基于虚拟机的并行体绘制*

邓俊辉 唐泽圣

(清华大学计算机科学与技术系软件研究所 北京 100084)

E-mail: {deng,ztang}@tsinghua.edu.cn

摘要 介绍了一种基于并行虚拟机结构的体绘制算法.该算法以切片为单位来划分和组织体数据,既降低了通信代价,也保证了各子任务的数据局部性.在任务分配时,维护并使用性能指数数据库,自适应式地确定各个子任务,实现了负载平衡.使用一种异步二分方法,所有局部图像可以在$O(\log n)$时间内完成合并.针对可视化算法在虚拟机环境中的并行化实现,自行设计并实现了一个基于TCP/IP和Socket标准开发平台.所提出的算法利用该平台而实现,系统采用客户/服务器结构.对系统在任务规模、虚拟机规模方面的可扩展性进行了实际测量,并对结果进行了分析比较.

关键词 体绘制,并行虚拟机,负载平衡,可扩展性.

中图法分类号 TP391

体绘制是显示和分析复杂三维数据的重要而有效的手段,光线投射是其中一种典型的算法,其计算包括重采样和颜色合成两部分.但是,这两类计算的复杂度都很大[1],串行绘制一幅图像通常需要几十分钟到几个小时.

近年来,分布式计算得到日益广泛的应用[2],并行虚拟机(parallel virtual machine)被证明是加速众多大规模问题计算的一种有效手段[3~7].针对并行虚拟机环境,本文介绍了一种高效的光线投射并行体绘制算法.该算法基于二维切片进行数据及任务划分,保证了子任务的数据局部性,在整个系统中存储、传输的数据几乎没有冗余.与基于图像块划分的算法相比,该算法对通信信道的要求很低,尤其适用于绘制大规模体数据.

该算法动态地维护一个数据库,记录各主机的计算和存储性能指数,任务划分和数据分配是依据不同主机的性能指数,以自适应的形式进行的.因此,即使是在由异构主机构成的虚拟机环境中,该算法也可以较好地实现负载平衡.

为了将各主机并行生成的局部图像合并成最终图像,我们采用了一种异步二分合并的策略.该策略灵活、实用,同时,其计算复杂度也得到优化,n幅局部图像可以在$O(\log n)$的时间内完成合并.

1 系统结构

1.1 软件/硬件平台

系统采用了并行虚拟机结构.虽然目前有众多免费的此类平台,例如,PVM[2,6,7],Linda[8],PARMACS[9]或MPI[10,11]等,但是,由于它们大多是面向通用应用领域而设计的,难以为体绘制算法的并行化实现提供直接而高效的支持,鉴于此,我们自行设计并实现了一个面向并行可视化算法的开发平台PIPVR(programming interface for parallel volume rendering).本文的算法就是基于该平台而实现的.PIPVR以TCP/IP作为基本的通信协议,编程接口采用Windows Socket v2.2.PIPVR面向并行体绘制系统的开发,提供体数据组织、图像压缩、存储管理、通信控制以及虚拟机管理等基本功能,同时也支持三维数据场、二维图像和绘制描述信息等基本数据格式.

---

* 本文研究得到清华大学骨干青年人才计划资助.作者邓俊辉,1970年生,博士,副教授,主要研究领域为计算机图形学,科学计算可视化,计算几何.唐泽圣,1932年生,教授,博士生导师,主要研究领域为计算机图形学,科学计算可视化,计算几何.

本文通讯联系人:邓俊辉,北京 100084,清华大学计算机科学与技术系软件研究所

本文1999-12-09收到原稿,2000-03-06收到修改稿

1.2 通信模式

• 客户进程/服务线程

系统采用客户/服务程序结构,每个虚拟机由一个客户进程和多个服务线程构成.客户进程运行于本地主机,接受用户提交的绘制任务;在其他各远程主机上,分别有一个服务线程等待并接受子任务.本地客户进程可以检测当前网络中的空闲主机,并根据用户要求将对应线程组成一个并行虚拟机.服务程序支持多线程,因此,在任何时刻,同一网络环境中可以有多个虚拟机,它们相互独立,对用户透明.

