哈尔滨工业大学计算机科学与技术学院

《算法设计与分析》

课程报告

|  |  |
| --- | --- |
| 学号 | 120L020527 |
| 姓名 |  |
| 班级 | 2003004 |
| 报告日期 | 2021年12月28日 |

# 论文题目 （1）题目：FAST: FPGA-based Subgraph Matching on Massive Graphs；

（2）作者：**Xin Jin†, Zhengyi Yang§, Xuemin Lin§, Shiyu Yang\,**

**Lu Qin‡, You Peng§**；

（3）刊物：**ICDE，19-22 April 2021；**

（4）出处链接：[**https://ieeexplore.ieee.org/document/9458810**](https://ieeexplore.ieee.org/document/9458810)

# 论文阅读报告

## 摘要

子图匹配是广泛应用于许多应用中的一种基本运算。然而，由于其np硬度和图数据的爆炸式增长，计算子图匹配具有挑战性，特别是在大型图中。在本文中，我们的目标是在单个机器上使用fpga放大子图匹配。具体来说，我们提出了一个CPU-FPGA协同设计框架。在CPU方面，我们首先开发了一种新的辅助数据结构候选搜索树(CST)作为子图匹配的完整搜索空间。cstc可以被分区并完全加载到fpga的片上内存中。然后，提出了一种负载估计技术来平衡CPU和FPGA之间的负载。在FPGA方面，我们设计并实现了第一个基于FPGA的子图匹配算法fast。为了充分利用fpga上的流水线机制，针对fast提出了任务并行化优化和任务生成器分离策略，实现了大规模并行化。此外，为了充分利用FPGA的片上存储器，我们精心开发了一种BRAMonly匹配过程，避免了FPGA的BRAM和DRAM之间昂贵的中间数据传输。综合实验表明，与目前最先进的算法mdafand ceci相比，fast分别达到了462.0x和150.0x的加速。此外，在我们的实验中，FASTis是唯一一种可以用一台机器处理10亿尺度图的算法。

## 问题定义

论文探讨了如何充分利用fpga的流水线机制来加速子图匹配问题。其中存在的问题有

（1）FPGA上的严格流水线设计。fpga采用流水线设计，在这种设计中，完全流水线的循环不需要迭代之间的数据依赖关系。因此，现有的基于回溯的算法不能直接在fpga上实现。此外，由于fpga的时钟频率比cpu低一个数量级(例如，300MHz vs. 2GHz)，因此需要在fpga上设计复杂的子图匹配单元来获得高性能；

（2）有限的FPGA片上内存。fpga具有较小的片上内存(BRAM)，通常只有几十兆字节;因此，在fpga上进行子图匹配时，由于图数据和中间结果庞大，很容易溢出BRAM。此外，由于从FPGA的外部存储器(DRAM)取数据需要比BRAM更多的周期(例如，8周期比1周期)，频繁的数据传输在BRAM和DRAM之间会显著地损害性能。因此，如何在fpga上有效地管理数据，以减少BRAM和DRAM之间的数据传输操作，是一个相当具有挑战性的问题。

