线程级猜测并行系统代码自动生成工具的设计与实现

王家龙 刘艳红 沈 立
(国防科技大学高性能计算国家重点实验室 长沙 410073)
(国防科技大学计算机学院 长沙 410073)

摘 要 虽然线程级猜测(Thread Level Speculation,TLS)执行机制可以简化多线程编程模型接口,并能获得较高的性能加速,但其并行程序的开发仍然比较困难。面向一个高效的软件TLS模型HEUSPEC,研究了代码自动生成工具C2H的设计与实现方法。具体包括3部分内容:首先,为HEUSPEC设计简单的标注语句,标注出可并行段的一些特征;其次,提出将标注语句和可并行段转换为猜测线程函数的算法;最后,设计生成HEUSPEC并行代码的算法。该方法已在开源编译器Clang上实现。面向Rodinia,OmpScr等基准程序的测试结果表明,C2H能够将带有简单标注语句的串行C代码转换为HEUSPEC并行代码,且其性能与手工编写的HEUSPEC并行代码的性能十分接近。

关键词 线程级猜测,HEUSPEC,源到源编译器,标注语句,Clang
中图法分类号 TP303 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.11.018

Design and Implementation of Automatic Code Generator for TLS System

WANG Jia-long LIU Yan-hong SHEN Li
(State Key Laboratory of High Performance Computing,National University of Defense Technology,Changsha 410073,China)
(School of Computer,National University of Defense Technology,Changsha 410073,China)

Abstract Although thread level speculation (TLS) mechanism can simplify the interface of multi-thread programming model and achieve high performance speedup,the development of its parallel program is still a tough task. Oriented to an effective software TLS model HEUSPEC,the design and implementation of automatic code generator C2H,which has three components. First,simple directives are designed for HEUSPEC,which indicates some key characteristics of parallel regions. Second,an algorithm is proposed to convert directives and parallel regions to speculative functions. At last,an algorithm is designed to generate HEUSPEC parallel codes. This method has been implemented on an open source compiler——Clang. Experimental results oriented to typical benchmarks from Rodinia and OmpSrc indicate that C2H can convert serial C codes with simple directives to HEUSPEC parallel codes,and the performance of automatic-generated parallel codes is very close to the codes developed by programmers manually.

Keywords Thread level speculation,HEUSPEC,Source-to-source compiler,Directive,Clang

1 引言

线程级猜测执行(Thread Level Speculation,TLS)的基本思想[1]是在原本顺序执行的程序中标识出多个可并发的区域,并通过专门的冲突检测/解决机制确保并行执行结果的正确性。与传统的非猜测的线程级并行机制相比,TLS既能提供简洁的多线程编程模型,将程序员从繁琐的多线程程序设计中解脱出来,又可以带来较高的性能加速,充分利用多核/众核处理器中集成的丰富计算资源。

TLS系统可以通过硬件、软件以及软硬结合等不同的方式实现。由于实现硬件TLS系统的开销过大,软件TLS系统逐渐成为了近年来人们关注的重点。HEUSPEC[2]是一种高效的软件TLS模型,其核心是HEUSPEC运行时库。该运行时库提供了执行多线程猜测必需的全部基本机制,包括冲突检测/解决、猜测变量的读/写控制、猜测任务的派生/分派/确认/回滚等,以及对"归并和"和"动态数组"两种类型冲突变量的特殊支持机制。这些基本机制都是以库函数的形式实现的,可分为4个层次,如图1所示。

使用HEUSPEC模型开发的应用包含两类线程:非猜测执行的主线程和猜测执行的猜测线程。运行过程中,主线程不做计算,主要完成控制活动,猜测线程完成计算。猜测线程只在第一次猜测读数据以及结果确认时通过消息与主线程通信,猜测线程之间不产生任何通信。HEUSPEC的工作原理如图2所示。

到稿日期:2016-10-26 返修日期:2016-12-15 本文受国家自然科学基金项目(61272143,61472431)资助。
王家龙(1989-),男,硕士生,主要研究方向为计算机体系结构,E-mail:15616167476@163.com(通信作者);**刘艳红**(1982-),女,硕士生,主要研究方向为计算机体系结构,E-mail:yanhongliu@nudt.edu.cn;**沈 立**(1978-),男,博士生导师,主要研究方向为计算机体系结构、并行计算,E-mail:lishen@nudt.edu.cn。

