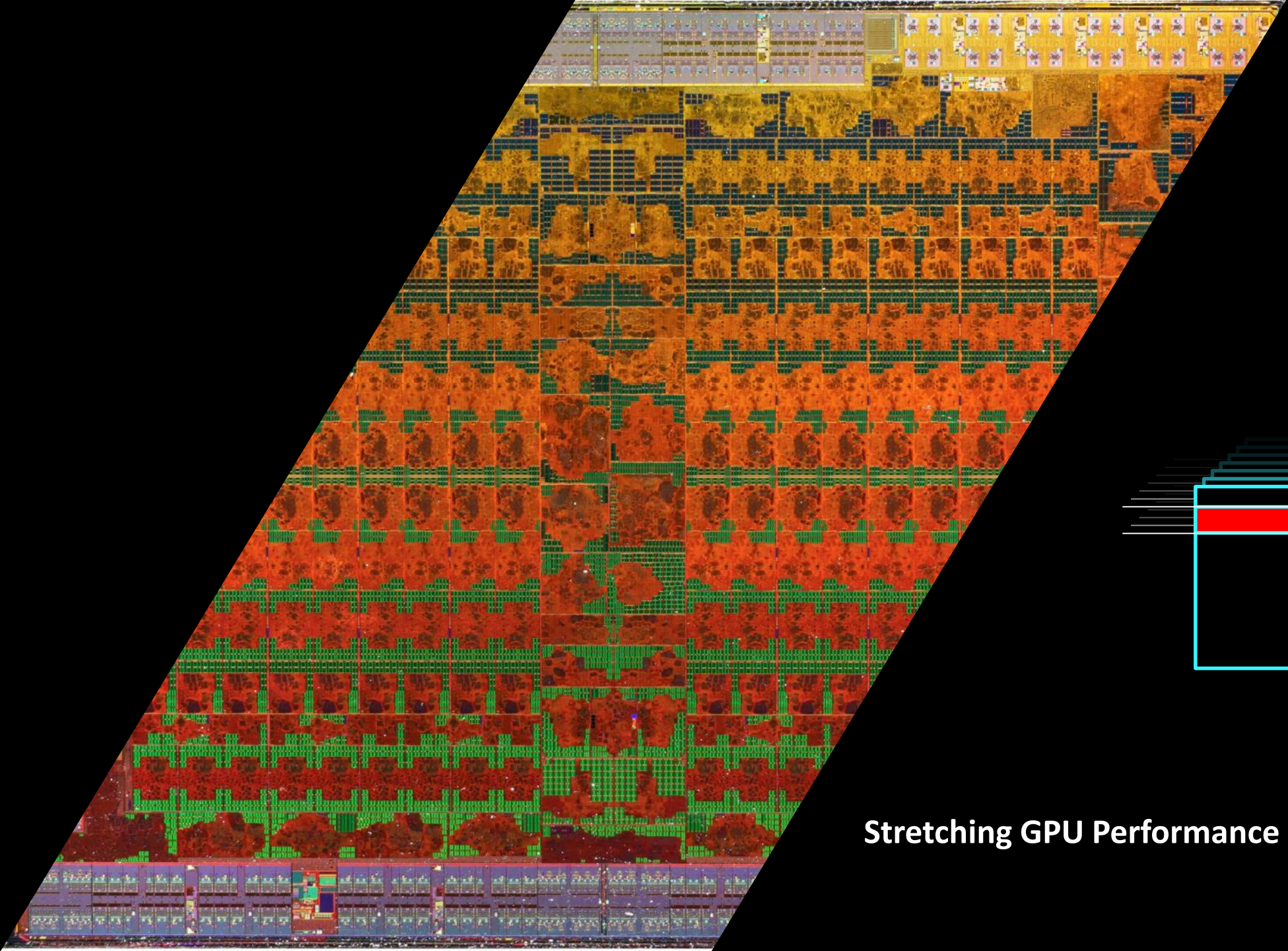
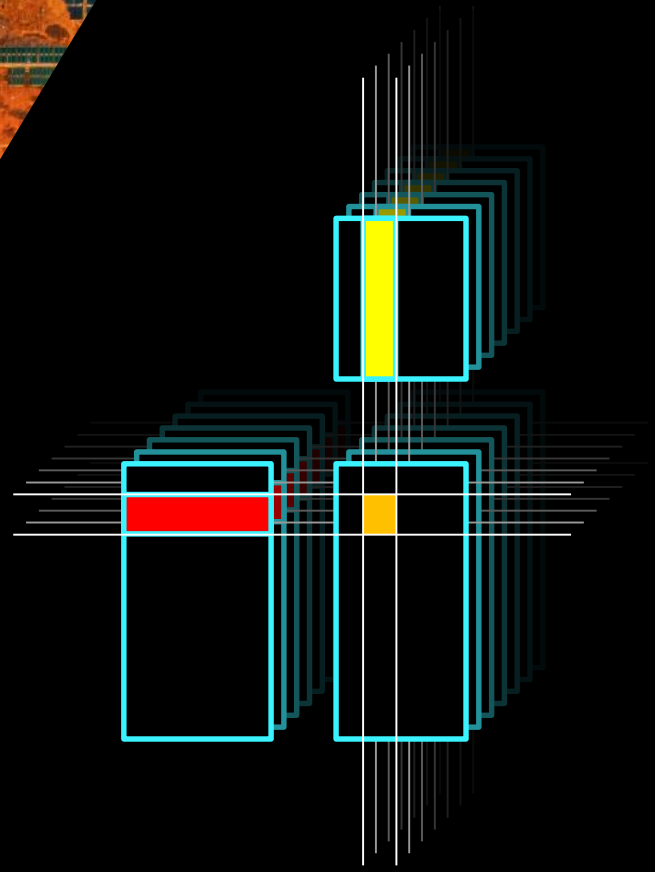


AMD



TENSILE

Stretching GPU Performance for Tensor Contractions

David E Tanner

OUTLINE

https://github.com/ROCmSoftwarePlatform/Tensile

AMD

- Terms
- Motivation
- Supported Problem Types
- GPU Performance Parameters
- General Kernel-Level Strategy
- Kernel Parameters
- Auto-Tuning
- Mapping Problem Sizes to Kernels
- Performance Experiments
- Future Work

TERMS

- Compute-Unit (CU)
- Local Data Share (LDS)
- Vector General-Purpose Register (VGPR)
- Instruction-Level Parallelism (ILP)
  - issue high latency memory operation
  - instructions independent of memory op
  - wait for memory op
  - instructions dependent on memory op

MOTIVATION

AMD

- GEMM API encapsulates trillions of problems, all of which behave differently.
  - Precisions, Transpose A, Transpose B, M, N, K, Strides
- Many problem types behave similar to GEMM.
  - $C_{ij} = \beta C_{ij} + \alpha \sum_k A_{ik} * B_{kj}$ (gemm NN)
  - $C_{ijk} = \sum_l A_{ilk} * B_{jlk}$ (batched gemm NT)
  - $C_{ijk} = \sum_{lmn} A_{ilmnk} * B_{jlmnk}$ (batched gemm w/ 3D summation)
  - $C_{ijk} = \sum_{lmn} A_{inlkm} * B_{mjlkn}$ (batched gemm w/ 3D summation different data layout)
- Can we auto-generate kernels which achieve peak performance
  - For all problem types?
  - For all problem sizes?
  - On all GPUs?

SUPPORTED PROBLEM TYPES

AMD

- Example:

$$C_{ijk} = \alpha \sum_{lmn} A_{inlkm} * B_{mjlkn} + \beta C_{ijk}$$

- Tensor indices are ordered shortest to largest stride, i.e., the zeroth index/dimension has a stride of 1 (typically).
- C indices are labeled alphabetically starting with "i".
- Summation indices are labeled alphabetically picking up where C indices left off.
- A, B indices are then labeled according to which summation or C index they correspond to.
- Tensile kernel will employ:
  - Enough work-groups to cover all the dimensions of C
  - Enough nested loops to carry out the multi-dimensional summation.
- A kernel for a given problem type will give correct answer for any problem sizes.

