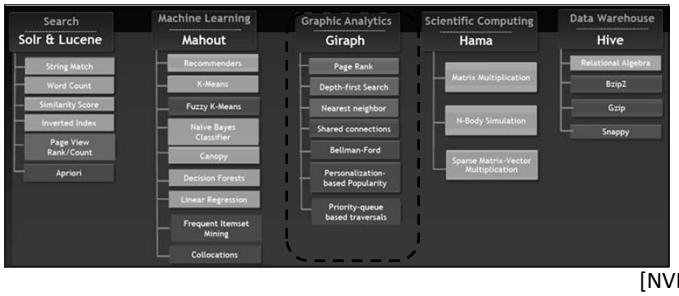
Memory Fast-Forward: A Low Cost Special Function Unit to Enhance Energy Efficiency in GPU for Big Data Processing

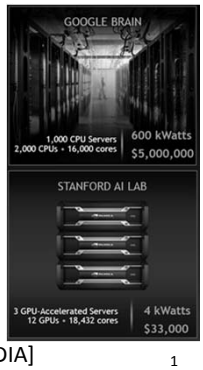
Eunhyeok Park, Junwhan Ahn, Sungpack Hong*, Sungjoo Yoo, and Sunggu Lee**

Seoul National University
Oracle Labs*
POSTECH**

GPU is Good for Server, But …

- **Large potential in cost/performance improvement**
  - **GPU-accelerated servers: IBM/NVIDIA, Stanford AI Lab, …**
- **GPU needs improvement to cover more workload**
  - **Not good at Graphic analytics (below)**

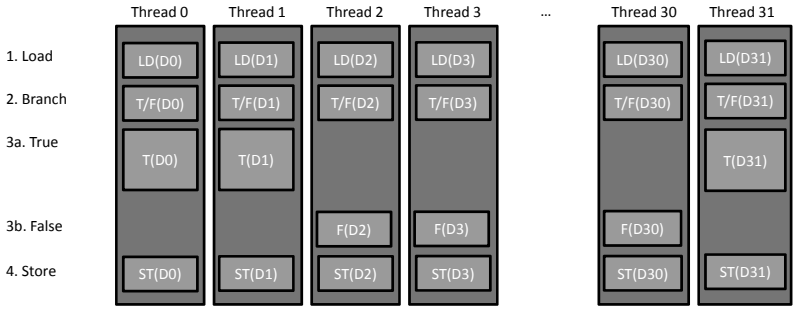




[NVIDIA]

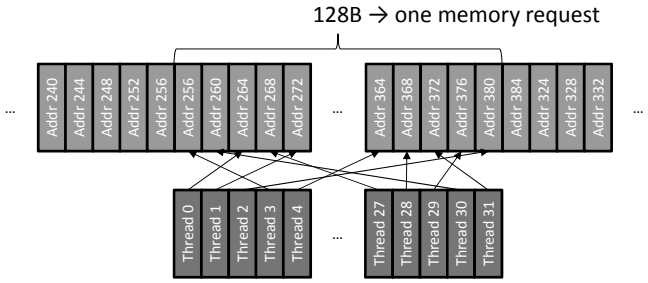
1

GPU



- **Single program multiple data (SPMD)**
  - **Many threads run the same program in parallel**
- **Significant memory requests are issued by many threads**
  - **How to reduce memory traffics? → Memory request coalescing**

Memory Request Coalescing



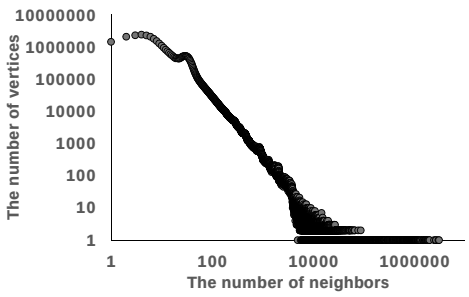
Is this enough for new memory-intensive workload, e.g., graph computation?

