Efficient Binary Instrumentation for GPU Profiling

Hsuan-Heng Wu

Scalable Tools Workshop 2022

Granlibakken Resort
Lake Tahoe, California

Motivation

Measure fine-grained performance data of GPU kernels with low cost

Our slowdown is 1.15x when measuring branch divergence …

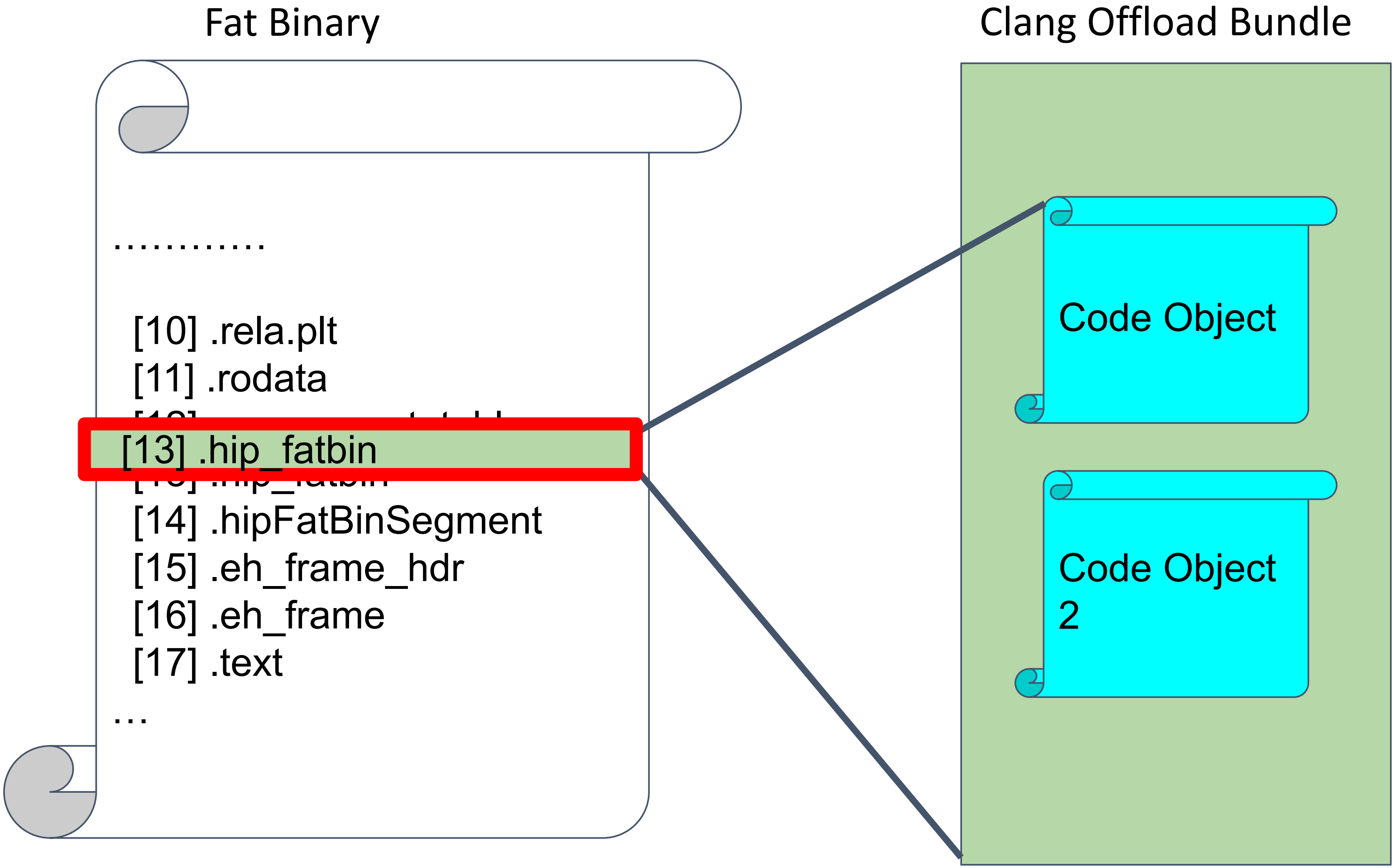
… compared to 26x slowdown when using CUPTI

Provide fine-grained (per branch per wavefront) instrumentation …

… while CUPTI & GTPIN report aggregated per branch data

AMD GPU Features

- SIMT programming model
- Threads
  - Own Vector General Purpose Registers (VGPR)
- Workgroups
  - Threads that can communicate through shared memory
  - Consists of *wavefronts*
- Wavefronts
  - 64 threads sharing the same program counter/ SGPR
  - On/off controlled by EXEC register (mask)



Fat Binary

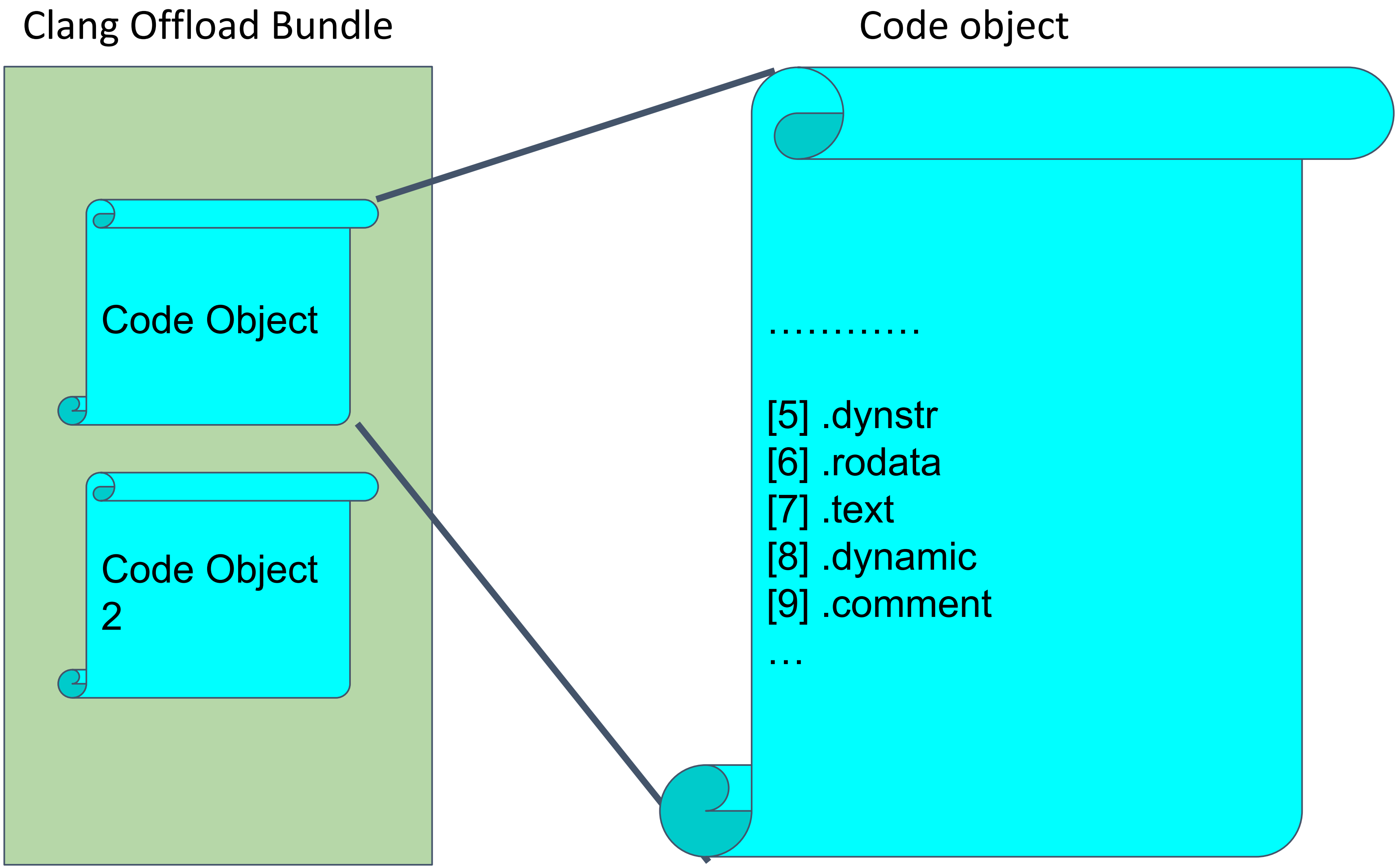
…………

[10] .rela.plt
[11] .rodata
[13] .hip_fatbin
[14] .hipFatBinSegment
[15] .eh_frame_hdr
[16] .eh_frame
[17] .text
…

Clang Offload Bundle

Code Object

Code Object 2



Clang Offload Bundle

Code Object

Code Object 2

Code object

....

[5] .dynstr
[6] .rodata
[7] .text
[8] .dynamic
[9] .comment
...

Code Objects

One code object for each compilation unit x #(GPU Architecture)

Standard ELF format

Kernel binaries in .text

Kernel descriptors in .rodata

Metadata in .comment

Kernel Descriptor

Resides in .rodata

Data used to initiate execution of kernels

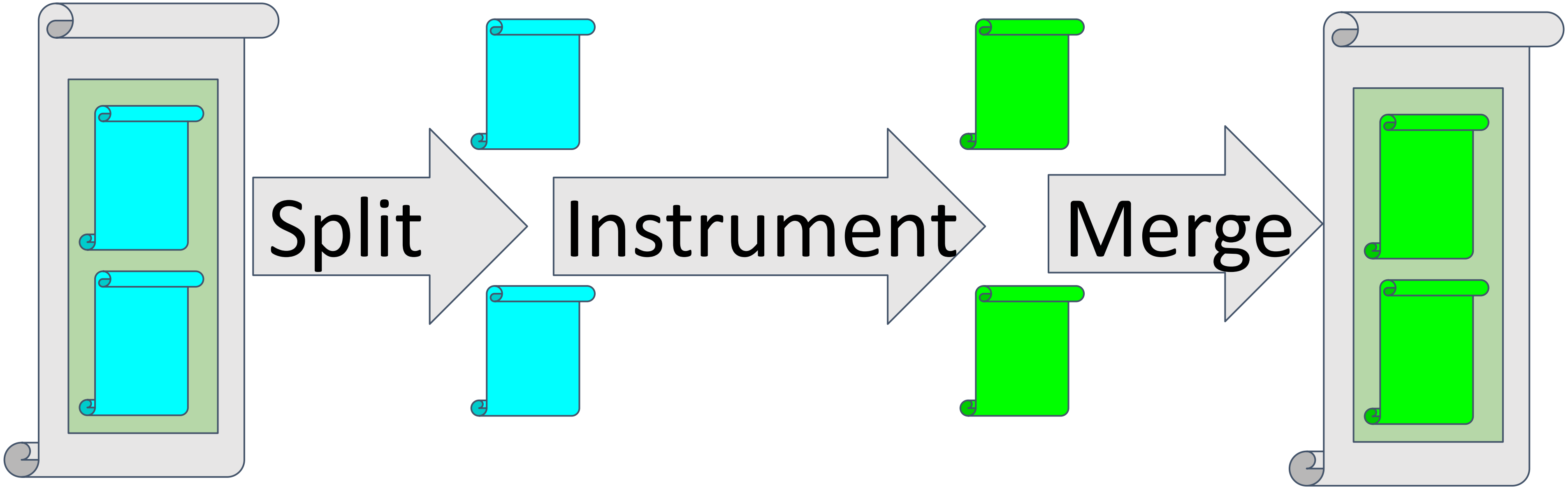
Specifies resources required by a kernel

- SGPR Usage
- VGPR Usage

Options that affects GPR initialization at runtime

- ENABLE_SGPR_DISPATCH_PTR
- ENABLE_VGPR_WORKITEM_ID
- …

Current Instrumentation Workflow



Instrumentation for Branch Divergence

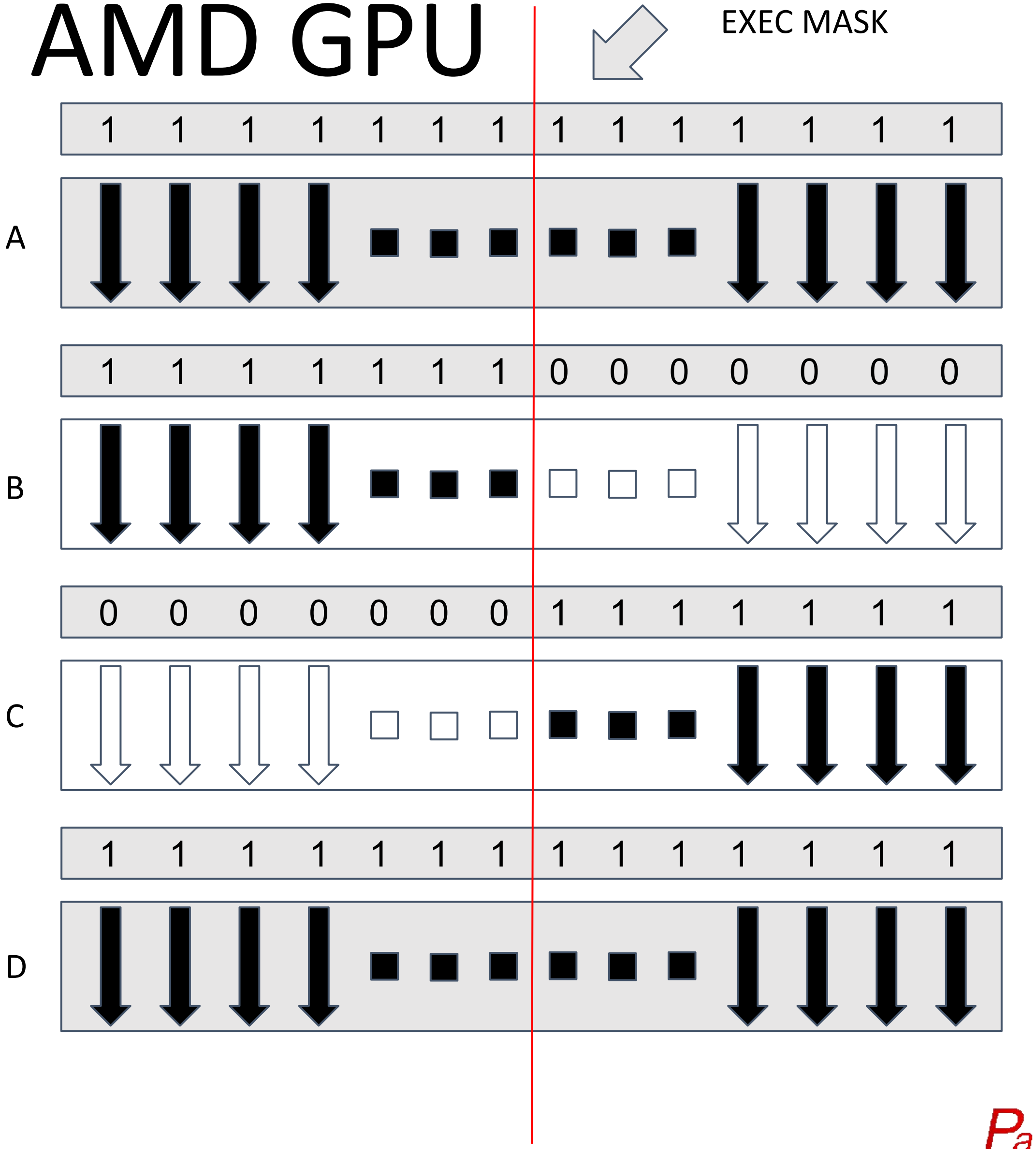
Why Branch Divergence

Some threads won’t be working actively

Provides Optimization Opportunity

Branching in an AMD GPU

```
A;

if(threadIdx.x < 32){

    B;


} else {
    C;

}
D;
```



