计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(10):82−88 [doi: 10.15888/j.cnki.csa.007605]
©中国科学院软件研究所版权所有.
E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于标记语言的跨平台并行编程框架设计①

唐佩佳[1,2], 徐　云[1,2], 钟旭阳[1,2]

[1](中国科学技术大学 计算机科学与技术学院, 合肥 230027)
[2](安徽省高性能计算重点实验室, 合肥 230026)
通讯作者: 唐佩佳, E-mail: pjtang@mail.ustc.edu.cn

摘　要: 大量遗留的串行代码需要进行并行化改造, 而并行程序复杂性及并行计算平台多样性导致改造成本较高. 为此, 设计了一种基于标记语言的三层并行编程框架, 完成了从串行程序层到并行中间代码层、并行中间代码层到目标并行编程语言程序层的二个转换阶段. 采用对串行代码进行语言标记的方法来实现并行中间代码层, 该代码层实际是共享存储、分布式存储并行平台编程语言的一种抽象. 该框架还实现了一种性能标记方法, 可用于并行参数自动寻优. 用于雷达数据处理的实验结果表明, 实现了对应并行代码的生成, 且并行加速比与人工实现的并行代码相当.

关键词: 标记语言; 跨平台; 并行编程框架; 并行计算; 串行程序

引用格式: 唐佩佳,徐云,钟旭阳.基于标记语言的跨平台并行编程框架设计.计算机系统应用,2020,29(10):82–88. http://www.c-s-a.org.cn/1003-3254/7605.html

Design of Cross Platform Parallel Programming Framework Based on Directives

TANG Pei-Jia[1,2], XU Yun[1,2], ZHONG Xu-Yang[1,2]

[1](School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)
[2](Key Laboratory of High Performance Computing of Anhui Province, Hefei 230026, China)

Abstract: A large number of legacy serial codes need to be parallelized, and the complexity of parallel programs and the diversity of parallel computing platforms lead to high cost of transformation. For this reason, a three-layer parallel programming framework based on markup language is designed, which completes the two transformation stages from serial program layer to parallel intermediate code layer and from parallel intermediate code layer to target parallel programming language program layer. The method of language marking of serial code is used to realize the parallel intermediate code layer, which is actually an abstraction of the programming language of the parallel platform of shared storage and distributed storage. The framework also implements a performance tagging method, which can be used for automatic optimization of parallel parameters. The experimental results for radar data processing show that the corresponding parallel code is generated, and the parallel speedup ratio is equivalent to that of the manual parallel code.

Key words: directives; cross-platform; parallel programming framework; parallel computing; sequential code

并行计算是目前主流的提高计算机系统计算速度和处理能力的一种有效手段, 可将原有的串行程序改造为并行程序来解决性能问题, 充分利用并行计算技术提高数据处理能力. 而串行程序并行化的过程需要专业的并行编程能力, 需要大量的工程成本. 同时, 随着各种并行计算平台数量的增加, 并呈现出多样化, 需要保证并行程序有一定的跨平台性, 使得并行程序可在多个并行平台上跨平台运行. 然而, 不同的并行计算

① 基金项目: 国家自然科学基金面上项目 (61672480)
Foundation item: General Program of National Natural Science Foundation of China (61672480)
收稿时间: 2020-02-19; 修改时间: 2020-03-17; 采用时间: 2020-03-27; csa 在线出版时间: 2020-09-30

© 中国科学院软件研究所 http://www.c-s-a.org.cn

平台的硬件、所适用的并行模型和运行时环境都各不相同, 这对程序的可移植性提出了严峻的挑战.

串行程序并行化的工具或系统可以分为两类: 一类是采用专用编译器等自动并行化工具直接将串行程序自动并行化, 编译器开发成本较高, 而并行效率不高. 比如斯坦福大学的 SUIF 编译器[1] 和 Mathews 的 ACP 编译器[2], 为了保持原有串行程序语义, 导致编译器的策略非常保守, 实际效果不太理想. 其他工具例如 Cetus[3]、Par4all[4]、Pluto[5] 仅可实现循环级别的并行, 不支持任务并行. S2P[6] 虽然提供了任务级并行, 但其并行粒度没有优化. 现有自动并行化工具支持的并行平台较少, 仅仅包括共享存储平台的少数系统和少量异构平台. 另一类是采用加编译制导的编译器或并行编程框架, 对串行程序进行辅助标记来实现并行化.

