第 6 引数はループの上限を表すが，ここには%22 が与えられている．図 3 の行 16，17 をみると%22 は，%19 からロードした値が入っていると分かる．詳細な計算は省略してあるが，%34 は図 2 のループ変数の初期値 (n+m+j) を，%37 は図 2 のループの終了条件 (m+j) を，%18 は図 2 のループのインクリメント (k+j+m) をそれぞれ保持している．これを基に行 9〜行 14 の計算を追っていくと，$\%19 = (((m+j)-(n+m+j)+(k+j+m))/(k+j+m))-1$ となる．この式の変数に図 2 でそれぞれの変数に与えられている値を代入すると，$\%19 = (12/2)-1 = 5$ となり，ループの上限がループの反復回数と等しくなっていることが分かる．

このことから，__kmpc_for_static_init_4 関数には正しい引数が与えられており，.omp_outlined. は並列実行しても正しい値を返すといえる．よって，並列化処理が生成する図 3 の LLVM IR コードは並列実行可能である．

```
1  #include<stdio.h>
2  int n = 10, m = 0;
3  int main(){
4     int i; int j=0, k=2; int a[n];
5     #pragma omp parallel for
6     for(i=n+m+j;i>=m+j;i-=(k+j+m)){
7           a[i] = i; printf("%d\n",i);
8     }
9  }
```

図 2: 並列化可能なループを持つ C++プログラム

```
1  define i32 @main() #1 {
2     [...]
3     call void @llvm.directive(metadata !0)
4     call void (%ident_t*, i32, void (i32*, i32*, ...)
          *, ...) @__kmpc_fork_call(%ident_t* @1, i32
           5, void (i32*, i32*, ...)* bitcast (void (
          i32*, i32*, i32*, i32*, i32, i32*, i32*)* @.
          omp_outlined. to void (i32*, i32*, ...)*),
          i32* %"1", i32* %"6", i32 %"15", i32* %"
          16", i32* %"21")
5     [...]
6  }
7  define void @.omp_outlined.(i32*, i32*, i32*, i32
       *, i32, i32*, i32*) {
8     [...]
9     %43 = sub nsw i32 %34, %37
10    %44 = load i32, i32* %18, align 4
11    %45 = add nsw i32 %43, %44
12    %46 = sdiv i32 %45, %44
13    %47 = sub nsw i32 %46, 1
14    store i32 %47, i32* %19, align 4
15    [...]
16    %50 = load i32, i32* %19, align 4
17    store i32 %50, i32* %22, align 4
18    [...]
19    call void @__kmpc_for_static_init_4(%ident_t*
          @0, i32 %48, i32 34, i32* %24, i32* %21,
          i32* %22, i32* %23, i32 1, i32 1)
20    [...]
21 }
```

図 3: 並列化処理により生成される LLVM IR コード

4 おわりに

本論文では，指示文による LLVM IR コードの並列化手法を提案した．この提案は，LLVM IR コードに挿入された並列化指示文を基に並列化を行うため，並列 LLVM IR コードの生成が指示文の挿入で実現できるという貢献を持つ．

本論文では一部の OpenMP 指示文に対応した並列化処理の実装を行ったが，並列化によるプログラムの実行能力向上にはより多くの並列性を利用できるようにする必要がある．そのため，LLVM IR で表現できる指示文の種類を増やすことが今後の課題として挙げられる．

謝辞

本研究は，一部 JSPS 科研費 15K00068，16K00068，17K00072 の助成による．

参考文献

[1] Xinmin Tian, Hideki Saito, Ernesto Su. LLVM Framework and IR Extensions for Parallelization, SIMD Vectorization and Offloading. LLVM Compiler Infrastructure in HPC (LLVM-HPC), 2016.