根据用户的要求,本地客户进程可以在任何一个权限开放的远程主机上启动(spawn)和终止(kill)一个服务线程.客户进程为用户提供了控制绘制计算的必要手段,比如,设置图像大小、视线方向、光线方向、光照参数以及物质分类等.客户进程是整个虚拟机的核心,它负责子任务的划分和调度;控制绘制过程中各服务线程之间的同步;负责收集绘制结果,并在本地进行保存和显示;当所有绘制任务结束后,它终止所有服务线程,并解散虚拟机.

• 信号/数据

算法中的通信可以发生在本地客户进程与远程服务线程之间,也可以发生在不同的服务线程之间.从传输的内容来看,这些通信可以分为信号与数据两类.为了保证同步,系统为两类通信提供了不同的通道,无论是客户进程还是服务线程,都分别开放了一个信号端口和一个数据端口.

2 算法描述

2.1 概 述

算法分为两个阶段:(1) 由各服务线程分别绘制,得到若干局部图像;(2) 所有局部图像经过合成,得到最终结果.这两个阶段都是以并行化方式进行的.原始体数据以切片簇(pile)为单位进行划分,每一簇由若干相邻的二维切片组成,作为一个了任务指派给某个服务线程.只要服务线程接受的各簇也是相邻的,数据的局部性就可以保证,所以,在绘制阶段,服务线程之间不需要进行任何通信.

本地客户进程对各服务线程的任务分配是按递增方式进行的.各服务线程不断循环,依次计算增加的各个数据切片;它们各自维护一个局部图像缓冲,在新的切片计算完成之后,进行相应的更新.

在没有子任务剩余时,本地客户进程会通知所有服务线程开始合并图像.该阶段也是并行实现的,各服务线程之间的通信在这个阶段才出现.局部图像在传输之前都经过了压缩;当被接收后,再被解压缩.对于一个包含 n 个服务线程的虚拟机,所有局部图像的合并可以在 $O(\log n)$ 时间内完成.

2.2 重采样和图像合成的局部性

体绘制计算可以分为两类:重采样和图像合成.重采样就是通过插值等方法得到体数据中任意位置的密度值,进而根据预先的定义得到其颜色属性.

图像合成计算是基于光学模型进行的.在模型中,颜色属性包括 4 个方面的信息:R(红)、G(绿)、B(蓝)和 α(不透明度).给定沿着某条视线有 a 和 b 两个相邻的采样点,颜色属性分别为 (R_a,G_a,B_a,α_a) 和 (R_b,G_b,B_b,α_b),如果 a 离视点更近,它们对观察者的作用可以通过 over 算子表示为

$$a \text{ over } b=(R_a+(1-\alpha_a)R_b,G_a+(1-\alpha_a)G_b,B_a+(1-\alpha_a)B_b,\alpha+(1-\alpha_a)\alpha_b).$$

这种计算是与视线方向相关的,因为通常 a over $b \neq b$ over a.

如果从某个像素沿视线方向发出一条光线,那么该像素的颜色将由分布在该光线上的若干个采样点决定.比如,在从像素 p 发出的一条光线上,依次有 n 重采样点 $\{r_1,r_2,\ldots,r_n\}$,那么该像素的颜色属性可以表示为

$$r_1 \text{ over } r_2 \text{ over } \ldots \text{ over } r_n.$$

根据其定义,不难验证 over 算子满足结合律,即 a over b over $c=a$ over (b over c).将光线任意分成 m 段,每一段都包括若干相邻的采样点,若记它们为 $\{r_1,\ldots,r_{k1}\},\{r_{k1+1},\ldots,r_{k2}\},\ldots,\{r_{k(m-1)+1},\ldots,r_n\}$,那么根据结合律,该像素的颜色属性可以等效地表示为

$$(r_1 \text{ over } \ldots \text{ over } r_{k1}) \text{ over } (r_{k1+1} \text{ over } \ldots \text{ over } r_{k2}) \text{ over } \ldots \text{ over } (r_{k(m-1)+1} \text{ over } \ldots \text{ over } r_n).$$

这意味着重采样和图像合成都可以在每段光线内分别进行.如果每个服务线程负责其中一段,整个绘制计算可以并行实现.一旦计算出每段光线的颜色效果,只需通过over算子将它们顺序地合成起来,就得到了对应像素的颜色属性.

