

**课程报告**

**课程名称：高级计算机系统结构**

**专业班级： 计算机硕2101班**

**学 号： M202173535**

**姓 名： 蒋子涵**

**授课教师： 曹强**

**报告日期： 2022.1.5**

**计算机科学与技术学院**

# Large-Scale Graph Processing on FPGAs with Caches for Thousands of Simultaneous Misses

## Abstract

高效的大规模图计算对很多领域来说都很重要。虽然图算法有大量并行的机会，但其性能却由于不规则的内存访问而受限于内存系统。最先进的FPGA图处理器，如ForeGraph和FabGraph，通过使用scratchpads和定期从DRAM中流出边来解决内存问题，但它们最终会在不需要的数据上浪费带宽。在本文中，我们采用了极端的非阻塞式高速缓存，可以处理数以万计的未处理缺失，极大地提高了存储器系统将多个加速器访问合并成更少的DRAM存储器请求的能力，以很小的成本带来了从一个非常大的高速缓存优势。我们用一个在亚马逊AWS f1上运行的可适应图加速器进行实验，在大型图上运行经典算法（PageRank、SCC和SSSP），与最先进的FPGA加速器相比，我们实现了3倍的几何平均加速，与多核CPU相比，带宽效率高1.1-5.8倍，功率效率高3.0-15.3倍，而且我们在GPU上支持的图比最先进的大得多。

## INTRODUCTION

图是很多领域内最高效的数据表示，图算法虽然可以并行，但由于对顶点和边的不规则访问，传统平台上的图算法性能受限。使用GPU运算需要大量的预处理；ASIC可以通过定制流水线和内存系统来提供杰出的性能，但制造周期长；FPGA性能不如ASIC，但部署成本低，且保留了ASIC的硬件的灵活性，使其成为加速算法的优质平台。

### The Challenge of Irregular Accesses

对于大规模图来说，以边为中心的模型比以点为中心的更为常见。考虑到边集通常比点集大，对边缘进行分流确实将不规则内存访问的范围限制在这些集合中最小的一个。如果整个点集都能被加载进片上内存，对外部内存的随机访问就被完全消除了。在一般情况下，根据源节点和目的节点对对边进行排序能将其序列化，但预处理的代价同样很高。由于不规则访问使传统cache效率低下，因此最前沿的FPGA加速器通过对点集进行划分来缓解这个问题，且只需将边也按照源和目的节点进行划分，复杂度远小于排序。然而，由于每次迭代不是所有节点都被访问，且划分的数量是节点的平方，会造成大量冗余的数据传输。

### Optimizing Miss Handling, Not Maximizing Hits

缺失优化内存系统（MOMS）能在访问不规则时提高DRAM带宽。其思想为最大限度地减少停顿，重复使用每次内存响应以尽可能多地处理待处理的缺失。通过增大最大缺失值，MOMS可以在不需要更大cache的情况下最大化数据重用性，接近理想cache系统。

### A Multi-Die FPGA Graph Accelerator

本文的主要思想是用一个MOMS来支持图处理中的不规则读访问。我们的想法是，边分布的不均匀性带来了许多请求合并的机会，因为一些节点的请求频率比其他节点高几个数量级。低度节点的普遍存在导致了传统cache的频繁停顿，而MOMS可以更好地容忍大量缺失。而且利用MOMS的动态、细粒度的操作，可以避免划分的冗余传输。由于MOMS只处理读请求，我们仍然将目的节点缓存在片上储存中，但现在静态调度节点传输是划分的线性倍而不是平方倍。

我们的图加速器适用于各种图算法，基于乱序处理单元（PE）的多线程并行。考虑到现在的多片FPGA，还需要对MOMS进行改进，将其扩展至私有和两层结构来适应大型FPGA。我们的图加速器在亚马逊AWS f1实例上的Xilinx UltraScale+ FPGA进行了评测，相较于最新的FPGA加速器达到了3倍的加速，相较于CPU上的框架(Ligra和GraphMat)提高1.1-5.8倍的带宽和降低3.0-15.3倍的能耗，相较于GPU上的框架(Gunrock)有4.7倍的几何加速。

## EXTENDING MISS-OPTIMIZED MEMORY SYSTEMS

Cache对图处理来说是低效的，非阻塞型cache通过暂时将缺失地址和请求ID储存到MSHR中而不停顿，来增加并行内存操作，实现更高的内存级并行。然而MSHR的大小是有限的，虽然真实的处理器突破这个限制没什么益处，但面向吞吐量的加速器能发射上千种这样的读请求，因此也许会从中受益。

本文使用的MOMS，是一个多bank非阻塞cache的优化版本，通过将MSHR储存进RAM中以及使用cuckoo哈希来支持数以万计的缺失。这样能大幅增加内存级并行，且FPGA能让它简单地被实现。

我们之前的工作只考虑多个加速器之间共享的MOMS，但没有考虑MOMS bank冲突，这会大量限制吞吐量，尤其是在大型系统上。为了解决这个问题，我们提出私有和两级MOMS。在两级MOMS中，我们用私有MOMS来减少共享MOMS中的堵塞，并最终减少冲突和改善吞吐量。并且我们之前只在连接单个DDR3控制器的中型，单片FPGA上设计和测试，而第四节中我们会介绍一种技术，能高效将其扩展至连接4个DDR4通道的大型多片FPGA上。

## GRAPH PROCESSING MODEL

图G包含点集V，大小为N，边集E，大小为M。考虑G为有向图，我们考虑将一个值与每个节点相关联的图算法并在固定的迭代次数中迭代更新，或直到收敛为止。

### Graph Partitioning

我们采用一个以边为中心的模型，遍历整个边集，乱序同时访问源节点和目的节点。将点划分为Q个不相交的部分，将边划分为$Q^2$个块，第$E\_{i→j}$块包含所有源节点和目的节点分别在Si和Dj中的边。

