第35卷 第10期 计 算 机 工 程 2009年5月
Vol.35 No.10 Computer Engineering May 2009
·博士论文· 文章编号：1000—3428(2009)10—0007—04 文献标识码：A 中图分类号：TP312

基于 CUDA 的矩阵乘法和 FFT 性能测试

肖 江，胡柯良，邓元勇
(中国科学院国家天文台，北京 100012)

摘 要：针对 NVIDIA 公司的 CUDA 技术用 Geforce8800GT 在 Visual Studio2008 环境下进行测试，从程序运行时间比较判断 CUBLAS 库、CUDA 内核程序、CUDA 驱动 API、C 循环程序与 Intel MKL 库以及 FFTW 库与 CUFFT 库运行响应的差异。测试结果表明，在大规模矩阵乘法和快速傅里叶变换的应用方面，相对于 CPU，利用 GPU 运算性能可提高 25 倍以上。
关键词：矩阵乘法；快速傅里叶变换；并行计算；GPU 通用计算

Ability Test for Matrix-Multiplication and FFT Based on CUDA

XIAO Jiang, HU Ke-liang, DENG Yuan-yong
(National Astronomical Observatories, Chinese Academy of Sciences, Beijing 100012)

【Abstract】This paper introduces the result of a test that evaluates the effectiveness of Compute Unified Device Architecture(CUDA) using NVDIA GeForce8800GT and the compiler Visual Studio 2008. It tests the speed of NVIDIA CUBLAS, CUDA kernel, common C program, Intel MKL BLAS, CUDA driver API program, FFTW and CUFFT Library in matrix-multiplication and Fast Fourier Transform(FFT). Test result of the large scale data shows that the computing ability of GPU is 25 times better than that of CPU.
【Key words】 matrix-multiplication; Fast Fourier Transform(FFT); parallel computation; GPGPU

1 概述

长期以来，人们对并行计算的需求是无止境的，如在气象、天文，资源以及时系跟踪等领域，它们对程序处理速度的要求都相当高，否则将导致结果出现偏差或失去其意义。文献[1]全面地综述了并行计算在各个方面的最新进展，包括并行计算机体系结构、并行算法、并行性能优化与评价、并行编程等。提高并行运算的速度一般采用以下 3 个方面的改进措施：

(1)处理速度更快的新的硬件设备，如更快的超级计算机、更大的内存以及更快的 I/O 设备。这是从根本上提升并行计算能力的途径。

(2)更优化的程序设计方法和函数库，如在程序中引入多线程、并行等处理方法。

(3)采用优化的软件，这也是一种简便有效且成本相对较低的方法。

采用基于 CUDA(Compute Unified Device Architecture)的 GPU 并行计算属于第(1)种和第(2)种方法的结合。CUDA 是一个新的基础架构，是一个软硬件协同的完整的解决方案。这种架构可以使用 GPU 处理复杂的科学计算问题，特别是极大数据量的并行计算问题。它提供了硬件的直接访问接口，而不必像传统 GPU 方式那样依赖图形 API 接口实现 GPU 的访问[2]。CUDA 在 GPU 架构上将晶体管更多地投入到数据处理，减少数据缓存和流量控制对晶体管资源的消耗。图 1 是最近几年 GPU 与 CPU 每秒浮点运算能力的增长情况[3]。CUDA 采用 C 语言作为编程语言，进行了适度的扩展，提供大量的高性能计算指令开发能力，使开发者能够在 GPU 强大计算能力的基础上建立起一种效率更高的密集数据计算解决方案[4]。

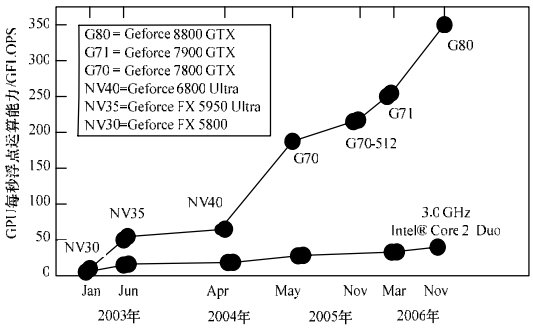

图 1 CPU 与 GPU 的浮点运算速度[3]