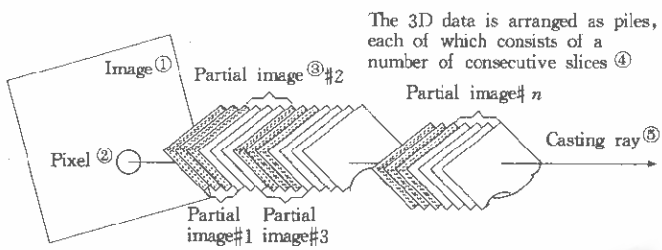
2.3 数据划分及组织

一般地,并行体绘制算法中的子任务划分策略可以分为基于图像划分和基于数据场划分两类.在前一种方法中,每个子任务对应于一个图像子块(image patch).虽然该方法易于编程实现,但是由于每次绘制的实现方向不可能是固定的,任何图像子块都与整个体数据相关,所以,此类算法通常都要在各个计算节点上分别保留一份原始体数据.这样,随着系统中节点数目的增加,无论是存储消耗还是通信负担都会线性增长,因此,系统的可扩展性往往很差.

我们的算法采用了数据划分策略,即将体数据划分并组织成一组二维切片,相邻的若干切片构成一个切片簇(slice pile).从数据角度来看,一个切片簇对应于一个子任务.这种策略直接的优点是,无论节点数目多少,系统的总体存储消耗都与一份原始体数据相当.

只要视线方向不与切片方向垂直,它将被各切片簇分割为若干光线段(ray segment).重要的是,由于从各像素发出的所有光线相互平行,所以无论其被分成的光线段数目,还是各段的长度都是一致的.根据重采样和颜色合成的局部性,位于同一切片簇内的所有光线段只与这些切片相关,因此,其绘制计算可以由一个节点完成.

本地客户进程向远程服务线程的子任务分派,需要通过网络传输必要数据.但是,由于每个子任务只与某一个切片簇相关,整个绘制过程中此方面的通信消耗与原始数据规模相当.此外,在开始局部图像合并之前,各服务线程之间不会有任何通信.



①图像,②像素,③局部图像,④三维数据场划分为切片簇,各自包含若干相邻接的切片,⑤投射光线.

Fig. 1 Data and subtask division
图1 体数据和子任务的划分

2.4 子任务分配及负载平衡

·静态策略/动态策略

静态策略是指在并行绘制开始前就确定各节点的任务,这种策略实现简单.在局域网等特定环境中,由于数据广播代价很低,而且数据通信消耗要远远低于绘制计算本身,只要对各节点计算、存储能力和网络带宽加以分析,就可以相对准确地确定各子任务的时间复杂度,所以,在这些环境中,静态策略是可行的.

分布式和异构网络环境与上述特定环境的情况大不相同[12~14],各节点的计算、存储和通信能力可能差异很大.同一个节点可能同时承担多项计算任务,可利用来参与绘制计算的资源因时而异,事先对某个子任务的计算时间进行准确的估计是不可能的.如果采用静态策略,并行效率将更多地取决于整个系统中性能最差的节点,有时甚至低于串行算法的水平.

·性能指数/自适应算法

为了改善负载平衡,我们采用了动态任务分配策略,其思想是,尽可能使每个节点承担的任务量与其当前计算能力一致.

该策略是以自适应的形式来实现的.每个主机都维护了一个数据库,记录各节点的性能指数,客户进程在分配子任务时,会根据不同节点的性能水平,估计出它们应承担的负载份额.性能指数记录包括两个参数:$\langle P,L\rangle$,其中P表示节点平均性能,L表示统计的次数.在第1次使用时,P被初始化为一个平均性能值,L被初始化为

0. 每次绘制完毕,节点的性能指数记录将根据其实际承担的负载作相应的修正.

