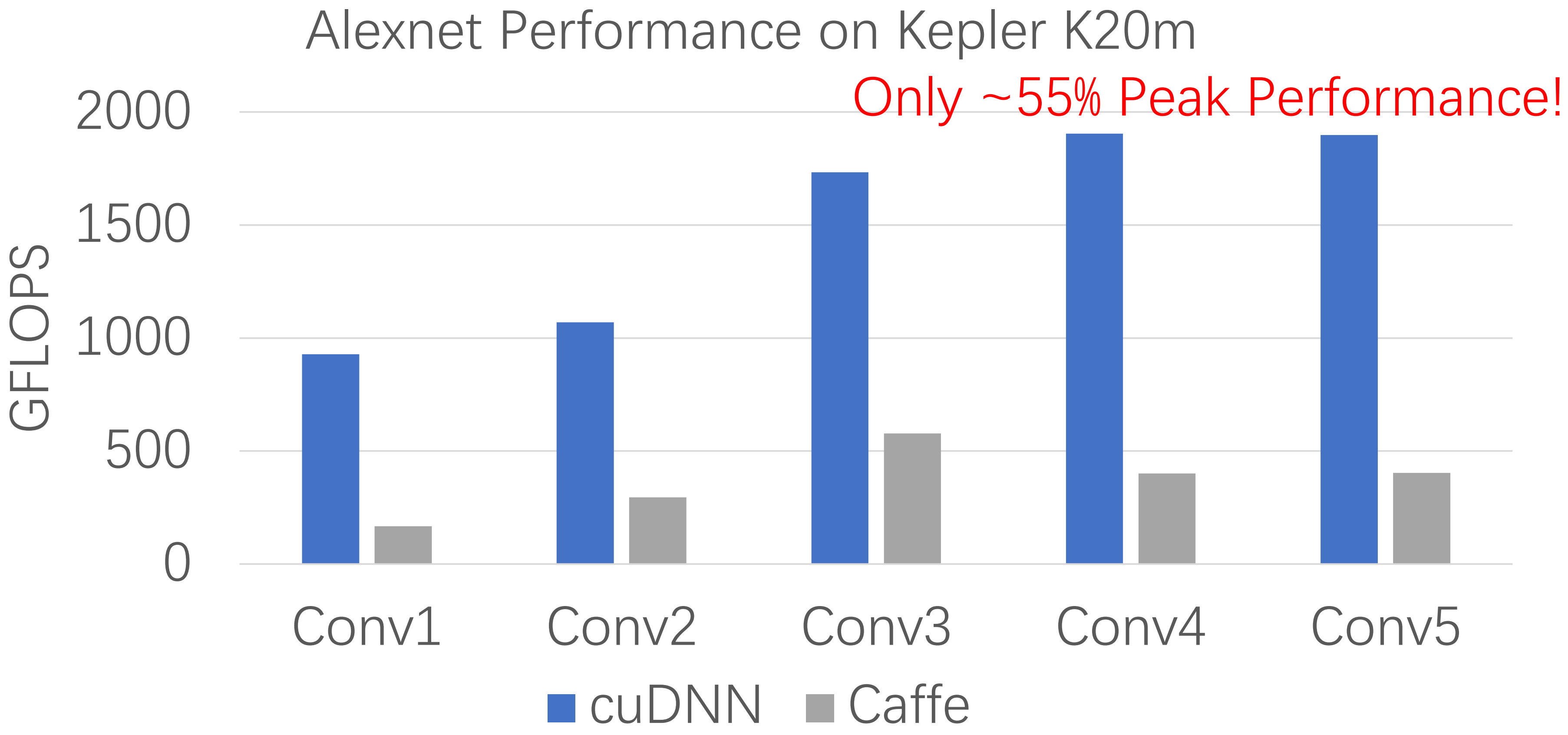
A Performance Analysis Framework for Exploiting GPU Microarchitectural Capability

Keren Zhou, Guangming Tan, Xiuxia Zhang, Chaowei Wang, Ninghui Sun

Institute of Computing Technology, Chinese Academy of Sciences

Performance analysis on GPU is critical

- Could the performance be higher?
- **Yes**, but we need a tool



Only ~55% Peak Performance!

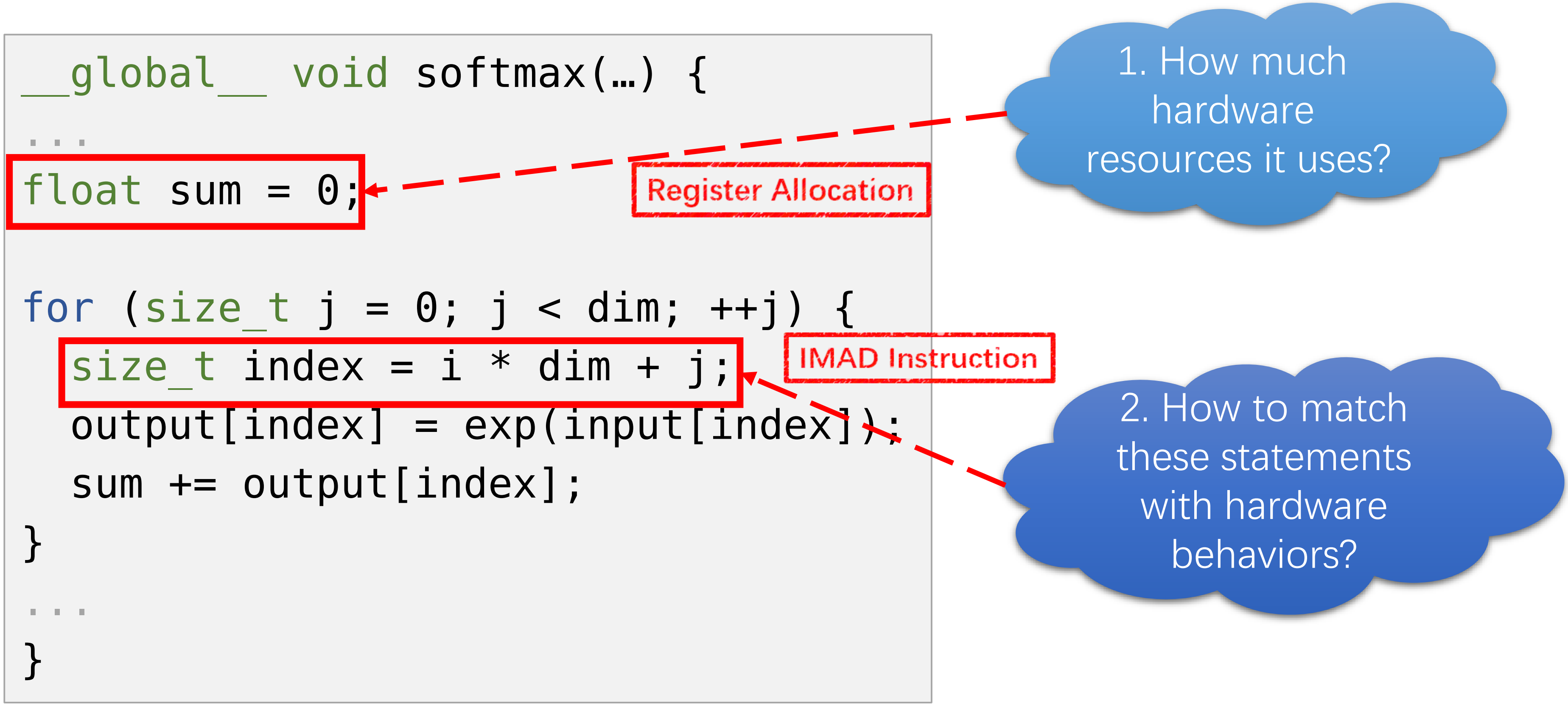
This paper builds a GPU performance model based on assembly instructions.

