CPU+GPU 海量信息集群高速显示技术①

罗明宇, 刘其军, 付燕平

(广东粤铁瀚阳科技有限公司, 广州 510630)

摘 要: 针对集群显示系统中存在的 CPU 多核闲置、GPU 利用不足、CPU 与 GPU 结合困难等问题, 研究了 CPU 多核多线程处理、GPU 并行处理及 CPU+GPU 整合运算等技术, 提出并构建了 CPU+GPU 集群并行显示系统, 提升了集群并行显示系统的综合运算能力, 实验结果表明 CPU+GPU 集群并行显示技术是有效的, 为海量信息高速显示提供了有效的解决方案.

关键词: CPU+GPU; 海量; 集群; 并行

CPU + GPU Technologies for Massive Information Display on Cluster

LUO Ming-Yu, LIU Qi-Jun, FU Yan-Ping

(Guangdong Railway & Sun Technology Co. Ltd, Guangzhou 510630, China)

Abstract: There is much waste of computing resources in cluster, such as setting aside multi-core CPU, underutilization of GPU, separating the use of CPU and GPU. To use CPU and GPU better, some technologies, such as multicore CPU processing, GPU parallel processing and CPU + GPU integrated processing, are researched in this paper. A CPU + GPU cluster parallel display system is presented to improve the parallel display capability of cluster tiled system. The experimental results show that the CPU + GPU cluster parallel display system presented is feasible and can greatly improve the display speed of massvie information.

Key words: CPU+GPU; massive information; cluster; parallel

1 引言

随着通讯、互联网等信息技术的迅猛发展, 承载信息的载体越来越先进, 而且载体承载的信息量也越来越多, 社会进入了一个信息海量化的时代. 各类系统中的海量信息数量远远超越了现有 IT 架构和基础设施的承载能力, 信息显示的实时性要求也大大超越现有信息显示设备的处理能力. 如何高速显示海量数据信息, 使其为社会管理、指挥决策乃至生产生活服务, 是信息显示系统必然的升级方向.

于是, 计算机集群并行显示系统应运而生, 作为当今最先进的超高分辨信息集中显示系统, 集群并行显示系统正逐步成为了铁路调度、交通管理、电力调度、公安指挥等信息可视化、决策指挥不可或缺的核心基础显示平台. 它以计算机集群为基础提供强大的计算资源, 以高速计算机网络为信息通道, 采用并行分布式模块化结构构建显示系统, 造价低廉、易于构筑, 对海量信息处理显示具有极大优势[1,2].

高性能计算集群由一台主节点计算机将运算显示任务分配到集群内多台子节点计算机进行并行运算处理, 完成超高分辨信息拼接显示. 因此, 必须挖掘并充分发挥各子节点计算机的运算处理能力, 才能够解决好海量数据信息的高速显示处理问题.

近年来, CPU 和 GPU 混合计算开始逐渐成为国内外高性能计算领域的热点研究方向, 为计算机系统提供了强大的显示运算能力[3,4], 其中, CPU 适合用于复杂的逻辑控制运算, 而 GPU 更适用于图形类或非图形类高并行的数值计算. 鉴于 GPU 在通用计算领域的优异表现, Macedonia[5]等断言 CPU 和 GPU 混合计算将成

① 基金项目:广东省科技计划(2012A080102003);广东省省部产学研结合项目(2012B090500012)
收稿时间:2014-07-16;收到修改稿时间:2014-08-18

© 中国科学院软件研究所 http://www.c-s-a.org.cn

为未来计算的主流. 在集群子节点计算机的显示运算过程中, 由于各种历史和现实原因的制约, CPU 和 GPU 混合计算仍然面临着诸多问题, 其中最突出的问题是由于程序开发困难, 导致多核 CPU 闲置、GPU 利用不足等计算资源浪费的现象比比皆是, 为此, 本文研究了 CPU 多核多线程处理、GPU 并行处理及 CPU+GPU 整合运算等技术, 最大限度地提升了集群并行显示系统的运算处理能力. 实验证明, CPU+GPU 的整合运算能够较好地完成卫星影像、地理信息等海量数据的高速显示, 为海量信息显示提供了有效的解决方案.

