GPU稀疏矩阵向量乘的性能模型构造

尹孟嘉[1,2] 许先斌[1] 何水兵[1] 胡 婧[1] 叶从欢[2] 张 涛[2]
(武汉大学计算机学院 武汉 430072)[1] (湖北工程学院计算机与信息科学学院 孝感 432000)[2]

摘 要 稀疏矩阵向量乘(Sparse matrix-vector multiplication,SPMV)是广泛应用于大规模线性求解系统和求解矩阵特征值等问题的基本运算,但在迭代处理过程中它也常常成为处理的瓶颈,影响算法的整体性能。对于不同形态的矩阵,选择不同的存储格式,对应的算法往往会产生较大的性能影响。通过实验分析,找到各种矩阵形态在不同存储结构下体现的性能变化特征,构建一个有效的性能度量模型,为评估稀疏矩阵运算开销、合理选择存储格式做出有效的指导。在14组CSR,COO,HYB格式和8组ELL格式的测试用例下,性能预测模型和测量之间的差异低于9%。

关键词 GPU,稀疏矩阵向量乘,性能模型
中图法分类号 TP312 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.04.040

Performance Model of Sparse Matrix Vector Multiplication on GPU

YIN Meng-jia[1,2] XU Xian-bin[1] HE Shui-bing[1] HU Jing[1] YE Cong-huan[2] ZHANG Tao[2]
(School of Computer,Wuhan University,Wuhan 430072,China)[1]
(School of Computer and Information Science,Hubei Engineering University,Xiaogan 432000,China)[2]

Abstract Sparse matrix-vector multiplication algorithm is widely used in large-scale linear system and solving matrix eigenvalue problems. It also often becomes the bottleneck of processing in iterative process and affects the whole algorithm performance. Choosing a different store format for different forms of matrix, the corresponding algorithms tend to generate large performance impact. Through experimental analysis, the variation performance characteristics were found under different storage structure, so as to build up an effective performance measurement model for assessment of sparse matrix computation overhead, and we selected reasonable storage format and made effective guidance. Based on 14 groups of CSR,COO,HYB format,8 groups of ELL format test cases, the difference between the performance prediction model and measurement is less than 9%.

Keywords GPU,Sparse matrix-vector multiplication,Performance model

1 引言

稀疏矩阵向量乘是一种广泛应用于大规模线性求解系统和求解矩阵特征值等问题[1]的基本运算,在迭代方法中其执行效率对整个处理性能的影响尤为突出。SPMV中大量的系统资源消耗在对存储器的访问上,运算器长期处于饱和状态,难以提高浮点运算吞吐量,存储器瓶颈类运算的特征明显,它的并行性特征使得利用现代多处理器平台研究并行SPMV来提高其性能成为一个可行的方向[2]。

Bell和Garland[3]提出了多种存储格式和SPMV CUDA实现内核,包括CSR,ELL,COO,HYB。CSR针对具有大量非零元素的稀疏矩阵;ELL通常适合每一行有几乎相同数量的非零元素的稀疏矩阵;HYB具有更好的性能矩阵,该矩阵中每行有少量的非零元素和许多行几乎相等,但可能会有一些不规则行有更多的非零元素;COO是最直观的存储格式,但性能比其他格式略差。不同的矩阵可能有它们自己最合适的单一的存储格式来实现最佳的性能。此外还存在一种可能性,分区矩阵时每一块都以在一个适当的格式存储,实现性能优于任何单一的存储格式。

本文通过分析,设计了一个基于profile-based的性能模型以预测CSR,ELL,COO,HYB SPMV内核执行时间。给定一个目标稀疏矩阵和存储格式,可以预测SPMV内核执行时间。性能建模包括两个阶段:1)根据GPU的体系结构特征生成基准矩阵;2)在GPU上对基准矩阵进行计算并获取执行时间,将基准矩阵的特征和执行时间作为下一个建模阶段的输入。在建模阶段,根据基准矩阵的实验结果,实例化性能模型的参数。最后,利用实例化模型来估计目标稀疏矩阵SPMV内核执行时间。本文根据硬件属性进行性能建模,生

到稿日期:2016-04-07 返修日期:2016-05-09 本文受国家自然科学基金面上项目(61572377),国家自然科学基金青年项目(61502154),湖北省教育厅项目(2016179)资助。

尹孟嘉(1979-),女,博士,副教授,CCF会员,主要研究方向为高性能计算、体系结构;许先斌(1954-),男,博士,教授,博士生导师,主要研究方向为高性能计算、体系结构;何水兵(1979-),男,博士,副教授,主要研究方向为高性能计算、体系结构;胡 婧(1983-),女,博士,讲师,主要研究方向为高性能计算、体系结构;叶从欢(1979-),男,博士,副教授,主要研究方向为高性能计算、云计算;张 涛(1980-),男,硕士,实验师,主要研究方向为高性能计算、体系结构。

成的基准矩阵和性能模型具有以下优点：1）相较传统分析模型，该模型更容易使用；2）相比并行架构模型，传统 profile-based 通常是不准确的[4]，本文的模型可以有效地获得性能 GPU 的影响。

本文使用 SPMV CUDA 开发内核 NVIDIA 和 NVIDIA Tesla C2050 性能进行建模和实验。采用 14 组测试实例进行 CSR，COO，HYB 格式评估，采用 8 组测试实例进行 ELL 格式评估。实际执行时间与使用本文构造的模型预测的时间较一致，性能预测和测量之间的差异不到 9%。针对 CSR，ELL，COO，HYB 格式，平均差异分别是 4.73%，4.68%，2.84%，5.22%。