~~Not on Cuda C language~~

~~Not on PTX codes~~

Cuda is hard to parse

- Compiler optimizations on instructions and registers



```
__global__ void softmax(…) {
...
float sum = 0;

for (size_t j = 0; j < dim; ++j) {
  size_t index = i * dim + j;
  output[index] = exp(input[index]);
  sum += output[index];
}
...
}
```

Register Allocation

IMAD Instruction

1. How much hardware resources it uses?

2. How to match these statements with hardware behaviors?

Cuda and PTX map to many instructions

- Same function in different languages

```
__global__ void kernel(float a, float b, float
*c) {
  *c = a / (exp(a) * b);
}
```

Cuda: 1 line

```
ld.param.f32  %f1, [_Z4testffPf_param_0];
ld.param.f32  %f2, [_Z4testffPf_param_1];
ld.param.u64  %rd1, [_Z4testffPf_param_2];
cvta.to.global.u64  %rd2, %rd1;
mul.ftz.f32  %f3, %f1, 0f3FB8AA3B;
ex2.approx.ftz.f32  %f4, %f3;
mul.ftz.f32  %f5, %f4, %f2;
div.approx.ftz.f32  %f6, %f1, %f5;
st.global.f32  [%rd2], %f6;
ret;
```

PTX: 10 lines

```
MOV  R1, c[0x0][0x44];
MOV32I  R0, 0x3fb8aa3b;
MOV  R2, c[0x0][0x148];
FMUL.FTZ  R0, R0, c[0x0][0x140];
MOV  R3, c[0x0][0x14c];
RRO.EX2  R0, R0;
MUFU.EX2  R0, R0;
FMUL.FTZ  R0, R0, c[0x0][0x144];
MUFU.RCP  R0, R0;
FMUL.FTZ  R0, R0, c[0x0][0x140];
ST.E  [R2], R0;
EXIT;
BRA  0x70;
NOP;
```

Assembly: 14 lines

Cuda and PTX do not have control codes

- cuobjdump --dump-sass

```
FFMA R9, R68, R72, R9;          /* 0xcc002404241d1026 */
FFMA R13, R68, R73, R13;        /* 0xcc003404249d1036 */
FFMA R8, R69, R73, R8;          /* 0xcc002004249d1422 */
FFMA R25, R70, R72, R25;        /* 0xcc006404241d1866 */
FFMA R12, R69, R72, R12;        /* 0xcc003004241d1432 */
FFMA R29, R70, R73, R29;        /* 0xcc007404249d1876 */
LDS.64 R29,[R116+0x300];        /* 0x7a680001801dd172 */
```