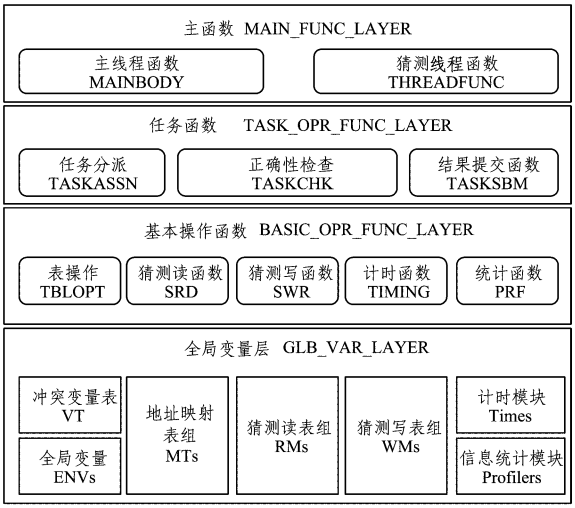

图 1　HEUSPEC 模型的层级结构

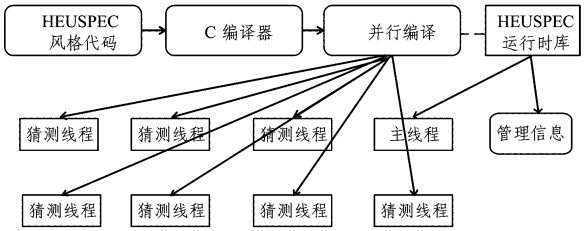

图 2　HEUSPEC 模型的工作原理

为了实现更高的性能加速，HEUSPEC 实现了一系列优化技术，包括通过启发式值预测技术[3]和跨线程取技术[3]降低误猜率，通过动态粒度调整技术[2-3]优化全局开销，通过乱序确认技术[3]降低控制开销。与传统的软件 TLS 模型相比，HEUSPEC 具有编程模型简单、运行性能高等优点。

图 3 通过一个结算 Pi 值的代码描述了 HEUSPEC 代码规范。图 3(a)是计算 Pi 值的串行 C 代码，其中的 for 循环是一个可并行段。图 3(b)是它对应的 HEUSPEC 风格代码，本文将其称为 HEUSPEC 内部代码。可以看出，HEUSPEC 内部代码的结构十分清晰，它通过在串行 C 代码的基础上增加以下 3 部分内容而得到：1)TLS 头文件引用和猜测线程函数声明，如图 3(b)中①所示；2)冲突变量声明和主线程函数调用，如图 3(b)中②所示；3)猜测线程函数体，如图 3(b)中③所示，包括冲突变量和私有变量声明、为冲突变量分配线程私有存储空间、冲突变量初始化、任务粒度初始化、猜测计算等内容。在任务执行过程中，若猜测任务对某个冲突变量进行猜测读/写操作，则还需要调用猜测读/写函数(TLS_spec_read/write)。需要强调的是，猜测线程函数体的结构十分规整，虚线框内的部分完成初始化、猜测线程与主线程通信等功能，对所有可并行段而言都是固定不变的。这简化了程序员编写猜测线程函数的工作，也降低了自动生成 HEUSPEC 内部代码的难度。为进一步减少编写 HEUSPEC 内部代码的工作量，HEUSPEC 主线程函数 TLS_main_body()被定义为 HEUSPEC 库函数，负责猜测线程的创建和管理。编写 HEUSPEC 内部代码时只需调用主线程函数即可，调用时需要将猜测线程函数名和线程任务数作为参数传递给它。

