文章编号: 1001- 2486(2009) 05- 0038- 06

GPU 异构系统中的存储层次和负载均衡策略研究*

马安国, 成 玉, 唐遇星, 邢座程
(国防科技大学 计算机学院, 湖南 长沙 410073)

摘 要: GPU 体系结构的革新和相应开发平台的发展使得 GPU 广泛地应用于科学计算领域。通过深入地分析 GPU 体系结构和存储层次的优缺点以及 GPU 上的关键性能特征, 阐明了 GPU 体系结构、编程模型和存储层次之间的关系。针对 GPU 异构系统上的应用映射提出三种基本负载均衡优化策略: 预取、流化、任务划分。试验结果揭示了不同的优化因子与优化效率之间的具体关联。

关键词: GPGPU; 存储层次; 负载均衡策略; 流计算; 任务划分
中图分类号: TP302. 7 文献标识码: A

Research on Memory Hierarchy and Load Balance Strategy in Heterogeneous System Based on GPU

MA An-guo, CHENG Yu, TANG Yu-xing, XING Zuo-cheng
(College of Computer, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: Owing to the revolution of GPU architecture and improvement of developing platforms, GPU is widely used in scientific computing nowadays. Relationships among GPU architecture, programming model and memory hierarchy are illustrated by analyzing memory hierarchy and exploring key performance features of GPU. Three basic load balance strategies on mapping applications onto GPU are presented: Prefetch, stream computing, task division. The effective relationships among different factors and optimization efficiency are tested and exposed by experiments.

Key words: GPGPU; memory hierarchy; load balance strategy; stream computing; task division

当前高性能计算的关键是如何充分挖掘强大的多核计算部件的峰值性能。虽然现已有针对底层硬件体系结构构建的软件平台, 并行计算硬件平台和相应软件环境之间的差距仍然越来越大[1]。面对当前困境, 除创建全新的体系结构和高效的并行编程模型外, 相关商业公司和学校研究人员还有两种解决方式, 即基于当前的硬件平台, 改进通用处理器和开发环境使之在各应用领域更强大, 或者使定制处理器更通用。如今已实现一些新的体系结构和相应编程模型, 例如 Imagine, Cell, RAW, Many-core 计划, Fusion 计划等等。同时新的多核编程开发环境快速发展以面对更加强大的多核处理硬件平台, 如 CUDA[2]、SWARM[3]、RapindMind[4]。

GPU 是定制处理器转向更通用领域的一个成功典范。GPU 的优势之一在于强大的计算能力, 其片上用于计算的晶体管数目占整片上晶体管总量的 80% 左右; 其次, GPU 采用了灵活的存储层次设计, 有利于不同层次上的负载平衡; 最后, GPU 的两级编程编译模型能够更好地暴露应用中蕴含的并行度和挖掘 GPU 强大的计算能力。

本文基于三种优化策略: 预取、流化以及任务划分, 在不同的硬件平台上优化矩阵乘。优化的核心思想是改进计算访存比以及各存储层次上的延迟隐藏。任务划分能够充分利用系统的计算资源, 使 CPU 和 GPU 并行执行。我们总结了划分因子和其它性能影响因素之间的关系, 其中包括 CPU 和 GPU 的计算能力以及它们之间的实际带宽。同时, 还通过将相关数据传输操作和计算操作打包成流, 在系统

* 收稿日期: 2009- 07- 03
基金项目: 国家自然科学基金资助项目(60873016); 国家 863 计划资助项目(2009AA01Z102); 教育部"高性能微处理器技术"创新团队资助项目(IRT0614)
作者简介: 马安国(1983—), 男, 博士生。

级隐藏传输延迟。实验中使用的平台特征如表 1 所示。平台 1 中的 GPU 属于 G80 体系结构, 平台 2 中的 GPU 属于 GT200 体系结构。它们对浮点计算的支持能力和峰值性能都不同。