Encoded Control Codes

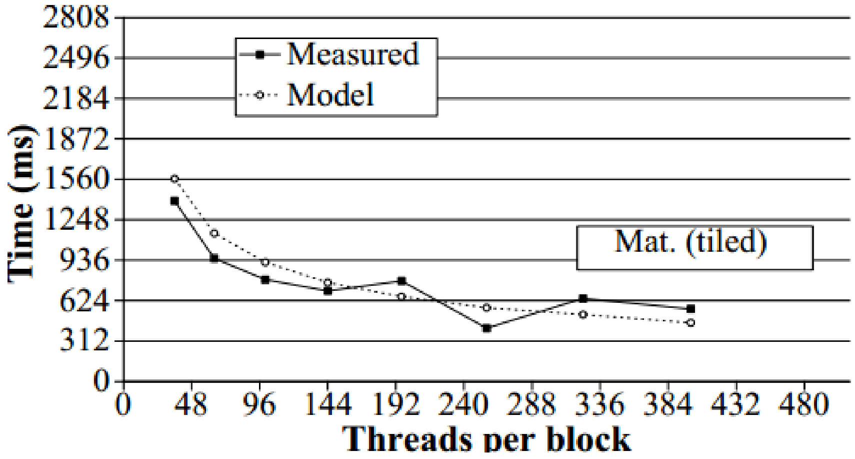
- KeplerAs --extract

Interpreted Instructions

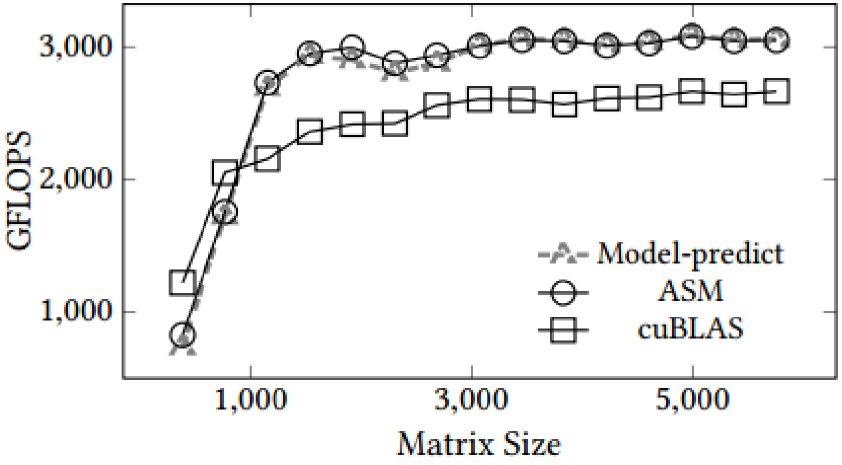
```
-:-:D:-:05   FFMA R9, R68, R72, R9;
-:-:D:-:04   FFMA R13, R68, R73, R13;
-:-:D:-:05   FFMA R8, R69, R73, R8;
-:-:D:-:04   FFMA R25, R70, R72, R25;
-:-:D:-:05   FFMA R12, R69, R72, R12;
-:-:D:-:04   FFMA R29, R70, R73, R29;
-:G:D:-:01   LDS.64 R29,[R116+0x300];
```

Cuda and PTX are inaccurate

- 5%-15% prediction errors by Cuda *[Hong2009An]*
- 2%-3% prediction errors by assembly instructions