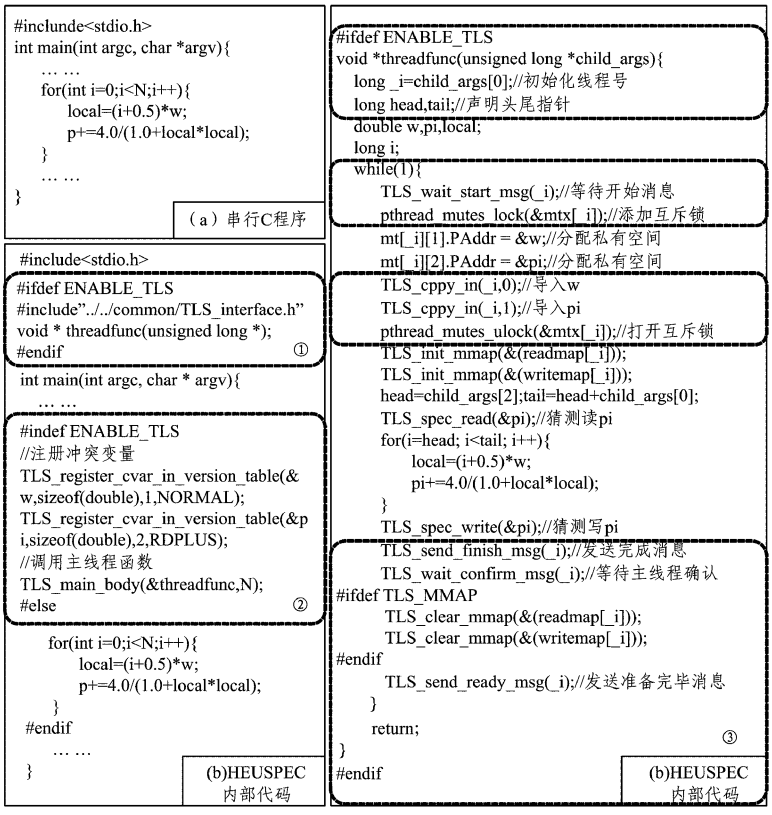

图 3　HEUSPEC 代码

虽然程序员可以手工地将 C 代码转换为 HEUSPEC 代码，但这需要花费大量的时间和精力，而且还会面临以下两个挑战。

(1)手工编写 HEUSPEC 内部代码的工作量较大。其要求程序员找出串行程序中的可并行段，以及其中的冲突变量、冲突变量类型、私有变量、私有变量类型；同时还需要程序员

熟悉 HEUSPEC 库函数及其使用方法。

(2)分析哪些冲突变量在运行中可能会出现猜测读/写的难度较大。程序员在编写猜测线程函数时,需要分析判断源程序中哪些冲突变量是需要进行猜测读/写的猜测变量,并调用猜测读/写函数对猜测变量进行猜测读/写。而对于一些复杂的串行程序,准确判断猜测变量是一件十分困难的工作。

因此,本文通过分析总结 HEUSPEC 内部代码的特点,为 HEUSPEC 定义了一套更加简洁的编程接口,程序员只需标注出串行 C 代码中可能的并行段以及其中可能的冲突变量即可。在此基础上,本文基于 Clang 编译器前端设计并实现了一个能够自动将标注后的串行 C 代码转换为 HEUSPEC 内部代码的源到源编译器 C2H,大大降低了 HEUSPEC 编程模型的复杂度。测试结果表明,使用 C2H 开发的 HEUSPEC 内部代码既能保证结果的正确性,又能获得接近手工编写的 HEUSPEC 内部代码的性能。

本文第 2 节介绍了 HEUSPEC 标注的格式和 C2H 源到源编译器框架;第 3 节介绍了 Clang 编译器前端,以及基于 Clang 实现 C2H 自动转换工具的方法;第 4 节对使用 C2H 生成的 HEUSPEC 内部代码进行了正确性测试和性能测试;最后总结全文。

2 编译器设计

为了实现串行 C 代码到 HEUSPEC 内部代码的自动转换,本文通过分析 HEUSPEC 内部代码的结构特点,设计了描述并行段及其中猜测变量的标注语句,并实现了一个由串行 C 代码到 HEUSPEC 内部代码的自动转换工具 C2H。该工具已作为一个模块集成在 Clang 编译器内。本节将介绍辅助自动转换的标注语句以及 C2H 的基本框架。

2.1 标注语句

为了标注串行 C 代码中的并行段及其冲突变量,C2H 提供了一条简单的标注语句,其格式如下:

#pragma heuspec for(ITER_NUMBER) CVT

其中,#pragma huspec 表示该语句为 HEUSPEC 编程模型专用;for 表示可并行段的循环类型,以后还会扩展出对 while 和 do…while 循环的支持;ITER_NUMBER 为循环的迭代次数或计算循环次数的表达式;冲突变量表(Conflict VariableTable,CVT)记录了这个可并行段中的冲突变量及其相关信息。每个冲突变量的描述如下(其中尖括号中的内容是可选的)。

