R高性能数据分析的基准测试II：GPU矩阵运算库

张丹丹 上海超级计算中心 上海 201203 ddzhang@ssc.net.cn

1. 概述

在大数据和数据实时分析的背景下，提高计算效率和减少计算时间成为数据分析和商业应用中的关键点。计算模型的构建以及大规模部署通常都是计算密集的任务。常规的统计任务，如数据特征分析，图形可视化和模型调参都需要大量的计算。并且，这些任务中都包含了通用的计算模式，如矩阵运算，而且他们都可以通过并行计算技术来加速。

R作为当前最流行的统计软件，具有非常多的优点，比如包含丰富的统计模型，灵活的数据处理能力，强大可视化能力。但随着数据量的日渐增大，R的内存使用方式和计算模式限制了R处理大规模数据的能力。从内存角度来看，R采用的是内存计算模式（In-Memory），被处理的数据需要预取到主存（RAM）中。其优点是计算效率和速度会很快，但缺点能处理的问题规模就非常有限（小于RAM的大小）。另一方面，R的核心（R core）是一个单线程的程序，因此利用多核机器提升性能会受到限制。然而GPGPU概念的出现，使得计算密集型的应用可通过GPU的大规模并发来加速，极大的提高了数据分析的效率。

在R中使用支持GPGPU的扩展包（package），能极大的提高R的运算性能。图1为历年应用于GPGPU环境的CRAN套件下载统计。由此可见，GPU越来越多的被R社区认可，其使用也频率也越来越高。

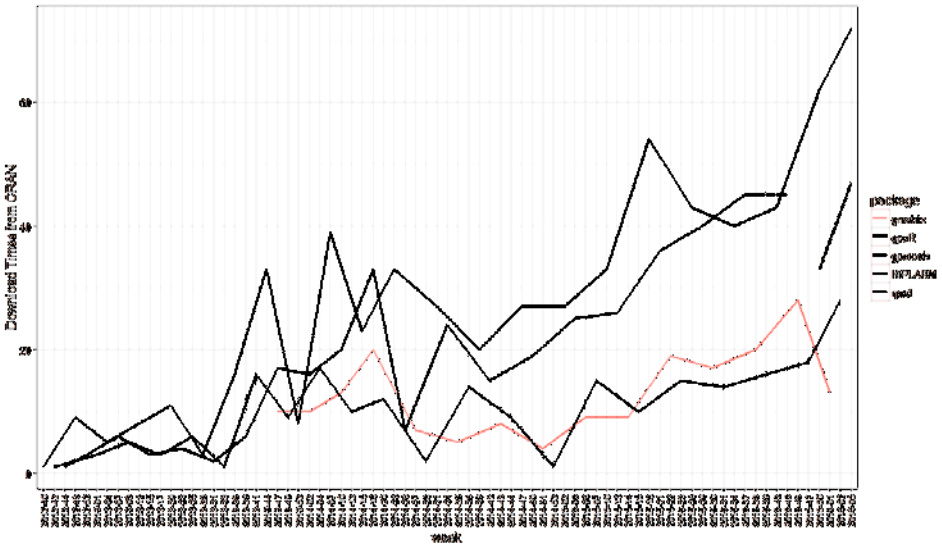

图1 历年应用于GPGPU环境的CRAN Package下载统计

本文主要集中分析了现有R社区最流行的几种GPU矩阵计算库并与NVIDIA提供的cuBLAS库进行了功能和性能的对比。本文结构如下，第二部分介绍GPU BLAS库及并行化包的基本特征，第三部分R中GPU并行化包性能测试以及分析，第四部分为结论及推荐。

2. GPU加速的矩阵计算包

矩阵运算（BLAS）是数据分析中最核心的操作之一。很多典型数据模型中的核心计算都可以用矩阵运算来表示，如推荐系统中的协同矩阵，深度学习中的卷积计算。R中矩阵以及向量的乘加等标准操作都可以在GPU上完成。

R中使用GPU最为直接的方法是通过NVIDIA提供的nvBLAS (cuBLAS)库。R扩展包中的BLAS包则进一步扩展了nvBLAS中的功能，也提供了更为丰富的软件和硬件接口，如表1所示。

表1 支持GPGPU软件包的功能对比

Function		nvblas	gputools	gmatrix	gpuR
Data Type	Integer	N	N	Y	N
	Float	N	N	Y	Y
	Double	Y	Y	Y	Y
Data Transfer	Non-synchronization mode	N	N	Y	Y
	out of cores （memory）	Y	N	N	N
Backend	Multiple GPUs	Y	N	N	N
	programming language	CUDA	CUDA	CUDA	openCL