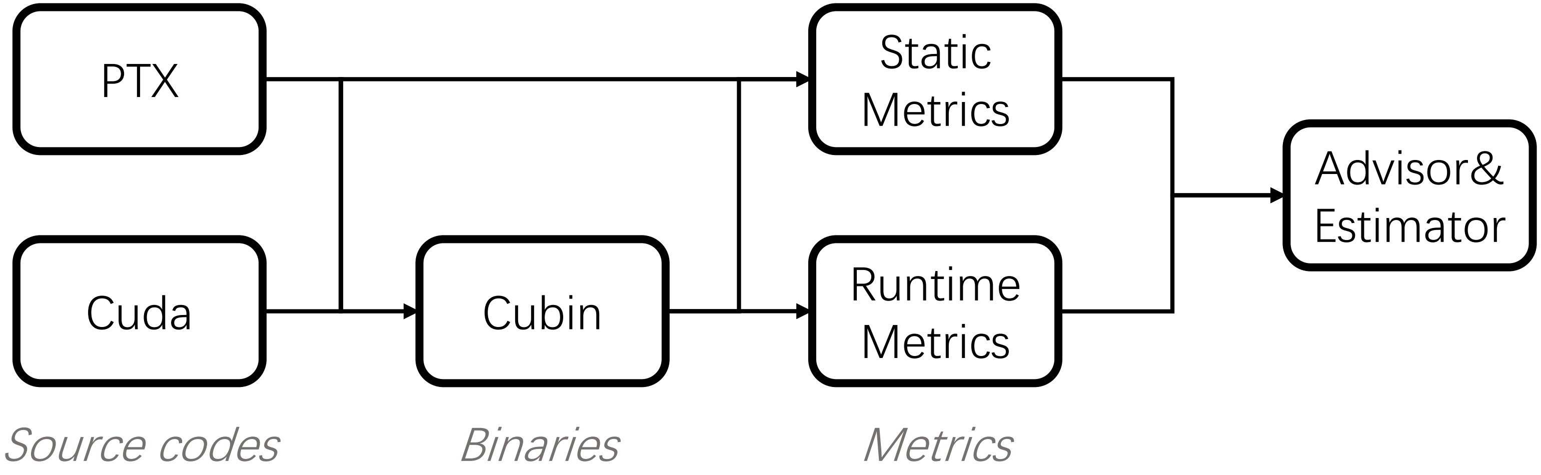
Previous results



Our results

Performance Analysis Approach

- Shorten analysis steps to make it more accurate

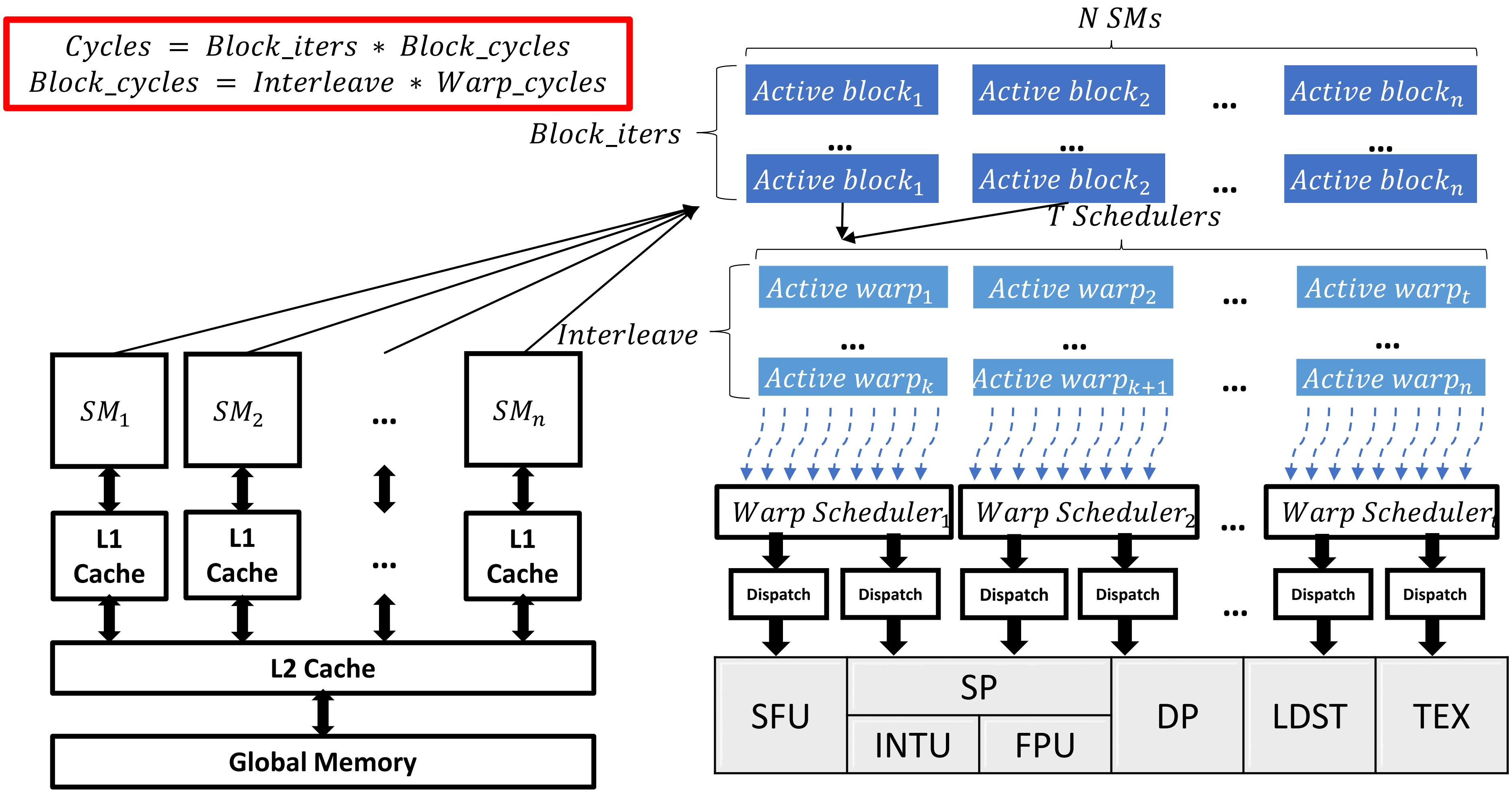


Contributions

- A performance analysis framework that benchmarks instruction characteristics, estimates running cycles, and points bottlenecks.
- Optimize two DNN routines—convolution and GEMM by eliminating bottlenecks.
- Compare with cuDNN and cuBLAS, showing 40% and 20% speedup respectively.

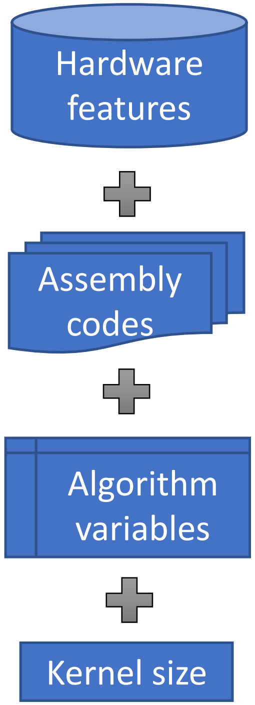
GPU Architectures

$$Cycles = Block\_iters * Block\_cycles$$
$$Block\_cycles = Interleave * Warp\_cycles$$



Analysis Framework

Input



Hardware features

Assembly codes

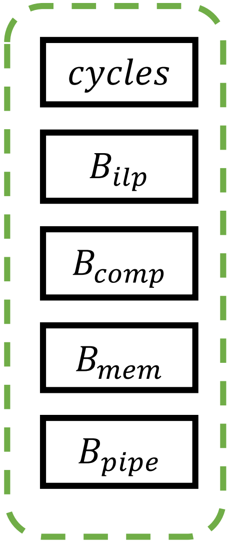
Algorithm variables

Kernel size

Steps

1. Benchmark instructions
2. Parse assembly instructions
3. Analyze instruction dependences and efficiencies
4. Construct a single-warp execution DAG
5. Calculate occupancy and blocks
6. Extend the DAG to multi-warp
7. Estimate running time and bottlenecks

