软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(Suppl.(2)):101−110

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

面向 OpenMP 自动并行化的代价模型*

李雁冰[1,2], 赵荣彩[1,2], 刘晓娴[3], 赵 捷[1,2]

[1](解放军信息工程大学,河南 郑州 450001)

[2](数学工程与先进计算国家重点实验室,河南 郑州 450001)

[3](广州军区联勤部 后勤信息中心,广东 广州 510000)

通讯作者: 李雁冰, E-mail: mr.liyanbing@gmail.com

摘 要: 现有的 OpenMP 代价模型较为简单,既没有充分考虑 OpenMP 程序的执行细节,也无法适应不同的循环并行执行方式.针对上述问题,对最先进的产品级优化编译器 Open64 中已有的代价模型进行扩展,以单个并行候选循环为对象,建立一种用于 OpenMP 自动并行收益分析的代价模型.该模型在改进了 Open64 原有 DOALL 并行代价模型的基础上,又增加了 DOACROSS 流水并行代价模型和 DSWP 并行代价模型.实验结果表明,建立的代价模型能够较好地评估循环并行执行开销的趋势,为 OpenMP 自动并行化中的收益分析提供了有效的支持.

关键词: 自动并行化;OpenMP;代价模型;DOACROSS;DSWP

中文引用格式: 李雁冰,赵荣彩,刘晓娴,赵捷.面向 OpenMP 自动并行化的代价模型.软件学报,2014,25(Suppl.(2)):101−110. http://www.jos.org.cn/1000-9825/14028.htm

英文引用格式: Li YB, Zhao RC, Liu XX, Zhao J. Cost model for automatic OpenMP parallelization. Ruan Jian Xue Bao/Journal of Software, 2014,25(Suppl.(2)):101−110 (in Chinese). http://www.jos.org.cn/1000-9825/14028.htm

Cost Model for Automatic OpenMP Parallelization

LI Yan-Bing[1,2], ZHAO Rong-Cai[1,2], LIU Xiao-Xian[3], ZHAO Jie[1,2]

[1](PLA Information Engineering University, Zhengzhou 450001, China)

[2](State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China)

[3](Logistics Information Center, Guangzhou Military Region Joint Logistics Department, Guangzhou 510000, China)

Corresponding author: LI Yan-Bing, E-mail: mr.liyanbing@gmail.com

Abstract: Existing OpenMP cost models does not give enough thought to the implementation details of OpenMP programs so they cannot be applied widely to different types of parallel loops. To solve this problem, this study extends the cost model in the most advanced product-level optimizing compiler Open64. Targeting single candidate parallel loop, it establishes a cost model suite for the OpenMP benefit analysis. Besides improving the original DOALL-loop faced cost model in the Open64 compiler, two additional models which are also designed for DOACROSS and DSWP (Decoupled Software Pipelining) loops respectively. The experimental results show that the proposed cost model suite can simulate the execution cost of parallel programs very well, and therefore can better support cost evaluation of OpenMP parallelization.

Key words: automatic parallelization; OpenMP; cost model; DOACROSS; DSWP

多核处理器已成为当前处理器设计的主流[1].OpenMP 编程模型[2]简单、高效,具有良好的可移植性,已广泛应用于多核平台.但要将已有的大量串行程序改写成 OpenMP 并行程序却并非易事.使用自动并行化技术[3],将串行程序自动变换成适合多核处理器的 OpenMP 程序,是获得 OpenMP 程序的一条重要途径.

自动并行化的对象主要是循环.然而不是所有的循环并行都能带来收益,为保证并行的有效性,编译器必须

* 基金项目: “核高基”国家科技重大专项(2009ZX01036-001-001-2); 数学工程与先进计算国家重点实验室开放课题(2013A11)
收稿时间: 2013-08-05; 定稿时间: 2014-03-13

在自动并行化过程中进行并行收益分析.OpenMP 程序的并行开销不仅与计算机系统的 CPU 指令类型,Cache 大小和访存时间有关,而且与并行执行模型、同步开销等因素有关[4],要实现精确高效的并行收益分析较为困难. 构建 OpenMP 程序执行的代价模型是实现精确收益分析的一条有效途径.根据蕴含并行性的不同,可以将循环分为串行循环,DOALL 循环和 DOACROSS 循环[5].针对不同的循环,目前有 3 种循环并行执行方式:DOALL 并行,DOACROSS 流水并行[6]和 DSWP 并行[7].本文所属课题开发的自动并行化系统 SW-VEC 中,已实现了上述 3 种不同执行方式 OpenMP 并行程序的自动生成.