(cvar1,〈type〉,〈DArraySize〉)

(cvar2,〈type〉,〈DArraySize〉)

…

其中,cvar1,cvar2,…为冲突变量名;type 为冲突变量类型,目前分为普通(NORMAL)、可规约(RDPLUS)、动态数组(DARRAY);DArraySize 指明冲突变量是标量还是向量,若该值为 1,则表示该变量为标量;否则,其为向量。当冲突变量为向量时,DArraySize 表示其元素个数。当然,CVT 表也可以为空,即可并行段中没有任何冲突变量。

冲突变量的识别是多线程程序设计时必须解决的一个关键问题。程序员可以在标注语句中列出可并行段中所有的冲突变量,也可以由 C2H 工具以保守方式识别出冲突变量,即若无法确定一个变量是否是线程的私有变量,则将它视作多个线程间的冲突变量。但这部分工作并不是本文的重点,因此不再赘述。

在使用标注语句时,将标注语句添加在串行代码中的可并行段之前即可。为了与 HEUSPEC 内部代码区分,将添加了标注语句后的串行 C 代码称为 HEUSPEC C 代码。显然,HEUSPEC C 代码大大简化了 HEUSPEC 编程模型的使用过程,使得程序员编写多线程程序的作用更加简单。C2H 工具的工作就是自动地将 HEUSPEC C 代码转换为 HEUSPEC 内部代码。

2.2 C2H 编译器框架

图 4 给出了本文设计并实现的 HEUSPEC C 代码到 HEUSPEC 内部代码转换工具 C2H 的总体结构。我们已将它作为一个独立的功能模块集成在 Clang 编译器中,即图中黑框中的 C2H 转换模块。这相当于对 Clang 编译器的功能进行了扩展,这样设计的好处是不仅不会对 Clang 的其他功能模块造成影响,而且还方便以后对 C2H 的功能进行优化或增强。

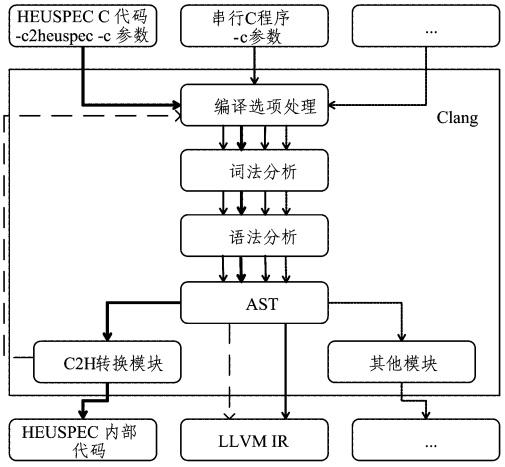

图 4 C2H 总体框架

为了指示 Clang 编译器进行 HEUSPEC 源代码转换,本文为 Clang 编译器增加了一个新的编译选项,即-c2heuspec。用户可以通过该编译选项来触发 HEUSPEC C 代码到 HEUSPEC 内部代码的代码自动转换功能,输出 HEUSPEC 内部代码,也可以继续编译 HEUSPEC 内部代码,生成 LLVM 中间代码(IR)。在不使用该编译选项的情况下,Clang 编译器将按照之前定义的方式对源程序进行处理,直接生成 LLVM IR。这为用户提供了一种灵活的使用方式。

C2H 转换模块的工作可分为 3 个部分:

(1)标注语句及可并行段的抽取与分析。具体包括识别#pragma 标注语句中的循环迭代次数、冲突变量及冲突变量类型,生成相应的冲突变量声明和主函数调用语句;抽取源程序中的可并行段代码并进行分析,获取可并行段中的私有变量、私有变量类型以及哪些冲突变量在运行中可能进行猜测读/写的信息。这部分工作的结果将被用于指导猜测线程函

数的生成。

(2)猜测线程函数的生成。从图 3(b)可以看出，猜测线程函数的结构十分规整，只需根据可并行段生成虚线框外的代码即可。这部分代码包括冲突变量和私有变量的声明与初始化，它可以根据(1)的分析结果得到；还包括猜测线程要完成的计算，这就需要简单地将可并行段中的循环体复制过来，修改其循环指针(head 和 tail)，并为冲突变量添加必要的猜测读/写函数调用。