2.1 数据类型

通过预加载nvBLAS库的方式在GPU上使用BLAS函数，GPU会依据R的默认双精度来调用cuBLAS中的双精度系列函数，如DGEMM。gmatrix和gpuR两个扩展包则可以更为灵活的选择单精度和双精度浮点计算，甚至gmatrix可以支持整数型的矩阵计算。所以，从数据类型的角度来看，R中的GPU加速库更为实用。

从硬件的角度考虑，对于GeForce系列的GPU，其主要计算能力来自于单精度的浮点运算；而双精度计算能力只有在比较高端的Tesla系列显卡上才能体现出来。因此，对于多数在个人电脑上使用GeForce系列GPU的用户来说，使用nvBALS库则无法发挥单精度的计算能力。

2.2 数据传输

当使用GPU作为协处理器来加速应用程序的时候，数据传输的开销通常会占据相当大一部分时间，并且GPU内置显存的大小也会影响到应用程序是否能够执行以及应用程序性能。对于nvBLAS来说，主机到设备的内存拷贝、GPU上的计算以及最后的设备到主机的结果拷回，都以同步方式执行，用户不能显示地控制数据的传输和计算，也就不能利用计算和通信重叠的优化方法。nvBLAS的一个主要优点是支持分块的内存拷贝方式，因此可以突破GPU显存太小的限制（out-of-cores）。gmatrix和gpuR两个扩展包则提供了异步模式的通信和计算，用户可以分离数据拷贝和真正的计算。比如，gpuR中提供的vcl*系列API，它在R中调用后会立即返回继续执行接下来的CPU命令，而GPU可以在后台完成内存的拷贝，从而实现完全的数据传输和CPU端的计算并行。

2.3 编程模型

cuBLAS，gmatrix的底层都是基于CUDA的编程模型，在NVIDIA系列的GPU上会有较好的性能，但可移植性较差。gpuR则是基于OpenCL开发的，其计算和使用更为灵活，用户程序通过OpenCL可以在异构平台无缝移植，如CPU、GPGPU、Intel Xeon Phi以及FPGA等协处理设备。

3. 性能测试及分析

3.1 测试环境

测试基于阿里云HPC平台进行，包括了安装有NVIDIA Tesla K40m的G2和Tesla M40 G4的硬件环境。

阿里云HPC是在阿里现有云服务器基础上提供的独立物理服务器+GPU加速卡，面向计算密集型应用，完全没有虚拟化开销。本次测试使用的是阿里云HPC物理机G2机型和G4机型，配置如表2和表3。

表2 测试阿里云硬件平台配置

	Ali ECS HPC G2 Server	Ali ECS HPC G4 Server
CPU	2-way Intel Xeon Ivy Bridge E5-2650 v2@2.6GHz	2-way Intel Xeon Broadwell E5-2600 v4,32 core
Memory	128GB DDR3	128GB DDR4
Disk	2TB HDD*8	1.92TB SSD*2
GPU	2* NVIDIA Tesla K40m	2*NVIDIA Tesla M40
Network	G-Ethernet	G-Ethernet
Operating System	CentOS7	CentOS7

表3 M40与K40m硬件参数列表

Specification	M40	K40m
GPU Architecture	NVIDIA Maxwell	NVIDIA Kepler
NVIDIA CUDA@ Cores	3072	2880
Single-Precision Performance	7 Teraflops with NVIDIA GPU Boost	4.29Teraflops
Double-Precision Performance	0.2 Teraflops	1.43Teraflops
GPU Memory	12 GB GDDR5	12GB GDDR5
Memory Bandwidth	288 GB/s	288GB/s
System Interface	PCI Express 3.0 x16	PCI Express 3.0 x16
Max Power Consumption	250 W	235W
Thermal Solution	Passive	Passive
Form Factor	4.4 ” H × 10.5 ” L,Dual Slot, Full Height	4.4 ” H × 10.5 ” L,Dual Slot, Full Height
Compute APIs	CUDA, DirectCompute,OpenCL, OpenACC	CUDA, DirectCompute,OpenCL, OpenACC

测试所用GPU卡K40m属于Kepler架构，M40属于Maxwell架构，M40是专为深度学习训练所设计的，其单精度浮点峰值性能达到7TFlops，核心频率达1140MHz，但双精度仅有0.2TFlops。

表4 测试所用软件及版本

Software or Package	Version
R	3.2.5
NVIDIA CUDA Toolkit	7.5
gputools	1.0
gpuR	1.1.2
gmatrix	0.3