由于我们采用MOMS来避免将源节点缓存进片上内存中，因此我们原则可以只根据目的节点将边分为Q块。但我们还是按照源节点划分，出于两个原因：（1）避免处理源节点所在块不包含上一轮迭代中更新过的节点的边；（2）应用边压缩技术。如图1所示，基于点分块Qs和Qd，我们将边分为Qs×Qd块。

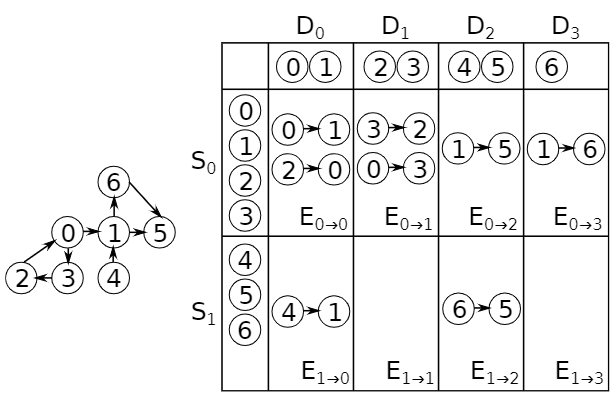
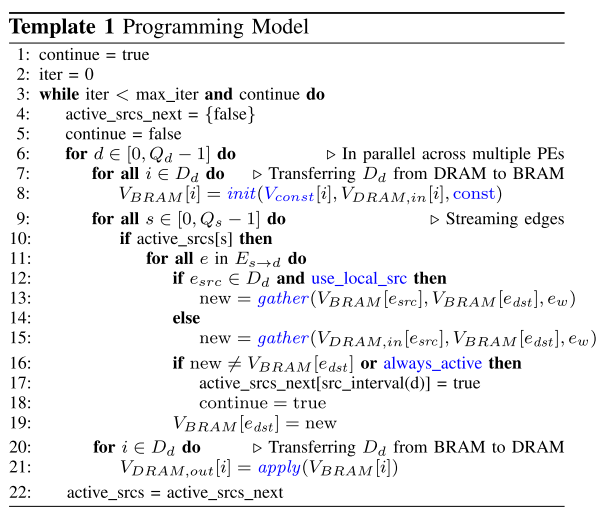


图1 对左侧图的图分块

### Programming Model



模板1表示我们的加速器实现的编程模型。它表示一个可配置为各种图算法的执行框架，只需自定义函数init()，gather()，apply()，初始节点值VDRAM,in，每个节点的常量向量Vconst，一个全局常量const，和两个控制标识use\_local\_src和always\_active。模型基于以边为中心的GAS模式，通过init()函数启用一些算法中额外的优化。

该模型支持同步和异步执行。对于同步执行，VDRAM,in和VDRAM,out在每次迭代后会交换，表示执行期间读取的点值只在每次迭代最后更新。对与异步执行，VDRAM,in和VDRAM,out指向内存中的同一数组，因此15行的gather()函数会在它们出现在DRAM中时立刻读取更新值。

### Graph Encoding and Memory Layout

我们的加速器接受COO格式的图，每个边表示为元组(src, dst, weight (可选))。

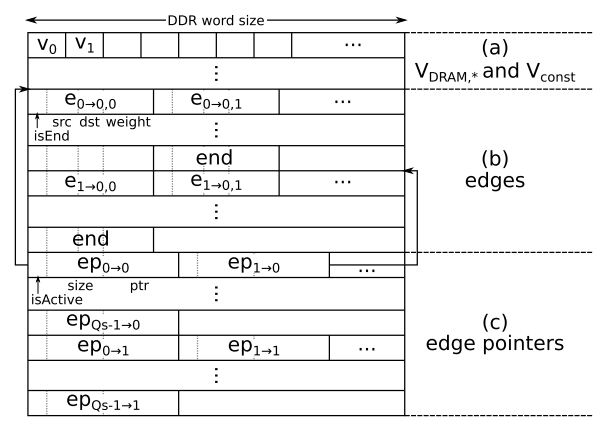


图2 图在内存中的布局

整体内存布局如图2所示。第一部分包含点数组VDRAM,in，Vconst和为同步模式下VDRAM,out分配的内存。第二部分是按照分块组织的所有边。由于每个从DRAM中获取的字足够长来包含多条边，因此我们在每个边块的末尾增加了一条特别的终止边，来保证PE会无视最后一个DRAM字中剩下的数据。我们每条边总是使用32比特，其中15比特为目的节点偏移，16字节为源节点偏移，1字节为isTerminatingEdge标记。对于权重边，源节点和目的节点后面就是边的权重。由于每个块的边数量是任意的，因此我们用一个边指针来指示每个块的起始地址和大小。

## SYSTEM ARCHITECTURE

### Die Crossing Logic

一个高性能FPGA由多个片和FPGA板组成，支持多片对于完全利用所有的资源很重要。多片设计必须考虑：（1）最少化片间联系；（2）确保所有的片间联系在两端都已注册且没有中间件。我们的片间逻辑如图3所示。

