·软件开发与应用·

面向地震数据处理的并行与分布式编程框架

赵长海*①　晏海华①　王宏琳②　史晓华①　王　雷①

（①北京航空航天大学计算机学院，北京 100191；②东方地球物理公司，河北涿州 072751）

赵长海，晏海华，王宏琳，史晓华，王雷. 面向地震数据处理的并行与分布式编程框架. 石油地球物理勘探，2010，45(1)：146～155

摘要　本文提出了一个适用于地震资料处理的并行与分布式编程框架 GeoPF。该框架构建在集群系统之上，采用粗粒度数据并行执行模型，它可以调度串行语言编写的处理模块，同时运行在多个计算节点或者单个节点内的多个 CPU 核上，隐藏了计算节点及其 CPU 核的调度、通讯与节点故障恢复、模块之间的数据传输等并行编程细节。经过实验评估，GeoPF 框架从串行到并行的线性加速性能有所提高，处理相同任务的时间从 21h 33min 缩减到 15min 27s，效果显著。GeoPF 与商用的地震数据处理系统相比，在业务流程方面有一些相同特点，其不同之处就是 GeoPF 的处理模块具有自动并行特点，而大部分地震处理模块只能是串行方式。

关键词　地震数据处理　并行编程　分布式编程　编程框架　粗粒度数据并行

1　概述

在石油工业领域，地球物理勘探的主要目的是获取地下构造圈闭、发现油气藏。为达到此目的，需要进行大量的地震勘测获取地下原始信息，并将这些信息经过复杂的迭代处理，从中反演出地质模型，进而推断油藏信息。在地球物理勘探中以地震勘探使用效果最好，但地震勘探的原始数据信噪比非常低，而且数据量非常大（达到 TB 级），因此地震数据处理对计算能力要求非常高。与天体演变研究、核爆炸模拟、中长期天气预报和大规模事务处理等为数不多的几个领域一样，一直属于高性能计算机的重要应用范畴[1]。

地震数据处理流程可分为两个阶段。第一阶段是应用信号处理算法提高信噪比，目前有几百种算法可应用于这个阶段；第二阶段是地震偏移成像，将地震波能量归位到其真实的空间位置，获取地下的真实构造图像，相比第一阶段，这个阶段的计算量更大，Kirchhoff 积分偏移、有限差分偏移和逆时偏移[2]属于这个阶段的算法。

地震道是地震数据的基本单位，一道地震数据的大小与采样时间长度和采样间隔有关，其容量从几千位（kB）到几十千位不等。道集是具有某一共同属性的地震道的集合，如共炮点（CSP）、共中心点（CMP）和共接收点（CRP）道集，一个道集包括几十到几千地震道不等。处理模块是地震处理算法的程序实现，在现代大多数地震处理系统中，处理模块是基本的编程单元，以动态链接库的形式存在，使用时多个处理模块首尾相连，一个执行程序需动态加载这些模块并控制数据流过模块链，不同系统的处理模块接口规范差异很大。为了并行编程的需要，本文根据算法的数据依赖性，将处理模块分为三类：

（1）单道模块　此类模块一次接收一个输入道，处理完毕后输出一道、几道或者没有任何输出；

（2）道集模块　也称为多道模块，每次接收一个道集输入，处理完毕后输出一个道集、一道或者没有任何输出；

（3）全局模块　这类模块对整个数据体都有依赖，叠前时间/深度偏移属于此类模块。

地震数据处理属于数据密集型和计算密集型兼备的高性能计算，一块勘探面积的地震数据往往需要花费数个月的处理时间。长期以来，计算性能的提高主要得益于处理器时钟频率的提升，但由于物理条件的限制，处理器主频提高的步伐缓慢，于是多核框架成为提高 CPU 性能的主流技术。然而单线

* 北京市海淀区航空航天大学计算机学院，100191
本文于 2009 年 2 月 1 日收到，修改稿于同年 6 月 24 日收到。
基金项目：国家高技术研究发展计划 863(2007AA060401)。

返回目录

程程序并不能随着处理器的升级而获得性能的提升[3]。可是并发编程非常困难[4],虽然如今部分面向特定领域的并行编程模型已经取得很大的成功[5],但仍缺乏通用的高生产率的并发编程模型能够充分发挥高性能计算机硬件的性能。U C Berkeley 的研究报告[6]指出,并行编程模型的研究应该采用"自顶向下"的研究方法,选取典型的并行应用,并以应用为驱动研究真正符合终端用户需求的模型。基于笔者经验,面向地震勘探领域的并行编程模型要取得成功,必须满足如下需求。

(1)易编程。如何为开发人员提供更加简单的编程手段,使其能够编写有效的并行与分布式程序是并行编程领域长期存在的难题。对于没有受过专业并行编程训练的地球物理领域专家来说,这个问题更加突出。一个简单并行编程模型,不仅仅意味着隐藏更多的并行编程细节,更少的代码量,而且编程模型的可理解性也非常重要,如果并行编程模型很难理解,那么就存在引发逻辑错误的可能性。

(2)兼容串行语言编写的算法。历史遗留下来大量使用 C、C++或者 Fortran 语言实现的算法,这些算法经过长时间的使用已经非常稳定,如果改换并行语言重写,势必会花费高昂的代价。

(3)适用于由多核或众核(many core)计算节点构成的集群系统。

基于上述需求,我们设计并实现了一个适用于地震资料处理的并行与分布式编程框架 GeoPF。该框架可以调度串行语言编写的处理模块同时运行在多个计算节点或者单个节点内的多个 CPU 核上。GeoPF 构建在集群系统之上采用粗粒度数据并行执行模型,隐藏了计算节点及其 CPU 核的调度、通讯与节点故障恢复、模块之间的数据传输等并行编程细节。

