总结

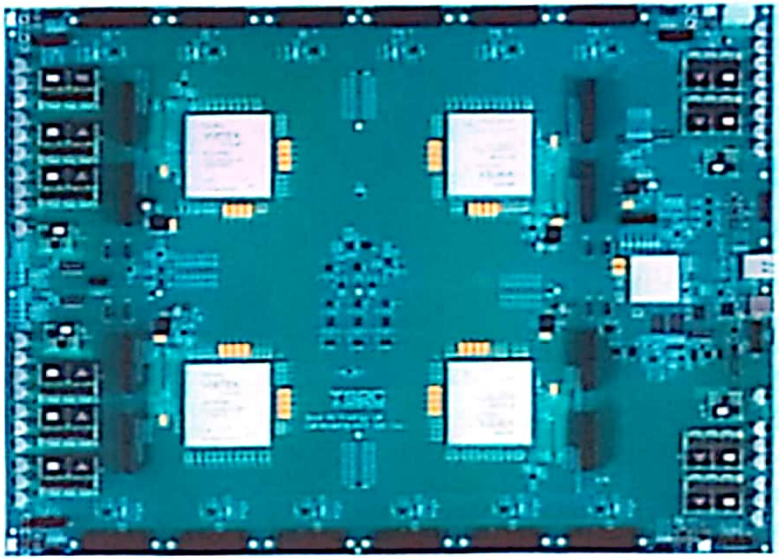
年度预期目标	阶段成果
① 完成加速器软件模拟器的改进和测试工作，满足相关指标 ② 实现FPGA的性能验证工作	• 针对图计算阶段性处理等特点，进行定制化的高并发流水设计，解决负载不规则与遍历不规则
	• 针对图计算访存瓶颈设计高效的存储模型，实现高带宽访存
	• 完成架构模拟，对性能进行了评估，满足相关指标
	• 搭建FPGA验证平台，完成部分IP核验证

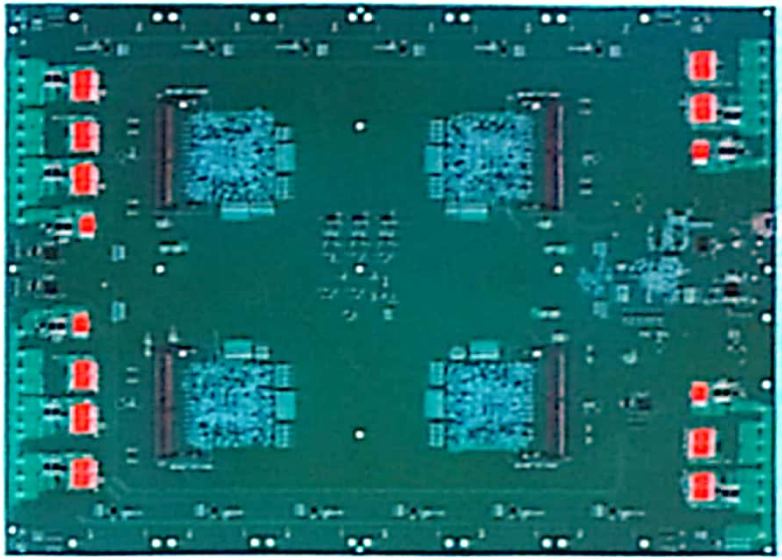
序号	论文和专利
1	一种面向宽度优先搜索算法的加速装置、方法及存储介质，发明人：曹华伟，安学军
2	一种面向多上下文粗粒度数据流结构的指令发射控制方法，发明人：李涵，严明玉，李文明
3	一种面向图计算应用的CAM结构存储系统，发明人：李涵，严明玉，李文明，叶笑春
4	基于动态通联图分析的电信诈骗事件检测方法及系统，发明人：贾瑞花，张承龙，曹华伟

芯片投片前期调研

●FPGA验证板卡

- S2C Quad VU440硬件板卡含4片Xilinx VU440系列芯片，可以扩展IO接口，其中120pin的接插件专门用于扩展PCIe等高速IO接口量
- 每颗VU440带有一个DDR插槽，DDR可以通过300 pin接口的接插件进行扩展，每颗VU440最多可接3个DDR通道，支持DDR3、DDR4
- S2C提供的工具player pro可以自动或手动分割设计到多块 FPGA 板





未来优化思路

- **预处理**
  - 增加预处理过程
  - 降低负载规模，缩小片上存储所需容量
- **算法模型**
  - 混合Vertex-Centric与Edge-Centric编程模型
  - 结合二者优势，提升图计算效率
- **执行模式**
  - 混合同步与异步执行模式
  - 提升不同算法收敛速度