本文主要通过 79 MB 的数据量对 NVIDIA 的 GPU 核心芯片(G92)和 Intel Pentium D830 芯片进行矩阵乘法和快速傅里叶变换测试，通过编程评估两者在最优化函数库下的并行运算能力。

2 基于 CUDA 的 GPU 软硬件测试环境

2.1 CUDA 测试硬件的选择

CUDA支持的GPU(CUDA-enabled GPU)包括NVIDIA公司的Geforce, Quadro和Tesla 3个产品线。其中，Geforce 和Quadro系列显示芯片可以直接插入普通PCI-Express×16插槽中，最大理论带宽为8 GB/s[5]，为了便于将CPU与GPU进行性

基金项目：国家“973”计划基金资助项目(2006CB806301)；国家自然科学基金资助项目(10473016, 10673016)；中国科学院知识创新工程基金资助项目(KJCX2-YW-T04)
作者简介：肖 江(1982－)，男，博士研究生，主研方向：并行计算，嵌入式软件环境，图像处理；胡柯良，副研究员；邓元勇，研究员、博士生导师
收稿日期：2008-10-20 **E-mail：**xj@bao.ac.cn

能比较，选用丽台公司PX8800GT显卡，显卡采用Geforce 8800GT显示芯片。该芯片采用G92显示核心，有112个流处理器，内核时钟为600 MHz，内存带宽为57.6 GB/s，支持并行数据高速缓存(parallel data cache)。

支持 CUDA 并行运算的 GPU 如表 1 所示。

表 1　支持 CUDA 并行计算的 GPU 芯片

产品型号	Number of Multiprocessors	Compute Capability
GeForce 8800 Ultra,8800 GTX	16	1.0
GeForce 8800 GT	14	1.1
GeForce 8800M GTX	12	1.1
GeForce 8800 GTS	12	1.0
GeForce 8800M GTS	8	1.1
GeForce8600 GTS，8600 GT, 8700M GT,8600M GT,8600M GS	4	1.1
GeForce8500 GT,8400 GS, 8400M GT,8400M GS	2	1.1
GeForce 8400M G	1	1.1
Tesla S870	4×16	1.0
Tesla D870	2×16	1.0
Tesla C870	16	1.0
Quadro Plex 1000 Model S4	4×16	1.0
Quadro Plex 1000 Model IV	2×16	1.0
Quadro FX 5600	16	1.0
Quadro FX 4600	12	1.0
Quadro FX 1700,FX 570, NVS 320M,FX 1600M,FX 570M	4	1.1
Quadro FX 370,NVS 290, NVS 140M, NVS 135M,FX 360M	2	1.1
Quadro NVS 130M	1	1.1

2.2　CUDA 测试软件的选择

在 CUDA 的软件层面，核心是 NVIDIA 编译器 nvcc.exe。CUDA 程序是 GPU 和 CPU 的混合编码，其程序文件包括.cu 和.cpp，且程序中混合着 C 语言函数和 CUDA 语言函数。程序首先通过 C 编译器将 GPU 与 CPU 的代码分离，GPU 代码被编译成能交给 GPU 计算的代码，而 CPU 代码编译成标准 CPU 机器码。因此，一个完整的 CUDA 软件开发环境需要有 2 个编译器：面向 CPU 的 C 编译器和面向 GPU 的编译器 nvcc.exe。CUDA 可以支持多种运行在 Windows XP 和 Linux 操作系统下的编译系统。考虑到通用性，在本文测试中采用 Microsoft Visual Studio 2008，其 C 编译器为 cl.exe。CUDA 软件架构如图 2 所示。