本文工作的主要贡献在于,提出的并行与分布式编程框架具备很好的可编程性和出色的快速运算性能。可编程性表现在地球物理领域专家可以使用 C、C++和 Fortran 等串行语言编写处理模块,不需要关注任何并行编程相关的细节,由框架的运行时系统自动调度处理模块并行执行。此外,本文提出的框架在实验以及生产中都表现出了优越的性能,如果模块的计算时间与数据 I/O 的比率足够大,随着计算节点数增多或者单节点内 CPU 核数目增多,框架的运行时系统可以达到接近线性增长的加速比。

下面首先介绍 GeoPF 的系统设计,包括执行模型、数据模型和编程模型;第二部分介绍框架的运行时系统,主要是关键技术的实现与优化;第三部分实验评估分析 GeoPF 的性能以及性能瓶颈;第四部分比较相关的研究工作;最后总结全文并给出进一步的研究方向。

2　系统设计

2.1　执行模型

GeoPF 用户有地球物理方法专家和地震数据处理操作员两类。前者负责开发处理模块、定义模块参数,并对模块属性进行描述,处理模块最终以动态链接库形式存在,经过充分测试后提交至模块库进行统一管理;后者负责编辑地震作业,即从模块库中选取合适的模块、组织处理流程和设定模块参数。上述操作都可以在作业编辑器上采用拖放(drag-and-drop)方式完成,作业编辑器会生成一个使用地震作业语言描述的作业脚本,该脚本也可以直接使用地震作业语言编写。GeoPF 的运行时系统解析作业脚本,加载指定的处理模块,然后由作业调度系统调度至可用计算节点上并执行。

图 1 为并行执行模型。GeoPF 采用如图 1b 所示的粗粒度数据并行执行模型,执行控制线程负责在运行时加载处理模块以及控制模块间的数据交换,每一个执行控制线程被映射到一个独立的 CPU 核。一个作业内的处理模块可能属于单道模块也可能是道集模块,因此将地震数据拆分为道集,以道集为数据单位分发给各个执行控制线程,这样可以保证数据流之间没有数据依赖关系,执行控制线程可以独立无干扰地执行。处理模块在控制线程内串行执行,模块必须是"无状态的",所谓无状态是指模块的输出结果只跟输入数据有关,与模块先前的执行情况没有依赖关系,文献[7]对无状态模块的特点以及设计原则进行了详细的论述。GeoPF 为提高性能,将数据流 I/O 与计算形成流水线并行,具体实现上可将数据流 I/O 操作由独立线程完成,I/O 线程与执行控制线程之间使用数据缓冲区协调其速度差异。GeoPF 作业的数据并行度由模块组处理单个道集的时间与道集数据I/O时间的比率决定,比率越大,就可以部署更多的执行控制线程加快计算

返回目录

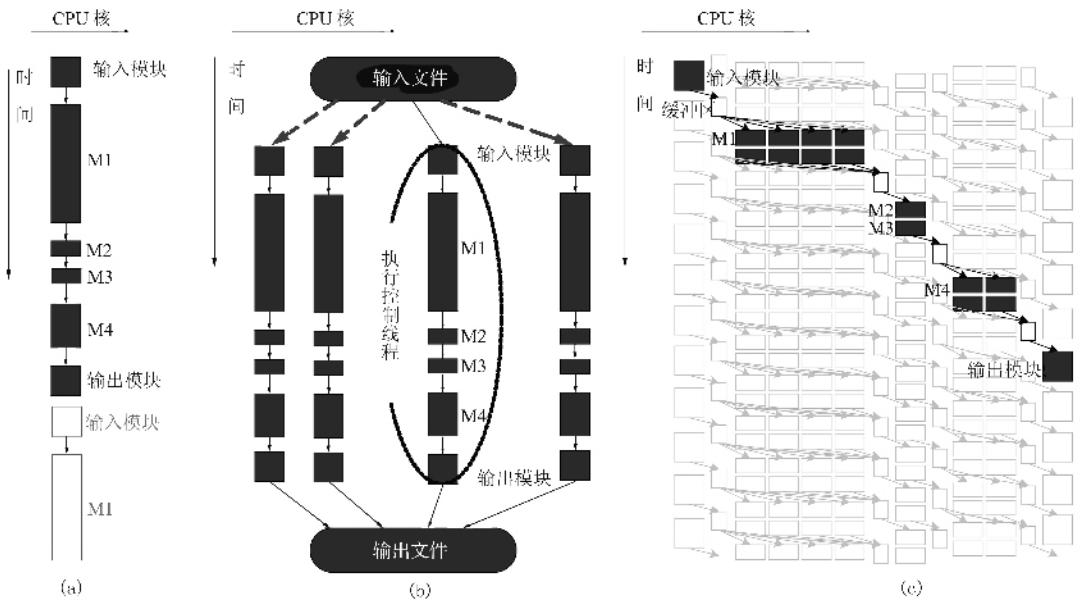

图 1 并行执行模型

(a)串行；(b)粗粒度数据并行；(c)任务、数据与流水线并行；
长方形代表处理模块，处理模块的高度反映了它的计算量

速度以匹配 I/O 速度。

GeoPF 初期设计曾经考虑实现如图 1c 所示的任务、数据与流水线并行执行模型，StreamIt[8] 流编程语言支持该执行模型，在多核架构下有出色的性能表现。该模型与粗粒度数据并行最显著的区别在于单数据流上处理模块的执行方式，前者采用流水线并行模式执行，后者在执行控制线程的协调下串行执行。在流水线并行模式下，处理模块被映射到不同的 CPU 核上，模块间采用数据缓冲区连接，前一模块作为“生产者”输出处理结果至缓冲区，后一模块作为“消费者”从缓冲区取数据作为模块的输入。为了缓解由于模块运算速度不匹配导致的流水线停顿(pipeline stall)问题，引入任务并行，对计算耗时较长的模块进行任务拆分，拆分后的任务映射到不同核上并行执行。从性能优化角度讲，该执行模型的实现可以将整个流水线部署在单个计算节点内以避免模块间的网络通信，位于不同计算节点上的流水线独立执行。直观上，相比于粗粒度数据并行，该模型可提供更深层次的并行，然而流水线上的模块间由于运算速度差异引入了额外的同步等待。此外模块之间设置的缓冲区占用了相当大的内存，对于数据密集型计算，内存是稀缺资源。最终促使我们放弃该模型的主要原因是数据 I/O 速度影响并行度，采用任务和流水线并行即使可以加快单个数据流的计算，但在数据读取速度不变的情况下，数据流的并行度就会下降。即虽然单个数据流计算加快，但整个应用程序的执行速度并没有快。下面定量分析这个问题。