1 1 1 1 1 1 1 1 1 1 1 1 1 1

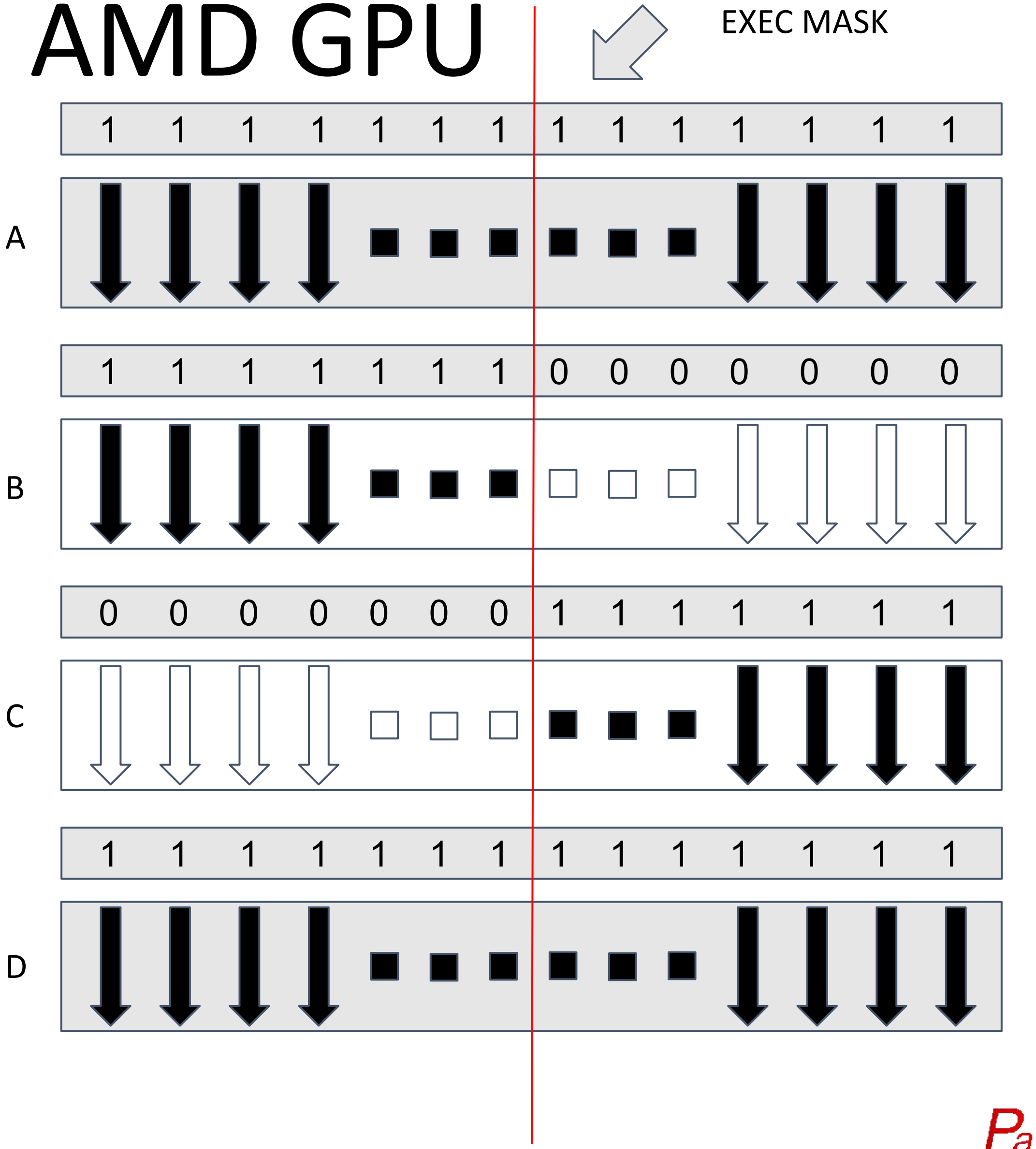
1 1 1 1 1 1 1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1

Branching in an AMD GPU

```
A;

if(threadIdx.x < 32){

    B;


} else {
    C;

}
D;
```



1 1 1 1 1 1 1 1 1 1 1 1 1 1

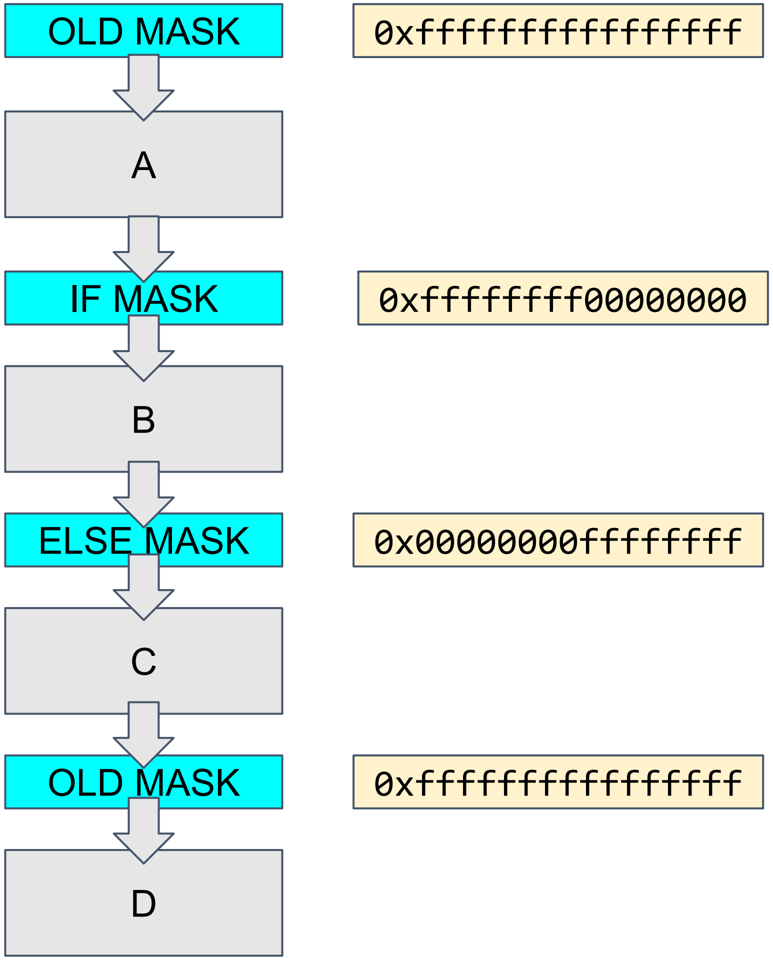
1 1 1 1 1 1 1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1

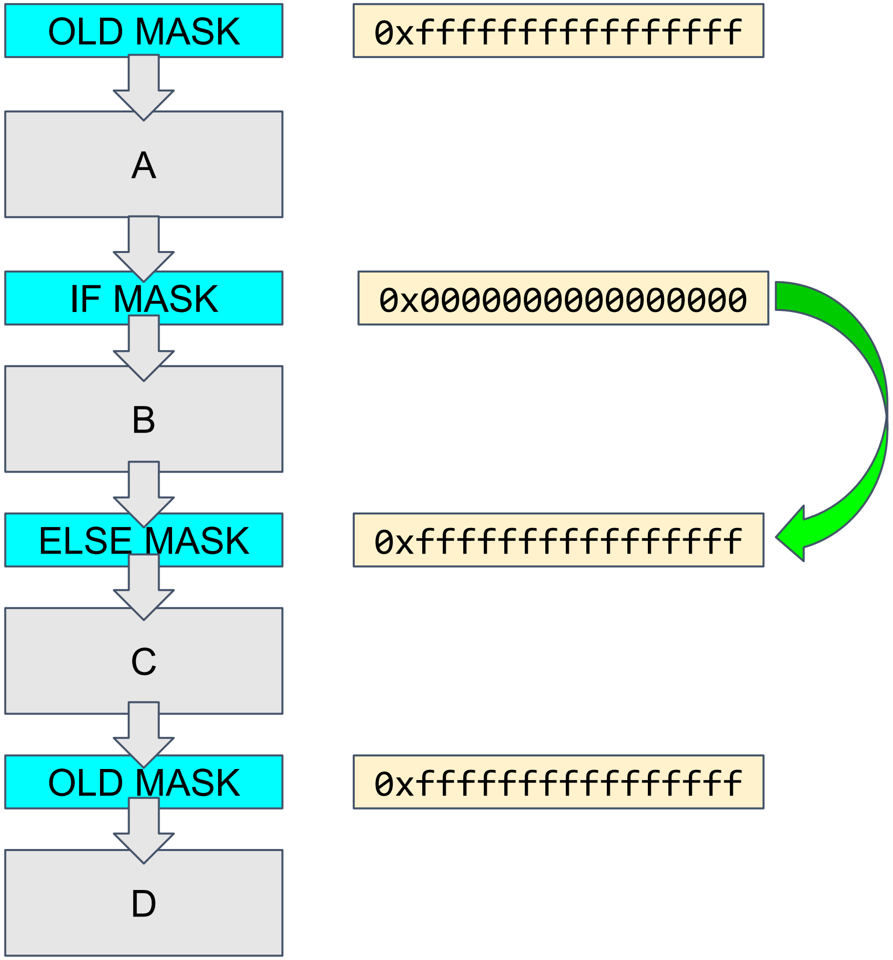
Example of Branch Divergence

```
A;

if(threadIdx.x < 32){

    B;


} else {
    C;

}
D;
```



Example of Non-divergence

```
A;

if(threadIdx.x > 63){

    B;


} else {
    C;

}
D;
```



Branching in AMD GPU

```
A;

if(threadIdx.x < 32){

    B;


} else {
    C;

}
D;
```

```
BB0_1:
…
v_cmp_lt_i32_e32 VCC, s0 , v0
s_and_saveexec_b64 s[2:3], VCC
s_xor_b64 s[2:3], exec, s[2:3]
```

Comparison stored the result in VCC

AND VCC with OLD MASK to generate IF MASK

Save Information in S[2:3] to generate ELSE MASK later

Branching in AMD GPU

```
A;

if(threadIdx.x < 32){

    B;


} else {
    C;

}
D;
```

```
BB0_1:
…
v_cmp_lt_i32_e32 VCC, s0 , v0
s_and_saveexec_b64 s[2:3], VCC
s_xor_b64 s[2:3], exec, s[2:3]
```

```
CODE IN IF BRANCH
```

Branching in AMD GPU

```
A;

if(threadIdx.x < 32){

    B;


} else {
    C;

}
D;
```

```
BB0_1:
…
v_cmp_lt_i32_e32 VCC, s0 , v0
s_and_saveexec_b64 s[2:3], VCC
s_xor_b64 s[2:3], exec, s[2:3]
```

```
CODE IN IF BRANCH
```

```
BB0_2:
s_or_saveexec_b64 s[2:3],s[2:3]
s_xor_b64 exec, exec, s[2:3]
```

Generate ELSE MASK

Branching in AMD GPU

```
A;

if(threadIdx.x < 32){

    B;


} else {
    C;

}
D;
```

```
BB0_1:
…
v_cmp_lt_i32_e32 VCC, s0 , v0
s_and_saveexec_b64 s[2:3], VCC
s_xor_b64 s[2:3], exec, s[2:3]
```

```
CODE IN IF BRANCH
```

```
BB0_2:
s_or_saveexec_b64 s[2:3],s[2:3]
s_xor_b64 exec, exec, s[2:3]
```

```
CODE IN ELSE BRANCH
```

Branching in AMD GPU

```
A;

if(threadIdx.x < 32){

    B;


} else {
    C;

}
D;
```

```
BB0_1:
…
v_cmp_lt_i32_e32 VCC, s0 , v0
s_and_saveexec_b64 s[2:3], VCC
s_xor_b64 s[2:3], exec, s[2:3]
```

```
CODE IN IF BRANCH
```

```
BB0_2:
s_or_saveexec_b64 s[2:3],s[2:3]
s_xor_b64 exec, exec, s[2:3]
```

```
CODE IN ELSE BRANCH
```

Recover OLD MASK

```
BB0_3:
s_or_b64 exec, exec , s[2:3]
```

Branching in AMD GPU

```
A;

if(threadIdx.x < 32){

    B;


} else {
    C;

}
D;
```

skip if block if no thread is active

```
BB0_1:
…
v_cmp_lt_i32_e32 VCC, s0 , v0
s_and_saveexec_b64 s[2:3], VCC
s_xor_b64 s[2:3], exec, s[2:3]
s_cbranch_execz BB0_2
```

```
CODE IN IF BRANCH
```

```
BB0_2:
s_or_saveexec_b64 s[2:3],s[2:3]
s_xor_b64 exec, exec, s[2:3]
```

```
CODE IN ELSE BRANCH
```

```
BB0_3:
s_or_b64 exec, exec , s[2:3]
```

Branching in AMD GPU

```
A;

if(threadIdx.x < 32){

    B;


} else {
    C;

}
D;
```

skip else block if no thread is active

```
BB0_1:
…
v_cmp_lt_i32_e32 VCC, s0 , v0
s_and_saveexec_b64 s[2:3], VCC
s_xor_b64 s[2:3], exec, s[2:3]
s_cbranch_execz BB0_2
```

```
CODE IN IF BRANCH
```

```
BB0_2:
s_or_saveexec_b64 s[2:3],s[2:3]
s_xor_b64 exec, exec, s[2:3]
s_cbranch_execz BB0_3
```

```
CODE IN ELSE BRANCH
```

```
BB0_3:
s_or_b64 exec, exec , s[2:3]
```

How to detect divergence?