Graph Computation

- Graph computation
  - Processing a query to graph database
- Utility of graph computation
  - Big data analytics (social network analysis, DNA analysis, …), machine learning, …

Graph Characteristics

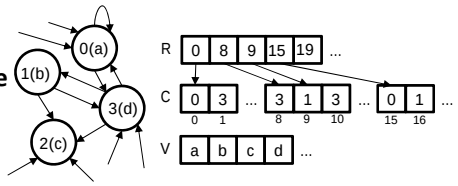
- Characteristics
  - Massively parallel computation → Good with GPU!
  - Significant amount of memory accesses to large data → OK with GPU?
  - Power Degree Law of Real Graph



5

Graph Computation Example

- **A graph representation**
  - **Compressed sparse row (CSR) often used to reduce graph size**



- **PageRank**
  - **Calculating popularity of a web page, i.e., rank**
  - **Weighted sum of neighbors' ranks**
  - **Continue iterations until convergence**

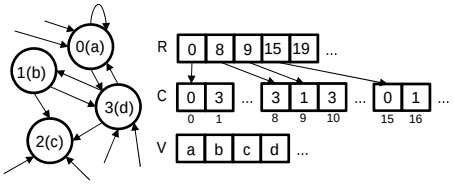
1. **Input:** $\{\mathbf{v}|\mathbf{v}\in\mathbf{V}\},\{\mathbf{e}=(\mathbf{j}\rightarrow\mathbf{k})|\,\mathbf{j},\mathbf{k}\in\mathbf{V},\mathbf{e}\in\mathbf{E}\}$
2. **Initialize:** $\mathbf{Rank_0(v)=p}$

1. **for** $\mathbf{v}\in\mathbf{V}$
2. $\quad\mathbf{T=0}$
3. $\quad$**for** $\{\mathbf{w}|(\mathbf{w}\rightarrow\mathbf{v})\in\mathbf{E}\}$
4. $\quad\quad\mathbf{T}\mathrel{+}=\mathbf{Rank_{l-1}(w)/\,N_{out}(w)}$
5. $\quad\mathbf{Rank_l(v)=p+(1-p)T}$

1. **p: random reset probability**
2. $\mathbf{N_{out}(w)}$**: the number of out-degree of w**

Graph Computation on GPU

- **One thread processes one vertex**
- **Memory requests with structural locality**



thread id :	0	1	2	3	···	j
Vertex id :	0	1	2	3		j
Step 1	R[0]	R[1]	R[2]	R[3]		R[j]
Step 2	R[1]-R[0]	R[2]-R[1]	R[3]-R[2]	R[4]-R[3]		R[j+1]-R[j]
Step 3-a	C[0]	C[8]	C[9]	C[15]		C[R[j]]
Step 4-a	V[0]	V[3]	V[1]	V[0]		V[C[R[j]]]
Step 3-b	C[1]	-	C[10]	C[16]		C[R[j]+1]
Step 4-b	V[3]	-	V[3]	V[1]		V[C[R[j]+1]]

time

1. **Input:** $\{\mathbf{v}|\mathbf{v}\in\mathbf{V}\},\{\mathbf{e}=(\mathbf{j}\rightarrow\mathbf{k})|\,\mathbf{j},\mathbf{k}\in\mathbf{V},\mathbf{e}\in\mathbf{E}\}$
2. **Initialize:** $\mathbf{Rank_0(v)=p}$