假设作业内共有 m 个处理模块，第 i 个模块处理单个道集需要的 CPU 时间是 $t_i(i=1,2,\cdots,m)$，整个数据体包含 N 个道集，从磁盘阵列读取一个道集并传输至计算节点需要时间是 $t_{I/O}$，并假定道集是由一个进程串行读取然后分发(多个线程或进程并行读取磁盘阵列上的同一个文件并不会获得很好的性能，所以串行读取的假设是合理的)。如果单个数据流上模块串行执行，忽略模块间内存数据传递耗时，处理一个道集需要的 CPU 时间为

$$t_S = \sum_{i=1}^{m} t_i \tag{1}$$

如果单个数据流上模块以任务和流水线模式并行，假设作业内计算量大的处理模块可以任务并行且正好匹配计算量最小的模块，流水线因此不会停顿，在此最理想情况下，流水线停顿时间可以忽略，处理一个道集需要的 CPU 时间为

$$t_P = \min\{t_1, t_2, t_3, \cdots, t_m\} \tag{2}$$

如果作业按图 1a 所示的串行模式执行，优化实现可以将数据 I/O 与计算形成流水线并行，一般情况下 $t_{I/O} < t_S$，则数据 I/O 时间可以被计算迭代，那么作业执行时间为

$$T_a = N \times t_S \tag{3}$$

返回目录

如果作业以粗粒度数据并行模式执行,一个性能最优的数据流并行度为

$$P_b = \frac{t_S}{t_{I/O}} \tag{4}$$

在这个并行度下,数据 I/O 与计算的速度匹配,不会出现互相阻塞等待的现象。由于数据是串行读取并分发的,最后一个数据流收到数据的时间相对于第一个数据流慢 t_S,所以作业执行时间为

$$T_b = \frac{N}{P_b}t_S + t_S = N \times t_{I/O} + t_S \tag{5}$$

同理,如果作业以任务、数据和流水线并行模式执行,性能最优的数据流并行度和作业执行时间分别为

$$P_c = \frac{t_P}{t_{I/O}} \tag{6}$$

$$T_c = \frac{N}{P_c}t_P + t_P = N \times t_{I/O} + t_P \tag{7}$$

从式(4)和式(6)可以看出,虽然单个数据流执行速度加快,但是 I/O 速度不变,数据流的并行度反而下降(即 $P_c < P_b$);在式(5)和式(7)中,典型的地震数据 N 值可以达到数百万,一般常规处理模块处理一个道集的 CPU 时间远小于整个地震数据的读取与传输时间,所以两种并行模式下,作业的执行时间 T_b 与 T_c 近似相等,即粗粒度数据并行执行模式与任务、数据和流水线并行执行模式的理论性能相差不大,影响作业执行时间的主要因素是 I/O 速度。

2.2　数据模型

数据并行程序中,并行数据流的粒度选择非常重要,粒度的选择与领域的数据特征和算法的数据依赖关系密切,同时要权衡普适性和实现难度。GeoPF 选择道集作为数据并行粒度,如果作业内只有道集模块和单道模块,道集数据流之间就不会有依赖关系,并行数据流的构建与控制都是由框架运行时系统完成的,对开发人员是透明的。

单个数据流上可以同时部署道集模块和单道模块,这里面临一个复杂的问题:模块的输出与毗邻的下一模块的输入不一定匹配。根据输入输出的不同,单道模块和道集模块又可以细分为:

(1)SISO(单道输入单道输出)模块,接收一个输入道,输出一道或者没有输出;

(2)SIMO(单道输入多道输出)模块,接收一个输入道,输出多道;

(3)MISO(道集输入单道输出)模块,接收道集输入,输出一道或者没有输出;

(4)MIMO(道集输入多道输出)模块,接收道集输入,输出多道。

DataClutter[9]、Dryad[10] 和 River[11] 等分布式编程框架把模块当作“过滤器”,数据依次流过所有的模块即可获得处理结果。但是对于地震作业处理模块来说,由于模块输入输出的多样性,把模块当作“过滤器”看待是不合适的。

执行控制线程采用“数据驱动”的方式进行循环控制以满足模块的输入要求,模块间的循环分内循环和外循环。内循环根据作业内模块的输入输出特征,控制它们的运行次数以满足下一模块输入要求,使得道集数据得到完整的处理。如图 2 所示,若前一模块输出未达到当前模块的输入需求时,前一模块就会被循环执行,直到当前模块的输入需求得到满足。外循环控制较为简单,它负责输出处理完毕的道集,并请求输入下一个道集。

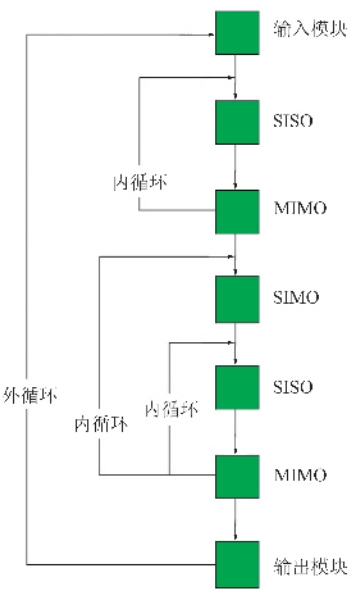

图 2　循环控制

地震数据格式是标准的,所以数据的输入和输出由 GeoPF 提供的输入模块和输出模块负责,不需要用户实现。外循环控制对用户也是透明的,输入模块检测到当前的输入道到达道集边界时,置“end-of-gather”标记,当该标记传递至输出模块,执行控制线程启动外循环。