本文第 2 节介绍相关工作；第 3 节对稀疏矩阵的存储格式与算法实现进行阐述；第 4 节提出性能建模；第 5 节进行性能建模及实现；第 6 节给出了实验测评及性能分析；最后总结全文。

2　相关工作

瓶颈问题指算法中每个浮点运算操作都需要多次访问存储器，SPMV 算法中存在瓶颈问题[5]。已有大量针对 SPMV 算法的优化工作，由于 SPMV 算法存在存储器瓶颈，因此主要从存储器层次的角度来提高计算性能[6-8]，其中大部分的优化工作集中在面向 CPU 这样的通用化的体系结构中[9]。

Bolz 等人提出第一个 SPMV CUDA 内核实现[10]。Bell 等人提出了多种存储格式和 SPMV CUDA 实现内核[3]，本文的建模方法利用其提出的方法。Vazquez 等人提出了一种新的格式（称为 ELLR-T），其在 GPU 上实现高性能[11]。文献[12]提出了两种压缩方案：1）CSR-DU，通过对列索引进行粗粒度增量编码来减少矩阵的结构性数据；2）CSR-VI，使用间接索引减少数值，但只能应用于含有少量唯一值的矩阵。文献[13]提出了一种新的稀疏矩阵存储格式来减少存储带宽。文献[14]介绍了一种新的压缩稀疏块的存储格式，以高效地并行计算稀疏矩阵和向量的乘法运算、稀疏矩阵转置、向量的乘法运算。

Xu 基于 ELL 的格式优化方法和 SpMV CUDA 的性能模型提出了 SpMV[15]。Zhang 和 Owens 采用了微基准开发吞吐量模型方法[16]，该模型的重点是识别性能瓶颈并指导程序员实现优化。本文模型侧重于预测执行时间，类似于文献[17-19]。Baghsorkhi 等人提出了一种基于编译器的 GPU 性能建模方法，使用程序分析和象征性的评价技术进行准确预测[17]。Hong 和 Kim 提出了一种简单的分析 GPU 模型来估计大规模并行程序的执行时间[18]，该模型根据运行线程的数量和内存带宽估计并行内存请求的数量。上述分析模型都是基于 GPU 的抽象体系结构，而本文模型基于分析和 profile-based 的建模技术。

3　稀疏矩阵存储格式与算法实现

稀疏矩阵具有多种存储格式，如 COO，CSR，ELLPACK 和 Hybrid 等格式。不同的矩阵有其最合适的单一存储格式来实现最佳的性能。矩阵分区时，每一块都以适当的格式存储，实现性能优于任何单一的存储格式。每种格式在存储要求、计算特点、访问和操作矩阵元素的方法等方面不同，存储格式的不同主要由矩阵的稀疏模式决定，即系数矩阵中非零元素的分布情况。可将存储格式划分为两类：

1）通用存储格式，适用于所有的稀疏矩阵，不需要考虑矩阵中非零值的分布情况。

2）非通用存储格式，需要考虑矩阵中非零值的分布情况；对于一些特殊的分布，需要专门的存储格式；该情况下，需要对稀疏矩阵进行预分析以得到适用的存储格式。

3.1　COO 格式

COO 是一种通用的易于理解与应用的存储格式，它分别采用 3 个独立数组存储：行标识、列标识和非 0 元素的值；其缺点是空间效率较低[20]。图 1 示出了一个 5 * 4 的稀疏矩阵，其 COO 存储格式如图 2 所示。

0	2	0	4	0
1	2	3	0	0
0	1	0	0	0
0	0	0	1	1

图 1　5 * 4 的稀疏矩阵

A =	2	4	1	2	3	1	1	1
Col_Idx =	1	3	0	1	2	1	3	4
Row_Idx =	0	0	1	1	1	2	3	3

图 2　COO 格式的稀疏矩阵数组

主要算法如下：

```
For(i=0;i<num_nonzeros;i++)
{y[rows[i]=data[i] * x[cols[i]];}
```

3.2　CSR 格式

CSR 格式（压缩行存储格式）[21-22]是一种使用最为广泛的存储格式，且其基于通用的目的，因此 CSR 格式也是通用存储格式。CSR 存储格式由 3 个数组构成：*A*，*Col_Idx*，*Row_Ptr*。

依照行宽方式对稀疏矩阵进行连续扫描，提取矩阵中的非零元素并保存在数组 *A* 中；将数组 *A* 中的各非零元素在原始矩阵中位置的列索引存储到数组 *Col_Idx* 之中；数组 *Row_Ptr* 中存放的是原始稀疏矩阵中每行的首个非零元素在数组 *A* 的 *Col_Idx* 中的索引[23]。设矩阵大小为 $M * N$，*Row_Ptr* 数组的长度则为 $M+1$，*Row_Ptr*[*i*] 中存储的是第 *i* 行的偏移量，最后将稀疏矩阵非零元素总个数存储在 *Row_Ptr*[*M*] 中。图 3 示出了利用 CSR 存储格式来表示原始稀疏矩阵的方法。