E: OLD MASK
F: Condition Code
IF MASK : (E & F)
ELSE MASK : (E & ~F)

If all threads take if branch:
OLD MASK == IF MASK

If all threads take the else branch:
OLD MASK == ELSE MASK

$$(E \,\&\, F) = E \Rightarrow (\text{IF MASK} = \text{OLD MASK})$$
$$E = (E \,\&\, {\sim}F) \Rightarrow (E \,\&\, F) = (E \,\&\, {\sim}F \,\&\, F) \Rightarrow (E \,\&\, F) = 0 \Rightarrow (\text{IF MASK} = 0)$$

# of Agrees = count[(IF MASK = OLD MASK) or (IF MASK = 0)]
# of Divergences = # of Executions - # of Agrees

What to Collect?

2 Counters for each branch for each wavefront

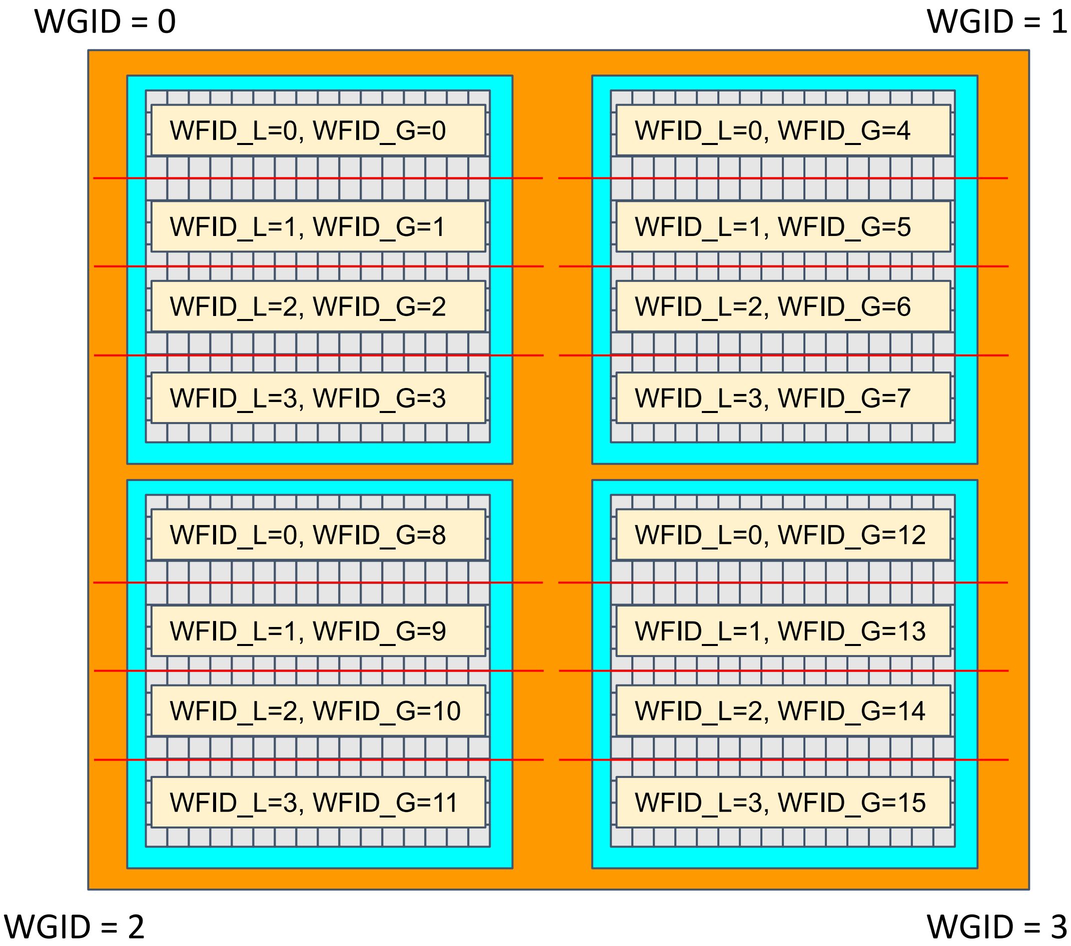
- 1 increments when all threads agree
- 1 increments whenever the branch is encountered

Wavefronts and Workgroups

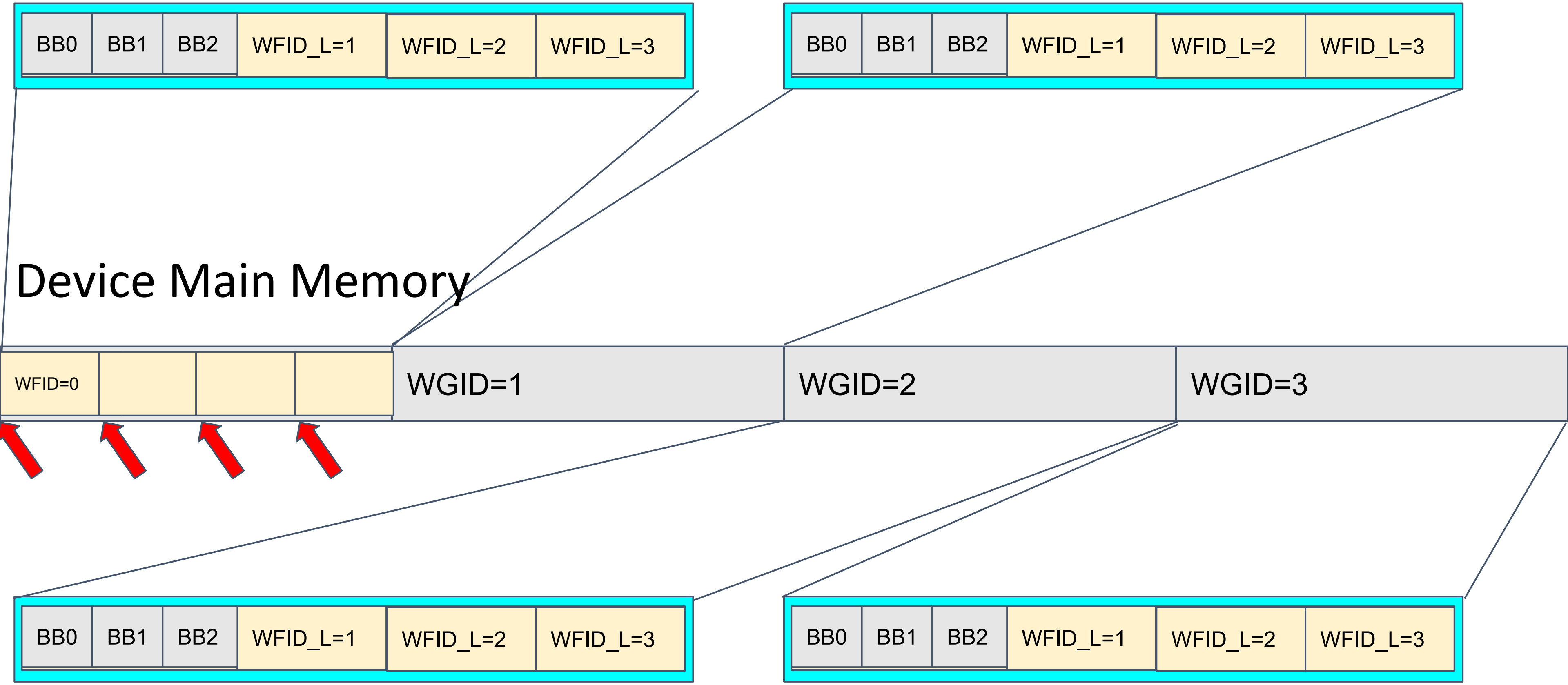
Each wavefront generates its own data

Each workgroup consists of multiple wavefronts

Generate a global wavefront ID that allows each wavefront to write in parallel



Complication - Wavefronts in Workgroups



Implementation - Branch Divergence Counters

After this instruction, EXEC is changed to IF MASK

```
s_and_saveexec_b64 s[16:17], s[0:1]
```

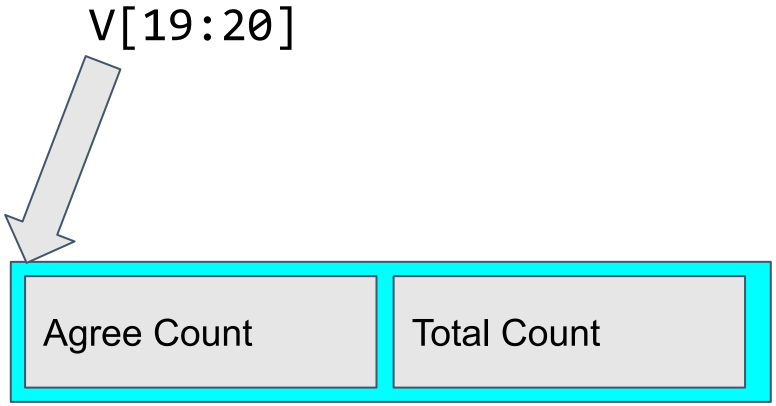
```
s_cbranch_execz BB0_3
```

Implementation - Branch Divergence Counters

```
s_and_saveexec_b64 s[16:17], s[0:1]
```

```
compute/store address in v[19:20]
backup exec in s[54:55]
change exec to 1
```

backup exec mask
compute index to
memory based on
wavefront ID


V[19:20]

Agree Count	Total Count

```
s_cbranch_execz BB0_3
```

Implementation - Branch Divergence Counters

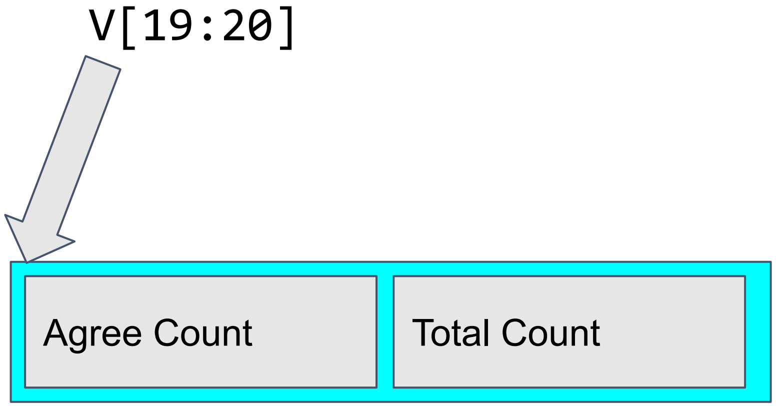
Compare IF MASK with EXEC
V56 = (IF MASK == EXEC) ? 1 : 0
Aggregate result in Agree Count

```
s_and_saveexec_b64 s[16:17], s[0:1]
```

```
compute/store address in v[19:20]
backup exec in s[54:55]
change exec to 1
```

```
s_cmp_eq_u64 s[16:17], s[54:55]
v_mov_b32_e32 v56, src_scc
global_atomic_add v[19:20],v56,off
```

```
s_cbranch_execz BB0_3
```



Implementation - Branch Divergence Counters

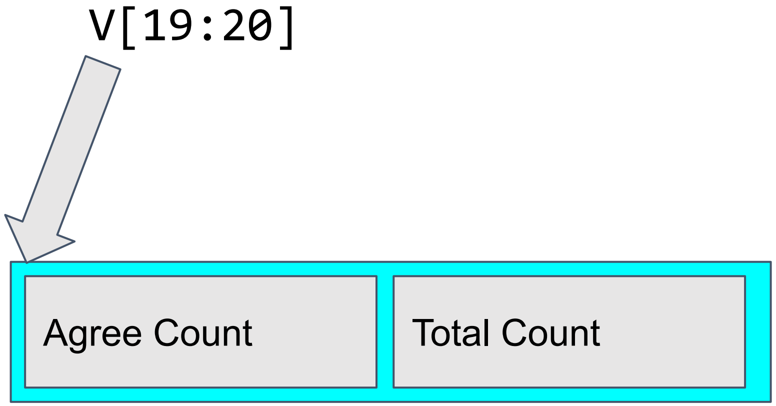
```
s_and_saveexec_b64 s[16:17], s[0:1]
```

```
compute/store address in v[19:20]
backup exec in s[54:55]
change exec to 1
```

Check if IF MASK is 0
V56 = (IF MASK == 0) ? 1 : 0
Aggregate result in Agree Count

```
s_cmp_eq_u64 s[16:17], s[54:55]
v_mov_b32_e32 v56, src_scc
global_atomic_add v[19:20],v56,off
```