Output



$cycles$

B_{ilp}

B_{comp}

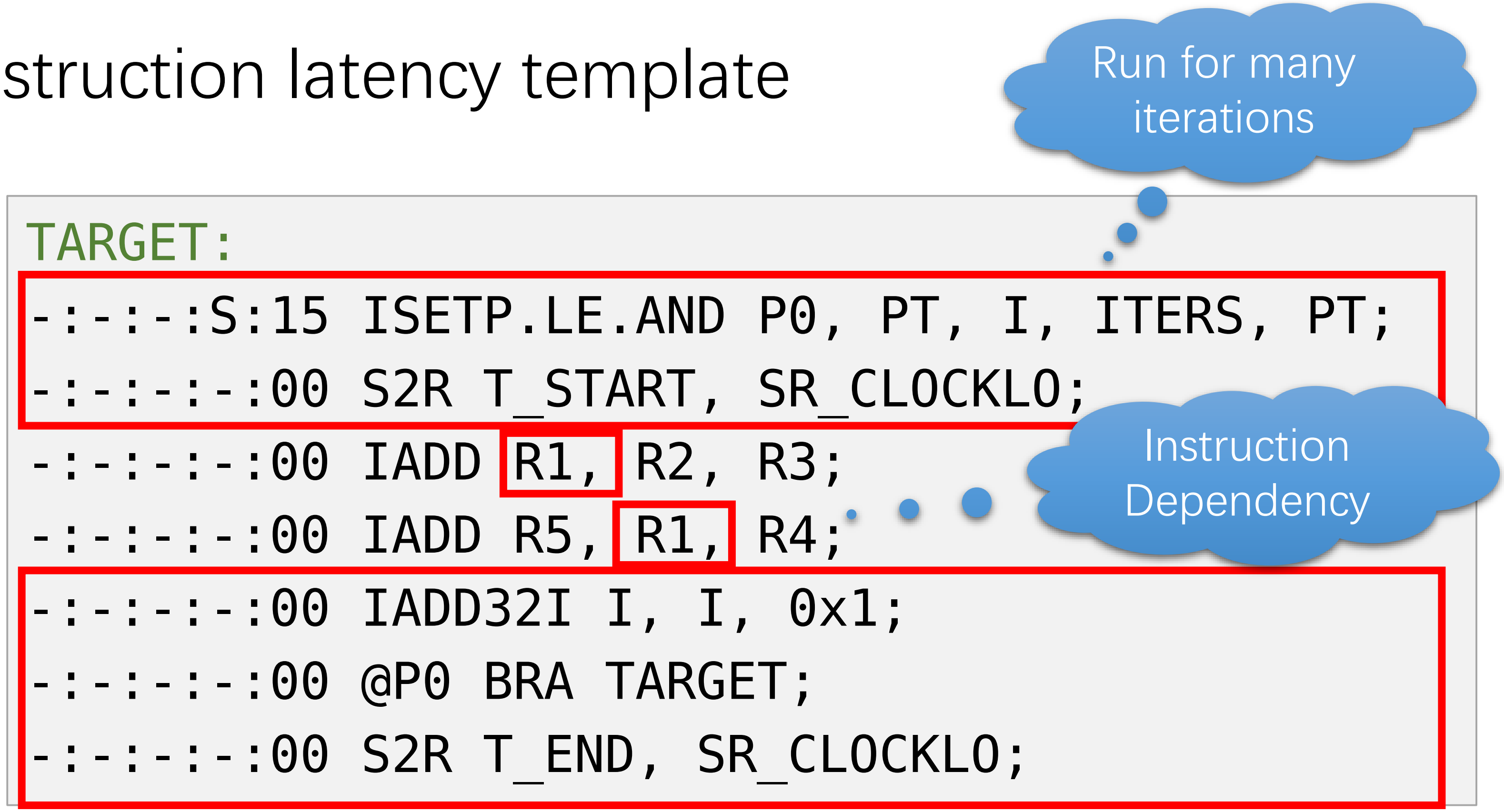
B_{mem}

B_{pipe}

Metrics

Benchmark

- Instruction latency template



```
TARGET:
-:-:-:S:15 ISETP.LE.AND P0, PT, I, ITERS, PT;
-:-:-:-:00 S2R T_START, SR_CLOCKLO;
-:-:-:-:00 IADD R1, R2, R3;
-:-:-:-:00 IADD R5, R1, R4;
-:-:-:-:00 IADD32I I, I, 0x1;
-:-:-:-:00 @P0 BRA TARGET;
-:-:-:-:00 S2R T_END, SR_CLOCKLO;
```

Run for many iterations

Instruction Dependency

Instruction Parser

- Parse each instruction, and push into a list.

```
-:-:-:-:00  LOP.AND R2, R1, 1;
```

```
-:-:-:-:00  LOP.AND R3, R4, 0x70;
```

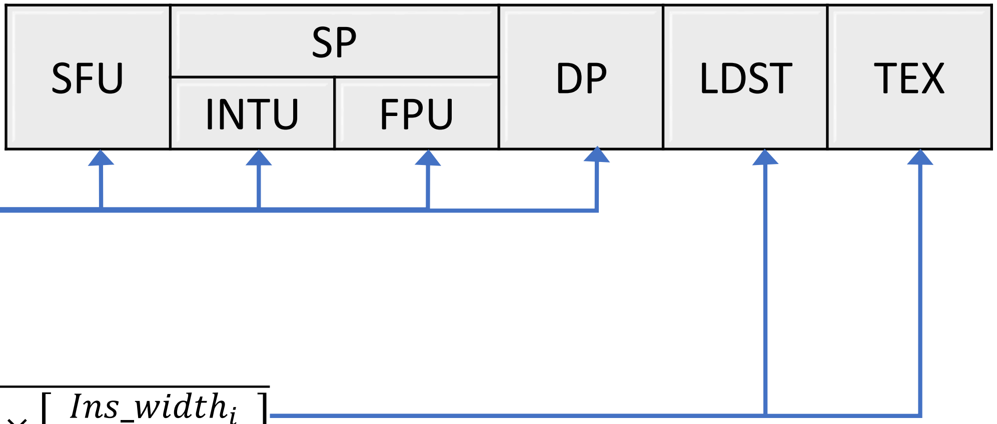
➡

```
ASM[0]
Name: LOP
Control: -:-:-:-:00
Modifier: AND
Dest: R2
Source1: R1
ASM[1]
Name: LOP
Control: -:-:-:-:00
Modifier: AND
Dest: R3
Source1: R4
```

Instruction Parser

- Efficiency = available units (bandwidth) / request units
- Compute efficiency:

$$E_{u_i} = \frac{1}{\left\lceil \frac{Dispatches_{u_i} \times Warp\_size}{Units_{u_i}} \right\rceil}$$



SFU

SP

INTU

FPU

DP

LDST

TEX

- Memory efficiency:

$$E_{shared} = \frac{1}{N_i \times \left\lceil \frac{Dispatches_{ldst} \times Warp\_size}{Units_{ldst}} \right\rceil \times \left\lceil \frac{Ins\_width_i}{Bank\_width_i} \right\rceil}$$

$$E_{global} = \frac{1}{N_i \times \left\lceil \frac{Dispatches_{ldst} \times Warp\_size}{Units_{ldst}} \right\rceil}$$

Instruction replay:
issued instructions =
instructions + replayed instructions

DAG Constructor

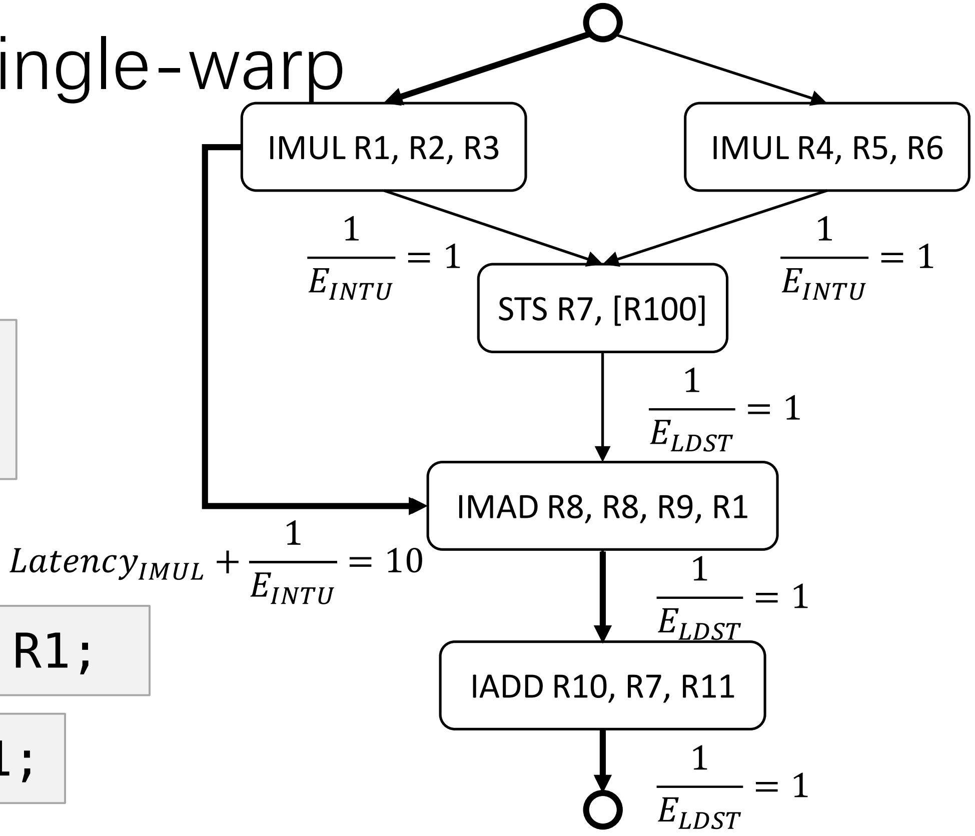
- Use a DAG to estimate single-warp running cycles

```
-:-:D:-:04  IMUL R1, R2, R3;
-:-:D:-:05  IMUL R4, R5, R6;
```

```
-:-:-:-:00  STS R7, [R100];
```

