**使用数据传输方法在PostgreSQL上执行**

**外部连接运算符**

译者: 朱君鹏， 李红艳

**作者简介**

**Ryota Takizawa，Hideyuki Kawashima，Ryuya Mitsuhashi，Osamu Tatebe，**来自日本筑波大学。这是一篇关于数据库中连接运算如何有效地实现在数据库中。

## 译者简介

**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、NVMe等）在数据库中的应用，架构设计，分布式系统，机器学习系统与并行计算技术。

**李红艳，**杭州衡数软件技术负责人，有百货公司（超过40家）、视光集团和知名电商新零售大数据分析经验。

## 摘要

随着传感设备的发展，人类管理的数据量迅速增加。为了管理如此庞大的数据，关系数据库管理系统（RDBMS）起着关键作用。 RDBMS将现实世界数据建模为n元关系表。Join运算符是最重要的关系运算符之一，其加速度已得到广泛而深入的研究。 RDBMS如何提供这样一个有效的连接运算符？连接算子的性能改进已经深入研究了十年，并且已经提出了许多技术方案。我们面临的问题是如何在真正的RDBMS中实际使用这些优秀的技术。我们建议通过数据传输方法实现有效的连接技术。该方法RDBMS内部创建一个钩子点，从RDBMS中的运算符管道中提取数据流，并将原始的连接运算符应用于数据，并最终将结果返回给RDBMS中的运算符管道。实验结果表明，与PostgreSQL相比，我们提出的方法实现了1.42倍的加速。我们的代码可以在GitHub上找到。

## 关键词

关系数据库， PostgreSQL，并行哈希连接运算

## 概述

### 1.1 背景

随着传感技术的发展，关系数据库的重要性不断增加。大型天气测量望远镜（LSST）每晚产生20TB，大型强子对撞机（LHC）每年产生30PB的数据。 Google和Facebook等社交网络公司每天都会收集大量人为生成的数据。

为了管理这些巨大的数据，关系数据库管理系统（RDBMS）起着关键作用。 RDBMS将现实世界数据建模为n元关系表。尽管为这样的大数据开发了各种NoSQL技术，但RDBMS仍然是一种流行的选择，因为RDBMS提供了用于数据运算的SQL查询语言。由于SQL，用户不需要研究新语言来进行数据运算，但是，RDBMS需要提供高性能，因为用户需要同时兼顾可用性和性能。

为了加速RDBMS，近年来开发了新的关系引擎。 Facebook提供Presto [6]，而Apache基金会提供基于Apache Spark的Impala [1]和SparkSQL [2]。 这些系统专为并行和分布式计算环境而设计，因此它们提供高并行性，以利用当今的多核架构，如Xeon Phi(至强融核)和高速互连技术，如InfiniBand。他们能高效地执行关系运算符，如选择，投影，聚合和连接。

连接运算符在RDBMS中是一个有用的运算符，因此~~它~~被频繁地讨论。另一方面，连接运算的缺点是其昂贵的运行代价。对于具有M和N的关系表R和S，连接的初始成本是O(MN)，这对于非等值连接是必需的。如果我们将连接运算符的类型限制为等值连接，那么可以将成本减少到O(N/k)，理想的哈希连接将散列函数应用于R，并且将k个线程应用于超过k个CPU核心。因为随着上面提到的数据量的增加，关系表的大小变得越来越大，所以N在今天往往是巨大的。这种理想的连接运算符应由RDBMS提供，以加速查询处理。

### 问题

真正的RDBMS如何在实际意义上提供这样一个有效的连接运算符？已经对连接算子的性能改进进行了长达~~数~~十年的深入研究，并且已经提出了许多技术方案[12] [3] [13] [5] [11]。我们关注的问题是如何在真正的在RDBMS中实际使用这些优秀的技术。

最近的分布式和并行数据库（例如Presto，Impala）提供了高效率，而其稳定性尚未证明，因为它们仍然很年轻，因此其代码库不成熟。如果一个人成功地在引擎内部实现了这么好的技术，那么它~~无需~~的确会加速系统，但是代码维护成本可能会很昂贵，因为大多数代码在这样的年轻引擎中都不稳定。