GPU PERFORMANCE PARAMETERS

VEGA10 FRONTIER EDITION

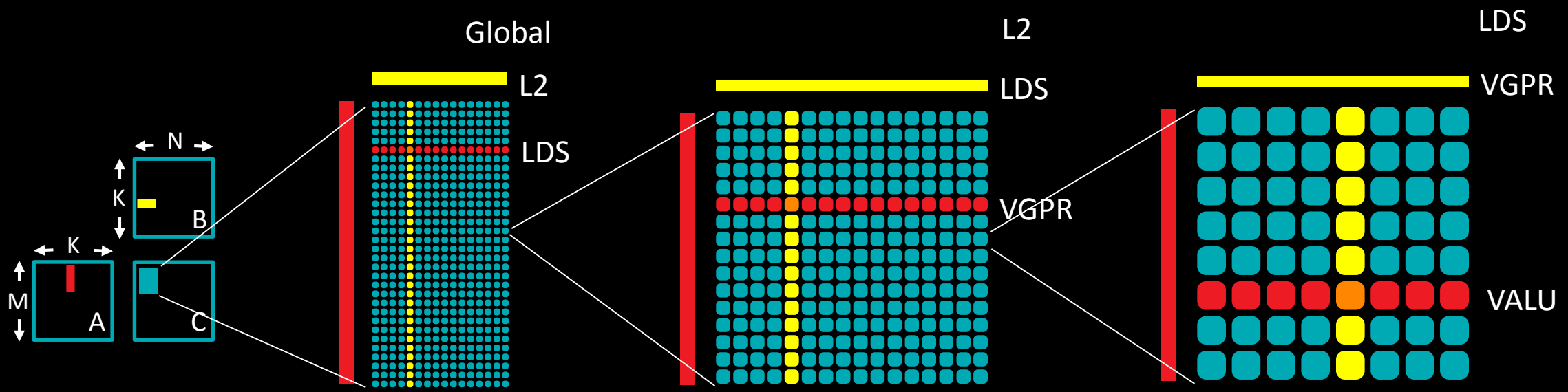
AMD

- Compute Throughput
  - 13.1 TFlops = 2 (flops/cycle)*64(CUs)*64(lanes/CU)*1600MHz
- Global Memory Bandwidth
  - 480 GB/s
  - coalescing
- Global Memory Latency
  - hundreds of cycles
  - hide using CU-occupancy or ILP
- CU Occupancy
  - resources (VGPRs, LDS) permitting, multiple workgroups are concurrently scheduled on CU
- Whole-GPU Occupancy
  - need enough total workgroups to fill up the GPU
- Caches
  - L2 shared by all CUs
  - L1 dedicated to CU
- LDS Bandwidth
  - TB/s
- LDS Latency
  - tens of cycles
  - hide using CU-occupancy or ILP
- Instruction Divergence
  - all threads within workgroup do same instruction else need to compute and apply execution masks
- Instruction Throughput
  - gemm requires 2*M*N*K instructions, all extras hurt efficiency
  - minimize instructions which don’t count
  - maximize dual-issuing of instructions which don’t count with instructions that do; must be from different wavefronts
    - VALU, SALU, LDS, global memory, branch
- Power
  - VALU, LDS, memory, caches

GENERAL KERNEL-LEVEL STRATEGY

TILING

- Tiling at all memory levels to read from lower-bandwidth memory less and from higher-bandwidth memory more to prevent bandwidth from being bottleneck



(0) GEMM

(1) Grid of 16x32 Workgroups

- Read from global once, store in L2
- Read from L2 and store into LDS 8-16x

(2) Workgroup of 16x16 Threads

- Read from LDS and store in VGPRs 16x

(3) Thread has 8x8 elements

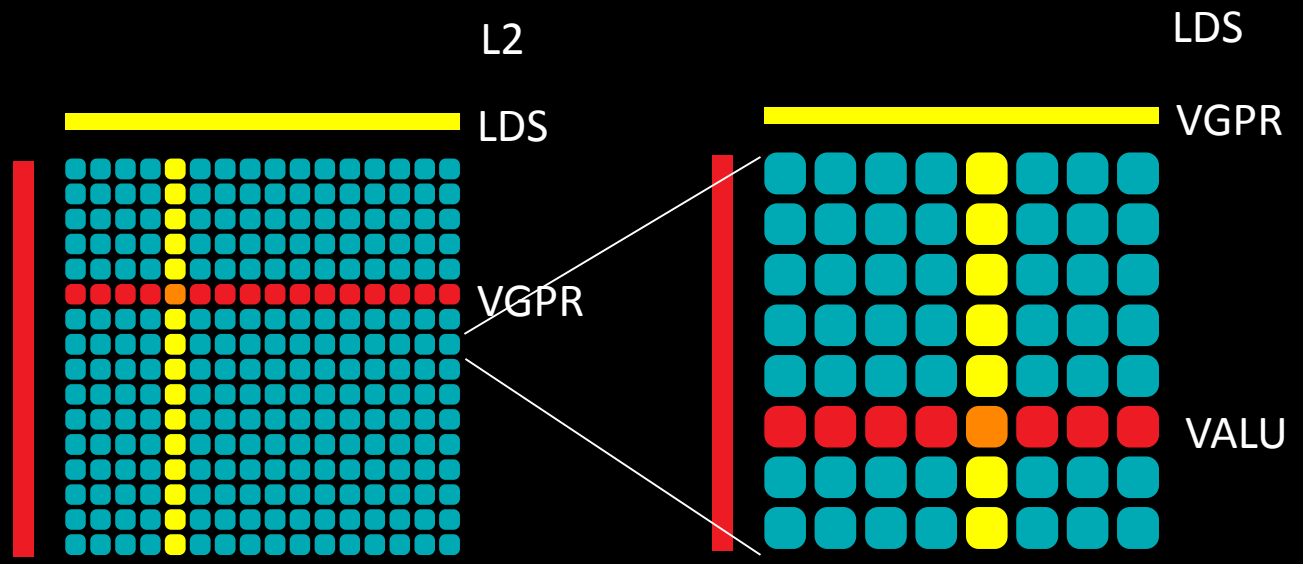
- 8 A VGPRs
- 8 B VGPRs
- 64 C VGPRs, 64 FMAs

THREAD-TILE SUB-ITERATION

WHERE WE WANT TO GET TO

```
// read 8 A elements
ds_read_b128
ds_read_b128

// read 8 B elements
ds_read_b128
ds_read_b128

// 64 MACs
v_mac_f32
v_mac_f32
v_mac_f32
v_mac_f32
+ 60 more

// Note: Prefetching and waits not shown
```

16 threads reading exact same address of LDS; coalesced.