3.2 NVIDIA Tesla K40下性能分析

首先，我们来比较各个软件包在Tesla系列显卡上的双精度计算性能。我们以nvblas的计算性能作为基准，对比了三种不同大小的矩阵运算的计算性能。在执行代码中，我们仅统计了核心API（%*%, gemm, gpuMatMult）的计算时间, 随后我们又深入分析了各个API的异同。测试代码如下，执行方式如以使用gpuR包为例为$Rscript *.R gpuR。

```
#R code
library(gpuR)
for(i in seq(1:7)) {
  ORDER = 256*(2^i)
  A = matrix(rnorm(ORDER^2), nrow=ORDER)
  B = matrix(rnorm(ORDER^2), nrow=ORDER)
  gpuA = gpuMatrix(A, type=" double " )
  gpuB = gpuMatrix(B, type=" double " )
  cputime <- system.time({gpuC = gpuA %*% gpuB})[3]
}

library(gmatrix)
for(i in seq(1:7)) {
  ORDER = 256*(2^i)
  A = gmatrix(rnorm(ORDER^2),ORDER,ORDER)
  B = gmatrix(rnorm(ORDER^2),ORDER,ORDER)
  C=gmatrix(0,ORDER,ORDER)
  cputime <- system.time({gmm(A,B,C)})[3]
}

library(gputools)
for(i in seq(1:7)) {
  ORDER = 256*(2^i)
  A = matrix(rnorm(ORDER^2), nrow=ORDER)
  B = matrix(rnorm(ORDER^2), nrow=ORDER)
  cputime <- system.time({C = gpuMatMult(A, B)})[3]
}


# nvblas, native code + PRE_LOADED
for(i in seq(1:7)) {
  ORDER = 256*(2^i)
  A = matrix(rnorm(ORDER^2), nrow=ORDER)
  B = matrix(rnorm(ORDER^2), nrow=ORDER)
  cputime <- system.time({C = A %*% B})[3]
}
```

总体来看，nvblas，gputools以及gmatrix的性能相当，因为他们后端都使用了cuBLAS数学库；而gpuR相对较低，而且不同大小下性能差距也较大。对于4096的矩阵大概只有nvblas的20%计算能力，8192的矩阵则能达到70%的计算效率。

下面我们来逐个分析各个软件包计算模式，从而理解他们的性能差异。从图2中可以看出，nvblas是唯一一个能够完成大内存（out of cores/memory）计算的。对于最大的矩阵32768， 其他几个GPU包都会显示内存溢出。在表5中我们利用nvprof分析了各个计算包内存拷贝次数，nvblas采用了矩阵分

块的方式将大矩阵分割成很多个小块，然后分别传输到GPU，再做计算。因此，其计算可以不受GPU显存大小的影响。

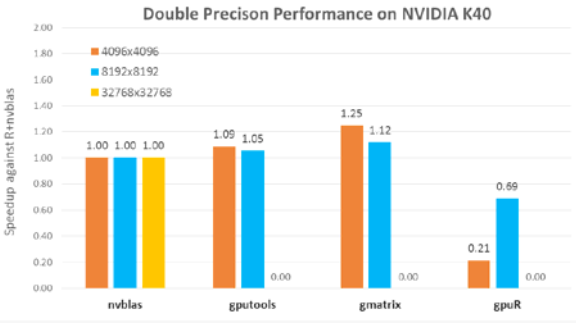

图2 K40m下各软件包随矩阵规模变化时的性能

表5 内存拷贝次数分析

Calls(size=8192)	nvblas	gputools	gmatrix	gpuR
GEMM	64	1	1	1
MEM H2D	130	3	3	2
MEM D2H	16	1	0	1

其次，gputools和gmatrix两个包都使用了cuBLAS的dgemm_sm_heavy_ldg_nn API接口一次性完成了矩阵计算，其计算效率会略高于nvblas的分块矩阵计算模式。

gmatrix的性能略高于gputools，主要原因在于gmatrix只会将A、B、C 3个矩阵拷贝到GPU并且计算完成后将C矩阵保存在GPU端，涉及3次主机到GPU端的数据传输；gputools则是将A、B矩阵拷贝到GPU，在GPU上计算完成后将C矩阵又拷贝回主机端，涉及2次主机到GPU的数据传输和1次GPU到主机的数据传输，而主机到GPU的数据传输要快于GPU到主机的传输速度，性能从而产生了差异。如表6中数据所示。

最后，我们来看gpuR的计算性能。gpuR计算性能较差的主要原因是调用的openCL矩阵计算库在NVIDIA的硬件上优化较少，如表6，其GEMM计算内核_prod_TT时间远慢于gputools和gmatrix。cublas的计算时间分别为911.4ms和912.3ms， 而openCL的计算时间为2172.5。