阶段进展——实验配置

- **实验对比平台**
  - **Graphicionado** [1]
    - 高度流水化的面向图计算领域的专用加速器
    - 通过片上存储降低访存延迟，解决遍历不规则
  - **Gunrock** [2]
    - 基于GPU的大容量同步图计算系统
    - 提出data-centric的概念
- **实验系统配置**

	GraphDynS	Graphicionado	Gunrock(V100)
Compute Unit	1Ghz 16×SIMT8	1Ghz 128×Streams	1.25Ghz 5120×cores
On-chip memory	32MB eDRAM	64MB eDRAM	34MB
Off-chip memory	512GB/s HBM 1.0	512GB/s HBM1.0	900GB/s HBM 2.0

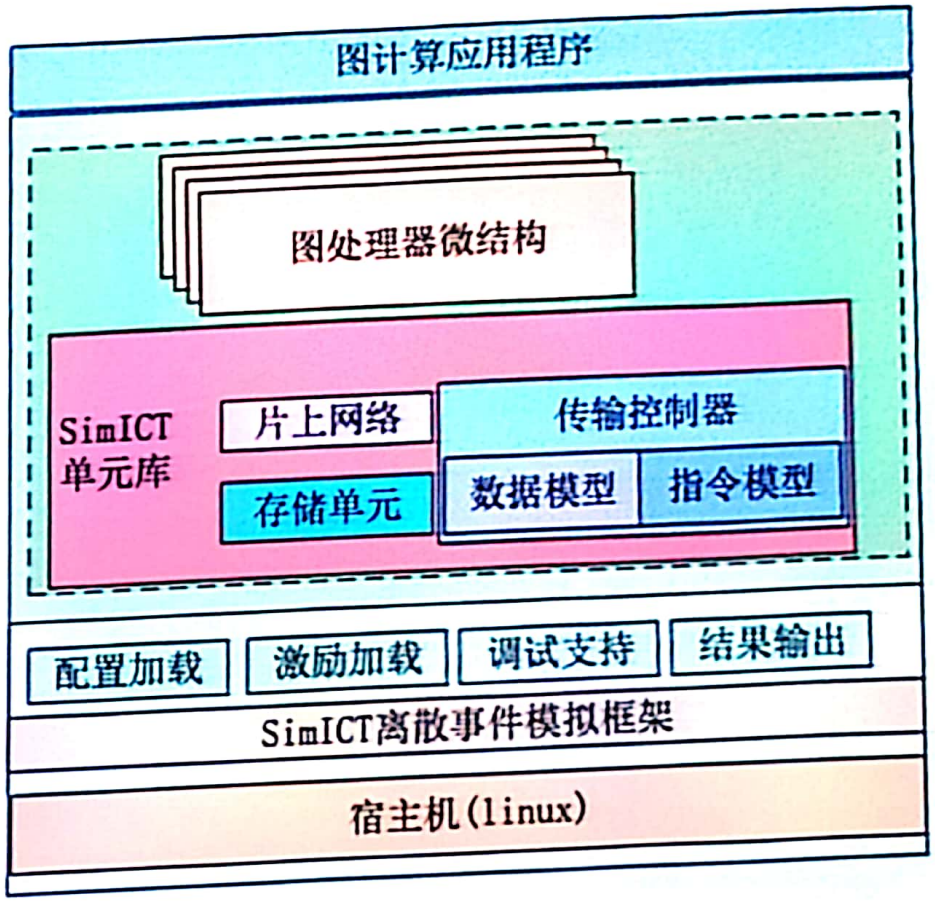
注：GPU的片上存储包括寄存器、共享存储与L2 cache

[1] Ham T J, et al. Graphicionado: A high-performance and energy-efficient accelerator for graph analytics[C] // MICRO2016.
[2] Wang Y, Davidson A, et al. Gunrock: a high-performance graph processing library on the GPU[J] // Acm Sigplan Notices2015