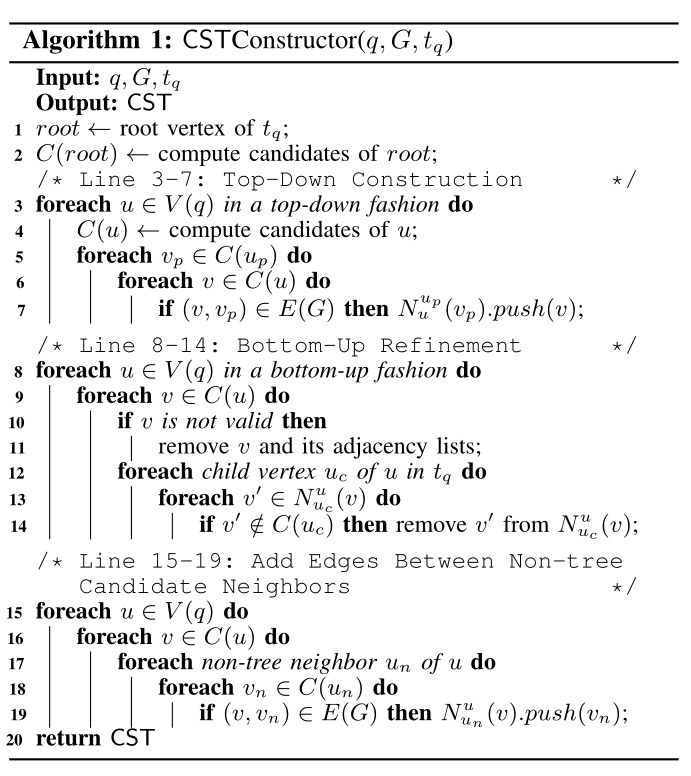
## 算法或证明过程

**一．软件实现**

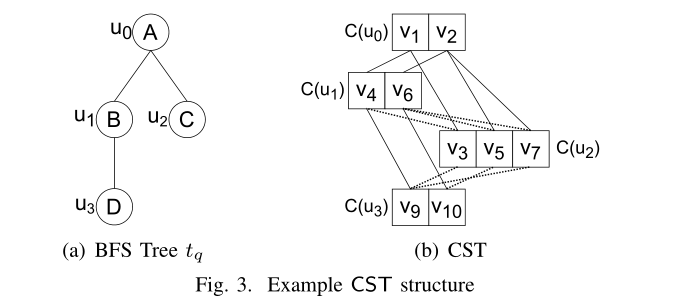
1：CST STRUCT

(1)描述：我们采用索引枚举框架;即构造一个辅助数据结构，然后根据该数据结构计算所有嵌入。按照传统的[13]、[31]技术，首先将查询图转化为生成树。给出一个查询图和一个数据图，在此基础上构建辅助数据结构候选搜索树(CST)。

(候选搜索树)给定一个查询图q和一个数据图G，候选搜索树ecst (q,G)是一个同构toq的图2。每个vertexuofCST(q,G)都有一个候选集，记为dc (u)，它存储G中所有可以映射的顶点。对于相邻的顶点uandu0incst (q,G)， env∈C(u)和v0∈C(u0)之间有一条边当且仅当(v, v0)∈E(G)。我们在cstif上下文清楚的情况下去记tecst (q,G)。给定查询图和它的BFS树q，我们称相邻顶点为uanduninCSTnon-tree neighborsif(u, un)∈E(q)，但(u, un)/∈E(tq)。v∈C(u)和vn∈C(un)相邻的非树邻居和unincstare称为非树候选邻居。我们用enu0 (v)表示v∈C(u)对(u, u0)inCST的邻接表，即Nuu0(v) ={v0∈C(u0)|(v, v0)∈E(CST)}。cst继承了tq的父子关系。我们用anducto分别表示ofu的父顶点和子顶点。vertexu inCSTis alaforrootvertex如果分别没有子顶点或父顶点。



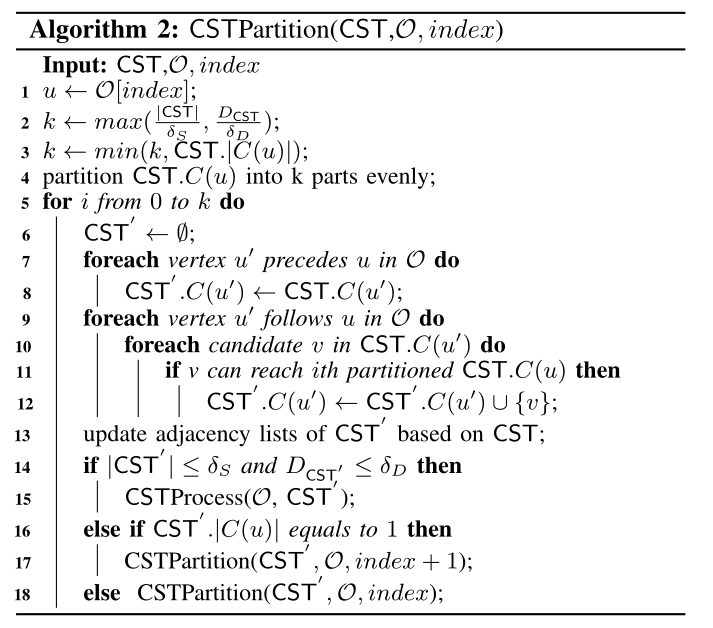
(2)例子：例如，给定查询图q，图1的数据图和图3的BFS树(a)，对应的csti在图3(b)。而v4∈C(u1)和v3∈C(u2)则被称为非树候选邻居，C(u1) ={v4, v6}，C(u2) ={v3, v5, v7}， Nu1u2(v6) ={v5, v7}和nu2u3 (v3) ={v9}。qing {(u0, v1)，(u1, v4)，(u2, v3)，(u3, v9)}和{(u0, v2)，(u1, v6)，(u2, v5)，(u3, v10)}的所有嵌入都可以通过只遍历cst来计算。