现有的 OpenMP 代价模型较为简单,既没有充分考虑 OpenMP 程序的执行细节,也无法适应不同循环并行执行方式.SUIF[8]编译器中包含一个简单的启发式方法,基于循环体的规模和循环迭代空间的大小对并行收益进行评估.Open64[9]编译器中包含了一个层次化代价模型,该模型将 OpenMP 执行开销分为:串行执行开销和并行执行开销.其中,串行执行开销依赖于与处理器、内存相关的开销;并行执行开销则在串行开销的基础上,还要考虑并行所带来的额外开销.Open64 中的代价模型考虑了更多的影响因素,与上述两种模型相比,能够对循环的执行开销进行较为准确的分析.但 Open64 中的代价模型在评估循环并行执行开销时仍有诸多不足:一方面,该模型只考虑了并行启动开销和各线程的固定同步开销,无法适应线程之间复杂同步的情况;另一方面,该模型中只对 DOALL 这种简单的循环并行执行方式进行分析,无法适应多种循环并行执行方式.Liao[4]对 Open64 中已有的代价模型进行扩展,实现了一个编译时的 OpenMP 代价模型,用于对 OpenMP 并行区的执行开销进行评估.在该模型中,一个并行区域的执行开销等于执行该区域的线程组中耗时最长线程的执行时间与 fork-join 的时间开销之和.在计算线程开销时,将其中的 work-sharing 区域(omp for,omp section 或 omp single)和同步结构(如 omp master, omp critical 等)所花费的开销都包含在内.该模型在保证整体精确性的同时,将可能的负载不平衡问题暴露出来.但其面向的是手写 OpenMP 程序,既无法完全满足不同并行执行方式的需求,也不适合在自动并行化过程中使用.

针对上述问题,本文对目前分析能力较强的 Open64 中的代价模型进行改进和扩展,在完善了 Open64 原有 DOALL 并行代价模型的基础上,又增加了 DOACROSS 流水并行代价模型和 DSWP 并行代价模型.测试结果表明,本文建立的代价模型评估结果与实际测试结果有着相同的趋势,能够在一定程度上对循环在不同 OpenMP 并行执行方式时的并行开销进行较为准确的评估,为 OpenMP 自动并行化过程中的收益分析提供了支持.

1 SW-VEC OpenMP 自动并行化框架

本文所属课题开发的自动并行化系统 SW-VEC 中已实现了 OpenMP 并行程序的自动生成.根据对串行程序中循环依赖关系和数据流分析的结果,选择不同的自动并行化模块,通过适当地插入 OpenMP 编译指示和运行时库函数调用,在不改变原串行程序语义的前提下,将其转变为 OpenMP 并行程序.SW-VEC OpenMP 自动并行化框架如图 1 所示.

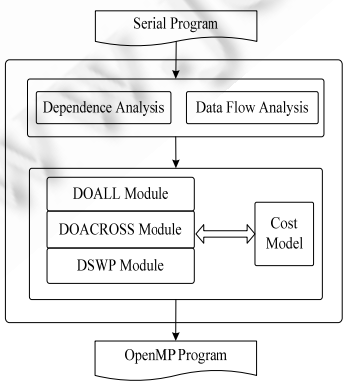


图 1 SW-VEC OpenMP 自动并行化框架

该框架在 Open64 编译器后端的嵌套循环优化模块(loop nest optimizer,简称 LNO)中实现,其中主要包含了

实现不同并行执行方式的 3 个模块,分别是 DOALL 并行模块,DOACROSS 流水并行模块和 DSWP 并行模块:

(1) DOALL 并行模块面向识别为 DOALL 类型的循环,根据代价分析的结果,使用各种循环变换技术将其转化为并行收益最优的等价循环;通过插入 OpenMP 编译指示和子句,创建循环对应的并行区,指示迭代空间的分配,调度方式并指定循环中变量的共享属性,实现循环的完全并行.

(2) DOACROSS 流水并行模块面向规则 DOACROSS 循环,首先使用启发式算法从循环各层中选出计算划分层和循环分块层,然后插入 OpenMP 的 schedule 子句将计算划分和调度方式指定为 static,之后基于代价模型计算分块大小来选择最佳流水并行粒度,最后使用计数信号量结合 OpenMP 的 flush 操作实现流水并行中线程间点到点同步.

(3) DSWP 并行模块面向包含复杂控制流,递归数据结构和多重指针访问的一般循环,首先构建循环的依赖图并将其划分为多个强连通子图,各部分对应于相互之间只存在单向依赖关系的任务,之后使用 OpenMP 编译指示将各个任务分配给不同的并行线程,使用数据共享属性子句完成线程之间的数据同步,从而实现循环的 DSWP 并行.

在这 3 个并行模块中,需要多次调用循环的代价模型,分别用于循环变换的选择和 DOALL 循环的并行收益分析,DOACROSS 流水并行流水粒度的选择和并行收益分析以及 DSWP 并行循环依赖图的划分和收益分析.现有的 OpenMP 代价模型无法满足这 3 个模块的需求,因此,建立一个面向 3 种不同循环并行执行方式的 OpenMP 代价模型是一个亟待解决的问题.

2 代价模型的建立

代价模型通常由一组底层模型组成,反映了应用程序在计算机系统上的具体执行细节,并且使用执行开销(一般是时间)对程序的执行效率进行评估[4].并行代价模型与一般的代价模型有所不同,不仅要反映计算机系统的具体细节,还要考虑程序的并行执行方式,并行同步时的开销等相关信息.本节首先介绍 Open64 中已有的代价模型,然后以此为基础建立适用于 OpenMP 自动并行化的代价模型.

2.1 Open64中的代价模型

Open64 编译器的 LNO 中包含了一组代价模型[4],用于对程序中的 SNL(singly nested loop)循环代价进行评估,如图 2 所示.这组模型将 OpenMP 执行开销分为两种类别:串行执行开销和并行执行开销.每一个类别对应一个相关的子模型:串行子模型和并行子模型.每个子模型的开销可以进一步分为粒度更细的开销类型,并且分别建模.其中,串行执行开销依赖于与处理器,内存相关的开销;并行执行开销则在串行开销的基础上,还要考虑并行所带来的额外开销.