2.3　编程模型

模块是 GeoPF 的基本编程单元,开发人员编写的处理模块必须实现“init”和“process”两个函数,

返回目录

“init”函数在模块初始化阶段被调用;“process”函数在作业执行期间反复被调用。每次从输入缓冲区内取满足计算需求的数据,处理完的数据放入输出缓冲区,模块的这种输入需求和输出状态需要开发人员显式地设置模块状态字(module status word),告知执行控制线程,后者根据该状态字执行内循环。该状态字共有三种状态值:

(1)正常状态,即本模块执行需要的数据已经满足,产生的输出已经在输出缓冲区中;

(2)等待输入状态,本模块需要的输入没有满足,等待继续输入;

(3)等待输出状态,本模块需要的输入已经满足,产生的输出部分在输出缓冲区,还有部分等待输出。

图 3 是 C 语言版本的模块编程模板,GeoPF 也支持使用 C++和 Fortran 编写模块,模块最后编译生成动态库,被执行控制线程在运行时加载。

```
/ * “init” function. The function is executed only once in the job * /
void init(int * pbuf,int * ibuf)
{
        Get_parameters();
        Code to validate parameters;
        Code to calculate the size of private buffer;
        Code for other purpose;
}

/ * “process” function. * /
void process(int * ibuf,int * kbuf,float * trace)
{
        if(there_are_enough_input_traces)
        {
            Code to process;
            Deliver_processed_traces();
            Set_MSW(); / * set module status word * /
        } else {
            Set_MSW(); / * set module status word * /
        }

}
```

图 3 模块编程模板

3 运行时系统

3.1 计算环境与运行时系统架构

GeoPF 运行时系统的实现策略依赖于具体的计算环境,我们的集群系统有如下特点:

(1)计算节点的典型配置是两个 64 位双核或四核 x86 处理器,4~8GB 内存,Linux 操作系统。

(2)集群包括 32~512 个计算节点,所有节点具有相同软硬件配置,节点之间千兆以太网互联。节点故障很常见,常见的故障源有内存、主板和磁盘,其中磁盘故障率最高,除了硬件故障外,我们还经历过磁盘占用率 100%导致系统运行异常缓慢,磁盘存在坏道导致频繁纠错引起的磁盘读性能急剧下降,文献[12]利用 Google 的基础设施对磁盘故障特征和趋势进行了统计分析,研究成果非常有价值。

(3)存储系统基于 NAS 或者 SAN,地震数据管理系统[13]管理磁盘阵列上的数据并提供数据操作接口。存放在磁盘阵列上的每一地震道都有对应的索引存放在数据库中,读地震数据之前需要读取索引,用索引定位地震数据;数据的写操作必须是串行的。

(4)用户提交作业至调度系统中(类似 Condor[14]),调度系统为作业分配相应计算节点并调度执行。

图 4 是 GeoPF 作业的执行过程,作业调度系统首先调度执行作业的 Master 进程,然后 Master 在调度系统分配的计算节点上启动 Worker 进程,Worker 创建执行控制线程(也称为计算线程)并将它们映射到节点内的 CPU 核上,每个计算线程独占一个 CPU 核。每个计算线程都加载了同样的处理模块组合,为避免模块内的全局变量或者静态变量引起线程安全性问题,必须保证处理模块处于不同的地址空间内。

3.2 数据流构建

如前所述,为避免数据流之间的依赖性,GeoPF 选择道集作为数据并行的粒度,一个道集数据大小是 10~60MB。GeoPF 支持两种读取地震数据的策略,一种是 Master 顺序读取道集数据然后分发给 Worker;另一种是 Master 分发道集的索引给 Worker,所有 Worker 用索引并行的读取道集数据。实验中后一种策略并没有提高 I/O 吞吐率,反而比前一种策略的性能略差,原因是并行读取致访问文件的位置有很强的随机性,无法利用文件系统的预取优化[15],降低磁盘的 Cache 命中率。