常见的并行编程平台主要有共享存储和分布式存储两类. 共享存储平台下, OpenMP 采用在串行程序中添加并行标记语句的方法来实现并行化, 这种方法会产生较好的并行效果但需要一定的标记工作量, 系统好坏取决标记方法和工作量的大小. Blumofe 提出的 Cilk/Cilk++[7,8] 是一种基于 C/C++的多线程并行编程模型, 是在原有 C/C++的基础上提供和并行相关的制导标记, 实现任务分解和并行. Habel 等 [9] 实现了一种基于标记的分布式并行编程框架的设计, 该框架提供了数据来源和分布的标记, 通过源到源编译器将标记解析为并行程序, 但该框架仅支持矩阵密集型的科学计算. 分布式存储平台下, MPI 并行编程模型采用消息传递机制完成并行计算和同步通信; 以 FORTRAN90 为基础的 HPF[10] 并行编程模型, 通过制导语句实现数据并行制导和数据映射制导. Sourouri 提出的 Panda[11] 异构并行编程框架可将带标记的串行代码转化为 MPI 或 MPI+CUDA 并行代码, 但该框架缺少运行时系统.

同时支持共享存储和分布式存储的并行编程框架不多见, 这些系统和框架往往与应用任务特性相关或一些特定并行编程模型相关, 实现或生成的并行程序效率也是一个问题. Liu 等[12,13] 提出了一个基于任务和管道 (TNC) 模型的并行编程框架. 应用程序分为主程序层和任务实现层. 主程序层与平台无关, 框架提供定义和启动任务和管道的接口; 任务实现层与平台相关, 具体各类型任务和管道由平台相关编程方法实现, 框架再将任务和管道封装, 管道负责各任务间的通信. 应用程序主程序层可以保持不变, 任务实现层根据不同目标平台调整, 从而提高并行程序的维护和可移植性, 但该方法在分布式环境下通信开销较大. Kutil[14] 提出了基于面向对象的 C++的编程框架, 支持共享存储平台和分布式存储平台. 该方法将矩阵运算分解为无通信的子运算, 由框架负责具体的数据通信, 但该方法的并行化设计仅针对矩阵运算领域, 无法扩展到其他计算任务.

为了解决上述问题, 本文提出了一种基于标记语言的三层并行编程框架. 我们对共享和分布式存储编程平台的基本并行特性进行抽象, 提取各并行编程平台并行程序的基本结构和共同点形成跨平台的标记语言. 实现了二个层间转换过程, 一个是串行程序层到并行中间代码层转换, 采用对串行程序进行并行标记得到有并行语义的程序代码; 另一个是并行中间代码层到目标并行编程语言程序层转换, 采用对并行中间代码进行解析映射的方法, 辅助并行编程框架实现目标并行平台的并行程序转换.

1 框架总体结构

1.1 框架的层次模型

为了使框架结构清晰, 设计了一个跨平台的分层并行编程模型, 共有 3 层, 分别是: 上层、中层、和下层. 如图 1 所示.

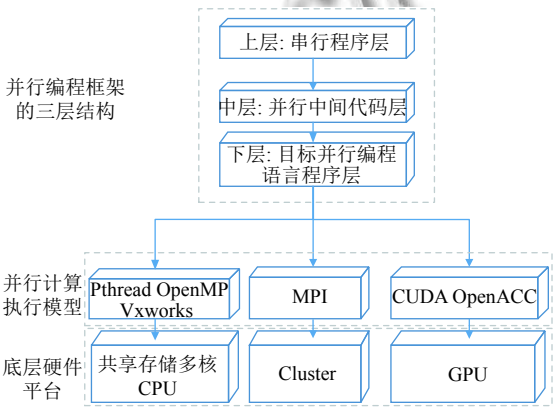

图 1 编程框架的层次模型

上层是用户的串行程序层, 是平台无关的. 中层是与串行程序对应的、带有制导标记的中间代码, 有并行语义但与平台无关. 下层是目标并行编程语言程序层, 可在并行计算执行模型上运行, 与平台相关.