PostgreSQL和MySQL等传统数据库提供了很高的稳定性，有许多用例的支持，而与最近的引擎相比，它的效率有限。对于这样的引擎，有效技术的实现并不是微不足道的，因为它的稳定性很好。例如，PostgreSQL使用进程~~并且它~~而不使用线程，因此它需要调用许多系统调用来运算并行查询处理，因为许多进程间通信是必需的，这在线程的情况下是不必要的。有可能重写PostgreSQL内核以便它利用线程，这需要巨大的努力。这种方法是不合理的，因此通常不被程序员接受。

因此，从稳定性和实施成本的角度来看，将实用技术应用到真实RDBMS的内部似乎是不恰当的方法。

### 贡献

为了解决这个问题，我们建议通过数据传输方法实现一种有效的连接技术。方法在RDBMS内部创建一个钩子点，从RDBMS中的运算符管道中提取数据流，并将我们的原始连接运算符应用于提取的数据，并最终将结果返回给RDBMS中的运算符管道。对于等值连接运算，PostgreSQL实现了一个非并行的散列连接运算符，它不利用多核架构提供的并行性。我们改为实现并行和分布式哈希连接运算符，它应该比PostgreSQL中的非并行运算符更高效。

为了评估所提出方法的性能，我们进行了多个节点的实验。结果如图1所示。有趣的是，所提出的方法可能因数据大小而异或更差。当数据大小很小时，PostgreSQL性能更好，但是当数据大小很大时，我们提出的方法获胜。我们将在本文的其余部分详细解释这些细节。我们在PostgreSQL之外执行连接运算符的代码可以在GitHub上获得[16]。

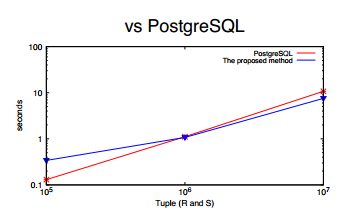


图1 PostgreSQL与本文提出的方法之间的对比

除了数据传输方法，（a）我们可以在PostgreSQL中实现一个新的物理连接运算符，或者（b）我们可以利用库运算系统库。然而，方法（a）需要很长时间，方法（b）可以在有限的情况下使用，这在第6节中描述。

### 组织

本文的其余部分安排如下。第2节描述了背景，包括哈希连接，PostgreSQL的连接性能和研究问题。第3节描述了提出的方法，数据传输方法。我们描述了设计和实现。第4节描述了我们在这项工作中实现的并行和分布式散列连接。第5节描述了实验结果及其实验设置。第6节描述了相关工作，包括将外部运算符集成到数据库系统中的连接技术和方法。最后，第7节总结。

## 2．背景

### 2.1 hash连接

Hash join [14]是一个关系运算符，在数据库系统中很流行。这里我们解释一下在PostgreSQL上实现的简单散列连接技术。请考虑加入两个关系R和S. R是外表，S是内表。当R能够完全放在物理内存中时，则R的所有部分都被散列。接着，对于S中元组的每个匹配过程，访问散列R。在理想情况下，成本是O(S)而没有哈希构造，这比嵌套循环连接要多得多。

简单的散列连接如图2所示。如上所述，关系R是第一个散列，S中的元组使用散列函数匹配，因此它可以在散列R中使用适当的元组。这样简单的技术仍然在PostgreSQL中使用，如过程中的explain命令所示。

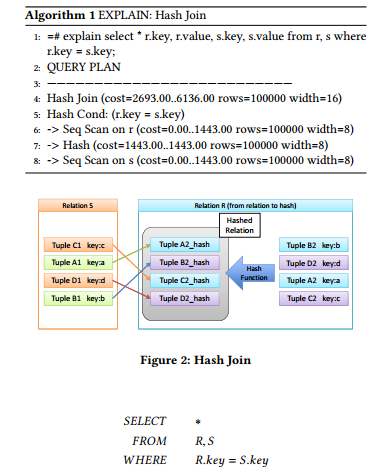
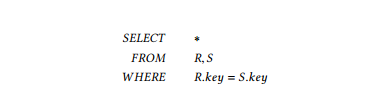


