

基于循环分块的流水粒度优化算法

刘晓娴^{1,2*}, 赵荣彩^{1,2}, 丁锐^{1,2}, 李雁冰^{1,2}

(1. 信息工程大学, 郑州 450002; 2. 数学工程与先进计算国家重点实验室, 郑州 450002)

(*通信作者电子邮箱 xiaoxian0321@gmail.com)

摘要: 当计算划分层迭代数目较大, 或是循环体单次迭代工作量较大, 但可用的并行线程数目较小时, 传统的基于循环分块的流水粒度优化方法无法进行处理。为此, 提出一种基于循环分块减小流水粒度的方法, 并根据流水并行循环的代价模型实现最优流水粒度的求解, 设计实现了一个流水计算粒度的优化算法。对有限差分松弛法(FDR)的波前循环和时域有限差分法(FDTD)中典型循环的测试表明, 与传统的流水粒度选择方法相比, 所提算法能够得到更优的循环分块大小。

关键词: 自动并行化; 流水并行; 流水粒度; 循环分块; 代价模型

中图分类号: TP314 **文献标志码:** A

Pipelining granularity optimization algorithm based on loop tiling

LIU Xiaoxian^{1,2*}, ZHAO Rongcai^{1,2}, DING Rui^{1,2}, LI Yanbing^{1,2}

(1. Information Engineering University, Zhengzhou Henan 450002, China;

2. State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou Henan 450002, China)

Abstract: When the pipelining loop has a great number of iterations, or the size of its body is large, but the number of available threads is small, the workload between two synchronizations of a thread is so heavy, which produces pretty low degree of parallelism. The traditional trade-off approach based on loop tiling cannot handle the above situation. To solve this problem, a pipelining granularity decreasing approach based on loop tiling was proposed. The optimal pipelining granularity was obtained by building the cost model for pipelining loop and a pipelining granularity optimizing algorithm was implemented. By measuring the wavefront loops of Finite Difference Relaxation (FDR) and the representative loops of Finite Difference Time Domain (FDTD), the loops show better performance improvement by using the proposed algorithm than the traditional one.

Key words: automatic parallelization; pipelining parallelization; pipelining granularity; loop tiling; cost model

0 引言

随着主频的提高, 处理器功耗以指数速度急剧上升, 使得以提高主频来提升处理器性能的方法不再有效^[1]。多核处理器已成为处理器体系结构发展的一个主要方向^[2]。尽管多核处理器能够提升多线程程序的性能, 但早已存在的诸多单线程程序无法从中获益, 程序员也习惯于编写单线程程序。自动并行化技术^[3]是将单线程程序移植到多核上的重要手段, 通过对单线程程序中蕴含并行性的分析与发掘, 采用程序变换技术自动生成适合多核处理器运行的多线程程序。根据Amdahl定律^[4], 应用程序通过并行获得的性能提升受限于程序中串行执行部分所占的比例。因此, 充分发掘程序的并行性是提升程序并行性能的有效手段。计算机科学中的二八法则表明, 程序中20%的代码占据了程序80%的执行时间, 这些花费大量时间开销的代码往往是程序中的循环。因此, 循环是发掘程序并行性的主要对象。

根据蕴含并行性的不同, Cytron^[5]将循环分为串行循环、能够完全并行的DOALL循环和DOACROSS循环。DOACROSS循环因为携带跨迭代的依赖关系, 并行性介于DOALL循环和串行循环之间。Chen等^[6]通过测试表明, 将程序中的DOACROSS循环串行执行会带来程序并行性的巨大损失。因此, 发掘DOACROSS循环中蕴含的并行性, 选择

合适的策略将其并行执行对提升程序的并行性能非常重要。通过判定循环携带依赖的依赖距离是可精确确定的常量还是变量, DOACROSS循环可分为规则DOACROSS循环和不规则DOACROSS循环^[7]。与不规则DOACROSS循环相比, 规则的DOACROSS循环因为携带依赖关系的规则性, 更适合通过编译器实现自动并行。流水并行是指将循环的各次迭代分配给不同的线程, 线程间流水执行来获得并行性, 迭代间的依赖通过某种方式的同步得到维持。流水并行是规则DOACROSS循环并行的重要方式。本课题在前期工作中以多核处理器为目标平台, 在Open64^[8]编译器后端实现了基于OpenMP^[9-10]的规则DOACROSS循环流水并行代码的自动生成, 并通过测试对其有效性进行了验证。

在循环流水并行过程中, 线程间的同步能够维持循环携带依赖, 保证循环执行的正确性, 但同时也会带来额外的开销。同一线程两次同步之间的计算工作量大小称为流水计算粒度, 简称流水粒度。流水粒度的大小与同步开销在循环执行总开销中所占的比例有直接关系, 影响着流水并行循环的性能。根据流水粒度的大小, 流水并行可分为细粒度流水和粗粒度流水。细粒度流水时, 线程两次同步之间的计算量较小, 下一线程能够较快获得所需数据进行计算, 减少了线程的空闲等待时间; 但因为粒度较小, 会引起较多次数的同步, 从而导致较高的同步代价和较低的流水并行性能。粗粒度流水