表 1 测试平台

Tab. 1 Experiment platform

	平台 1	平台 2
CPU	Intel Core2 E7200 (2.533GHz)	Intel Core i7 (2.67GHz)
GPU	NVIDIA GeForce 9800 GTX(1.84 GHz+ 512MB 显存)	NVIDIA GTX280(1.3GHz+ 1GB 显存)
主板	GIGABYTE EDP43- DS3L(兼容 PCIe 2.0 × 16)	GIGABYTE EX58- UD4(兼容 PCIe 2.0 × 16)

1 GPU 体系结构和 GPGPU 研究

统一架构以前的传统 GPU 体系结构采用固定功能部件, 以有效地处理图形应用操作。不同的计算单元只能负责执行特定操作, 如顶点着色器只处理顶点 T&L 操作、光栅单元仅负责处理光栅化操作等, 从而面对不同特征的图形应用, 计算资源不能充分利用。当前 GPU 采用了统一架构设计, 利用不同的指令分派单元, 大量单一的标量计算单元能够处理顶点、像素以及几何操作。2005 年初, AMD 在 Xenos 中首先实现了统一处理体系结构, 并于 2006 年宣布了面向高性能计算的流处理器和相应的 CTM(Close To Metal)运行时。同时, NVIDIA 也于 2006 年 11 月引入了 CUDA(Compute Unified Device Architecture), 其中包括 G80 硬件核心和 CUDA 运行时。CTM 和 CUDA 均是通过动态调度海量线程以有效利用计算资源。

相比于 CPU, GPU 强调吞吐率而不是单线程的延时。同时, GPU 的体系结构设计是针对于计算密度高的数据并行应用。NVIDIA 在 G80 体系结构的基础上扩展了计算单元、寄存器文件、纹理单元以及全局存储器, 提出了支持双精度浮点计算的 GT200 体系结构。它由 10 个线程处理群(Thread Processing Clusters, TPC)组成, 其中每个 TPC 包括 3 个流多处理器(Streaming Multiprocessor, SM)。每个 SM 则包括 8 个流处理器(Streaming Processor, SP)、一个 64 位混合乘加单元和一个特殊功能单元(Special Function Unit, SFU), 它们共享同一个前端。

GPGPU(General Purpose Computing on GPU)就是将传统图形领域之外的应用映射到 GPU 上。目前大量成功的例子可以证明 GPU 可以被广泛并且有效地应用于传统图形领域之外的很多科学计算和工程应用领域, 如生物学[5-6]、物理仿真学[7-9]、生物物理学[10-12]以及信号图像处理等[13-14]。GPU 尤其在高性能计算领域已取得很好的成就并展示了广阔的应用空间。Volkov[15] 总结了当前 GPU 的一些关键特性以及代码优化方面有用的经验, 深入分析了 GPU 存储系统中 Cache 和 TLB 的特征和延迟, 并考虑了异构系统中 CPU 和 GPU 的并行执行。他们优化了矩阵乘算法, 相对于官方的库 CUBLAS1.1, 性能提升了 60%, 并取得了 GEMM 峰值性能的 80%~ 90%。Massimiliano Fatica[16] 分析了 Host 和 Device 之间的带宽, 通过任务划分使任务在 CPU 和 GPU 上并行执行, 并提取了理想的划分因子。Castillo[17] 等封装了 CUBLAS, 将 PLAPACK 编程接口——libFlame 移植到多 GPU 系统上, 并取得了一定性能加速。Quintana-Orti[18] 等通过将 FLAME 移植到拥有多种加速部件的异构系统上, 有效地验证了 FLAME 方法学。

2 GPU 统一架构中的存储层次

面向不同应用中不同类型的数据结构和访存模式, GT200 体系结构中包括了一个复杂的存储层次和相应的纹理输出流水线, 从而可以重用包括临时计算结果的片上数据, 充分挖掘计算 Kernel 间的生产者—消费者局部性, 并隐藏数据传输延时。GT200 使用了 1GB 的片外全局存储器, 通过 PCIe 总线与 CPU 进行数据传输。在 GPU 上执行的计算核能够直接访问全局存储器, 或者通过两级纹理 Cache 和常量 Cache 访问只读的纹理存储器和常量存储器。如图 1 所示, 每个 SM 有 16KB 的片上共享存储器和 16KB 的 32 位寄存器。共享存储器被组织为 16 个体, 并被动态划分给线程块。寄存器被动态地划分给 SM 上的活跃线程块, 然后静态地分配给给定线程。在没有体冲突的情况下, 访问共享存储器和访问寄

