**基于GPU的PostgreSQL扩展，用于可扩展的高吞吐量模式匹配**

## 摘要

许多领域需要大规模模式匹配以实现各种计算目标。在这里，我们提出了新的图形处理单元（GPU）扩展，以促进PostgreSQL数据库中的高吞吐量模式匹配。我们开发了一个扩展框架，使用流处理设计执行大型模式数据集的数据块处理，从而产生全局k-最近邻居匹配。该框架专门用于支持数据库环境中GPU上的模式匹配。这种方法避免了将整个数据集存储到GPU硬件上的必要性，这有利于模式数据库的显著扩展。这为结合或利用辅助（元）数据提供了巨大的潜力，作为模式匹配过程的一部分，以及将结果流水线化为传统的关系代数表达式。通过将模式匹配结果流水线化为关系表达式，可以利用数据库的功能来基于查询模式和结果之间的各种参数化相关性来构建结果集。在这项初步工作中，我们将基于GPU的高吞吐量p-norm度量函数集成到数据库服务器中。这允许人们设计异构数据处理技术，将大规模基于内容的图像检索（CBIR）与数据库的传统数据处理能力（例如关系，空间或文本搜索）相结合。我们提供各种模式大小和度量组合的时间特征，以及解决数据库和GPU参数化的平衡问题。我们的特征向量数据集在数据库表存储大小中的范围为18到85 GB，达到1亿个128维向量。我们能够从数据库中有效地执行全局前k个搜索。

关键词：模式匹配，异构数据，图形处理单元（GPU），PostgreSQL，高通量计算（HTC），高性能计算（HPC）

## 概述

许多领域需要大规模模式匹配以实现各种计算目标。匹配模式的细节变化很大，从索引到修剪到强力匹配。可扩展性通常是一个问题，因为内容和模式数据库越来越大，变成了TB级。这种增长受许多因素驱动，例如增加的数据库成员（例如，图像数量）以及越来越多的测量和测量编码大小。

大规模模式匹配问题的高性能解决方案通常利用分布式系统或计算集群和网络编程。这些通常是复合硬件 - 软件解决方案，它们利用分布式硬件，网络技术和软件的组合来实现健壮性，可扩展的模式匹配。 MapReduce架构（例如，Hadoop）是实现可伸缩模式匹配的常用方法。在[1]和[2]中，Hadoop有助于搜索大型时间序列集。在[3]中使用Hadoop匹配字符串模式和正则表达式。但是，诸如此类的分布式方法会增加网络和管理减少的开销。另外，这些方法通常依赖于唯一定制的索引或分区数据结构以促进可伸缩模式匹配。但是，在考虑与其他数据或元数据集成时，依赖于定制数据结构和分布式平台存在局限性。例如，难以利用任何性质的图像元数据来减少这种设计中的基于内容的图像检索（CBIR）搜索空间。

上述方法通常是针对大数据集的模式匹配所必需的，以克服单个机器的处理器，存储和/或存储器限制。通常，真正的限制是可扩展性要求，在不显着增加硬件资源的情况下，特定解决方案无法实现。与定制的分布式体系结构或数据结构相比，开源数据库管理系统（例如PostgreSQL）已经支持大型数据集和异构限制标准的使用，例如，具有空间约束的关系属性过滤。此外，PostgreSQL自然可以作为数据存储和处理框架进行扩展。然而，数据库管理系统传统上不配备高吞吐量数值计算，例如大规模模式匹配所需的数据计算。

图形处理单元（GPGPU）上的通用编程代表了用于模式匹配问题的有前途的复合硬件-软件解决方案。已经利用GPGPU编程来解决与模式匹配和检索相关的各种问题，例如[4] - [8]。各种研究工作，如[9] -[11]，列举了一些，利用GPU硬件执行描述符匹配，这是CBIR应用程序中的常见任务。 GPGPU技术已被用于密切相关的领域，如分类[12]和字符串匹配[13]。不幸的是，许多现有GPU驱动的模式匹配方法的一个共同缺点是需要将整个数据库物化到GPU上。

