基于 CUDA 架构矩阵乘法的研究

马梦琦，刘 羽，曾胜田
（桂林理工大学 信息科学与工程学院，广西 桂林 541004）

摘 要：首先介绍了 CUDA 架构特点，在 GPU 上基于 CUDA 使用两种方法实现了矩阵乘法，并根据 CUDA 特有的软硬件架构对矩阵乘法进行了优化。然后计算 GPU 峰值比并进行了分析。实验结果表明，基于 CUDA 的矩阵乘法相对于 CPU 矩阵乘法获得了很高的加速比，最高加速比达到 1 079.64。GPU 浮点运算能力得到有效利用，峰值比最高达到 30.85%。

关键词：CUDA；矩阵乘法；加速比；峰值比
中图分类号：TP301 **文献标识码**：A **文章编号**：1674-7720(2011)24-0062-03

Research of matrix multiplication based on CUDA architecture

Ma Mengqi, Liu Yu, Zeng Shengtian
(School of Information Science and Engineering, Guilin University of Technology, Guilin, 541004, China)

Abstract: This paper firstly introduced the characteristics of CUDA architecture, realized matrix multiplication using two ways on the GPU, and optimized the matrix multiplication according to unique hardware and software architecture based on CUDA. Then calculated and analyzed the peak ratio of GPU. Experimental results showed that CUDA-based matrix multiplication on the GPU achieved a higher speed-up ratio compared with that on the CPU. The maximum speedup to 1 079.64. The capability of floating-point calculations on the GPU was effectively taken advantage of, the highest peak ratio reached more than 30.85%.

Key words: CUDA; matrix multiplication; speed-up ratio; peak ratio

随着多核 CPU 和众核 GPU 的快速发展，计算行业正在从只使用 CPU 的“中央处理”向 CPU 与 GPU 并用的“协同处理”发展，并行系统已成为主流处理器芯片。传统的 GPU 架构受其硬件架构的影响不能有效利用其资源进行通用计算，NVIDIA（英伟达）公司推出的统一计算设备架构 CUDA（Compute Unified Device Architecturem），使得 GPU 具备更强的可编程性，更精确和更高的性能，应用领域也更加广泛。

矩阵乘法是一种大计算量的算法，也是很耗时的运算。CPU 提高单个核心性能的主要手段比如提高处理器工作频率及增加指令级并行都遇到了瓶颈，当遇到运算量大的计算，CPU 进行大矩阵的乘法就变得相当耗时，运算效率很低下。因此，GPU 凭借其超强计算能力应运而生，让个人 PC 拥有了大型计算机才具备的运算能力。本文运用 GPU 的超强计算能力在 CUDA 架构上实现了大矩阵乘法。

1 CUDA 架构

NVIDIA 及时推出 CUDA 这一编程模型，在应用程序中充分结合利用 CPU 和 GPU 各自的优点，特别是 GPU 强大的浮点计算能力。CPU 主要专注于数据高速缓存（cache）和流处理（flow control），而 GPU 更多地专注于计算密集型和高度并行的计算。尽管 GPU 的运行频率低于 CPU，但 GPU 凭着更多的执行单元数量使其在浮点计算能力上获得较大优势[1]。当前的 NVIDIA GPU 中包含完整前端的流多处理器（SM），每个 SM 可以看成一个包含 8 个 1D 流处理器（SP）的 SIMD 处理器。主流 GPU 的性能可以达到同期主流 CPU 性能的 10 倍左右。图 1 所示为 GPU 与 CPU 峰值浮点计算能力的比较。

CUDA 的编程模型是 CPU 与 GPU 协同工作，CPU 作为主机（Host）主要负责逻辑性强的事务处理及串行计算，GPU 作为协处理器或者设备（Device）负责密集型的大规模数据并行计算。一个完整的 CUDA 程序=CPU 串行处理+GPU Kernel 函数并行处理。

一个 CUDA 架构下的程序分为两个部分，即上述的 Host 端和 Device 端。通常情况下程序的执行顺序如下：Host 端程序先在 CPU 上准备数据，然后把数据复制到