图2 哈希连接运算

### 2.2 PostgreSQL上的等值连接性能

散列连接是否会加速PostgreSQL中的查询处理？我们使用PostgreSQL-9.4.4测量了性能。我们使用了只有两个类型为整数的属性（键和值）的小关系。查询~~是一个~~非常简单~~的查询~~，如下所示。



实验结果如图3所示。纵轴表示查询处理所需的时间，横轴表示元组的大小。当元组大小为105时，执行时间为0.1秒。另一方面，当元组的大小为107时，则执行时间为10.7秒。 因此，PostgreSQL中散列连接的性能与关系的大小成正比。结果与上述成本为O（S）的理论一致。

另一方面，由于PostgreSQL不提供并行散列连接，因此根本不利用多核~~或多核~~架构。当数据大小相对较小（例如105）时，当前方法就足够了。但是，如果数据的大小要大得多（即大于107），则需要更多的加速。

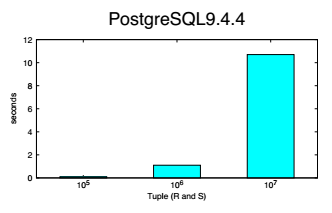


图3 PostgreSQL-9.4.4性能

### 2.3 研究问题

PostgreSQL如何提供有效的连接运算符？连接算子的性能改进已经深入研究了十年，并且已经提出了许多技术[4,9,20]。我们面临的问题是如何在PostgreSQL中实际使用这些优秀的技术。

PostgreSQL提供了许多用例历史支持的高稳定性，而其效率如上所述受到限制。对于这样的引擎，由于其稳定的系统架构，高效技术的实现并非无足轻重。 PostgreSQL使用进程来处理同时执行的任务，并且它不使用更轻量级的线程。 因此，它需要调用系统调用进行并行查询处理，如果我们在PostgreSQL中实现并行连接技术，则在线程的情况下不需要。重写PostgreSQL内核以便它利用线程并不容易。任务并行的抽象实现是任何数据系统的核心，PostgreSQL具有很长的代码稳定历史。 因此它显然需要大量的努力去实现。 因此，我们认为这种方法并不合理。

总的来说，从稳定性和实现成本的角度来看，在PostgreSQL内核中实现高效技术似乎是不恰当的方法。

## 3．提出的方法：数据传输方法

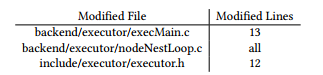
我们提出数据传输方法来设计一种有效的连接技术。方法是在PostgreSQL内部设置一个钩点，并从PostgreSQL中的运算符管道中提取数据流，并将原始的连接运算符应用于数据，并最终将查询处理的结果返回给PostgreSQL中的运算符管道。

### 3.1 在PostgreSQL中的修改

在PostgreSQL中，和其他关系数据库管理系统一样，接收到的SQL查询首先被转换为由关系运算符组成的有向无环图。接着，运算符管道按ExecutorRun，ExecutePlan和ExecProcNode的顺序执行。 PostgreSQL的查询管道基于volcano [8]。在~~火山~~volcano系统中，运算符调用低级运算符来生成元组。如果调用的运算符需要输入元组来生成结果元组，那么它也会调用低级的运算符。 因此，ExecProcNode执行关系运算符并将运算符处理的结果返回给调用的节点。

在PostgreSQL内部，我们在位于“executor”目录下的execMain.c中修改了执行程序模块。 Executor有四个子模块：ExecutorStart，ExecutorRun，ExecutorFinish和ExecutorEnd，如图4所示。每个子模块都有一个可以切换调用函数的钩子点。为了我们的目的，我们设计了一个钩点函数“ExecProcNode hook”。 ExecProcNode挂钩在ExecPlan中挂钩ExecProcNode。为了实现我们的方案，我们在2个文件中只修改了25行，并且在PostgreSQL的nodeNestLoop.c中完成了整个部分。我们总结了表1中的修改。

