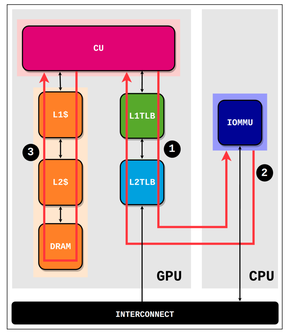
EFFICIENT ORGANIZATION OF TLB HIERARCHY FOR GPUs

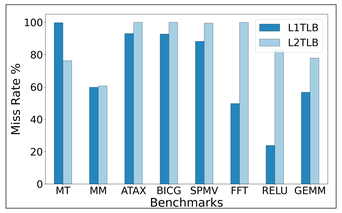
Aman Choudhary | Govindarajan Ramaswamy

Introduction

- GPUs widely used for accelerating HPC apps.
- VM support on GPUs:
  1. Reduces programmer effort by handling page migration
  2. Allows memory over-subscription
- VM has overheads, like **address translation**.
- Prior works show page sharing across private L1-TLBs.
- We thus propose to modify TLB hierarchy.

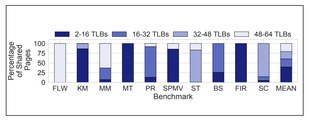


Motivation

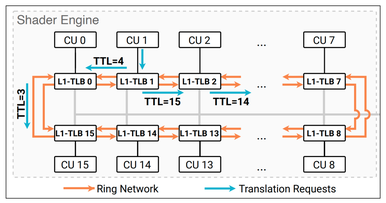


GPU apps suffer from very high (**~99%**) L1-TLB and L2-TLB miss rates. This triggers long latency page table walks, thus hurting performance.

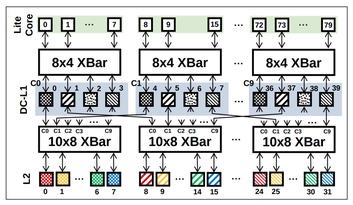
There exists a significant amount of **replication of entries in the private L1-TLBs** of GPU cores. This wastes precious TLB storage.



Related Work



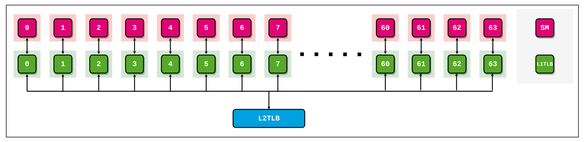
Baruah et. al. use **probing** of neighboring L1-TLBs to gain from replication of L1-TLB entries.



Adwait Jog et. al. use **aggregation** & **clustering** to deal with replication in L1-caches in GPUs.

Baseline

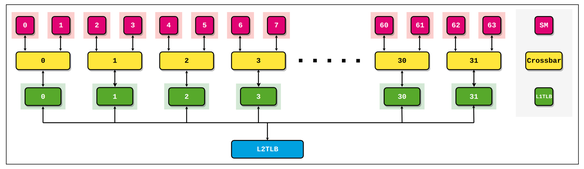
- We model an ***AMD R9 Nano GPU*** system on ***MGPUSim***.



Aggregated L1-TLBs

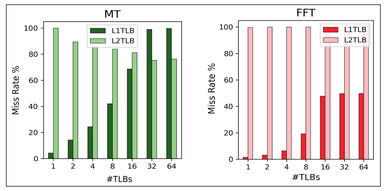
▸ DESIGN

#TLBs per GPU	*#CUs sharing each TLB*	*#Entries per TLB*	*#Sets per TLB × Ways*	*#MSHR Entries per TLB*	*Access Latency*	*Xbar Latency*
64	1	128	1 X 128	4	1 cycle	-
32	2	256	2 X 128	8	2 cycles	1 cycle
16	4	512	4 X 128	16	4 cycles	2 cycles
8	8	1024	8 X 128	32	6 cycles	3 cycles
4	16	2048	16 X 128	64	8 cycles	4 cycles
2	32	4096	32 X 128	128	10 cycles	5 cycles
1	64	8192	64 X 128	256	12 cycles	6 cycles

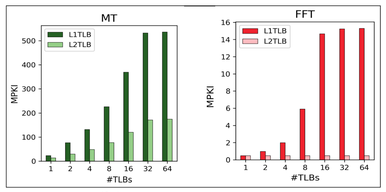


▸ PERFORMANCE

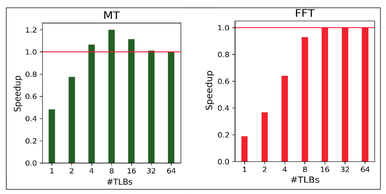
Replication-sensitive (RS)
Replication-insensitive (RIS)



(a) TLB miss rate summary



(b) TLB MPKI summary



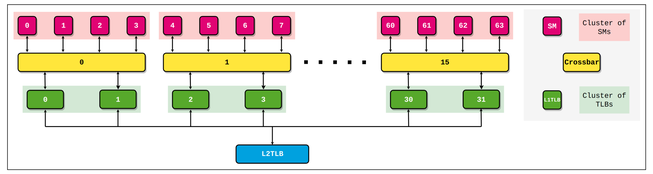
(c) Speedup summary

- **L1-TLB miss rates** decrease rapidly, due to reduction in replication. Drop to **≤ 5%** for all apps except *spmv*.
- **L2-TLB miss rate** is different for all apps.
- **L1-TLB MPKI** mimics L1-TLB miss rates.
- For, RS apps **L2-TLB MPKI** drops due to decrease in L1-TLB lookup misses, for RIS apps, it stays same.
- Competing trends contribute to the performance of RS apps. Avg: **2.8x**, Max:**6.1x**, Min:**1.2x**
- Performance degrades for RIS apps. *spmv* shows **4%** gain, others show none.