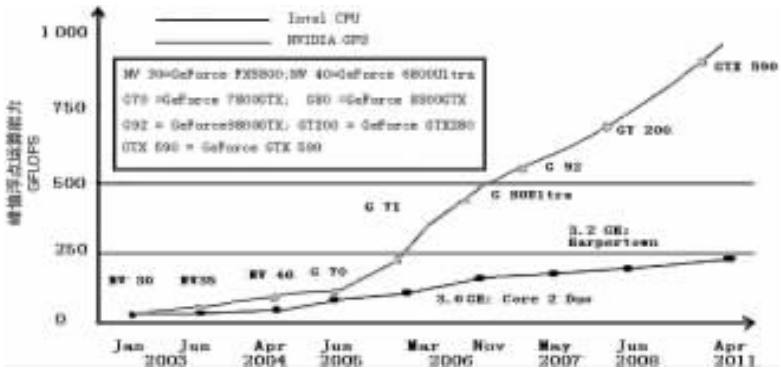

图 1 GPU 与 CPU 峰值浮点计算能力的比较

显存中，再由 GPU 执行 Device 端程序来处理这些数据，最后 Host 端程序再把结束运算后的数据从显存中取回。

图 2 为 CUDA 编程模型，从中可以看出，Thread 是 GPU 执行运算时的最小单位。也就是说，一个 Kernel 以线程网格 Grid 的形式组织，每个 Grid 由若干个线程块 Block 组成，而每个线程块又由若干个线程 Thread 组成。一个 Kernel 函数中会存在两个层次的并行，Grid 中 Block 之间的并行和 Block 中 Thread 之间的并行，这样的设计克服了传统 GPGPU 不能实现线程间通信的缺点[2]。

图 2 CUDA 编程模型

同一个 Block 下的 Thread 共用相同的共享存储器，通过共享存储器交换数据，并通过栅栏同步保证线程间能够正确地共享数据。因此，一个 Block 下的 Thread 虽然是并行的，但在同一时刻执行的指令并不一定都相同，实现了不同 Thread 间的协同合作。这一特性可以显著提高程序的执行效率，并大大拓展 GPU 的适用范围。

2 基于 CUDA 架构矩阵乘法的实现

2.1 一维带状划分

给定一个 $M\times K$ 的矩阵 A 和一个 $K\times N$ 的矩阵 B，将矩阵 B 乘以矩阵 A 的结果存储在一个 $M\times N$ 的矩阵 C 中。此种矩阵乘法使用了一维带状划分，每个线程将负责读取矩阵 A 中的一行和 B 中的一列，矩阵进行乘法运算并将计算结果存储在全局存储器。

全局存储器会对矩阵 A 进行 N 次读取，对矩阵 B 进行 M 次读取。假设数组在每个维度上的尺寸都是 BLOCK_SIZE 的整数倍。若矩阵大小为 32×32，则可表示为(2×16)×(2×16)。下面的内核定义中，结果矩阵 C 中的每个元素由一个线程负责，for()循环完成 A 中第 X 行与 B 中第 X 列对应元素的乘加运算，并将结果累加到 Cvalue。

```
For( int e=0;e < A.width;++e)
Cvalue+=A.elements[row*A.width+e] *
                    B.elements[e*B.width+col];
C.elements[row*width+col]=Cvalue;
```

在矩阵相乘实现中，这个内核运算的速度不尽人意，主要瓶颈在于对内存的重复读取，计算量是 $2\times M\times N\times K$ flop，而全局内存的访问量为 $2\times M\times N\times K$ B[3]。若矩阵维数为 1 024×1 024，则此次矩阵相乘的计算量就有 2 G flop，当矩阵维数更大时，这个运算量就相当大，在内存的读取上会浪费大量的时间。

2.2 二维棋盘划分

因为矩阵 A 的行和矩阵 B 的列多次被读取，为了避免重复加载，选择把矩阵进行分块运算，使用 shared memory 来实现矩阵乘法。运用 shared memory 的好处在于其延迟小于 global memory，并且还能使线程间进行通信。矩阵 A 只被读了 N/BLOCK_SIZE 次，矩阵 B 仅被读了 M/BLOCK_SIZE 次，节省了大量的 global memory 带宽。

首先把划分的小矩阵块加载到 share memory，则小矩阵本身的乘法就不用去存取外部的任何内存了，因此在二维棋盘划分中，矩阵乘法的计算量仍然是 $2\times M\times N\times K$ flop，b 是矩阵 B 划分的小矩阵块的大小，则全局内存访问量是 $2\times M\times N\times K/b$ B。