(2) Workgroup of 16x16 Threads

- Read from LDS and store in VGPRs 16x

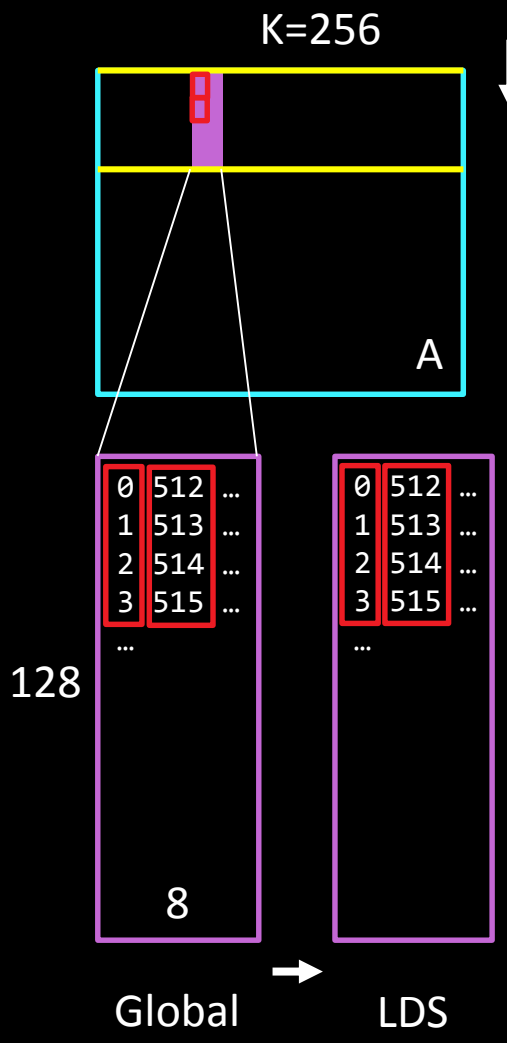
(3) Thread has 8x8 elements

- 8 A VGPRs
- 8 B VGPRs
- 64 C VGPRs, 64 FMAs

GLOBAL TO LDS

AMD

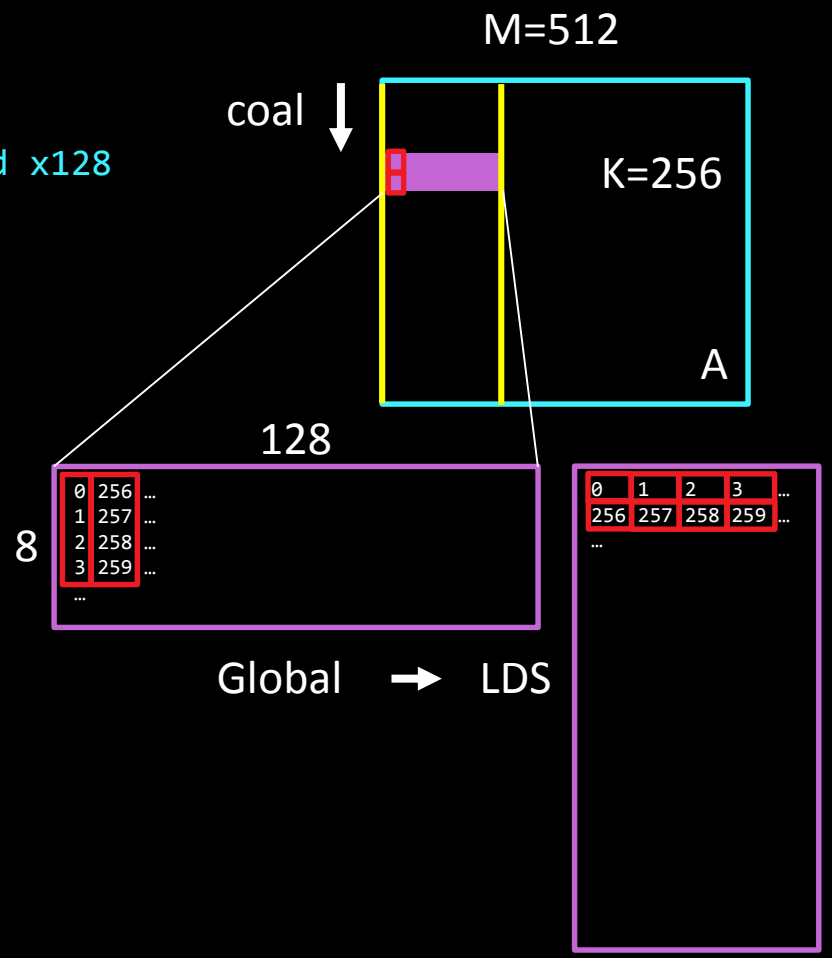
Don't Transpose Data



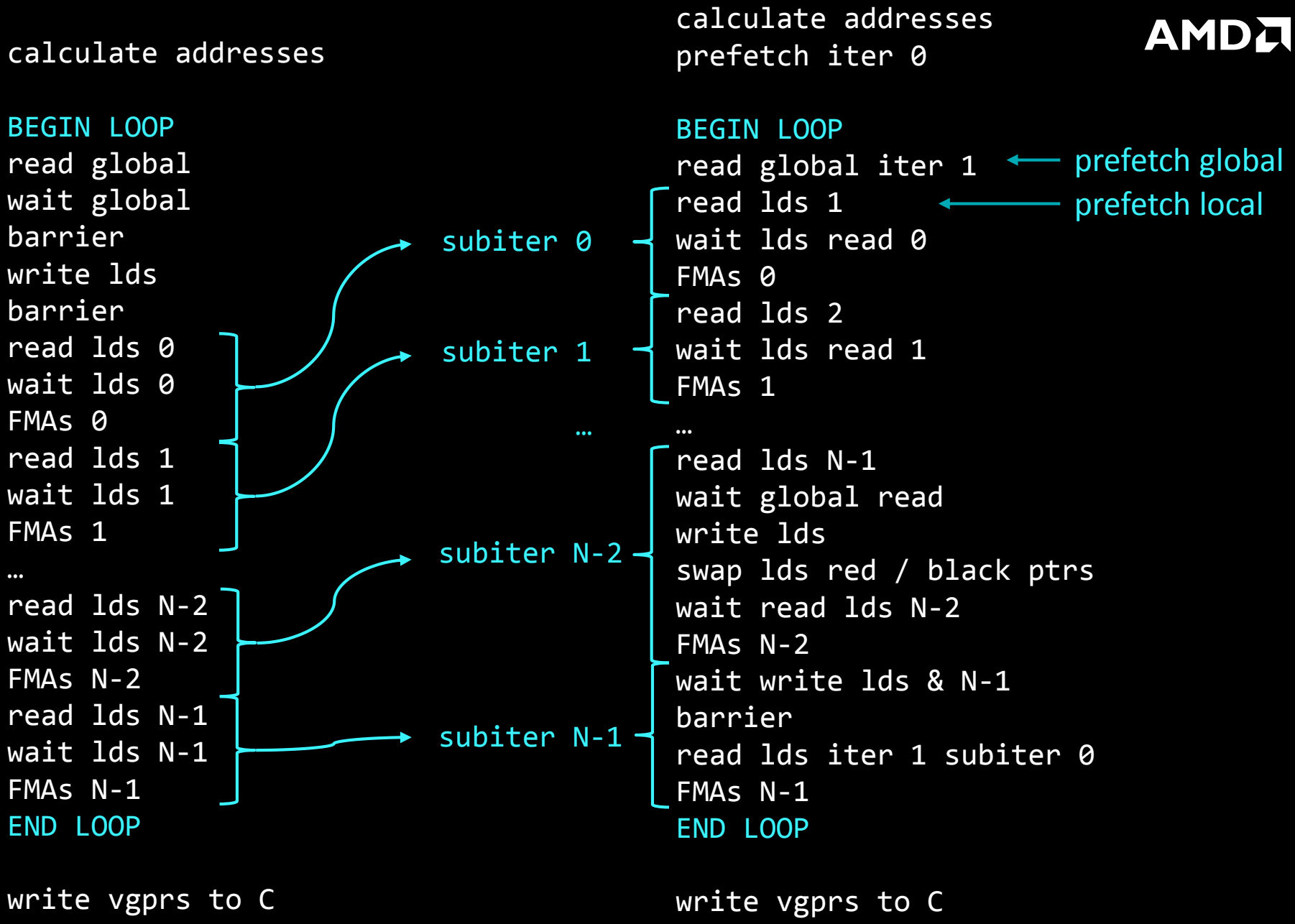

```
// read 4 A elements
// 512 bytes coalesced x8
flat_read_dwordx4

// read 4 B elements
flat_read_dwordx4

// write A to LDS
ds_write_b128

// write B to LDS
ds_write_b128
```

Do Transpose Data

```
// read 4 A elements
// 32 bytes coalesced x128
flat_read_dwordx4

// read 4 B elements
flat_read_dwordx4

// write A to LDS
ds_write_b32
ds_write_b32
ds_write_b32
ds_write_b32

// write B to LDS
ds_write_b32
ds_write_b32
ds_write_b32
ds_write_b32
```