表1在PostgreSQL中的修改



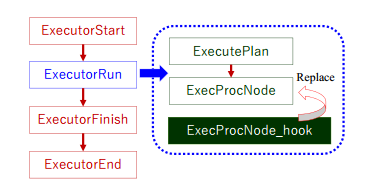
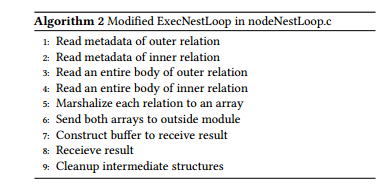


图4 PostgreSQL中的钩子运算符

我们强制PostgreSQL选择NestLoopJoin作为物理连接运算符，并将原始nodeNestLoop.c替换为我们的。我们在算法2中显示伪代码。它首先读取两个关系的元数据，然后读取它的主体。每个主体都被编组成一个数组，以通过单个发送系统调用传输所有内容。 接着，发送两个数组，并收到结果。对于编组，我们定义了自己的数据结构，并且它的主体是由PostgreSQL提供的palloc调用形成的。我们使用PostgreSQL提供的“load”命令将我们自己的ExecNestLoop集成到PostgreSQL二进制文件中。



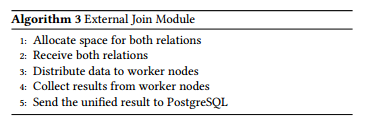
我们的实施有一些缺点。我们只支持原始类型的C语言。 因此，不支持数组和变长对象。目标关系的大小应该小到足以在内存中。

请注意，我们的方法对~~于~~基于传统volcano [8]的查询管道方法是有效的，其中每个运算符返回元组或它的调用元组。另一方面，这种方法是HyPer采用的不合适的运算符合并技术[17]，因为它在内部将多个运算符合并到一个运算符。

例如，HyPer [15]中的morsel驱动管道系统将三个连接运算符合并为一个具有三个forloops的运算符，以消除过多的中间数据生成和函数调用。在此设计中，系统外的运算符无法合并。因此，HyPer不能采用数据传输方法。

### 3.2 外部连接运算符

要从PostgreSQL接收数据，需要在提出查询之前启动外部连接运算符模块。它从PostgreSQL接收数据，并执行并行和~~分散~~分布式的散列连接，并将结果返回给PostgreSQL。详细信息显示在算法3中。



## 4 并行哈希连接运算

PostgreSQL仅提供非并行哈希连接。我们建议使用并行和分布式哈希连接来利用数据传输方法来利用多核和多节点并行。在这里我们解释我们的计划。在我们的方案中，我们首先对一个关系进行分区，然后将子关系分配给多个节点，如图5所示。

接着，我们在节点中构建了哈希表，并使用多个线程并行探测哈希表。在节点内部，并行散列连接由无分区散列连接方式执行。我们在图6中展示了这个模式。 这项技术只为R创建一个哈希表，多个线程探测哈希R。

这里我们描述算法。第一步是主节点上的数据分发。详细信息在算法4中提供。主节点上的PostgreSQL读取两个关系R和S。请注意，对于我们的方法，我们让PostgreSQL选择嵌套循环连接而不是散列连接。主节点应用具有工作节点节点“worker”的散列函数，并将R和S分区为Ri(i = 1, ···, n)和Si(i = 1,···, n)。 然后，每个子关系Ri和Si被转移到工作节点workeri(i = 0, ··· ,n - 1)。接着，join阶段在每个工作节点上运行。在算法5中描述了细节。工作节点工作者从主节点主机接收Ri和Si。 workeri执行简单的并行散列连接。它首先为Ri构建一个哈希表，然后多个线程使用散列R和S并行执行探测。因为S中的元组是独立的，所以并行连接有效地执行。使用fetch\_and\_add指令完成元组的赋值，该指令比mutex轻得多。探测阶段的结果被聚合到resulti。每个节点workeri都将resulti传输到主节点。在接收到所有resulti时，master获得R 连接S的结果，并且完成连接运算。