A =	2	4	1	2	3	1	1	1
Col_Idx =	1	3	0	1	2	1	3	4
Row_Ptr =	0	2	5	6	8			

图 3　稀疏矩阵数组的 CSR 格式存储示意

采用 CSR 格式除了可减少存储空间之外，数组 *Row_Ptr* 可更加容易和便利地对稀疏矩阵进行快速查询；利用 *Row_Ptr* 可以得到一些其他信息，例如可以计算特定行（*Row_Ptr*[*i*+1]−*Row_Ptr*[*i*]）之间的非零元素个数，主要算法如下：

```
For(i=0;i<num_rows;i++)
{start=ptr[i];end=ptr[i+1];sum=y[i];
For(jj=star;jj<end;jj++)
```

```
Sum=x[indices[jj]] * data[jj];
Y[i]=sum;}
```

3.3 ELLPACK 格式

ELLPACK 格式的构成方式与 CSR 类似，即非零元素被移动到矩阵的左侧，而零元素被移动到矩阵的右侧，如图 4 所示。

确定最大的行长 Max 之后，位于 Max 之后的所有零元素将被丢弃。按列宽方式对结果矩阵进行扫描，并将结果矩阵的所有元素存储在数组 A 中，数组 A 中非零元素在原始矩阵对应位置的列索引值存储在数组 Col_Idx 中。数组 Col_Idx 中的符号 X 不需要考虑，因为数组 A 中填充的零元素对计算来说不重要，仅仅是方便对元素进行相同寻址操作，图 4 和图 5 是原始稀疏矩阵的 2D ELLPACK 数组表示和 1D ELLPACK 数组表示。对于每行最多有 K 个非零元素的 $M*N$ 稀疏矩阵，ELLPACK 格式需要用一个 $M*K$ 的稠密数组 A 来存储，当行中非零元素的个数小于 K 时就用零元素填充。

A =

2	4	0
1	2	3
1	0	0
1	1	0

Col_Idx =

1	3	X
0	1	2
1	X	X
3	4	X

图 4 2D ELLPACK 数组

A =

2	1	1	1	4	2	0	1	0	3	0	0

Col_Idx =

1	0	1	3	3	1	X	4	X	2	X	X

图 5 1D ELLPACK 数组

主要算法如下：

```
For(n=0;n<max_ncols;n++)
{for(i=0;i<num_rows;i++)
Y[i]=data[n * num_rows+i] * x[indices[n * num_rows+i]];}
```

3.4 Hybrid 格式

Hybrid 格式由 ELLPACK 格式和 COO 格式结合而成，其旨在提高 SPMV 内核的计算性能。当矩阵每行所含的非零元素个数变化很大时，ELLPACK 格式的性能就会呈级数地锐减。相反，COO 格式基于通用目的，其计算性能不受每行所含非零元素个数的影响。在实验中，最好情况时 ELLPACK 格式的 SPMV 性能超过 COO 格式的，所以 Hybrid 格式中矩阵的大部分元素用 ELLPACK 来存储，剩下的部分用 COO 格式来存储，Hybrid 格式的稀疏矩阵数组如图 6 所示。

0	2	0	4	0
1	2	3	0	0
0	1	0	0	0
0	0	0	1	1

=

0	2	0	4	0
1	2	0	0	0
0	1	0	0	0
0	0	0	1	1

+

0	0	0	0	0
0	0	3	0	0
0	0	0	0	0
0	0	0	0	0

图 6 Hybrid 格式的稀疏矩阵数组

4 SPMV 性能建模

性能建模根据硬件属性分析生成基准矩阵和性能模型，性能建模过程如图 7 所示，先计算矩阵 strip 的大小，生成基准矩阵，然后测试基准矩阵的执行时间。找到各种矩阵形态在不同存储结构下体现的性能变化特征，从而构建一个有效的性能度量模型。CSR 和 ELL 格式计算矩阵 strip 的大小和每行非零元素的数量；COO 格式只计算矩阵 strip 的大小作为目标矩阵；HYB 格式则分别计算 ELL 部分和 COO 部分，再合起来预测目标矩阵的执行时间。最后根据基准矩阵的实验结果实例化参数化模型，使用性能模型估计 SPMV 内核执行时间并将其作为目标矩阵。

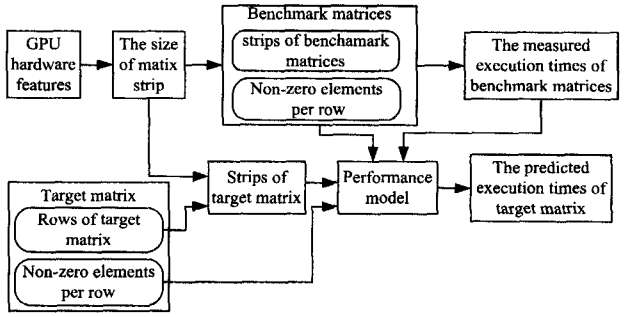

图 7 性能建模工作流程图

4.1 strip 的大小

strip 是 GPU 在线程满载情况下一次迭代中能处理的最大子矩阵。对于大型矩阵，它可能需要多次迭代来处理矩阵，因此可能包含多个 strip。矩阵的大小由 GPU 的物理特性决定。对于 SPMV，矩阵 strip 的大小计算如下[24]：

$S_{CSR}=SM$ 的数量 $*\ warps/multiprocessor$

$S_{ELL}=S_{COO}=SM$ 的数量 $*\ threads/multiprocessor$

4.2 基准矩阵

(1)行数的产生 R：$R=S\times I$

S 是一个矩阵 strip 的大小，即指 GPU 在一个迭代内处理行(CSR 和 ELL)或非零元素(COO)的最大数量；I 是基准矩阵或目标稀疏矩阵 strip 的数量。