(3)生成 HEUSPEC 内部代码。在(1)生成的猜测变量声明和主函数调用语句和(2)生成的猜测线程函数的基础上对 HEUSPEC C 代码进行重构，得到 HEUSPEC 内部代码并输出(若需要)。

C2H 转换模块的各个部分的具体实现将在第 3 节中结合 Clang 来进行介绍。

3　实现

3.1　Clang 简介

Clang 是由 Apple 公司使用 C++及 STL 开发的一个新的 LLVM 的编译器前端，支持 C，C++，Objective C 和 Objective C++，目前已更新到 3.9 版本。Clang 的开发目标是提供一个可以替代 GCC 的前端编译器，以便在集成开发环境中进行代码编写及重建[7]。

相比于 GCC，Clang 采用了基于库函数的设计模式，在这种情况下，Clang 编译器可以被清晰地分成多个独立的库，不同用户可以根据不同的目的灵活地将这些库进行一定的组合。同时，Clang 为用户提供了简单易用的接口以便调用各库所提供的功能，使开发者可以方便地使用 Clang 来构建各样新的工具。另外，Clang 还为用户提供了友好的错误诊断提示，可以直接通过波浪线及尖括号来清晰地为用户指出程序出错的位置。

3.1.1　Clang 编译器架构

Clang 编译器采用了基于函数库的架构设计，其函数库包括核心函数库(如 Basic，Lex，Parse，AST，Sema，CodeGen 等)、应用类函数库(如 Analysis，Rewrite，Driver，Frontend 等)以及提供基础支持的 LLVM Support 函数库[7-9]等，如图 5 所示。

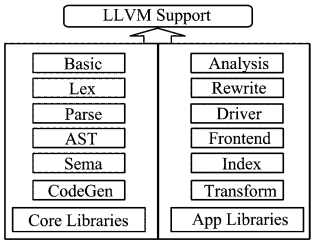

图 5　Clang 编译器的框架结构

核心函数库(Core Libraries)主要提供编译器前端需要的所有支持，能够完成源代码到编译器中间表示的所有工作。应用类函数库(App Libraries)主要为 Clang 编译器的扩展提供支持。可以利用应用类库函数对 Clang 的功能进行扩展。LLVM Support 函数库主要提供众多底层库的支持及相关数据结构，其中包括命令行选项处理、各种各样的容器、一个用来负责访问文件系统的系统抽象层。

本文设计并实现的 C2H 转换模块为 Clang 添加了一个应用类函数库 C2H，是通过对 AST 库、Analysis 库、Frontend 库以及 Transform 库进行重新组合而生成的一个新的 Clang 工具。

3.1.2　Clang 的内部表示

Clang 编译器的内部表示与现有大多数编译器架构的内部表示相同，采用了树形结构，即抽象语法树[8]。抽象语法树代表了一个源程序的抽象表示，其中每一个节点都代表了源代码中每一个模块或者元素，这样做的好处在于可以将源程序代码转换成编译器可以自动识别处理的结构，以便编译器对程序源代码进行统一的查找、分析、转换等操作。此外，程序员也可以根据自己的需要扩充 Clang 的内部表示。

Clang AST 节点有很多类型，其中 Stmt 和 Decl 是最重要的两个基类节点，通过这两个基类节点还派生出了许多子类节点。本文设计并实现的 C2H 转换模块就是在 Stmt 基类节点的基础上增加了 HEUSPECFor 制导语句用于识别标注语句#pragma heuspec for(ITER_NUMBER) CVT。

3.2　C2H 转换模块的实现

本文 2.2 节对 C2H 转换模块完成的工作进行了简单介绍，本节以图 3 中计算 Pi 值的代码为例对各个部分的具体实现进行了分析。

3.2.1　标注语句和可并行段的抽取与分析

对标注语句及其对应的可并行段代码的抽取与分析是 C2H 转换模块的重点，抽取可并行段代码后，需要结合 HEUSPEC 内部代码的结构特点来对该可并行段及对应的标注语句进行分析，主要包括以下两方面。