Worker 采用流水线并行迭代计算和数据通信,Worker 内设置多个道集数据缓冲区,接收线程将 Master 发来的数据写入空的缓冲区,计算线程从缓

返回目录

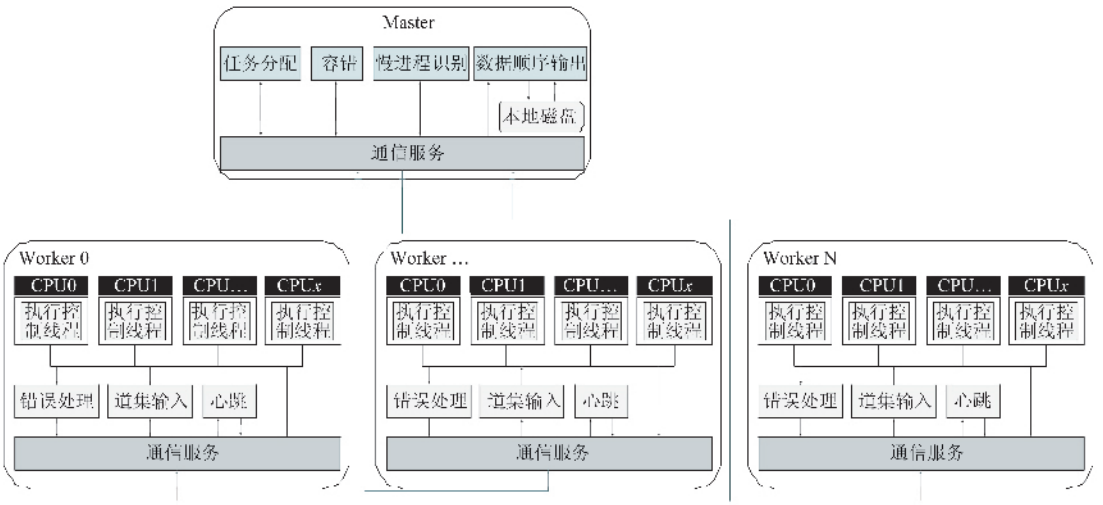

图 4　运行时系统架构

冲区读取数据,如果计算时间与 I/O 和通信的比率足够大,这种优化机制为计算线程提供持续不断的数据流,降低计算线程等待数据的时间。

Master 分发数据基于“PULL”模式,Worker 内一旦有空的道集缓冲区,就向 Master 发送请求,请求不会阻塞等待,而是立即返回。Master 将 Worker 的请求放入一个环形队列,道集分发线程采用先来先服务的原则,从环形队列中取请求并向 Worker 发送道集,如果队列为空,分发线程阻塞,这种分发模式可以保证不会分配给 Worker 过量的任务,平衡 Worker 的负载。

3.3　顺序输出

Worker 将处理过的道集发送给 Master,由 Master 顺序写入输出文件。处理过的道集在输出文件内的位置必须与其被处理前输入文件内的顺序一致,Master 发送给 Worker 道集都标记有序号,处理过的道集按此序号依次写入输出文件。

Worker 所在节点负载不同,道集内地震道数目的差异都有可能导致处理过的道集返回 Master 的顺序与道集被分发出去的顺序不同,所以 Master 需要使用本地磁盘缓存处理过的道集。一个最直观的输出策略是,在 Master 内创建一个辅助线程,该线程从磁盘缓冲区内依次读取道集,写入位于磁盘阵列的输出文件,如果按序该输出某一道集还未被 Worker 处理完,那么就形成辅助线程阻塞。这种输出策略一个最严重的问题是在同时读写磁盘的情况下,数据读取速度将会呈几十倍的下降,主要是因为文件系统写的优先级高于读的优先级,读操作被长时间延迟,导致 Master 节点缓冲区内积累大量数据无法及时写入输出文件,磁盘空间耗尽引起程序运行失败。

图 5 显示 GeoPF 采用了一种高效输出策略,用到三个线程,两级缓冲区。内存缓冲区存放即将写入输出文件的道集,假设当前需要输出的道集序号是 S,内存缓冲区内槽的个数是 N,道集序号 G 在 $S\sim S+N$范围内的即可认为是即将输出的道集,存放在第 wIdx(wIdx$=G$ Mod N)个槽内。Receiver 线程接收 Worker 处理完的道集,如果该道集是即将输出的道集则写入内存缓冲区,否则写入本地磁盘缓冲区,Receiver 线程不会被阻塞;Output 线程从内存缓冲区第 rIdx(rIdx = S++ Mod N)个槽读取道集并输出,如果槽为空,该线程阻塞;Tuning 线程周期性地将磁盘缓冲区内即将输出的道集转移至内存缓冲区。从图中可以看出,磁盘缓冲区用到了两块磁盘,以此来避免读写竞争,采取“写避让读”

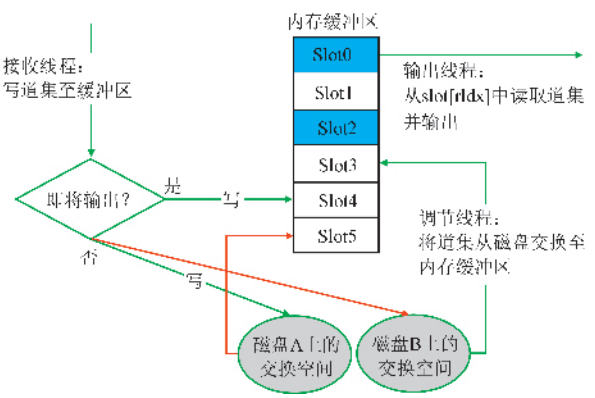

图 5　顺序输出策略示意图

返回目录

的原则，即 Receiver 线程总是绕开 Tuning 线程正在读的磁盘，将道集缓存在另外一块磁盘上。上述 GeoPF 采用输出策略有三个方面的优化效果：一是减少本地磁盘的 I/O 次数；二是避免在同一块磁盘上同时进行读写操作；三是读磁盘缓冲区与写输出文件可以并行执行。

3.4 慢进程识别与清除

集群运算环境内经常发生某个计算节点非常慢，运行于该节点的 Worker 花费过长的时间才能处理完一个道集，拖慢整个作业的运行速度，我们称这样的 Worker 进程为“慢进程”，在 MapReduce[16] 中称这样的计算节点为“straggler”，并通过任务备份机制减轻其对性能的影响。GeoPF 作业中道集是按序号依次输出，如果道集在慢进程内长时间处理不完，输出就会停滞，影响作业的整体性能。引起节点变慢的原因有很多，例如作业调度系统在该节点上分配了过多的作业，就会造成 CPU、内存、本地磁盘或者网络带宽的争用。

GeoPF 使用一个有效的方法识别和清除慢进程。Master 周期性地度量所有 Worker 的效率，Worker 的效率可以由其最近处理的几个道集所花费的平均时间来度量，假设作业有 m 个 Worker，某一 Worker 最近 n 个道集的处理时间构成集合 $W_i=\{t_{i1}, t_{i2}, t_{i3}, \cdots, t_{in}\}$，用 A_i 表示一个 Worker 的效率，O 表示所有 Worker 的平均效率，即

$$A_i=\frac{\sum_{j=0}^{n} t_{ij}}{n} \tag{8}$$

$$O=\frac{\sum_{i=0}^{m}\sum_{j=0}^{n} t_{ij}}{m\times n} \tag{9}$$

如果 $A_i>\mu * O$，该 Worker 被认定是慢进程并立即终止运行，它正在处理的道集被重新分发给其他 Worker。μ 值总大于 1，可以由用户配置，值越小对慢进程越敏感。