CSR：$S=S_{CSR}$，$I\in N$

ELL：$S=S_{ELL}$，$I\in N$

COO：$S=S_{COO}$，$I\in N$

(2)列数的产生 C

列数一定要大于基准矩阵或目标稀疏矩阵每一行的非零元素的数量。由于稀疏矩阵用压缩格式进行存储，因此 C 的值不会影响性能。

(3)每行非零元素的个数 P_{NZ}

在基准矩阵中，假设每一行有相同数量的非零元素。根据最大非零元素的个数可以推断出每行非零元素的个数，这些元素依照相应的稀疏矩阵的格式存储在 GPU 全局存储器中，每个非零元素的值是随机的，G_M 是 GPU 全局内存的大小(字节数)。

$$\text{CSR}:P_{NZ}\in[1,\frac{G_M-sizeof(\text{int})\times(R+1)}{(sizeof(float)+sizeof(\text{int}))\times R})$$

$$\text{ELL}:P_{NZ}\in[1,\frac{G_M}{(sizeof(float)+sizeof(\text{int}))\times R})$$

$$\text{COO}:P_{NZ}\in[1,\frac{G_M}{(sizeof(float)+2\times sizeof(\text{int}))\times R})$$

(4)strips 的数量 I

为了获得准确的性能模型，生成一系列的基准矩阵。行数和每行中非零元素的个数决定一个基准矩阵。$R=S\times I$(S 是固定的)，根据上面的标准计算 I 和 P_{NZ} 的值，将其组合后每个组合可以表示一个基准矩阵。

CSR 和 ELL：$I=1,2,3,\cdots,10$。在实验中，最大的基准

矩阵包含 10 条 strip，这个数量足够测量性能。

COO：$I=1$。每个非零元素由一个线程处理，当在不同的基准矩阵中增加每一行的非零元素数量时，只增加 strip 次数而不增加 strip 的数量。

(5)每一行非零元素的个数(P_{NZ})

CSR 和 ELL：$P_{NZ}=4,16,\cdots,1024,2048,\cdots$

COO：$P_{NZ}=10,20,30,\cdots,100$

(6)基准矩阵的执行时间 T

删除基准矩阵中初始化延迟，执行时间如下：

$$T=\frac{\sum_{i=1}^{\beta}\phi((M_{R\times C})\times v_c)-\sum_{j=1}^{\alpha}\phi((M_{R\times C})\times v_c)}{\beta-\alpha}$$

其中，$M_{R\times C}$是基准矩阵，R,C 表示基准矩阵的维数，即行数和列数；V_C 是一个随机向量，C 表示随机向量的维数；$\phi(M\times V)$是矩阵向量乘法的执行时间，M 表示基准矩阵，V 表示随机向量；α,β是自然数，$\alpha<\beta$。

4.3 目标矩阵

(1) strips (I)的数量

给定一个目标矩阵具有 N_R 行和 N_{NZ} 个非零元素，strips 的数量可以按如下计算：

$$I_{CSR}=\left\lceil\frac{N_R}{S_{CSR}}\right\rceil$$

$$I_{ELL}=\left\lceil\frac{N_R}{S_{ELL}}\right\rceil$$

$$I_{COO}=\left\lceil\frac{N_{NZ}}{S_{COO}}\right\rceil$$

(2)每一行的非零元素的数量 P_{NZ}

在一个目标矩阵的每一行中，D 是由非零元素的个数组成的集合。D_s 是在一个目标矩阵的每一行中非零元素的数量组成的数据集。

CSR：P_{NZ}是集合 D_s 中的众数。

ELL：P_{NZ}是集合 D_s 中的最大值的集合。

5 性能建模及实现

strip 的数量即为每一行非零元素的数量，基准矩阵的执行时间具有数量之间的关系，因此可以根据这些关系估计目标矩阵中 SPMV 内核的执行时间。

5.1 CSR 内核

首先建立以下关系：当基准矩阵集中矩阵 strip 的数量相同时，对每行非零元素的数量(x)和基准矩阵的执行时间(T)建立关系 Relationship-1：$T=E(x)$，如图 8－图 10 所示。基准矩阵集中矩阵每一行的非零元素数量相同时，strip 的数量(y)和执行时间基准矩阵(T') 建立关系 Relationship-2：$T'=E'(y)$，如图 11 和图 12 所示。

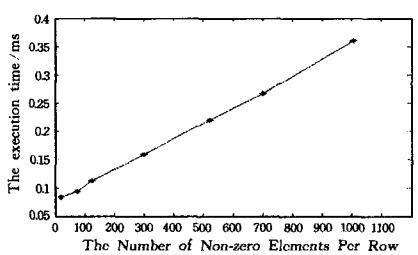

图 8　每行非零元素的个数和执行时间的关系(P_{NZ}<threshold)

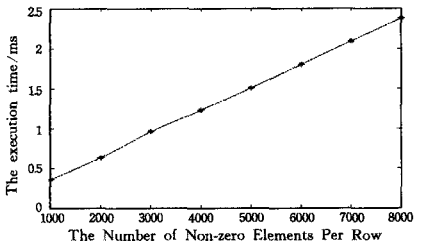

图 9　每行非零元素的个数和执行时间的关系(P_{NZ}>threshold)

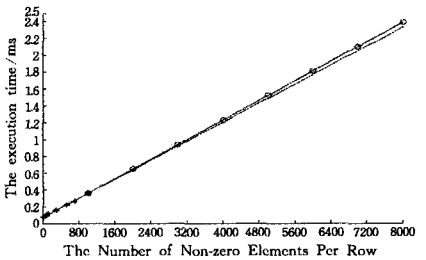

图 10　每行非零元素的个数和执行时间的关系

由图 10 可知，根据数据点拟合的曲线在 1000 附近出现交点，表明当每行非零元素个数在 1000 左右时存在一个交界的阈值，每一行的非零元素比它更小或更大时线性关系是不同的(取 1024 作为阈值)。