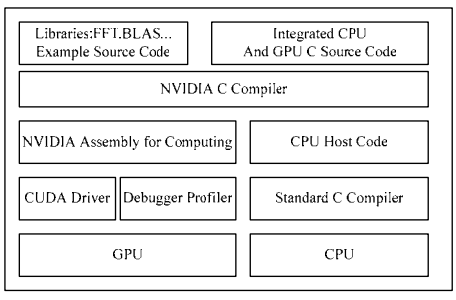

图 2　CUDA 软件架构

从图 2 可以看出，在 CUDA 软件堆的最上层提供了 CUBLAS(CUDA Basic Linear Algebra Subprograms) 和 CUFFT(CUDA Fast Fourier Transform)函数库，它是基于 NVIDIA CUDA 驱动程序上的子程序，可以访问 NVIDIA GPU 的计算资源。CUBLAS 和 CUFFT 函数库自身包含 API 层，可以不与 CUDA 驱动程序直接相互作用。这是 Nvidia 公司优化过的函数库，在进行大规模矩阵乘法运算和快速傅里叶变换的测试中，该函数库将提供 GPU 最雀的运算性能。

3　基于 MKL 的 CPU 软硬件测试环境

3.1　CPU 测试硬件的选择

为便于比较，CPU 采用 Pentium D830 64 位处理器，该处理器主频为 3.0 GHz，内存带宽为 4 429.8 MB/s，峰值运算能力约为 31.4 GFLOPS。该处理器的浮点运算能力基本代表了 Intel 系列高端 CPU 的水平。

3.2　CPU 测试软件的选择

为配合 Intel CPU 的测试，选用英特尔数学核心函数库(英特尔 MKL)。该函数库是经过高度优化和广泛线程化的数学例程函数库，专为需要高性能的科学、工程、天体物理及分子动力模拟等领域的应用而设计。它能针对不同的处理器采用不同的线程处理方式，在科学计算领域，使处理器的运算性能得以充分发挥。该函数库中包括了经过高度优化的 BLAS 例程，其提供的性能改善非常明显，远远超出其他替代性实现[6]，MKL BLAS 函数库能发挥 Pentium D830 处理器在矩阵运算中的最高运算能力。

本文采用 FFTW 进行快速傅里叶变换的测试。FFTW 是目前运行速度最快的 FFT 软件包，用 C 语言开发，其核心技术(编码生成器)采用面向对象设计方式和面向对象语言 Caml 编写。FFTW 能自动适应 CPU 环境，可移植性强，运行速度快，可进行一维和多维的离散傅里叶变换(discrete Fourier transform)，其数据类型可以是实型、复型或半复型，用 FFTW 进行傅里叶变换能充分发挥 CPU 的运行能力。

4　大数据量矩阵乘法测试

4.1　测试指标

本文主要包括 3 个测试指标：(1)数据计算量；(2)运算时间；(3)浮点运算能力。

4.2　测试方法

测试中调用了 CUDA CUBLAS 函数库的程序代码、Intel MKL 函数库的程序代码和 Nvidia CUDA 通用内核程序代码，普通的 C 语言代码放到一个程序段中进行编译，测试程序分 6 步完成。

4.2.1　矩阵数组

调用 srand, rand 函数产生一个行数为 5×512、列数为 3×512 的 A 矩阵以及一个行数为 3×512、列数为 8×512 的 B 矩阵，将产生的随机数保存到.txt 文件下。

4.2.2　CUBLAS 库矩阵乘法

首先在 CPU 中建立存储空间，将.txt 文档中的数据调入主机(host)的内存中，在 GPU(device)存储空间中建立矩阵和矢量对象，填入主机内存中的数据，调用一连串的 CUBLAS 功能，最后返回计算结果到主机。CUBLAS 提供的辅助函数可以在 GPU 空间建立和结束对象，可以从这些对象处写入和重新得到数据。CUBLAS 测试代码如下：

```
//初始化 CUBLAS 函数
status = cublasInit();
//在 GPU 上创建存储空间 d_Ablas 用于存放矩阵 A，d_Bblas 用
//于存放矩阵 B，d_Cblas 用于存放 A*B 的结果
status = cublasAlloc(size_A, sizeof(d_Ablas[0]), (void**)&
d_Ablas);
status = cublasAlloc(size_B, sizeof(d_Bblas[0]), (void**)&
d_Bblas);
status = cublasAlloc(size_Cblas, sizeof(d_Cblas[0]), (void**)&
```

```
d_Cblas);
    //单精度 BLAS3 矩阵乘法函数
    cublasSgemm('n', 'n', 5×32×16, 8×32×16, 3×16×32, alpha, d_Ablas, HA, d_Bblas, HB, beta, d_Cblas, HC);
    //将 GPU 测试结果返回到主机
    cublasGetVector(size_Cblas, sizeof(h_Cblas[0]), d_Cblas, 1, h_Cblas, 1);
```