2 CPU多核处理

CPU 在设计之初是单核的, 执行的是串行代码, 其优化计算也多通过复杂的控制逻辑、线程调度、快速缓存等[6,7]提升运算执行的效率, 以加速完成指令操作. 为了充分发挥 CPU 的计算能力, CPU 处理多采用了多线程并行的方式, 把计算任务分成多个子任务, 每个子任务采用独立线程运行, 如图 1 所示:

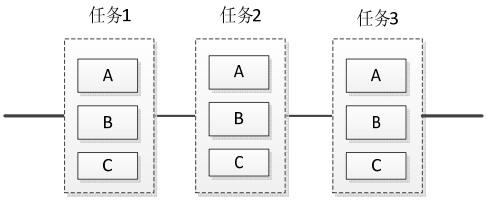

图 1 CPU 多线程计算

CPU 多线程计算在同一时间只有一个任务处于执行状态, 任务之间是串行的. 虽然操作系统采用了多任务并行的方式, 充分开发了 CPU 的计算能力, 但单核 CPU 的多线程处理并非真正意义上的同时并行计算, 而是通过分配 CPU 时间片切换调度多线程实现并行计算的, 仅靠多线程技术提升单核 CPU 的处理能力是远远不够的.

为提升 CPU 的处理能力, CPU 进入了多核时代, 单个 CPU 拥有了多个处理核心, 但由于历史原因, 多核并行计算的程序开发并未完全普及, 现有的显示软件系统在计算时仍大都以单核 CPU 为处理核心, 并未充分发挥出多核CPU的计算处理能力, 造成了CPU多核的闲置浪费.

多核多线程并行处理采用的是多核并行运算处理, CPU 的每个核心都可以采用多线程并发的方式进行计算: 通过统一调度合理分解计算任务, 把任务分成多个可并行计算的子任务, 每个子任务分别运行于 CPU 的不同核心上, 每个核心可采用单核多线程的方式高效运行, 如图 2 所示:

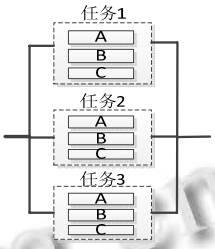

图 2 多核多线程计算

线程中启动多核 CPU 的程序如下:

```
cpu_set_tcpuset;
CPU_ZERO(&cpuset);
intcpuCoreNum=sysconf(_SC_NPROCESSORS_CONF);
for (int j = 0; j <cpuCoreNum; j++)
    CPU_SET(j, &cpuset);
sched_setaffinity(0,sizeof(cpuset),&cpuset);
```

通过上述多核 CPU 的并行、单核多线程并发的计算处理, 可调动 CPU 多个核心的计算能力, 充分发挥多核 CPU 复杂的控制逻辑计算优势, 大大提高计算机系统的信息并行处理性能.

3 GPU并行处理

为实现专业图形显示的快速处理, GPU 应运而生, 随着通用计算需求的发展和各 GPU 生产厂商的大力推动, 限制 GPU 通用计算的障碍(如硬件结构、编程模型)在很大程度上得到了克服[8,9], 而 GPU 把大部分的芯片面积用在了执行单元上, 而非传统的数据缓存和调度控制上, 使得 GPU 可同时调度大量相同的工作, 所有子任务的指令完全一样, 不需要像 CPU 那样在每线程都用一套控制器, 而是用一个控制器同时调度大量的线程, 且没有线程间切换的开销, 实现了真正的并行处理, 提高了对并行计算的支持, 由此获得了大大超越 CPU 的并行运算能力、高存储带宽、低功耗以及较好的可编程性, 使 GPU 得以广泛应用于通用计算, 通过并行处理提升计算速度.

GPU 的卓越计算能力体现在并行计算方面, 在 GPU 并行计算中, 通常分为任务并行(把一个任务分

© 中国科学院软件研究所 http://www.c-s-a.org.cn

解为能够同时执行的多个子任务)和数据并行(同一个任务内, 它的各个部分同时执行)两类. GPU 采用统一的渲染架构和网络线程结构, 如图 3 所示, 可以大量的任务并行与数据并行的计算处理, 而且 GPU 的线程切换开销比 CPU 的小, 在线程并行处理上有更好的性能.