假设体数据由 n 个切片构成,共有 m 个节点参与了上次绘制,如果某个节点实际承担的计算量为 x 个切片,那么它对该次绘制的贡献可以估计为

$$P_{latest}=m*x/n.$$

如果该节点原来的性能指数记录为$\langle P_{old},L_{old}\rangle$,那么新的记录将会被修正为

$$P_{new}=(P_{old}*L_{old}+P_{latest})/(L_{old}+1),$$

$$L_{new}=L_{old}+1.$$

• 首次任务分配

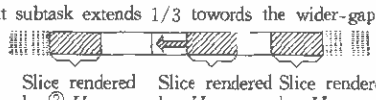
由于在每次绘制计算过程中,每个节点将负责可能相邻的切片,所以,子任务的首次分配就是要根据其性能指数,为各节点确定一个分配切片的起始位置.假设在网络中有$\{H_0,H_1,\ldots,H_{m-1}\}$共 m 个节点,体数据由$\{S_0,S_1,\ldots,S_{n-1}\}$共 n 个切片簇组成,如果各节点的性能指数分别是$\{P_0,P_1,\ldots,P_{m-1}\}$,那么分配给各节点的切片的起始位置$\{O_0,O_1,\ldots,O_{m-1}\}$将被确定为

$$O_k=\begin{cases}0, & k=0,\\ n\left(\sum_{i=0}^{k-1}P_i+P_k/2\right)\Big/\sum_{i=0}^{m-1}P_i, & 1\leqslant k\leqslant m-1,\\ n-1, & k=m-1.\end{cases}$$

在首次子任务分配中,分配给节点 H_k 的是 $n\cdot\left(P_k\Big/\sum_{i=0}^{m-1}P_i\right)\Big/3$ 个相邻切片.通常,这些切片平均分布在各自起始位置的两侧,但是,首次分配给 H_0 的各切片起始于 S_0,而首次分配给 H_{m-1} 的各切片终止于 S_{n-1}.

Fig. 2 First round of subtask assignment
图2 首次任务分配

• 动态任务分配



①下一子任务向宽的一侧伸展1/3,②已经计算过切片.

Fig. 3 Dynamic subtask assignment
图3 动态任务分配

首次任务分配之后,其他切片的分配是以动态竞争方式进行的.每个服务线程完成其上次任务后,客户进程将会追加分配若干新的切片,并保证它们与该线程此前处理的各切片在位置上是相邻的,也就是说,动态任务分配是按照这种递增方式进行的.

如图 3 所示,对一个节点而言,追加分配的切片可能向左或向右增长.在本算法中,该方向是根据其与两侧节点已分配切片的距离来决定的,即向距离更大的方向增长,增长的切片数目为相应距离的 1/3.如果其左、右两侧不再有切片缝隙,该线程将结束绘制计算.

2.5 局部图像的合并

在文献[15]中,Ma 提出了一种二分对换方式的合并算法,但该算法并不适用于分布式环境.首先,该算法强制要求参加图像合并的节点数目为 2 的幂.其次,该算法是以节点间两两兑换数据的方式实现合并的,这要求系统支持节点间的同步双向通信,而在分布式环境中,这一点无法保证.另外,该算法并没有降低整体通信量,由于在分布式环境中,图像合成的时间更多地取决于网络通信的速度,其改进效果并不明显.最后,只有在所有节点都完成绘制计算后,该算法才允许开始合并计算,因此,在各节点计算能力差别较大时,合成计算可能会由于等待最慢的节点而不能及时启动.

本文所介绍的合成算法复杂度为 $O(\log n)$,其计算是异步进行的,任何两个完成绘制计算的相邻节点,可以

立即开始合并计算.

3 测试及分析

我们的测试是在一个由 8 台 PC 主机构成的网络环境中进行的,每个主机的配置为 PIII-450 CPU,64MB 内存,8GB 硬盘,安装的操作系统为 Windows 2000,网络速度为 10MB/s.

测试使用的体数据是通过医用 CT 扫描仪得到的,数据规模为 $128(x)\times128(y)\times197(z)$,每个体素的密度由一个 8 位字节表示.