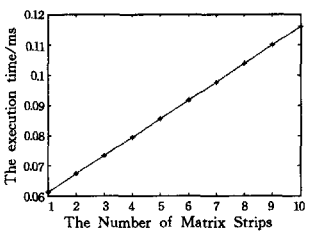

图 11　strip 的数量和执行时间的关系(P_{NZ}<threshold)

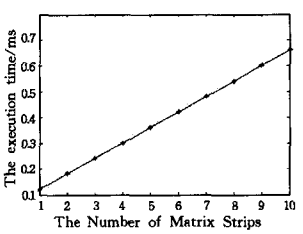

图 12　strip 的数量和执行时间的关系(P_{NZ}>threshold)

根据目标矩阵的每一行非零元素的数量 (用 x_0 表示)，从 Relationship-1 得到 T_1，$T_1=E(x_0)$。T_2 是任意已经测试的基准矩阵 M 的执行时间。根据目标矩阵的 strip 数量 (用 y_0 表示)，由 Relationship-2 得出 $T_3=E'(y_0)$，矩阵 M 每一行的非零元素的个数设定为 Relationship-2 中每行非零的数量。估计目标的执行时间矩阵 T_0，$T_0=(T_1/T_2)*T_3$。

5.2 ELL Kernel

首先建立如下关系：通过一个具有相同 strip 数量的基准矩阵集(y_1 表示 strip 数量)，建立每行非零元素的数量 x 和基准矩阵的执行时间 T 之间的关系 Relationship-1：$T=f(y_1)*x+g(y_1)$(见图 13)。

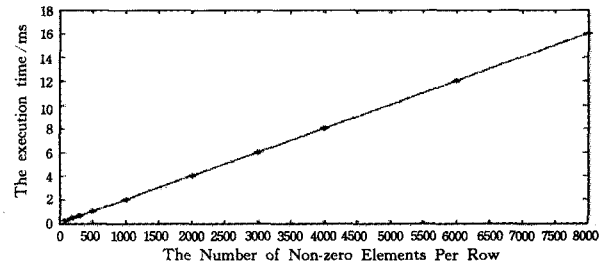

图 13　每行非零元素的个数和执行时间的关系

通过 strip 数量不同的基准矩阵集合，利用Relationship-1

建立基准矩阵strip的数量 y 与相应的线性方程组系数 f 之间的关系 Relationship-2：$f(y)$(见图 14)。设基准矩阵集合中各矩阵每一行非零元素数量相同(用 x_1 表示非零元素数量)，建立 strip 数量(y)和基准矩阵的执行时间(e)的关系 Relationship-3：$e(y)=f(y)*x_1+g(y)$(见图 15)，从而得到 $g(y)=e(y)-f(y)*x_1$。

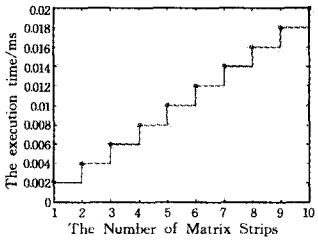

图 14 strip 的数量和线性方程系数的关系

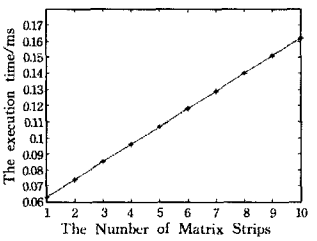

图 15 strip 的数量和执行时间的关系

给定一个目标矩阵，估算目标矩阵的执行时间 T_0，$T_0=f(Y)*X+g(Y)$，这里 X,Y 分别是目标矩阵中每行非零元素的个数和 strip 的数量。根据目标矩阵中 strip 的数量，由 Relationship-2 获得线性方程的系数 $f(Y)$，由 Relationship-3 得到 $g(Y)=e(Y)-f(Y)*Y$。

5.3 COO Kernel

首先建立 strip 的数量(x)和执行基准矩阵时间(T)的关系 Relationship-1：$T=E(x)$，如图 16 所示。

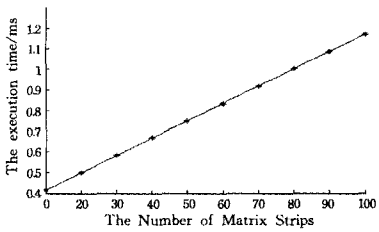

图 16 strip 的数量和执行时间的关系

计算目标矩阵中非零元素的总数，利用 Relationship-1，估计目标矩阵的执行时间 T_0，得到 $T_0=E(x_0)$。

5.4 HYB Kernel

因为 HYB 是 ELL、COO 的组合，可以利用 5.2 节和 5.3 节的关系。目标矩阵计算 HYB 的临界值分为两部分：ELL 和 COO。计算目标矩阵 COO 部分非零元素的总数量。计算目标矩阵 strip 的数量，在 ELL 中用 x_1 表示，在 COO 中用 x_2 表示。用 HYB 中 threshold 的数量 z_0 作为目标矩阵 ELL 部分每行非零元素的数量 y_1，$y_1=z_0$，估计目标矩阵 ELL 部分 $T_1=f(y_1)*x_1+g(y_1)$，COO 部分 $T_2=E(x_2)$，目标矩阵的执行时间 $T_0=T_1+T_2$。