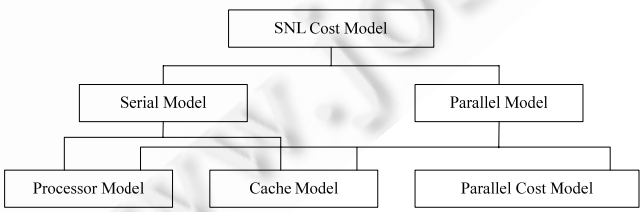


图 2 Open64 中的代价模型

公式(1)[4]为 Open64 的并行代价模型.其中,$Compute\_overhead_c$ 为循环并行时的计算开销,$Parallel\_overhead_c$ 为并行时的同步开销.该模型实现了对 DOALL 并行循环代价的分析.首先,将循环的并行执行开销分为计算开销和并行带来的额外开销.其中,计算开销通过调用处理器模型和 Cache 模型计算得到,等于循环的串行执行时间除以并行执行的线程数;并行带来的额外开销等于并行启动开销加上每个线程固定的并行同步开销与线程数的乘积.

$$\begin{aligned} Total_c &= Compute\_overhead_c + Parallel\_overhead_c \\ Parallel\_overhead_c &= Parallel\_startup_c + Parallel\_const\_factor_c * Num\_threads \\ Compute\_overhead_c &= Machine_c + TLB_c + Cache_c + Loop\_overhead_c \\ Loop\_overhead_c &= Loop\_overhead\_per\_iter_c * Num\_loop\_iter / Num\_threads \end{aligned} \quad (1)$$

该代价模型能够对循环的串行执行开销进行较为准确的分析,但在评估循环并行执行开销时仍有诸多不足:一方面,该模型只考虑了并行启动开销和各线程的固定同步开销,并使用常量对这两个参数定值,不能精确地对 OpenMP 中不同同步子句的并行开销进行评估,因此不足以适应线程之间复杂同步的情况;另一方面,该模型中只对 DOALL 这种简单的循环并行执行方式进行分析,无法适应多种循环并行执行方式.

因此,本文对 Open64 原有 DOALL 并行代价模型进行了改进,并在此基础上增加了 DOACROSS 流水并行代价模型和 DSWP 并行代价模型.其中,DOALL 并行代价模型在 Open64 原有并行模型的基础上,增加了对 SPMD 类型的 OpenMP 并行区域中可能出现多个同步构造开销的考虑.DOACROSS 流水和 DSWP 并行代价模型是通过对循环在这两种执行方式下的执行过程进行分析建模而得到的.此外,对于代价分析中需要使用,而静态编译过程中无法得到的信息,通过程序的预运行来动态获得,进一步提升了代价模型的精确性.

2.2 3种并行代价模型的建立

2.2.1 DOALL 并行代价模型

DOALL 循环的不同迭代可以按照任意顺序调度执行.图 3 和图 4 分别给出了一个实现 DOALL 并行的 OpenMP 循环和 OpenMP 的并行执行模型.图 3 所示为 Jacobi 迭代算法的核心循环经 SW-VEC OpenMP 自动并行化系统变换后所得的 OpenMP 并行循环.

```
for(k=0; k<max_iter; k++)
{
#pragma omp parallel
  {
#pragma omp for private(j)
    for(i=0;i<N;i++)
      for(j=0;j<M;j++)
        uold[i][j] = u[i][j];
#pragma omp for private(j)
    for(i=1;i<(N-1);i++)
      for(j=1;j<(M-1);j++)
        u[i][j] = uold[i-1][j] + uold[i+1][j]+uold[i][j-1]+uold[i][j+1];
  }
}
```

图 3 Jacobi 迭代算法核心

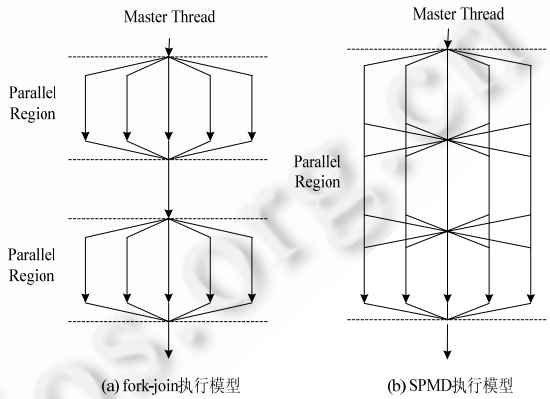


图 4 OpenMP 执行模型

Open64 的并行模型虽然较好地对 DOALL 并行循环的执行过程进行了建模,但是不能直接应用于 SW-VEC OpenMP 自动并行化系统中实现的 DOALL 循环自动并行,主要有以下两个原因:

(1) 该模型中使用的是编译器静态分析所获取的循环相关信息.对于循环边界为变量,在静态分析时无法获取的情况,直接跳过代码分析,将其并行.这样往往会导致循环并行后的执行时间大于串行执行时间.

(2) 该模型面向的是 fork-join 模式的并行循环,不能对 SPMD 类型的 OpenMP 区域进行分析.

针对第 1 个问题,将编译时的静态分析,与通过预运行和与程序员交互等动态方式相结合来获取代价分析过程中需要的信息,从而实现更精确的代价评估.下面对第 2 个问题进行分析.简单的 OpenMP 程序严格遵守 fork-join 执行模式,fork-join 模式很灵活且易于处理串行部分计算,然而每一个并行循环在开始执行前必须去共享内存读取数据,在并行循环末尾处等待线程组内所有线程完成相应计算.这种模型需多次创建和汇合并行线程,开销很大,图 4(a)中显示了 fork-join 的并行执行模型.