(3)分析健壮性：cst应该作为给定查询图在数据图g上的完整搜索空间。为了实现这一点，cst必须满足以下健全约束:对于CST中的每个顶点u，如果G中有q的嵌入将u映射到v，那么v必须在C（u）中。注意，尽管在可靠性要求中我们只考虑查询顶点的候选项，候选项之间的边会根据我们的cst定义自动包含。

2. CST Partition：

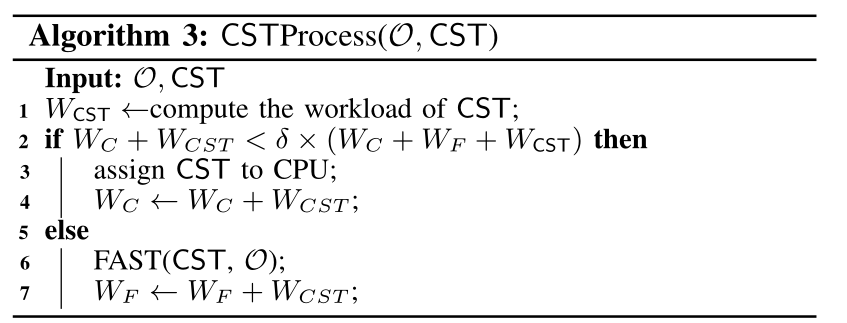
(1)描述：由于fpga上的片上资源有限，csti通常太大而不能完全加载到BRAM中。一般来说，BRAM的读时延为1个周期，而DRAM的读时延大约为7-8个周期。我们的实验表明，当我们从DRAM访问cstfrombram时，性能显著下降(章节VII-A)。另一方面，访问cstare是随机且不可预测的，这消除了从DRAM到BRAM预取数据的可能性。因此，有必要对cstand进行分区，将它们一个一个地卸载到FPGA上。除了CST的大小，记为|CST|，我们还设置了候选inCST的最大程度的限制，即DCST。原因是邻接表的最大访问端口数在fpga上是有限的，这将在VI-A节中详细讨论。我们分别用δ sand δ d表示|、CST、|和dcst的阈值。我们将cstif划分为|CST|> δSorDCST> δD。csti的分区策略如算法2所示。注意，本文中我们采用基于路径的方法来计算匹配顺序oin，它通过对tq的根到叶路径进行排序来确定esoby。然而，我们的方法被设计用于任意连接的匹配顺序。首先，在对cst进行分区时，对根顶点inCST进行候选分区。如果只有一个根顶点inCST的候选，我们继续划分下一个vertexuinO的候选。第一步是确定分区系数k,这等于最大值之间的比率|中科| andDCSTto相应的阈值(2)行。Ifkexceeds候选人的数量,例如,C (u) | |,我们setk | C (u) |(第3行)。我们partitionC (u) intokparts均匀,然后构造一个newCSTlevel-by-level自上而下的方式。对于那些在uino之前的顶点，我们选择与旧dcst相同的候选点(第7-8行)。对于uinO之后的那些顶点，我们在olddcst中选择至少可以到达partitionedC(u)中的一个候选点(第9-12行)。一旦它满足CST和Dcst 约束(第15行)，就会被卸载到FPGA或分配给CPU。否则，将进一步递归地进行分区。



(2)例子：如图4(a)所示，假设k=2，我们首先将根候选{v1, v2}划分为{v1}和{v2}两部分。然后，为了构造由v1产生的cst，我们选择u1和u2的候选，它们与v1相邻，它们是{v3, v5}和{v6, v8}。在这之后，我们选择u3可以到达v1的候选项，它们是{v9, v10}。显然，图4(b)和图4(c)两个分区的cst之间的搜索空间没有重叠，因此不会报告重复的结果。

3. Schedule the Matching Tasks

（1）描述：完成cst分区后，主机端共享一小部分匹配任务，进一步提高整体吞吐量。考虑到CPU和FPGA之间的负载均衡，首先需要估计cst的负载，记为Wcst。由于实际图的幂律特性，在不同的CST中，搜索空间的大小通常会有很大的不同。在不考虑任何假阳性的情况下，我们使用cst中的嵌入数来估计wcst。它可以使用动态规划算法以自底向上的方式计算。对于每个候选的ev∈C(u)，我们计算由从musuch thatuis映射到tov的匹配顺序的后缀诱发的qgraph的子图的嵌入个数。