SUMMATION LOOP



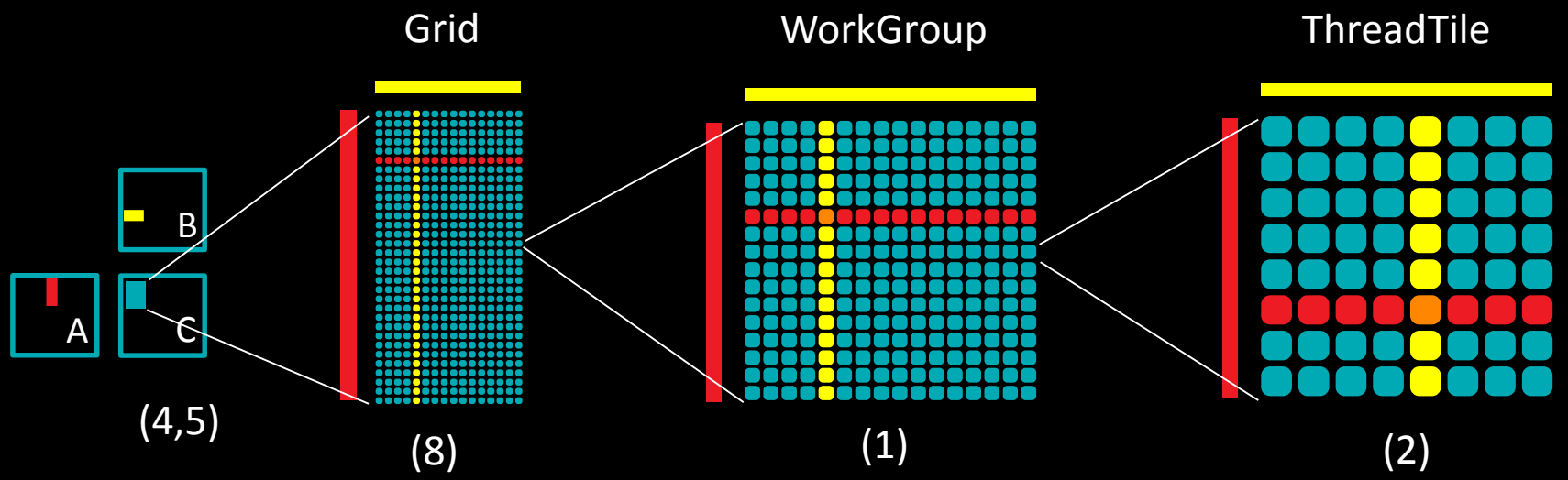
GENERAL KERNEL-LEVEL STRATEGY

OTHER

- High CU-Occupancy and prefetching to prevent latencies from being bottleneck
- High GPU-Occupancy to create enough parallel work to fill all CUs
- Eliminate divergence from summation loop

KERNEL PARAMETERS

1) WorkGroup, LocalSplitU
2) ThreadTile
3) VectorWidth
4) GlobalSplitU
5) GlobalSplitUWGM
6) PrefetchGlobalRead
7) PrefetchLocalRead
8) WorkGroupMapping
9) LoopUnroll
10) NumLoadsCoalesced
11) GlobalReadCoalesceGroup
12) GlobalReadCoalesceVector
13) KernelLanguage
14) NonTemporal



(3,10,11,12)

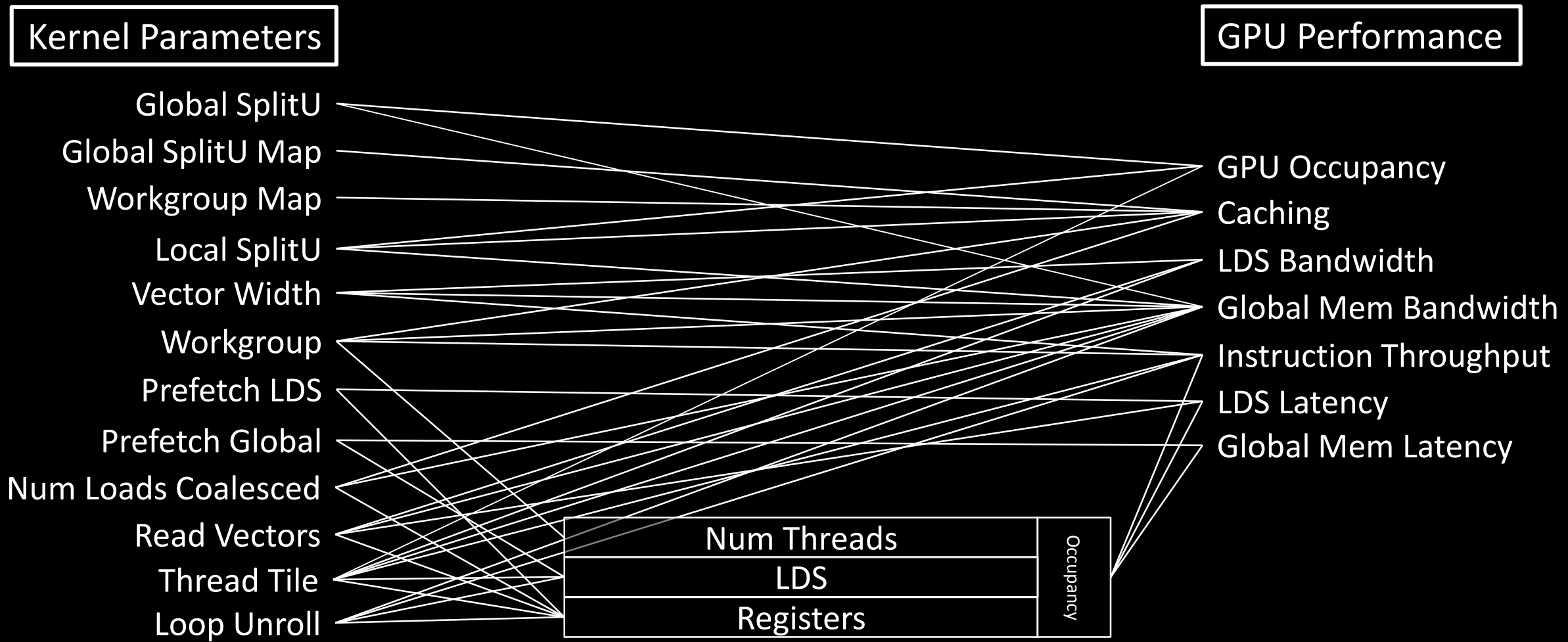
(3,6)

(7)