3.5 容错

GeoPF 作业要使用很多计算节点处理海量的数据，执行过程中节点发生故障的可能性是存在的，不仅仅是硬件能引发机器故障，也可能是某台机器运行引发的故障，所以 GeoPF 的运行时系统必须具备容错机制保障节点故障不会导致整个作业运行失败。

(1)Worker 故障　使用 Master 周期性地 ping(检测)各个 Worker 进程，如果 Worker 没有响应，Master 就将该 Worker 标记为 failed，并将该 Worker 正在处理的道集标记为 redispatch 状态，处于 redispatch 状态的道集会被优先分发给正常运行的 Worker。

(2)Master 故障　Master 采用应用级检查点机制周期性记录当前道集的处理状态和 Worker 状态等信息，一旦 Master 出现故障，下次作业从最近的检查点处开始执行。因为每个作业只有一个 Master，发生故障的概率非常小，所以我们的策略是一旦 Master 发生故障就放弃作业运行。一个图形化的客户端可以帮助用户检查 Master 故障原因并重新启动作业。

3.6 数据错误处理

地震数据记录或者处理模块参数设置有可能存在错误，此类错误可能会引起计算结果的偏差，导致决策失误，因此 GeoPF 的原则是遇到此类错误立即终止作业运行。由于面向的应用领域不同，MapReduce 选择忽略数据记录错误让程序继续运行。GeoPF 提供了类似 MPI[17] 的 MPI_Abort 和 MPI_Error_string 的函数，便于模块开发者报告和处理此类错误，Master 收到模块发来的错误消息后将错误信息写入 Log 文件，然后终止作业运行。

模块源代码中有可能忽略了此类错误，错误的数据记录或参数导致程序运行期间崩溃。为应对这种情况，在每个 Worker 内安装信号处理函数捕捉 SIGFPE、SIGSEGV 和 SIGBUS 等导致程序崩溃的异步信号，信号处理函数将当前道集的序号以及堆栈轨迹(stack trace)发送给 Master，Master 将这些信息写入 Log 文件，帮助开发人员调试模块。

4 实验评估

通过实验评估 GeoPF 的性能，并分析在什么情况下影响 GeoPF 程序可扩展性的主要因素。实验硬件为集群，包括 32 个计算节点，每个节点有两个主频为 2.6GHz 的双核 AMD Opteron 处理器(共 4 个 CPU 核)，8GB 内存，4 块 146GB 希捷 SCSI 硬盘。

实验作业由两个相同的名为 LinearNoiseRemoval 的处理模块组成，模块分别配置不同的参数，用来消除大约 13GB 地震数据里的线性噪声，该地震数据包括 442 个炮集，共 992907 个地震道，消除噪声后的数据大小与输入数据相同。处理这块数据需要 21h 33min 的 CPU 时间。

返回目录

集群内一个节点运行 Master,它在其他节点上启动 Worker,最多可使用 31 个 Worker,Worker 内创建 4 个计算线程,一个 CPU 核对应一个计算线程。作业使用 29 个节点(共 116 个计算线程)时运行速度最快,共花费 15min 27s,其中还包括 3min 作业结束时间,这个时间包括作业处理完毕后等待 Master 将输出缓冲区内的数据写入输入文件、搜集 Worker 的执行信息和写作业运行报告。

图 6 是加速比随节点数变化曲线,加速比是以串行执行时间为基准计算得到的。图 4 可见在 29 个节点之前,加速比接近线性增长,在 29 个节点处加速比达到峰值 22.9,随后加速比开始波动,并行效率下降。在计算规模恒定的情况下,有两个可能影响 GeoPF 作业的可扩展性的因素:Master 节点的通信带宽和数据 I/O。

作业运行期间 Master 通过网络接收和发送共计至少 4 倍的输入数据量,包括经由 NFS 读入地震数据、分发数据给 Worker、接收 Worker 处理完的数据和通过 NFS 写输出文件。Master 所在节点的平均网络吞吐率如图 7 所示,可以看出网络吞吐的峰值远未达到网络带宽的极限,因此网络带宽并非性能瓶颈。