存器一样快。而访问全局存储器则需要几百个时钟周期。如果分配给线程的寄存器不够用,数据将会溢出到位于全局存储器中的私有局部存储器中。

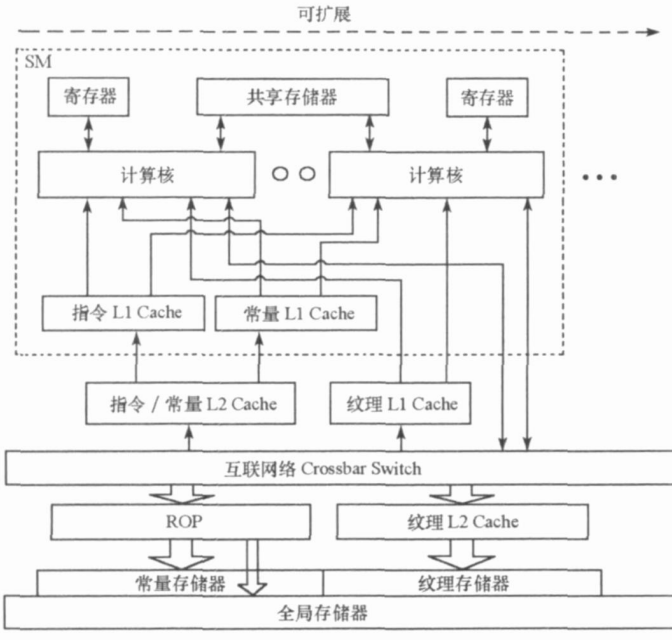

图 1 NVIDIA GPU 存储层次

Fig. 1 NVIDIA GPU memory hierarchy

多级存储层次和对访存模式多样性的支持,使得开发人员能够有效地挖掘应用的并行性以及单个 Kernel 和 Kernel 之间的时间、空间局部性。通过有效地利用片上存储层次,能够达到各级上的负载平衡,从而避免全局存储器和主处理器(Host)上内存之间的带宽限制。

3 GPU 存储层次间的负载均衡策略

应用映射及优化的基本准则就是尽可能地在应用执行期间充分利用计算单元。通过优化典型应用矩阵乘算法,我们提出了基于三个层次上的负载均衡优化策略。首先,提高 Kernel 的计算访存比,增加数据重用度以减少片外访存带来的延时。通过比较得到适合的预取跨度因子,并参照汇编代码,指导手动调度指令。其次,在 Kernel 划分层次上,将相关传输和计算操作打包成多个流,从而重叠一些 Kernel 执行和数据传输,并分析具体流划分和性能之间的关系。再次,从系统级任务划分角度,划分应用任务,不同任务在系统中不同计算资源上并行执行,并提取最佳的划分因子。最后,基于以上工作,融合三种负载均衡策略对算法进行优化。

3.1 预取

预取是一种常用的隐藏访存延时策略。GPU 中的全局存储器访存会耗时几百个时钟周期,严重影响应用性能。因此采用预取策略优化矩阵乘算法。在保证程序正确性的前提下,将一些全局访存指令提前,放置到数据相关指令之间,重叠计算和访存。不过相对于 CPU,GPU 的特殊复杂性在于,活跃线程 warp 在 SM 上周期性调度执行。因此,不同硬件平台上不同的应用可能预取跨度因子不同,需要比较不同跨度的配置以取得最佳效果。

在平台 1 上进行测试,配置矩阵维度都为 2048 × 2048,当预取跨度因子取 2 时,性能提升了 8.5%。然而当取 4 或 8 时,应用性能仅达到原性能的 38%。