收稿日期: 2013-02-18; 修回日期: 2013-04-10。 基金项目: “核高基”国家科技重大专项(2009ZX01036-001-001-2)。

作者简介: 刘晓娴(1985-), 女, 江西宜春人, 博士研究生, 主要研究方向: 并行编译、高性能计算; 赵荣彩(1957-), 男, 河南洛阳人, 教授, 博士生导师, CCF高级会员, 主要研究方向: 并行编译、高性能计算、反编译技术; 丁锐(1984-), 男, 河南滑县人, 博士研究生, 主要研究方向: 并行编译、高性能计算; 李雁冰(1989-), 男, 甘肃陇西人, 硕士研究生, 主要研究方向: 并行编译。

增大了同步之间的计算量和同步数据量,因此减少了同步次数,同步代价相对较小,但后续的计算节点在开始计算之前需要更多的等待时间,降低了循环的并行性。

为平衡并行性和同步代价两者之间的关系,使得流水循环获得趋近于最优的并行性能,本课题前期实现的自动并行化算法中使用传统的流水粒度优化方法,从循环的各层(除计算划分层)中选择一层实施循环分块^[11]来调节流水的粒度。然而,当计算划分层包含的迭代数目很大,或是循环体单次迭代的工作量很大,或是这两种情况同时存在,但可用的并行线程数目较小时,线程的两次同步之间的工作量很大,同步开销对并行性能的影响变得很弱。但因为流水粒度过大,对循环的并行性造成了极大的损失。传统的循环分块方法无法处理上述情况。

针对传统流水粒度选择方法的不足,本文提出一种流水粒度优化算法来进一步提升自动生成的流水并行代码的性能。首先给出了基于循环分块减小流水粒度的方法,从而避免出现线程的两次同步之间的工作量过大,给循环并行性带来损失的情况;然后基于循环流水执行时的代价模型综合考虑流水粒度过大和过小的情况,实现最优流水粒度的求解;最后设计实现了流水粒度的优化算法。对有限差分松弛法(Finite Difference Relaxation, FDR)的波前(Wavefront)循环^[11]和时域有限差分法(Finite Difference Time Domain, FDTD)^[12]中典型循环的测试表明,与传统的流水粒度选择方法相比,本文算法能够得到更优的循环分块大小,基于本文算法自动生成的流水并行代码能够在多核处理器上获得明显的性能提升。

1 基于循环分块增大流水粒度

本章以FDR的波前循环为例介绍传统的循环分块技术如何对循环的流水计算粒度进行调节。FDR的波前循环如下所示:

```
DO I = 2, N - 1
  DO J = 2, N - 1
    A(I, J) = 0.25 * (A(I - 1, J) + A(I, J - 1) +
      A(I + 1, J) + A(I, J + 1))
  ENDDO
ENDDO
```

其中包含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。根据依赖关系分析的结果得出该循环的I层和J层都携带依赖,无法完全并行,是一个DOACROSS循环。

该循环的流水并行代码如下:

```
DOACROSS I = 2, N - 1
  POST(EV(1))
  DO J = 2, N - 1
    WAIT(EV(J - 1))
    A(I, J) = 0.25 * (A(I - 1, J) + A(I, J - 1) +
      A(I + 1, J) + A(I, J + 1))
    POST(EV(J))
  ENDDO
ENDDO
```

其中:I层作为计算划分层,其迭代被分配给各并行线程分别执行;J层迭代则保留在线程内部。并行线程在开始一个J层迭代的计算之前和完成一个J层迭代的计算之后,需要分别与其左侧和右侧的线程进行同步。该流水并行代码中使用POST和WAIT两个操作来示意并行时的同步。POST(EV(I))操作发布事件EV(I)已经发生的信号而WAIT(EV(I))阻塞直至事件被发送。该流水并行循环的执行过程如图1。以上是理想情况下的流水并行循环,因为可

用的并行线程数量充足,因此被划分的I层循环以单次迭代为单位对线程进行分配。

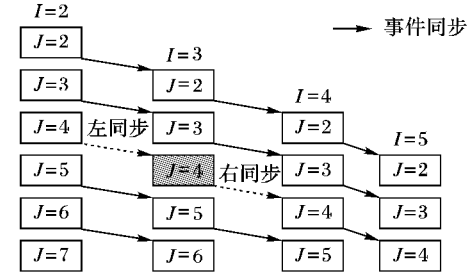


图1 理想情况下FDR波前循环的流水并行执行过程

然而在实际情况中,线程数量是有限的,因此每个并行线程获得的I层迭代数常常大于1。这时对循环的流水并行代码进行适当调整,得到以下循环:

```
DO J = 2, N - 1
  POST(EV(1))
  WAIT(EV(J - 1))
  DOACROSS I = 2, N - 1
    A(I, J) = 0.25 * (A(I - 1, J) + A(I, J - 1) +
      A(I + 1, J) + A(I, J + 1))
  ENDDO
  POST(EV(J))
ENDDO
```