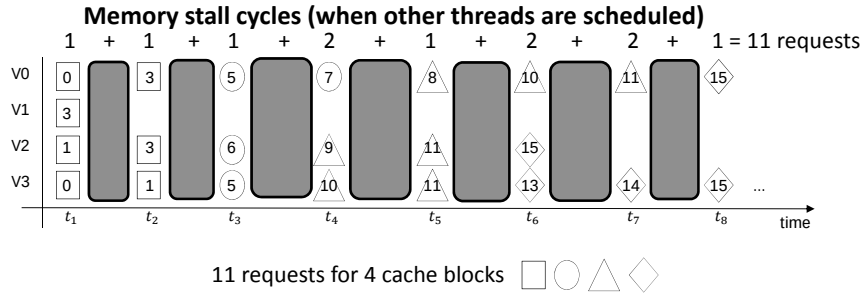
1. **for** $\mathbf{v}\in\mathbf{V}$
2. $\quad\mathbf{T=0}$
3. $\quad$**for** $\{\mathbf{w}|(\mathbf{w}\rightarrow\mathbf{v})\in\mathbf{E}\}$
4. $\quad\quad\mathbf{T}\mathrel{+}=\mathbf{Rank_{l-1}(w)/\,N_{out}(w)}$
5. $\quad\mathbf{Rank_l(v)=p+(1-p)T}$

1. **p: random reset probability**
2. $\mathbf{N_{out}(w)}$**: the number of out-degree of w**

Memory accesses with **pointer chasing** for vertex 0
R[0] → C[0] → V[0] → C[1] → V[3] → …

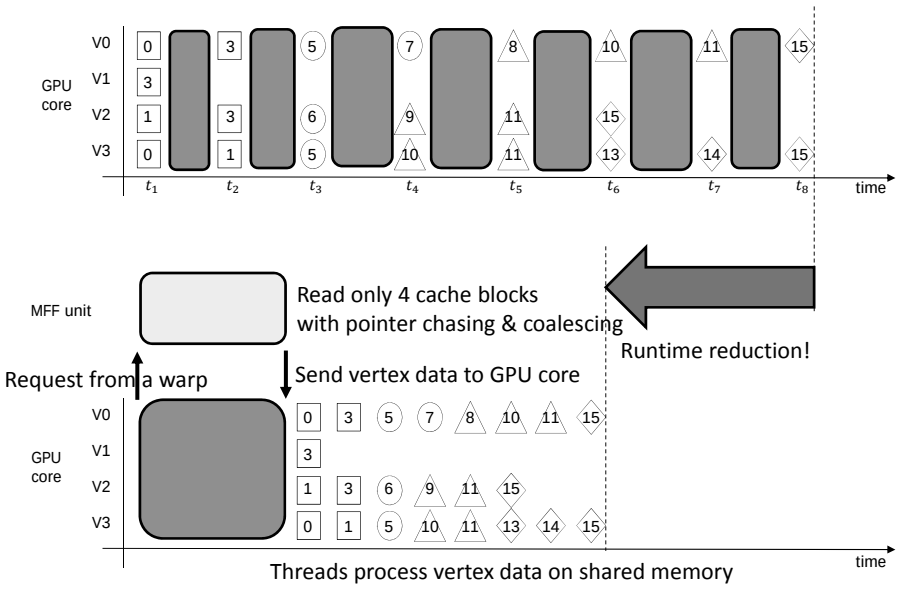
Problem: Lack of Memory Request Coalescing in Existing GPU

- **Requests are scattered over a long period**
  - **Due to multi-threading on memory stalls**
- **Hard to find required data on caches**
  - **Cache hit rates for graph computation: 0%/9% on L1/L2 cache (ljournal-2008 on GTX 970)**



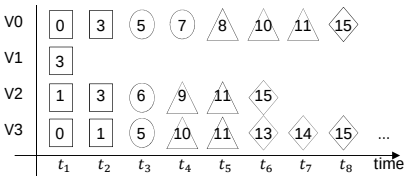
11 requests for 4 cache blocks

Basic Idea



Memory Fast Forward (MFF) Unit

- **Co-processor unit in GPU**
- **Coalesce memory requests issued in different cycles**
- **Perform pointer chasing on behalf of threads**