(1)分析标注语句中的(N)(pi，w)部分。第一个括号内的变量为可并行段的迭代次数，在调用主线程函数 TLS_main_body()时传递给主线程函数的第二个参数，主线程函数的第一个参数表示猜测线程函数名，它必须与生成的猜测线程函数名一致；第二个括号中的内容为冲突变量名及冲突变量类型，用于生成冲突变量声明语句，具体如描述算法 1 所示。

算法 1　标注语句分析算法

```
输入：形如#pragma heuspec for(iter)CVT 的标注语句
输出：CHKenel rd
Func DirectiveAnalysis(){
    rd. cnt=iter;//记录可并行段的迭代次数
    int i=0;
    for each element e in CVT
        rd. cvt[i];//记录可并行段的冲突变量
    for 可并行段中的每条语句 s {
        if 含有私有变量声明语句
            rd. pvt[j]=varj;//记录可并行段的私有变量
        rd. ir[k]=s;//记录可并行段
    }
}
```

(2)分析可并行段代码，生成猜测函数的相关信息。它包括 3 个部分：1)检测可并行段中除冲突变量外的私有变量及其

类型;2)检测可并行段中哪些冲突变量可能在运行中产生猜测读写;3)将可并行段剥离出来作为猜测线程函数的一部分。

分析 HEUSPEC C 中的标注语句及其对应的可并行段的主要目的是指导 HEUSPEC 内部代码的生成,因此需要一种有效的方式将分析结果保存起来。为此,本文定义了一种专用数据类型 CHkernel,用来记录可并行段中的冲突变量和私有变量及其类型、可能发生猜测读/写的冲突变量、可并行段的循环体等信息。

3.2.2 猜测线程函数的生成

猜测线程函数是根据 3.2.1 节所获取的信息生成的,具体描述如算法 2 所示。它由一个循环结构组成,该循环结构保证了猜测线程能够持续地从主线程接受任务并执行,直到全部计算任务完成;互斥锁语句使得每次只有一个线程从共享数据空间中读取数据,保证了数据读取的正确性;mt□□.PAddr是 HEUSPEC 的内部数据结构,其功能是为冲突变量分配私有空间;TLS_copy_in()是 HEUSPEC 的内部函数,其功能是将冲突变量的数据从共享数据空间拷入冲突变量的私有空间,各个猜测线程在执行猜测任务时只使用自己私有空间中的数据,彼此之间是相对独立的,保证了猜测执行的正确性;head 和 tail 是猜测线程将要执行的猜测任务的头、尾指针,其与猜测任务的粒度划分有关。根据算法 2 生成的猜测线程函数如图 3(b)所示。为了简化这部分的实现,将那些固定不变的部分制作成为一个模板,只需根据可并行段的信息在模板的特定位置增加相应内容即可。

算法 2 猜测线程函数生成算法

```
输入:CHKenel rd
输出:猜测线程函数 ThreadFunc()的 C 代码
Func CreateSpecThread(){
    输出函数头、线程号初始化及头尾指针定义语句
    for each element e in rd.cvt
      输出冲突变量声明语句
    for each element e in rd.pvt
      输出私有变量声明语句
    输出 wile(1)循环语句{
      输出等待计算开始消息、添加互斥锁语句
      for each element e in rd.cvt{
      输出初始化冲突变量的私有空间创建语句
          输出初始化冲突变量初值语句
        }
        输出初始化地址映射表语句
        输出猜测度函数
        for each element e in rd.ir
          输出对应的 c 语句
        输出猜测写函数
        输出与主线程通信、函数尾等语句
    }
}
```

3.2.3 重构代码输出

重构代码输出结合了 3.2.1 节、3.2.2 节中生成的冲突变量声明、主函数调用语句及猜测线程函数,按照 HEUSPEC 内部代码的编程规范,对 HEUSPEC C 代码进行重写,从而生成 HEUSPEC 内部代码,具体描述如算法 3 所示。

算法 3 HEUSPEC 内部代码生成算法

```
输入:CHKernel rd,ThreadFunc()和 HEUSPEC C 代码
输出:HEUSEPC 内部代码
Func GeneratHeuspec(){
    输出 HEUSPEC C 代码头文件引用语句
    输出 TLS 接口头文件引用和猜测线程函数生命语句
    输出 HEUSPEC C 代码{
      if 标注语句{
          for each element e in rd.cvt{
              输出冲突变量声明语句
              输出主函数调用语句
          }
      }
      输出 ThreadFunc()
}
```