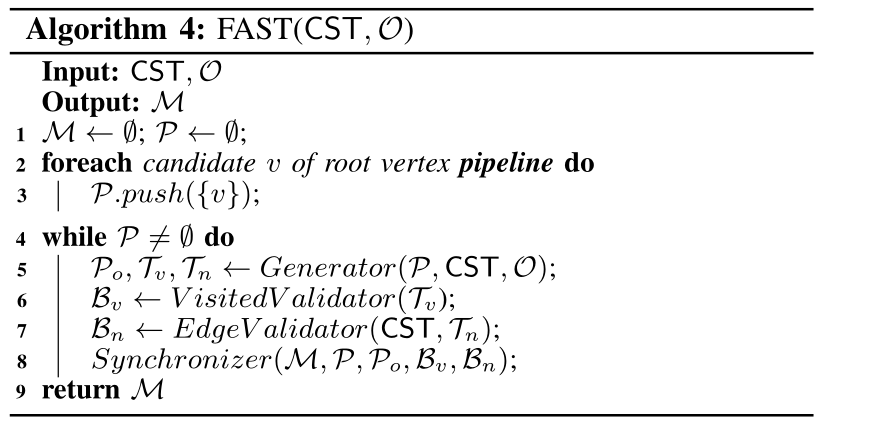


（2）例子：考虑到图4(a)和图3(a)，负载估计结果如图4(d)所示。对于叶verticesu3andu2,cu3(v9) =cu3(v10) =cu2(v6) =cu2(v7) =cu2(v8) = 1。然后我们以自底向上的方式计算u(v)，例如，cu0(v1) = (cu1(v3) +cu1(v5))∗(cu2(v6) + cu2(v8)) = 4。最后，WCST=cu0(v1)+cu0(v2) = 4+3 = 7。

二．硬件实现

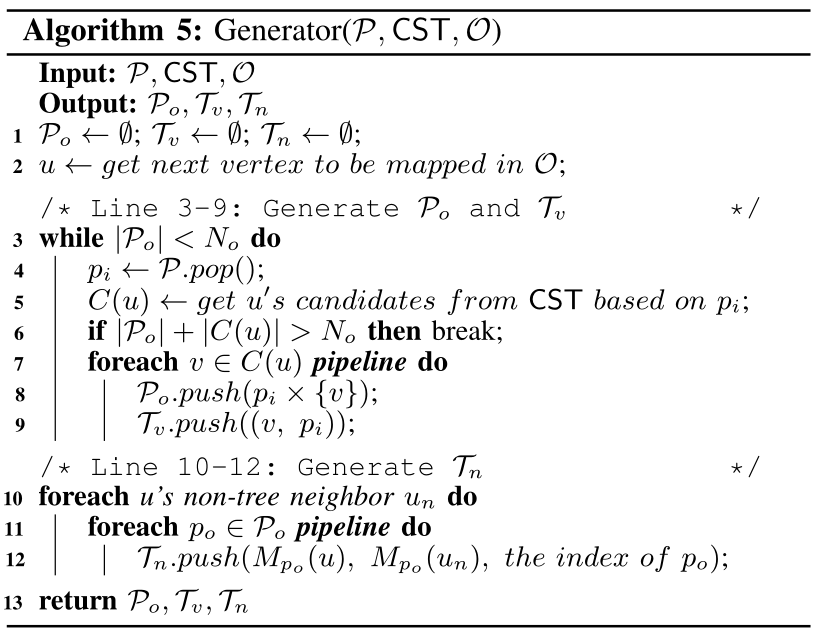
4. Basic Pipeline of Subgraph Matching

（1）描述：在典型的回溯算法[12]，[13]，[17]，[18]中，通过按照匹配顺序将下一个顶点与候选顶点进行匹配，一次扩展一个部分结果。由于迭代之间的数据依赖关系，这种顺序设计不能流水线化。为了解决这一问题，我们将匹配过程分解为三个步骤:(1)生成器按照匹配顺序对下一个顶点进行匹配，展开部分结果; (2)V验证器验证新部分结果是否有效;(3) Synchronizercollects结果。与典型的算法不同，我们的方法在这些步骤中一次处理数千个局部结果，使每一步都能充分利用FPGA的流水线机制。我们的基本管道设计如算法4所示，表示为fast。