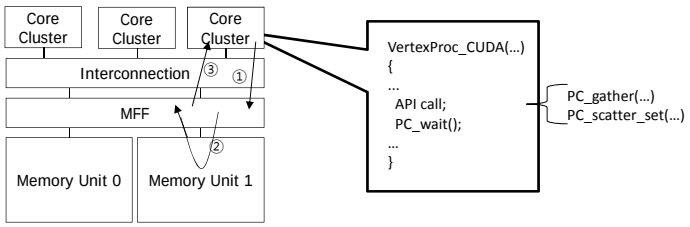
11 requests → 4 requests

thread id :	0	1	2	3	...	j
Vertex id :	0	1	2	3		j
Step 1	R[0]	R[1]	R[2]	R[3]		R[j]
Step 2	R[1]-R[0]	R[2]-R[1]	R[3]-R[2]	R[4]-R[3]		R[j+1]-R[j]
Step 3-a	C[0]	C[8]	C[9]	C[15]		C[R[j]]
Step 4-a	V[0]	V[3]	V[1]	V[0]		V[C[R[j]]]
Step 3-b	C[1]	-	C[10]	C[16]		C[R[j]+1]
Step 4-b	V[3]	-	V[3]	V[1]		V[C[R[j]+1]]

time

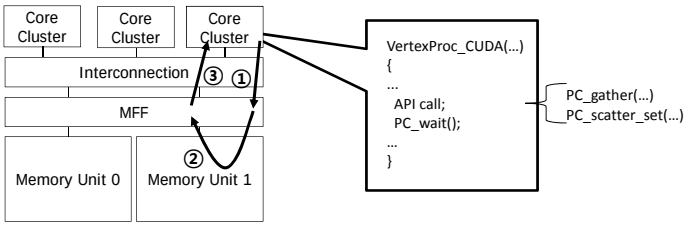
R → C → V → C → V → …
R → C → V
R → C → V → C → V → …
…

MFF Overview: API



- **GPU program can call the following functions**
  - **PC_gather: Fetch vertex data following pointer chains**
  - **PC_scatter_set: Set multiple scattered bits in a cache block**
    - **Can replace multiple atomic writes**
  - **PC_wait: Wait for a reply from MFF unit**

How GPU Core and MFF Works?



- **Step 1: Each thread calls an MFF function**
- **Step 2: Memory accesses, pointer chasing, coalescing**
- **Step 3: MFF unit returns data (e.g., a set of vertex attributes) to core cluster (scratch pad memory) and resumes thread executions**

Code Modification to Use MFF Unit

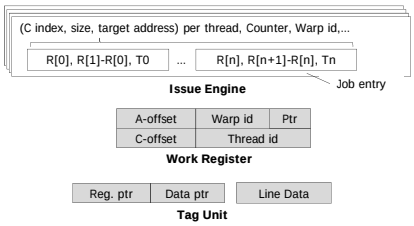
```
inEdgeOffset = R[vID];
inEdgeNum      = R[vID+1]-R[vID];
newValue       = initial_value;

for( i = 0 ; i < inEdgeNum ; i++)
{
        neighborValue = V[C[inEdgeOffset]+i];
        process(newValue,neighborValue);
}

….
update(vID,  newValue);
…
```

Original GPU code

```
__shared__sharedPCStorage[ blockDim.x * PC_size ];
PCstorage = sharedPCStorage +threadIdx.x*PC_size

…
inEdgeOffset = R[vID];
inEdgeNum      = R[vID+1]-R[vID];
newValue       = initial_value;

for( i = 0 ; i < inEdgeNum ; i+=PC_size )
{

        reqSize =
         (inEdgeNum -i > PC_size) ? PC_size : inEdgeNum – i;

        PC_gather(PCstorage,inEdgeOffset+i,reqSize);
        PC_wait();

        for(i = 0 ; i  < reqSize ; i++ )
        {
                process(newValue,PCstorage[i]);
        }
}

….
update(vID,  newValue);
…
```