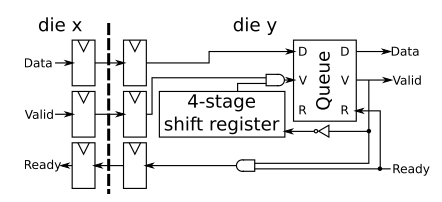


图3 片间交叉电路

### Top-Level Architecture

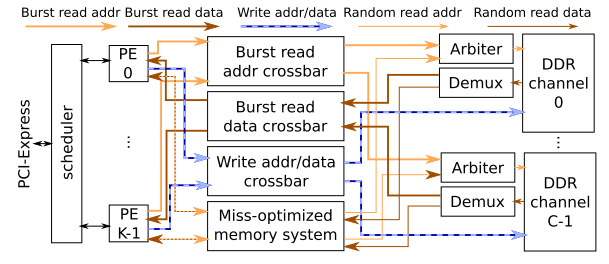


图4 系统顶层架构

图4展示系统的顶层架构。调度器包含内存映射寄存器，用于传递配置参数和启动加速器，以及告知主处理器任务完成。执行期间，PE从调度器中获取任务，每个任务与一个目的节点集合关联。为了最大化资源利用率，PE分散在多个片上，PE和DRAM之间有两条不用的路径：一个多片感知MOMS，用于通过边来获取源节点的值，以及一个用于突发读写请求的逻辑单元。这种互连逻辑是通过以下方式实现的：(1)将三个crossbars分割成每个片的第一个crossbars，来将请求路由到相应的片上。(2)每个芯片的一组仲裁器，将请求转发到同一片的适当的资源处，如图5所示。

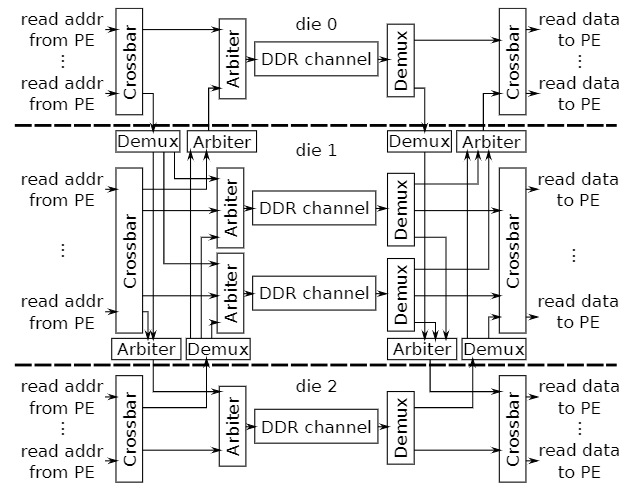


图5 处理突发读请求的多片感知互联结构

我们的多片感知MOMS与原始的MOMS有两点不同：（1）从MOMS出发或到MOMS的连接使用图5所示的逻辑单元。（2）每个bank被静态分配给一个DRAM通道。

除此之外，我们也考虑了一种结构，该结构（1）只有每个PE私有的MOMS，（2）级联私有MOMS和一个共享MOMS，就像一个两级cache。所有结构如图6。私有MOMS能被每个PE无冲突地访问，两层结构结合二者的优势，提供最高的性能。

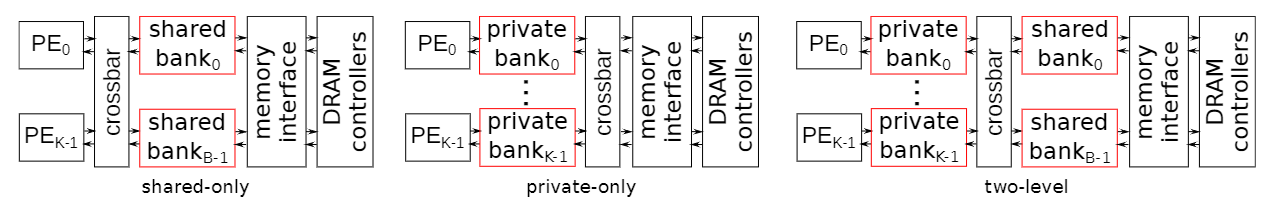


图6 MOMS的整体结构

### PE Architecture

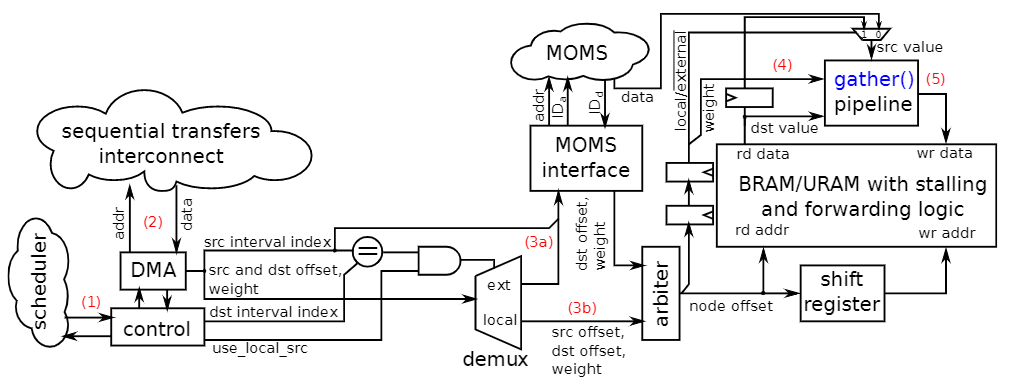


图7 PE结构

图7展示了一个PE的内部结构，每个PE包含一个DMA单元来处理序列数据传输。在接受一个任务之前，PE会先读取目的节点块中所有节点的初始值。为了最小化初始化时间，我们每周期写4个节点值。一旦节点初始化完成，PE请求边指针并开始请求边。对于每条边，源节点值要么通过MOMS从DRAM中获得，要么直接从本地BRAM中获得。如果use\_local\_src为真，并且源节点在当前的目标节点区间内。一旦源节点的值由MOMS或BRAM获得，它就会和边状态一起被发送到gather()流水线。因为写入gather()流水线的输出是对目的节点的读-改-写操作，所以我们使用转发或停顿逻辑来确保gather()流水线总是收到目的节点值的最新版本。对于always\_active为假的算法（如SCC和SSSP），gather()流水线也会返回一个更新的标志，只要目标节点被更新，该标志就会被设置，用于实现模板1的第16行。