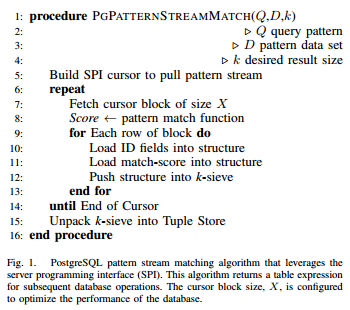
在[14]中，作者强调需要使用快速持久性内存来增强GPU内存，例如现代固态硬盘（SSD）。随着模式数据库大小的增长超过GPU内存的增长，这越来越有必要。同样，[15]的作者看到了能够在GPU上处理大量数据的重要性。他们研究内核拼接和调度，以说明使用较小数据块和多次调用GPU内核的重要性。使用这种方法，作者比一次复制所有数据然后使用单个内核调用计算增加了7-23%。类似的经验激发了我们的方法，从而我们解决了模式匹配数据集的可扩展性以及吞吐量的最大化。

在本文中，我们为PostgreSQL数据库提供了基于GPU的新型流处理扩展，这有助于高吞吐量模式匹配以生成全局k-最近邻居匹配。该框架旨在从数据库环境中支持GPU上的模式匹配，其中结果是表表达式，其可以用于随后的关系代数运算中。此外，我们使用数据库的本机存储工具来提供模式数据，从而提供可扩展性并与相关数据集成。这使我们能够在高吞吐量模式匹配之前对数据应用关系或其他约束，即将表表达式传递到模式匹配框架中。

本文的其余部分安排如下。第二部分概述了我们对PostgreSQL数据库的扩展，以支持高吞吐量模式匹配。然后，第三部分提供了在这个高吞吐量框架内利用GPGPU的工作细节。我们在第四部分中详细介绍了基于度量，模式和GPU内核参数化特性的时间初步实验。本节还提供了初始CBIR数据集的实验。最后，第五部分提供了一些结论性意见和对未来工作和扩展的讨论。

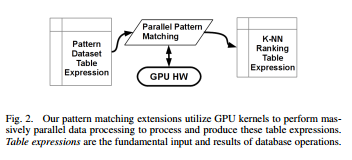
## POSTGRESQL的模式流匹配框架

我们最初的目标是开发一种通用算法，使用服务器编程接口（SPI）在PostgreSQL环境中实现模式块的流式处理。SPI使用对数据库内部的非常低级别的访问来促进PostgreSQL服务器的扩展。图1提供了用于处理模式流的算法。该算法内置于共享对象库中，并通过使用标准C语言PostgreSQL扩展函数的桥接函数链接到数据库中。将流中的每个模式与查询模式进行比较，然后将得到的距离与源元组的其他字段组合以形成结果表的潜在行。

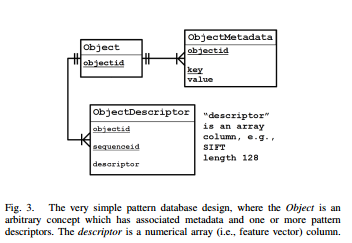


我们设计的一个关键功能是在处理模式流时生成全局k近邻。这是通过实现结果收集结构来实现的，我们称之为k-sieve。然而，物理筛子处理沉积物流，仅保留最大的成员，我们的k-sieve处理模式流，仅保留k个最佳模式匹配。 k-sieve是基于最大堆的数据结构，旨在从近线性时间复杂度O（k + n log（k））中收集来自集合的k个最佳匹配，其中n是比较的模式的数量。我们使用一组中的前k个元素来填充k-sieve，然后将每个元素与堆的顶部进行比较。通过单次比较拒绝这些候选者的能力允许我们立即忽略大于我们集合的最大值的所有值并接受来自任意数量的输入组的元素；原子或其他。

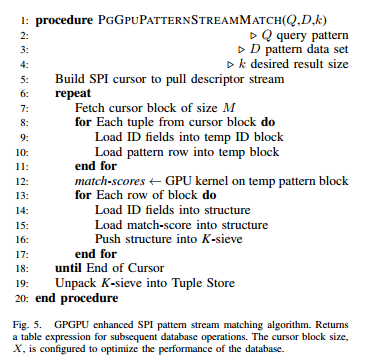
k-sieve是一种非常有效的工具，用于从一组或多组数据中收集最佳匹配的子集。k-sieve的这个属性可以有效地利用SPI通过游标从PostgreSQL中获取数据块。来自游标的每个模式块可以通过k筛单独处理，而不会破坏筛的当前内容物的完整性。最终，k-sieve收集的最后一组k项将被解压缩为一个排序数组，形成我们生成的表表达式。