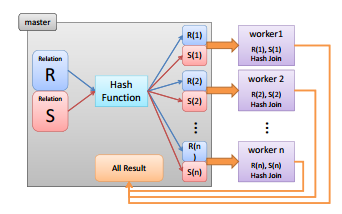


图5 数据分布和结果收集

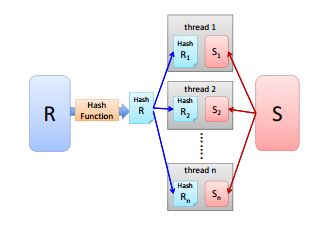
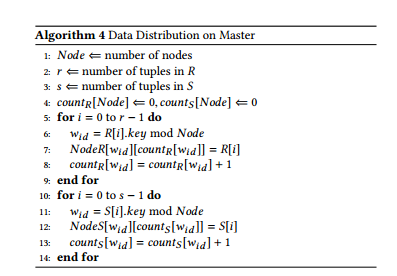
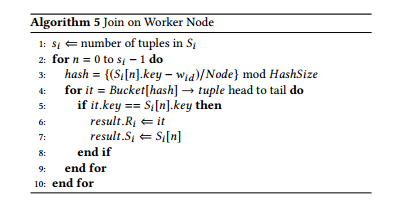


图6 无分隔哈希连接





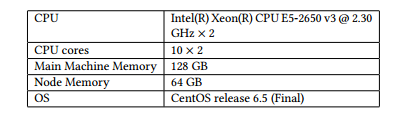
## 5 实验评估

我们评估提出的方法的性能，并将其与原始的PostgreSQL进行比较。

### 5.1 实验环境

我们在表中显示评估环境。每个工作节点都使用多个线程进行简单的并行散列连接。在我们的服务中，每个工作节点使用20个线程进行探测，因为每个节点有20个核心。

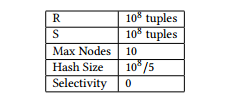
表2 实验环境



### 5.2 并行哈希连接

为了验证并行散列连接模块本身，我们评估其性能。数据集的信息如表3所示。

表3 数据集



我们评估并行和分布式散列连接，改变工作节点的数量。为了消除结果元组的影响，我们调整了数据集，使其不会产生任何结果（即零连接选择率。）。在每次测量中，输入关系的大小设置为相同。

图7显示了结果。输入元组的大小为108。当节点为1和10时，分别对应的执行时间为9.67秒和2.51秒，实现了3.85倍的加速。尽管我们观察了横向扩展的趋势，但分布式计算的局限性仍然有限。

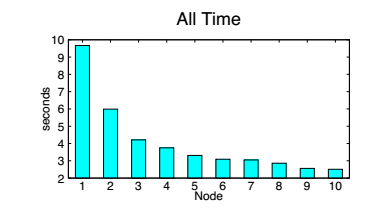


图7 并行和分布式哈希连接性能

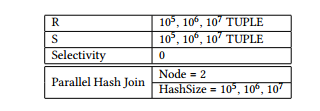
### 5.3 使用PostgreSQL进行评估

我们在这里评估数据传输方法。我们使用了一个简单的查询，包括R和S的单个连接，如下所示。我们将此查询提供给修改后的PostgreSQL，并测量执行时间。



该评估的数据集如表4所示。元组的大小为105,106,107。将选择~~性~~率设置为零。节点数量限制为两个。

表4 PostgreSQL评估时使用的数据集



实验结果如图8所示。有趣的是，根据数据的大小，提出的方法可能更糟或更差。当数据大小很小（105）时，原始方法更好。但是，当数据大小较大时（106和107），本文提出的方法会获胜。对于107情况，提出的方法比原始方法快1.42倍。因此，对于等值连接运算符，所提出的方法仅在数据大小相对较大时才~~执行~~有效。

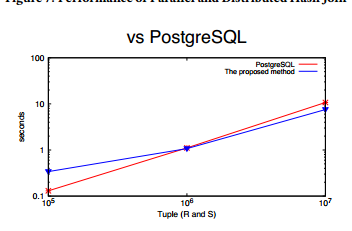


图8 PostgreSQL与本文提出的方法之间的对比

## 6. 相关工作

### 6.1快速连接技术

通过在现代架构体系~~结构~~中利用关系元组的独立性和高并行性，对并行哈希连接技术进行了全面的研究。He等人和Kaldewey等人利用GPU [12]。 Blanas等人，Balkesen等人和Kim等人利用多核CPU [3] [13] [5]。 Jha等人利用Xeon Phi [11]。随着新硬件的出现，新的连接技术将诞生。