一旦所有的边都已流入，目标节点的内存就会被写回DRAM，并且PE将作业的完成情况与目标组的更新标志一起通知给调度器。

### Handling Efficiently Out-of-Order Responses

当数据在多个DRAM通道上交错时，即使每个通道的响应都是有序的，但只要碰到多个通道，响应就可能乱序返回。由于节点必须按照特定的顺序进行初始化，为了防止乱序响应并避免昂贵的突发重排，PE将永远不会为初始节点值发出超过一个未完成的读请求。我们发现，对于这个问题只需使用一个64个行、512位宽的DMA队列，并在队列有足够的空间容纳时立即发出下一个32位请求。另一方面，边的请求可能比32位更短，因为块中的边数不一定是32位请求中边数的倍数，而且我们发现将每个PE限制在一个未处理的边请求中会导致过于频繁地读取空队列。然而，与节点初始值不同，边可以乱序流入，只要每个请求与相应的源节点块匹配，因为在我们的压缩边格式中，它定义了源节点的高比特位。因此，我们给每个边突发请求标记一个ID，这个ID对每个源节点块来说是唯一的，并使用与边数据一起返回的ID将正确的边前缀流向下层逻辑。

由于每条边可以独立处理，因此我们将其作为独立的线程：当源节点的数据请求被发送到MOMS时，我们存储线程状态（目的节点偏移量和边的权重）并暂停；当响应返回时，我们检索相应的状态并恢复线程执行。这一机制由MOMS接口和MOMS本身实现，如图8所示。

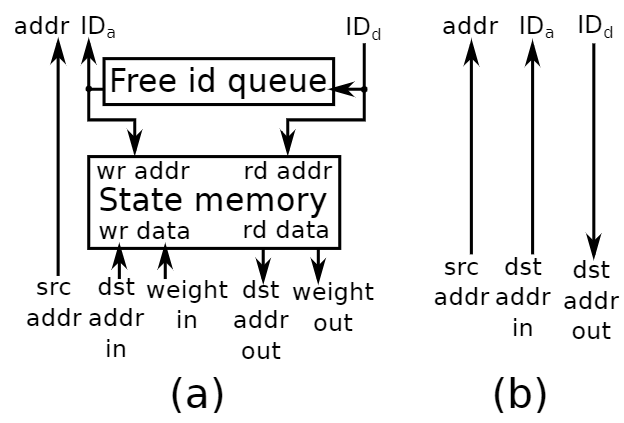


图8 可用的MOMS接口

对于权重图，将每个请求用一个由ID队列提提供的ID标记，也用来储存和接收状态内存中的目的节点偏移和边权重。对于无权重图，状态就减少到15位的目的节点偏移。如果用图8(a)的结构，将每个目的节点偏移与ID配对需要3×IDsize比特，我们通过直接将目的节点偏移作为ID来将其降低到IDsize比特，如图8(b)所示。这样最大线程数就只受限于MOMS容量而不是MOMS容量与状态内存的最小值。

### Node Reordering

COO格式的图表示隐式地给每个节点赋予了一个随机的整数编号，但对性能有很大影响。将相互连接的节点存放在内存中邻近的位置能提高cache命中率，或MOMS的内存相应重用性。

另外，不同目的节点块的任务通常是并行执行的，因此希望在给目的节点分块时尽量使入边数量平衡。ForeGraph和FabGraph发现将连续的节点分为同一块会导致严重的负载不均衡，因此他们提出用基于哈希的标记，改善负载分配。

然而在我们的情况中，由于任务是被动态调度到PE的，因此平衡问题没有那么重要，在没有哈希标记的情况下也能达到负载均衡，且哈希分块会破坏cache复用性，因此我们保持原有的cache行并将其哈希到目的节点块中。

除此之外，我们也评测了一种称为DBG重排的技术，来处理那些初始标签没有保留紧密连接关系的图。DBG粗略地根据出度将接节点分为8组，其思想为将高出度的节点分组会得到更高的cache行复用率。

## EXPERIMENTAL RESULTS

我们在本节分析了不同PE数量以及不用MOMS结构在PageRank，SCC和SSSP算法中的影响。我们也分析了不同的预处理技术，内存通道的数量，即带宽对性能的影响和代价。我们也将性能与最新的CPU、GPU和FPGA框架做了对比。

### Experimental Setup

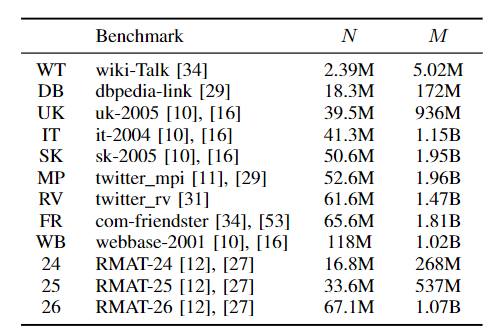
该系统用Chisel由RTL编写，用Vivado 2019.1仿真，并在亚马逊AWS f1实例上执行，该实例有一个Virtex UltraScale+ FPGA，通过PCI连接到本地PC以及4个16GB DDR4通道，每个通道理论带宽为16GB/s，但实际带宽大约在8GB/s。