并行计算执行模型主要包括: 成熟的通用并行计算执行模型, 例如 Pthread、OpenMP、MPI 和 CUDA、

© 中国科学院软件研究所 http://www.c-s-a.org.cn

OpenACC 等; 特殊应用并行编程模型, 例如 Vxworks 下多任务并行编程模型.

底层硬件平台主要包括共享存储多核 CPU 平台、分布式存储 Cluster、异构 GPU 三种类型.

1.2 框架处理流程

利用框架将一个串行程序实现并行化的流程如图 2 所示. 首先输入选择待转化的串行程序文件, 进入第一个层间转换阶段: 在串行程序定义的位置添加并行标记或者在需要并行参数寻优和同步控制等功能的位置添加相应的标记, 得到有并行语义的程序代码. 然后进入第二个层间转换阶段: 并行编程框架将逐一解析各并行标记, 结合用户界面输入的目标平台参数, 经过一系列映射过程后得到目标并行编程语言程序.

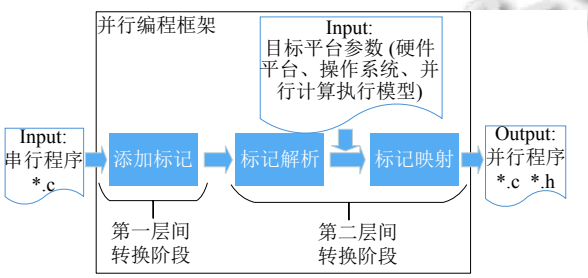

图 2　编程框架的处理流程模型

2 跨平台并行化实现

对串行程序进行简单标记, 再经过解析映射得到能够并行地在共享存储平台、分布存储平台和 GPU 平台等多种平台上正常运行的目标并行编程语言程序. 主要研究基于标记的串行 API 跨平台并行化方法, 包括两个层间转换过程: (1) 并行中间代码生成; (2) 基于标记信息的解析映射.

2.1 并行中间代码生成

标记语言用于辅助并行编程框架实现目标并行平台的并行程序转换, 是并行编程框架的重要组成部分. 我们对多种并行平台语言的基本并行特性进行抽象, 因此标记语言是一种可表示并行语义的跨平台标记语言, 命名为 Sigma 标记语言 (以下简称标记语言). 以简洁易用、功能齐备的指导原则, 设计了类似 OpenMP 的三段式标记语言. 标记语言的指令结构如图 3 所示.

标记字段:	#sigma	directive-name	clause [clause …]
说明:	标记前缀	标记名 用于标识标记	标记子句 用于提供标记参数

图 3　标记语言结构

标记语言共有 5 类 19 条, 包括并行、辅助并行、性能、平台等类型的标记, 标记语言与平台无关.

本文研究的工程背景是雷达数据处理. 以雷达数据为代表的此类工业数据有以下几个特点:

(1) 数据规整. 每个数据包都有固定的大小, 包含多批固定长度和类型的数据.

(2) 数据独立性. 各个数据包彼此独立不相关, 可以单独处理, 对有的处理过程, 数据包的各批次数据可以被单独处理.

(3) 周期性. 每个数据包按固定周期到达.

针对上述数据特点, 并行方式采用数据并行. 并行标记设计为#sigma parallel_task [标记子句].

并行标记提供了数据并行所需要的信息, 有 4 个标记子句:

(1) src_data: 提供串行 API 输入数据源、数据类型和数据长度;

(2) dst_data: 提供串行 API 输出数据源、数据类型和数据长度;

(3) group: 提供分组 (批) 数, 即数据源可以划分为若干批或组来相对独立的处理;

(4) varlist: 串行 API 的其他参数.

采用对串行程序进行并行标记得到有并行语义的程序代码, 即第一层间转换阶段.

在一个需要并行化的数据处理串行 API 的定义位置添加并行标记得到并行中间代码, 如图 4 所示.

```
#sigma parallel_task src_data (srcname; srcdatatype; srcsize)
                     dst_data (dstname; dstdatatype; dstsize)
                     group (num)
                     varlist (var1, var2)
Func_serial (srcname, dstname, num, var1, var2)
{
... ...
}
```

图 4　并行中间代码

2.2 基于标记信息的解析映射

采用对并行中间代码进行解析映射得到目标并行编程语言程序, 即第二层间转换阶段.