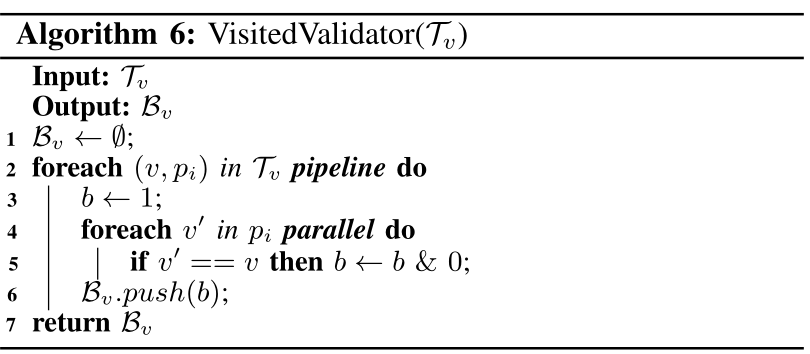
5.Generator

（1）描述：Generatoris用来展开部分结果并生成访问的validationtasksTvandedge validationtasks Tn.算法5显示了generator的工作流程。首先，我们展开部分结果并生成已访问的验证任务Tv(第3-9行)。这个过程可以完全流水线化。受片上资源的限制，我们控制每轮新扩展的局部结果的最大数量，表示为no(第6行)。我们将在VI-B节详细讨论如何选取noin的值。生成过程的内部循环是完全流水线化的(第11-12行)。对于每个新的partial resultpo，我们有一个特定的访问验证tasktv，而边缘验证taskstnis的数量由查询结构和匹配顺序决定。管道循环的一个先决条件是循环体的循环是固定的。因此，我们必须将etn生成过程与其他两个步骤分开。由于同样的原因，Tngeneration过程的外部循环(第10行)不能被流水线化。



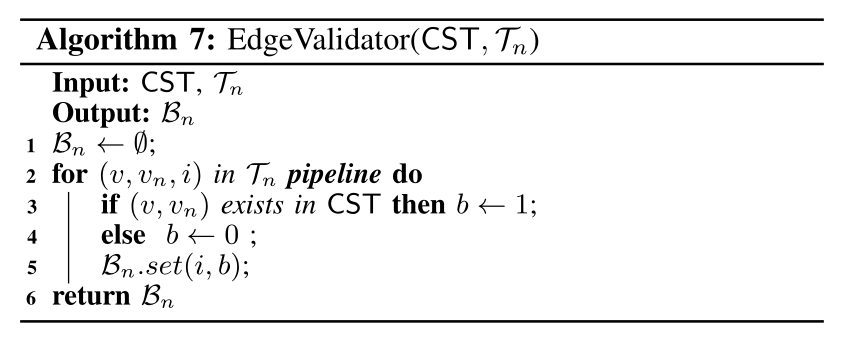
6. Visited Validator

（1）描述：如算法6所示，Visited V验证器用于验证新映射的候选顶点是否在之前被访问过，方法是将V与每个顶点inpi进行比较(第4-5行)。我们使用fpga中的阵列划分机制，即将一个阵列划分为独立的元素，以有效地增加存储的读写端口数量。该机制提供了与po的每个元素并行比较的可能性。每个poa都有两个比特来分别反映它是否通过访问验证和边缘验证。ifv已经被访问过，被访问的位被设置为零(第6行)。这个模块可以完全流水线化。