在图 4(b)所示的 SPMD 执行模型中,所有线程执行整个程序,串行部分或者重复执行或者限制为单线程执

行,当遇到并行循环时,线程根据线程编号执行相应部分的迭代,当线程间存在依赖关系时则插入同步.在 SPMD 模型中,线程一直处于活动状态.为 SPMD 并行区构建并行模型的关键是改进 Open64 并行模型中的并行额外开销 $Parallel\_overhead_c$ 的模型,以适应 SPMD 并行区的并行额外开销.因此,在计算 $Parallel\_overhead_c$ 时,使用以下式子:

$$Parallel\_overhead_c = Parallel\_startup_c + Parallel\_const\_factor_c * Num\_threads * Num\_synchronizations,$$

其中, $Parallel\_startup_c$ 为线程启动开销; $Parallel\_const\_factor_c$ 为线程同步因子,可通过对基准测试集的测试得到.这样就将 SPMD 并行区中可能存在多个栅障同步的情况考虑进来,能够更精确地对并行区代价进行评估.于是,DOALL 并行执行模型则为以下公式:

$$DOALL_c = Compute\_overhead_c + Parallel\_overhead_c,$$

$$Parallel\_overhead_c = Parallel\_startup_c + Parallel\_const\_factor_c * Num\_threads * Num\_synchronizations.$$

2.2.2 DOACROSS 流水并行代价模型

图 5 给出了有限差分松弛法 FDR(finite difference relaxation)波前计算循环的串行版本和 DOACROSS 流水并行版本.

Code segment 7-1: Sequential form of example loops

```
1   do j = 2, jend
2      do i = 2, iend
3         a(i, j) = 0.25 * (a(i-1, j) + a(i, j-1)
4                           + a(i+1, j) + a(i, j+1))
5      end do
6   end do
```

(a)

Code segment 7-2: Pipelined form of example loops

```
1        include “omp_lib.h”
2        real a(iend, jend)
3        integer i, j
4        integer isync(0: 256), mthreadnum, iam
5        common /threadinfo1/ isync
6        common /threadinfo2/ mthreadnum, iam
7     !$omp threadprivate(/threadinfo2/)
8     !$omp parallel default(shared) private(i, j) shared(a)
9          mthreadnum = 1
10    !$   mthreadnum = omp_get_num_threads()
11         iam = 1
12    !$   iam = omp_get_thread_num() + 1
13       isync(iam) = 0
14    !$omp barrier
15         do i = 2, iend, b
16            call sync_left(iend, jend, a)
17    !$omp do schedule( static )
18            do j = 2, jend
19               do ii = i, min((i+b-1), iend), 1
20                  a(i, j) = 0.25 * (a(i-1, j) + a(i, j-1)
21                                    + a(i+1, j) + a(i, j+1))
22               end do
23            end do
24    !$omp end do nowait
25            call sync_right(iend, jend, a )
26         end do
27    !$omp end parallel
```

(b)

Code segment 7-3: subroutine sync_left(di, dj, a)

```
1        integer di, dj
2        real a(di, dj)
3        integer isync(0: 256), mthreadnum, iam
4        common /threadinfo1/ isync
5        common /threadinfo2/ mthreadnum, iam
6     !$omp threadprivate(/threadinfo2/)
7        integer neighbour
8        if( iam .gt. 0 .and. iam .le. mthreadnum ) then
9           neighbour = iam - 1
10          do while( isync( neighbour) .eq. 0)
11    !$omp    flush(isync)
12          end do
13          isync(neighbour) = 0
14    !$omp  flush(isync, a)
15       end if
16       return
17       end
```

(c)

Code segment 7-4: subroutine sync_right(di, dj, a)

```
1        integer di, dj
2        real a(di, dj)
3        integer isync(0: 256), mthreadnum, iam
4        common /threadinfo1/ isync
5        common /threadinfo2/ mthreadnum, iam
6     !$omp threadprivate(/threadinfo2/)
7        if( iam .lt. mthreadnum ) then
8           do while( isync(iam) .eq. 1)
9     !$omp    flush(isync)
10          end do
11    !$omp flush(isync, a)
12          isync(iam) = 1
13    !$omp flush(isync)
14       end if
15       return
16       end
```

(d)

图 5 FDR 波前计算循环的串行版本和 DOACROSS 流水并行版本

图 5(a)所示循环包含 *i*, *j* 两层循环,四个数组 *a* 的读引用和一个数组 *a* 的写引用.依赖关系分析的结果显示该循环的 *i* 层和 *j* 层都携带依赖,无法完全并行,是一个 DOACROSS 循环.该循环经过 SW-VEC OpenMP 自动并行化系统变换后,得到的并行代码如图 5(b),图 5(c)和图 5(d)所示.