### 6.2将外部运算符集成到数据库系统中

将原始的外部运算符有效地集成到真实的数据库系统中是很困难的，就我们所知的方法有三种。 第一种方法是修改系统的内部。 它有时需要对数据库系统进行基本的设计更改， 并迫使开发人员进行大量实施。因此，通常需要很长时间。 PostgreSQL-9.6提供并行连接[19]，但它没有并行化单个连接运算，这可能意味着并行连接运算符的实现对PostgreSQL来说不是一件容易的事。

第二种方法是我们在本文中采用的数据传输方法。 PG-Strom采用相同的方法[18]。如上所述，数据传输成本是一个严重的问题。即使散列连接减少了查询处理时间，数据传输成本也会增加。 本文只处理对这种方法最有效的等值连接算子。 因此，更多计算密集型任务，例如非等值连接，需要O(MN)代表R，S与M， N元组，可以加速。

第三种方法是使用库运算系统。通过使用库运算系统如Barrelfish，可以显着降低数据传输成本[7]。但是，这种方法失去了~~记忆~~内存保护功能，因此可以在特殊情况下采用。

## 7 结论

RDBMS如何提供这样一个有效的连接运算符？这是这项工作的研究问题。连接算子的性能改进已经深入研究了十年，并且已经提出了许多技术方案[4,9,20]。我们考虑的问题是如何在真正的RDBMS中实际使用这些优秀的技术。我们建议通过数据传输方法实现高效的连接技术。方法在PostgreSQL内部做了一个挂钩点，并从PostgreSQL中的运算符管道中提取数据流，并将并行和分布式散列连接运算符应用于数据，并最终将结果返回给PostgreSQL中的运算符管道。有趣的是，根据数据的大小，提出的方法更糟或更糟。当数据量很小（105）时，原始版本更好。然而，当数据大小很大时（106和107），本文提出的方法获胜。对于107的情况，提出的方法比原始方法快1.42倍。我们得出结论，对于等值连接运算符，所提出的方法仅在数据大小相对较大时才有效。

当计算成本高昂时，数据传输方法应该表现更有效。这些运算符包括非等值连接运算符和加密数据处理[10]。我们计划在未来的工作中解决这些问题。

## 引用

[1] Apache. [n. d.]. Apache Impala (incubating): the open source, native analytic database for Apache Hadoop. https://impala.apache.org. ([n. d.]).

[2] Apache. [n. d.]. Apache SparkSQL. https://spark.apache.org/sql. ([n. d.]).

[3] Cagri Balkesen, Gustavo Alonso, Jens Teubner, and M. Tamer Ozsu. 2013. Multi-core, Main-memory Joins: Sort vs. Hash Revisited. Proc. VLDB Endow. 7, 1 (Sept.2013), 85–96. https://doi.org/10.14778/2732219.2732227

[4] R. Barber, G. Lohman, I. Pandis, V. Raman, R. Sidle, G. A?aluri, N. Chainani, S. Lightstone, and D. Sharpe. 2014. Memory-e?cient Hash Joins. Proc. VLDB Endow. 8, 4 (Dec. 2014), 353–364. https://doi.org/10.14778/2735496.2735499

[5] Spyros Blanas, Yinan Li, and Jignesh M. Patel. 2011. Design and Evaluation of Main Memory Hash Join Algorithms for Multi-core CPUs. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD’11). ACM, New York, NY, USA, 37–48. https://doi.org/10.1145/1989323.1989328

[6] Facebook. [n. d.]. Presto — Distributed SQL query Engine for Big Data.https://prestodb.io. ([n. d.]).