我们的模式集存储在一个简单的数据库结构中，如图3所示。在该模型中，每个对象都与一个或多个描述符模式相关联。这些描述符作为数组列本地存储在数据库中。该阵列存储简化了模式流处理。此设计允许使用数据库本机的标准连接和限制操作来减少模式流。这些操作可用于通过对象元数据表显示的执行基于属性的过滤，以及使用可添加到此设计并包含更多高级数据的任意数量的附加表。例如，如果对象具有关联的空间属性，则可以利用PostGIS扩展和空间索引来过滤模式流。以类似的方式，可以利用其他现有扩展，例如文本搜索能力或基于内容的检索技术。



我们的技术允许在模式匹配之前和之后设计异构数据处理技术。作为输入表表达式的基本示例，请考虑表视图;可以使用任意数量的相关数据（属性，空间，连接等）来生成输入表表达式。由于输出也是表表达式，因此它自然适合标准的关系代数模型（限制，连接，聚合等）。该概念在图2中示出，其中模式匹配的结果可以与数据库的传统数据处理能力一起使用，例如关系，空间或文本搜索。

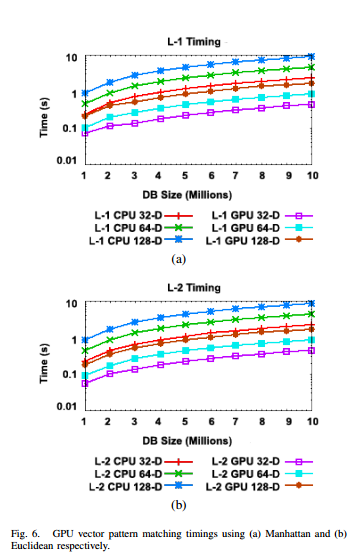


## POSTGRESQL中的GPGPU模式匹配

现在我们已经概述了扩展PostgreSQL数据库以进行高吞吐量模式处理的算法，我们将重点转移到将高性能计算（HPC）技术整合到此扩展中。目标是在数据库环境中处理模式时，通过GPGPU技术实现模式匹配加速。使用度量函数将数百万个模式特征向量与查询模式匹配自然很适合在GPU上进行处理。 GPU具有数百个核心，允许它同时度量数百万个特征向量的查询模式。与必须遍历每个特征向量并与查询模式进行比较的CPU相比，GPU只需要少量的时间和资源来匹配模式。图6显示了GPU在CPU上实现的模式匹配的数量级加速。

我们使用CUDA [16]作为我们的GPGPU框架，用于开发大规模并行模式匹配数据库扩展模块。在CUDA的上下文中，可以应用各种逻辑并行化策略，通过组织内核网格来划分数据进行处理。内核函数是要在GPU硬件上并行计算的工作单元。在这方面，解空间被映射到由处理块组成的逻辑网格。单个块由一组线程组成，其中每个线程将使用内核函数调用以处理一个数据。现代GPU通常将同时处理可用计算单元上的多个块。因此，我们使用数千个块来处理数百个模式，以有效地并行化网格中数百万个模式的匹配。

为了将GPGPU解决方案集成到我们的模式流匹配中，我们扩展了图1的算法，以合并适合GPU内核函数的模式块的生成。图4（b）提供了我们如何将特定模式块（参见图5，步骤12）映射到CUDA网格上的概念描述。我们定义每块的线程（TPB），（见图7），因此定义网格中的块数。在我们当前的实现中，TPB定义了每个GPU逻辑块的模式数量。注意，GPU逻辑块比SPI游标模式块小得多，实际上模式块是CUDA网格下面的数据。将每个大模式块推送到GPU设备，然后启动内核，之后将结果（即，度量距离）复制回来。结果度量与其他行数据合并以形成结果元组，其被传递到k-sieve，如第2部分所述。这些结果元组对于将数据库管理系统中的模式匹配与异构数据处理（例如关系或空间操作）集成是至关重要的。图5说明了这种算法，其中SPI游标用于构造传递给GPU以进行模式匹配的模式块。