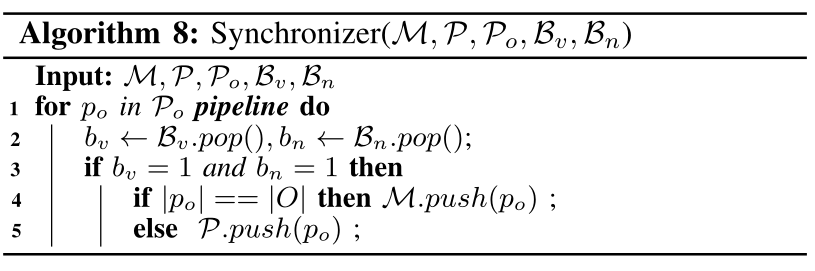
7. Edge Validator

（1）描述：如算法7所示，边缘V验证器检查新映射的候选对象是否与allvn相邻，即ofu的非树邻居的映射。它通过将vn与v的所有非树候选邻居进行比较来检查边缘是否存在(第3行)。这里我们也采用了数组划分机制，使得边的存在性检验可以在inO(1)中完成。然而，这种机制会消耗更多的片上资源，这限制了数组的最大访问端口数量，表示为asP ortmax。因此我们分区cstifdcstexceedsp ortmax。如果在vandvn之间没有边，边位被设为0(第4行)。应该注意的是，每个pomm可以有一个以上的etn，任何一个失败都会导致一个invalidpo。edge V alidatormodule也可以完全流水线化。



8. Synchronizer

（1）描述：如算法8所示，Synchronizeris被设计用来收集部分结果。对于每个po，它首先从bvandbn(第2行)获取两个验证位，如果其中任何一位为零，thispon将被丢弃(第3行)。然后比较|po|和|O|是否为一个完整的结果(第4行)。



## 实验结论

我们在一个alveso U200数据中心加速卡上实现了fasttin c++，它配备了64GB的片外DRAM, 35MB的片内BRAM，并通过PCIe gen3×16与主机通信。它在FPGA卡上以300mhz的频率运行。所有的实验都是在一台配备8核Intel Xeon E5-2620 v4 CPU (2.1GHz)， 250G主机内存，NVIDIA Tesla V100(5120流处理器，16GB全局内存)，运行Ubuntu 16.04的机器上进行的。在实验中采用LDBC社交网络基准测试(LDBC- snb)[7]来模拟真实世界的工作负载。为了评估算法，我们以毫秒为单位度量执行时间。我们为每个查询设置了3小时的时间限制。每个查询运行三次，并报告平均时间。我们将带超时的查询的执行时间表示为‘INF’，将内存耗尽的查询表示为‘OOM’。

在DG03和DG10的所有查询中，f AST-BASIC的性能都优于sf ast - dramm。尽管从DRAM到BRAM读取数据的初始开销很大，但F ast - basic与F AST-DRAMon相比，平均速度约为5.0倍。加速率接近于读延迟的比率。此外，不断增长的加速(DG01为4.50倍，DG03为5.18倍，DG10为5.93倍)也证实了初始传输开销对更大的图的整体性能的影响有所下降。这些结果表明，有必要对cst结构进行分区，以避免数据直接访问DRAM。

# 领域综述

**1. Impact Vertices-Aware Diffusion Walk Algorithm for Efficient Subgraph Pattern Matching in Massive Graphs（**[**IEEE Access 7**](https://dblp.org/db/journals/access/access7.html#AiRL19)**: 44555-44561 (2019)）**

子图模式匹配是许多应用程序的基本构建块。从哪里开始模式匹配任务以及如何进行是海量图中的基本问题。在本文中，我们提出了基于查询图和数据图的扩散演练的最大影响顶点。我们提出了一种新颖的冲击顶点感知扩散走算法，一种名为DiffWalk的分布式算法，用于子图模式匹配。我们的算法使用查询图中影响最大的顶点来定位初始搜索位置，然后通过扩散走遍历大规模数据图。我们给出了基于概率推理和光谱图的理论分析，证明从最具冲击力的顶点开始的图模式匹配可以首先防止低概率事件的比较开销，也证明了扩散走可以有效地遍历图。我们已经进行了一系列实验，证明了我们的算法在运行时间和通信大小方面的效率。

2. SUMMA: Subgraph Matching in Massive Graphs

图可以表示大量的数据类型，如在线社交网络、互联网链接、过程依赖图等。对海量图进行索引是一个迫切需要解决的具有重要现实意义的研究问题。主要的挑战是规模。每个图可能包含至少数千万个顶点。由于数据库图的大小较大，工作内存可能无法存储数据库图，这会大大增加处理时间。我们提出了一种新的基于索引的子图匹配方案，即SUMMA，用于海量图的查询。我们设计了两种新的索引，既能捕获数据库图的局部信息，又能捕获数据库图的全局信息。SUMMA通过使用匹配方案进一步优化，以减少冗余计算和磁盘访问。最后，我们利用大量的合成数据集来评估我们所提出方法的效率和可扩展性。