3.2 流计算

CUDA 运行时中的很多操作都是异步的,如异步拷贝操作 cudaMemcpyAsync、Kernel 发射操作、事件记录操作 cudaEventRecord 等等,这些操作发射之后控制返回给 Host。因此可以利用系统级并行和流级

并行, 将任务划分为多个 Kernel, 然后与其相关的数据传输操作打包成多个流。在没有资源冲突条件下, 不同流中的异步操作可以并行执行。

传统的 GPU 计算可以描述为: 传输数据和代码到设备中, Kernel 在设备上计算, 然后将结果传回 Host。同时, 由于应用中的数据并行度和 GPU 的 SPMD 执行模式, 应用中的计算可以划分为多个计算 Kernel, 然后部分地并行执行, 如图 2 所示。

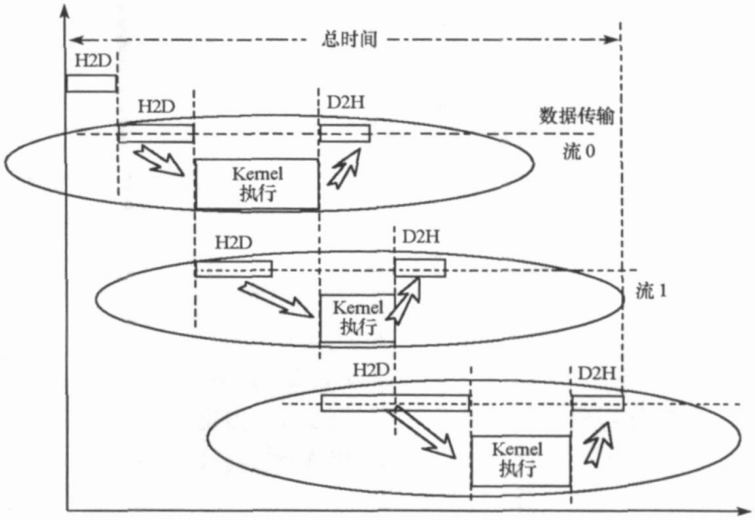

图 2 GPU 上流计算

Fig. 2 Stream computing on GPU

流化应用会增加操作数量, 从而带来相应消耗。此外, 划分后的数据集如果偏小, 则不能充分利用 CPU 和 GPU 之间的带宽。同时, 由于流化策略针对隐藏传输延时, 如果计算访存比过高或过低, 加速可能不会显著, 甚至结果会更差。

如图 3 所示, 在平台 1 上对原始程序进行流化。相对于未流化的程序, 当划分为 3 个流时, 稳定情况下的加速比为 4%; 划分为 6 个流时, 加速比达到了 5%。但是当矩阵维度相对较小时, 流化应用的性能优势不甚明显, 甚至更差。这也印证了上面的分析。

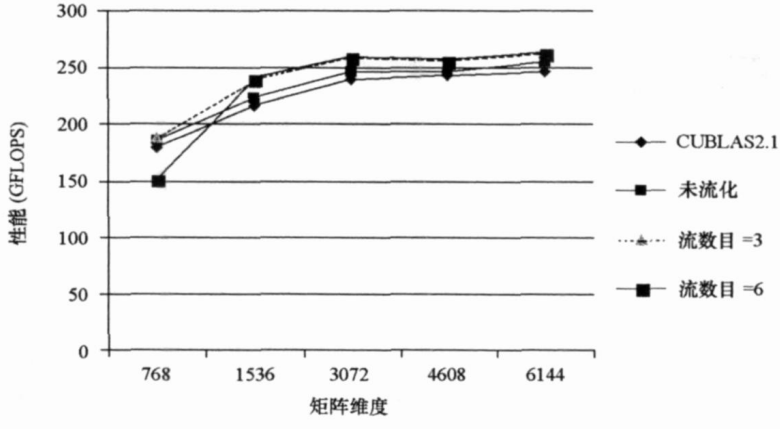

图 3 平台 1 上流化应用性能对比

Fig. 3 Stream computing performance on platform1

3.3 任务划分

Kernel 发射之后控制权返回给 Host, 因此 Host 和 Device 可以并行执行任务。因此将应用划分为两个工作集, 如图 4 所示。

划分因子依赖于 Device 和 Host 的计算能力以及两者之间的数据传输带宽。CPU 上任务可以和 GPU 上所有相关操作并行执行。考虑到数据结构规则性问题, 图 4 最右侧的椭圆部分是并行任务到达