重构代码输出的结果就是最后转换获得的 HEUSPEC 内部代码(见图 3)。

4 测试

4.1 实验环境及测试程序

本文在 Intel® Core™ i5-3200 四核处理器平台上对 C2H 工具进行了正确性和性能测试。所用的操作系统是 Ubuntu 15.04。表 1 列出了测试所用的全部测试程序,表 1 中右边的 3 列数据(冲突变量、动态数组、归并和)分别描述了测试程序中冲突变量的数量、有无动态数组类型的冲突变量及有无归并和类型的冲突变量。表 1 中冲突变量的个数是由程序员手工识别得到的。特地标出测试程序中是否有动态数组和归并和变量的原因在于 HEUSPEC 为这两类变量提供了特殊支持。

表 1 测试程序

测试程序	所属程序包	用途	冲突变量	动态数组	归并和
hotspot	Rodinia 2.1	热力学模拟	10	无	无
Heartwall	Rodinia 2.1	医学成像	2	无	无
LU	OmpScr 2.0	科学计算	4	有	无
Mandelbort	OmpScr 2.0	随机过程	3	有	有
Pi	OmpScr 2.0	科学计算	2	无	有
lavaMD	Rodinia 2.1	物理模拟	5	无	无

4.2 功能测试

功能测试的主要目的是测试本文设计并实现的 C2H 工具是否能够将串行 C 代码转换为结果正确的 HEUSPEC 内部代码。

图 3(b)给出了测试程序 Pi 使用 C2H 工具生成的 HEUSPEC 内部代码。可以看出,使用 C2H 工具生成的 HEUSPEC 内部代码结构简单清晰,可读性较强。受篇幅所限,其余 5 个测试程序对应的 HEUSPEC 代码不再逐一介绍。

对于每个测试程序,对比其串行版本、程序员手工得到的 HEUSPEC 内部代码和使用 C2H 工具生成的 HEUSPEC 内部代码的运行结果可知,3 个版本的结果完全一致,这在一定程度上说明了 C2H 的正确性。实际上,影响 HEUSPEC 内部代码(包括手工编写的和 C2H 生成的)正确性的主要原因在于能

否标识出可并行段中的所有猜测变量。在这一方面，C2H 反而更具优势，因为它采用了保守的机制，即它会把那些在编译时无法确定为线程私有的变量都标为猜测变量。

需要强调的是，本文的重点在于对 HEUSPEC 代码的结构分析以及 C2H 工具的设计与实现，因此在接下来的性能测试中，HEUSPEC 标注语句的猜测变量列表由程序员手工生成，即对于同一个测试程序，其手工编写的 HEUSPEC 代码和 C2H 生成的 HEUSPEC 内部代码中的猜测变量是完全相同的。

4.3　性能测试

为了测试 C2H 工具所生成的代码的性能，本文对各个测试程序的串行代码（Seq）、手工 HEUSPEC 内部代码（Man）、自动生成的 HEUSPEC 内部代码（C2H）和 OpenMP 代码（OMP）进行了测试和对比，结果如表 2 所列。

表 2　测试程序的不同版本程序的测试结果对比

测试程序	粒度	Seq	Man	C2H	OMP
LU	5	14.926	6.364	6.464	6.579
	50		5.747	5.797	6.583
	500		7.205	7.525	6.565
	5000		15.260	16.450	6.561
heartwall	1	18.261	6.166	6.196	6.146
	2		6.477	6.567	6.114
	3		6.531	6.643	6.109
	4		6.867	6.895	6.115
hotspot	3	0.809	0.360	0.358	0.303
	30		0.361	0.376	0.302
	300		0.400	0.398	0.303
	3000		0.820	0.832	0.304
Mandelbrot	100	16.972	5.770	5.780	5.707
	200		5.736	5.766	5.704
	400		5.761	5.781	5.703
	800		5.721	5.731	5.710
Pi	200	8.615	3.999	4.011	2.872
	2000		3.001	3.101	2.870
	20000		2.899	2.869	2.871
	200000		27.603	28.233	2.869
lavaMD	2	23.696	7.962	7.956	8.512
	5		7.534	7.575	8.528
	10		7.965	7.993	8.505
	15		8.246	8.458	8.555