[7] Jana Giceva, Gerd Zellweger, Gustavo Alonso, and Timothy Rosco. 2016. Customized OS Support for Data-processing. In Proceedings of the 12th International Workshop on Data Management on New Hardware (DaMoN ’16). ACM, New York, NY, USA, Article 2, 6 pages. https://doi.org/10.1145/2933349.2933351

[8] Goetz Graefe. 1990. Encapsulation of Parallelism in the Volcano Query Processing System. In Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data (SIGMOD ’90). ACM, New York, NY, USA, 102–111. https://doi.org/10.1145/93597.98720

[9] Jiong He, Mian Lu, and Bingsheng He. 2013. Revisiting Co-processing for Hash Joins on the Coupled CPU-GPU Architecture. Proc. VLDB Endow. 6, 10 (Aug. 2013), 889–900. https://doi.org/10.14778/2536206.2536216

[10] Kentaro Horio, Hideyuki Kawashima, and Osamu Tatebe. 2017. Efficient parallel summation on encrypted database system. In 2017 IEEE International Conference on Big Data and Smart Computing, BigComp 2017, Jeju Island, South Korea, February 13-16, 2017. 178–185. https://doi.org/10.1109/BIGCOMP.2017.7881735

[11] Saurabh Jha, Bingsheng He, Mian Lu, Xuntao Cheng, and Huynh Phung Huynh. 2015. Improving Main Memory Hash Joins on Intel Xeon Phi Processors: An Experimental Approach. Proc. VLDB Endow. 8, 6 (Feb. 2015), 642–653. https://doi.org/10.14778/2735703.2735704

[12] Tim Kaldewey, Guy Lohman, Rene Mueller, and Peter Volk. 2012. GPU Join Processing Revisited. In Proceedings of the Eighth International Workshop on Data Management on New Hardware (DaMoN ’12). ACM, New York, NY, USA, 55–62. https://doi.org/10.1145/2236584.2236592

[13] Changkyu Kim, Tim Kaldewey, Victor W. Lee, Eric Sedlar, Anthony D. Nguyen, Nadathur Satish, Jatin Chhugani, Andrea Di Blas, and Pradeep Dubey. 2009. Sort vs. Hash Revisited: Fast Join Implementation on Modern Multi-core CPUs. Proc. VLDB Endow. 2, 2 (Aug. 2009), 1378–1389. https://doi.org/10.14778/1687553.1687564

[14] Masaru Kitsuregawa, Hidehiko Tanaka, and Tohru Moto-Oka. 1983. Application of Hash to Data Base Machine and Its Architecture. New Generation Comput. 1,1 (1983), 63–74. https://doi.org/10.1007/BF03037022

[15] Viktor Leis, Peter Boncz, Alfons Kemper, and ?omas Neumann. 2014. Morseldriven Parallelism: A NUMA-aware Query Evaluation Framework for the Manycore Age. In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD ’14). ACM, New York, NY, USA, 743–754. htps://doi.org/10.1145/2588555.2610507

[16] Ryuya Mitsuhashi. [n. d.]. PostgreSQL Extension for External Join. ([n. d.]). https://github.com/ryumt/ExternalJoin-pgsql9.5.2

[17] Tomas Neumann. 2011. Efficiently Compiling E?cient query Plans for Modern Hardware. PVLDB 4, 9 (2011), 539–550. http://www.vldb.org/pvldb/vol4/p539-neumann.pdf

[18] PGStrom. [n. d.]. PG-Strom: Limit Breaker of PostgreSQL. ([n. d.]). http://strom.kaigai.gr.jp

[19] PostgreSQL. [n. d.]. Parallel Plans on PostgreSQL-9.6. ([n. d.]). https://www.postgresql.org/docs/9.6/static/parallel-plans.html

[20] Jens Teubner, Gustavo Alonso, Cagri Balkesen, and M. Tamer Ozsu. 2013. Mainmemory Hash Joins on Multi-core CPUs: Tuning to the Underlying Hardware. In Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013) (ICDE ’13). IEEE Computer Society, Washington, DC, USA, 362–373. htps://doi.org/10.1109/ICDE.2013.6544839

文章链接：https://dl.acm.org/citation.cfm?id=3149480&dl=ACM&coll=DL