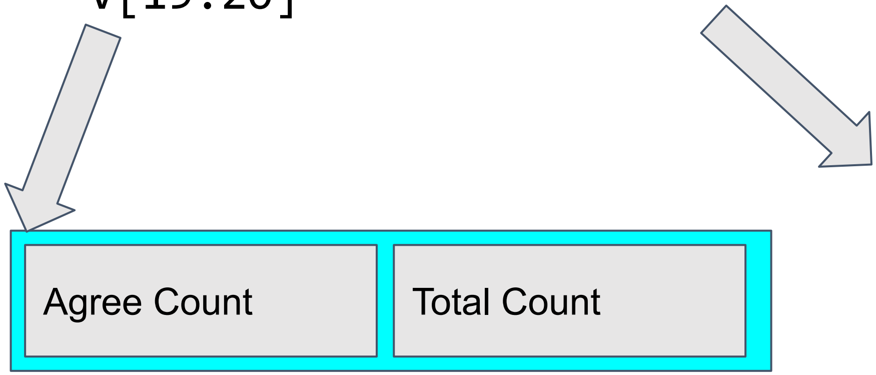
```
s_cmp_eq_u64 0, s[54:55]
v_mov_b32_e32 v56, src_scc
global_atomic_add v[19:20],v56,off
```



```
s_cbranch_execz BB0_3
```

Implementation - Branch Divergence Counters

```
s_and_saveexec_b64 s[16:17], s[0:1]

compute/store address in v[19:20]
backup exec in s[54:55]
change exec to 1

s_cmp_eq_u64 s[16:17], s[54:55]
v_mov_b32_e32 v56, src_scc
global_atomic_add v[19:20],v56,off

s_cmp_eq_u64 0, s[54:55]
v_mov_b32_e32 v56, src_scc
global_atomic_add v[19:20],v56,off

global_atomic_inc v[19:20],v18,off 4

s_cbranch_execz BB0_3
```



Increment Total Count

V[19:20]

Agree Count	Total Count

Implementation - Branch Divergence Counters

```
s_and_saveexec_b64 s[16:17], s[0:1]
```

```
compute/store address in v[19:20]
backup exec in s[54:55]
change exec to 1
```

```
s_cmp_eq_u64 s[16:17], s[54:55]
v_mov_b32_e32 v56, src_scc
global_atomic_add v[19:20],v56,off
```

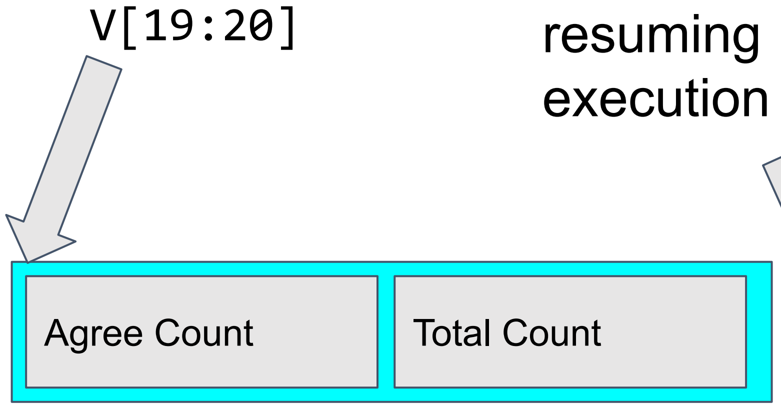
```
s_cmp_eq_u64 0, s[54:55]
v_mov_b32_e32 v56, src_scc
global_atomic_add v[19:20],v56,off
```

```
global_atomic_inc v[19:20],v18,off 4
```

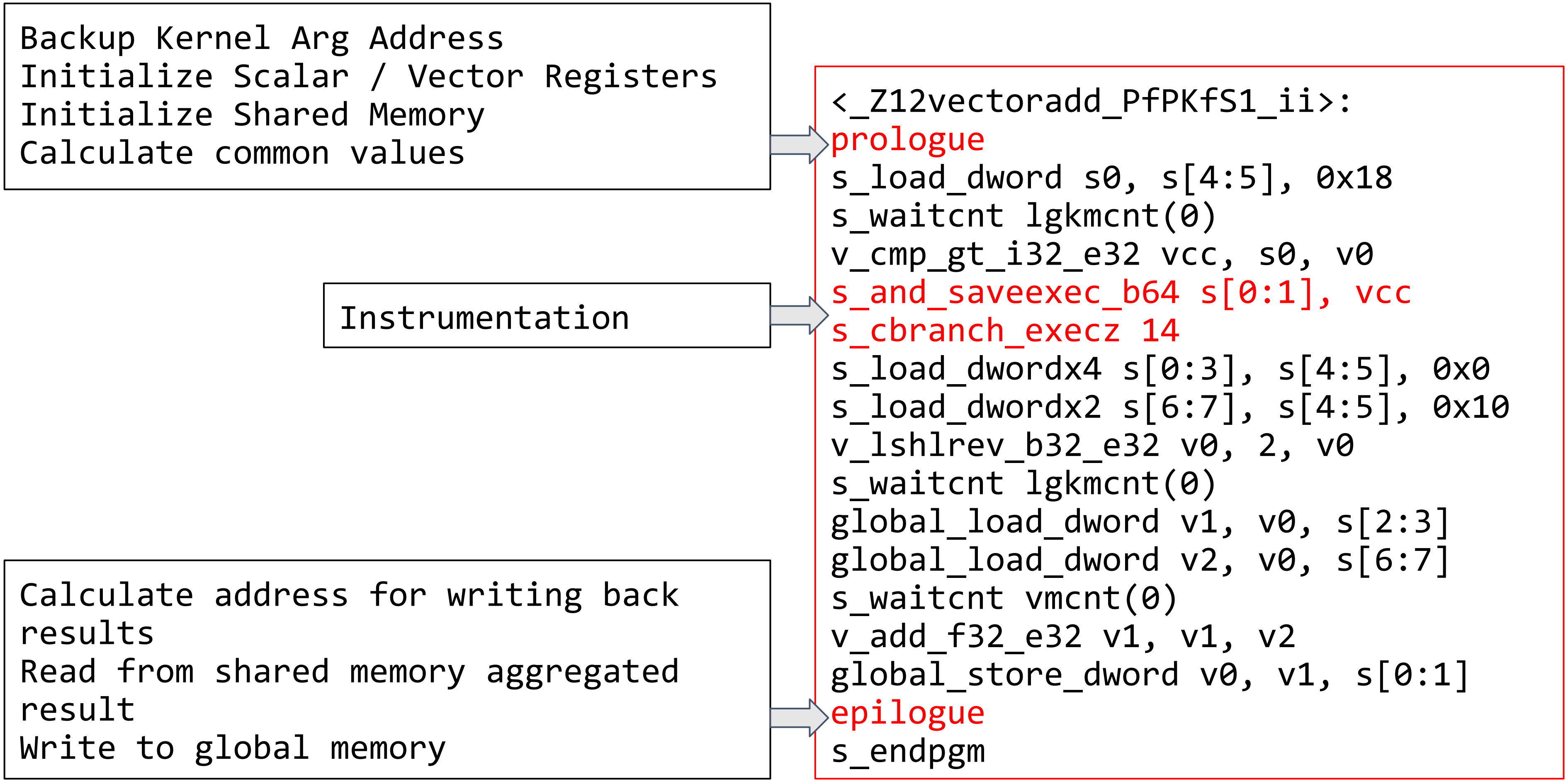
```
recover exec mask using s[54:55]
```

```
s_cbranch_execz BB0_3
```

recover exec mask before resuming execution



Implementation - Prologue / Epilogue



```
Backup Kernel Arg Address
Initialize Scalar / Vector Registers
Initialize Shared Memory
Calculate common values
```

```
Instrumentation
```

```
Calculate address for writing back
results
Read from shared memory aggregated
result
Write to global memory
```

```
<_Z12vectoradd_PfPKfS1_ii>:
prologue
s_load_dword s0, s[4:5], 0x18
s_waitcnt lgkmcnt(0)
v_cmp_gt_i32_e32 vcc, s0, v0
s_and_saveexec_b64 s[0:1], vcc
s_cbranch_execz 14
s_load_dwordx4 s[0:3], s[4:5], 0x0
s_load_dwordx2 s[6:7], s[4:5], 0x10
v_lshlrev_b32_e32 v0, 2, v0
s_waitcnt lgkmcnt(0)
global_load_dword v1, v0, s[2:3]
global_load_dword v2, v0, s[6:7]
s_waitcnt vmcnt(0)
v_add_f32_e32 v1, v1, v2
global_store_dword v0, v1, s[0:1]
epilogue
s_endpgm
```

Potential Optimization using Local Data Share (LDS)

Local Data Share (LDS)

- Fast on chip memory
- Allocation per block

Potential Optimization

- Aggregate results in LDS
- Writeback to global memory in the epilogue

Evaluation

Evaluation Setup

Compare device side measured time span with and without instrumentation

For CUPTI, we modify the source to collect timestamp using clock64 at the start/end of a kernel, and check how turning on CUPTI_ACTIVITY_KIND_BRANCH affects the collected value

For AMD GPU we surround the unmodified code blocks with timestamp reads as baseline

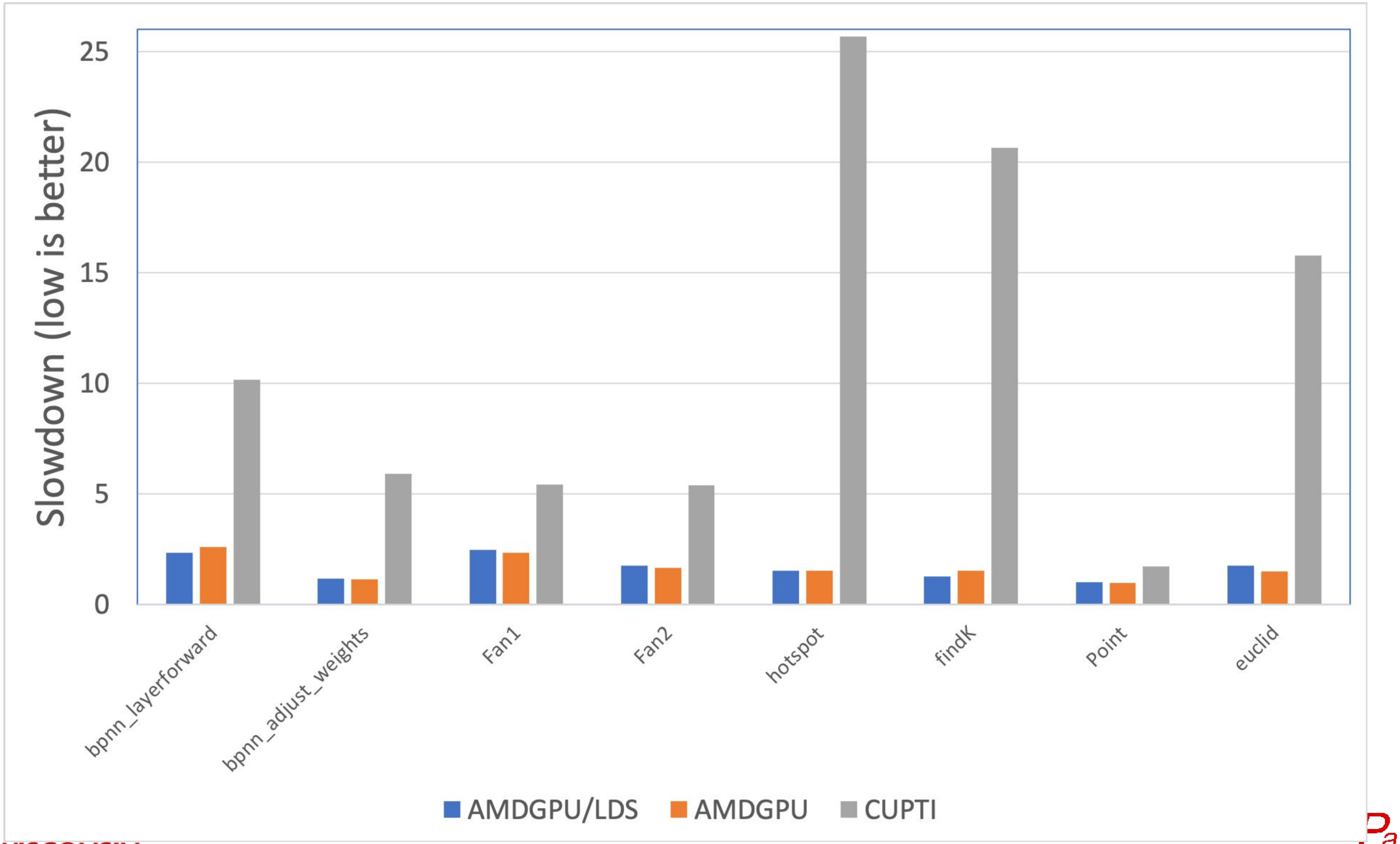
Result AMDGPU Raw data

Benchmark	Baseline	LDS	LDS slowdown	Global	Global slowdown
bpnn_layerforward	3901	9168	2.34x	10225	2.62x
bpnn_adjust_weights	7406	8850	1.19x	8525	1.15x
Fan1	1326	3281	2.47x	3974	2.36x
Fan2	1607	2844	1.77x	2681	1.67x
hotspot	10635	16376	1.54x	16265	1.53x

Result AMDGPU Raw data

Benchmark	Baseline	LDS	LDS slowdown	Global	Global slowdown
findK	9203	11861	1.29x	14016	1.53x
invert_mapping	586299	582997	0.99x	548837	0.94x
kmeansPoint	210740	215269	1.02x	209869	0.99x
euclid	3608	6392	1.77x	5447	1.51x

Slowdown caused by Instrumentation



Questions?