6 实验结果及分析

本文的 GPU 运行环境为 NVIDIA，采用了 Windows 7 操作系统，使用 Intel(R) Core(TM) i7 920 CPU，开发环境为 VS 2010 IDE，2 块显卡中 NVIDIA Geforce 9600 用来显示，NVIDIA Tesla C2050 用来计算。使用 NIVIDA CUDA 的编译器 NVCC 将 CUDA 内行代码编译，生成执行在 GPU 上的设备代码。测试的稀疏矩阵来自于 Timothy Davis 的稀疏矩阵集[25]，这些稀疏矩阵来自于各种不同的实际应用，包括经济学、流行病学、有限元分析模型、蛋白质、网络结构等，也经常出现在其他科研工作中[26-28]。数据的规模如表 1 所列。

表 1 测试矩阵集

矩阵	大小	非零元素个数	平均每行非零元素个数
Protein	36k×36k	4.3M	119
FEM/Spheres	83k×83k	6.0M	72
FEM/Cantilever	62k×62k	4.0M	65
Economics	207k×207k	1.27 M	6
Epidemiology	526k×526k	2.1M	4
FEM/Accelerator	121k×121k	2.62M	22
Webbase	1M×1M	3.1M	3
Dense	2k×2k	4.0M	2000
FEM/Harbor	47k×47k	2.37M	50
QCD	49k×49k	1.90M	39
FEM/ship	141k×141k	3.98M	28
Wind Tunnel	218k×218k	11.6M	53
Circuit	171k×171k	959k	6
LP	4k×1.1M	11.3M	2825

性能建模的准确性至关重要，因为它是准确报告最优解和性能优化的基础，本文对其执行时间和性能差异率进行评价。CRS，COO，HYB 格式使用 14 组测试实例，ELL 格式使用 8 组测试实例，在 NVIDIA Tesla C2050 上运行。

图 17－图 20 示出了 CSR，ELL，HYB，COO 格式的 SpMV 性能模型预测时间和实际测量时间。

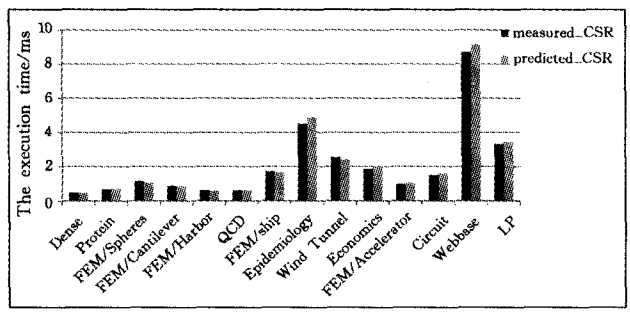

图 17 CSR 在性能模型的预测时间

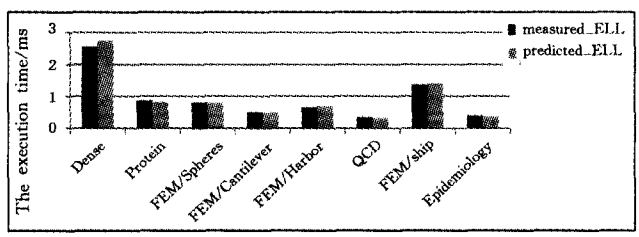

图 18 ELL 在性能模型的预测时间

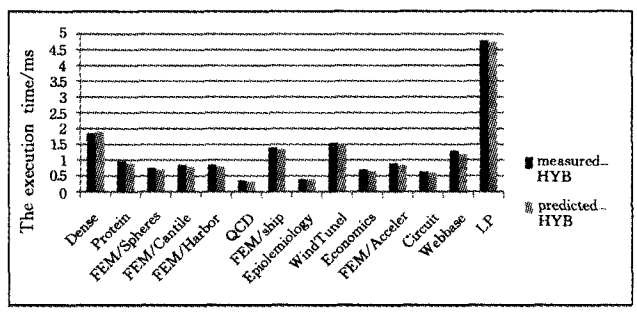

图 19 HYB 在性能模型的预测时间

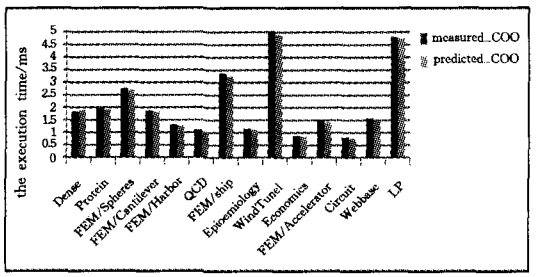

图 20　COO 在性能模型的预测时间

图 21 示出了绝对性能差异率百分比。有 14 组测试实例，包括 CSR，COO，HYB 格式，用 8 组测试实例评估 ELL 格式。实际执行时间与使用本文构造的模型预测的时间较一致，性能预测和测量之间的差异不到 9%。对 CSR，ELL，COO，HYB 格式的平均差异分别是 4.73%，4.68%，2.84%，5.22%。