图 8 所示的是数据平均读取速度与节点数的关系曲线,可以看出节点数达到一定数目之后,道集读

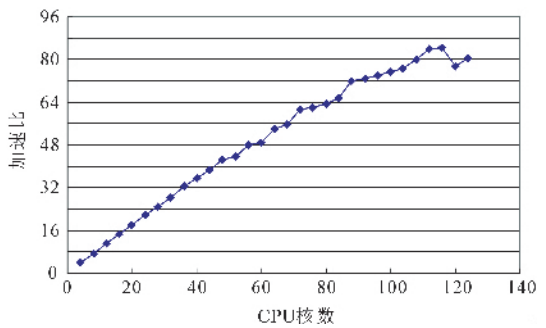

图 6　加速比与 CPU 核数的关系

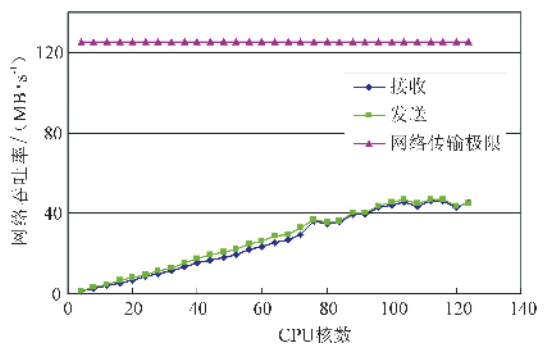

图 7　Master 节点的平均网络吞吐量

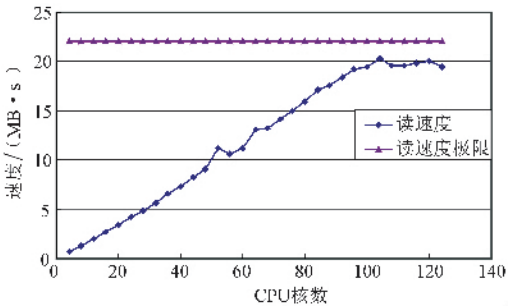

图 8　平均读速度

取速度接近极限值,无法匹配计算速度,计算线程因为缺少数据而阻塞等待。图中的数据读取速度的极限值是在 Master 节点上使用地震数据管理系统提供的数据操作接口反复读 13GB 的输入数据,来测量读取速度,取其中的最大值。为得到作业的数据平均读取速度,在 Master 的数据分发线程内记下道集开始读取分发直至分发完毕的时间间隔。前文已经提到,分发线程是基于 PULL 模式分发数据的,如果没有 Worker 请求,该线程阻塞,显然作业使用的节点数越多,处理速度越快,分发线程阻塞的时间越短。该时间间隔内每秒钟分发的数据量就是平均读取速度,由于这个时间间隔也包括了道集数据从 Master 到 Worker 的通信时间,所以该方法得到的平均读取速度比实际读取速度偏低,图 8 中的读取速度的峰值永远无法到达极限值。

从前面分析可以得出,当前 GeoPF 作业的可扩展性受限于数据 I/O 性能,但是 I/O 的优化潜力非常大,我们也正在进行这方面工作。

5　相关研究

GeoPF 主要应用于地震数据处理领域,它与商用的地震数据处理系统 GeovecteurPlus[18],Focus[19],Omega SPS[20],CubeManager[21],以及开源的 SIA[22] 和 SU[23] 相比,在业务流程方面有一些共同特点,例如,处理模块以动态库形式存在以便于维护,在运行时加载,SU 比较特殊,它的处理模块都是可执行的程序;使用地球物理语言描述模块参数和处理流程;配备有图形化用户界面的编辑、配置、提交执行和监控作业。GeoPF 与这些系统之间的最大区别是:GeoPF 中处理模块具备自动并行的特点,用户用常规的串行语言编写模块,不需要关注任何并行编程的细节,提高了并行编程的效率;而其他大部分地震数据处理系

返回目录

统的大部分模块只能串行处理数据，对于计算量比较大的算法，则需要使用 MPI，PVM[24] 或者多线程技术来实现。如 SU 系统实现了模块（应用程序）间的流水线并行，但是流水线并行模式对性能的提升非常有限，SIA 代表了最新的地震处理系统研究成果，它的并行化是通过作业“克隆”来实现，即将作业处理流程手动复制多份，部署到不同的机器上执行，每一个作业副本处理地震数据的不同部分，它也是利用地震数据的结构化特征和处理流程内涉及到的算法对数据的依赖性，对输入数据进行合理的分拆保证作业副本之间没有依赖关系，它的数据并行的粒度比 GeoPF 大得多。

GeoPF 作为粗粒度数据并行和分布式编程框架，借鉴了很多并行编程模型的研究成果，其中与 MapReduce，Dryad，River 和 DAC[25] 和 GeoPF 的研究最为相关，这些系统有两个显著特点：一是它们是面向特定领域或者某一类应用受限制的编程模型，并利用这个限制提供计算的自动化并行；二是它们要求开发人员显式地展现计算的数据依赖性。

MapReduce 分布式编程模型的灵感来自函数语言，它屏蔽了容错、数据传递和负载平衡等并行编程的细节，为数据密集型计算提供了更加简单的编程方式。MapReduce 的输入和输出数据都是 key-value 对的集合，这种数据格式是非常灵活的，可以根据特定算法的需要转换输入数据。用户需要实现 Map 和 Reduce 两个函数表达算法，用户在 Map 函数将输入数据转换为 key-value 对，具有相同 key 值的数据由 MapReduce 的运行时系统传送给 Reduce 函数，数据传输对用户是透明的；Reduce 函数接收并处理具有相同 key 的数据，生成 0 个或者多个 key-value 对；最后的输出结果根据 key 值进行排序。从 MapReduce 的语义可以看出，它非常适合数据查询处理方面的应用。Ask.com 的 DAC 框架与 MapReduce 非常类似，它也有两个类似的函数：local 和 reduce。

Dryad 提供了一种基于数据流图的粗粒度数据并行编程模型，相比 MapReduce 有更好的通用性。Dryad 的应用程序类似一个数据流图，用户编写的模块（或可执行程序）作为图的顶点，边代表数据传输通道，提供了 C++语言接口用来构建数据流图。用户编写的模块部署在很多计算节点上执行，模块间的消息和数据可以通过文件、TCP 管道或者共享内存的 FIFO 来传输。Dryad 模块使用串行语言编写，用户不需要关注并行编程的细节，相比 MPI 等并行编程语言，Dryad 并行程序的编程难度降低不少，但是它需要用户掌握数据流图操作接口，因此学习难度比 MapReduce 大，它牺牲了架构的简单性，换取更好的通用性。

River 是一个数据流编程环境，主要应用于并行数据库，解决“性能异构”导致 I/O 吞吐率下降的问题，性能异构是指集群内节点由于硬件差异或者节点负载不同引起的磁盘的 I/O 性能不一致，这是集群环境非常的一个常见问题。River 采用一种运行时自适应的机制，保证在遇到性能异构的情况下 I/O 性能不受太大影响；模块是 River 的基本编程单元，跟 GeoPF 的编程模型类似，River 运行时系统可以将模块部署在不同的计算节点上，模块间使用分布式队列传递消息和数据。River 也可应用于一些数据密集型计算，提供优化的 I/O 性能，但 River 的编程模型表达能力有限，适用的应用类型非常少。River 给我们很大启发，目前采用磁盘阵列集中存储的数据 I/O 是偏移算法的性能瓶颈，严重影响到程序的可扩展性，我们正在扩展 GeoPF 的编程模型，其功能之一是偏移计算前将地震数据分布到集群内的计算节点上，采用 River 类似的优化手段为偏移程序提供稳定的高性能的 I/O 吞吐。