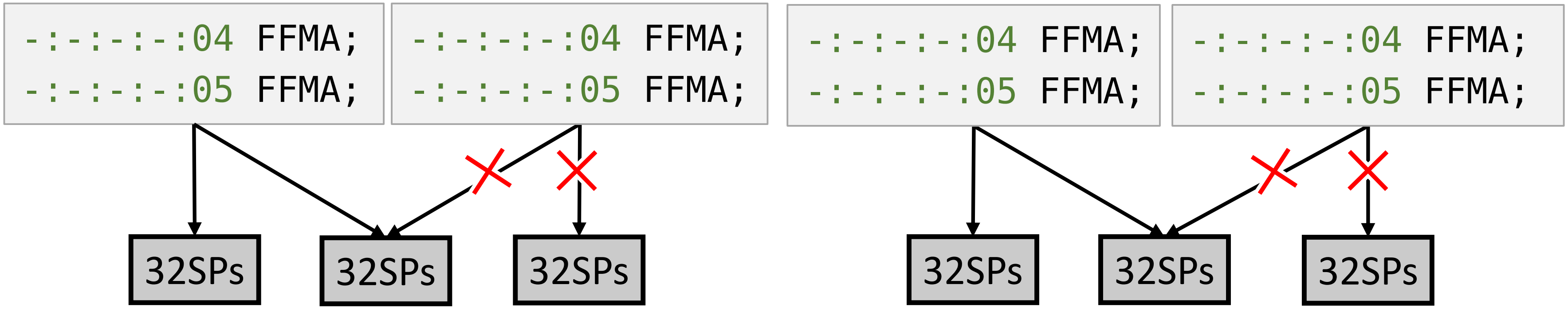
```
-:-:-:-:00  IMAD R8, R8, R9, R1;
```

```
-:-:-:-:00  IADD R10, R7, R11;
```



Multi-warp Resource Conflicts

- Extend to multi-warp



```
-:-:-:-:04 FFMA;
-:-:-:-:05 FFMA;
```

```
-:-:-:-:04 FFMA;
-:-:-:-:05 FFMA;
```

```
-:-:-:-:04 FFMA;
-:-:-:-:05 FFMA;
```

```
-:-:-:-:04 FFMA;
-:-:-:-:05 FFMA;
```

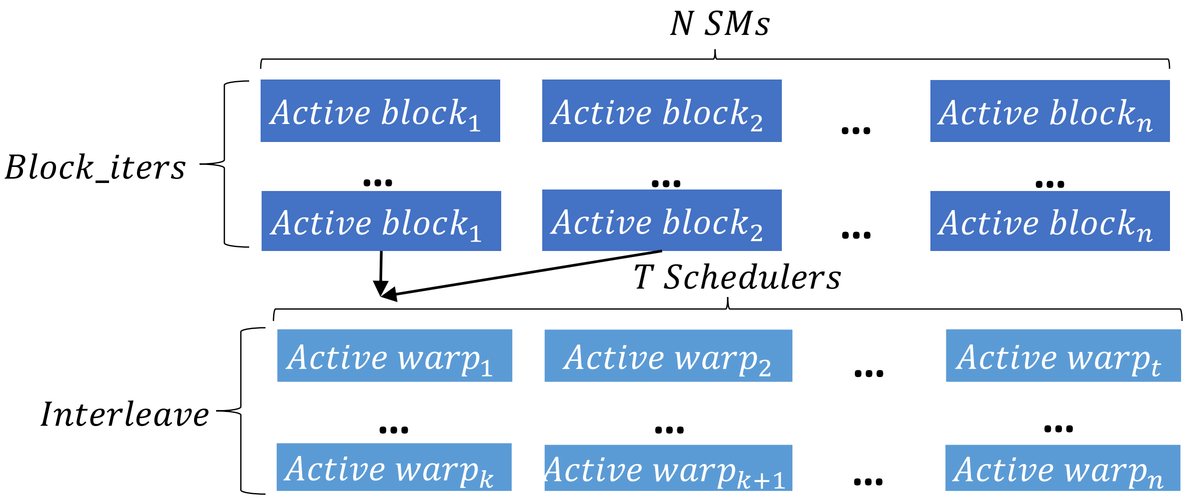
$$E_{u_i} = \frac{1}{\left\lceil \frac{Dispatches_{u_i} \times Warp\_size}{Units_{u_i}} \right\rceil}$$

$$Dispatches_u = \sum_{s=0}^{S-1} Dispatches_u^s \Rightarrow E_{SP} = \frac{1}{\left\lceil \frac{8 \times 32}{192} \right\rceil} = \frac{1}{2}$$

Occupancy

- Number of active blocks and warps

$$Cycles = Block\_iters * Block\_cycles$$
$$Block\_cycles = Interleave * Warp\_cycles$$