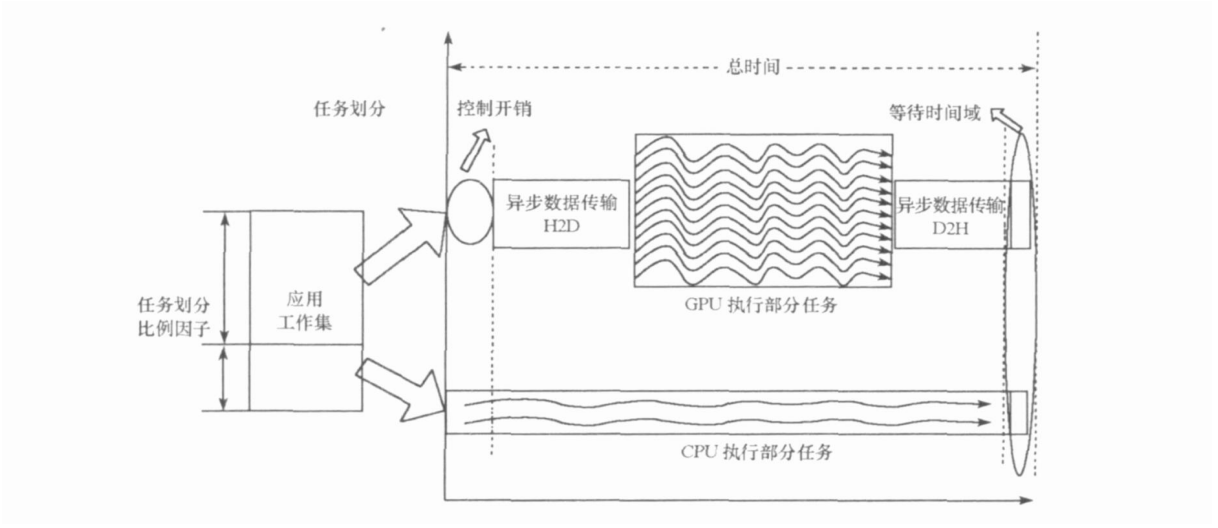

图 4 系统级任务划分

Fig. 4 Task division in system level

同步时间的偏斜区域。在任务划分中如何减少该区域处的差值也需要重点考虑。对于矩阵乘 A × B= C, 根据划分因子, 矩阵 A 被划分为两个子矩阵 Ahost 和 Adevice。因此公式 A × B= C 被划分为 Ahost × B = Chost 和 Adevice × B= Cdevice。由于发射代价可以被忽略, 因此, 可以认为理想的划分因子应该是设备计算能力和系统计算能力之比: Gdevice/(Ghost+ Gdevice)。对于 GPU, 矩阵 Adevice 和 B 被传输到 Device 上, 然后发射 Kernel, 执行, 结果矩阵 Cdevice 异步传回 Host。同时, Ahost × B 在 Host 上计算, 调用 Intel 函数库 MKL10. 1. 0. 018, 结果矩阵 Chost 直接被保存。