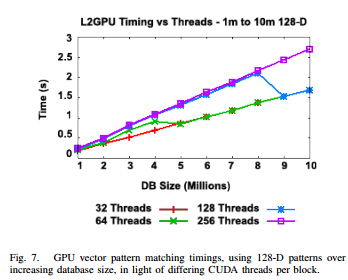
我们方法的关键新颖之一是我们避免了将整个模式数据库存储在GPU硬件上的必要性。图4（a）说明了数据库的组织，动态链接的共享对象和CUDA运行时。正如第2部分中所讨论的，模式匹配过程的结果是数据库环境中的表表达式。

## 实验

我们通过验证结合GPU驱动的HPC技术是一种明智的方法来开始我们的实验。如第一部分所述，对GPGPU的大量研究实例化了GPU上的整个数据集。然而，对于真正大规模的模式数据集来说，这显然不是一种实用的，经济有效的解决方案。如第3部分所述，我们的扩展依赖于将数据从本机数据库表复制到GPU硬件上，然后将结果复制回来。因此，我们使用多个度量函数，不同的特征向量维度和增加的数据库大小进行了时间实验。

### 基线GPU时间

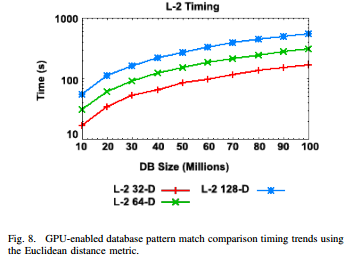
图6显示了在增加大小和不同向量长度的数据集上计算的基于CPU和GPU的p范数度量函数的时间趋势比较。显然，CPU与GPU相比，时间增长速度要快得多。在每个图中，数据集大小从100万到1000万个向量。为了进行比较，我们展示了32-D，64-D和128-D特征向量的时间，每个块使用32个线程用于GPU内核功能。图6（a）显示了计算曼哈顿距离的趋势，（b）显示了欧几里德。图7提供了相对于GPU内核功能的另一个关键参数（即TPB）的性能。在该图中，使用128-D模式，数据库大小从100万增加到1000万。我们在这些结果中包含了GPU主机设备数据和结果传输时间，因为这是处理数据库模式块流的关键组件。我们可以看到，即使考虑到GPU硬件上的数据传输，基于GPU的匹配也比CPU快一个数量级（注意y轴是对数级）。

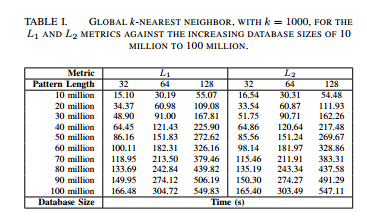


### 支持GPU的PostgreSQL时间

应当注意，来自数据库内的匹配模式的定时测量使用比第IV-A部分实验大一个数量级的数据集大小，因为我们想要处理简单地无法实现到GPU上的数据集。此外，从数据库中提取数据还有额外的成本，即从磁盘读取数据和从本机元组中解压缩。存在许多数据库调优技术可用于加速模式数据访问（在我们的实验案例中进行表扫描），但这些讨论超出了本文的范围，留待将来工作。

我们使用32维，64维和128维浮点特征向量进行了定时实验。我们在数据库中的时间安排以1000万的模式数据集开始，并以1000万的增量增加到1亿。使用的GPU硬件是Nvidia Tesla C2075 GPU协同处理器，内核使用CUDA Toolkit 5开发。我们的数据库管理系统是x86 64 Linux上的PostgreSQL版本9.2.4。数据库存储表空间位于RAID 5文件系统上。每个X维特征集的数据库表大小分别为18GB，36GB和54GB;这远远超过任何当前的GPU硬件设备存储容量。数据库扩展使用的模式块大小是PostgreSQL的内存参数配置和描述符大小的一个因素。对于我们的时间实验，我们将模式块固定为150万个模式以适应所有模式大小，并在测试系统的其他参数期间提供一致性。 PostgreSQL配置了适度的内存限制，即2GB共享缓冲区，24MB工作内存和其他默认设置。在测量变量数据库和描述符大小的实验中，我们使用每块32个线程的CUDA块大小。在这些实验中，我们能够计算全局k-最近邻居，对于32-D模式，对于大小为1亿的数据库，k = 1000，平均为166.5秒。表I提供了我们测试模式数据库大小从1000万到1亿的完整时序统计。正如预期的那样，由于数据库明显大于所需数量的顶部模式匹配结果k，因此在线性时间内找到全局最近邻。