表6 8192*8192 矩阵乘GPU端时间开销

GPU Time Breakdown (8192)				
function (ms)		gputools	gmatrix	gpuR
GEMM	dgemm_sm_heavy_ldg_nn	911.4	912.3	
	_prod_TT			2172.5
Memory Copy	Host to Device	552.6	818.7	537.4
	Device to Host	409.5		297.6

3.3 NVIDIA M40下的性能分析

OpenBLAS，nvBlas以及gputools都使用了R默认的双精度（DP）计算模式，而M40的双精度计算能力只有0.2T。因此，可以看出在M40下GPU的计算能力只有CPU的一半。在这组测试中，我们以CPU端的OpenBLAS为性能基准。

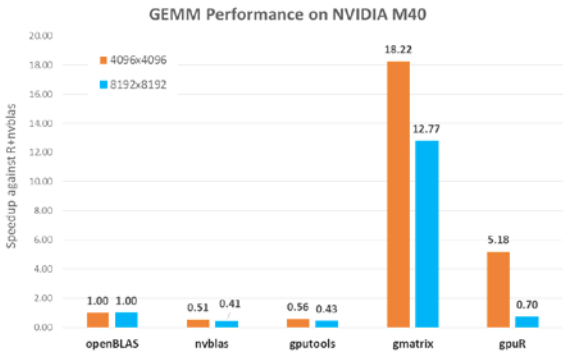

图3 M40上单精度计算模式下各Package性能

从图3可以看出，gmatrix和gpuR支持的单精度计算模式显示出来了非常优秀的性能。对于4096大小的矩阵乘，gmatrix比openBLAS快18倍，同时比nvblas则要快近37倍（18.22/0.51）。在表7中，对于矩阵大小为8196的计算任务进行了细分，可以看出单精度（SP）的时间远远小于双精度（DP）的计算时间。双精度的计算时间在6000ms左右，而单精度计算时间只有200ms左右，快了大约30倍。

```
#R code
library(gpuR)
for(i in seq(1:7)) {
  ORDER = 256*(2^i)
  A = matrix(rnorm(ORDER^2), nrow=ORDER)
  B = matrix(rnorm(ORDER^2), nrow=ORDER)
  gpuA = gpuMatrix(A, type=" float " )
  gpuB = gpuMatrix(B, type=" float " )
  cputime <- system.time({gpuC = gpuA %*% gpuB})[3]
}
library(gmatrix)
```

```
for(i in seq(1:7)) {
  ORDER = 256*(2^i)
  A = gmatrix(rnorm(ORDER^2),ORDER,ORDER,
type= " single " )
  B = gmatrix(rnorm(ORDER^2),ORDER,ORDER,
type= " single " )
  C=gmatrix(0,ORDER,ORDER, type= " single " )
  cputime <- system.time({
          gmm(A,B,C);
          h(C);
          })[3]
  }
```

gpuR对于小规模矩阵，其单精度计算能力也很好，比CPU快5倍，比nvblas快10倍。但是，当矩阵尺寸变大，计算量上升，由于其后端的OpenCL计算内核性能较弱。如表7所示，openCL的GEMM计算速度约是cuBLAS内核的两倍。

从内存使用角度来看，gpuR在CPU端就是用了单精度类型数据，而gmatrix在CPU端仍然是双精度，只是在GPU计算前进行了数据转化。由表7，可以看出gputools和gmatrix的内存传输时间相同，而gpuR里内存传输时间只有其一半。gmatrix默认情况下并没有用到MEM　D2H，为了便于和其他包比较内存传输性能，这里加入了H(C)来做完全一致的比较。

表7 8192*8192 各Package在M40上的单（双）精度性能表现

NVIDIA M40 ： 8192x8192				
time (ms)	nvblas (DP)	gputools (DP)	gmatrix (SP)	gpuR (SP)
GEMM	6572.83	6050.00	209.29	574.67
MEM H2D	739.15	155.17	153.4	77.7
MEM D2H	48.55	237.10	231.9	24.5

注：这里M40上GEMM kernel API为magma_lds128_dgemm_kernel。

3.4 异步模式下的计算性能分析

在文中分析的几个GPU包中，gpuR提供了一组异步模式的接口。对于异步接口，R程序在调用了vcl*的接口后会立即返回到R的CPU程序端，用户可以继续在CPU上执行其他任务。当用户再次显示访问和使用vcl*的返回数据时，如果计算还没有完成，R会继续等待；如果计算已经完成，则可以直接使用。所以用户可以通过CPU和GPU的任务同时并发来隐藏GPU端的通信和计算时间。