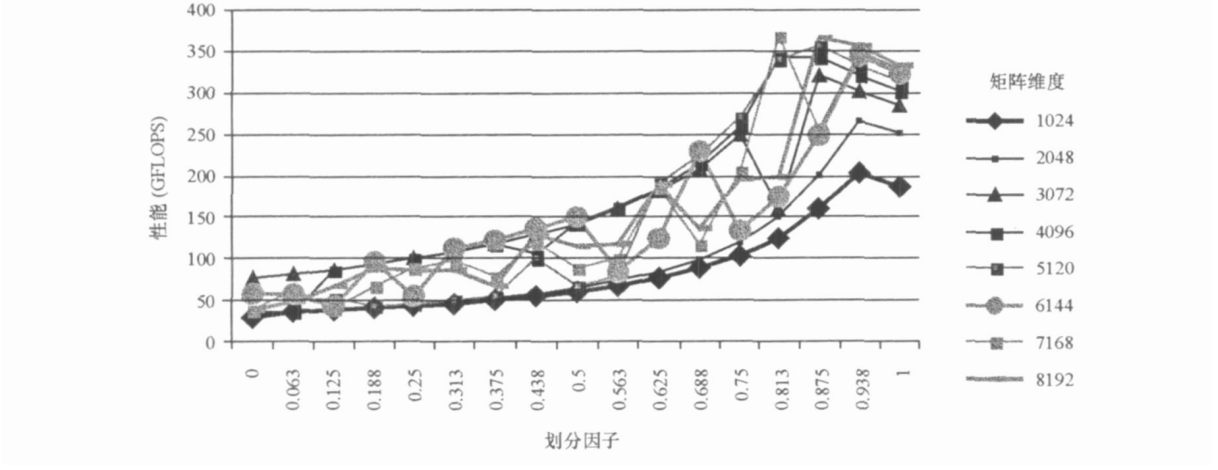

图 5 平台 2 上的系统级任务划分

Fig. 5 Task division on platform2

如图 5 所示, 对于 GPU 执行整个应用, 划分后的应用性能加速比大致为 Ghost/Gdevice。举例说明, 在平台 2 上, 矩阵维度为 4096 时, Ghost 为 35GFLOPS, Gdevice 为 341GFLOPS。当划分因子为 0. 875 时, 系统取得峰值性能 341GFLOPS, 加速比为 13%。而理论划分因子为 0. 89, 理论加速比为 12%。实际结果与理论值非常接近。

3. 4 流化 GPU 上的划分任务

基于以上工作, 在平台 2 上测试了三个优化的任务划分程序, 其中 GPU 的任务分别使用 CUBLAS2. 2、普通优化代码以及流化代码。

如图 6 所示, 大体上, 普通优化的任务划分程序和流化的任务划分程序性能均优于直接调用 CUBLAS2. 2 的任务划分代码的性能。在图中前三分之二段中, 普通优化的任务划分程序性能优于流化任

务划分程序, 在后三分之一段中, 流化任务划分程序性能最优。主要由于额外的发射代价和传输带宽影响, 流化带来的优势直到矩阵尺寸足够大到充分利用了带宽时才显露出来。使用合适的划分因子, 流化因子为 6 的流化任务划分程序的性能达到了 436GFLOPS, 相对于未流化的程序, 性能提升了 2%; 相对于直接调用 CUBLAS 的程序, 性能提升了 18%; 相对于未流化未任务划分的程序, 性能提升了 23.5%。

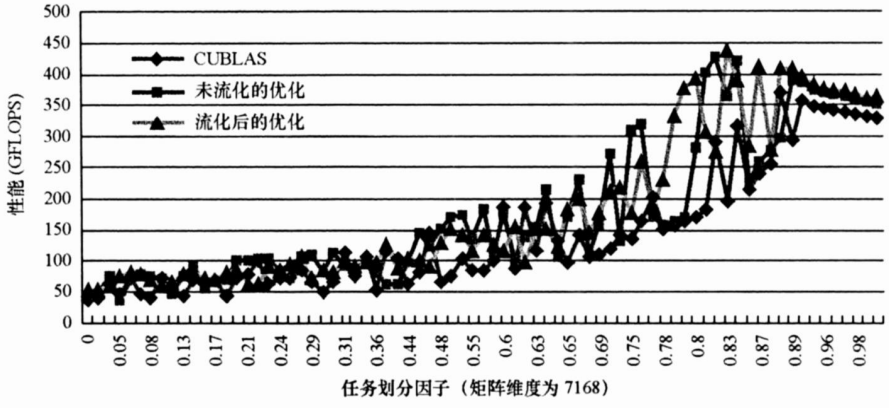

图 6 三种系统级任务划分优化的性能对比

Fig. 6 Comparison of three ways of task division

4 结论和未来工作

通过对 GPU 的体系结构和存储层次的深入分析, 我们认为只有从系统角度充分利用灵活的多级存储层次, 在应用映射时实现负载均衡, 才能够充分挖掘 GPU 的峰值性能, 将 GPU 有效地应用于各种应用领域。因此提出三种不同层次的负载均衡策略, 并有效地应用在矩阵乘算法映射中。