3.1 任务规模的可扩展性

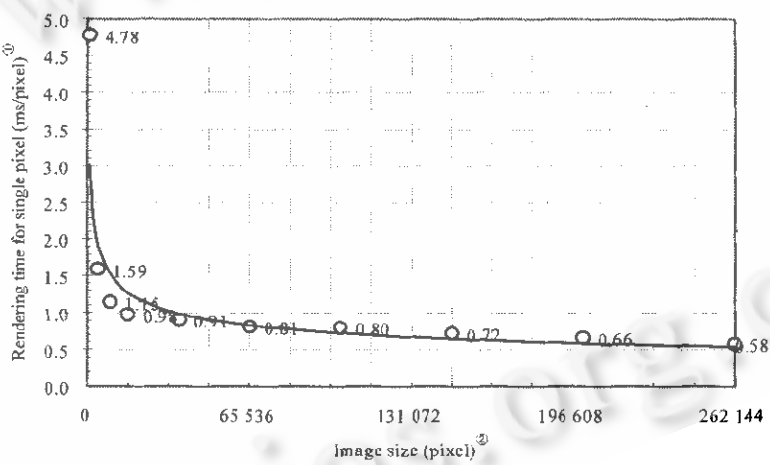
我们以绘制图像的大小来衡量计算任务的规模,并对不同大小的图像所需要的绘制时间进行了统计和比较.这里,绘制时间定义为从用户提交绘制任务到最终图像在用户屏幕上显示出来之间的间隔.为了消除网络涨落的影响,每个数据都是 3 次以上测试的平均值.表 1 给出了实验的统计结果.

Table 1 Rendering time for image of different sizes (8 servers employed)

表 1 绘制不同大小图像的时间消耗(并行虚拟机由 8 个节点构成)

Image size (pixel)①	32^2	64^2	96^2	128^2	192^2	256^2	320^2	384^2	448^2	512^2
Rendering times (s)②	4.9	6.5	10.6	16.0	33.5	53.1	81.7	106.0	132.5	151.4

①图像大小(像素),②绘制时间(秒).



①像素平均绘制时间(毫秒/像素),②图像大小(像素).

Fig. 4 Scalability of the task size (8 servers employed)

图 4 任务规模的可扩展性(并行虚拟机由 8 个节点构成)

我们以单个像素的平均绘制时间来衡量算法对任务规模的可扩展性.由图 4 可以看出,在绘制大型图像时,这项指标是稳定的.需要指出的是,随着图像面积的缩小,虽然总体的绘制时间和传输时间相应地减少,但是数据压缩以及解压缩的时间不会有明显变化,所以,单个像素的平均绘制时间会出现逆向增长的情况.

3.2 虚拟机规模的可扩展性

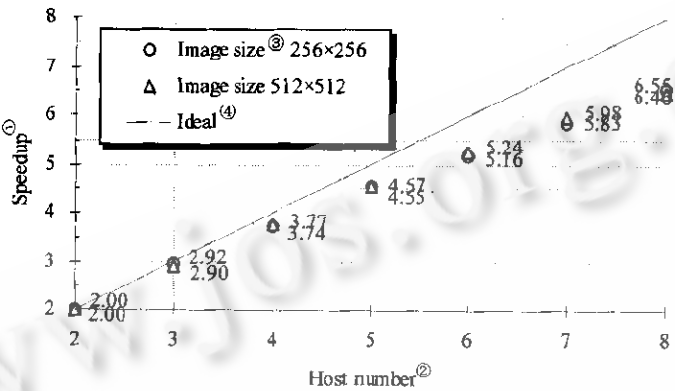
我们以其中包含节点的数目来衡量虚拟机的规模,并对不同规模的虚拟机对相同图像的绘制时间进行了统计和比较.这里,绘制时间定义为从用户提交绘制任务到最终图像在用户屏幕上显示出来之间的间隔.为了消除网络涨落的影响,每个数据都是 3 次以上测试的平均值.表 2 给出了实验的统计结果.

我们以包含两个节点的虚拟机为基准,用加速比来衡量算法对虚拟机规模的可扩展性.由图 5 可以看出,该项指标是稳定的.

Table 2 Rendering time on virtual machines of different sizes
表 2 不同规模虚拟机分别绘制两幅图像的时间

# hosts①	2	3	4	5	6	7	8
Rendering time on 256×256 image②(s)	174.1	119.1	93.1	76.2	67.5	59.8	53.1
Rendering time on 512×512 image③(s)	488.9	337.0	259.4	214.9	186.6	163.6	151.4

①主机数目,②绘制 256×256 图像所需时间,③绘制 512×512 图像所需时间.



①加速比,②节点数目,③图像大小,④理想.