其中I层仍然是计算划分层,J层被交换至最外层。假定可用的并行线程数为4,则每个线程分得的I层迭代数为 $(N - 2) / 4$ 。为简单起见,假设 $N = 33$,则每个线程各分得8次迭代。这时,线程之间的同步也发生相应改变,在开始计算之前与其左侧的线程进行同步,完成I层的8次迭代计算之后,与其右侧线程进行同步,如图2所示。

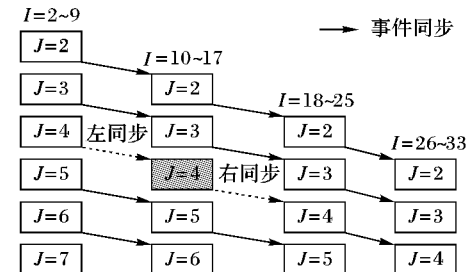


图2 实际情况下FDR波前循环的流水并行执行过程

循环在进行如图2所示的流水并行时,如果计算划分层包含的迭代数目较少,或循环体本身的工作量较小,也可能是这两种情况同时存在,从而导致流水计算粒度较小,同步引起的开销完全抵消甚至超过了并行带来的收益。也就是说,线程两次同步之间的工作量不足以作为调度的单位,这时,流水并行循环的性能低下,甚至会出现并行负收益。对于这种情况,往往通过循环分块的方式将更多的迭代组成集合,每个集合作为一个调度单位,从而减少同步开销在循环并行执行开销中所占的比重,提升流水并行循环的并行性能。通过对J层进行分块可得到循环:

```
DO J = 2, N - 1, 2
  K = 0
  POST(EV(1))
  K = K + 1
  WAIT(EV(K))
  DOACROSS I = 2, N - 1
    DO m = J, MIN(J + 1, N - 1)
      A(I, m) = 0.25 * (A(I - 1, m) + A(I, m - 1) +
        A(I + 1, m) + A(I, m + 1))
    ENDDO
```

```

ENDDO
POST (EV(K + 1))
ENDDO

```

这时线程的两次同步之间执行 J 层的两次迭代,则同一线程的两次同步之间完成循环体 16 次迭代的计算,与图 2 中的情况相比增大了流水的粒度,其执行过程如图 3 所示。

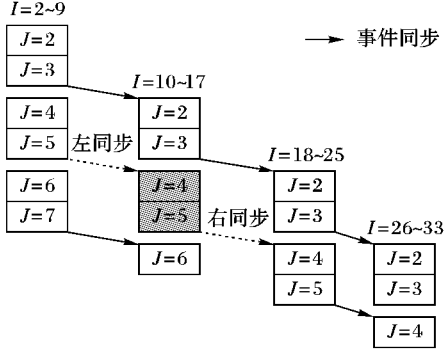


图3 采用循环分块后 FDR 波前循环的流水并行执行过程

根据以上流水执行过程,对循环并行时使用上述循环分块技术可调节的流水粒度范围进行分析。假定循环体(内层循环包含在内)进行单次迭代的工作量为 L ,循环体外有两层循环,其中一层进行计算划分,其迭代次数为 N_1 ,另一层通过循环分块来调节流水粒度的大小,其迭代次数为 N_2 ,设流水循环由 p 个线程并行执行,循环分块大小为 b ,则该循环流水并行时,流水粒度 n 可用下面这个式子进行计算:

$$n = \frac{N_1}{p} * b * L; 1 \leq b \leq N_2$$

其中 $\frac{N_1}{p}$ 表示各线程分得的计算划分层的迭代数。则当 b 的值为其取值范围的上界(N_2)和下界(1)时,得到可调节的流水粒度 n 的范围如式(1)所示:

$$\frac{N_1}{p} * L \leq n \leq \frac{N_1 * N_2}{p} * L \quad (1)$$

当 $b = 1$ 时,即不进行循环分块。因此,通过对循环分块的分块大小($1 \sim N_2$)的选择,可对循环并行时的流水粒度进行调节来增大流水计算的粒度,优化流水并行循环的性能。

然而,如果计算划分层包含的迭代数目很大,或是循环体单次迭代的工作量很大,也可能是这两种情况同时存在,但可用的并行线程数目较小。在这种情况下,线程的两次同步之间的工作量很大,同步开销对并行性能的影响变得很弱。但因为流水粒度过大,给循环的并行性带来了极大的损失,从而导致循环无法获得最优的并行性能。传统的循环分块方法无法处理上述问题。