棋盘划分运算可以表示为：C 矩阵的(0,0)~(15,15)=A(0~15,0~15)×B(0~15,0~15)+A(0~15,16~31)×B(16~31,0~15)+A(0~15,32~47)×B(32~47,0~15)+…+A(0~15,(16×(n−1)−1)~(16×(n−1))×B ((16×(n−1)−1)~(16×(n−1)),0~15)。

```
for (int j=0;j<wA;j+=BLOCK_SIZE)
{       //声明用于存储 A,B 子块的 share memory 数组
  __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
  __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];
}…
    //两个子块的乘加，每个线程负责 C 中一个元素值的计算
  for (int k = 0; k < BLOCK_SIZE; ++k)
  {
    float t;
    C sub + = As [ ty ] [ k ] * Bs [ k ] [ tx ];
    Cs [ ty ] [ tx ] = C sub;
  }
__syncthreads();
….
C [(by*BLOCK_SIZE +ty)*wA +bx*BLOCK_SIZE +tx] =
Csub;
dim3 myblock(BLOCK_SIZE,BLOCK_SIZE,1);
```

```
dim3mygrid(((wB+BLOCK_SIZE-1)/BLOCK_SIZE),
                (wB+BLOCK_SIZE-1)/BLOCK_SIZE,1);
```

根据 NVIDIA CUDA Programming Guide，一个 Block 里至少要有 64 个 Thread，最多有 512 个 Thread。官方建议 256 个 Thread 是最合适的，因为此时有足够多的 active warp 有效地隐藏延迟，使得 SM 能够尽量满负荷工作[4]。为便于理解，假设矩阵为 $n\times n$，此时 BLOCK_SIZE 设置为 16，使用 dim3 来设计，每个 Block 包含 16×16 个 Thread，一个 Grid 共有 $(n/16)\times(n/16)$ 个 Block。

BLOCK_SIZE 是不是越大越好呢？这样一个 SM 里的 Thread 就更多，虽然 Thread 越多越能隐藏 latency，但 G80/G92 架构每个 SM 上 shared memory 仅有 16 KB，这会让每个 Thread 能使用的资源更少，效率反而会下降。

2.3 根据 CUDA 架构对矩阵乘法进行优化

因为棋盘划分中涉及到的是二维数组，cudaMalloc2D()能确保分配二维数组并且能分配适当的填充以满足对齐要求，还能确保在访问行地址或者二维数组与其他设备内存之间的数据复制能达到最佳性能。

二维棋盘划分方法仅限于数组大小必须是 BLOCK_SIZE 的整数倍，若矩阵维数并不是 16 的整数倍，则会造成运算效率的下降，此时可以利用 CUDA 架构特点和 CUDA 提供的 cudaMallocPitch()函数来解决此问题。cudaMallocPitch()可以自动地以最佳倍数来分配内存。

呼叫 Kernel 部分需要修改成：

```
matrixMul<<<mygrid,myblock>>>(d_A,d_B,d_C,wA,wB,
d_pitchA/sizeof(float),d_pitchB/siz
eof(float),d_pitchC/sizeof(float));
```

cudaMalloc 部分改成：

```
float* d_A;
cutilSafeCall(cudaMallocPitch((void**)&d_A,&d_pitchA,
wA*sizeof(float),wB));
float* d_B;
cutilSafeCall(cudaMallocPitch((void**)&d_B,&d_pitchB,
wB*sizeof(float),wA));
float* d_C;
cutilSafeCall(cudaMallocPitch((void**)&d_C,&d_pitchC,
wB*sizeof(float),wB));
```

矩阵内存与显存之间的读取都需要做相应的修改：

```
cutilSafeCall(cudaMemcpy2D(d_A,d_pitchA,A,wA*sizeof(float),
wA*siz
eof(float),wB,cudaMemcpyHostToDevice));
cutilSafeCall(cudaMemcpy2D(d_B,d_pitchB,B,wB*sizeof(float),
wB*sizeof(float),wA,cudaMemcpyHostToDevice));
cutilSafeCall(cudaMemcpy2D(C,wB*sizeof(float),d_C,d_pitchC,
wB*sizeof(float),wB,cudaMemcpyDeviceToHost));
```

在数值分析，Kahan 求和算法(也称作补偿总和)能显著减少浮点数运算的误差，在 CUDA 矩阵乘法中可以通过使用 Kahan 求和算法来提高计算精准度[5]。算法如下：

```
for (int k = 0; k < BLOCK_DIM; ++k)
{
        float t;
    comp -= AS[ty][k] * BS[k][tx];
    t = Csub - comp;
    comp = (t - Csub) + comp;
    Csub = t;
}
```

3 测试环境及实验结果

测试的硬件环境：CPU 使用的是 AMD Athlon II X2 245 处理器，核心数为 2，该处理器主频为 2.9 GHz，峰值运算能力约为 17.4 GFLOPS；GPU 使用的是 NVIDIA GeForce 9800M GTS，有 8 个 SM 即有 64 个 SP 单元，显存带宽为 51.2 GB/s，GPU 核心频率为 0.625 GHz，单精度浮点计算能力为 240 GFLOPS，属于 NVIDIA 中端显卡。测试的软件环境：Windows XP 系统，CUDA toolkit 3.0，Visual Studio 2008，CUDA 计算能力为 1.1。