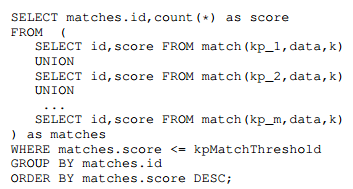
### 与SQL匹配的图像

此外，我们使用图像描述符进行了初步实验。我们已经开始从[17]提供的图像数据集构建图像检索数据库。使用公开的HessianAffine区域检测器和SIFT关键点描述符处理图像[18]。为了处理数据集，我们提取了通过Hessian-Affine区域检测器找到的关键点的SIFT特征描述符。每个图像生成一个输出数据文件，每行一个关键点。我们解析了数据文件，创建了对象记录，然后加载了Object Descriptor表（参见图3）。将数据加载到Object Descriptor表后，可以使用我们支持GPU的模式流匹配扩展来立即搜索数据。正如预期的那样，关键点的数量在不同图像之间存在显着差异，加载了98103个图像，每个图像的关键点数量从1到8686不等。生成的数据库有超过8500万个SIFT关键点。对象描述符表大约为50 GB。

如果查询图像具有m个关键点，我们可以定义关系代数表达式以将数据库中的图像排序为



其中kpi是关键点的特征向量，data是模式数据集表表达式，k是每个要插入结果的关键点的匹配数，而match的结果是表表达式。ObjectId是唯一的图像标识符。使用SQL，这可以表示如下：



在此查询中，最近邻居的计数k和关键点匹配质量阈值kpMatchThreshold是可调参数。我们承认这种方法不是CBIR关键点特征的最新技术使用（例如，可视代码书等）。我们只是提供了相关应用程序域中GPU硬件和PostgreSQL之间集成的说明性示例。

GPU硬件和PostgreSQL数据库的这种新颖集成代表了支持大规模模式匹配的现有架构的重要替代方案。这些技术将使研究人员能够充分利用异构数据，将模式匹配的结果与更传统的数据处理算法融合在一起。我们看到在各种模式识别领域中利用此研究的重要机会，例如基于内容的检索，生物识别，图像和视频分析以及其他信号处理。

## 结论

在本文中，我们将基于GPU的高吞吐量p-norm度量函数集成到PostgreSQL数据库服务器中。我们介绍了使用支持GPU的模式匹配扩展PostgreSQL后端的技术。这有助于使用GPU硬件进行大规模模式匹配，因为数据集大小不限于可以存储在GPU上的数据集。另外，这允许人们设计异构数据处理技术，其将大规模模式匹配与数据库的传统数据处理能力（例如关系，空间或文本搜索）相结合。这对于未来在各种领域的工作具有巨大的潜力，包括CBIR，生物识别和大型数据集分析，仅举几例。

我们的扩展提供模式匹配结果作为数据库环境中的表表达式，允许生成的元组考虑到传统的关系代数表达式。这固有地利用了关系数据库引擎的强大功能，使用数据库的传统SQL将信息关联到多个表中。我们已经为实现了p-norm度量函数，即曼哈顿和欧几里德。

在未来的工作中，我们将进行额外的计时和数据库调优实验。我们将确定数据库的最佳内存设置，以及这些设置如何影响用于为各种模式大小构建GPU扩展的参数。我们打算将这些技术开发为PostgreSQL的开源扩展，并生成有关如何将扩展合并到需要模式匹配的各种应用程序域的综合文档。我们将在未来开始探索更专业的模式度量和度量，例如马哈拉诺比斯距离和其他。此外，我们将研究替代内核和块组织，例如使用多个线程来计算遵循并行化还原方法的单个模式匹配分数。

此外，我们将利用这些扩展来创建可扩展的异构内容检索系统，该系统将使用图像内容，图像元数据和其他可用于查询构建的信息。我们将开发PostgreSQL集成以支持可视化码本方法，即作为预匹配表表达式。由于数据库本质上是多进程并发系统，因此有必要开发GPU协调策略。我们的目标是提供一组强大的开源扩展，以支持PostgreSQL中的大规模，高吞吐量模式匹配，使研究人员能够探索与其他数据匹配的模式的新颖开发。

## 引用

[1] A. Berard and G. Hebrail, “Searching time series with hadoop in an electric power company,” in Proceedings of the 2Nd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, ser. BigMine ’13. New York, NY, USA: ACM, 2013, pp. 15–22.