2 基于循环分块的流水粒度优化算法

本章首先讨论如何基于循环分块减小流水粒度,从而避免出现线程的两次同步之间的工作量过大,造成循环并行性的损失,从而导致循环无法获得最优并行性能的情况;然后基于循环流水执行时的代价模型综合考虑流水粒度过大和过小的情况,实现最优流水粒度的求解;最后给出流水粒度优化算法。

2.1 基于循环分块减小流水粒度

考虑第1章给出的循环分块可调流水粒度范围的下界 $\frac{N_1}{p} * L$,其中单次迭代的工作量 L 和并行线程数 p 是固定的,则对于流水计算粒度过大的情况,应该通过减少同一并行线程的两次同步之间的迭代次数,即 N_1 来减小流水的粒度。因

此考虑对循环的计算划分层进行循环分段^[11],并选择分段后的内层作为新的计算划分层,从而达到减少同一并行线程两次同步之间的迭代次数的目的。下面仍然以图1中的循环为例进行说明。

对 FDR 的波前循环,仍然将 I 层作为计算划分层,为减小流水的粒度,对 I 层进行循环分段,假设每段中包含 4 次迭代。将分段后的内层循环 m 作为新的计算划分层,得到代码:

```

DO J = 2, N - 1
  DO I = 2, N - 1, 4
    DOACROSS m = I, MIN(I + 3, N - 1)
      A(m, J) = 0.25 * (A(m - 1, J) + A(m, J - 1) +
        A(m + 1, J) + A(m, J + 1))
    ENDDO
  ENDDO
ENDDO

```

假定 4 个并行线程流水执行循环迭代,则每个线程分得 m 层的一次迭代,减小了流水计算的粒度。但是,为了维持原 I 层循环携带的跨迭代依赖,并行线程间需要进行如图 4 所示的同步,即第 1 个线程在执行 $m = 6$ 这一迭代时,需要使用 $m = 5$ 迭代中写入的数据,因此需要与第 4 个线程进行同步。这样的流水方式导致并行性能比串行执行的性能更差。

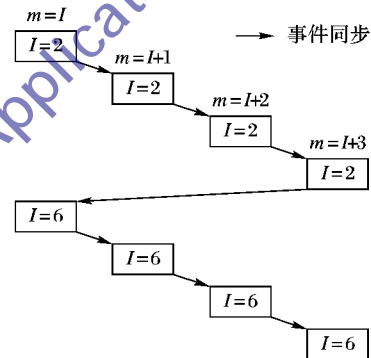


图4 计算划分层采用循环分段后 FDR 波前循环的流水并行执行过程

为提升循环分段后的并行性能,考虑在进行循环分段的同时,将 J 层循环与 I 层循环进行交换^[11],得到以下循环:

```

DO I = 2, N - 1, 4
  DO J = 2, N - 1
    DOACROSS m = I, MIN(I + 3, N - 1)
      A(m, J) = 0.25 * (A(m - 1, J) + A(m, J - 1) +
        A(m + 1, J) + A(m, J + 1))
    ENDDO
  ENDDO
ENDDO

```

将 J 层与 I 层交换后, J 层索引变量的变化先于 I 层索引变量,因此,当 I 层索引变量保持不变时,并行线程间保持单向的同步关系;当 I 层索引变量发生变化时,第 1 个线程所需要的数据已经由第 4 个线程计算完成,因此第 1 个线程可以保持运行状态而不用等待第 4 个线程。从图 5 中可以看出只需在第 1 个线程执行 $I = 6, J = 2$ 的迭代前,第 4 个线程完成了 $I = 2, J = 2$ 的迭代的执行,那么第 1 个线程无需等待,能在完成 $I = 2, J = 33$ 迭代的执行之后,立即与第 4 个线程同步,之后开始下一迭代的执行。也就是,当 J 层循环的迭代数 N_2 与并行线程数 p 之间满足 $N_2 > p$ 时能够保证线程执行连续性,该条件对一般的规则 DOACROSS 循环而言极易满足。

以上通过对循环分段技术和循环交换技术的结合使用达到了减小流水计算粒度的目的。实际上,这两种技术合并后即为循环分块技术,因此,可基于循环分块来减小流水的粒

度。根据图5所示的流水执行过程,对上述循环分块后可调节的流水粒度范围分析如下。假定循环体(内层循环包含在内)进行单次迭代的工作量为 L ,循环体外有两层循环,其中一层同时进行计算划分和循环分块,其迭代次数为 N_1 ,另一层通过循环交换来保证流水过程的连续性,其迭代次数为 N_2 ,设流水循环由 p 个线程并行执行($p < N_2$),循环分块大小为 b ,则该循环流水并行时,流水粒度 n 可用下面这个式子进行计算:

$$n = \frac{b}{p} * L; p \leq b \leq N_1$$

其中: $\frac{b}{p}$ 表示各线程分得的分块后所得计算划分层的迭代数, b 取值的下界为 p 是为了保证每个并行线程在一轮流水中至少能分得计算划分层的一次迭代。则当 b 的值为其取值范围的上界(N_1)和下界(p)时,得到可调节的流水粒度 n 的范围如下:

$$L \leq n \leq \frac{N_1}{p} * L \quad (2)$$

当 $b = N_1$ 时,即不进行循环分块。根据式(2),当对计算划分层使用循环分块后,可将流水粒度最小调节为单次迭代的工作量。将式(1)与(2)合并后,得到优化后的可选流水粒度范围如下:

$$L \leq n \leq \frac{N_1 * N_2}{p} * L \quad (3)$$

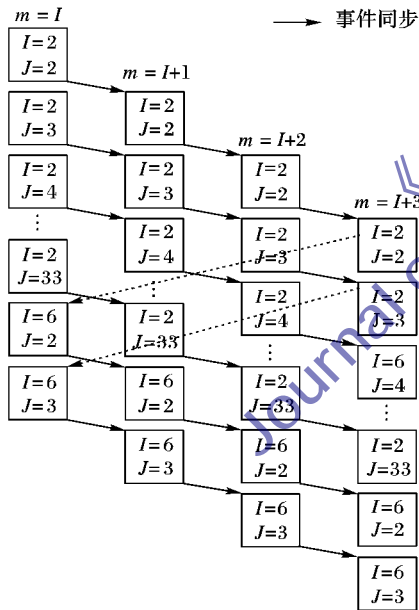


图5 计算划分层采用循环分块和循环交换后FDR波前循环的流水并行执行过程

2.2 流水并行代价模型

本节讨论流水并行代价模型的构建,然后基于代价模型综合考虑流水粒度过大和过小的情况,实现最优循环分块大小的求解。首先基于以下几点假设建立流水计算的模型^[13]:

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,类似于分布存储结构下的同步通信模式。

2) 线程数目为 p 。

3) 流水循环是二维矩阵计算区域 $N_1 * N_2$,通过循环分块增大流水粒度时, N_1 是计算划分层的迭代数, N_2 是循环分块层的迭代数;通过循环分块减小流水粒度时, N_1 既是计算划分层,又是循环分块层, N_2 是用于循环交换的层。

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

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 b 。

6) 流水计算的调度:①Thread0 计算分块 $B(0,0)$,进行右同步,Thread1 进行左同步;②Thread0 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。

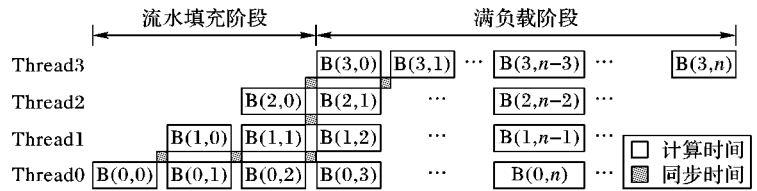


图6 流水计算过程

流水并行过程中计算与同步是不可重叠的,因此每一分块的执行时间是分块计算与同步时间的总和。下面分别对增大流水粒度和减小流水粒度两种情况进行分析。

当通过循环分块增大流水粒度时,单个分块的执行时间为

$$T_{tile} = T_{comp} + T_{comm} = \left(\frac{N_1}{p} * b * t_1 \right) + t_2$$

其中: t_1 是循环内层计算语句单次迭代的执行开销(即式(1)中的 L), t_2 是单线程同步的时间开销。整个流水执行时间描述为

$$T = (M_s + M_p) * T_{tile}$$

其中: M_s 是流水填充阶段最后一个计算节点开始计算前的分块个数,其值为 $p-1$; M_p 是最后一个节点需计算的分块总数,其值为 $\frac{N_2}{b}$ 。流水计算的运行时间 T 具体表示为

$$T = \left(p - 1 + \frac{N_2}{b} \right) * \left(\frac{N_1}{p} * b * t_1 + t_2 \right)$$

对上式求解 $\frac{dT}{db}$,得到最佳分块

$$b_{opt1} = \sqrt{\frac{N_2 * t_2 * p}{N_1 * t_1 * (p - 1)}} \quad (4)$$

当通过循环分块减小流水粒度时,单个分块的执行时间为

$$T_{tile} = T_{comp} + T_{comm} = \left(\frac{b}{p} * t_1 \right) + t_2$$

其中: t_1 是循环内层计算语句单次迭代的执行时间, t_2 是单线程同步的时间。整个流水执行时间描述为

$$T = (M_s + M_p) * T_{tile}$$

其中: M_s 是流水填充阶段最后一个计算节点开始计算前的分块个数,其值为 $p-1$; M_p 是最后一个节点需计算的分块总数,其值为 $\frac{N_1 * N_2}{b}$ 。流水计算的运行时间 T 具体表示为

$$T = \left(p - 1 + \frac{N_1 * N_2}{b} \right) * \left(\frac{b}{p} * t_1 + t_2 \right)$$