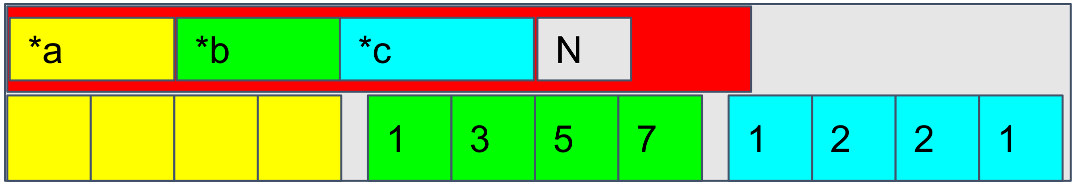
Extra Slides

Example Kernel : Vector Add

```
void vectoradd(float* a,
    float* b,  float* c, int N)
{

    int i = hipThreadIdx_x;
    if(i < N)
        a[i] = b[i] + c[i];
}
```

```
<_Z12vectoradd_PfPKfS1_ii>:
s_load_dword s0, s[4:5], 0x18
s_waitcnt lgkmcnt(0)
v_cmp_gt_i32_e32 vcc, s0, v0
s_and_saveexec_b64 s[0:1], vcc
s_cbranch_execz 14
s_load_dwordx4 s[0:3], s[4:5], 0x0
s_load_dwordx2 s[6:7], s[4:5], 0x10
v_lshlrev_b32_e32 v0, 2, v0
s_waitcnt lgkmcnt(0)
global_load_dword v1, v0, s[2:3]
global_load_dword v2, v0, s[6:7]
s_waitcnt vmcnt(0)
v_add_f32_e32 v1, v1, v2
global_store_dword v0, v1, s[0:1]
s_endpgm
```

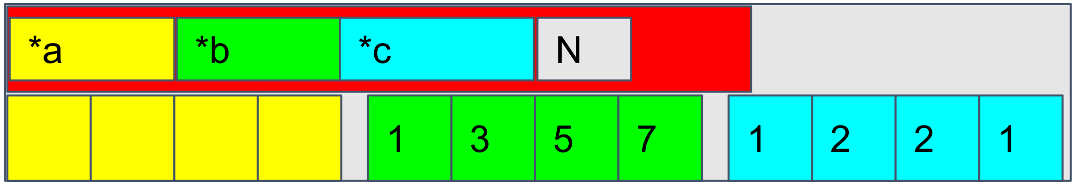
```
<_Z12vectoradd_PfPKfS1_ii>:
s_load_dword s0, s[4:5], 0x18
s_waitcnt lgkmcnt(0)
v_cmp_gt_i32_e32 vcc, s0, v0
s_and_saveexec_b64 s[0:1], vcc
s_cbranch_execz 14
s_load_dwordx4 s[0:3], s[4:5], 0x0
s_load_dwordx2 s[6:7], s[4:5], 0x10
v_lshlrev_b32_e32 v0, 2, v0
s_waitcnt lgkmcnt(0)
global_load_dword v1, v0, s[2:3]
global_load_dword v2, v0, s[6:7]
s_waitcnt vmcnt(0)
v_add_f32_e32 v1, v1, v2
global_store_dword v0, v1, s[0:1]
s_endpgm
```



s0	s1	s2	s3	s4	s5	s6	s7	vcc	exec
				*kernarg					

	0	1	2	3	…	…	63
v0	0	1	2	3	…	…	63
v1							
v2							

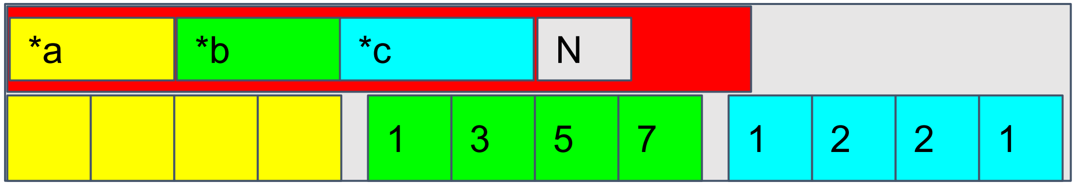
```
<_Z12vectoradd_PfPKfS1_ii>:
s_load_dword s0, s[4:5], 0x18
s_waitcnt lgkmcnt(0)
v_cmp_gt_i32_e32 vcc, s0, v0
s_and_saveexec_b64 s[0:1], vcc
s_cbranch_execz 14
s_load_dwordx4 s[0:3], s[4:5], 0x0
s_load_dwordx2 s[6:7], s[4:5], 0x10
v_lshlrev_b32_e32 v0, 2, v0
s_waitcnt lgkmcnt(0)
global_load_dword v1, v0, s[2:3]
global_load_dword v2, v0, s[6:7]
s_waitcnt vmcnt(0)
v_add_f32_e32 v1, v1, v2
global_store_dword v0, v1, s[0:1]
s_endpgm
```



s0	s1	s2	s3	s4	s5	s6	s7	vcc	exec
4				*kernarg					

	0	1	2	3	…	…	63
v0	0	1	2	3	…	…	63
v1							
v2							

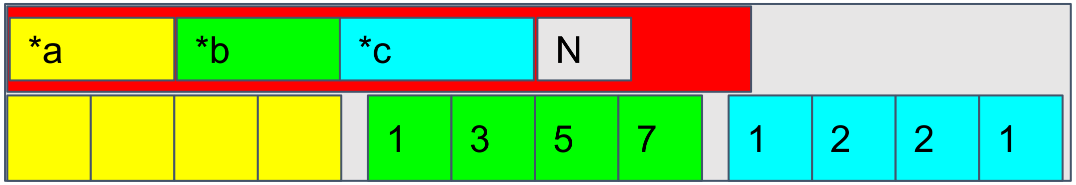
```
<_Z12vectoradd_PfPKfS1_ii>:
s_load_dword s0, s[4:5], 0x18
s_waitcnt lgkmcnt(0)
v_cmp_gt_i32_e32 vcc, s0, v0
s_and_saveexec_b64 s[0:1], vcc
s_cbranch_execz 14
s_load_dwordx4 s[0:3], s[4:5], 0x0
s_load_dwordx2 s[6:7], s[4:5], 0x10
v_lshlrev_b32_e32 v0, 2, v0
s_waitcnt lgkmcnt(0)
global_load_dword v1, v0, s[2:3]
global_load_dword v2, v0, s[6:7]
s_waitcnt vmcnt(0)
v_add_f32_e32 v1, v1, v2
global_store_dword v0, v1, s[0:1]
s_endpgm
```



s0	s1	s2	s3	s4	s5	s6	s7	vcc	exec
4				*kernarg				0xf	

	0	1	2	3	…	…	63
v0	0	1	2	3	…	…	63
v1							
v2							

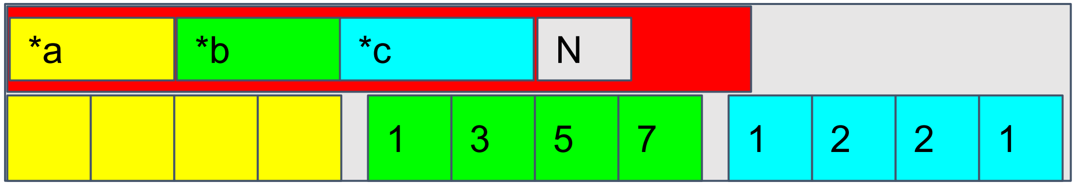
```
<_Z12vectoradd_PfPKfS1_ii>:
s_load_dword s0, s[4:5], 0x18
s_waitcnt lgkmcnt(0)
v_cmp_gt_i32_e32 vcc, s0, v0
s_and_saveexec_b64 s[0:1], vcc
s_cbranch_execz 14
s_load_dwordx4 s[0:3], s[4:5], 0x0
s_load_dwordx2 s[6:7], s[4:5], 0x10
v_lshlrev_b32_e32 v0, 2, v0
s_waitcnt lgkmcnt(0)
global_load_dword v1, v0, s[2:3]
global_load_dword v2, v0, s[6:7]
s_waitcnt vmcnt(0)
v_add_f32_e32 v1, v1, v2
global_store_dword v0, v1, s[0:1]
s_endpgm
```



s0	s1	s2	s3	s4	s5	s6	s7	vcc	exec
0xffffffffffffffff				*kernarg				0xf	0xf

	0	1	2	3	…	…	63
v0	0	1	2	3	…	…	63
v1							
v2							

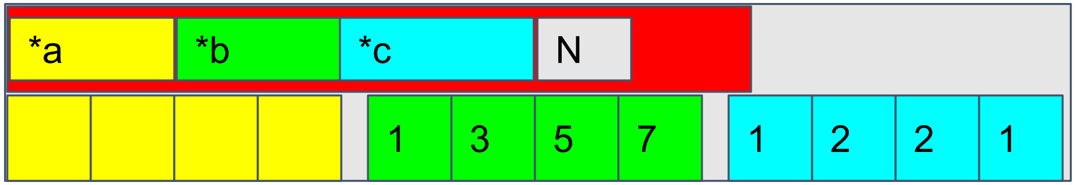
```
<_Z12vectoradd_PfPKfS1_ii>:
s_load_dword s0, s[4:5], 0x18
s_waitcnt lgkmcnt(0)
v_cmp_gt_i32_e32 vcc, s0, v0
s_and_saveexec_b64 s[0:1], vcc
s_cbranch_execz 14
s_load_dwordx4 s[0:3], s[4:5], 0x0
s_load_dwordx2 s[6:7], s[4:5], 0x10
v_lshlrev_b32_e32 v0, 2, v0
s_waitcnt lgkmcnt(0)
global_load_dword v1, v0, s[2:3]
global_load_dword v2, v0, s[6:7]
s_waitcnt vmcnt(0)
v_add_f32_e32 v1, v1, v2
global_store_dword v0, v1, s[0:1]
s_endpgm
```



s0	s1	s2	s3	s4	s5	s6	s7	vcc	exec
*a		*b		*kernarg				0xf	0xf

	0	1	2	3	…	…	63
v0	0	1	2	3	…	…	63
v1							
v2							

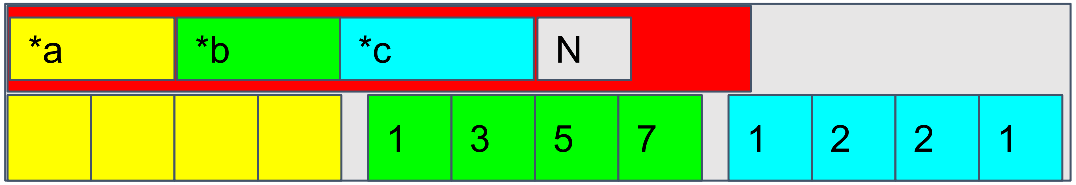
```
<_Z12vectoradd_PfPKfS1_ii>:
s_load_dword s0, s[4:5], 0x18
s_waitcnt lgkmcnt(0)
v_cmp_gt_i32_e32 vcc, s0, v0
s_and_saveexec_b64 s[0:1], vcc
s_cbranch_execz 14
s_load_dwordx4 s[0:3], s[4:5], 0x0
s_load_dwordx2 s[6:7], s[4:5], 0x10
v_lshlrev_b32_e32 v0, 2, v0
s_waitcnt lgkmcnt(0)
global_load_dword v1, v0, s[2:3]
global_load_dword v2, v0, s[6:7]
s_waitcnt vmcnt(0)
v_add_f32_e32 v1, v1, v2
global_store_dword v0, v1, s[0:1]
s_endpgm
```



s0	s1	s2	s3	s4	s5	s6	s7	vcc	exec
*a		*b		*kernarg		*c		0xf	0xf

	0	1	2	3	…	…	63
v0	0	1	2	3	…	…	63
v1							
v2							

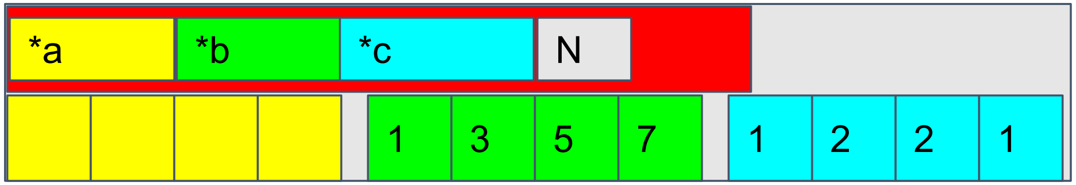
```
<_Z12vectoradd_PfPKfS1_ii>:
s_load_dword s0, s[4:5], 0x18
s_waitcnt lgkmcnt(0)
v_cmp_gt_i32_e32 vcc, s0, v0
s_and_saveexec_b64 s[0:1], vcc
s_cbranch_execz 14
s_load_dwordx4 s[0:3], s[4:5], 0x0
s_load_dwordx2 s[6:7], s[4:5], 0x10
v_lshlrev_b32_e32 v0, 2, v0
s_waitcnt lgkmcnt(0)
global_load_dword v1, v0, s[2:3]
global_load_dword v2, v0, s[6:7]
s_waitcnt vmcnt(0)
v_add_f32_e32 v1, v1, v2
global_store_dword v0, v1, s[0:1]
s_endpgm
```



s0	s1	s2	s3	s4	s5	s6	s7	vcc	exec
*a		*b		*kernarg		*c		0xf	0xf

	0	1	2	3	…	…	63
v0	0	4	8	12	…	…	63
v1							
v2							

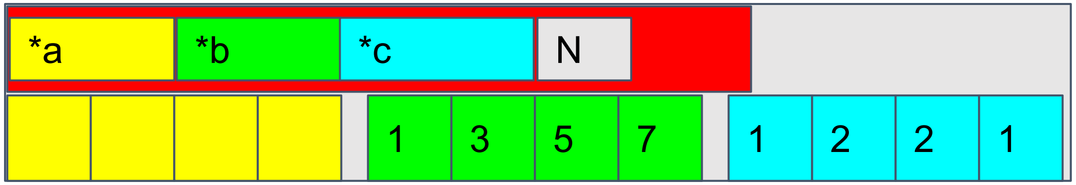
```
<_Z12vectoradd_PfPKfS1_ii>:
s_load_dword s0, s[4:5], 0x18
s_waitcnt lgkmcnt(0)
v_cmp_gt_i32_e32 vcc, s0, v0
s_and_saveexec_b64 s[0:1], vcc
s_cbranch_execz 14
s_load_dwordx4 s[0:3], s[4:5], 0x0
s_load_dwordx2 s[6:7], s[4:5], 0x10
v_lshlrev_b32_e32 v0, 2, v0
s_waitcnt lgkmcnt(0)
global_load_dword v1, v0, s[2:3]
global_load_dword v2, v0, s[6:7]
s_waitcnt vmcnt(0)
v_add_f32_e32 v1, v1, v2
global_store_dword v0, v1, s[0:1]
s_endpgm
```