**3. A subgraph matching algorithm based on subgraph index for knowledge graph.** [Frontiers Comput. Sci. 16(3)](https://dblp.org/db/journals/fcsc/fcsc16.html#SunLDNC22): 163606 (2022)

子图匹配问题是图搜索中的一个基本问题，即NP-Complete问题。近年来，子图匹配成为知识图分析领域的热门研究课题，其应用范围广泛，包括问答和语义搜索。在本文中，我们研究了知识图谱上的子图匹配问题。具体来说，给定一个查询图q和一个数据图G，子图匹配的问题是在G上进行q 的所有可能的子图同构映射. 知识图形成为在一对顶点之间具有多条边的有向标记多图，它比一般图具有更密集的语义和结构特征。为了加速知识图上的子图匹配，我们提出了一种新的基于知识图子图索引的子图匹配算法，称为FGq T -Match。子图匹配算法由两个关键设计组成。一种设计是匹配驱动流图的子图索引（FGq T)，提前减少了冗余计算。另一种设计是多标签权重矩阵，它评估接近最优的匹配树以最小化中间候选。借助这两个关键设计，只需遍历FGq T即可快速进行所有子图同构映射。对真实图和合成图的大量实证研究表明，我们的技术优于最先进的算法。

**4.Subgraph Matching With Effective Matching Order and Indexing.** [IEEE Trans. Knowl. Data Eng. 34(1)](https://dblp.org/db/journals/tkde/tkde34.html#SunL22): 491-505 (2022)

子图匹配从数据图中查找与查询图相同的所有嵌入。最近的算法通过基于查询图在数据图上生成树结构索引，在树中逐个路径对顶点进行排序，并按照匹配顺序枚举嵌入来工作。然而，我们发现这种基于路径的排序和基于树结构索引的枚举本质上限制了性能，因为没有考虑跨树路径的顶点之间的边。为了解决这个问题，我们提出了一种基于成本模型生成匹配顺序的方法，同时考虑了查询顶点之间的边和候选者的数量。此外，我们为候选顶点及其在数据图中选择的邻居创建了一个双图索引，并使用此索引沿匹配顺序执行枚举。我们在真实世界和合成数据集上的实验表明，我们的方法在数量级上优于现有技术。

# 方法不足与改进

## 方法不足

1. Task Parallelism：在我们的基本流水线设计中，模块是串行执行的。由于BRAM上访问普通内存区域的端口数量有限，所以两个模块不能同时访问同一内存。因此，在所有tvandn生成之前，已访问的V alidator and edge V alidator不能启动。同步器将在所有验证任务完成之前处于空闲状态。
2. Generator：在我们的任务并行化版本中，tn生成过程需要等待ltvgeneration过程完成，从而降低了整体吞吐量。同步器也会等待fedge V校验器的输出，尽管所有访问的poare位都准备好了。

## 方法改进

1. Task Parallelism： 我们利用FPGA上的任务并行机制来允许模块并行执行。与循环并行性不同，当部署任务并行性时，允许不同的执行模块同时操作。任务并行性是通过利用模块之间引入的额外缓冲来实现的。该缓冲器在FPGA上由fifo (First in, First out)实现。每个模块的输出将被流到缓冲区中，只要缓冲区不是空的，下一个模块就会处理数据。如算法5所示，一旦生成cetvis(第9行)，它就被流到FIFOs，然后visited V alidator开始工作。类似地，一旦生成了cetnis(第12行)，它就会流到FIFOs，然后dedge V验证器启动它的过程。当它的两个验证位准备好后，将很快被synchronizeras收集。
2. Generator：为此，我们对Generatormodule进行了优化。Generatormodule被拆分为totvgeneratorandtngenerator。一旦生成一个newpois，它将被复制，以便pocan的源和副本分别流到两个生成器的不同fifo。当同步器同时开始收集部分结果时，thtvgenerator和tnGenerator都可以开始工作。这种优化是通过复制数据和使用更多的片上资源(如fifo)来实现的。

## 改进结果理论分析

1. Task Parallelism：考虑这个任务并行化版本的总周期，标记为astask。在这个优化设计中，Generator的第一个循环(算法5中的3-9行)和visited V验证器(算法6)并行执行。第二个循环是Generator(算法5中的10-12行)、Edge V alidator(算法7)和synchronizer(算法8)并发执行。为了简化这个方程，假设我们选了一个合适的No。(3)与式2相比，该优化方案理论上性能提高了50%。