对上式求解 $\frac{dT}{db}$,得到最佳分块

$$b_{opt2} = \sqrt{\frac{N_1 * N_2 * t_2 * p}{t_1 * (p - 1)}} \quad (5)$$

在求解流水并行循环的最佳分块大小时,对于以上两个公式,首先使用式(4)进行计算:

1) 如果计算得到的分块大于1,则该循环需要增大流水计算粒度,循环的分块大小取 $\lceil b_{opt1} \rceil$;

2) 如果计算得到的分块小于1,则该循环需要减小流水计算粒度,继续使用式(5)进行计算,分块大小取 $\lceil b_{opt2} \rceil$ 。

2.3 流水粒度优化算法及实例分析

完整的流水粒度优化算法如下:

```
procedure Pipeline_Grularity_Optimize (L)
//功能:获得流水并行循环的最佳分块大小
//输入:一个由中间语言表示的规则 DOACROSS 循环 L
//输出:最佳分块大小  $b_{opt}$ 
 $b_{opt} = 1$ ;
使用循环选择算法从循环 L 中选出计算划分层  $l_1$  和循环分块层  $l_2$ ;
if  $l_1 == \text{NULL} \parallel l_2 == \text{NULL}$  then
    return  $b_{opt}$ ;
else begin
    使用  $b_{opt1}$  计算公式计算分块大小;
    if  $b_{opt1} \geq 1$  then
         $b_{opt} = b_{opt1}$ ;
    else begin
        使用  $b_{opt2}$  计算公式计算分块大小;
         $b_{opt} = b_{opt2}$ ;
    end
    return  $b_{opt}$ ;
end
end Pipeline_Grularity_Optimize
```

该算法以规则 DOACROSS 循环 L 的中间表示作为输入,对其进行分析,输出最优分块大小 b_{opt} 。

下面使用流水粒度优化算法对如下所示的 FDTD 典型循环进行分析:

```
do i = 2, iend - 1
    do j = 2, jend - 1
        do k = 2, kend - 1
            invmu = 1.0/mu(i,j,k)
            tmpx = rx * invmu
            tmpy = ry * invmu
            tmpz = rz * invmu
            hx(i,j,k) = hx(i,j,k) + tmpz * (ey(i,j,k+1) - ey(i,j,k)) - tmpy * (ez(i,j+1,k) - ez(i,j,k))
            hy(i,j,k) = hy(i,j,k) + tmpx * (ez(i+1,j,k) - ez(i,j,k)) - tmpz * (ex(i,j,k+1) - ex(i,j,k))
            hz(i,j,k) = hz(i,j,k) + tmpy * (ex(i,j+1,k) - ex(i,j,k)) - tmpx * (ey(i+1,j,k) - ey(i,j,k))
        end do
        do k = 2, kend - 1
            invp = 1.0/ep(i,j,k)
            tmpx = rx * invp
            tmpy = ry * invp
            tmpz = rz * invp
            ex(i,j,k) = ex(i,j,k) - tmpz * (hy(i,j,k) - hy(i,j,k-1)) + tmpy * (hz(i,j,k) - hz(i,j-1,k))
            ey(i,j,k) = ey(i,j,k) - tmpx * (hz(i,j,k) - hz(i-1,j,k)) + tmpz * (hx(i,j,k) - hx(i,j,k-1))
            ez(i,j,k) = ez(i,j,k) - tmpy * (hx(i,j,k) - hx(i,j-1,k)) + tmpx * (hy(i,j,k) - hy(i-1,j,k))
        end do
    end do
end do
```

首先使用循环选择算法从该循环的循环层中选出计算划分层和循环分块层,分别为 i 层和 j 层;然后使用 b_{opt} 计算公式,得到的分块小于1,则对计算划分层 i 层进行分块,继续使