在程序运行的测试中，对矩阵规模由 256×256～2 048×2 048 逐渐增大，实验数据均是三次测试取得的平均值，这样实验的结果更准确。加速比是指程序在 CPU 上运行的时间与程序在 GPU 上运行所需的时间之比。峰值比是指运算速度与 GPU 单精度浮点运算能力之比。最后求得在各种矩阵规模运行下的加速比及峰值比。实验结果如表 1 所示。

实验结果表明：当矩阵维数小于 320×320 时，带状划分加速比小于 1，说明 CPU 运算时间要小于一维带状划分时 GPU 的运算时间，这说明 GPU 计算时，从内存复制矩阵到显存和把结果矩阵从显存拷贝回内存过程中消耗了一些时间[6]。随着矩阵维数的增大，CPU 的运算时间呈现级数增长，而 GPU 运算时间只是小幅度增长。此时 GPU 强大的浮点运算能力凸显出来，加速比在矩阵维数为 2 048 时最大为 1 079.64，CPU 上 Intel MKL 矩

表 1 不同矩阵规模的加速比及 GPU 峰值比

矩阵规模	带状划分		棋盘划分		优化的矩阵乘法	
	加速比	峰值比/%	加速比	峰值比/%	加速比	峰值比/%
256×256	0.59	0.08	1.57	0.23	3.57	0.53
320×320	1.004	0.12	2.56	0.32	5.46	0.70
512×512	4.57	0.43	11.6	0.88	28.31	2.16
1 024×1 024	51.88	1.47	109.18	3.10	301.65	8.58
2 048×2 048	216.93	6.19	599.23	15.97	1 079.64	30.85

阵乘法比文中所用的 CPU 矩阵乘法快了 200 多倍，但是依靠 GPU 流多处理的并行执行能力，GPU 上的实现方法还是比 Intel MKL 快了 5 倍左右。运用 CUDA 的软硬件架构使得 GPU 合理组织数据，使得内存的读取节省了大量时间。峰值比也有很大的提高，峰值比说明了算法对 GPU 强大浮点运算能力的利用，对 GPU 相应算法的对比具有很高的参考价值。

通过矩阵乘法在 CPU 与 GPU 上不同的性能表现可以发现，NVIDIA 公司推出的 CUDA 使某些大运算量的计算可以从大型计算机或者超级计算机转移到个人 PC，这一新技术不仅使科研缩减了成本，同时也为科学领域进行大规模运算提供了新方法[7]。对于它的未来值得期待，毕竟 CUDA 已经在影视制作、计算金融、流体力学、医学成像、石油天然气数据收集、地质勘探及超级计算机的建立等领域取得了成功。

参考文献

[1] NVIDIA Corporation.NVIDIA CUDA Programming Guide Version3.0[EB/OL].(2010-02-10)[2011-08-20].http://cuda.csdn.net/.

[2] 张舒，褚艳利，赵开勇，等.GPU 高性能并行运算之 CUDA[M].北京：中国水利水电出版社，2009.

[3] Ye Zhenyu.GPU assignment 5KK70[DB/OL].(2009-11-05)[2011-09-01].http://wenku.baidu.com/view/9cd2e372027-68e9951e738e5.html.

[4] NVIDIA Corporation.NVIDIA CUDA CUBLAS library PG-00000-002_V3.0[EB/OL].(2010-02-10)[2011-09-10].http://cuda.csdn.net/.

[5] Hotball.深入浅出谈 CUDA 技术[DB/OL].(2008-11-21)[2011-09-15].http://www.pcinlife.com/article/graphics/2008-06-04/1212575164d532_3.html.

[6] 刘进锋，郭雷.CPU 与 GPU 上几种矩阵乘法的比较与分析[J].计算机工程与应用，2011，47(19)：9-23.

[7] 肖江，胡柯良，邓元勇.基于 CUDA 的矩阵乘法和 FFT 性能测试[J].计算机工程，2009.35(10)：7-10.

(收稿日期：2011-09-23)

作者简介：

马梦琦，男，1988 年生，研究生在读，主要研究方向：并行计算、计算机网络。

刘羽，男，1961 年生，博士，教授，主要研究方向：计算机网络、计算机应用技术。

曾胜田，男，1982 年生，研究生在读，主要研究方向：并行计算、计算机应用技术。