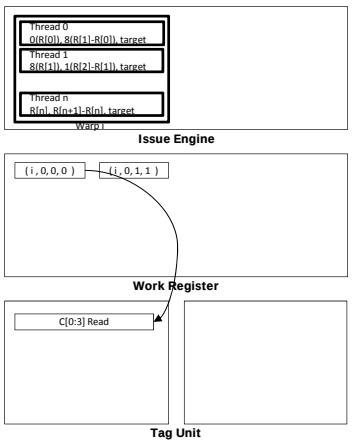
Modified code

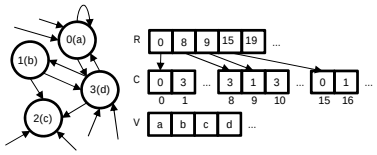
Internal Structure of MFF Unit



- **Issue Engine**
  - **Keep commands from warps and initiate pointer chasing**
- **Work Register**
  - **One register = one read to C array and one read to V array**
    - **C state for column index array access**
    - **V state for vertex data array access**
  - **For request coalescing, a linked list is formed to include registers (=requests) for the same cache block**
- **Tag Unit (~memory request buffer + cache)**
  - **Memory request status and data array (line data)**

How MFF Unit Works

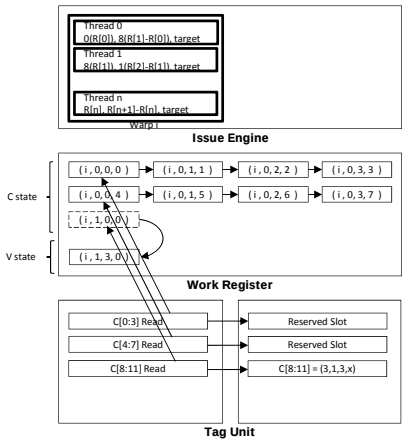


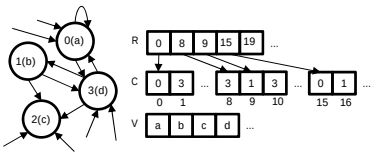



```
...
PC_gather(target,R[n],R[n+1]-R[n]);
PC_wait()
...
```

4) Generate a memory request (e.g., for C[0]),
Search Tag Unit,
If no matching previous request, issue it
Else make a link to an outstanding request
for request coalescing

How MFF Unit Works



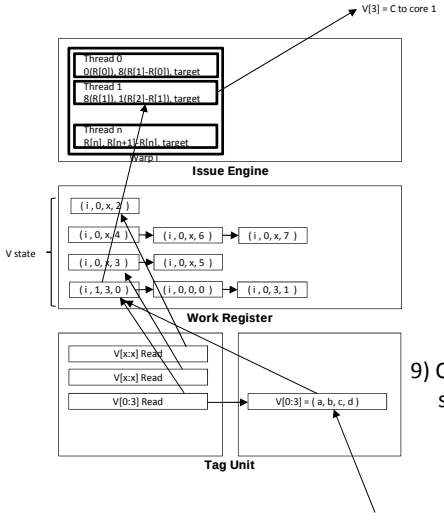


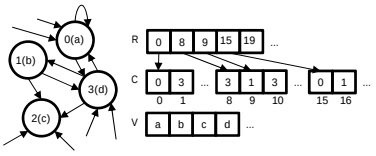
...
PC_gather(target,R[n],R[n+1]-R[n]);
PC_wait()
...

8) Change the state of work register
from C- state to V-state and
create memory requests for vertex attributes

*One register = one read to C array and one read to V array

How MFF Unit Works

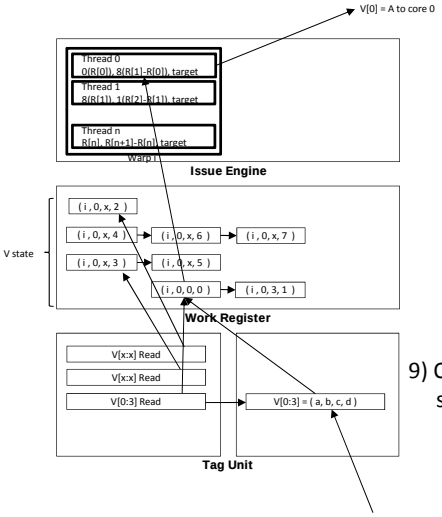


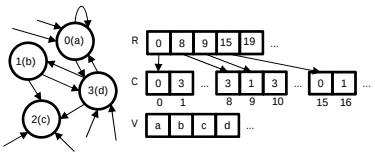


...
PC_gather(target,R[n],R[n+1]-R[n]);
PC_wait()
...