用 b_{opt2} 计算公式,得到最佳分块大小。因此该循环在进行流水并行时需要减小流水粒度,得到的并行循环如下:

```
! $omp parallel default(shared) private(i,j,ii,k,invmu,invep,
! $omp& tmpx,tmpy,tmpz) shared(hx,hy,hz,ex,ey,ez,mu,ep)
mthreadnum = 1
! $mthreadnum = omp_get_num_threads()
iam = 1
! $iam = omp_get_thread_num() + 1
isync(iam) = 0
! $omp barrier
do i = 2, iend - 1, b
    do j = 2, jend - 1
        call sync_left(iend,jend,kend,hx,hz)
        ! $omp do schedule(static)
        do ii = i, min((i+b-1),(iend-1))
            do k = 2, kend - 1
                invmu = 1.0/mu(i,j,k)
                tmpx = rx * invmu
                tmpy = ry * invmu
                tmpz = rz * invmu
                hx(i,j,k) = hx(i,j,k) + tmpz * (ey(i,j,k+1) - ey(i,j,k)) - tmpy * (ez(i,j+1,k) - ez(i,j,k))
                hy(i,j,k) = hy(i,j,k) + tmpx * (ez(i+1,j,k) - ez(i,j,k)) - tmpz * (ex(i,j,k+1) - ex(i,j,k))
                hz(i,j,k) = hz(i,j,k) + tmpy * (ex(i,j+1,k) - ex(i,j,k)) - tmpx * (ey(i+1,j,k) - ey(i,j,k))
            end do
            do k = 2, kend - 1
                invp = 1.0/ep(i,j,k)
                tmpx = rx * invp
                tmpy = ry * invp
                tmpz = rz * invp
                ex(i,j,k) = ex(i,j,k) - tmpz * (hy(i,j,k) - hy(i,j,k-1)) + tmpy * (hz(i,j,k) - hz(i,j-1,k))
                ey(i,j,k) = ey(i,j,k) - tmpx * (hz(i,j,k) - hz(i-1,j,k)) + tmpz * (hx(i,j,k) - hx(i,j,k-1))
                ez(i,j,k) = ez(i,j,k) - tmpy * (hx(i,j,k) - hx(i,j-1,k)) + tmpx * (hy(i,j,k) - hy(i-1,j,k))
            end do
        end do
        ! $omp end do nowait
        call sync_right(iend,jend,kend,hx,hz)
    end do
end do
! $omp end parallel
```

3 实验结果与分析

本章对本文实现的流水粒度优化算法的有效性进行测试。本课题的前期工作在 Open64 编译器的后端实现了基于 OpenMP 的规则 DOACROSS 循环流水并行代码的自动生成。本文在此基础上实现了流水粒度优化算法,对自动生成的流水并行代码的流水粒度进行优化。

测试平台为 IBM x3650 系列的服务器,其中包含4个 Intel Xeon X5670 CPU,每个 Intel Xeon X5670 CPU 中包含6个主频为 2.93 GHz 的处理器核,内存为 40 GB,使用的操作系统为 Red hat Enterprise 5.5。选择 FDR 的波前循环和求解电磁问题的 FDTD 的典型循环作为测试用例。物理学和其他学科领域的许多问题在被分析研究之后,往往归结为偏微分方程

(Partial Differential Equation, PDE) 的求解问题。直接求解 PDE 比较困难, 所以常常使用有限差分法 (Finite Differential Method, FDM) 来实现 PDE 的数值求解, 在 PDE 中用差商代替偏导数, 得到相应的差分方程, 通过解差分方程得到 PDE 的近似解。FDM 被广泛应用于诸如气象预报、流体力学的模拟、弹流力学等研究领域。FDM 中采用各种迭代法, 如点逐次超松弛方法、迭代的交替方向隐式方法等, 求解差分方程组, 其中蕴含大量的 DOACROSS 并行性, 适用于 DOACROSS 循环自动并行的测试。本章使用的两个测试用例均属于 FDM 的迭代求解法。

3.1 对 FDR 的测试

本节对 FDR 波前循环进行测试, 测试时选择 256×256 的循环规模, 并记录循环在不同线程数目下的三种加速比, 分别是: 使用手工选择最优分块大小时的并行加速比 (by hand)、基于传统循环分块方法选择分块大小时的加速比 (traditional) 和基于本文的粒度优化算法选择分块大小时的加速比 (automatic), 结果如图 7 所示。

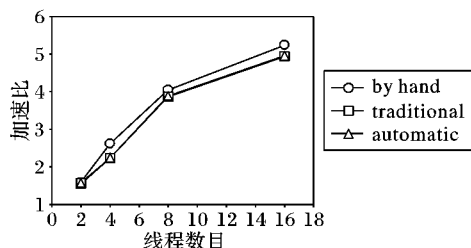


图7 FDR 波前循环分块大小选择测试结果

由图 7 所示的 FDR 波前循环的测试结果可看出, 基于传统循环分块方法选择分块大小时的加速比和基于本文的粒度优化算法选择分块大小时的加速比两条曲线重合, 表明对于该循环, 传统方法和本文算法在各线程数目下求得的最优分块大小相同。FDR 波前循环的循环体较小, 需要使用传统的循环分块方法来增大并行时的流水粒度, 因此出现了图 7 所示的情况。另外, 从图中可得: 自动选择分块大小时的加速比在不同线程数目下能够达到手工选择分块加速比的 85.0%, 在 2 线程时最为接近, 为 98.2%。

3.2 对 FDTD 的测试

本节对 FDTD 典型循环 512×512 的规模进行测试。与 3.1 节类似, 测试时分别记录循环在不同线程数目下的三种加速比。将三个加速比进行对比, 结果如图 8 所示。需要指出的是, 在图 8 中还给出了 FDTD 典型循环在 128×128 规模时的加速比曲线用于对比。