$N\ SMs$

$Block\_iters$

$Active\ block_1$ $Active\ block_2$ … $Active\ block_n$

… … …

$Active\ block_1$ $Active\ block_2$ … $Active\ block_n$

$T\ Schedulers$

$Interleave$

$Active\ warp_1$ $Active\ warp_2$ … $Active\ warp_t$

… … …

$Active\ warp_k$ $Active\ warp_{k+1}$ … $Active\ warp_n$

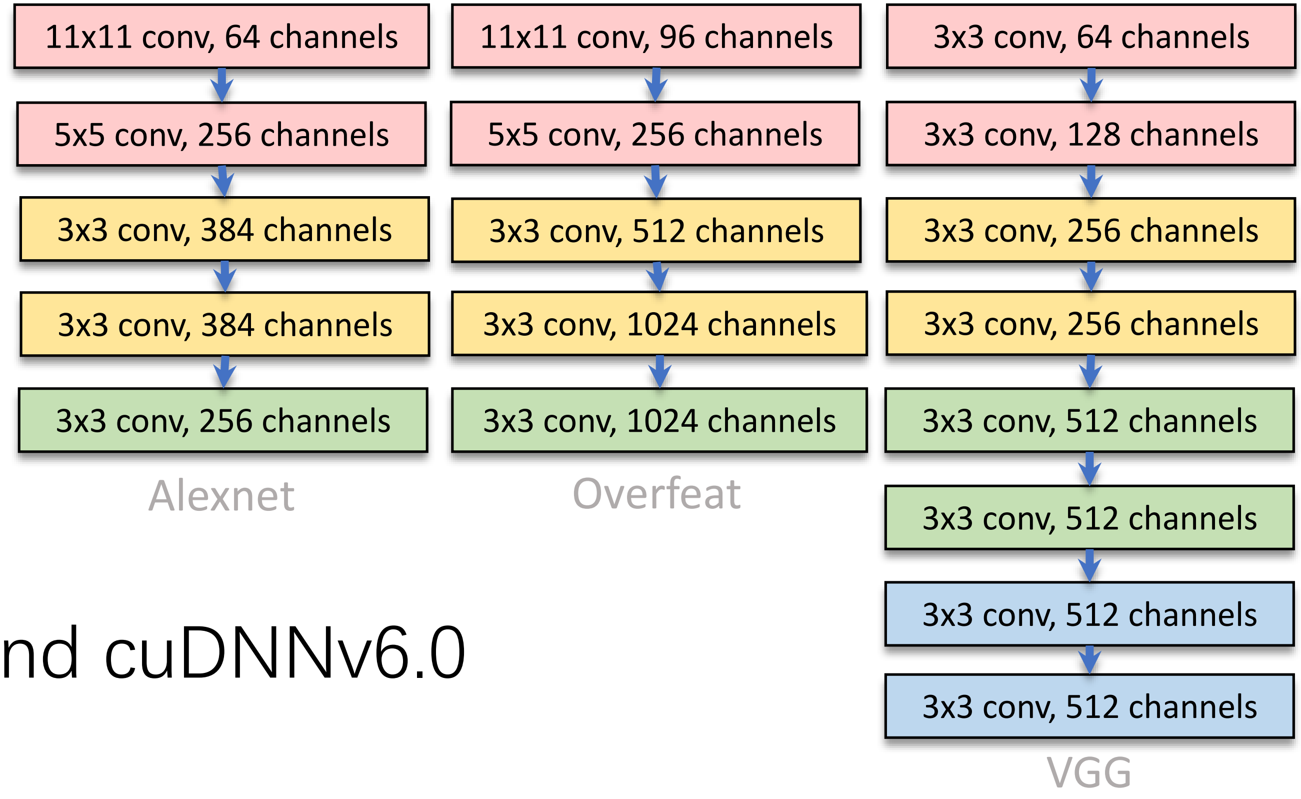
$$Block\_iters = \left\lceil \frac{Blocks}{Active\_blocks \times NSM} \right\rceil$$

$$Interleave = \left\lceil \frac{Active\_warps}{Schedulers} \right\rceil$$

$Active\_blocks$ and $Active\_warps$ depend on hardware resource usage

Evaluations

- Kepler K20m
- Convolution
  - Alexnet
  - Overfeat
  - VGG
- GEMM
- cuBLASv8.0 and cuDNNv6.0
  - FLOPS
  - Runtime metrics from *nvprof*



11x11 conv, 64 channels
5x5 conv, 256 channels
3x3 conv, 384 channels
3x3 conv, 384 channels
3x3 conv, 256 channels
Alexnet

11x11 conv, 96 channels
5x5 conv, 256 channels
3x3 conv, 512 channels
3x3 conv, 1024 channels
3x3 conv, 1024 channels
Overfeat

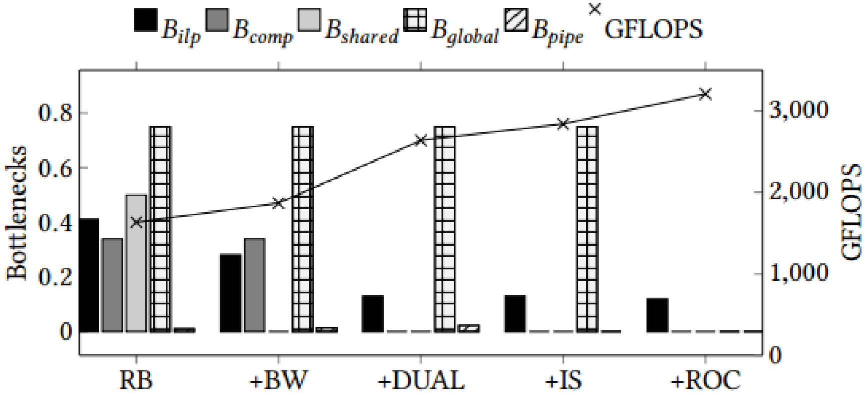
3x3 conv, 64 channels
3x3 conv, 128 channels
3x3 conv, 256 channels
3x3 conv, 256 channels
3x3 conv, 512 channels
3x3 conv, 512 channels
3x3 conv, 512 channels
3x3 conv, 512 channels
VGG