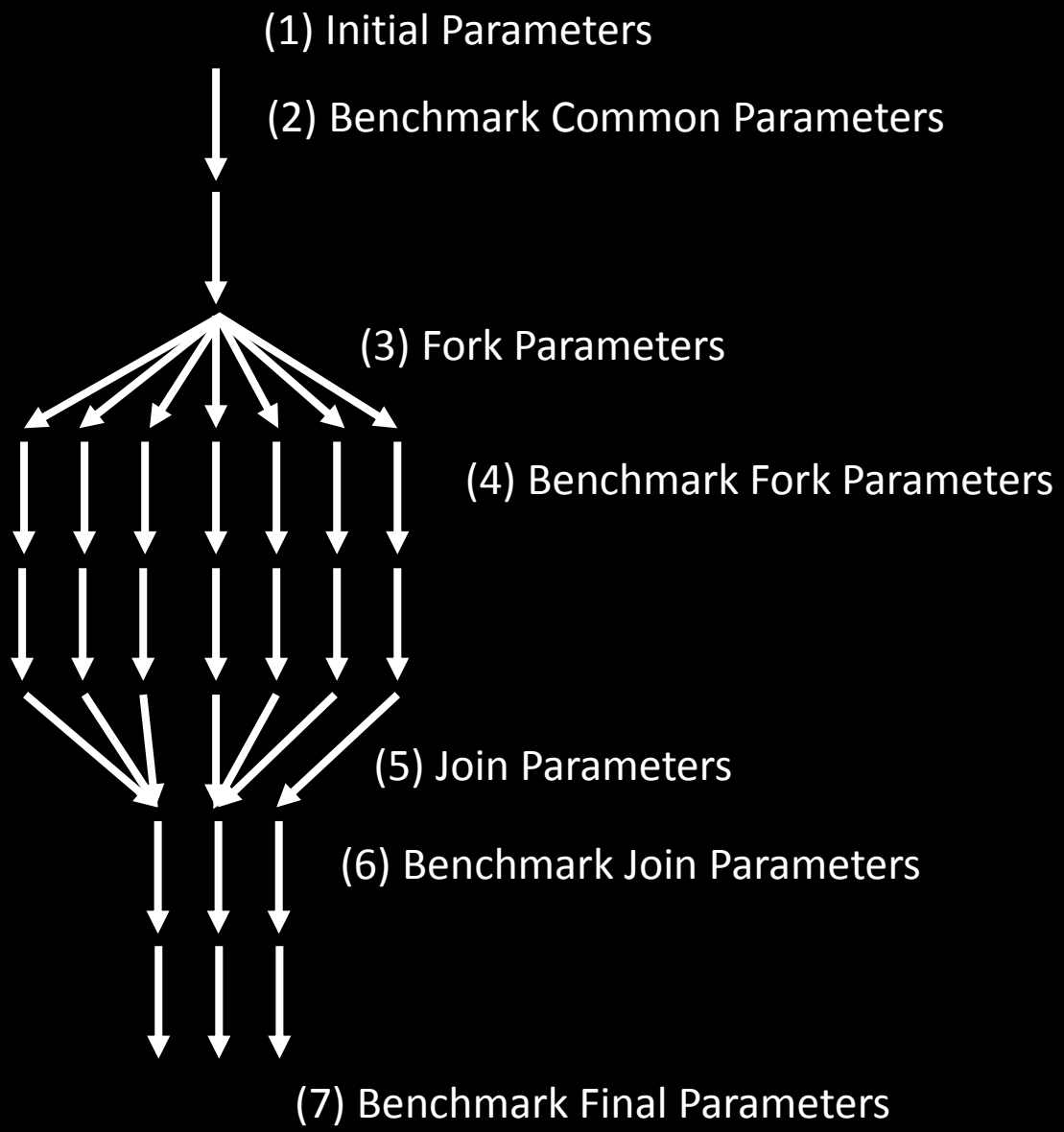
(9)

```
calculate addresses          read lds N-1
prefetch iter 0              wait global read
                             write lds
BEGIN LOOP                   swap lds red / black ptrs
load global iter 1           wait read lds N-2
read lds 1                   FMAs N-2
wait lds read 0              wait write lds & N-1
FMAs 0                       barrier
read lds 2                   read lds iter 1 subiter 0
wait lds read 1              FMAs N-1
FMAs 1                       END LOOP
…            (13)
                             write vgprs to C     (14)
```

KERNEL PARAMETERS AFFECT GPU PERFORMANCE



AUTO-TUNING



- Steps 1-6 create a set of candidates which will be benchmarked against all problem sizes.
- User can limit final kernels to keep backend library size manageable. Otherwise each problem size could have own custom kernel.
- User can create multiple sets of candidates and benchmark each against a set of problem sizes.
  - small/skinny tiles for small/skinny problem sizes
  - large tiles for large problem sizes

MAPPING PROBLEM SIZES TO KERNELS

```
// Exact Sizes
if (sizeI==101 && sizeJ==202 && sizeK==303)
        return function_ptr_0;
if (sizeI==4004 && sizeJ==5005 && sizeK==6006)
        return function_ptr_1;
```

```
// Range Sizes – Recursive Size Splitting
if (sizeI < threshold_I_0) {
        if (sizeJ < threshold_J_0) {
                if (sizeK < threshold_K_0)
                        return function_ptr_2;
                if (sizeK < threshold_K_1)
                        return function_ptr_3;
                // more K thresholds
                return function_ptr_4;
        }
        if (sizeJ < threshold_J_1) {
                // more K thresholds
        }
        // more J, K thresholds
}
if (sizeI < threshold_I_1) {
        // more J, K thresholds
}
// more I, J, K thresholds
```

EXPERIMENT 1 – 2D

AMD

- Set of 100 kernels
  - ThreadTile = 8x8, 8x4, 8x2, 4x8, 4x4, 4x2, 2x8, 2x4, 2x2
  - WorkGroup = 16x16x1, 16x8x2, 8x16x2, 8x8x4
  - GlocalSplitU = 1, 2, 4, 6, 8
  - DepthU = 8
  - VectorWidth = max
  - PrefetchGlocalRead = True
  - PrefetchLocalRead = True
  - WorkGroupMapping = 8
- Benchmark kernels against 2D range of sizes
  - M = 16 -> 5632
  - N = 16 -> 5632
  - K = 3104
- Analyze performance and kernel properties for each data point.