架构模拟

- 基于SimICT框架进行图处理器的模拟器组件开发

- **SimICT[2]**
  - 千线程并行的众核模拟器平台
  - 用于大规模系统的性能与功耗评估
  - 组件化系统模拟
  - 灵活易改、模拟速度快



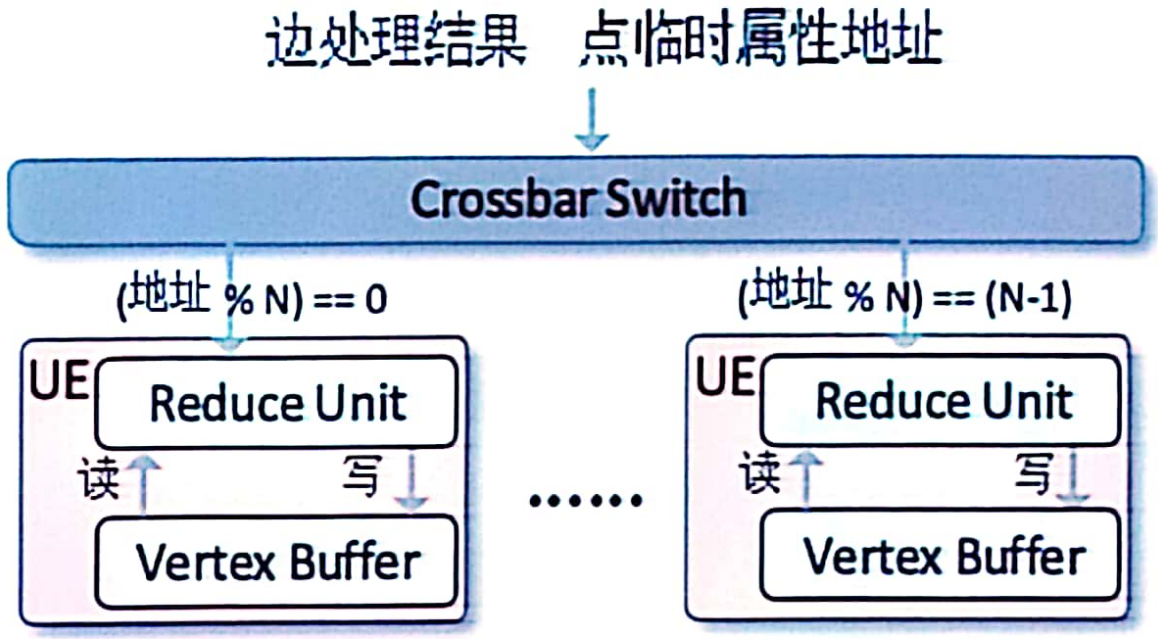
[2] Ye X, Fan D, Sun N, et al. SimICT: A fast and flexible framework for performance and power evaluation of large-scale architecture

编程接口

- **Process_Edge(src_data, edge_data, pro_info)**
  - get_src_data：获取源点数据信息
  - get_edge_data: 获取对应出边数据信息
  - put_pro_info：输出边处理结果信息
- **Reduce(dst_tmpdata, pro_info)**
  - get_dst_tmpdata：获取目的点数据信息
  - get_pro_info：获取边处理结果信息
  - upd_dst_tmpdata：更新目的点数据信息
- **Apply(dst_tmpdata, dst_data)**
  - get_dst_data：获取目的点数据信息
  - upd_dst_data：更新目的点数据信息