我们的FPGA有3片芯片，大约25-35%的底层和中央SLR资源为shell保留。中央SLR有两个内存控制器，另外两个SLR各有一个。我们把共享的MOMS crossbar分配到中央SLR上，把每个bank分配到各自内存通道的SLR上。我们发现，将30%、15%和55%的PE分别分配给底部、中央和顶部的SLR，可以提供良好的空间平衡。私有的MOMS，如果存在的话，被分配到与各自的PE相同的SLR上。

每个PE在URAM中持有32,768个目标节点；每个节点在SCC和SSSP中需要32位，在PageRank中需要64位。对于在单精度浮点上操作的PageRank PE，我们使用Vivado HLS实现了gather()和apply()函数。由于gather()流水线有4个周期的延迟，它可能不得不停顿下来以处理RAW冲突。用Chisel实现SCC和SSSP的gather()函数对32位无符号整数进行操作，且是全连接的，这意味着不需要停顿。SSSP PE的状态存储器和空闲ID队列有8,192个槽，并且是用BRAM实现的。我们将PageRank 执行10次迭代，其他算法计算到收敛。

作为benchmark，我们使用了一组真实世界合成的大图，其主要属性总结于表1。对于SSSP，我们添加了0到255之间的随机整数权重。如果没有指定，我们同时启用哈希和DBG。

表1 benchmark属性



### Architecture Exploration

我们进行了大量的设计空间探索，结果如图9所示。

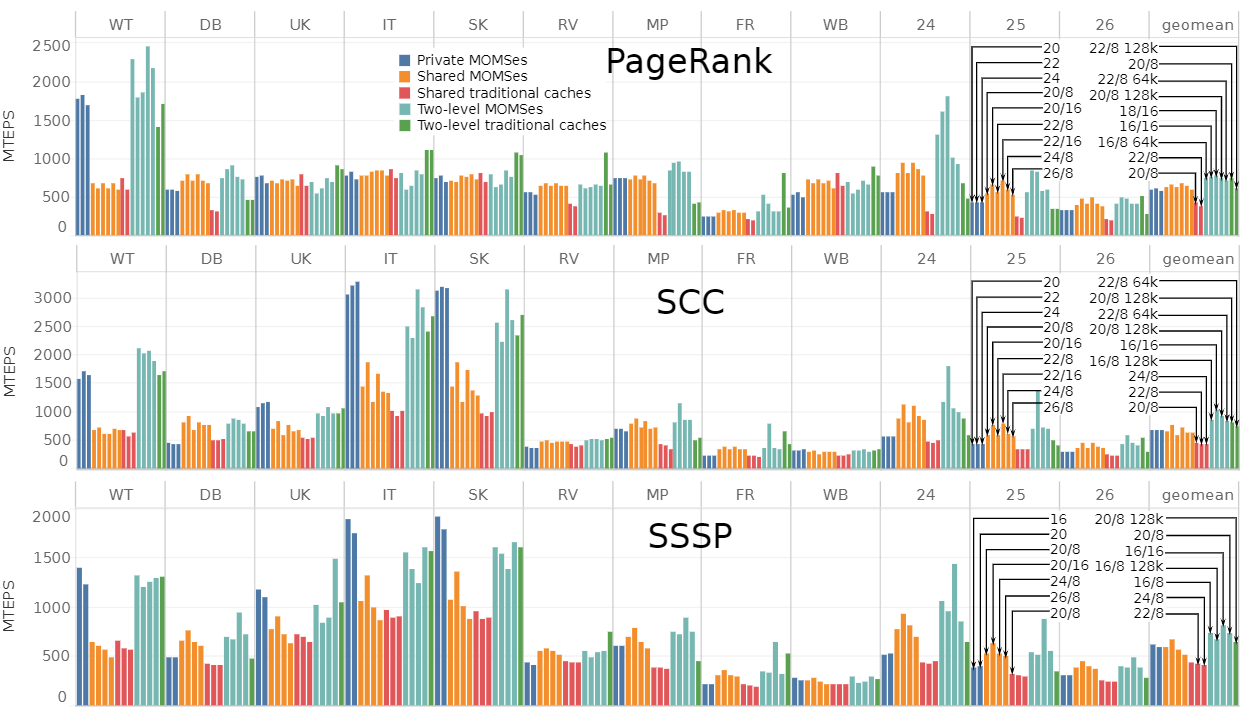


图9 不同结构下PageRank，SCC和SSSP算法的吞吐量

如图所示，两层结构提供了最高的性能，因为到达共享MOMS的请求更少了，带了更少的冲突，同时提供PE之间的复用。16bank的结构胜过8bank，说明PE间的冲突仍然严重。通过减少bank冲突，私有和两层MOMS相比于共享MOMS提供了3倍以上的吞吐量。仅仅私有MOMS受限于请求处理数。

SCC在所有算法中获得最高性能，PageRank由于4周期gather()受限于RAW停顿，对于SSSP，（1）状态存储器和自由ID队列的开销降低了并行性和操作频率；（2）权重边比非权重边消耗两倍的带宽。

图10展示了图11结构上SCC吞吐量与cache命中率的关系。传统的架构需要高命中率才能达到其峰值性能，而MOMSes尽管命中率较低，但其性能往往超过它们，这表明cache行在MOMSes中不那么关键。为了验证这个假设，我们还考虑了同样的系统，所有的cache行都被停用，因此总是达到0%的命中率。虽然传统的高速缓存自然会失去其所有的性能，但MOMSes在大多数基准上几乎没有吞吐量的下降，这意味着当延迟不重要时，成千上万的MSHRs基本上可以取代高速缓存阵列，而其面积成本只是一小部分。