用户在并行编程框架界面选择目标平台参数 (硬件平台、操作系统和并行计算执行模型等) 后, 框架将 2.1 中的并行中间代码进行解析映射可以得到目标平台并行语言编程程序. 具体步骤如下:

(1) 标记解析. 并行编程框架将并行中间代码进行解析, 得到标记提供的数据并行参数, 如输入数据源

© 中国科学院软件研究所　http://www.c-s-a.org.cn

srcname、输入数据源大小 srcsize、输出数据源 dstname、输出数据源大小 dstsize、分批数 num 和其他参数;

(2) 标记映射. 并行编程框架将并行中间代码映射为基于数据并行的目标平台并行语言编程程序,

框架对多种并行平台语言的基本并行特性进行抽象. 以 Pthread 多线程线程池并行编程模型和 MPI 并行编程模型为例, 展示共享存储平台和分布式平台的并行程序架构对比如表 1 所示.

表 1 共享存储平台和分布存储平台并行程序结构对比

序号	共享存储下多线程线程池并行编程	分布存储下MPI并行编程
1	//主线程执行…	//各进程独立执行…
2	//初始化并行环境 Threadpool_INIT()	//初始化MPI环境 MPI_INIT();
3	//数据分配到线程 //数组操作	//数据分发 MPI_Scatterv()
4	//绑定Func_serial()的线程进行数据计算	//数据计算 Func_serial()
5	//数据从各线程回收 //数组操作	//数据收集 MPI_Gatherv()
6	//结束多线程环境 Threadpool_destroy()	//结束MPI环境 MPI_FINALIZE()
7	//主线程执行…	//各进程独立执行…

由表 1 可知, 并行程序的主要结构包括数据划分和分发、调用原串行 API、数据收集等部分. 其他可用于数据并行的 OpenMP 模型和 Vxworks 系统并行模型等并行编程也满足类似结构.

因此, 并行编程框架根据这个主体结构封装了多个目标平台下的基础并行代码段, 主要用于并行初始化、数据分发和收集、并行计算执行等. 这些代码段将使用上述 (1) 标记解析得到的并行参数. 标记映射过程, 即目标并行平台代码生成过程如算法 1 所示.

算法 1. 并行标记映射算法

输入: 数据并行参数 srcname、srcsize、dstname、dstsize、num 等, 串行 API 名称, numprocs 为并行处理单元数 (共享存储平台的计算单元是线程, 分布式存储平台的计算单元是节点进程); 目标平台参数 (硬件平台、操作系统和并行计算执行模型等).
输出: 并行代码.

1) 写初始化部分代码. 主要实现并行初始化和临时变量定义, 变量空间申请, 框架根据不同的目标平台使用不同的并行初始化代码段.
2) 写数据划分部分代码. 主要实现将 num 批数据划分为 numprocs 份; 划分方式有静态和动态划分两种, 框架根据用户选择的划分方式调用不同的函数实现, 函数参数将使用 num 和 numprocs.
3) 写数据分发部分代码. 主要实现将数据分发到每个并行处理单元. 共享存储平台使用自定义函数 Data_trans, 以传递存储地址实现, 分布式存储平台的数据分发使用 MPI 自带消息传递函数, 以传消息实现. 框架根据不同的目标平台调用不同的数据分发代码段, 代码段将使用 srcname 和 srcsize 参数.
4) 写数据处理部分代码. 主要实现数据的并行处理. 共享存储平台以线程方式调用串行 API, 分布式存储平台以进程方式调用串行 API. 框架根据不同的目标平台调用不同的并行计算执行代码段.
5) 写数据收集部分代码. 方法与 3) 类似, 主要实现收集各并行计算单元结果. 框架提供的代码段将使用 dstname 和 dstsize 参数.
6) 写结束收尾部分代码. 主要实现变量空间回收和并行环境结束工作.