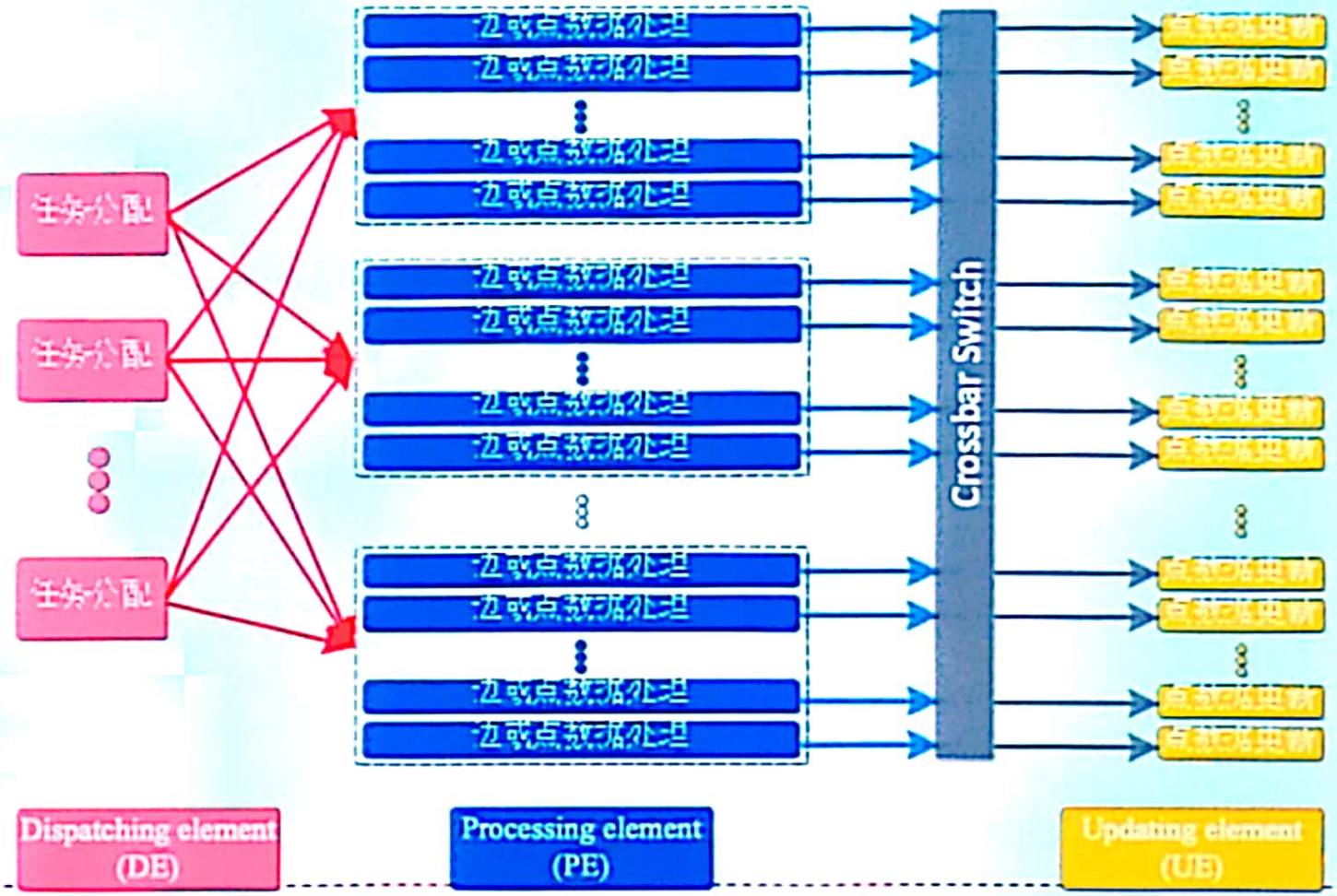
高并发流水线

- **缓解原子操作停顿**
  - PE通过Crossbar Switch访问on-chip memory
  - 根据DST地址路由入相应的Updating Element
  - 缩小原子域，缓解read-after-write冲突