在应用映射时还发现, GPU 自身的体系结构和执行机制还存在着一定的缺陷, 如对 Kernel 级时空局部性支持不足。此外, 执行线程间的通讯和同步模式略微单一, 面对稍微控制复杂的应用, 系统性能大幅下降。因此, 未来我们将通过映射覆盖面更广泛的 Benchmark, 深入探索 GPU 体系结构和运行机制的优缺点, 分析 GPU 在 GPGPU 领域中面临的困境和挑战, 并通过模拟手段, 在 GPU 平台上测试一些有效的体系结构技术和手段。

参 考 文 献:

[1] Halfhill T R. Parallel Processing with CUDA[R]. Microprocessor Report, 2008.
[2] Ghuloum A, Sprangle E, Fang J, et al. Ct: A Flexible Parallel Programming Model for Tera-scale Architectures[R]. Technical Report, Intel Research, 2007.
[3] Gutowitz H. A Tutorial Introduction to Swarm[R]. Technical Report, The Santa Fe Institute, Santa Fe Institute Preprint Series, 1993.
[4] Monteyne M. RapidMind: Multi-core Development Platform[Z]. RapidMind, 2007.
[5] Stone J. Accelerating Computational Biology by 100x with CUDA[R]. Presentation NVISION, 2008.
[6] Hartley T D R, Catalyurek U, Ruiz A, et al. Biomedical Image Analysis on A Cooperative Cluster of Gpus and Multicores[C]// Proceedings of the 22nd Annual International Conference on Supercomputing, New York, NY, USA: ACM, 2008: 15- 25.
[7] Bond A. Havok F X: GPU-accelerated Physics for PC Games[C]// Proceedings of Game Developers Conference, 2006.
[8] Hagen T R, Lie K A, Natvig J R. Solving the Euler Equations on Graphics Processing Units[C]// Proceedings of the 6th International Conference on Computational Science, Lecture Notes in Computer Science, Springer, 2006, 3994: 220- 227.
[9] Zeller C. Cloth Simulation on the GPU[C]// ACM SIGGRAPH 2005 Conference Abstracts and Applications, 2005(8).
[10] Elsen E, Houston M, Vishal V, et al. BN-body Simulation on GPUs[C]// Proc. 2006 ACM/IEEE Conf. on Supercomputing, 2006: 188.
[11] Phillips J C, Braun R, Wang W, et al. Scalable Molecular Dynamics with NAMD[J]. J. Comp. Chem., 2005, 26: 1781- 1802.
[12] Stone J E, Phillips J C, Freddolino P L, et al. Accelerating Molecular Modeling Applications with Graphics Processors[J]. J. Comp. Chem., 2007, 28: 2618- 2640.
[13] Stone S S, Haldar J P, Tsao S C, et al. Accelerating Advanced MRI Reconstructions on GPUs[C]// ACM Computing Frontier Conference, 2008.
[14] OpenVIDIA: GPU-accelerated Computer Vision Library[EB]. openvidia.sourceforge.net, 2006.
[15] Volkov V, Demmel J W. Benchmarking GPUs to Tune Dense Linear Algebra[C]// SC' 08: Proceedings of the 2008 ACM/IEEE Conference on Super-computing, Piscataway, NJ, USA, 2008: 1- 11.
[16] Fatica M. Accelerating Linpack with CUDA on Heterogenous Clusters[C]// GPGPU' 09 ACM, 2009.
[17] Castillo M, Chan E, Igual F D, et al. Making Programming Synonymous with Programming for Linear Algebra Libraries[R]. FLAME Working Note # 31, the University of Texas at Austin, Department of Computer Sciences. Technical Report TR- 08- 20, April 17, 2008.