算法 1 的第 2) 步中的数据划分方式有静态划分和动态划分. 静态划分是均匀划分, 数据划分的块数与并行处理单元数相等. 动态划分指的是根据并行参数自动寻优的结果来进行划分, 并行参数寻优的方法见 3.2.假设有 4 个并行处理单元 (共享存储平台的计算单元是线程, 分布式存储平台的计算单元是节点进程) 来处理一个数据集, 两种划分方式如图 5 所示.

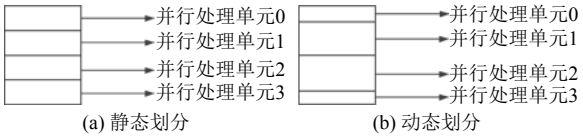

图 5 数据划分方式示意图

划分后的数据是相对独立的, 在数据处理过程中各并行处理单元之间没有数据交换和同步, 不会出现数据竞争. 对于某些有数据交换和同步需求的程序, 参考 fork/join 模型, 将源程序中数据交换和同步部分保持不变, 仍然采用串行执行, 将剩余部分封装为一个或多个新的 API, 再对新 API 进行并行化. 原串行程序的可并行部分通过并行化提高了运行效率, 不可并行部分实现了数据交换和同步功能, 在保证原程序的总体功能实现的前提下实现了原串行 API 的并行化. 第 4 节中的副瓣对消计算实例即按此方法实现.

算法 1 的第 4) 步中共享存储平台的多线程实现所涉及的并行计算执行代码段包括: Pthread 模型、OpenMP 模型和 VxWorks 系统并行模型, 分别使用 Pthread 线程池、OpenMP 多线程制导和 VxWorks 多任务实现.

算法 1 的第 1) 和 6) 步在全部并行计算完成前仅需执行一次, 为了提高易用性, 将算法 1 的 2) 到 5) 封装为并行 API 以供多次调用, 同时保持并行 API 和串行 API 保持参数列表的一致, 增加了可读性.

针对 GPU 平台并行程序的特殊程序结构, 即 host/device 混合编程模型, 并行框架增加了面向 GPU 平台的 kernel 标记手段, 可将 kernel 标记解析映射为 CUDA API 并行实现.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

3 并行参数自动寻优

并行参数自动寻优的目的是帮助用户选择最优的并行参数, 优化并行效果. 并行参数包括线程数、节点数和数据划分比例等, 综合影响因素和优化效果, 共享存储平台和分布式存储平台分别挑选线程数和数据划分比例作为优化对象. 使用并行参数自动寻优标记后, 框架将会生成目标并行平台下的并行参数自动寻优程序.

(1) 共享存储平台

框架可自动计算不同的线程数下程序的耗时, 并择优确定最佳线程数. 具体过程见算法 2.

算法 2. 共享存储平台并行参数自动选优算法

输入: 线程数集合 P=2:N, N 为物理核数, 最小耗时初始值 $T_{\min}$ 初始化为最大值;
输出: 最佳线程数 P_{opt};

1) for each 线程 p 属于 P
2) 统计多次并行函数运行平均耗时 t;
3) if $t < T_{\min}$
4) 记录最短时间 $T_{\min}=t$;
5) 记录最优线程数 $P_{\text{opt}}=p$;
6) end if
7) end for
8) return 最佳线程数 P_{opt};

标记使用方法: 将标记#sigma auto_optization_SM_start 和#sigma auto_optization_SM_end 分别加在并行域的开始和结束位置即可.

(2) 分布式存储平台:

框架可自动收集同一个算例在不同的节点运行耗时后确定最佳数据分配比例, 实现负载均衡. 具体过程见算法 3.

由节点耗时计算任务分配比例的公式如下所示:

$$L_i = 1 - \frac{T_i}{\sum_{i=1}^{N} T_i} \tag{1}$$

其中, T_i是多次标准算例运行平均耗时, N 是分布式存储平台的节点数, L_i是节点 i 的任务分配比例.

由任务分配比例计算任务量的公式如下所示:

$$W_i = \begin{cases} \lfloor W * L_i \rfloor & i < N \\ W - \sum_{j=1}^{N-1} W_j & i = N \end{cases} \tag{2}$$

其中, W 是待分配的任务总量, W_i是节点 i 所分配得到的任务量.