PERFORMANCE

AMD

M \ N	16	32	64	112	176	256	352	464	592	736	896	1072	1264	1472	1696	1936	2192	2464	2752	3056	3376	3712	4064	4432	4816	5216	5632
16	29	62	118	205	314	448	596	735	890	1,024	1,131	1,204	1,401	1,502	1,562	1,625	1,646	1,750	1,754	1,823	1,833	1,887	1,937	1,890	1,976	1,977	2,036
32	60	116	238	406	615	880	1,137	1,424	1,722	2,006	2,271	2,304	2,628	2,877	2,953	3,120	3,171	3,365	3,452	3,617	3,619	3,654	3,800	3,710	3,848	3,896	3,992
64	121	236	452	774	1,168	1,635	2,030	2,402	2,785	3,173	3,539	3,625	4,206	4,506	4,723	4,959	4,936	5,146	5,483	5,846	5,644	5,979	6,404	5,869	6,251	6,449	6,510
112	206	409	790	1,270	1,831	2,365	2,758	3,275	3,590	3,989	4,353	4,398	5,002	5,487	5,422	6,033	5,625	5,791	6,362	6,879	6,652	6,891	7,436	7,058	7,203	7,016	7,458
176	316	633	1,201	1,848	2,404	3,178	3,457	4,012	4,506	4,695	5,479	5,397	6,223	5,894	6,204	6,908	6,755	7,237	7,001	7,332	7,864	7,630	8,156	7,868	7,704	8,343	8,023
256	456	878	1,575	2,342	3,065	4,021	4,271	4,847	5,052	5,810	5,780	5,968	6,579	6,898	6,811	7,593	7,095	7,458	7,936	8,597	8,012	8,558	9,211	8,357	8,761	8,387	8,835
352	604	1,186	2,141	2,971	3,566	4,566	4,720	5,458	5,924	6,001	6,555	6,683	7,363	7,298	7,224	7,899	7,811	7,934	7,886	8,573	8,636	8,741	9,013	8,483	8,534	9,188	8,854
464	755	1,447	2,489	3,340	4,113	5,100	5,480	5,877	5,810	6,906	7,229	7,075	7,789	7,932	7,927	8,217	8,232	8,403	8,562	8,950	8,526	8,708	9,437	8,882	9,007	8,905	9,378
592	947	1,711	2,892	3,749	4,510	5,561	6,022	5,942	6,393	7,288	7,512	7,550	7,734	8,013	8,120	8,403	8,677	8,430	8,848	9,118	8,947	8,898	9,573	9,414	9,861	9,461	9,707
736	1,073	2,018	3,415	4,254	4,966	6,241	6,099	6,967	7,254	7,666	7,913	8,111	8,476	8,693	8,424	9,010	8,796	9,114	9,079	9,678	9,715	9,801	10,293	9,534	9,505	10,182	9,853
896	1,228	2,225	3,755	4,450	5,394	6,442	6,447	6,794	6,950	7,527	7,974	8,695	8,841	8,860	9,428	9,395	9,785	9,516	9,655	9,758	10,327	10,173	10,338	10,442	10,216	10,456	10,786
1072	1,304	2,380	3,895	4,616	5,536	6,392	6,786	7,242	7,635	8,173	8,763	8,442	9,037	9,050	9,180	9,259	9,446	9,620	9,836	10,038	9,745	10,054	10,334	10,281	10,016	10,399	10,338
1264	1,427	2,716	4,490	5,222	6,367	7,212	7,401	7,782	7,755	8,474	9,031	9,012	9,125	9,373	9,506	9,739	9,929	9,614	10,071	10,300	10,133	10,140	11,079	10,544	11,055	10,580	11,158
1472	1,509	2,924	4,700	5,562	6,147	7,643	7,476	7,889	7,948	8,619	8,981	8,968	9,299	9,585	9,790	9,854	9,897	9,804	10,213	10,185	10,124	10,565	10,824	10,323	10,434	10,564	10,728
1696	1,608	3,128	4,896	5,594	6,562	7,673	7,503	8,015	8,196	8,409	9,484	9,100	9,521	9,792	9,641	9,605	9,943	9,926	10,056	10,052	10,381	10,304	10,753	10,358	10,234	10,841	10,593
1936	1,673	3,162	5,085	6,223	6,961	8,097	7,939	8,244	8,373	9,018	9,442	9,245	9,734	9,894	9,617	10,001	10,076	10,313	10,328	10,756	10,301	10,183	10,874	10,575	10,411	10,514	11,049
2192	1,666	3,291	5,152	6,069	7,078	8,025	7,870	8,285	8,692	8,773	9,876	9,419	9,929	9,894	9,946	10,081	9,841	10,083	10,276	10,515	10,485	10,543	10,970	10,806	10,726	10,663	10,651
2464	1,746	3,438	5,511	6,385	7,275	8,370	8,098	8,424	8,438	9,103	9,541	9,608	9,661	9,778	9,924	10,247	10,056	10,339	10,607	10,577	10,396	10,549	11,076	11,013	11,005	11,025	11,065
2752	1,805	3,501	5,703	6,353	7,109	8,311	8,062	8,530	8,825	9,046	9,846	9,765	10,032	10,185	10,053	10,301	10,260	10,617	10,726	10,485	10,647	11,294	11,298	10,608	10,669	10,732	10,868
3056	1,848	3,623	5,879	6,856	7,724	8,901	8,576	8,932	9,107	9,681	9,918	10,008	10,310	10,268	10,040	10,734	10,520	10,516	10,508	11,474	10,751	11,497	11,552	10,864	11,024	11,207	11,393
3376	1,836	3,570	5,910	6,907	7,934	8,733	8,671	8,510	8,930	9,739	10,468	9,701	10,125	10,145	10,398	10,296	10,481	10,394	10,669	10,722	10,744	10,801	10,989	11,199	10,775	11,037	11,290
3712	1,890	3,701	6,048	6,773	7,709	8,632	8,643	8,630	8,760	9,770	10,161	9,957	10,041	10,525	10,275	10,136	10,441	10,462	11,227	11,372	10,682	10,935	11,159	11,420	11,065	11,343	11,700
4064	1,948	3,824	6,511	7,413	8,252	9,440	8,982	9,380	9,553	10,282	10,555	10,289	11,005	10,828	10,748	10,806	10,915	11,081	11,257	11,483	10,935	11,228	11,521	11,210	11,548	11,328	11,685
4432	1,938	3,700	6,049	7,225	7,945	9,047	8,473	8,873	9,381	9,555	10,561	10,279	10,599	10,338	10,382	10,563	10,819	11,061	10,643	10,856	11,197	11,554	11,277	11,100	11,480	11,371	11,310
4816	1,966	3,819	6,334	7,244	7,879	9,041	8,608	9,070	9,868	9,534	10,245	9,994	11,052	10,466	10,211	10,420	10,701	11,044	10,694	11,023	10,786	11,199	11,605	11,480	11,416	11,375	11,396
5216	1,976	3,920	6,601	7,198	8,310	8,942	9,198	8,960	9,429	10,168	10,722	10,314	10,655	10,538	10,822	10,464	10,578	11,008	10,755	11,146	10,974	11,427	11,338	11,310	11,311	11,364	11,433
5632	2,034	3,996	6,180	7,169	7,800	9,051	8,528	9,295	9,440	9,765	10,740	10,183	10,998	10,577	10,521	10,906	10,524	10,999	10,797	11,250	11,160	11,654	11,602	11,166	11,244	11,355	11,513

K = 3104

GFlops

WINNERS

AMD

N

M

	16	32	64	112	176	256	352	464	592	736	896	1072	1264	1472	1696	1936	2192	2464	2752	3056	3376	3712	4064	4432	4816	5216	5632
16	3	3	3	3	3	3	3	3	1	3	1	7	11	26	1	26	1	26	26	18	18	14	18	15	15	1	15
32	3	3	3	3	3	3	3	3	3	3	1	7	1	18	1	18	1	18	14	18	18	14	18	15	15	29	15
64	3	3	3	3	3	3	3	12	7	12	7	12	12	14	23	30	12	0	30	30	23	30	30	0	30	5	5
112	8	3	3	3	3	1	12	23	10	0	28	25	25	28	28	28	25	22	5	5	28	5	5	28	5	28	28
176	3	3	3	3	23	4	23	13	28	25	25	28	28	28	5	5	28	5	28	28	5	28	5	5	6	6	5
256	8	3	3	1	26	14	9	10	17	22	22	17	22	5	22	24	22	5	24	24	5	6	6	5	24	5	24
352	8	3	3	4	4	2	25	28	28	25	5	28	5	28	28	5	5	6	5	5	5	5	20	24	27	27	5
464	3	3	1	7	4	10	28	28	30	5	28	28	5	5	5	20	5	5	5	20	24	5	27	24	5	5	24
592	3	3	7	4	25	25	28	25	30	5	28	28	28	20	5	5	5	5	24	27	5	5	20	24	27	24	20
736	3	3	4	2	25	28	25	5	5	28	28	28	20	5	5	20	24	5	5	24	24	24	27	20	5	27	24
896	11	1	4	2	17	2	21	17	17	28	28	6	5	5	24	24	27	24	24	24	27	24	6	27	24	6	27
1072	8	1	4	25	28	25	25	28	28	28	6	5	5	5	27	5	5	5	5	24	5	6	24	27	24	6	6
1264	11	1	4	25	28	28	5	5	28	6	5	5	5	27	5	20	24	5	20	24	24	27	27	20	27	20	27
1472	11	18	4	28	25	5	25	5	6	5	28	5	27	5	24	27	24	5	6	20	6	24	27	6	6	6	6
1696	1	1	7	10	25	28	25	5	28	28	24	27	5	24	5	20	24	24	27	24	27	24	27	6	20	27	27
1936	11	1	17	28	28	5	5	20	5	20	24	5	20	27	20	27	24	27	24	27	24	24	27	27	24	6	27
2192	1	26	12	25	25	28	5	5	5	24	27	5	24	24	24	24	6	16	24	27	24	6	27	27	27	27	27
2464	26	26	17	28	5	5	28	5	5	27	24	5	5	5	5	27	6	24	27	20	24	6	27	27	27	27	27
2752	26	26	28	5	28	28	28	5	24	5	5	5	20	24	27	24	24	27	24	24	24	27	27	20	20	20	27
3056	18	26	22	28	28	24	5	20	27	24	5	24	24	20	24	27	27	20	24	27	16	27	27	27	27	27	27
3376	11	11	28	25	5	5	5	5	5	6	27	5	24	16	27	24	24	24	24	6	24	27	27	27	27	27	27
3712	3	11	22	5	28	6	5	27	5	24	24	24	27	24	24	24	24	24	27	27	27	27	27	27	27	27	27
4064	18	18	22	5	5	6	20	27	20	27	20	24	27	27	27	27	27	27	27	27	27	27	27	27	27	27	27
4432	8	15	17	28	5	5	24	24	24	20	27	27	20	16	16	27	27	27	16	27	27	27	27	27	27	27	27
4816	26	15	22	5	28	28	5	5	27	5	24	24	27	16	24	20	27	27	16	27	27	27	27	27	27	27	27
5216	11	19	5	28	5	28	27	5	24	27	20	24	20	27	27	16	27	27	20	27	27	27	27	27	27	27	27
5632	19	19	22	24	5	5	24	24	6	24	27	24	27	24	27	27	27	27	27	27	27	27	27	27	27	27	27

Chaotic behavior necessitates recursive size splitting for map.