Thread(0,0)	Thread(0,1)	Thread(0,2)	Thread(0,3)
Thread(1,0)	Thread(1,1)	Thread(1,2)	Thread(1,3)
Thread(2,0)	Thread(2,1)	Thread(2,2)	Thread(2,3)
Thread(3,0)	Thread(3,1)	Thread(3,2)	Thread(3,3)

图 3　GPU 线程结构

为降低直接 GPU 编程的复杂度, nVidia 使用了 CUDA (Compute Unified Device Architecture)接口支持 GPU 的并行计算处理, 它封装了 GPU 并行处理的细节并提供了外部开发接口, 采用统一处理架构和扩展 C 语言的方式, 使 GPU 并行开发相对简单[10-11].

GPU 执行并行计算的基本单位是线程, 线程通常组织成 Grid(线程网格)的形式, 一个 Grid 包含多个相互合作的 Thread Block(线程块), 而每个 Thread Block 可由一系列 Thread 组成. 根据 GPU 并行处理的数据大小分配 Grid 和 Block 的维度. 如: 对于任务共有 256 个子任务的 GPU 并行处理, 可以给每个 Grid 分配 16 个 Block, 每个 Block 分配 16 个线程, 相关启动 GPU 并行处理的程序如下:

```
void   launch_kernel()
{
  dim3   blockPerGrid(4,4);//每个 grid 中有 4×4 个 block
  dim3   threadsPerBlock(4,4); //每个 block 中有 4×4 个线程
  gpu_kernel<<< grid, block>>>();
    // gpu_kernel 是需在线程中并行完成的子任务
}
```

在 nVidia 的 Kepler(开普勒)架构显卡中, 一个 Grid 最大支持 2^{32}-1 个 Block, 每个 Block 可支持 2048 个线程, 这样, GPU 可以支持(2^{32}-1)×2048 个并行线程, 因此, 充分发挥 GPU 的并行处理能力, 对于有效提升海量信息的处理速度作用相当显著.

充分发挥 GPU 的性能, 必须将任务分解成大量相同的子工作, 控制逻辑复杂的工作并不是 GPU 的强项, 而实际计算任务并不只是大量简单的相同计算, 可能存在复杂的控制逻辑关系, 因此, 需要在 CPU 的控制调度下完成计算任务的分解处理.

4　CPU+GPU集群并行显示

海量信息集群并行显示就是要把复杂的海量信息显示处理任务分解为多个能同时并行处理的子任务. 为实现海量信息的高速并行显示、充分发挥 CPU 和 GPU 的运算能力, 构建的 CPU+GPU 集群并行显示系统由一台主节点机与多台子节点机组成, 如图 4 所示:

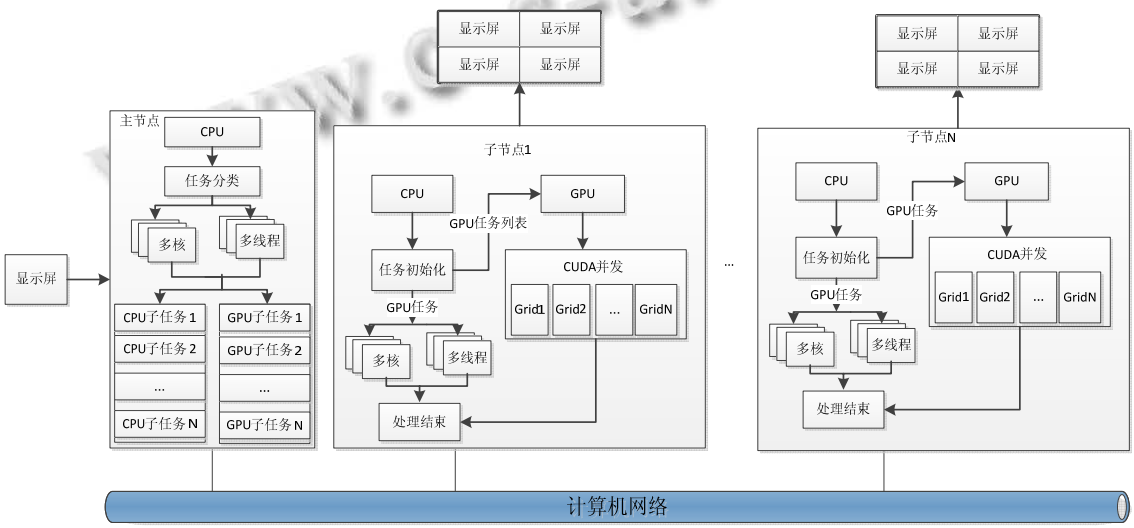

图 4　CPU+GPU 集群并行显示系统

© 中国科学院软件研究所 http://www.c-s-a.org.cn

主、子节点机间通过高速网络传递通信消息，每台子节点计算机都连接一块或多块显示屏，主节点负责调度管理各子节点机的 CPU 和 GPU 计算资源，子节点机通过 CPU 给 GPU 提供计算数据、串行逻辑和任务调度，由 GPU 进行并行计算处理，最后子节点机完成拼接显示任务. 为了充分发挥 CPU 与 GPU 的计算性能，在 CPU 端，采用多核并行与核内多线程结合的技术，并行完成数据读取、指令处理等控制逻辑任务，提高 CPU 的使用效率；在 GPU 端，将信息解码、渲染显示等任务分解为相同的并行模块，利用 CUDA 框架实现 GPU 处理能力的最大化.

在 CPU+GPU 集群并行显示计算过程中，CPU 端为 Host，用“__host__”进行标识，GPU 端为 Device，用“__device__”进行标识. 在 GPU 上并行执行的程序为 Kernel 内核程序，在 CPU 上执行的为 Host 宿主程序. 由宿主程序控制与 GPU 的数据通信，负责串行部分的执行，控制 Kernel 的启动、加载或保存，Device 在执行 Kernel 时创建很多的 Thread. 如图 5 所示，完整的 CPU+GPU 并行显示处理程序是由一系列 GPU 上执行的 Kernel 函数并行和 CPU 上执行的串、并行计算共同组成.

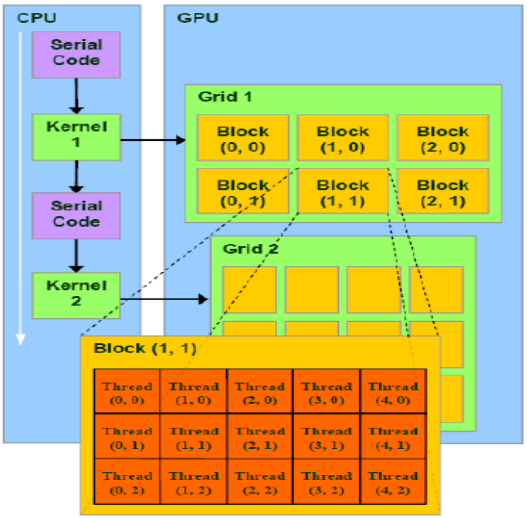

图 5 CPU+GPU 并行处理

对于海量信息显示处理任务，先划分为若干 CPU 子任务(由 CPU 处理)和 GPU 子任务(由 GPU 处理). 在主节点完成任务初始化后，根据任务的总数和子节点数量，把任务按子节点进行平均分配，同时记录下每个子节点需要处理的 CPU 和 GPU 子任务.

子节点收到主节点分配的任务后，首先对任务进行初始化和分类，然后采用 CPU+GPU 并行处理，其中 CPU 的处理是把 CPU 子任务按 CPU 核心的数量进行分组执行，同时在每个 CPU 核内使用多线程并发执行；GPU 的处理，是在 CPU 数据准备及初始化上的基础实现 GPU 并行处理. 具体根据 GPU 任务的数量，分配 Grid 上 Block 和 Thread 的数量，把每个 GPU 子任务交给 Grid 的线程执行处理，同时通过 Host 宿主程序返回任务处理结果.

当所有的 GPU 子任务都处理完成后，系统通过网络通知主节点，主节点统计所有子节点的任务完成返回信息后，进入集群显示状态，把需进行显示的信息通过网络通知所有子节点. 子节点收到需显示的信息分块后，把每个分块的显示交由 GPU 的 Device 多线程并行处理，由 OpenGL 完成海量信息的集群并行渲染显示.