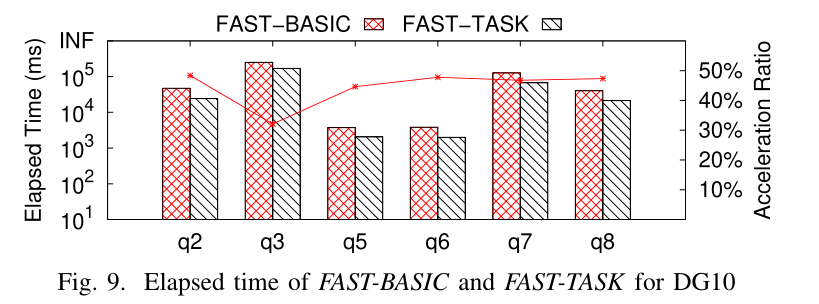
2. Generator：由于FPGA的循环并行特性，副本的成本不会降低性能。我们分析了这个优化版本的总循环。所有模块同时执行。(4)与方程3相比，该优化方案理论上性能提高最多可达33%。

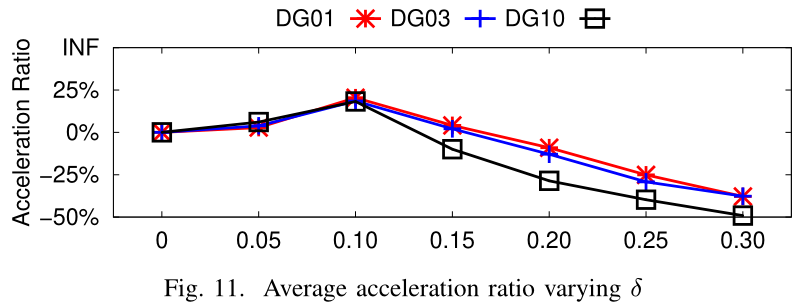
## 改进结果实验验证

### 实验设计

在实验中采用LDBC社交网络基准测试(LDBC- snb)[7]来模拟真实世界的工作负载。为了评估算法，我们以毫秒为单位度量执行时间。我们为每个查询设置了3小时的时间限制。每个查询运行三次，并报告平均时间。我们将带超时的查询的执行时间表示为‘INF’，将内存耗尽的查询表示为‘OOM’。

### 实验结果





### 实验结果分析

1. Task Parallelism：可以看出，任务并行优化可以达到50%的提升(如q8)。任务并行性的理论改进将在第VI-C节中讨论。由式(2)和式(3)可以看出，对于misis大于n的密集查询，任务并行度优化获得了更好的性能。q3的加速比比其他查询要低得多，因为它的N M要高得多(q3约为2m，其他查询约为1m)。
2. Generator：任务并行性允许所有模块同时执行。但是，受第一个modulegenerator的限制，它只能并行生成两种任务，这样以后的模块就可以同时开始工作了。F ast - sepp通过使用更多的片上资源和复制数据来解决这个问题。与AST-TASK的平均运行时间相比，ast - sepf达到了大约35%的改进(例如q8)。任务生成器分离的有效性与我们在公式3和公式4中的循环分析是一致的。此外，当n M>1时，任务生成器分离实现了最好的改进。

# 附录

[1]L. Ai, L. Ramaswamy and S. Luo, "Impact Vertices-Aware Diffusion Walk Algorithm for Efficient Subgraph Pattern Matching in Massive Graphs," in IEEE Access, vol. 7, pp. 44555-44561, 2019, doi: 10.1109/ACCESS.2019.2908930.

[2] X. Jin, Z. Yang, X. Lin, S. Yang, L. Qin and Y. Peng, "FAST: FPGA-based Subgraph Matching on Massive Graphs," 2021 IEEE 37th International Conference on Data Engineering (ICDE), 2021, pp. 1452-1463, doi: 10.1109/ICDE51399.2021.00129.

[3] R. Chang, A. Podgurski, and J. Yang. Discovering neglected conditions in software by mining dependence graphs. Proc. of ISSTA, 2008.

[4] S. Sun and Q. Luo, "Subgraph Matching With Effective Matching Order and Indexing," in IEEE Transactions on Knowledge and Data Engineering, vol. 34, no. 1, pp. 491-505, 1 Jan. 2022, doi: 10.1109/TKDE.2020.2980257.

[5] Sun, Y., Li, G., Du, J. et al. A subgraph matching algorithm based on subgraph index for knowledgegraph.