6 总结和展望

GeoPF 构建在集群系统之上采用粗粒度数据并行执行模型，可以调度串行语言编写的处理模块同时运行在多个计算节点或者单个节点内的多个 CPU 核上，在实验和生产中都表现出了优越的性能。我们专注于框架的可编程性和架构的简单性，开发人员不需要掌握任何并发编程技术就可以开发出有效的并行与分布式程序。

目前框架的可扩展性受限于 I/O 性能，而且框架编程模型的可表达性受限，不适用一部分有复杂数据依赖关系的重要算法，例如叠前时间/深度偏移等，这类算法是典型的数据密集型和计算密集型计算，不仅仅数据量大，而且需要几百到几千个 CPU 核参与计算，在我们使用 MPI 实现和优化这类算法过程中，经历了 I/O、通信、故障转移、检查点、负载均衡和可扩展性的挑战，需要在硬件和软件架构等多个层次系统地解决这些问题。目前正在尝试从下面两个方向扩

返回目录

展 GeoPF 编程模型,一是改变从集中存储设备上读取数据的方式,处理前将地震数据分布到计算节点,借鉴 Google 文件系统[26]的技术和 River 的优化手段,提供高效和稳定的 I/O 性能,支持更大规模的并行;二是设计类似 MapReduce 的编程模型,把业务逻辑从故障转移、消息传递、调度等细节中抽象出来,降低开发并行和分布式偏移程序的复杂度。

参 考 文 献

[1] 张军华,仝兆岐. 地震资料处理中的并行计算机技术(综述). 物探化探计算技术, 2002,(01):31～37

[2] Sudhakar Yerneni, Suhas Phadke, Dheeraj Bhardwaj, Subrata Chakraborty, Richa Rastogi. Imaging subsurface geology with seismic migration on a computing cluster. *Current Science*, 2005, 88(3):468～474

[3] Herb Sutter. The free lunch is over: a fundamental turn toward concurrency in software. Dr. Dobb's Journal, 2005, 30(3): http://www.gotw.ca/publications/concurrency-ddj.htm

[4] Sutter H, Larus J. Software and the concurrency revolution. *ACM Queue*, 2005, 3(7):54～62

[5] Steven Fraser, Dennis Mancl. No silver bullet: software engineering reloaded. *IEEE Software*, 2008, 25(1): 91～94

[6] Krste Asanovic, Ras Bodik, James Demmel, Tony Keaveny, Kurt Keutzer, John D Kubiatowicz. The Parallel Computing Laboratory at U. C. Berkeley: a research agenda based on the Berkeley view. Technical Report, No. UCB/EECS-2008-23. 2008. http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-23.html

[7] L de Alfaro, T A Henzinger. Interface theories for component-based design. In: Proc. of EMSOFT 2001, Tahoe City, CA, Springer-Verlag, 2001,148～165

[8] Michael I Gordon, William Thies, Saman Amarasinghe. Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. In: *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, New York, NY, USA, ACM Press, 2006,151～162

[9] Michael D Beynon, Tahsin Kurc, Umit Catalyurek, Chialin Chang, Alan Sussman, Joel Saltz. Distributed processing of very large datasets with DataCutter. *Parallel Computing*, 2001,(27):1457～1478

[10] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, Dennis Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In: *European Conference on Computer Systems* (*EuroSys*), Lisbon, Portugal, 2007

[11] Remzi H Arpaci-Dusseau, Eric Anderson, Noah Treuhaft, David E Culler, Joseph M Hellerstein, David Patterson, Kathy Yelick. Cluster I/O with River: making the fast case common. In: *Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems* (*IOPADS '99*), Atlanta, Georgia, 1999, 10～22

[12] Eduardo Pinheiro, Wolf-Dietrich Weber, Luiz André Barroso. Failure trends in a large disk drive population. In: *5th USENIX Conference on File and Storage Technologies*, 2007,17～29

[13] 王宏琳. 地震软件技术——勘探地球物理计算机软件开发. 北京:石油工业出版社, 2005

[14] Douglas Thain, Todd Tannenbaum, Miron Livny. Distributed computing in practice: The Condor experience. *Concurrency and Computation: Practice and Experience*, 2004,(17):323～356

[15] Elizabeth Shriver, Christopher Small, Keith A Smith. Why does file system prefetching work? In: *Proceedings of the annual conference on USENIX Annual Technical Conference*, 1999,6～6

[16] Jeffrey Dean, Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 2008, 51(1):107～113

[17] William Gropp, Ewing Lusk, Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. *Cambridge*, MA: MIT Press, 1999

[18] CGGVeritas Corporation,CGG World 2000, at URL:http://www0.cgg.com/corporate/publications/cggworld/cggw31/cggw31.pdf

[19] Data Processing & Imaging, Paradigm Corporation, http://www.paradigmgeo.com/Content.aspx? id=47

[20] WesternGeco,Ω-Suite,WesternGeco Corporation,http://www.westerngeco.com/content/services/dp/omega

[21] CubeManager™-Petroleum Geo-Services, PGS Corporation, http://www.pgs.com/Geophysical_Services/Data_Processing/Technology

[22] Glenn Chubak, Igor Morozov. Integrated software framework for processing of geophysical data. *Computers & Geosciences*, 2006, 32(9):1403～1410

[23] AE Murillo, J Bell, Distributed Seismic Unix: a tool for seismic data processing. *Concurrency: Practice and Expericence*, 1999,11(4):169～187

[24] Geist A, Beguelin A, Dongarra J, Jiang W, Manchek R, Sunderam V. *PVM: Parallel Virtual Machine: a Users' Guide and Tutorial for Networked Parallel Computing*. Cambridge, MA: MIT Press, 1994

[25] Chu L, Tang H, Yang T and Shen K. Optimizing data aggregation for cluster-based internet services. In: *Proc. of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. San Diego, California: ACM, 2003,119～130

[26] Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung. The Google File System. In: *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, 2003,20～43

(本文编辑:任敦占)

返回目录