图10 SCC吞吐量与cache命中率的关系

### Preprocessing Cost and Impact

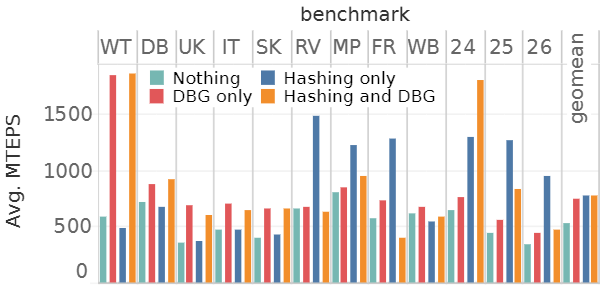
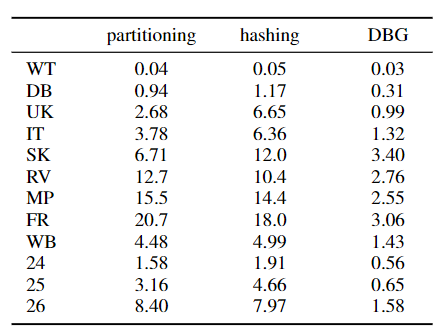


图11 在18/16 MOMS上不同预处理技术下的PageRank性能

图11展示了在18/16 MOMS上不同预处理技术下的PageRank性能。大多数benchmark受益于哈希，特别是那些节点较少的benchmark，这导致了较少的任务和更改善的负载平衡。没有哈希的较高速度表明，在这些情况下，将连接最多的节点分组在同一目标点区间，序列地处理它们的边，比拥有统一的任务大小更重要。此外，当标签不保留原始图块（FR、MP、RV和RMATs）时，使用DBG提供了一个显著的速度提升。

表2显示了在20核Intel Xeon E5-2698上所有benchmark的预处理时间，不包括磁盘I/O。我们使用OpenMP来并行化大部分操作。我们的预处理通常是轻量级的，除了分区之外，所有的步骤都是可选的，可以用预处理时间换取运行时的效率，或者快速探索预处理的设计空间，使给定的应用程序的性能最大化。

表2 秒级预处理时间



### Memory Bandwidth Scalability

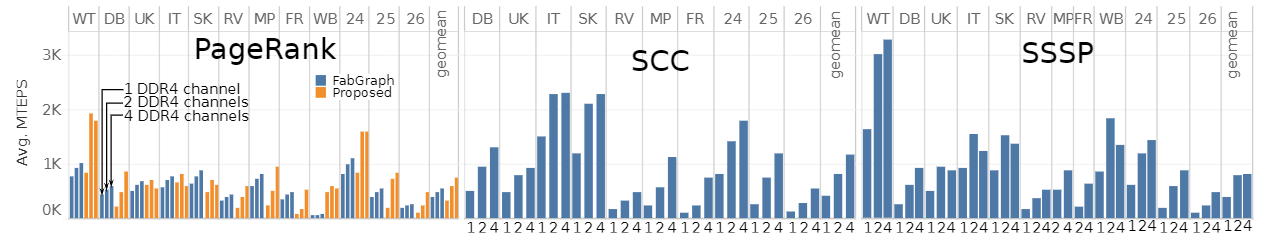


图12 两层16/16 MOMS和FabGraph上的PageRank下吞吐量于DDR4通道数的关系

图12展示了两层16/16 MOMS和FabGraph上的PageRank下吞吐量于DDR4通道数的关系。我们为它计算了一个非常理想的估计，可以确定两类benchmark：(1) 计算约束型（IT、SK、UK、WB和WT）和(2) 内存约束型（所有其他的）。第一类benchmark具有良好的位置性，需要少于4个通道来达到峰值性能，但受到PE并行性的限制，在PageRank中，由于gather()函数有4个周期的延迟，所以会出现RAW冲突。这些确实是图9中从私有MOMS或传统cache中受益最大的benchmark。在PageRank和SSSP上，一些计算约束的benchmark在4通道系统上甚至降低了它们的性能，因为它们在较低的频率下运行，这是因为使用所有的SLR导致较多的SLR交叉。相反，内存约束benchmark的性能基本上与内存带宽呈线性关系。在几何意义上，FabGraph在一个内存通道上的表现比我们的系统好，但扩展性不理想，因为性能变得越来越受其L1和L2高速缓存之间的内部带宽的限制，在大型图上的传输特别多。此外，作为一个纯粹的模拟分析，它没有考虑到SLR是如何影响路由拥堵的。

### Impact of Caches

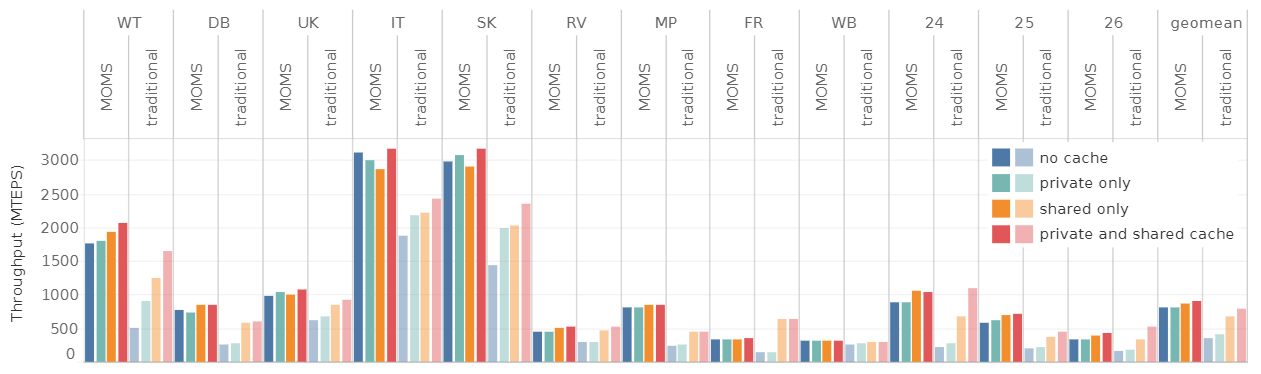


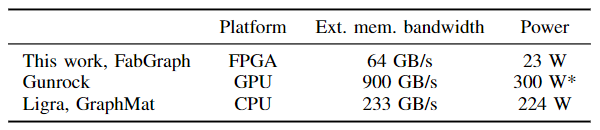
图13 20/8 两层MOMS和传统cache上有或没有私有和共享cache对SSC吞吐量的影响