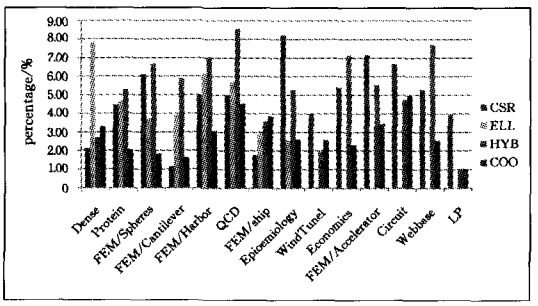

图 21　绝对性能差异率百分比

结束语　度量不同形态的矩阵在各种存储结构下乘法运算的时间效率，是大数据平台规划中一个重要的观测指标，同时也是构造此类系统的重要指标。本文针对 CSR，ELL，COO，HYB 4 种格式，通过大量测试数据集合实际运行得到的观测数据，找到运行时间、数据集合形态以及存储模型之间的关系，并建立起它们的关系模型函数，分别提出性能模型、预测目标矩阵的执行时间。通过实践测试对比，使用模型函数预测的测试数据集合的执行时间与实际运行时间之间的误差在稳定的有限偏差阈值之内，证明了该模型具有一定的实用性。

参 考 文 献

[1] SHERESHEVSKY M, CUKIC B, CROWEL J, et al. Software Aging and Multifractality of Memory Resources[C]// Proceedings of DSN 2003. Los Alamitos, USA: IEEE Computer Society, 2003: 721-730.

[2] BAI H T, OUYANG D T, LI X M, et al. Optimizing Sparse Matrix-vecotr Multiplication Based on GPU[J]. Computer Science, 2010, 37(8): 168-171, 181. (in Chinese)
白洪涛，欧阳丹彤，李熙铭，等. 基于 GPU 的稀疏矩阵向量乘优化[J]. 计算机科学，2010，37(8)：168-171，181.

[3] BELL N, GARLAND M. Implementing Sparse Matrix-Vector Multiplication On Throughput-Oriented Processors[C]// Proceedings of the Conference on High Performance Computing Networking. New York: Association for Computing Machinery, 2009: 1-11.

[4] RESIOS A. GPU Performance Prediction Using Parametrized Models[D]. Utrecht: Utrecht University, 2011.

[5] VÁZQUEZ F, GARZON E M, FEMANDEZ J J. A matrix approach to grahpic reconstruction and its implementation on GPUs[J]. Journal of Structural Biology, 2010, 170(1): 146-151.

[6] BIK A J C, WIJSHOFF H A G. Automatic data structure selection and transformation for sparse matrix computations[J]. IEEE Transactions on Parallel and Distributed Systems, 1996, 7(2): 109-126.

[7] IM E J, YELICK K. Optimizing Sparse Matrix-Vector Multiplication on SMPs[EB/OL]. http://www.cs.berkeley.edu/yelick/ejim/ppsc99.ps.

[8] LEE BENJIAMIN C, VUDUC RICHARD W, YELICK D J W, et al. Performance Model for Evaluation and Automatic Tuning of Symmetric Sparse Matrix-Vector Multiply[C]// 2004 International Conference on Parallel Processing. United States: IEEE Computer Society, 2004: 169-174.

[9] FATAHAIAN K, SUGERMAN J, HANRAHAN P. Understanding the efficiency of GPU algorithms for matrix-matrix multiplications[C]// Proceedings of the SIGGRAPH/Eurographics Workshop on Graphics Hardware. New York: Association for Computing Machinery, 2004: 133-138.

[10] BOLZ J, FARMER I, GRINSPUN E, et al. Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid[J]. ACM Transactions on Graphics, 2003, 22(3): 917-924.

[11] VÁZQUEZ F, ORTEGA G, FERNANDEZ J J, et al. Improving the Performance of the Sparse Matrix Vector Product with GPUs[C]// 10th IEEE International Conference on Computer and Information Technology. Bradford, United States: IEEE Computer Society, 2010: 1146-1151.

[12] KORNILIOS K, GEORGIOS G, NECTARIOS K. Improving the performance of multithreaded sparse matrix-vector multiplication using index and value compression[C]// Proceedings of the International Conference on Parallel Processing. United States: Institute of Electrical and Electronics Engineers, 2008: 511-519.

[13] MONAKOV A, AVETISYAN A. Implementing Blocked Sparse Matrix Vector Multiplication on NVIDIA GPUs[C]// 9th International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation. Germany: Springer Verlag, 2009, 5657: 289-297.

[14] BULUC A, FINEMAN J T, MATTEO F, et al. Parallel Sparse Matrix-Vector and Matrix-Transpose-Vector Multiplication Using Compressed Sparse Blocks[C]// Proceedings of the Twenty-first Annual Symposium on Parallelism in Algorithms and Architectures. New York: Association for Computing Machinery, 2009: 233-244.

[15] XU S M, XUE W, LIN H X. Performance Modeling and Optimization of Sparse Matrix-Vector Multiplication on NVIDIA CUDA Platform[J]. Journal of Supercomputing, 2013, 63(3): 710-721.

[16] ZHANG Y, OWENS J D. A Quantitative Performance Analysis Model for GPU Architectures[C]// International Symposium on High-Performance Computer Architecture. United states: IEEE Computer Society, 2011: 382-393.

（下转第 206 页）

[16] LIU Y,DONG Y,LI Z H. Discussion of data security about space-earth all-in-one networking and analysis of CCSDS SCPS [J]. Chinese Space Science and Technology,2004,24(1):31-36. (in Chinese)
刘泳,董勇,李泽慧. 空间通信协议分析与一体化网络安全问题探讨[J]. 中国空间科学技术,2004,24(1):31-36.

[17] LI J Y. The design and implementation of SCPS-SP/IPSec protocol conversion[D]. Harbin: Heilongjiang University,2013. (in Chinese)
李敬媛. SCPS-SP/IPSec 协议转换的方案设计与实现[D]. 哈尔滨:黑龙江大学,2013.