4.2.3 GPU 通用内核矩阵乘法

GPU 通用内核矩阵乘法包含 2 个文件：Mul.cu 文件(主文件)是 Mul-kernel.cu 文件(内核文件)。主文件调用内核文件用于 GPU 并行处理，内核文件将 A 和 B 划分成若干个小块，每个小块分别交给 GPU 不同的 block 处理，每个块大小为 32×32，即每个 block 并行处理 32×32 个线程，运算速度大大加快。并行运算代码如下：

```
    //在 GPU 上分配存储空间，用于存放 A,B 矩阵
    float* d_A;
    CUDA_SAFE_CALL(cudaMalloc((void**) &d_A,mem_size_ A));
    float* d_B;
    CUDA_SAFE_CALL(cudaMalloc((void**) &d_B, mem_size_B));
    //将 CPU 上的数组元素传入 GPU
    CUDA_SAFE_CALL(cudaMemcpy(d_A, h_A, mem_size_A, cuda MemcpyHostToDevice) );
    CUDA_SAFE_CALL(cudaMemcpy(d_B, h_B, mem_size_B, cudaMemcpyHostToDevice) );
    //将 GPU 运算单元分成 16×16 运算块，需要分配的格点的坐标
    //为 C 的列数和行数与 Block 大小的比值
    dim3 threads(32, 32);
    dim3 grid(WC / threads.x, HC / threads.y);
    //执行矩阵乘法内核运算
    Mul<<< grid, threads >>>(d_C, d_A, d_B, WA, WB);
    //矩阵乘法内核，返回计算结果到主机
    for (int k = 0; k < BLOCK_SIZE; ++k)
            {Csub += AS(ty, k) * BS(k, tx);}
    int c = wB×32×by + 32 * bx;
    C[c + wB×ty + tx] = Csub;
```

4.2.4 CUDA 驱动 API 程序

CUDA 驱动 API 程序的编程思路与内核程序类似，其主要思想是先在 CPU 上分配内存空间，将 CPU 上的数据传入 GPU 中，再在 GPU 上进行并行处理，但是驱动 API 在 CUDA 软件堆中属于底层函数库，其编写较内核程序繁琐，但是效率更高。以下是 CUDA 驱动 API 运算部分的程序代码，其中，matrixMul 是内核执行程序；BLOCK_SIZE 是自定义线程块的大小：

```
    CU_SAFE_CALL(cuFuncSetBlockShape( matrixMul, BLOCK_ SIZE, BLOCK_SIZE, 1 ));
        CU_SAFE_CALL(cuFuncSetSharedSize( matrixMul, 2 × BLOCK_SIZE×BLOCK_SIZE×sizeof(float) ) );
        CU_SAFE_CALL(cuParamSeti( matrixMul, 5×512×8×512, d_C ));
        CU_SAFE_CALL(cuParamSeti( matrixMul, 3×512×5×512, d_A ));
        CU_SAFE_CALL(cuParamSeti( matrixMul, 3×512×8×512, d_B ));
    CU_SAFE_CALL(cuParamSeti( matrixMul, 3×512, WA ));
        CU_SAFE_CALL(cuParamSeti( matrixMul, 8×512, WB ));
        CU_SAFE_CALL(cuParamSetSize( matrixMul, 20 ));
    CU_SAFE_CALL(cuLaunchGrid( matrixMul, WC/BLOCK_SIZE, HC/BLOCK_SIZE ));
```

4.2.5 Intel MKL 库矩阵乘法

Intel MKl 库函数内容极其丰富，涵盖线性代数、随机数产生和 FFT 等方面，为测试 CPU 的矩阵运算性能，本文采用线性代数函数库(BLAS)，该函数库包括 BLAS1, BLAS2, BLAS3 三种类型的运算，本文选用 BLAS3(matrix- to-matrix)进行运算。Intel MKL-blas 运算代码如下：

```
    //矩阵 A 与矩阵 B 相乘并将结果返回到主机 Cintelblas 矩阵
    cblas_sgemm(CblasColMajor,CblasNoTrans,CblasNoTrans, HA, WB, HB, alpha, h_A, HA, h_B, HB, beta, Cintelblas, HC);
```