图13通过比较两级20/8 MOMS和传统缓存的SCC吞吐量（分别有和没有2.5 MiB和2 MiB的私有和共享缓存），更详细地显示了增加cache行和/或大幅扩大MSHR阵列对性能的影响。传统的cache在没有cache行的情况下有2.2倍的吞吐量下降，但MOMS的性能下降只有10%，这意味着MSHRs在实践中可以取代cache行，在吞吐量方面没有什么差别。事实上，无缓存的MOMS存在时，共享缓存通常比私有缓存更有用，但IT和SK除外，它们拥有大量的私有复用机会。事实上，在这些benchmark测试中，无缓存的MOMS比只有私有或共享缓存阵列的MOMS表现得更好。当有一个共享的高速缓存阵列时，共享的命中率将比失误率低得多，这给私有MOMS积累二次缺失的时间更少。因此，私有MOMS需要处理更多的响应，这就减慢了请求的吞吐量，因为两者都在竞争同一条流水线。

### Comparison with the State of the Art

对每个图计算算法，我们考虑：（1）具有最高几何平均吞吐量的两个架构-预处理组合，以及（2）在给定benchmark上具有最高性能的架构-预处理组合。我们认为方案(1)代表了通用的、可能经过加固的图处理器，而方案(2)显示了通过利用FPGA的可重编程性和使用为特定情况高度优化的架构所能实现的最高性能。我们将其与(a)FabGraph, (b)Gunrock, (c)Ligra,和(d)GraphMat进行比较。表3总结了带宽和能耗比较。

表3 内存带宽和能耗



结果如图14所示。在PageRank上，我们的通用架构比原始的Ligra、FabGraph和Gunrock分别高出2.1倍、1.4倍和2.1倍（geomean），而专用架构的速度分别提高到4.5倍、3.0倍和4.5倍，比GraphMat和Ligra加DBG分别提高了1.3倍和1.9倍。在SCC和SSSP上，我们的架构在绝对值上仍然比CPU有竞争力，而且带宽效率高1.1-3.5倍（通用），2.3-5.8倍（专用），功率效率高3.0-9.4倍（通用），6.1-15.3倍（专用）。在SSSP上，Gunrock通过在单个节点的粒度上保持对前沿的跟踪，而不是更大的源节点分块，实现了出色的性能；然而，由于只有16GB的内存，它只能运行五个最小的benchmark，而在相同的内存量下，我们可以运行除FR和MP以外的所有benchmark。

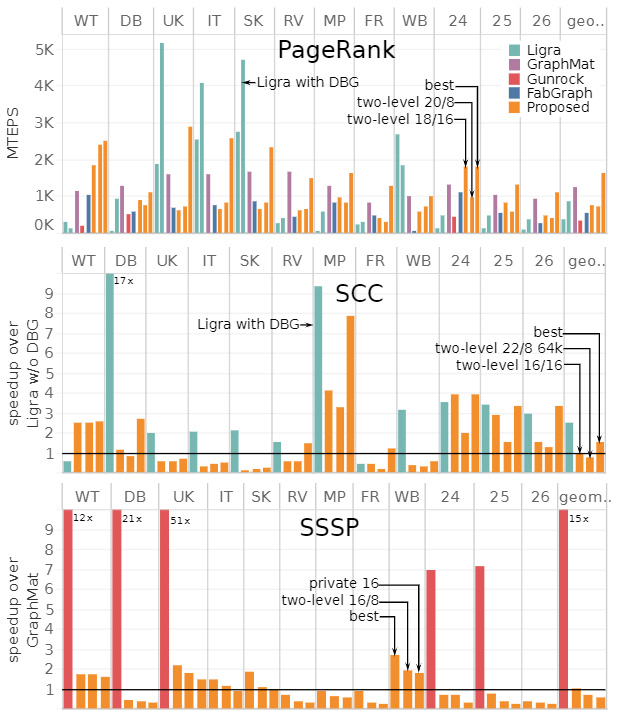


图14 与最新的CPU，GPU，FPGA框架的比较

### Resource Utilization

图15显示了性能最高的设计的资源利用率。虽然LUTs和FFs主要用于互连，但BRAMs和URAMs都用于PE和MOMSes。一般来说，DSP的利用率很低，甚至在浮点PageRank中也是如此。我们报告的是整个未被shell占用的区域的平均利用率；每个SLR的利用率更高，在两级16/16 PageRank系统中，中央SLR的LUT达到了90%的峰值，而没有明显影响工作频率，所有这些设计的工作频率保持在196 MHz和227 MHz之间。