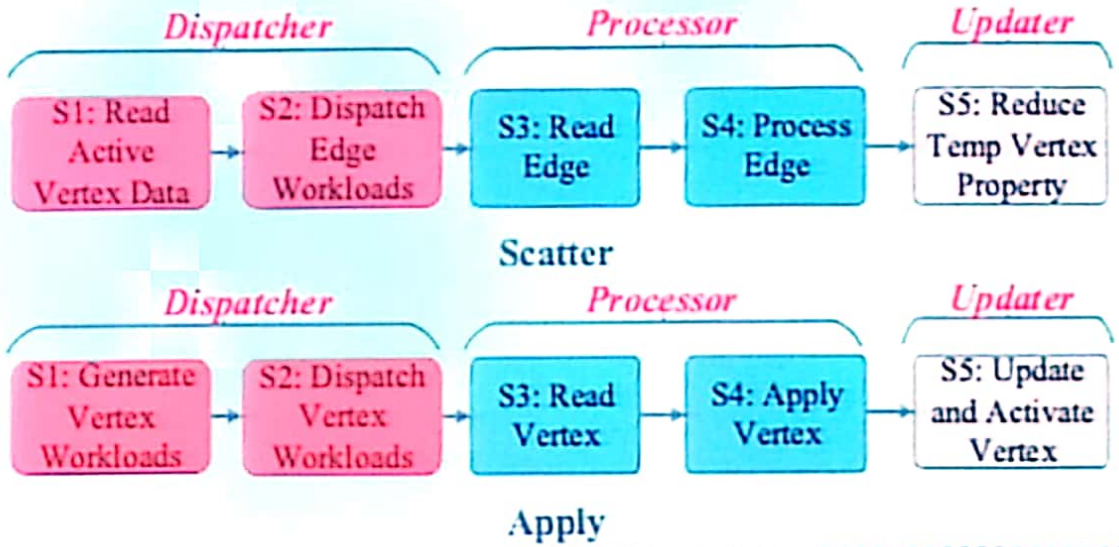
高并发流水线

- **流水级间工作协调**
  - 均衡分配负载 → 高效处理边/点信息 → 片上不规则访存路由



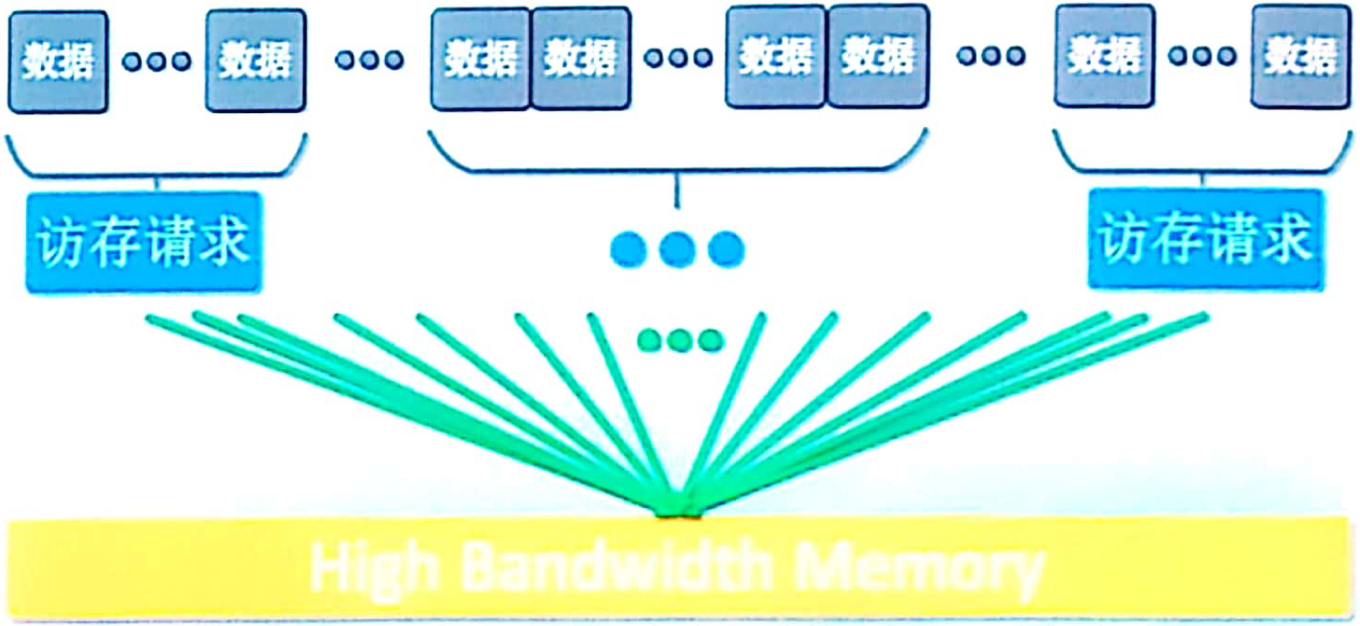
高并发流水线

- **解耦合数据路径**
  - **Scatter** : Dispatching Stage -> Dispatcher
    Processing Stage -> Processor
  - **Apply** : Dispatching Stage -> Dispatcher
    Processing Stage -> Processor