编译 MKL 库函数需要在函数中添加 MKL.H 头文件，将应用程序的 lib 库指向 MKL 函数库，连接器命令行添入 cublas.lib mkl_c.lib mkl_c_dll.lib mkl_intel_thread.lib，为使测算函数运行时间的准确还需调用函数 QueryPerformance Frequency();QueryPerformanceCounter()计算 CPU 时钟的开销周期数。

4.2.6 普通矩阵乘法代码

普通矩阵乘法代码是采用普通 C 循环语句做的矩阵乘法，通过反复调用自定义函数实现矩阵乘法运算。矩阵乘法代码如下：

```
Mul(float* C, const float* A, const float* B, unsigned int HA, unsigned int WA, unsigned int WB)
{ for (unsigned int m = 0; m < HA; ++m)
            for (unsigned int n = 0; n < HB; ++n)
 {     double sum = 0;
                for (unsigned int k = 0; k < WA; ++k)
            {double a = A[m×WA + k];
                    double b = B[k×WB + n];
                    sumC += a * b;                    }
                C[m * WB + n] = (float)sumC;       }}
```

5 大数据量快速傅里叶变换测试

5.1 测试指标

这里主要包括 2 个测试指标：(1)数据量大小；(2)运算量和运算时间。

5.2 测试方法

测试中分别调用了 CUDA CUFFT 函数库的程序代码和 FFTW 函数库的程序代码，测试程序用 4.2.1 节产生的数据完成了数据的 2 次一维傅里叶变换，分 2 步完成。

5.2.1 CUFFT 库程序

首先需要将.txt 文档中的数据装入 GPU memory 中，定义数据为复数的内存空间：

```
Complex × h_signal=(Complex*)malloc(sizeof(Complex) × SIGNAL_ SIZE);
```

建立 plan，完成 FFT 变换，变换部分的代码如下：

```
CUFFT_SAFE_CALL(cufftExecC2C(plan1, (cufftComplex ×) d_signal, (cufftComplex×)d_signal, CUFFT_FORWARD));
CUFFT_SAFE_CALL(cufftExecC2C(plan2, (cufftComplex ×) d_filter_kernel, (cufftComplex *)d_filter_kernel, CUFFT_ FORWARD));
```

5.2.2 FFTW 库程序

FFTW 库函数程序首先需要产生相应的 libfftw3-3.lib libfftw3f-3.lib 或 libfftw3l-3.lib 文件，这些文件是编译文件调用的库函数，需要用户自己在 DOS 环境下编译生成。FFTW 库函数程序与 CUFFTW 库函数程序类似，其过程也是先建立一个 plan，再完成傅里叶变换函数的调用，程序如下：

```
plan=fftw_plan_dft_1d(SIGNAL_SIZE,h_signal,h_signal,FFTW_ FORWARD,FFTW_ESTIMATE);
```

```
plan=fftw_plan_dft_1d(FILTER_KERNEL_SIZE,h_filter_kernel,h
_filter_kernel,FFTW_FORWARD,FFTW_ESTIMATE);
```

6 测试结果

图 3 是运算测试结果。

图 3 测试结果

从图3可以看出，在79 MB 数据量和每秒32.2 Billion运算量的情况下，利用GPU进行大规模矩阵乘法运算能力最强，CUBLAS平均浮点运算能力达到530.28 GFLOPS，利用通用内核矩阵和CUDA驱动程序(API)的浮点运算能力分别可以达到444.299 GFLOPS和430.925 GFLOPS，而利用CPU最优化的函数库Intel MKL的浮点运算能力为21.47 GFLOPS，与GPU最优化的函数库CUBLAS相比相差25倍。利用GPU进行快速傅里叶变换所需要的处理时间为64.27 ms，而利用FFTW在CPU上计算相同的数据量处理时间为1 000 ms。由此可见，在大规模并行运算方面通用GPU的运算能力更强。