Optimization Steps

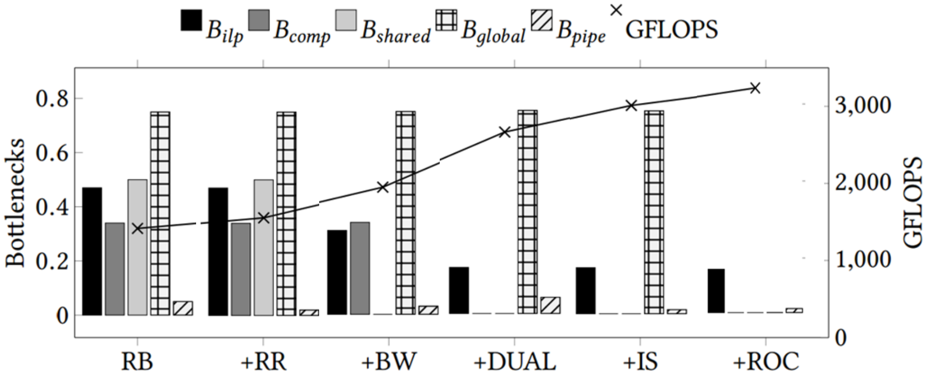
- Advisor:
- B_{ilp} ：Instruction level parallelism bottleneck, optimized by issuing instructions simultaneously. (*+DUAL*)
- B_{comp} ：Compute resource usage bottleneck, optimized by using more compute units. (*+DUAL*)
- B_{mem} ：Memory access bottleneck, optimized by high bandwidth instruction (*+BW*) and read-only-cache (*+ROC*).
- B_{pipe} ：Instruction pipeline bottleneck, optimized by instruction scheduling (*+IS*) and register reuse (*RR*).

Optimization Steps

- GEMM: 82% improvement
- Convolution: 114% improvement



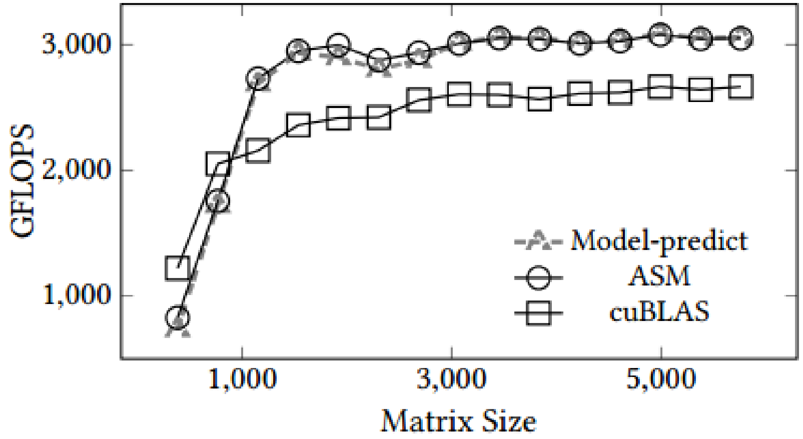
GEMM



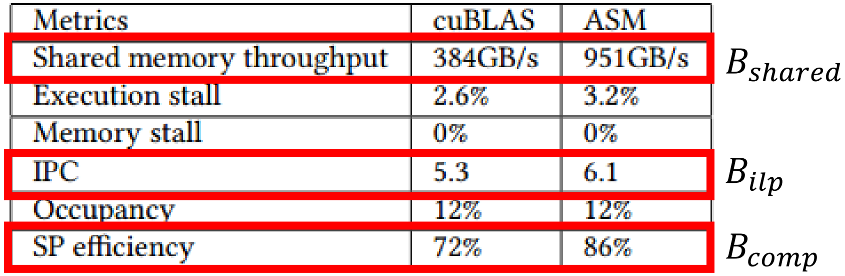
Convolution

GEMM Performance

- Compare performance and runtime metrics with *cuBLASv8.0*
- 20% speedup



Performance



Metrics	cuBLAS	ASM
Shared memory throughput	384GB/s	951GB/s
Execution stall	2.6%	3.2%
Memory stall	0%	0%
IPC	5.3	6.1
Occupancy	12%	12%
SP efficiency	72%	86%

B_{shared}

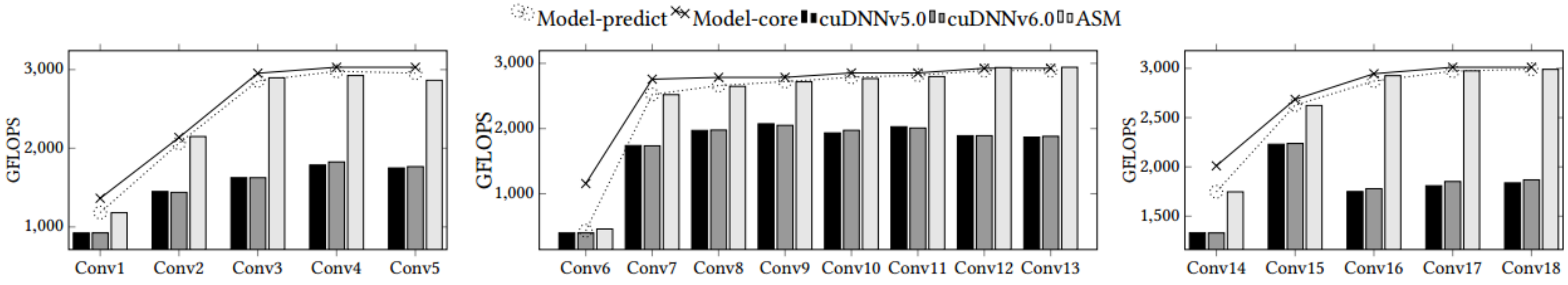
B_{ilp}

B_{comp}

Metrics

Convolution Performance

- Compare with *cuDNNv5.0* and *cuDNNv6.0*
- 40%-60% speedup

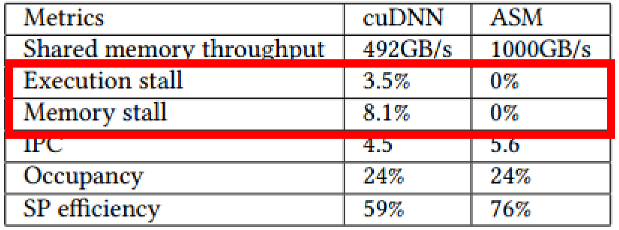


(a) Alexnet performance

(b) VGG performance

(c) Overfeat performance

Performance



Metrics	cuDNN	ASM
Shared memory throughput	492GB/s	1000GB/s
Execution stall	3.5%	0%
Memory stall	8.1%	0%
IPC	4.5	5.6
Occupancy	24%	24%
SP efficiency	59%	76%

B_{pipe}

Metrics

Conclusion

- A performance analysis framework that allows programmers to identify bottlenecks precisely.

- Advantages: accurate for estimation and bottlenecks
- Limitations: assembly instructions, so less portable
- Extension: multi-kernel program