[18] DUAN X F,AN H Z,XIE S M. Space communication security protocols[J]. Communications Technology,2009,42(12):101-103. (in Chinese)
段小芳,安红章,谢上明. 空间通信安全协议研究[J]. 通信技术,2009,42(12):101-103.

[19] CCSDS. Encryption Algorithm Trade Survey: CCSDS. 350. 2-G-1 [S]. 2008.

[20] YANG Y H. Research on the authenticated encryption technology in CCSDS[D]. Shenyang: Shenyang Aerospace University,2011. (in Chinese)
杨亚辉. CCSDS 认证加密技术研究[D]. 沈阳:沈阳航空航天大学,2011.

[21] MCGREW D,VIEGA J. The security and performance of the galois/counter mode (GCM) of operation[J]. Lecture Notes in Computer Science,2004,3348:343-355.

[22] LI H X. Research on data encryption/decryption strategy of CCSDS space communication system[D]. Shenyang: Shenyang Ligong University,2011. (in Chinese)
李海霞. CCSDS 空间通信系统中数据加密/解密策略研究[D]. 沈阳:沈阳理工大学,2011.

[23] LIAO Y,GUO X Q. A design method of multi-level security and anti-replay function of SCPS-SP[P]. 2015. (in Chinese)
廖勇,郭修琼. 一种 SCPS-SP 多安全等级及抗重放功能的设计方法[P]. 2015.

[24] LI H,FAN X X,MI J N,et al. Analysis of security technologies in integrated space-air-ground networks[J]. Journal of CAEIT,2014,9(6):592-597. (in Chinese)
李华,范鑫鑫,秘建宁,等. 空天地一体化网络安全防护技术分析[J]. 中国电子科学研究院学报,2014,9(6):592-597.

[25] CCSDS. Next generation space internet (NGSI)-end-to-end security for space mission communications: CCSDS 733. 5-O-1[S]. 2003.

[26] LIANG L,LU S W. Design and implementation of gateway between CCSDS standard and TCP/IP protocols[J]. Computer Engineering,2006,32(4):134-136. (in Chinese)
梁莉,鲁士文. CCSDS 标准和 TCP/护协议间的网关设计与实现[J]. 计算机工程,2006,32(4):134-136.

(上接第 187 页)

[17] SARA S B,MATTHIEU D,PATEL S J,et al. An Adaptive Performance Modeling Tool for GPU Architectures[C]// Proceedings of the 2010 ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York: Association for Computing Machinery,2010:105-114.

[18] HONG S,KIM H. An Analytical Model for a GPU Architecture with Memory-Level and Thread-Level Parallelism Awareness [C]// International Symposium on Computer Architecture. United States: Institute of Electrical and Electronics Engineers,2009:152-163.

[19] KISHORE K,RISHABH M,REHMAN S M,et al. A Performance Prediction Model for the CUDA GPGPU Platform[C]// 16th International Conference on High Performance Computing. United States: IEEE Computer Society,2009:463-472.

[20] LIANG T. The Research on Optimizing Sparse Matrix Computation Based on GPU[D]. Wuhan: Huazhong University of Science and Technology,2012. (in Chinese)
梁添. 基于 GPU 的稀疏矩阵运算优化研究[D]. 武汉:华中科技大学,2012.

[21] BARRETT R,et al. Templates for the solution of Linear Systems: Building blocks for Iterative Methods[R]. Philadelphia: SIAM Press,1994.

[22] RUKHSANA S,ANILA U,CHUGTAI I R. Implementation and Evaluation of Parallel Sparse Matrix-Vector Products on Distributed Memory Parallel Computers[C]// IEEE International Conference on Cluster Computing. United States: Institute of Electrical and Electronics Engineers,2006.

[23] WANG Z W,CHENG L L,ZHAO W Q. Parallel Computer Performance Analysis Model Based on GPU[J]. Computer Science,2014,41(1):31-38. (in Chinese)
王卓薇,程良伦,赵武清. 基于 GPU 的并行计算性能分析模型[J]. 计算机科学,2014,41(1):31-38.

[24] PING G,WANG L Q,CHEN P. A Performance Modeling and Optimization Analysis Tool for Sparse Matrix-Vector Multiplication on GPUs[J]. IEEE Transactions on Parallel And Distributed Systems,2014,25(5):1112-1123.

[25] DAVIS TIMOTHY A,HU Y F. The University of Florida Sparse Matrix Collection[J]. ACM Transactions on Mathematical Software,2011,38(1):1-28.

[26] SAMUEL W,LEONID O,RICHARD V,et al. Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms[J]. Parallel Computing,2009,35(3):178-194.

[27] VASILEIOS K,GOUMAS G,NECTARIOS K. Performance Models for Blocked Sparse Matrix-vector Multiplication Kernels[C]// 2009 International Conference on Parallel Processing. United States: IEEE,2009:356-364.

[28] FENG X W. Research on Key Technology of Similarity Calculation based on GPU[D]. Wuhan: Huazhong University of Science and Technology,2014. (in Chinese)
冯晓文. 基于 GPU 的相似度计算关键技术研究[D]. 武汉:华中科技大学,2014.

[29] CHEN R X. Parallelized XML Query Based on Functional Intermediate Language[J]. Journal of Chongqing Uiversity of Technology(Natural Science),2011,25(7):81-86. (in Chinese)
陈荣鑫. 基于函数式中间语言的 XML 查询并行化[J]. 重庆理工大学学报(自然科学),2011,25(7):81-86.