[2] C.-W. Lu, C.-M. Hsieh, C.-H. Chang, and C.-T. Yang, “An improvement to data service in cloud computing with content sensitive transaction analysis and adaptation,” in Computer Software and Applications Conference Workshops (COMPSACW), 2013 IEEE 37th Annual, 2013, pp. 463–468.

[3] N. Rizvandi, J. Taheri, and A. Zomaya, “On using pattern matching algorithms in mapreduce applications,” in Parallel and Distributed Processing with Applications (ISPA), 2011 IEEE 9th International Symposium on, 2011, pp. 75–80.

[4] C.-H. Lin, C.-H. Liu, and S.-C. Chang, “Accelerating regular expression matching using hierarchical parallel machines on GPU,” in Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE, 2011, pp. 1–5.

[5] K. Derpanis, M. Sizintsev, K. Cannons, and R. Wildes, “Action spotting and recognition based on a spatiotemporal orientation analysis,” Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 35, no. 3, pp. 527–540, 2013.

[6] L. Wang, S. Chen, Y. Tang, and J. Su, “Gregex: GPU based high speed regular expression matching engine,” in Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on, 2011, pp. 366–370.

[7] C. Beleznai, D. Schreiber, and M. Rauter, “Pedestrian detection using GPU-accelerated multiple cue computation,” in Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on, 2011, pp. 58–65.

[8] G. Vasiliadis, M. Polychronakis, and S. Ioannidis, “Parallelization and characterization of pattern matching using GPUs,” in Workload Characterization (IISWC), 2011 IEEE International Symposium on, 2011, pp. 216–225.

[9] A. Kooijman and J. Vergeest, “GPU implementation of the lft shape matching algorithm,” in Parallel Computing in Electrical Engineering (PARELEC), 2011 6th International Symposium on, 2011, pp. 111–116.

[10] M. Rauter and D. Schreiber, “A GPU accelerated fast directional chamfer matching algorithm and a detailed comparison with a highly optimized cpu implementation,” in Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on, 2012, pp. 68–75.

[11] D.-D. Truong, V.-T. Nguyen, A.-D. Duong, C.-S. N. Ngoc, and M.- T. Tran, “Realtime arbitrary-shaped template matching process,” in Control Automation Robotics Vision (ICARCV), 2012 12th International Conference on, 2012, pp. 1407–1412.

[12] M. Nabiyouni and D. Aghamirzaie, “A highly parallel multi-class pattern classification on gpu,” in Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on, May 2012, pp. 148–155.

[13] N.-P. Tran, M. Lee, S. Hong, and J. Choi, “High throughput parallel implementation of aho-corasick algorithm on a gpu,” in Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International, May 2013, pp. 1807–1816.

[14] A. Cevahir and J. Torii, “Gpu-enabled high performance online visual search with high accuracy,” in Multimedia (ISM), 2012 IEEE International Symposium on, Dec 2012, pp. 413–420.

[15] J. Zhong and B. He, “Kernelet: High-throughput gpu kernel executions with dynamic slicing and scheduling,” IEEE Transactions on Parallel and Distributed Systems, vol. 99, no. PrePrints, pp. 1–11, 2013.

[16] NVIDIA Corporation, NVIDIA CUDA C Programming Guide, June 2011.

[17] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, “Object retrieval with large vocabularies and fast spatial matching,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2007.

[18] K. Mikolajczyk and C. Schmid, “Scale and affine invariant interest point detectors,” International Journal of Computer Vision, vol. 60, no. 1, pp. 63–86, 2004. [Online]. Available: http://www.robots.ox.ac.uk/∼vgg/research/affine/index.html

**翻译作者简介：朱君鹏，华东师范大学博士研究生，个人兴趣主要集中在：新型硬件（GPU、RDMA、FPGA）等在数据库中的应用，架构设计与并行计算。**

**文章链接：https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6977041**