对于每个测试程序中除 seq 外的 3 个版本，表 2 还列出了不同任务粒度下测试程序的性能数据，它表示每个线程每次执行的任务量（迭代次数）。对于猜测并行执行的程序，粒度的大小会影响程序执行的全局控制开销的大小，从而影响程序执行的效率。而且误猜密集的程序在小粒度下的性能较高；误猜稀疏的程序在大粒度下的性能较高。由于本文的测试是在 4 个处理器 Intel(R) Core(TM) i5-2300 上完成的，因此 HEUSPEC 代码的猜测线程数量为 3，OpenMP 代码的线程数也为 3。

从表 2 中可以看出，粒度大小对 HEUSPEC 代码的执行性能的影响较大，在粒度最优的情况下，本文设计实现的自动转换工具转换获得的 HEUSPEC 内部代码在性能上不弱于程序员手工编写的 HEUSPEC 内部代码及 OpenMP 代码。

结束语　本文通过对 HEUSPEC 并行程序编写规范的分析，基于 Clang 编译器前端设计并实现了一个能够完成从串行 C 程序到 HEUSPEC 并行程序的自动转换工具。并从测试程序集 Rodinia 2.1 和 OmpScr 2.0 选取了 6 个测试程序来测试 C2H 转换工具的正确性和性能。在性能测试中分别对每个测试程序的串行代码、手工 HEUSPEC 内部代码、自动生成的 HEUSPEC 内部代码和 OpenMP 代码进行了测试。通过对测试结果的对比发现，在未进行优化的前提下，自动生成的 HEUSPEC 内部代码在性能上不弱于手工生成的 HEUSPEC 内部代码和 OpenMP 版本的代码。能否准确地识别出可并行段中的冲突变量，对 C2H 工具生成的 HEUSPEC 内部代码的正确性和性能都有十分重要的影响。因此，接下来将深入研究可并行段中冲突变量的静态和动态识别方法。

参 考 文 献

[1] HAMMOND L, WILLEY M, OLUKOTUN K. Data Speculation Support for a Chip Multiprocessor[C]// Proceedings of ASPLOS-VIII. 1998.

[2] XU F, SHEN L, WANG Z Y. HEUSPEC: a software speculation parallel model [C]// Proceedings of ICPP. 2013.

[3] SHEN L, XU F, ZHANG Z Y. Optimization Strategies Oriented to Loop Characteristics in Software Thread Level Speculation Systems [J]. Journal of Computer Science and Technology, 2016, 31(1): 60-76.

[4] LOPES B C, AULER R. Getting Started with LLVM Core Libraries[M]. First. 35 Livery Street: Packt Publishing Ltd., 2014.

[5] BAE H, MUSTAFA D, LEE J W, et al. The Cetus Source-to-Source Compiler Infrastructure: Overview and Evaluation[J]. International Journal of Parallel Programming, 2013, 41(6): 753-767.

[6] CHRIS V. LLVM: An infrastructure for multi-stageoptimization [R]. University of Illinois at Urbana-Champaign, 2008.

[7] VOUFO L, ZALEWSKI M, LUMSDAINE A. ConceptClang: an implementation of C++ concepts in clang[C]// Proceedings of 7th ACM SIG-PLAN Workshop on Generic Programming. 2011: 71-82.

[8] LATTNER C. LLVM and Clang: Next generation compiler technology[C]// Proceedings of the BSD Conference. 2008.

[9] Clang: a C language family frontend for LLVM[OL]. http://clang.blvm.org.

[10] GUNTLI C. Architecture of clang[J/OL]. www.docin.com/p-163698152m.html.

[11] STEVE N. Clang Internals[R]. Apple. Inc., 2009.

[12] 张清泉. 基于 Clang 的 C++代码混淆工具设计与实现[D]. 北京：北京邮电大学，2013.

[13] 张代远. 基于 Clang 的 C 语言代码并行化转换工具的设计与实现[D]. 吉林：吉林大学，2015.

[14] 章磊. Clang 上的 C/C++过程间分析和漏洞挖掘[D]. 北京：中国科学技术大学，2009.

[15] 王燕燕. OpenMP-to-OpenCL 代码自动转换工具的设计与实现[D]. 吉林：吉林大学，2015.