高并发访存

- **精确预取机制**
  - 于解耦的数据路径中获取数据预取信息
  - 实现精确预取机制
  - 提升片外访存请求率、带宽利用率

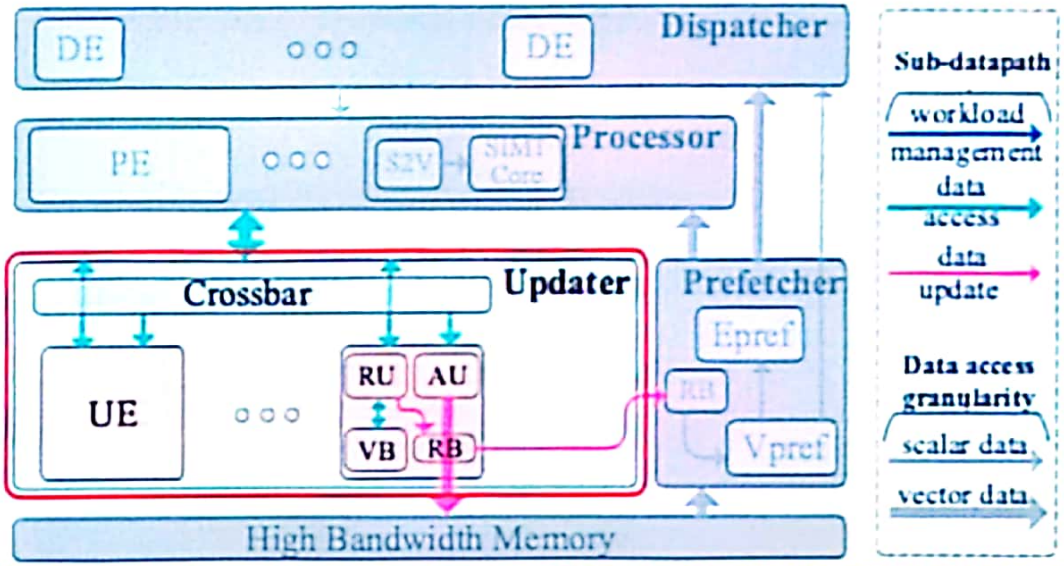


关键技术

- **高并发访存**
  - 实现边数据的精确预取
  - 提升访存的并发度
- **高并发流水线**
  - 解耦合微架构数据路径
  - 定制并优化流水级
  - 减少流水停顿，提高数据路径的通量

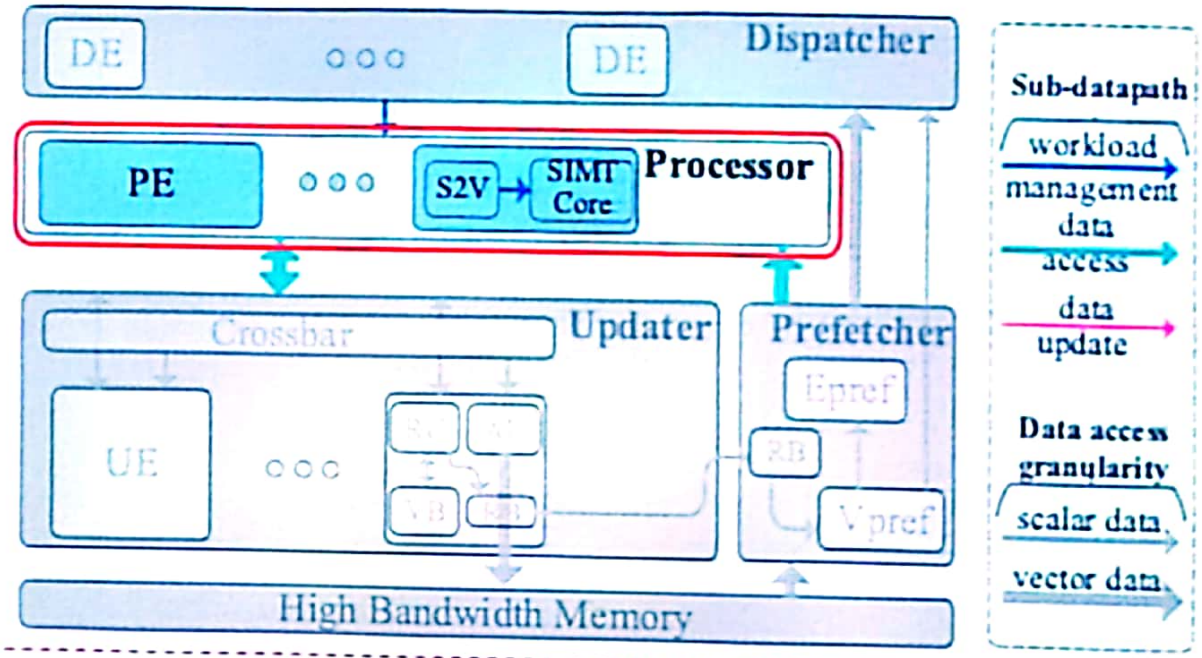
架构设计-Updater

- **基本任务和架构**
  - 由128个Updating Elements(UEs)、crossbar switch组成
  - UE = Vertex Buffer(VB)+Ready-to-update Bitmap(RB)+Reducing Unit(RU)+Activating Unit(AU)
  - 负责后半程流水线
    - Scatter阶段：节点Reduce函数计算与临时属性更新
    - Apply 阶段：节点的更新与激活