s0	s1	s2	s3	s4	s5	s6	s7	vcc	exec
*a		*b		*kernarg		*c		0xf	0xf

	0	1	2	3	…	…	63
v0	0	4	8	12	…	…	63
v1	1	3	5	7			
v2							

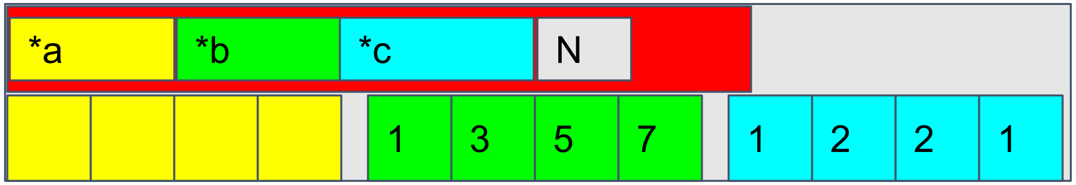
```
<_Z12vectoradd_PfPKfS1_ii>:
s_load_dword s0, s[4:5], 0x18
s_waitcnt lgkmcnt(0)
v_cmp_gt_i32_e32 vcc, s0, v0
s_and_saveexec_b64 s[0:1], vcc
s_cbranch_execz 14
s_load_dwordx4 s[0:3], s[4:5], 0x0
s_load_dwordx2 s[6:7], s[4:5], 0x10
v_lshlrev_b32_e32 v0, 2, v0
s_waitcnt lgkmcnt(0)
global_load_dword v1, v0, s[2:3]
global_load_dword v2, v0, s[6:7]
s_waitcnt vmcnt(0)
v_add_f32_e32 v1, v1, v2
global_store_dword v0, v1, s[0:1]
s_endpgm
```



s0	s1	s2	s3	s4	s5	s6	s7	vcc	exec
*a		*b		*kernarg		*c		0xf	0xf

	0	1	2	3	…	…	63
v0	0	4	8	12	…	…	63
v1	1	3	5	7			
v2	1	2	2	1			

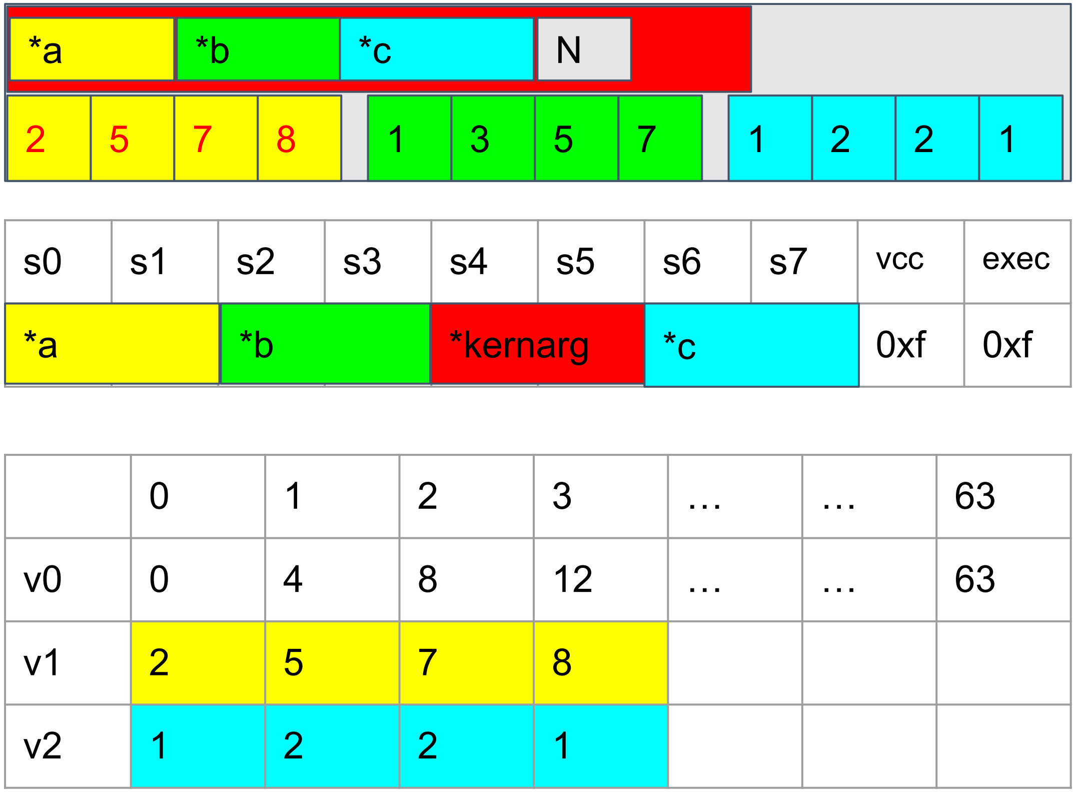
```
<_Z12vectoradd_PfPKfS1_ii>:
s_load_dword s0, s[4:5], 0x18
s_waitcnt lgkmcnt(0)
v_cmp_gt_i32_e32 vcc, s0, v0
s_and_saveexec_b64 s[0:1], vcc
s_cbranch_execz 14
s_load_dwordx4 s[0:3], s[4:5], 0x0
s_load_dwordx2 s[6:7], s[4:5], 0x10
v_lshlrev_b32_e32 v0, 2, v0
s_waitcnt lgkmcnt(0)
global_load_dword v1, v0, s[2:3]
global_load_dword v2, v0, s[6:7]
s_waitcnt vmcnt(0)
v_add_f32_e32 v1, v1, v2
global_store_dword v0, v1, s[0:1]
s_endpgm
```



s0	s1	s2	s3	s4	s5	s6	s7	vcc	exec
*a		*b		*kernarg		*c		0xf	0xf

	0	1	2	3	…	…	63
v0	0	4	8	12	…	…	63
v1	2	5	7	8			
v2	1	2	2	1			

```
<_Z12vectoradd_PfPKfS1_ii>:
s_load_dword s0, s[4:5], 0x18
s_waitcnt lgkmcnt(0)
v_cmp_gt_i32_e32 vcc, s0, v0
s_and_saveexec_b64 s[0:1], vcc
s_cbranch_execz 14
s_load_dwordx4 s[0:3], s[4:5], 0x0
s_load_dwordx2 s[6:7], s[4:5], 0x10
v_lshlrev_b32_e32 v0, 2, v0
s_waitcnt lgkmcnt(0)
global_load_dword v1, v0, s[2:3]
global_load_dword v2, v0, s[6:7]
s_waitcnt vmcnt(0)
v_add_f32_e32 v1, v1, v2
global_store_dword v0, v1, s[0:1]
s_endpgm
```



*a	*b	*c	N

2	5	7	8	1	3	5	7	1	2	2	1

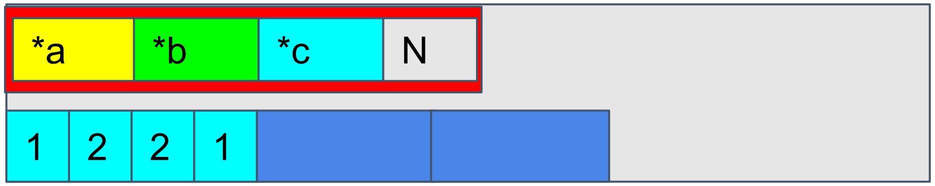
s0	s1	s2	s3	s4	s5	s6	s7	vcc	exec
*a		*b		*kernarg		*c		0xf	0xf

	0	1	2	3	…	…	63
v0	0	4	8	12	…	…	63
v1	2	5	7	8			
v2	1	2	2	1			

Design of GPU Instrumentation

Example Instrumentation: Vector Add

```
void vectoradd(float* a,
    float* b,  float* c, int N)
{
long long int * record = &(C[N]);
start_time = clock64();
    int i = hipThreadIdx_x;
    if(i < N)
        a[i] = b[i] + c[i];
end_time = clock64();
record[ 0 ] = start_time;
record[ 1 ] = end_time;
}
```

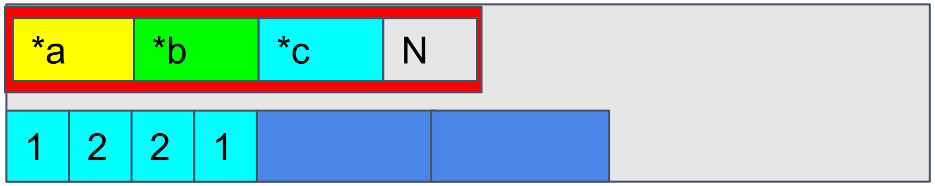


s4	s5	s8	s9	s10	s11	s12	s13	exec
*kernarg								

	0	1	2	3
v8				
v9				
v10				
v11				

Example Instrumentation: Prologue

```
s_load_dwordx2 s[10:11], s[4:5], 0x10
s_mov_b32 s8, 16
s_add_u32 s8, s10 , s8
s_addc_u32 s9 , s11 , 0
s_memtime s[10:11]
s_waitcnt lgkmcnt(0)
```

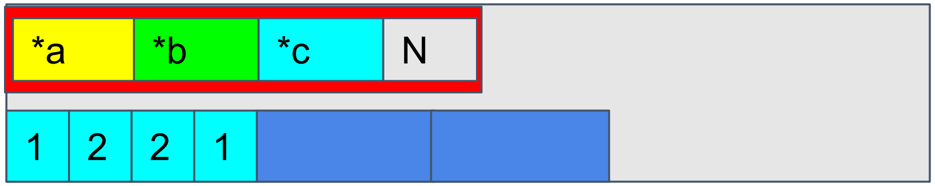


s4	s5	s8	s9	s10	s11	s12	s13	exec
*kernarg								

	0	1	2	3
v8				
v9				
v10				
v11				

Example Instrumentation: Prologue

```
s_load_dwordx2 s[10:11], s[4:5], 0x10
s_mov_b32 s8, 16
s_add_u32 s8, s10 , s8
s_addc_u32 s9 , s11 , 0
s_memtime s[10:11]
s_waitcnt lgkmcnt(0)
```

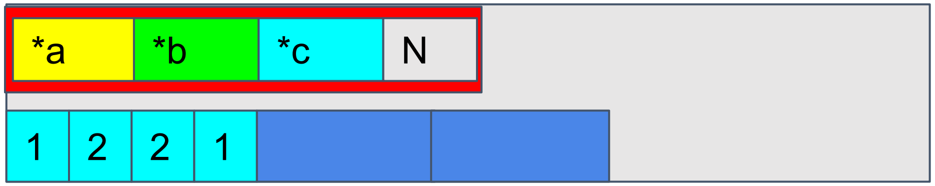


s4	s5	s8	s9	s10	s11	s12	s13	exec
*kernarg				*c				

	0	1	2	3
v8				
v9				
v10				
v11				

Example Instrumentation: Prologue

```
s_load_dwordx2 s[10:11], s[4:5], 0x10
s_mov_b32 s8, 16
s_add_u32 s8, s10 , s8
s_addc_u32 s9 , s11 , 0
s_memtime s[10:11]
s_waitcnt lgkmcnt(0)
```

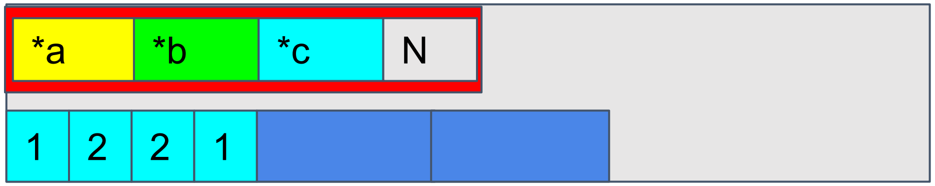


s4	s5	s8	s9	s10	s11	s12	s13	exec
*kernarg		*c + 16		*c				

	0	1	2	3
v8				
v9				
v10				
v11				

Example Instrumentation: Prologue

```
s_load_dwordx2 s[10:11], s[4:5], 0x10
s_mov_b32 s8, 16
s_add_u32 s8, s10 , s8
s_addc_u32 s9 , s11 , 0
s_memtime s[10:11]
s_waitcnt lgkmcnt(0)
```

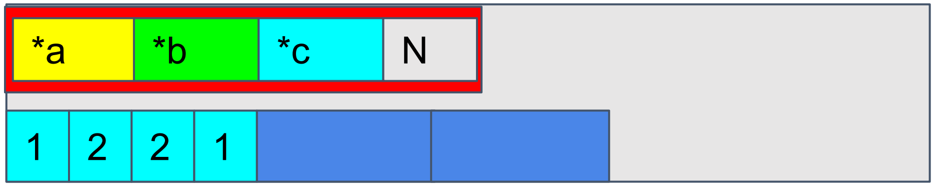


s4	s5	s8	s9	s10	s11	s12	s13	exec
*kernarg		*c + 16		start_time				

	0	1	2	3
v8				
v9				
v10				
v11				

Example Instrumentation: Epilogue

```
s_mov_b64 exec, 1
s_memtime s[12:13]
s_waitcnt lgkmcnt(0)
v_mov_b32 v8 , s8
v_mov_b32 v9 , s9
v_mov_b32 v10, s10
v_mov_b32 v11, s11
global_store_dword v[8:9], v[10:11]
v_mov_b32 v10, s12
v_mov_b32 v11, s13
v_add_co_u32_e32 v8,vcc,8,v8
v_addc_co_u32_e32 v9,vcc,0,v9
global_store_dword v[8:9], v[10:11]
```

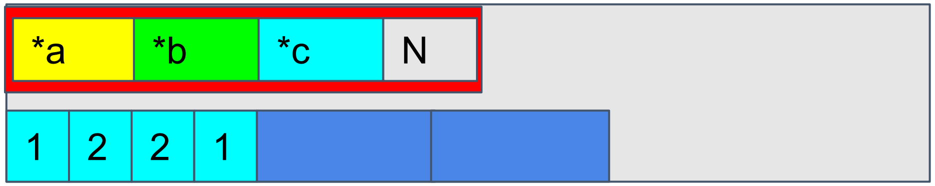


s4	s5	s8	s9	s10	s11	s12	s13	exec
*kernarg		*c + 16		start_time				1

	0	1	2	3
v8				
v9				
v10				
v11				

Example Instrumentation: Epilogue

```
s_mov_b64 exec, 1
s_memtime s[12:13]
s_waitcnt lgkmcnt(0)
v_mov_b32 v8 , s8
v_mov_b32 v9 , s9
v_mov_b32 v10, s10
v_mov_b32 v11, s11
global_store_dword v[8:9], v[10:11]
v_mov_b32 v10, s12
v_mov_b32 v11, s13
v_add_co_u32_e32 v8,vcc,8,v8
v_addc_co_u32_e32 v9,vcc,0,v9
global_store_dword v[8:9], v[10:11]
```