基于以下几点假设建立 DOACROSS 流水并行代价模型:

① 线程之间的同步是阻塞的,即同步完成前,不能进行其他的工作;

② 线程数目为 p;

③ 流水循环是二维矩阵计算区域 $N_1 \times N_2$,N_1 是计算划分层的迭代数,N_2 是循环分块层的迭代数;

④ 线程间调度采用 static 方式,每个线程分得的计算区域是$(N_1/p)\times N_2$;

⑤ 使用循环分块优化流水并行循环,调整计算代价与同步代价之间的比率,尽可能获得好的并行性和较低的同步代价,分块大小为 n_2;

⑥ 流水计算的调度:第 1 步,Thread0 计算分块 B(0,0),进行右同步,Thread1 进行左同步;第 2 步,Thread0 和 Thread1 开始计算 B(0,1)和 B(1,0);之后的步骤依此类推,整个流水计算过程如图 6 所示.

根据图 6 的 DOACROSS 流水并行计算过程, DOACROSS 流水并行的执行代价由单个分块的执行开销与线程执行分块数的乘积得到,可用下面这个式子表示:

$$\begin{aligned} Total_c &= Single\_tile\_overhead_c \times Number\_of\_tile \\ &= Single\_tile\_overhead_c \times (M_s + M_p), \end{aligned}$$

其中,M_s 是流水填充阶段最后一个计算结点开始计算前的分块个数,其值为 p–1,M_p 是最后一个结点需计算的分块总数,其值为 $\frac{N_2}{n_2}$.单个分块的执行开销 $Single\_tile\_overhead_c$ 可用以下式子计算得到:

$$Single\_tile\_overhead_c = Compute\_tile\_overhead_c + Synchronization\_tile\_overhead_c,$$

其中计算开销 $Compute\_tile\_overhead_c = ((N_1/p)\times b \times t_1 + t_2)$,t_1 是循环内层语句单次执行的时间开销;$Synchronization\_tile\_overhead_c$ 为单次线程同步的时间开销 t_2.于是 DOACROSS 流水计算的运行代价为

$$Total_c = ((N_1/p)\times b \times t_1 + t_2)\times\left(p - 1 + \frac{N_2}{b}\right).$$

对于图 5(a)中的循环,jend-1 对应上面公式中的 N_1,iend-1 对应于 N_2,p 可在运行时调用 OpenMP 的库函数 *omp_get_thread_num*()得到,t_1 和 t_2 的值通过调用编译器中包含的代价模型和对基准测试集的测试得到.

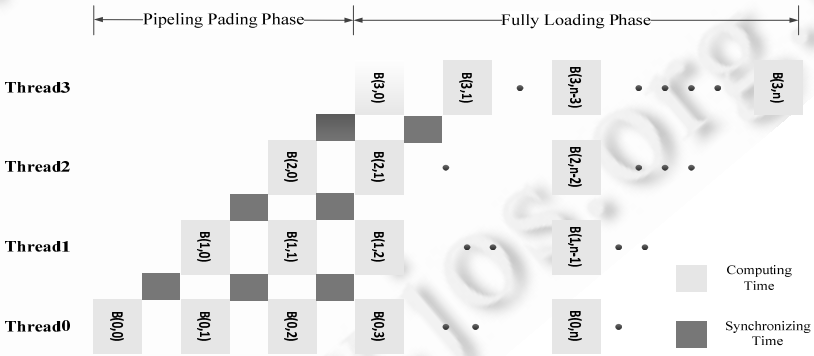


图 6 DOACROSS 流水并行计算过程

2.3 DSWP循环并行模型

下面以图 7(a)所示 NPB2.3-omp-C 中 EP 程序的热点循环为例建立 DSWP 并行模型,图 7(b)和 7(c)分别是 7(a)中循环对应的 DSWP 并行版本和执行模型示意图.

根据如图 7(c)所示的 DSWP 并行执行模型,从线程总是最后完成执行的线程.因此,将 DSWP 循环的执行时间分为两部分,一部分是从线程执行最后一个任务花费的时间,另一部分是从线程启动最后一个任务之前循环已执行的时间.在负载平衡的理想情况下,主线程执行任务花费的时间与从线程执行任务花费的时间相等,那么主线程完成最后一个任务的同时,从线程完成倒数第 2 个任务的执行,开始执行最后一个任务,那么循环并行执行开销的计算公式如:

$$T_{Parallel} = T_{Parallel\_construct} + T_{Master\_construct} + T_{Master\_work} + T_{Master\_work} \times (Num\_loop\_iter - 1) + T_{Auxiliary\_work} \quad (2)$$

或者

$$T_{Parallel} = T_{Parallel\_construct} + T_{Master\_construct} + T_{Master\_work} + T_{Auxiliary\_work} \times (Num\_loop\_iter - 1) + T_{Auxiliary\_work} \quad (3)$$

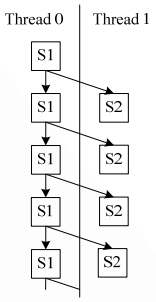
以上两个公式均可表示循环的并行执行开销,因为 T_{Master_work} 与 $T_{Auxiliary\_work}$ 相等.

```
for (k = 1; k <= np; k++) {
    kk = k_offset + k;
    t1 = S;
    t2 = an;
    for (i = 1; i <= 100; i++) {
        ik = kk / 2;
        if (2 * ik != kk) t3 = randlc(&t1,
t2);
        if (ik == 0) break;
        t3 = randlc(&t2, t2);
        kk = ik;
    }
    vranlc(2*NK, &t1, A, x-1);
    for ( i = 0; i < NK; i++) {
        x1 = 2.0 * x[2*i] - 1.0;
        x2 = 2.0 * x[2*i+1] - 1.0;
        t1 = pow2(x1) + pow2(x2);
        if (t1 <= 1.0) {
            t2 = sqrt(-2.0 * log(t1) / t1);
            t3 = (x1 * t2);
            t4 = (x2 * t2);
            l = max(fabs(t3), fabs(t4));
            qq[l] += 1.0;
            sx = sx + t3;
            sy = sy + t4;
        }
    }
}
```