Fig. 5 Scalability of the number of hosts
图 5 虚拟机规模的可扩展性

4 结 论

本文着重讨论了在虚拟机环境中的并行体绘制问题,介绍并实现了一种算法.该算法基于切片来组织体数据,保证了子任务的数据局部性;采用自适应的启发式任务调度策略,实现了负载平衡;采用异步二分算法,在$O(\log n)$时间内完成对 n 幅局部图像的并行合并.本文所提出的算法已在我们自行实现的平台上开发完成.该平台将为此方面的进一步研究提供帮助.

致谢 本文列举的实验数据是在清华大学计算机科学与技术系系统结构实验室中获得的.在此,作者谨向郑纬民教授及其实验室的全体人员表示感谢.

参考文献

1 Jacq J J, Roux C J. A direct multi-volume rendering method aiming at comparisons of 3-D images and models. IEEE Transactions on Information Technology in Biomedicine, 1997,1(1):30~43

2 Anderson T E, Culler D E, Patterson D A. A case for NOW (networks of workstation). IEEE Micro, 1995,15(1):54~64

3 Cap C H, Strumpen V. Efficient parallel computing in distributed workstation environments. Parallel Computing, 1993,19(9):1221~1234

4 Giertsen C, Peterson J. Parallel volume rendering on a network of workstations. IEEE Computer Graphics and Applications, 1993,13(6):16~23

5 Deng Jun-hui, Tang Ze-sheng. Parallel frequency domain volume rendering on workstation cluster. Chinese Journal of Advanced Software Research, 1997,4(4):331~342

6 Singh J P, Gupta A, Levoy M. Parallel visualization algorithms: performance and architectural implications. IEEE Computer, 1994,27(7):45~55

7 Sen V, Sen M K, Stoffa P L. PVM based 3-D kirchhoff depth migration using dynamically computed travel-times: an

application in seismic data processing. Parallel Computing, 1999,25(3):231～248

8 Elenbogen B S, Maxim B R, Tsui L *et al*. Parallel and distributed algorithms laboratory assignments in Joyce/Linda. Engineering Science and Education Journal, 1999,8(2):81～88

9 Wyatt B B, Kavi K. Hufnagel S. Parallelism in object-oriented languages: a survey. IEEE Software, 1992,9(6):56～66

10 Zalewski J. MPI: the complete reference book review. IEEE Concurrency, 1997,5(1):80～81

11 Wang Cho-li, Bhat P B, Prasanna V K. High-Performance computing for vision. Proceedings of the IEEE, 1996,84(7):931～946

12 Aversa R, Mazzeo A, Mazzocca N *et al*. Heterogeneous system performance prediction and analysis using PS. IEEE Concurrency, 1998,6(3):20～29

13 Kafil M, Ahmad I. Optimal task assignment in heterogeneous distributed computing systems. IEEE Concurrency, 1998,6(3):42～50

14 Balsamo S, Donatiello L, Van Dijk N M. Bound performance models of heterogeneous parallel processing systems. IEEE Transactions on Parallel and Distributed Systems, 1998,9(10):1041～1056

15 Ma Kwan-liu, Painter J S, Hansen C D *et al*. Parallel volume rendering using binary-swap compositing. IEEE Computer Graphics and Applications, 1994,14(4):59～68

Parallel Volume Rendering on a Virtual Machine

DENG Jun-hui TANG Ze-sheng

(*Software Research Institute Department of Computer Science and Technology Tsinghua University Beijing* 100084)

Abstract An algorithm for volume rendering in an environment of parallel virtual machines is presented in this paper. In order to reduce the communication cost, as well as to guarantee the locality of all subtasks, the volume data are divided and organized as a series of slices. By maintaining and employing a database of the performance index, the task subdivision algorithm produces an acceptable load balancing. An asynchronous binary method is introduced, which merges all partial images in $O(\log n)$ time. An efficient developing platform based on the TCP/IP and Socket standards is built. It helps researchers to parallelize various rendering algorithms on a virtual machine. The algorithm introduced in this paper has been implemented on this platform, exploiting the classical client/server paradigm. The scalabilities of both the task size and the number of hosts are tested. The experimental results are demonstrated and analyzed.

Key words Volume rendering, parallel virtual machine, load balancing, scalability.