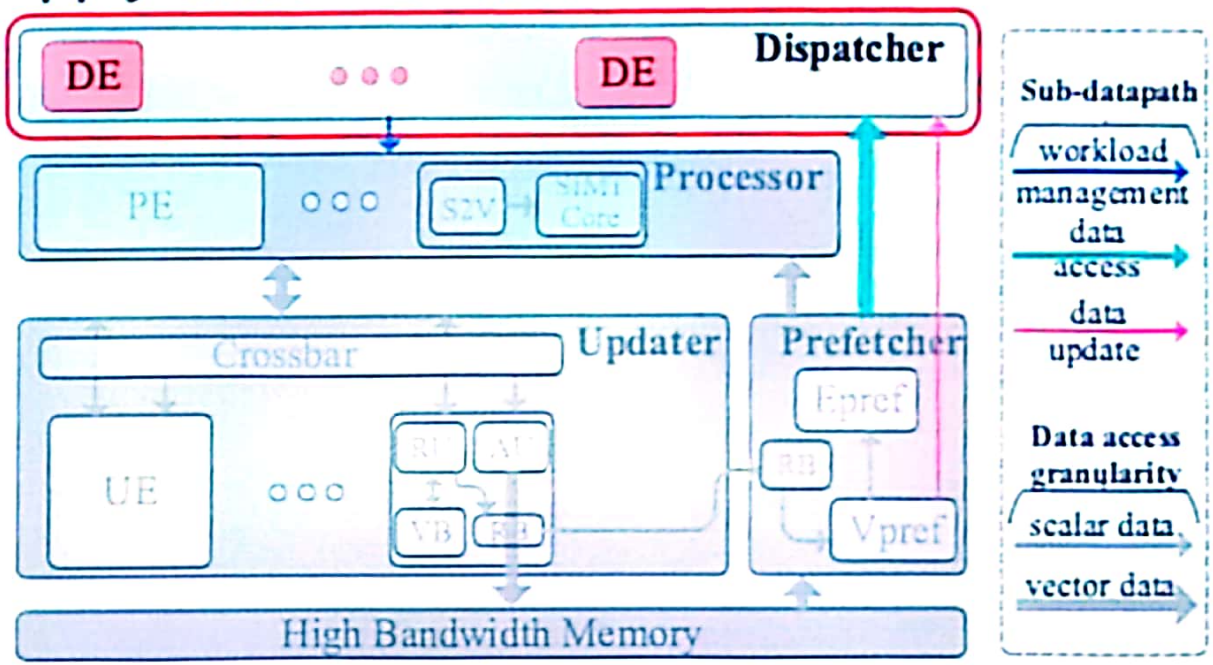
架构设计-Processor

- **基本任务和架构**
  - 由16个Processing Elements（PEs）构成
  - PE = Scalar to Vector单元（S2V）+ SIMT核
  - 负责前半程流水线
    - Scatter阶段：激活节点出边的处理
    - Apply 阶段：Apply函数计算，为节点激活做准备