(a) 串行循环

```
#pragma omp parallel threads(2)
{
#pragma omp master
  {
for (k = 1; k <= np; k++){
    kk = k_offset + k;
    t1 = S;
    t2 = an;
    for (i = 1; i <= 100; i++){
        ik = kk / 2;
        if (2 * ik != kk) t3 = randlc(&t1, t2);
        if (ik == 0) break;
        t3 = randlc(&t2, t2);
        kk = ik;
      }
#pragma omp task binding(1) firstprivate(t1)
      {
    vranlc(2*NK, &t1, A, x-1);
    for ( i = 0; i < NK; i++){
        x1 = 2.0 * x[2*i] - 1.0;
        x2 = 2.0 * x[2*i+1] - 1.0;
        t1 = pow2(x1) + pow2(x2);
        if (t1 <= 1.0){
            t2 = sqrt(-2.0 * log(t1) / t1);
            t3 = (x1 * t2);
            t4 = (x2 * t2);
            l = max(fabs(t3), fabs(t4));
            qq[l] += 1.0;
            sx = sx + t3;
            sy = sy + t4;
          }
        }
      }
    }
  }
}
```

(b) DSWP 并行循环



(c) DSWP 并行执行模型

图 7 DSWP 并行示例

在实际情况中,因为受到主线程生成从线程任务的开销以及调度从线程任务的开销等因素的影响,主线程与从线程在单次迭代中的执行时间常常不相等.当 $T_{Master\_work} > T_{Auxiliary\_work}$ 时,循环的并行执行代价使用公式(2)计算;反之,使用公式(3)计算.使用下面的公式将公式(2)和公式(3)统一起来:

$$T_{Parallel} = T_{Parallel\_construct} + T_{Master\_construct} + T_{Master\_work} + \max(T_{Master\_work}, T_{Auxiliary\_work}) \times (Num\_loop\_iter - 1) + T_{Auxiliary\_work}.$$

其中, $T_{Master\_work} = T_{Portion\_for\_master} + T_{Task\_generation}$, $T_{Auxiliary\_work} = T_{Task\_scheduling} + T_{Portion\_for\_auxiliary}$, $T_{Portion\_for\_master}$ 和 $T_{Task\_scheduling}$ 分别对应于原循环经划分后分配给主线程和从线程的两部分代码的执行开销, 可调用 Open64 中已有的处理器模型和 Cache 模型进行计算; $T_{Parallel\_construct}$, $T_{Master\_construct}$, $T_{Task\_generation}$ 和 $T_{Task\_scheduling}$ 这 4 个部分分别对应于 OpenMP 中 parallel 构造的并行启动开销, master 构造所带来的开销,主线程遇到 task 构造时生成一个新任务的开销和从线程从任务池中调度一个任务执行的开销.这 4 个参数的值与 OpenMP 运行库的具体实现有关,可以通过在目标平台上对 microbenchmark 的测试来获得.

3 实验结果及分析

本文提出的代价模型已在课题开发的 SW-VEC OpenMP 自动并行化系统中实现.本节使用的测试平台为

IBM 3850 服务器,其中包含 2 个 Intel Xeon E7420 CPU,每个 CPU 中包含 4 个主频为 2.13GHz 的处理器核,内存为 4GB.使用一组基准测试程序对本文提出的代价模型进行测试,包括精确度测试和性能提升测试.

3.1 精确度测试

精确度测试包括绝对精确度测试和相对精确度测试.模型的绝对精确度通过将模型评估的开销与对测试程序的时间测试开销进行对比得到,相对精确度通过公式((modeled-measuredc)/measuredc)来计算,其中modeledc 表示模型评估的开销,measuredc 表示时间测试的开销.

选择 3 个测试程序分别作为 DOALL 并行循环,DOACROSS 并行循环和 DSWP 并行循环的代表,测试时对各个测试用例选择了不同的规模和线程数目,用于详细测试这 3 种并行代价模型在不同情况下代价评估的准确性.Jacobi 迭代算法的核心被选作 DOALL 并行代价模型的测试用例,如图 3 所示.对于该测试用例,选择了 512×512,768×768 和 1024×1024 三种不同的数组大小来考察不同的数据输入集对本文模型的影响.图 5 所示的 FDR 波前计算循环的流水并行版本被选作 DOACROSS 流水并行代价模型的测试用例,使用的 3 种规模分别是 64×64×64,128×128×128 和 256×256×256.图 7 所示的基准测试集 NPB2.3-omp-C 中 EP 程序的热点循环被选作 DSWP 并行代价模型的测试用例,使用的 3 种规模分别是 S 规模、W 规模和 A 规模.

图 8 和 9 分别给出了 Jacobi 迭代算法核心循环和 FDR 波前计算循环通过模型评估所得结果与实际测试结果的对比图.大多数情况下,模型评估的结果与实际的测试结果有着相同的趋势.但用虚线表示的模型评估结果基本上是平滑的,而实线表示的实际测试结果往往存在不规则的起伏,这是因为在实际测试时系统中存在的噪音所导致的.在图 8 所示的 Jacobi 迭代算法核心循环测试结果中,当循环输入规模为 512×512,线程数目大于 4 时,随着线程数目的增加,循环的实际执行时间呈增大趋势,而模型给出开销呈缓慢减小趋势,这是因为随着线程数目的增大,循环中数组访问的局部性受到影响,导致 cache 不命中率增加,而在模型中没有对该类因素进行考虑.