7 结束语

通过对 GPU 和 CPU 在不同的最优化函数库下进行性能测试可以发现，在科学计算方面 GPU 有更强大的运算能力。作为刚开发出的一项新技术，国内还缺少对 CUDA 技术广泛的开发和应用。CUDA 新技术的诞生将人们需要的大型计算从大型计算机转移到普通电脑上，为科学领域进行大规模运算提供了新的研究方法，在未来的天文数值模拟、相关算法的应用以及电信、金融、证券数据分析等领域具有广阔的应用前景。

参考文献

[1] Dongarra J, Foster I, Fox G, et al. Sourcebook of Parallel Computing[M]. [S. l.]: Elsevier Science, 2003.

[2] 杨 兵, 李凤霞, 战守义, 等. GPU 在复杂场景的阴影绘制中的应用[J]. 计算机工程, 2006, 32(2): 220-222.

[3] CUDA Programming Guide 1.1[Z]. (2007-09-11). http://developer.nvidia.com/object/cuda.html.

[4] CUDA——走向 GPGPU 新时代[J]. 程序员, 2008, 10(3): 36-39.

[5] 龚敏敏. GPU 精粹 2[M].北京：清华大学出版社, 2007.

[6] Intel Math Kernel Library for the Windows* Operating System Users' Guide[Z]. (2007-09-11). http://www.intel.com.

编辑 张正兴

(上接第 6 页)

表 1 各模块的面积及在网络接口中所占比例

模块名称	面积/μm²	所占比例/(%)
数据包输出缓存	102 481.9	33.5
控制包输出缓存	44 645.6	14.6
数据包输入缓存	74 735.7	24.4
控制包输入缓存	14 891.4	4.9
AHB 从接口	1 630.9	0.5
AHB 主接口	1 481.1	0.5
本地读写请求模块	8 030.5	2.6
远程读写请求模块	18 543.7	6.1
DMA 控制器	22 914.4	7.5
寄存器读写模块	15 914.3	5.2
网络接口(总计)	305 768.8	100.0

可见，主要的面积用于输入/输出缓存的 FIFO 及路由表中。使用工具自动布局布线后的 NI 版图如图 5 所示。

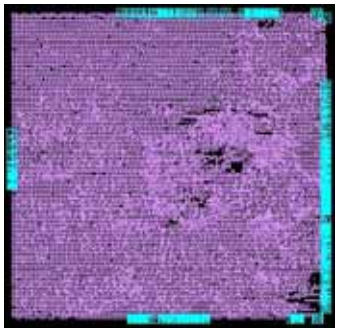

图 5 NI 版图

6 结束语

本文介绍连接双通道路由器及资源块的网络接口结构和模块，给出其在各种模式下的相应操作。提出一种使用 AMBA 接口协议的适用于双通道路由器组成的片上网络系统的网络接口，并对其进行物理实现。

参考文献

[1] Kumar S, Jantsch A, Millberg M, et al. A Network on Chip Architecture and Design Methodology[C]//Proc. of ISVLSI'02. Pittsburgh, Pennsylvania, USA: IEEE Press, 2002: 105-112.

[2] 朱樟明, 周 端, 杨银堂. 片上网络体系结构的研究与进展[J]. 计算机工程, 2007, 33(24): 239-241.

[3] Radulescu A, Dielissen J, Pestana S G, et al. An Efficient on-Chip NI Offering Guaranteed Services, Shared-memory Abstraction, and Flexible Network Configuration[J]. IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, 2005, 24(1): 4-17.

[4] Bertozzi D, Benini L. Xpipes: A Network-on-chip Architecture for Gigascale Systems-on-Chip[J]. IEEE Circuits and Systems Magazine, 2004, 4(2): 18-31.

[5] Xu Shidong, Lin Yuxuan, Zhou Zheming. Design of a Dual-mode NoC Router Integrated with Network Interface for AMBA-based IPs[C]//Proc. of ASSCC'06. Hangzhou, China: IEEE Press, 2006: 211-214.

编辑 金胡考