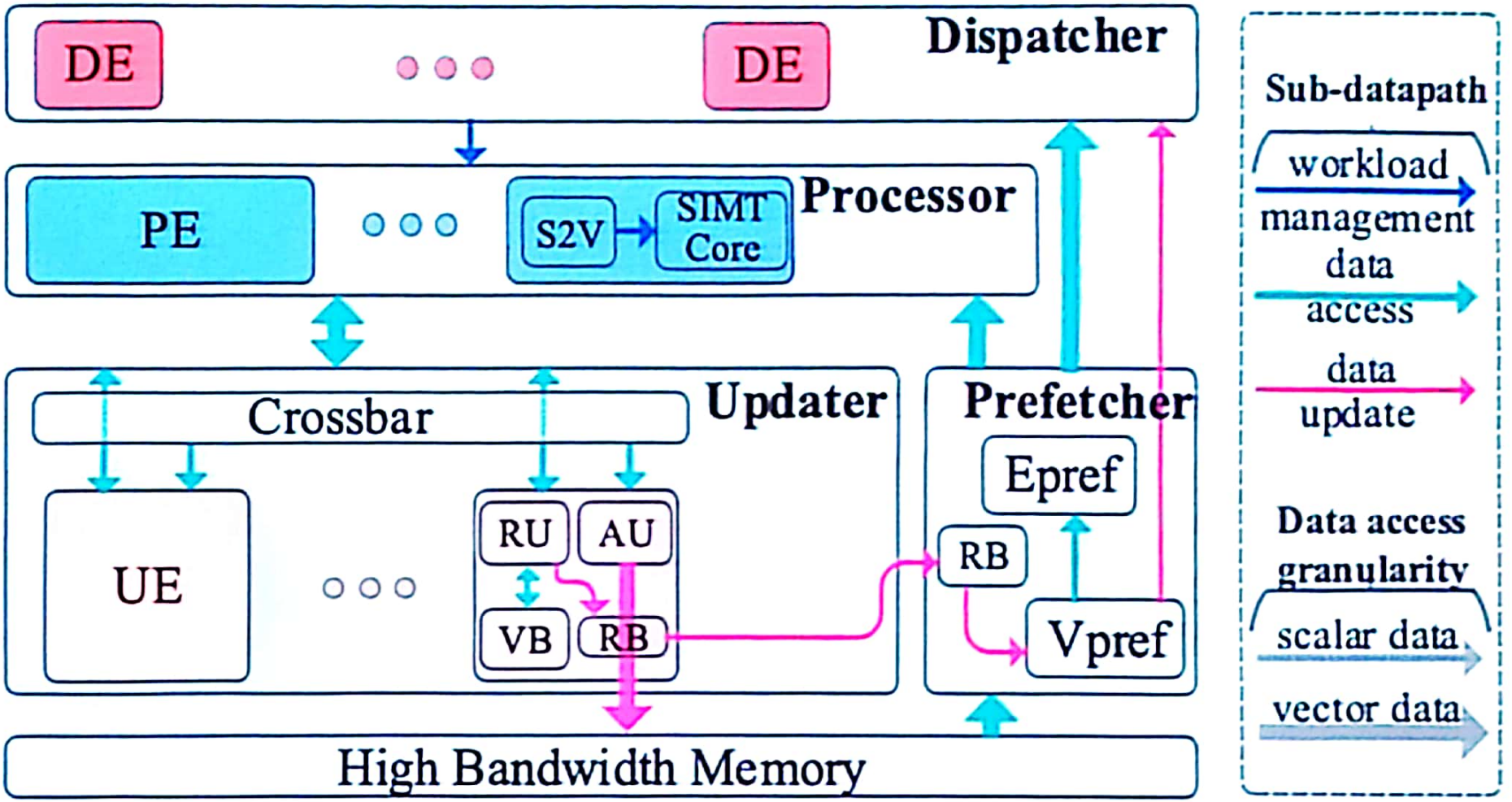


架构设计-Dispatcher

- **基本任务和架构**
  - 由16个Dispatching Elements（DEs）构成
  - 负责在Scatter和Apply阶段中，为Processor均衡分配任务
    - Scatter阶段：小于阈值->分配给单个PE
      大于阈值->均分给所有PE
    - Apply 阶段：均分给所有PE



架构设计



- 由Dispatcher、Processor、Updater、Prefetcher、HBM模块组成

化的算法模型

Scatter Phase

```
1  for each active vertex u do
2     Dispatch (src.prop, src.offset, src.edgeCnt) to PE;
3  end
4  for e(u,v) <- EdgeArray[u.offset : u.offset+u.edgeCnt] do
5     edgeProResult <- Process_Edge(u.prop, e.weight);
6     v.tempProp <- Reduce(v.tempProp, edgeProResult);
7  end
```

Apply Phase

```
1  for each vertex list do
2     dispatch(start id vListStartID and size vListSize of vertex list) to PE;
3  end
4  for vid <- vListStartID :  vListStartID + vListSize do
5     edgeCnt <- OffsetArray[vid+1] - OffsetArray[vid];
6     applyResult <- Apply(v_vid.prop, v_vid.tempProp, v_vid.constProp);
7     if v_vid.prop != applyResult then
8        v_vid.prop <- applyResult;
9        activate vertex with v_vid.prop, OffsetArray[vid] and edgeCnt;
10    end
11 end
```

Decoupled Stages

- Dispatching Stage
- Processing Stage
- Dispatching Stage
- Processing Stage

优化的算法模型

Scatter Phase

```
1  for each active vertex u do
2     Dispatch (src.prop, src.offset, src.edgeCnt) to PE;
3  end
4  for e(u,v) <- EdgeArray[u.offset : u.offset+u.edgeCnt] do
5     edgeProResult <- Process_Edge(u.prop, e.weight);
6     v.tempProp <- Reduce(v.tempProp, edgeProResult);
7  end
```

Apply Phase

```
1  for each vertex list do
2     dispatch(start id vListStartID and size vListSize of vertex list) to PE;
3  end
4  for vid <- vListStartID : vListStartID + vListSize do
5     edgeCnt <- OffsetArray[vid+1] - OffsetArray[vid];
6     applyResult <- Apply(v_vid.prop, v_vid.tempProp, v_vid.constProp);
7     if v_vid.prop != applyResult then
8        v_vid.prop <- applyResult;
9        activate vertex with v_vid.prop, OffsetArray[vid] and edgeCnt;
10    end
11 end
```