通过在集群并行显示系统的各节点机上应用 CPU+GPU 混合计算处理的方式，既发挥了多核 CPU 在管理调度、事务逻辑等方面强大的综合计算处理能力，又充分利用了 GPU 高速并行计算完成信息渲染显示任务的能力，使集群并行显示系统可完成高负载、高吞吐的海量信息高速显示任务.

5 实验测试

为验证 CPU+GPU 集群并行显示系统的显示处理能力，搭建了由一台主节点和五台子节点组成的集群显示系统，其中每台子节点的配置为 4 核英特尔 Xeon 3.20GHzCPU，GPU 为 nVidia GeForce GTX 465，内存为 6 GB.

针对多路 1080P 视频和海量卫星影像文件的显示实验结果如表 1.

表 1 实验结果对照表

显示信息	单核CPU计算显示处理时间（s）	CPU+GPU计算显示处理时间（s）
5路1080P视频	≥15	≤1
12路1080P视频	难以显示	≤2
1G卫星影像	≥30	≤1
5G卫星影像	≥300	≤2
16G卫星影像	难以显示	≤5

海量信息处理显示过程中，由于传统的单核 CPU 计算处理能力有限，而数据交换量大，不可避免地导致显示窗体长时间无信息输出，而 CPU+GPU 集群并

© 中国科学院软件研究所 http://www.c-s-a.org.cn

行显示系统充分利用了多核 CPU 的综合运算处理能力和 GPU 高速并行渲染显示能力, 快速读取并完成了相关海量信息的显示任务.

不难看出, CPU+GPU 集群并行显示系统充分利用了 CPU 与 GPU 的计算资源, 解决了传统计算机 CPU、GPU 资源闲置的问题, 为有效解决海量信息的并行显示提供了有效的解决方案.

6 结语

随着高处理能力的芯片的不断推出、计算机集群技术的不断突破, 集群并行显示系统正迅速进入铁路调度、交通管理、军事指挥等领域的集中信息显示系统, 成为指挥决策显示系统的核心重要组成部分.

本文针对集群显示系统中存在的 CPU 多核闲置、GPU 利用不足、CPU 与 GPU 结合困难等问题, 研究了 CPU 多核多线程处理、GPU 并行处理及 CPU+GPU 整合运算等技术, 提出的 CPU+GPU 集群并行显示系统提升了集群并行显示系统的综合运算能力, 实验结果证明 CPU+GPU 集群并行显示系统是有效的, 为海量信息高速显示提供了有效的解决方案.

参考文献

1 孙晓娟.基于网络的大屏幕显示系统的设计与实现.东北大学,2010.

2 王海涛,刘淑芬.基于 Linux 集群的并行计算.计算机工程与设计,2010,36(1).

3 卢风顺,宋君强,银福康,张理论.CPU/GPU 协同并行计算研究综述.计算机科学,2011,(3).

4 王伟,郭绍忠,王磊,冯颖.一种基于 CPU-GPU 异构计算的混合编程模型.信息工程大学学报,2010,(6).

5 Macedonia M. The GPU enters computing's mainstream, IEEE Computer, 2003,36(10):106-108.

6 冯颖,袁庆华,沈健炜.基于 CPU+GPU 异构计算的编程方法研究.通信技术,2011,(2).

7 王维平,余文广,侯洪涛,李群.多核 CPU-GPU 异构平台下并行 Agent 仿真负载均衡方法.系统工程与电子技术,2012,(11).

8 Moreland K, Angel E. The FFT on GPU. Proc. of Graphics Hardware. San Diego. 2003, 117(17): 112.

9 Sacerdoti FD, Chandra S, Bhatia K. Grid systems deployment & management using rocks. IEEE Int. Conf. on Cluster Computing, San Diego. September 2004.

10 Nickolls J, Buck I, Garland M, et al. Scalable parallel programming with CUDA. ACM Queue, 2008, 6(2): 40–53.

11 Nukada A, Ogata Y, Endo T, et al. Bandwidth intensive 3D FFT kernel for GPUs using CUDA. Proc. of the 2008 ACM/IEEE Conf. on Super Computing. IEEE Press, 2008.

© 中国科学院软件研究所 http://www.c-s-a.org.cn