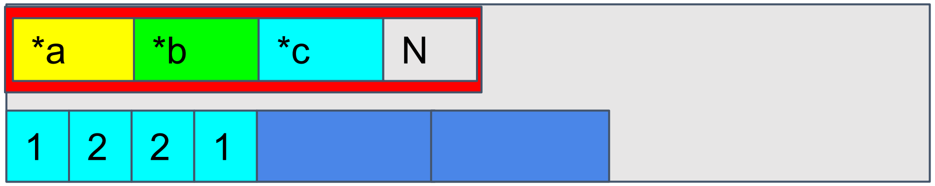


s4	s5	s8	s9	s10	s11	s12	s13	exec
*kernarg		*c + 16		start_time		end_time		1

	0	1	2	3
v8				
v9				
v10				
v11				

Example Instrumentation: Epilogue

```
s_mov_b64 exec, 1
s_memtime s[12:13]
s_waitcnt lgkmcnt(0)
v_mov_b32 v8 , s8
v_mov_b32 v9 , s9
v_mov_b32 v10, s10
v_mov_b32 v11, s11
global_store_dword v[8:9], v[10:11]
v_mov_b32 v10, s12
v_mov_b32 v11, s13
v_add_co_u32_e32 v8,vcc,8,v8
v_addc_co_u32_e32 v9,vcc,0,v9
global_store_dword v[8:9], v[10:11]
```

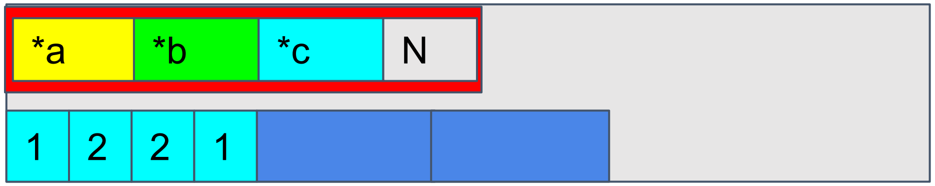


s4	s5	s8	s9	s10	s11	s12	s13	exec
*kernarg		*c + 16		start_time		end_time		1

	0	1	2	3
v8	*c + 16			
v9				
v10				
v11				

Example Instrumentation: Epilogue

```
s_mov_b64 exec, 1
s_memtime s[12:13]
s_waitcnt lgkmcnt(0)
v_mov_b32 v8 , s8
v_mov_b32 v9 , s9
v_mov_b32 v10, s10
v_mov_b32 v11, s11
global_store_dword v[8:9], v[10:11]
v_mov_b32 v10, s12
v_mov_b32 v11, s13
v_add_co_u32_e32 v8,vcc,8,v8
v_addc_co_u32_e32 v9,vcc,0,v9
global_store_dword v[8:9], v[10:11]
```



s4	s5	s8	s9	s10	s11	s12	s13	exec
*kernarg		*c + 16		start_time		end_time		1

	0	1	2	3
v8	*c + 16			
v9				
v10	start_time			
v11				

Example Instrumentation: Epilogue

```
s_mov_b64 exec, 1
s_memtime s[12:13]
s_waitcnt lgkmcnt(0)
v_mov_b32 v8 , s8
v_mov_b32 v9 , s9
v_mov_b32 v10, s10
v_mov_b32 v11, s11
global_store_dword v[8:9], v[10:11]
v_mov_b32 v10, s12
v_mov_b32 v11, s13
v_add_co_u32_e32 v8,vcc,8,v8
v_addc_co_u32_e32 v9,vcc,0,v9
global_store_dword v[8:9], v[10:11]
```

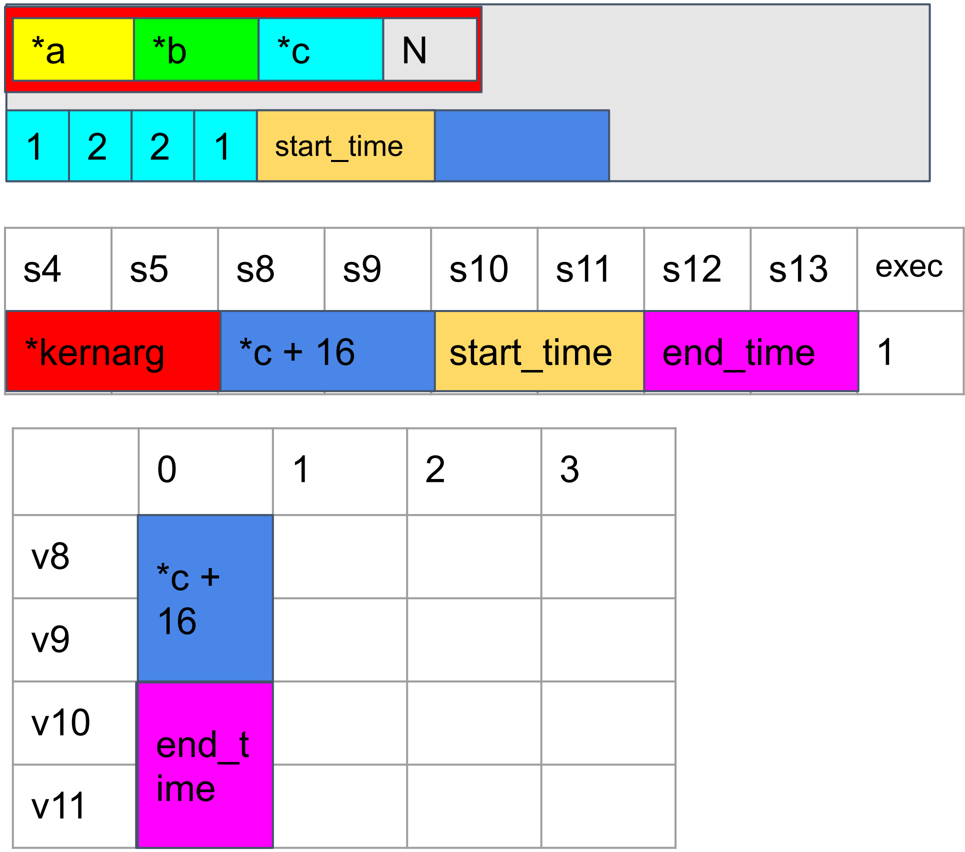
*a	*b	*c	N		
1	2	2	1	start_time	

s4	s5	s8	s9	s10	s11	s12	s13	exec
*kernarg		*c + 16		start_time		end_time		1

	0	1	2	3
v8	*c + 16			
v9				
v10	start_ time			
v11				

Example Instrumentation: Epilogue

```
s_mov_b64 exec, 1
s_memtime s[12:13]
s_waitcnt lgkmcnt(0)
v_mov_b32 v8 , s8
v_mov_b32 v9 , s9
v_mov_b32 v10, s10
v_mov_b32 v11, s11
global_store_dword v[8:9], v[10:11]
v_mov_b32 v10, s12
v_mov_b32 v11, s13
v_add_co_u32_e32 v8,vcc,8,v8
v_addc_co_u32_e32 v9,vcc,0,v9
global_store_dword v[8:9], v[10:11]
```



*a	*b	*c	N

1	2	2	1	start_time	

s4	s5	s8	s9	s10	s11	s12	s13	exec
*kernarg		*c + 16		start_time		end_time		1

	0	1	2	3
v8	*c + 16			
v9				
v10	end_time			
v11				

Example Instrumentation: Epilogue

```
s_mov_b64 exec, 1
s_memtime s[12:13]
s_waitcnt lgkmcnt(0)
v_mov_b32 v8 , s8
v_mov_b32 v9 , s9
v_mov_b32 v10, s10
v_mov_b32 v11, s11
global_store_dword v[8:9], v[10:11]
v_mov_b32 v10, s12
v_mov_b32 v11, s13
v_add_co_u32_e32 v8,vcc,8,v8
v_addc_co_u32_e32 v9,vcc,0,v9
global_store_dword v[8:9], v[10:11]
```

*a	*b	*c	N

1	2	2	1	start_time	

s4	s5	s8	s9	s10	s11	s12	s13	exec
*kernarg		*c + 16		start_time		end_time		1

	0	1	2	3
v8	*c + 24			
v9				
v10	end_time			
v11				

Example Instrumentation: Epilogue

```
s_mov_b64 exec, 1
s_memtime s[12:13]
s_waitcnt lgkmcnt(0)
v_mov_b32 v8 , s8
v_mov_b32 v9 , s9
v_mov_b32 v10, s10
v_mov_b32 v11, s11
global_store_dword v[8:9], v[10:11]
v_mov_b32 v10, s12
v_mov_b32 v11, s13
v_add_co_u32_e32 v8,vcc,8,v8
v_addc_co_u32_e32 v9,vcc,0,v9
global_store_dword v[8:9], v[10:11]
```

*a	*b	*c	N		
1	2	2	1	start_time	end_time

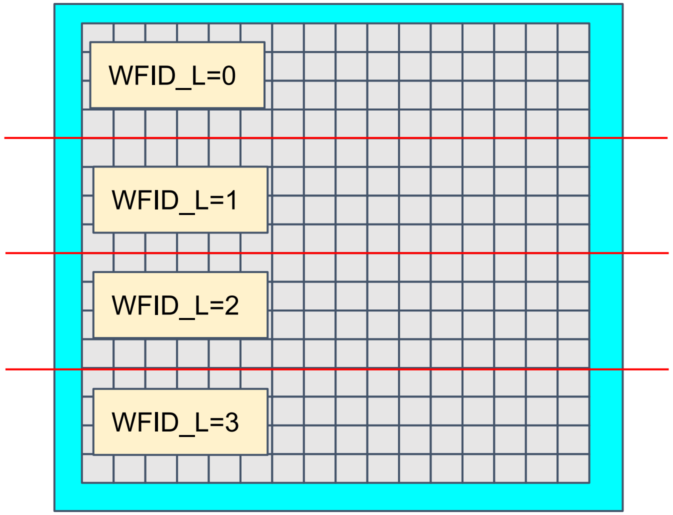
s4	s5	s8	s9	s10	s11	s12	s13	exec
*kernarg		*c + 16		start_time		end_time		1

	0	1	2	3
v8	*c + 24			
v9				
v10	end_time			
v11				

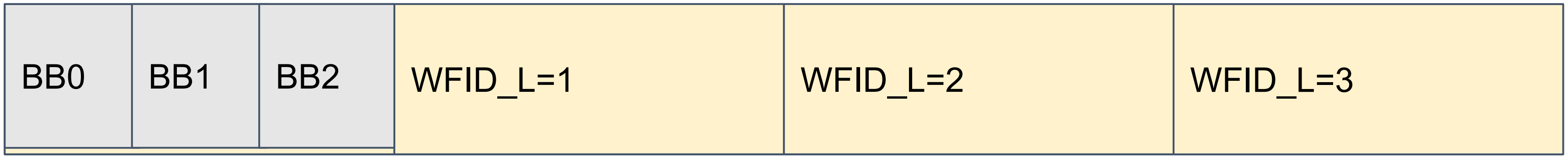
Complication - Aggregation of Data

```
PROLOGUE:
TMPVR[4:5] = WFID_L
```

```
Basic Block Entrance :
TMPR[0:1] = READ_TIMESTAMP()
Original Code
TMPR[2:3] = READ_TIMESTAMP()
TMPR[2:3] = TMPR[2:3] - TMPR[0:1]
LDS[WFID_L*DATASIZE_PER_WF+BB_ID*
    DATA_SIZE_PER_REOCRD] += TMPR[2:3]
Exit Instruction
```



LDS



Implementation - Branch Divergence Counters

```
s_and_saveexec_b64 s[16:17], s[0:1]
```

```
s_cbranch_execz BB0_3
```

Implementation - Branch Divergence Counters

Back up the execution mask,
Change it to 1

```
s_and_saveexec_b64 s[16:17], s[0:1]
s_mov_b64 s[54:55], exec
s_mov_b64 exec, 1
```

```
s_cbranch_execz BB0_3
```

Implementation - Branch Divergence Counters

Address Computation for Accessing Shared Memory

```
s_and_saveexec_b64 s[16:17], s[0:1]
s_mov_b64 s[54:55], exec
s_mov_b64 exec, 1
s_mul_i32 s43, 5, s46
s_add_u32 s43, 2, s43
s_lshl_b32 s43, s43, 3
s_add_u32 s43, 0x1c20, s43
v_mov_b32_e32 v55, s43
```

```
s_cbranch_execz BB0_3
```

Implementation - Branch Divergence Counters

If exec mask stays the same, scc = 1 add that value to LDS pointed by v55

```
s_and_saveexec_b64 s[16:17], s[0:1]
s_mov_b64 s[54:55], exec
s_mov_b64 exec, 1
s_mul_i32 s43, 5, s46
s_add_u32 s43, 2, s43
s_lshl_b32 s43, s43, 3
s_add_u32 s43, 0x1c20, s43
v_mov_b32_e32 v55, s43
s_cmp_eq_u64 s[16:17], s[54:55]
v_mov_b32_e32 v56, src_scc
ds_add_u32 v55, v56
```

```
s_cbranch_execz BB0_3
```

Implementation - Branch Divergence Counters

```
s_and_saveexec_b64 s[16:17], s[0:1]
s_mov_b64 s[54:55], exec
s_mov_b64 exec, 1
s_mul_i32 s43, 5, s46
s_add_u32 s43, 2, s43
s_lshl_b32 s43, s43, 3
s_add_u32 s43, 0x1c20, s43
v_mov_b32_e32 v55, s43
s_cmp_eq_u64 s[16:17], s[54:55]
v_mov_b32_e32 v56, src_scc
ds_add_u32 v55, v56
s_cmp_eq_u64 0, s[54:55]
v_mov_b32_e32 v56, src_scc
ds_add_u32 v55, v56
```

if exec becomes 0, then all threads must agree on the else branch, we also increase the agree count

```
s_cbranch_execz BB0_3
```