算法 3. 分布式存储平台并行参数自动选优算法

输入: 分布式存储集群有 N 个节点, 待分配的任务总量 W, 标准算例;
输出: 各节点的任务量 w;

1) for each node i
2) 统计多次标准算例运行平均耗时 T_i;
3) 收集其他节点的耗时 T_j, $j \in \{1{:}N\}$但 $j \neq i$;
4) 通过式 (1) 计算节点 i 的任务分配比例;
5) 通过式 (2) 计算节点 i 的任务量;
6) end for
7) return 各节点的任务量 w;

标记使用方法: 将标记#sigma auto_optization_DM 加在并行域的开始位置即可.

4 实验分析

雷达信号处理是典型的工程应用, 涉及大量的数学变换和矩阵运算, 我们选取脉冲压缩、副瓣对消、雷达图像处理 3 个处理过程来测试我们的并行框架.

首先, 对并行框架的功能性进行测试, 即考察并行框架是否能正确实现跨平台并行化.

以副瓣对消串行 API 程序并行化为例. 首先, 对副瓣对消串行 API 程序添加并行标记, 选择目标并行平台为“共享存储多核平台-Linux-Pthread”和“分布式存储平台-Linux-MPI”后, 并行框架对标记进行解析映射后, 可以生成共享存储多核平台 (Linux) 上的 线程池的并行程序和分布式存储平台 (Linux) 上的基于 MPI 的并行程序. 两个并行程序的运行结果和原串行程序一致, 证明本文的并行框架具备跨平台并行能力.

其次, 对框架生成的并行编程语言程序的性能也进行了测试.

(1) 共享存储平台

以脉冲压缩串行程序为例, 对于框架产生的 Pthread 脉冲压缩并行程序, 将并行度分别取 2、4、8、12 时, 即线程数分别取为上述值时, 统计程序运行耗时. 并行程序运行的硬件条件为: 共享存储平台 Intel(R) Core (TM) i7-8700K CPU @ 3.40 GHz, CPU 6 核 12 线程, Linux.

由程序运行耗时计算的加速比如图 6 所示, 由图可知, 两种方式产生的并行程序均取得了较好的加速比, 由于框架自动生成的并行程序的冗余代码较少, 代码规范且执行高效, 其加速比要略优于人工实现的并行程序.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

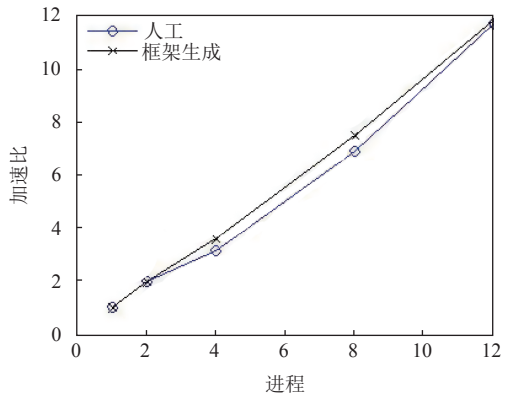

图 6 脉冲压缩并行程序取不同并行度时并行化加速比

(2) 分布式存储平台

以雷达图像处理串行程序的并行化为例, 对于框架产生的分布式图像处理并行程序, 将并行度分别取 1、8、16、24 时, 即进程数分别取为上述值时, 统计程序运行耗时. 并行程序运行的硬件条件为: 双 AMD(R) Opteron 6168 处理器, 每个处理器有 12 个计算核心, Linux 操作系统.

由程序运行耗时计算的加速比如图 7 所示, 由图可知, 由并行框架生成的并行程序的加速比与人工实现的并行程序相当.