9) On receiving vertex data from memory,
send them to GPU core

How MFF Unit Works






```
...
PC_gather(target,R[n],R[n+1]-R[n]);
PC_wait()
...
```

9) On receiving vertex data from memory, send them to GPU core

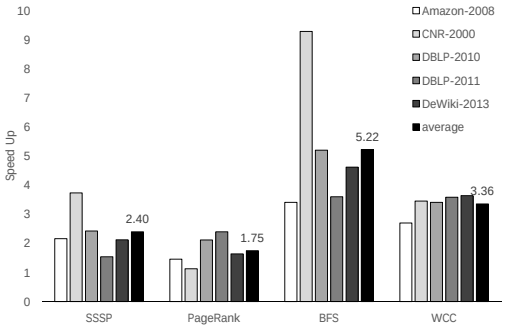
Experimental Setup

- **GPGPU-Sim**
  - **Fermi GTX480 – 15 core clusters, 1.4 GHz, 177.4 GB/s**
- **GPUWattch, Micron Power Calculator, and CACTI**
  - **Power estimation (GPU, DRAM, and MFF unit)**
- **Five real graphs**

Graph name	Vertices	Edges	Type
DBLP-2010	326186	1615400	Bibliography network
DBLP-2011	986324	6707236	
CNR-2000	325557	3216152	Italian CNR domain
Amazon-2008	735323	5158388	Similarity lookup
DeWiki-2013	1532354	36722696	Wikipedia snapshot

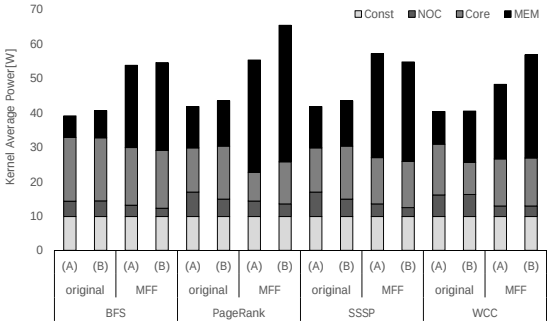
- **Four graph computation algorithms**
  - **PageRank**
  - **Breath First Search (BFS)**
  - **Single Source Shortest Path (SSSP)**
  - **Weakly Connected Component (WCC)**

Results: Performance



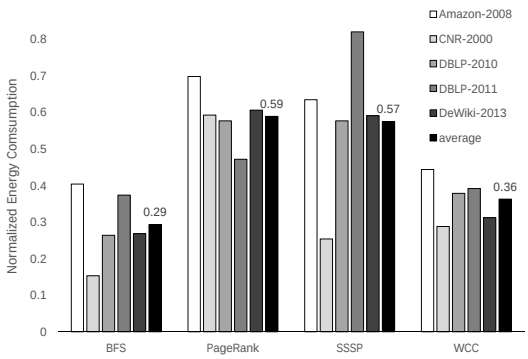
- **1.75~5.22x improvement**
- **Benefit of API depends on access pattern and graph data.**

Results: Power Consumption



- **Lower power consumption in NoC and GPU core since MFF unit handles pointer chasing**
- **Higher memory power due to denser memory traffics from MFF unit**

Results: Energy Consumption



- **Energy reduction: average 54.6%**
- **Negligible overhead from MFF unit: 0.3%**

Summary: Memory Fast Forward

- **Problem**
  - **GPU lacks in request coalescing capability for graph computation**
  - **Vertex Scheduling is serialized due to power degree law of graph**
- **Proposed solution**
  - **MFF unit for pointer chasing and memory request coalescing**
- **Experimental results**
  - **1.75~5.22X performance improvement**
  - **41~71% reduction in energy consumption**