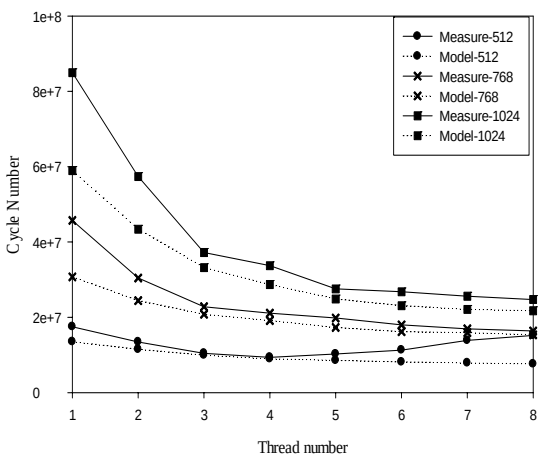


图 8 Jacobi 迭代算法核心模型估计值与实测值的对比

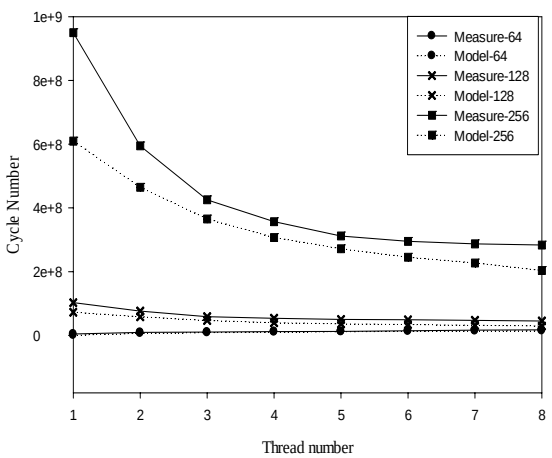


图 9 FDR 波前计算循环模型估计值与实测值的对比

图 10 和图 11 分别给出了前个测试循环模型评估开销的相对精确度.目前 DSWP 并行实现的是两线程的并行,对于图 7 所示 EP 程序的热点循环不给出模型估计值与实测值的对比图,相对精确度见表 1.当为正值时表示估计值大于测试值,而负值则表示估计值小于测试值.

相对精确度的测试结果表明,本文实现的模型精确度不够高,还有很多有待改进和调整的空间.

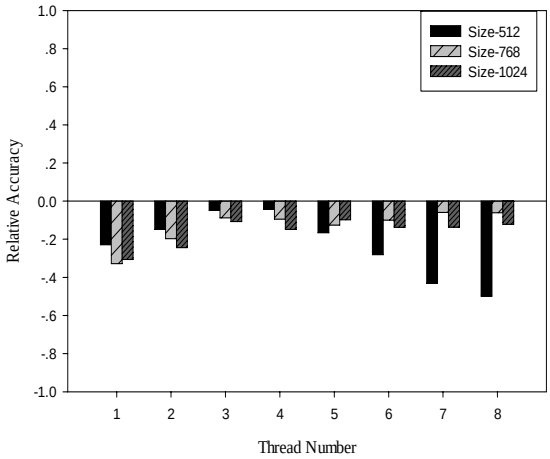


图 10 Jacobi 迭代算法核心相对精确度

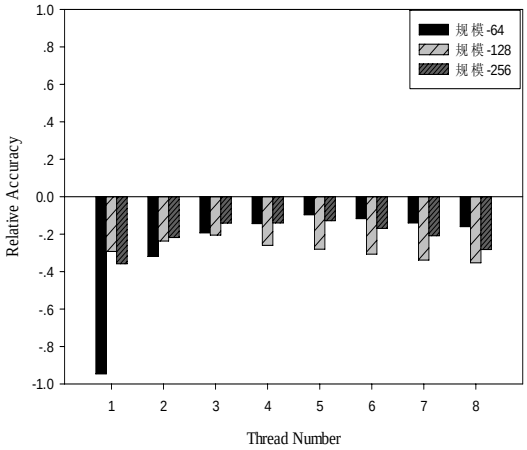


图 11 FDR 波前计算循环相对精确度

表 1 EP 热点循环模型的相对精确度

	单线程	两线程
S 规模	0.262	−0.261
W 规模	−0.114	−0.225
A 规模	−0.997	−0.173

3.1 性能提升测试

性能提升测试选择了 NPB3.3.1(NAS Parallel Benchmark)中的 SP,CG,BT,UA,LU 和 Poisson 程序作为测试用例.选择测试程序时有以下考虑:SP 和 CG 中有较多 DOALL 循环,LU 和 Poisson 中有较多 DOACROSS 循环,BT 和 UA 中有较多 DSWP 可并行循环.

分别使用实现了本文代价模型的 OpenMP 自动并行化系统和使用 Open64 原有代价模型的 OpenMP 自动并行化系统,对以上 6 个测试程序进行自动并行化变化,所得的 OpenMP 并行程序经过基础编译器编译后在测试平台上运行,通过比较两者的加速比对本文实现的代价模型进行评估.测试时 NPB 中的测试用例选择 C 规模,并在 4 和 8 两种线程数目下计算加速比.