同步模式下，gmatrix在性能上最好，在图4中对比了gpuR的异步接口和gmatrix在同步计算模式下的计算时间。由图可以看出，同步模式的API时间随着计算任务增加而增加，而异步接口并不包括任何实际计算，所以它会立即返回，因此计算时间和任务大小没有关系。

gpuR单精度矩阵乘异步模式代码：

```
ibrary(gpuR)

for(i in seq(1:7)) {
ORDER = 256*(2^i)
vclA_f = vclMatrix(A, nrow = ORDER, type= " float " )
vclB_f = vclMatrix(B, nrow = ORDER, type= " float " )
  cputime <- system.time({vclC_f = vclA_f %*% vclB_
f})[3]
}
```

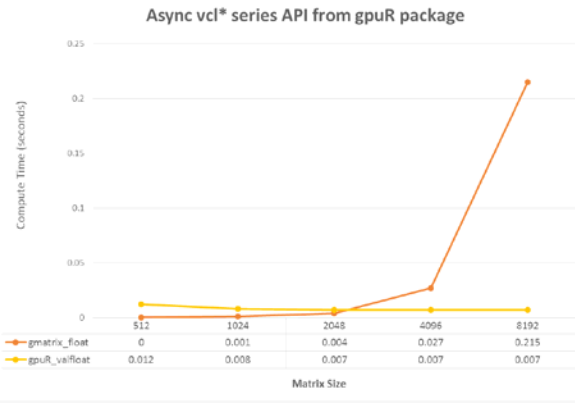

图4 gpuR异步接口和gmatrix的同步计算模式对比

4. 结论和推荐

本文通过对当前在R平台下最流行的几种GPU计算软件包的性能分析，可以看出每个软件包都有其独特之处，也各有优缺点。所以在使用时，需要根据具体的需求来选择。基于计算平台，计算模式以及使用难易程度，推荐如下：

1） nvblas适合：

- NVIDIA显卡平台
- 双精度计算
- 大内存消耗的计算，nvblas提供了很好的性能以及可扩展性；
- 初级用户

2） gputools适合：

- NVIDIA显卡平台
- 双精度计算
- 使用简单，和R的API接口功能对应性好
- 初级用户

3）gmatrix适合：

- NVIDIA显卡平台
- 单精度计算
- 多级BLAS的接口（level 1，2，3）
- 其他一些常用计算的GPU实现 (colsum，sort)
- 内存传输优化，用户需要知道内存在哪里保存
- 中级用户

4）gpuR适合：

- 异构系统，支持非NVIDIA平台，如AMD，Intel Xeon Phi，Intel GPU
- 异步计算模式，可以更好的隐藏通信时间
- 高级用户

要闻集锦

美国计划2021年完成全新架构E级超级计算机

据www.top500.org网站2016年12月10日消息报道，近日，美国能源部E级计算计划（ECP）更改了E级超级计算机的时间表，计划提前于2021年研制完成美国第一台E级超级计算机，并于9个月之后完成接收。据悉，ECP内部在经过充分讨论后，于2016年11月17日确定了新的时间表。此外，ECP还计划于2022年推出第二台E级超级计算机，2023年完成接收。

按照ECP最初的计划，美国将在2022年完成首台E级超级计算机，这一进度与其它竞争者相比，无疑是落后的。中国的目标是在2020年实现首台E级超级计算机，日本最初的计划是2019/2020年，后来宣布推迟一年，而法国也想抢在2020年完成这一目标。

除了更改时间表以外，ECP还表示，美国的首台E级超级计算机将使用从未在TOP500中出现过的、全新的技术和架构来建造。这不由得使人联想到ARM、AMD的APU，以及其它新兴的技术，例如电阻存储器（ReRAM）、硅光电子，甚至是Optalysis正在研发的光学计算机等。

不过，其结果是，这样一台全新的超级计算机很可能在一鸣惊人之后就此偃旗息鼓，再无下文。因为美国政府很可能在完全不考虑商业利益的前提下来制造首台E级超级计算机。例如当年的首台T级超级计算机“ASCI红色”，Intel公司在交付该机后就关闭了其超级计算机部门。而首台P级超级计算机“走鹃”在上线之后，其研制公司IBM就放弃了该机所使用的PowerX-Cell 8i处理器的后续研发。

尽管美国能源部和任何一家制造商都不会承认这种可能性，但事实就是如此：制造一台不受商业利益束缚的E级超级计算机会更加容易。美国能源部的计划具体如何，目前还不得而知。不过ECP将在2017年1月宣布首份FastForward计划的合同，届时这台全新的E超级计算机将采用何种硬件，或许会少露端倪。

（珂 庆）