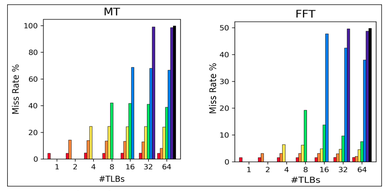
Clustered L1-TLBs

▸ DESIGN

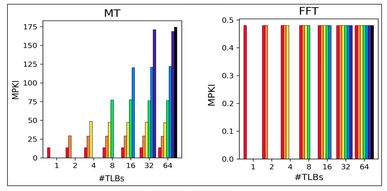
- Aggregated L1-TLBs and CUs grouped together to form a **cluster**. No replication of entries in a cluster.
- Each VA is assigned a unique set in a cluster, so almost one copy is present.
- Many-to-many relationship between CUs and L1-TLBs.
- 1 ≤ Number of clusters ≤ Number of L1-TLBs.
- Bigger crossbars required, so increased latency.



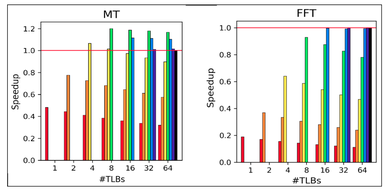
▸ PERFORMANCE



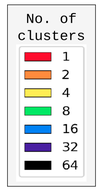
(a) L1-TLB miss rate summary



(b) L2-TLB MPKI summary



(c) Speedup summary



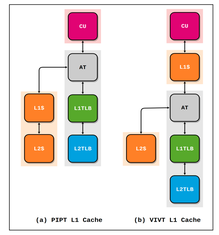
App.	*#TLBs × #Clusters*	*Speedup*
MT	8 × 8	1.20x
MM	32 × 16	1.90x
ATAX	4 × 4	2.40x
BICG	4 × 4	2.40x
GEMM	16 × 16	6.17x
SPMV	8 × 8	1.04x
FFT	64 × 64	1.00x
RELU	64 × 64	1.00x

- **L1-TLB miss rates** decline further due to clustering; decrease for RS apps due to decline in L1 lookup misses, while for RIS apps due to drop in MSHR hits.
- **L2-TLB miss rates** are hard to judge. **L1-TLB MPKI** mimics L1-TLB miss rates (so, both figures omitted).
- Like before, for RS apps **L2-TLB MPKI** drops due to decrease in L1-TLB lookup misses; for RIS apps, it stays almost same.
- As a result, RS apps show speedup gains, while RIS apps don't.
- Like before, speedup first improves for RS apps, reaches a peak & then drops. Avg: **2.8x**, Max: **6.2x**, Min: **1.2x**
- As before, performance degrades for RIS apps. *spmv* shows **4%** gain, others show none.
- Best performance corresponds to configs with **pure aggregation**.
- Clustering benefits only when #clusters **closer to 64**. For larger clusters, high X-bar latency hurts performance.
- Best speedup achieved for #TLBs between **4-16**, where clustering fares badly.

VIVT L1-Cache

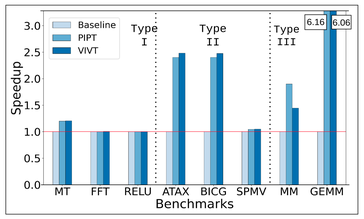
▸ DESIGN

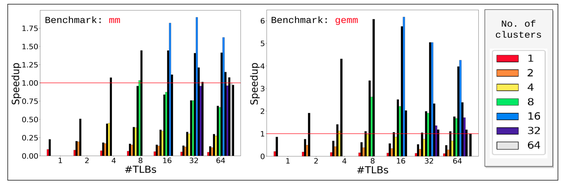
- Aggregation and clustering improve L1-TLB hit rate, but increase access time.
- So, VIVT design explored, to avoid address translation.
- Issues with VIVT caches:
  a) **Homonym** b) **Synonym**



▸ PERFORMANCE

- **Type I**: No speedup (**~99%** miss), **Type II**: Some improvement (**7-8%** for *atax*, *bicg*, **0.4%** for *spmv*) due to **5-10%** L1-$ hit rate.
- **Type III**: VIVT configs outperform PIPT except for a few anomallies. Avg. **30%** more speedup for *mm*, **140%** for *gemm*.
- Anomalous cases due to high TLB miss rates.





Conclusion

- VM has key benefits, hence its performance is critical. Address translation acts as bottle-neck; new designs need to mitigate its costs.
- GPU apps share pages across L1-TLBs. So, we proposed aggregation and clustering.
- Replication reduced & performance improved, but bigger L1-TLBs and X-bars have high latency.
- Hence, we explored VIVT L1-$ to remove address translation from critical path.
- **Aggregated TLB hierarchy, along with virtual cache subsytem can reduce overhead of address translation.**

▸ FUTURE WORK

- Verify L1-TLB access and crossbar latency.
- Evaluate area and power costs for our design.
- Design a scheme for reconfiguring #L1-TLBs dynamically.
- Improve our VIVT implementation.

Computer Science & Automation | IISc Bangalore