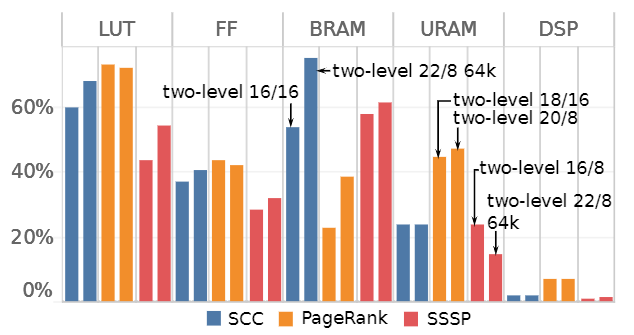


图15 资源利用率

## RELATED WORK

在CPU上，GraphChi是一个乱序图处理系统，它首先引入了分块的概念，将随机访问限制在一个较小的范围内，可以在主存中缓存。X-Stream引入了以边为中心的分散-聚集模型，其中边是流式的，不需要排序，只需要分区。内核处理的框架包括GraphMat、Galois和Totem，该框架也支持CPU-GPU混合系统。Aasawat等人报告说，在具有28个内核、240 W TDP和136 GB/s的内存带宽组合的双插槽英特尔至强E5-2695 v3上，Galois、GraphMat和Totem分别为RMAT-24上的PageRank提供了1.3、1.8和9.0 的GTEPS。我们用一半的DRAM带宽和15倍的功率实现了1.8 GTEPS。Galois和Totem都只支持CSR格式的图，其中边是按源节点排序的。在预处理过程中，Totem也按顶点程度对边缘进行排序。我们的方法不需要任何边的排序，但需要更快的线性时间分区。GPOP是一个缓存、工作和内存高效的框架，它也不需要任何边排序，并实现了比Ligra和GraphMat的显著加速。当在表二中列出的RV和FR基准上运行PageRank、SSSP和SCC时，我们的最佳架构要快0.22-5.0倍，能效要高2.2-49倍。

GPU的内存带宽至少比FPGA大一个数量级；然而，将图典型的不规则工作负载和内存访问纳入GPU的SIMD执行模型是一个挑战。例如，CuSha采用的一种解决方案是依靠离线预处理来平衡工作负载和分组内存访问，这可能构成动态或仅使用几次的图的相关开销。Gunrock将这种开销转移到运行时，而Tigr则使用较轻的离线预处理，将不规则的图转换成等价的、更规则的图。在21M和59M节点的图上，与Gunrock相比，Tigr在PageRank上实现了最多10%的速度提升，这比我们的系统在相同应用上对Gunrock的1.5-12倍的速度提升要低得多。

ASIC提供了比FPGA高一个数量级的时钟速率、密度和能源效率，但代价是NRE成本和制造时间大大增加。例如，Graphicionado在RV图上实现了4.5 GTEPS的PageRank和0.2 GTEPS的SSSP，其内存带宽与我们的相似（我们分别实现了1.5和0.7 GTEPS），而GraphDynS在RMAT-26上实现了超过85 GTEPS，其带宽仅比我们DDR4大8倍。这两个解决方案的时钟都比我们的快4-5倍，并且使用的片上存储器也比我们多得多：64 MB和32 MB，而我们最多只有9 MB。

在FPGA上，FabGraph代表了在单个FPGA上进行大规模图处理的最先进水平，只需要进行线性复杂度的预处理。它的扩展，FabGraph+，侧重于在FPGA DRAM不能存储整个图时优化PCIe传输，FPGP也解决了这一问题，它与在其专用DRAM中有效处理一次图是正交的。ForeGraph使用与FabGraph非常相似的模型，但也支持多FPGA处理。HitGraph在RMAT-24上的表现优于我们的系统，但在WT这样的极度稀疏的图上却不尽如人意，此外，还需要按目的节点对边进行排序。除了FPGP，其在TW上的BFS性能比我们在同一图上更复杂的SSSP的带宽效率低1.7倍，所有这些工作都只在模拟中进行了测试，没有解决现代大型FPGA的多ie分区相关的挑战。

## CONCLUSION

图处理是不同领域应用的关键技术；然而，由于不规则的工作负载分布、控制流和存储器访问，特别是当图很大时，实现良好的性能是具有挑战性的。我们表明，FPGA是解决前两个问题的一个有吸引力的、可获得的选择，而缺失优化的存储器系统有助于在具有数千万节点和数十亿边的图上最大限度地利用存储器带宽。我们在PageRank、SCC和SSSP上演示了我们的方法，与FPGA上的先进技术相比，实现了3倍的几何平均速度，与多核CPU相比，带宽效率高1.1-5.8倍，功率效率高3.0-15.3倍，与参考GPU实现相比，能够扩展到更大的大规模图。据我们所知，我们的系统是第一个可以在多片FPGA上运行图处理的系统，扩展了在云端单节点上的高效大规模图分析的边界。

# 2.论文内容分析

## 2.1 Brief summary

本文利用改进的MOMS结构设计了一套运行在FPGA平台上的图计算缓存系统，通过MOMS能够容忍大量访问缺失的特性来解决图算法的不规则访存导致的性能下降问题。

在设计系统时，考虑了现代的多片FPGA结构，设计了片间互联逻辑，最大化利用FPGA资源。提出了二级MOMS和私有MOMS结构来减少冲突和改善吞吐量。并给出了一个可定制化的图算法编程模型。

## 2.2 Strengths

本文提出的缓存系统大大改善了图的不规则访存导致的性能下降问题，从内存级并行进行优化，与FPGA上的先进技术相比，实现了3倍的几何平均速度，与多核CPU相比，带宽效率高1.1-5.8倍，功率效率高3.0-15.3倍，与GPU实现相比，能够扩展到更大的大规模的图。

## 2.3 Weaknesses

## 2.4 Can you do (much) better? Present your thoughts/ideas

## 2.5 What have you learned/enjoyed/disliked in the paper? Why?