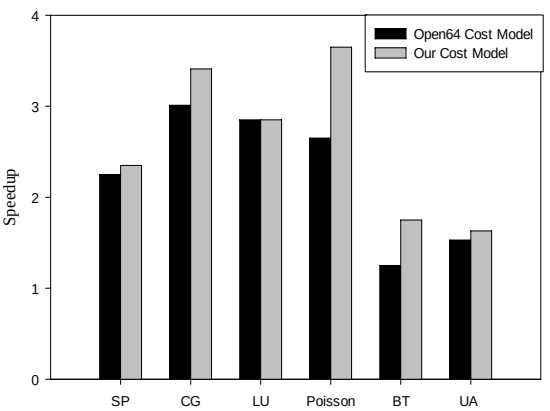


图 12 4 线程时性能对比

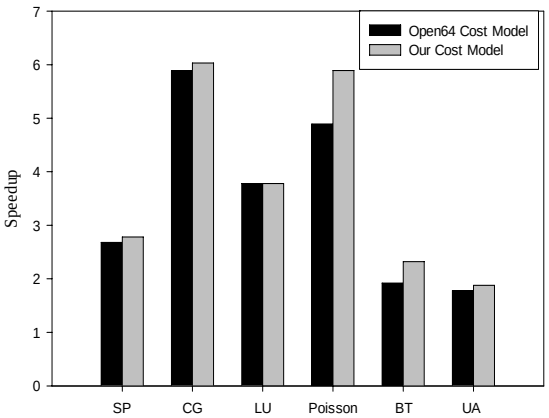


图 13 8 线程时性能对比

4 线程和 8 线程时测试结果分别如图 12 和图 13 所示,其结果表明实现了本文代价模型的 OpenMP 自动并行化系统变换得到的并行程序其性能明显优于使用 Open64 原有代价模型的 OpenMP 自动并行化系统.

以上的测试结果,本文建立的代价模型虽然在精确度上仍然有待进一步的改进,但是模型评估的结果与实际的测试结果有着相同的趋势,能够在一定程度上对循环在不同 OpenMP 并行执行方式时的并行开销进行较为准确地评估.在应用于 OpenMP 自动并行化系统只要考虑到精度上的不足,利用模型评估的结果与实际的测试结果有相同趋势的优势,就能提升生成代码的并行性能.

4 结束语

本文对面向 OpenMP 自动并行化过程中的收益分析构建了一个代价模型.实验结果表明,本文建立的代价模型能够较好地评估循环并行时执行开销的趋势,为 OpenMP 自动并行化中的收益分析提供了有效的支持.

致谢 向对本文研究工作提供基金支持的单位和评阅本文的审稿专家表示衷心的感谢,向为本文研究工作提供基础和研究平台的前辈致敬.

References:

[1] Jin HQ, Jespersen D, Mehrotra P, Biswas R, Huang L, Chapman B. High performance computing using MPI and OpenMP on multi-core parallel systems. Parallel Computing, 2011,37(9):562−575.

[2] OpenMP Architecture Review Board. OpenMP Application Program Interface Version 3.1. 2011.

[3] Qian Y. Automatic parallelization tools. In: Proc. of the World Congress on Engineering and Computer Science. New York: Springer-Verlag, 2012. 97−101.

[4] Liao CH. A compile-time OpenMP cost model [Ph.D. Thesis]. University of Houston, 2007.

[5] Kim H, Raman A, Liu F, Lee J, August D. Scalable speculative parallelization on commodity clusters. In: Proc. of the 2010 43rd Annual IEEE/ACM Int'l Symp. on Microarchitecture. New York: IEEE, 2010. 3−14.

[6] Unnikrishnan P, Shirako J, Barton K, Chatterjee S, Silvera1 R, Sarkar V. A practical approach to DOACROSS parallelization. In : Spirakis P, Luc Bougé, Papatheodorou T, eds. Proc. of the Euro-Par 2012. Heidelberg: Springer-Verlag, 2012. 219−231.

[7] Liu XX, Zhao RC, Ding R. Expansion to OpenMP task scheduling mechanism for DSWP Parallelization and its Implementation. Computer Science, 2013,40(9):38−43.

[8] Dou J, Cintra M. A compiler cost model for speculative parallelization. ACM Trans. on Architecture and Code Optimization, 2007, 4(2):12.

[9] Gao W, Li X, Zhao B. Source to source translation process of Open64. Journal of Information Engineering University, 2013, 14(5):612−618(in Chinese with English abstract).

附中文参考文献:

[7] 刘晓娴,赵荣彩,丁锐.面向 DSWP 并行的 OpenMP 任务调度机制的扩展与实现.计算机科学,2013,40(9):38−43.

[9] 高伟,李骁,赵博.Open64 源源翻译流程研究.信息工程大学学报,2013,14(5):612−618.

李雁冰(1989－),男,甘肃陇西人,硕士生,主要研究领域为并行编译.
E-mail: mr.liyanbing@gmail.com

刘晓娴(1985－),女,博士,主要研究领域为并行编译.
E-mail: xiaoxian0321@gmail.com

赵荣彩(1957－),男,教授,博士生导师,主要研究领域为并行编译,高性能计算,反编译技术.
E-mail: rczhao126@126.com

赵捷(1987－),男,博士生,主要研究领域为并行编译.
E-mail: zjbc2005@163.com