FLOPS/BYTE OF TILE

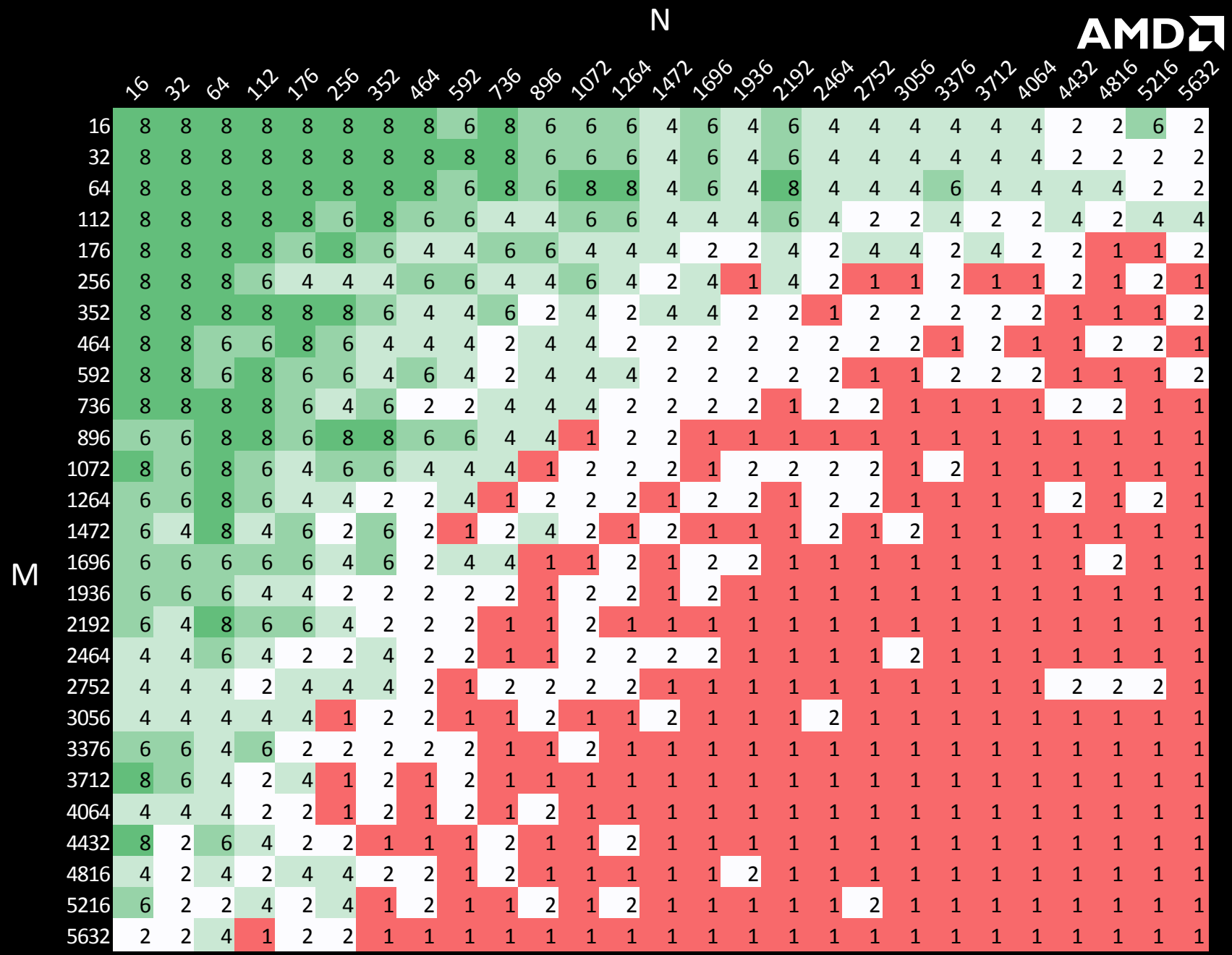
Flops/Byte = 2*MT0*MT1 / (MT0+MT1)/4

For example 128x128 tile:

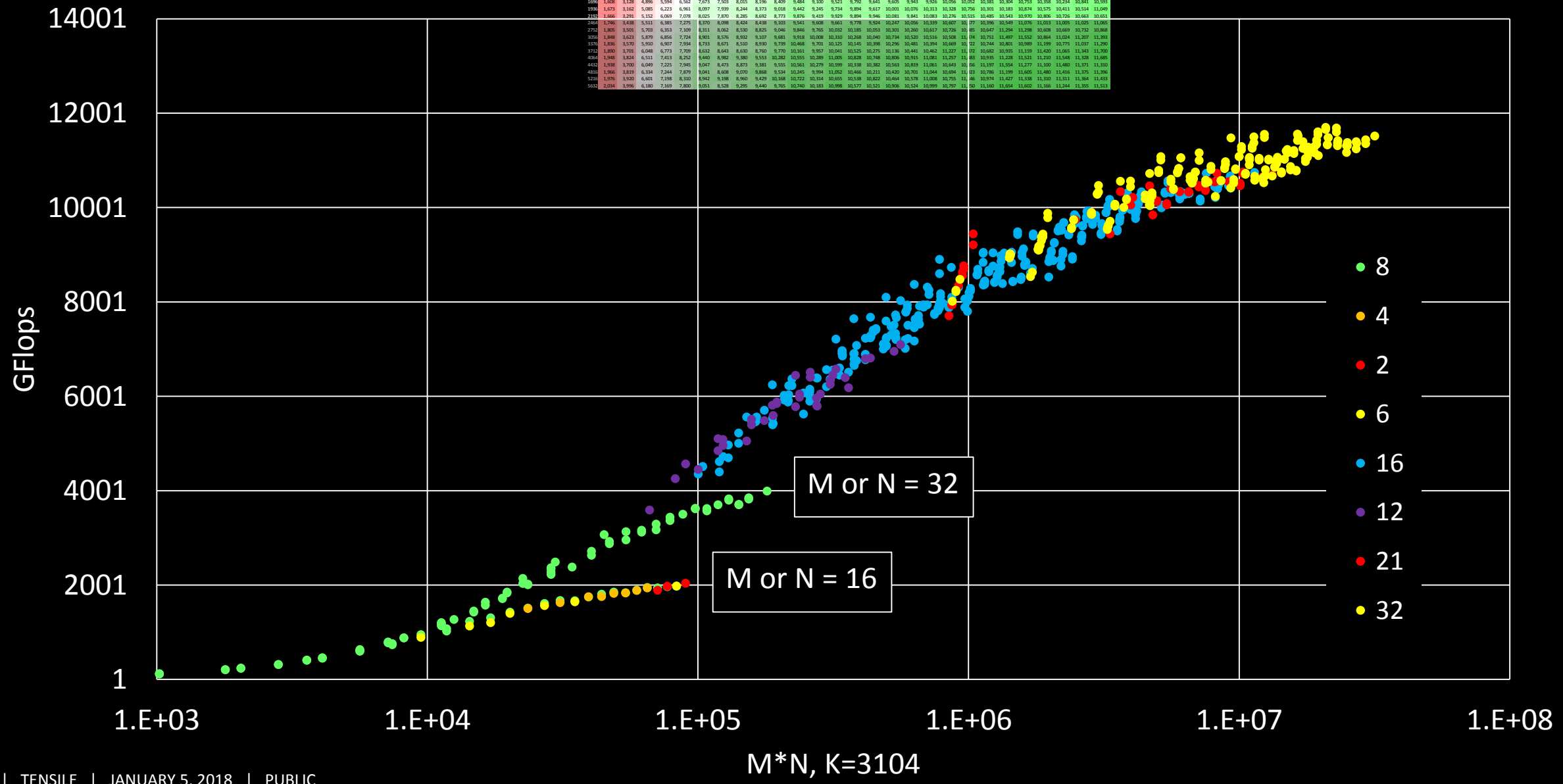
2*128*128/(128+128)/4 = 32

M \ N	16	32	64	112	176	256	352	464	592	736	896	1072	1264	1472	1696	1936	2192	2464	2752	3056	3376	3712	4064	4432	4816	5216	5632
16	8	8	8	8	8	8	8	8	6	8	6	6	6	4	6	4	6	4	4	4	4	4	4	2	2	6	2
32	8	8	8	8	8	8	8	8	8	8	8	11	8	8	8	8	8	8	11	8	8	11	8	8	8	11	8
64	8	8	8	8	8	8	8	11	11	11	11	11	11	11	11	13	11	11	13	13	11	13	13	11	13	16	16
112	8	8	8	8	8	8	11	11	13	11	16	16	16	16	16	16	16	13	16	16	16	16	16	16	16	16	16
176	8	8	8	8	11	11	11	11	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	21	21	16
256	8	8	8	8	8	11	11	13	13	13	13	13	13	16	13	16	13	16	16	16	16	21	21	16	16	16	16
352	8	8	8	11	11	13	16	16	16	16	16	16	16	16	16	16	16	21	16	16	16	16	32	16	32	32	16
464	8	8	8	11	11	13	16	16	13	16	16	16	16	16	16	32	16	16	16	32	16	16	32	16	16	16	16
592	8	8	11	11	16	16	16	16	13	16	16	16	16	32	16	16	16	16	16	32	16	16	32	16	32	16	32
736	8	8	11	13	16	16	16	16	16	16	16	16	32	16	16	32	16	16	16	16	16	16	32	32	16	32	16
896	8	8	11	13	13	13	13	13	13	16	16	21	16	16	16	16	32	16	16	16	32	16	21	32	16	21	32
1072	8	8	11	16	16	16	16	16	16	16	21	16	16	16	32	16	16	16	16	16	16	21	16	32	16	21	21
1264	8	8	11	16	16	16	16	16	16	21	16	16	16	32	16	32	16	16	32	16	16	32	32	32	32	32	32
1472	8	8	11	16	16	16	16	16	21	16	16	16	32	16	16	32	16	16	21	32	21	16	32	21	21	21	21
1696	8	8	11	13	16	16	16	16	16	16	16	32	16	16	16	32	16	16	32	16	32	16	32	21	32	32	32
1936	8	8	13	16	16	16	16	32	16	32	16	16	32	32	32	32	16	32	16	32	16	16	32	32	16	21	32
2192	8	8	11	16	16	16	16	16	16	16	32	16	16	16	16	16	21	21	16	32	16	21	32	32	32	32	32
2464	8	8	13	16	16	16	16	16	16	32	16	16	16	16	16	32	21	16	32	32	16	21	32	32	32	32	32
2752	8	8	16	16	16	16	16	16	16	16	16	16	32	16	32	16	16	32	16	16	16	32	32	32	32	32	32
3056	8	8	13	16	16	16	16	32	32	16	16	16	16	32	16	32	32	32	16	32	21	32	32	32	32	32	32
3376	8	8	16	16	16	16	16	16	16	21	32	16	16	21	32	16	16	16	16	21	16	32	32	32	32	32	32
3712	8	8	13	16	16	21	16	32	16	16	16	16	32	16	16	16	16	16	32	32	32	32	32	32	32	32	32
4064	8	8	13	16	16	21	32	32	32	32	32	16	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32
4432	8	8	13	16	16	16	16	16	16	32	32	32	32	21	21	32	32	32	21	32	32	32	32	32	32	32	32
4816	8	8	13	16	16	16	16	16	32	16	16	16	32	21	16	32	32	32	21	32	32	32	32	32	32	32	32
5216	8	11	16	16	16	16	32	16	16	32	32	16	32	32	32	21	32	32	32	32	32	32	32	32	32	32	32
5632	11	11	13	16	16	16	16	16	21	16	32	16	32	16	32	32	32	32	32	32	32	32	32	32	32	32	32

GLOBAL SPLIT U



PERFORMANCE VS M*N

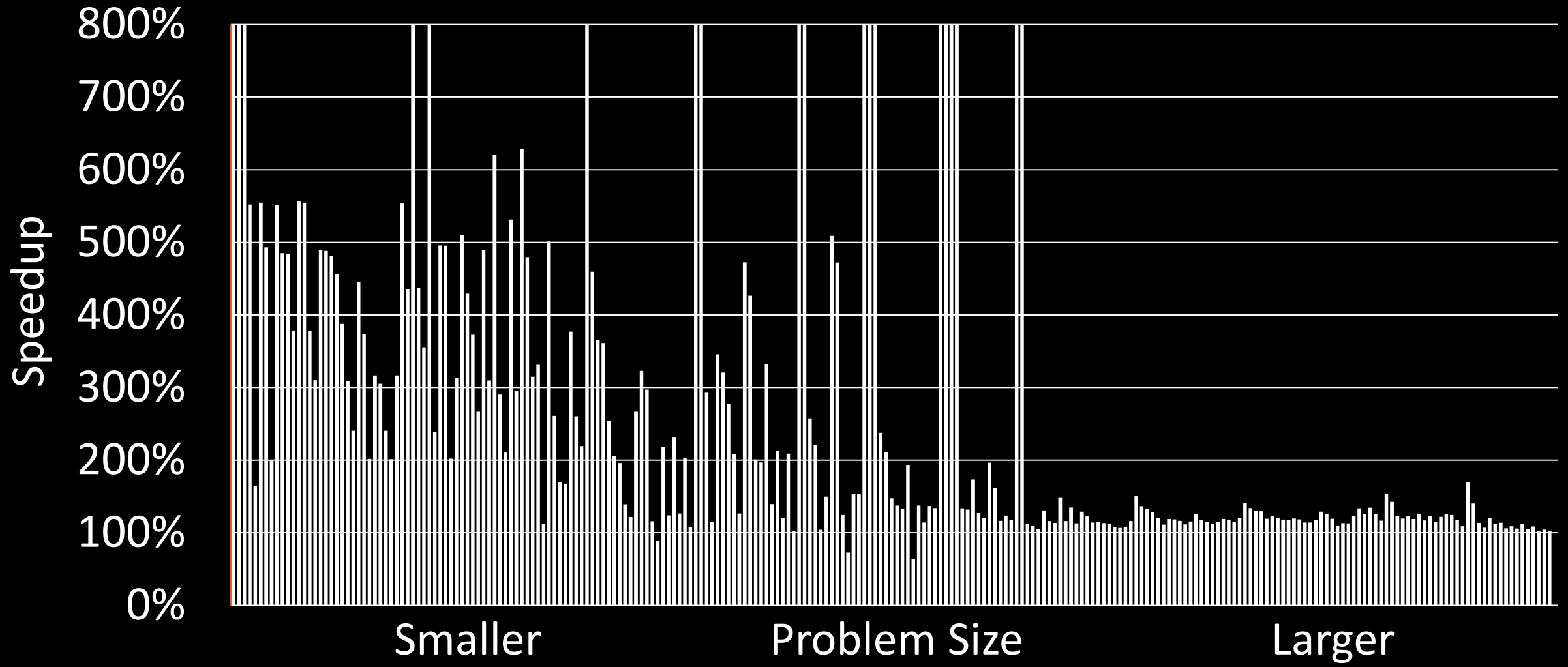


EXPERIMENT 2 - DEEPBENCH

AMD

- Set of thousands of kernels.
- Benchmark kernels against ~250 problem sizes of DeepBench.
- Analyze speedup of kernel tuned for exact problem size vs “fastest” 128x128 tile kernel.

SPEEDUP OF DEEPBENCH SIZES



100% means equal performance

FUTURE WORK