Implementation - Branch Divergence Counters

```
s_and_saveexec_b64 s[16:17], s[0:1]
s_mov_b64 s[54:55], exec
s_mov_b64 exec, 1
s_mul_i32 s43, 5, s46
s_add_u32 s43, 2, s43
s_lshl_b32 s43, s43, 3
s_add_u32 s43, 0x1c20, s43
v_mov_b32_e32 v55, s43
s_cmp_eq_u64 s[16:17], s[54:55]
v_mov_b32_e32 v56, src_scc
ds_add_u32 v55, v56
s_cmp_eq_u64 0, s[54:55]
v_mov_b32_e32 v56, src_scc
ds_add_u32 v55, v56
v_add_u32_e32 v55, 4, v55
ds_inc_u32 v55, v58

s_cbranch_execz BB0_3
```

increment the lds address to point to ex ec count, and increment it by 1

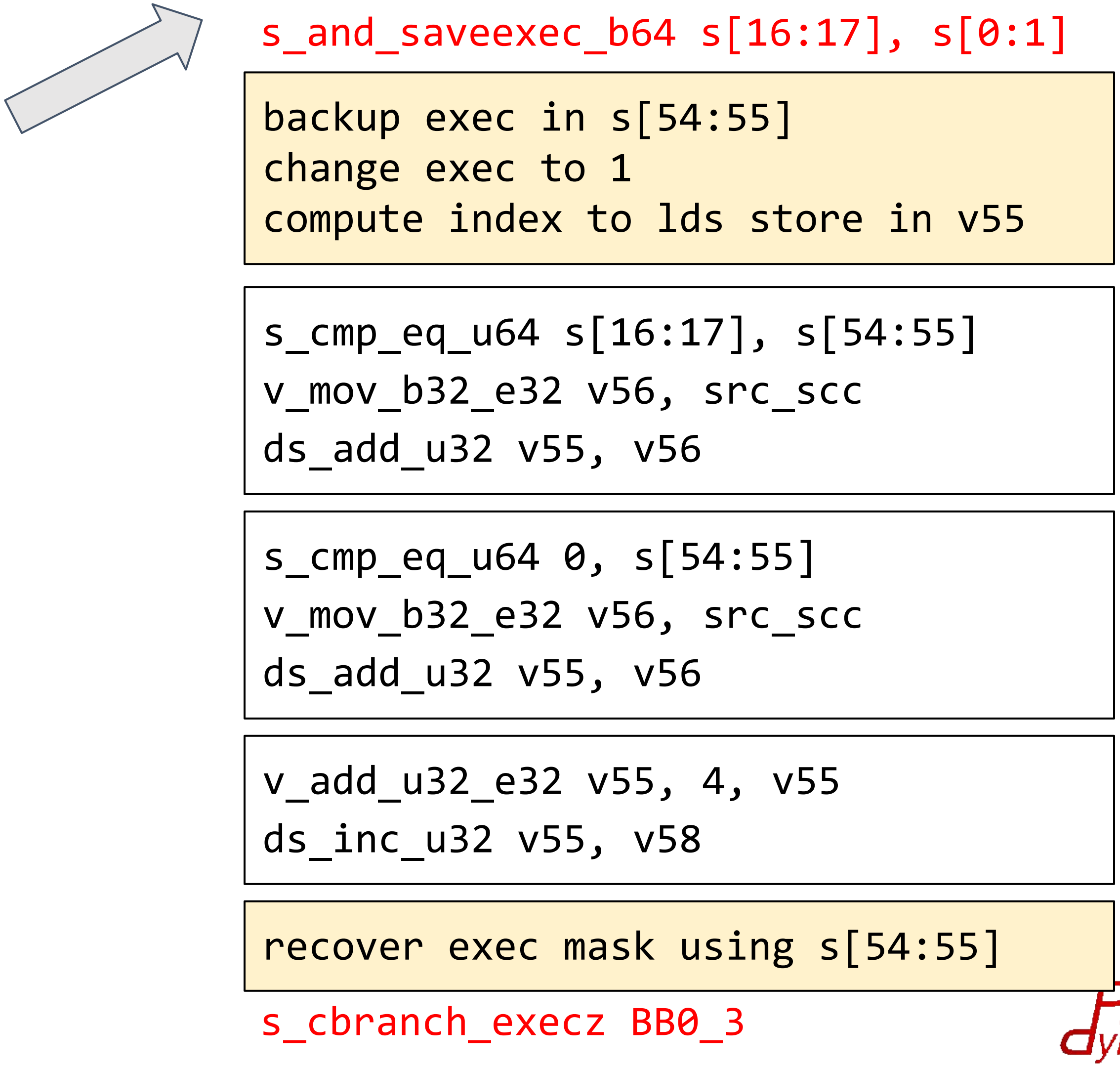
Implementation - Branch Divergence Counters

```
s_and_saveexec_b64 s[16:17], s[0:1]
s_mov_b64 s[54:55], exec
s_mov_b64 exec, 1
s_mul_i32 s43, 5, s46
s_add_u32 s43, 2, s43
s_lshl_b32 s43, s43, 3
s_add_u32 s43, 0x1c20, s43
v_mov_b32_e32 v55, s43
s_cmp_eq_u64 s[16:17], s[54:55]
v_mov_b32_e32 v56, src_scc
ds_add_u32 v55, v56
s_cmp_eq_u64 0, s[54:55]
v_mov_b32_e32 v56, src_scc
ds_add_u32 v55, v56
v_add_u32_e32 v55, 4, v55
ds_inc_u32 v55, v58
s_mov_b64 exec, s[54:55]
s_cbranch_execz BB0_3
```

recover exec mask

Implementation - Branch Divergence Counters

After this instruction, EXEC is changed to IF MASK



```
s_and_saveexec_b64 s[16:17], s[0:1]
```

```
backup exec in s[54:55]
change exec to 1
compute index to lds store in v55
```

```
s_cmp_eq_u64 s[16:17], s[54:55]
v_mov_b32_e32 v56, src_scc
ds_add_u32 v55, v56
```

```
s_cmp_eq_u64 0, s[54:55]
v_mov_b32_e32 v56, src_scc
ds_add_u32 v55, v56
```

```
v_add_u32_e32 v55, 4, v55
ds_inc_u32 v55, v58
```

```
recover exec mask using s[54:55]
```

```
s_cbranch_execz BB0_3
```

Implementation - Branch Divergence Counters

backup exec mask
compute index to
shared memory

```
s_and_saveexec_b64 s[16:17], s[0:1]
```

```
backup exec in s[54:55]
change exec to 1
compute index to lds store in v55
```

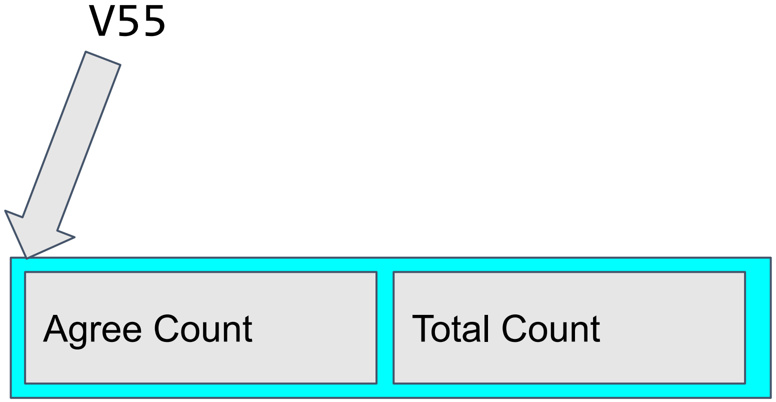
```
s_cmp_eq_u64 s[16:17], s[54:55]
v_mov_b32_e32 v56, src_scc
ds_add_u32 v55, v56
```

```
s_cmp_eq_u64 0, s[54:55]
v_mov_b32_e32 v56, src_scc
ds_add_u32 v55, v56
```

```
v_add_u32_e32 v55, 4, v55
ds_inc_u32 v55, v58
```

```
recover exec mask using s[54:55]
```

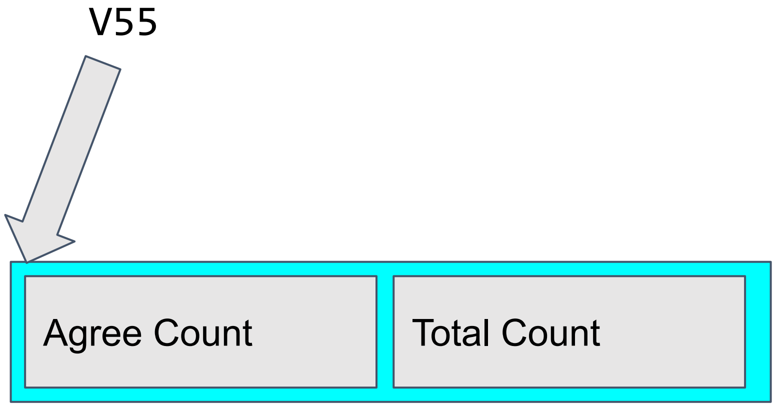
```
s_cbranch_execz BB0_3
```



Implementation - Branch Divergence Counters

Compare IF MASK with EXEC
V56 = (IF MASK == EXEC) ? 1 : 0
Aggregate result in Agree Count

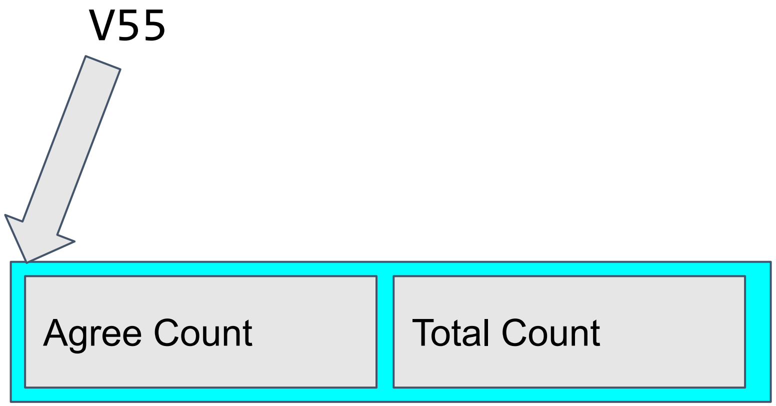
```
s_and_saveexec_b64 s[16:17], s[0:1]

backup exec in s[54:55]
change exec to 1
compute index to lds store in v55

s_cmp_eq_u64 s[16:17], s[54:55]
v_mov_b32_e32 v56, src_scc
ds_add_u32 v55, v56

s_cmp_eq_u64 0, s[54:55]
v_mov_b32_e32 v56, src_scc
ds_add_u32 v55, v56

v_add_u32_e32 v55, 4, v55
ds_inc_u32 v55, v58

recover exec mask using s[54:55]

s_cbranch_execz BB0_3
```



Implementation - Branch Divergence Counters

```
s_and_saveexec_b64 s[16:17], s[0:1]

backup exec in s[54:55]
change exec to 1
compute index to lds store in v55

s_cmp_eq_u64 s[16:17], s[54:55]
v_mov_b32_e32 v56, src_scc
ds_add_u32 v55, v56

s_cmp_eq_u64 0, s[54:55]
v_mov_b32_e32 v56, src_scc
ds_add_u32 v55, v56

v_add_u32_e32 v55, 4, v55
ds_inc_u32 v55, v58

recover exec mask using s[54:55]
s_cbranch_execz BB0_3
```

Check if IF MASK is 0
V56 = (IF MASK == 0) ? 1 : 0
Aggregate result in Agree Count



Implementation - Branch Divergence Counters

```
s_and_saveexec_b64 s[16:17], s[0:1]
```

```
backup exec in s[54:55]
change exec to 1
compute index to lds store in v55
```

```
s_cmp_eq_u64 s[16:17], s[54:55]
v_mov_b32_e32 v56, src_scc
ds_add_u32 v55, v56
```

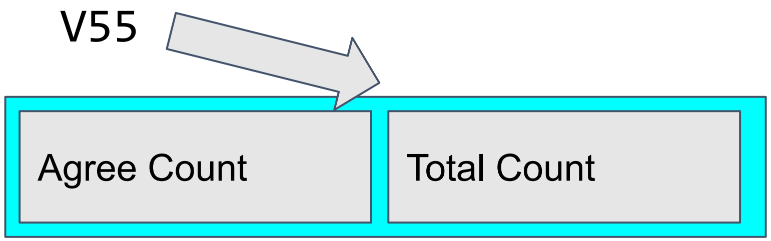
```
s_cmp_eq_u64 0, s[54:55]
v_mov_b32_e32 v56, src_scc
ds_add_u32 v55, v56
```

Move V55 to point to Total Count
Increment Total Count

```
v_add_u32_e32 v55, 4, v55
ds_inc_u32 v55, v58
```

```
recover exec mask using s[54:55]
```

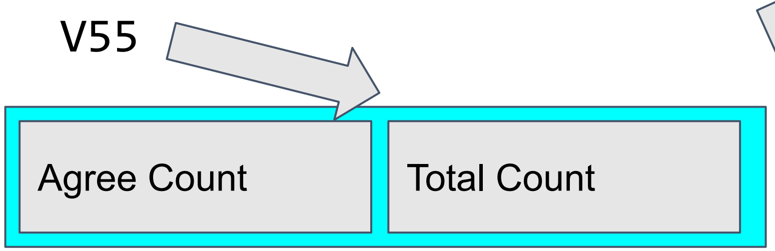
```
s_cbranch_execz BB0_3
```



Implementation - Branch Divergence Counters

```
s_and_saveexec_b64 s[16:17], s[0:1]

backup exec in s[54:55]
change exec to 1
compute index to lds store in v55

s_cmp_eq_u64 s[16:17], s[54:55]
v_mov_b32_e32 v56, src_scc
ds_add_u32 v55, v56

s_cmp_eq_u64 0, s[54:55]
v_mov_b32_e32 v56, src_scc
ds_add_u32 v55, v56

v_add_u32_e32 v55, 4, v55
ds_inc_u32 v55, v58

recover exec mask using s[54:55]

s_cbranch_execz BB0_3
```

recover exec mask before resuming execution


V55
Agree Count
Total Count