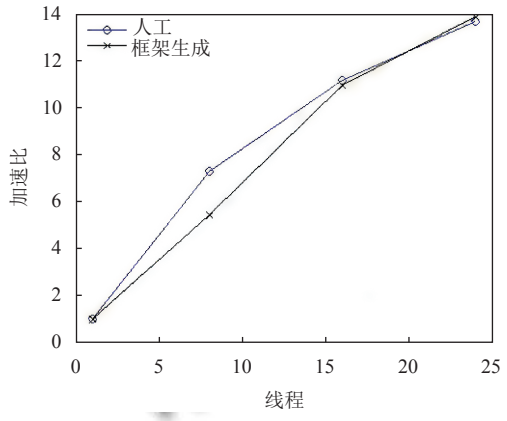

图 7 图像处理并行程序取不同并行度时并行化加速比

5 结论与展望

我们设计了一种基于标记语言的上层 (串行程序层)、中层 (并行中间代码层) 和下层 (目标并行编程语言程序层) 的三层并行编程框架. 我们对多种并行平台语言的基本并行特性进行抽象, 形成了一种可表示并行语义的跨平台标记语言. 标记语言辅助并行编程框架完成了二个层间转换过程, 实现了目标并行平台的并行程序转换, 解决了串行代码自动跨平台并行化的问题, 仅需一次标记, 即可得到多个平台的并行程序. 将框架应用于雷达数据处理程序并行化改造, 实验结果表明, 该框架可以生成目标并行编程平台的并行代码, 且经过系统参数优化的并行代码加速比相当或优于人工实现的并行代码.

参考文献

1 Wilson RP, French SS, Wilson CS, *et al*. SUIF: An infrastructure for research on parallelizing and optimizing compilers. ACM SIGPLAN Notices, 1994, 29(12): 31–37. [doi: 10.1145/193209.193217]

2 Mathews M, Abraham JP. Automatic code parallelization with openMP task constructs. Proceedings of 2016 International Conference on Information Science. Kochi, India. 2016. 233–238.

3 Dave C, Bae H, Min SJ, *et al*. Cetus: A source-to-source compiler infrastructure for multicores. Computer, 2009, 42(12): 36–42. [doi: 10.1109/MC.2009.385]

4 Amini M, Creusillet B, Even S, *et al*. Par4all: From convex array regions to heterogeneous computing. Proceedings of the Second International Workshop on Polyhedral Compilation Techniques. Paris, France. 2012.

5 Bondhugula U, Hartono A, Ramanujam J, *et al*. A practical automatic polyhedral parallelizer and locality optimizer. Proceedings of the 2008 ACM SIGPLAN Conference on Programming Language Design and Implementation. Tucson, AZ, USA. 2008. 101–113.

6 Athavale A, Ranadive P, Babu MN, *et al*. Automatic sequential to parallel code conversion. GSTF Journal on Computing, 2012, 1(4): 119–125.

7 Blumofe RD, Joerg CF, Kuszmaul BC, *et al*. Cilk: An efficient multithreaded runtime system. Journal of Parallel and Distributed Computing, 1996, 37(1): 55–69. [doi: 10.1006/jpdc.1996.0107]

8 Leiserson CE. The Cilk++ concurrency platform. The Journal of Supercomputing, 2010, 51(3): 244–257. [doi: 10.1007/s11227-010-0405-3]

9 Habel R, Silber-Chaussumier F, Irigoin F, *et al*. Combining data and computation distribution directives for hybrid parallel programming: A transformation system. International Journal of Parallel Programming, 2016, 44(6): 1268–1295. [doi: 10.1007/s10766-016-0428-3]

10 Loveman DB. High performance Fortran. IEEE Parallel & Distributed Technology: Systems & Applications, 1993, 1(1): 25–42.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

11 Sourouri M, Baden SB, Cai X. Panda: A compiler framework for concurrent CPU+GPU execution of 3D stencil computations on GPU-accelerated supercomputers. International Journal of Parallel Programming, 2017, 45(3): 711–729. [doi: 10.1007/s10766-016-0454-1]

12 Liu C, Leeser M. A framework for developing parallel applications with high level tasks on heterogeneous platforms. Proceedings of the 8th International Workshop on Programming Models and Applications for Multicores and Manycores. Austin, TX, USA. 2017. 74–79.

13 Liu C, Leeser M. Unified and lightweight tasks and conduits: A high level parallel programming framework. Proceedings of 2016 IEEE High Performance Extreme Computing Conference. Waltham, MA, USA. 2016. 1–7.

14 Kutil R. Towards an object oriented programming framework for parallel matrix algorithms. Proceedings of 2016 International Conference on High Performance Computing & Simulation. Innsbruck, Austria. 2016. 776–783.

www.c-s-a.org.cn

www.c-s-a.org.cn

© 中国科学院软件研究所 http://www.c-s-a.org.cn