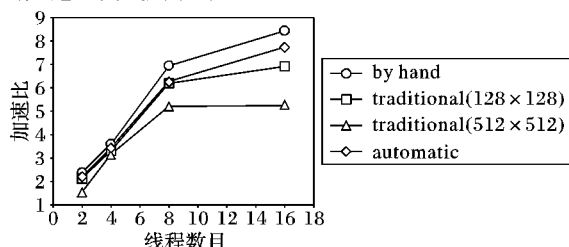


图8 FDTD 典型循环分块大小选择测试结果

根据图 8 的测试结果, 首先对比使用传统循环分块方法时两种不同规模循环的流水并行加速比, 发现循环规模为 128×128 时的加速比优于循环规模为 512×512 时。一般情况下, 循环迭代规模越大, 流水并行的同步开销占循环执行总开销的比例越小, 从而在并行线程数目确定时, 流水并行循环的性能加速随着循环迭代规模的增大而愈加明显。出现不符合上述规律的情况是因为使用传统循环分块选择方法在 512×512 规模下无法选择出最优的循环分块大小。使用本文

算法后, 加速比有明显提升, 与传统方法相比, 在不同线程数目时, 加速比均提升 10% 以上, 在 16 线程时提升最为显著, 达到 47.5%。另外, 从图中可得: 自动选择分块大小时的加速比在不同线程数目下能够达到手工选择分块加速比的 90.0%, 在 4 线程时最为接近, 为 95.6%。

4 结语

本文针对流水并行代码自动生成过程中传统流水粒度选择方法的不足, 提出了一种流水粒度优化算法, 以进一步提升自动生成的流水并行代码的性能。本文首先介绍了传统的基于循环分块增大流水粒度的方法, 然后详细讨论了在流水粒度过大的情况下如何基于循环分块减小流水粒度, 之后根据流水并行代价值模型得到了求解最优循环分块大小的方法, 最后给出了完整的流水粒度优化算法。实验结果表明, 与传统的流水粒度选择方法相比, 本文算法能够得到更优的循环分块大小。下一步工作的重点是面向基准测试集进行大量测试, 根据所得的实验数据, 对流水并行代价值模型进行进一步修正, 以求得更加精确的最优分块大小, 提升自动生成代码的并行性能。

参考文献:

- [1] BENOIT A, MELHEM B, RENAUD-GOUD P, *et al.* Power-aware Manhattan routing on chip multiprocessors [C]// Proceedings of 2012 IEEE 26th International Parallel and Distributed Processing Symposium, Piscataway: IEEE, 2012: 189–200.
- [2] JIN H Q, JESPERSEN D, MEHROTRA P, *et al.* High performance computing using MPI and OpenMP on multi-core parallel systems [J]. Parallel Computing, 2011, 37(9): 562–575.
- [3] BONDHUGULA U K R. Effective automatic parallelization and locality optimization using the polyhedral model [D]. Ohio: The Ohio State University, 2008.
- [4] AKHTER S, ROBERTS J. Multi-core programming: increasing performance through software multi-threading [M]. Hillsboro: Intel Corporation, 2006: 13–27.
- [5] CYTRON R. Doacross: beyond vectorization for multiprocessors [C]// Proceedings of the 1986 International Conference on Parallel Processing, Piscataway: IEEE, 1986: 836–844.
- [6] CHEN D-K, YEW P-C. An empirical study on DOACROSS loops [C]// Proceedings of Supercomputing, New York: ACM, 1991: 620–632.
- [7] HURSON A R, LIM J T, KAVI K M, *et al.* Parallelization of DO-ALL and DOACROSS loops — a survey [J]. Advances in Computers, 1997, 45: 53–103.
- [8] LIN Y-T, WANG S-C, SHIH W-L, *et al.* Enable OpenCL compiler with Open64 infrastructures [C]// 2011 IEEE 13th International Conference on High Performance Computing and Communications, Piscataway: IEEE, 2011: 863–868.
- [9] 富弘毅, 丁滢, 宋伟, 等. 一种利用并行复算实现的 OpenMP 容错机制 [J]. 软件学报, 2012, 23(2): 411–427.
- [10] THOMAN P, JORDAN H, PELLEGRINI S, *et al.* Automatic OpenMP loop scheduling: a combined compiler and runtime approach [C]// IWOMP'12: Proceedings of 8th International Conference on OpenMP in a Heterogeneous World, Berlin: Springer-Verlag, 2012: 88–101.
- [11] ALLEN R, KENNEDY K. Optimizing compilers for modern architectures: a dependence-based approach [M]. San Francisco: Morgan Kaufmann Publisher, 2001: 63–68.
- [12] TAFLOVE A. Computational electrodynamics [M]. London: Artech House Publishers, 1995.
- [13] 马琳. 反馈指导的流水计算性能调优 [D]. 北京: 中国科学院计算技术研究所, 2005.