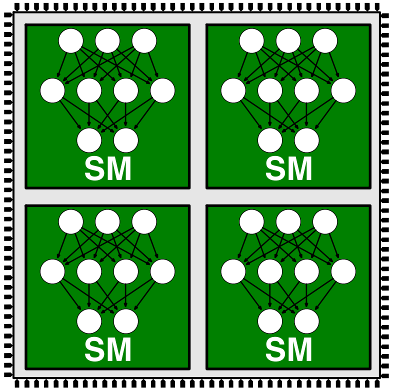
Neural Acceleration for GPU Throughput Processors

Amir Yazdanbakhsh Jongse Park Hardik Sharma
Pejman Lotfi-Kamran[*] Hadi Esmaeilzadeh

Alternative Computing Technologies (ACT) Lab
Georgia Institute of Technology

[*]The Institute for Research in Fundamental Sciences



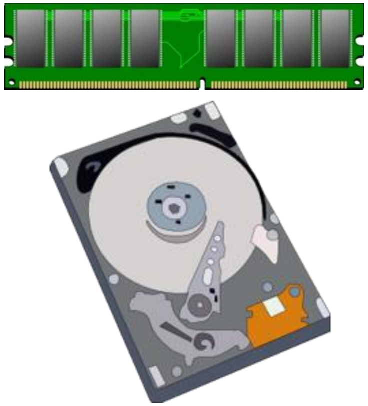
NGPU
Neurally Accelerated GPU

Approximate computing

Embracing imprecision

Relax the abstraction of "***near perfect***" **accuracy in**

Data Processing



Storage

Communication

Accept **imprecision** to improve

performance

energy dissipation

resource utilization **efficiency**

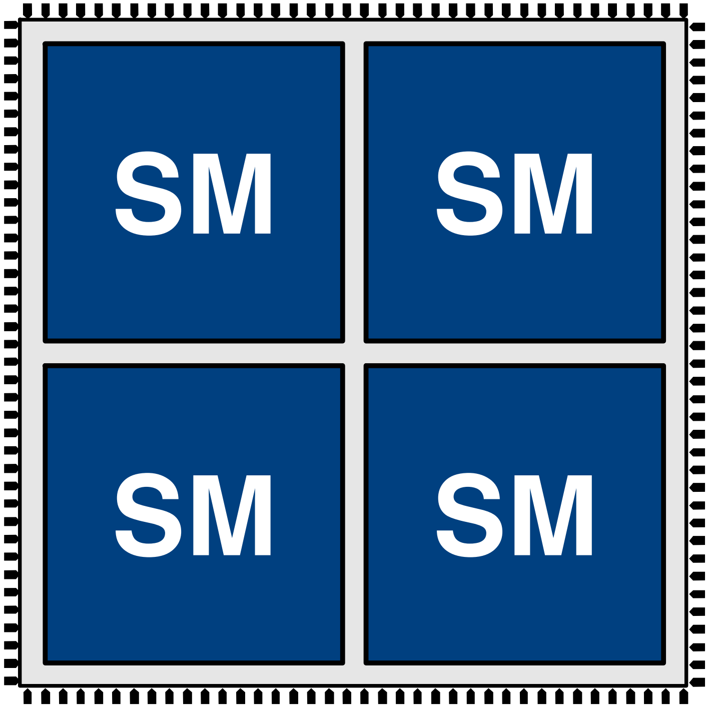
Opportunity

Many GPU applications are amenable to approximation

Augmented Reality

Computer Vision

Robotics

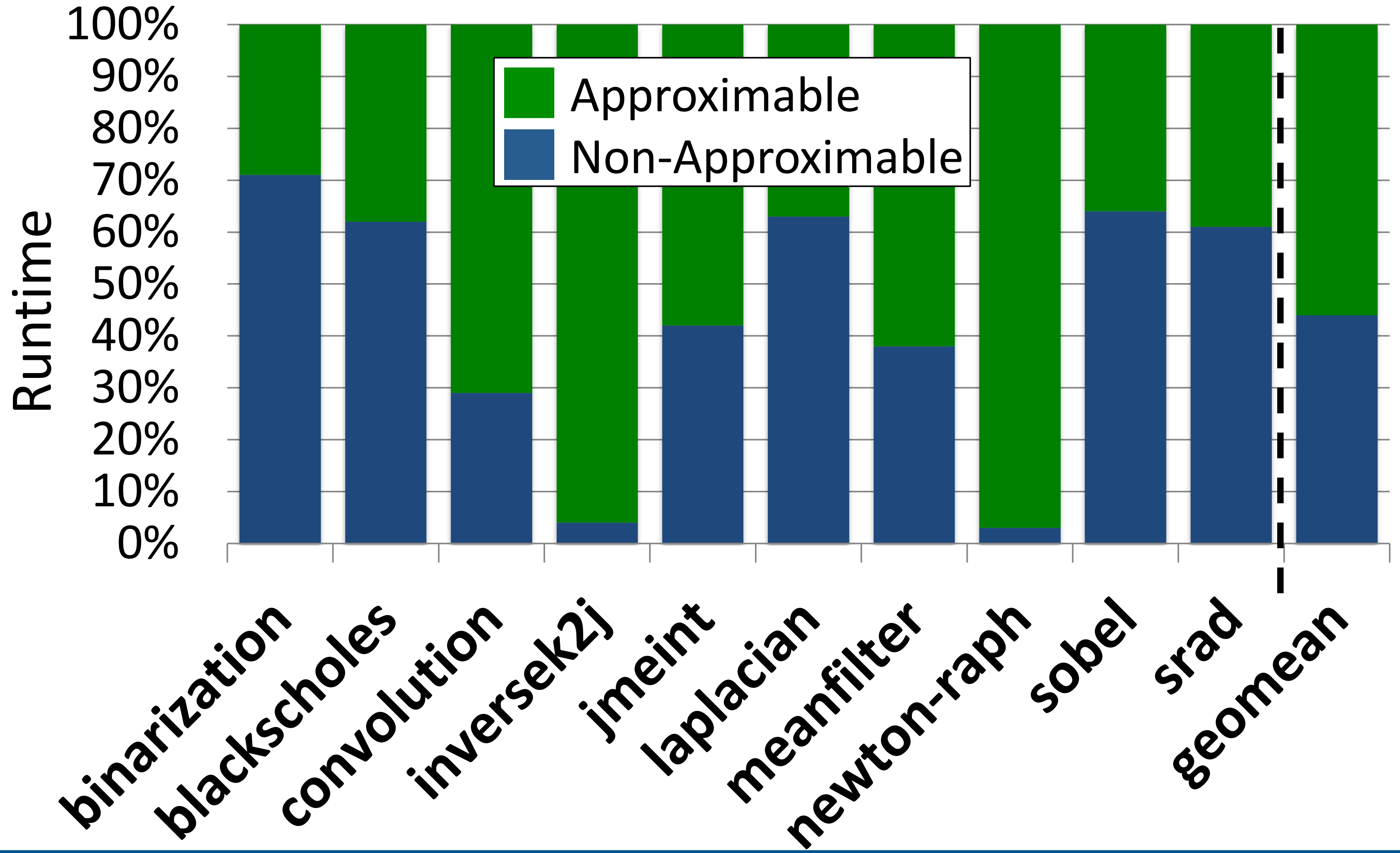


Machine Learning

Sensor Processing

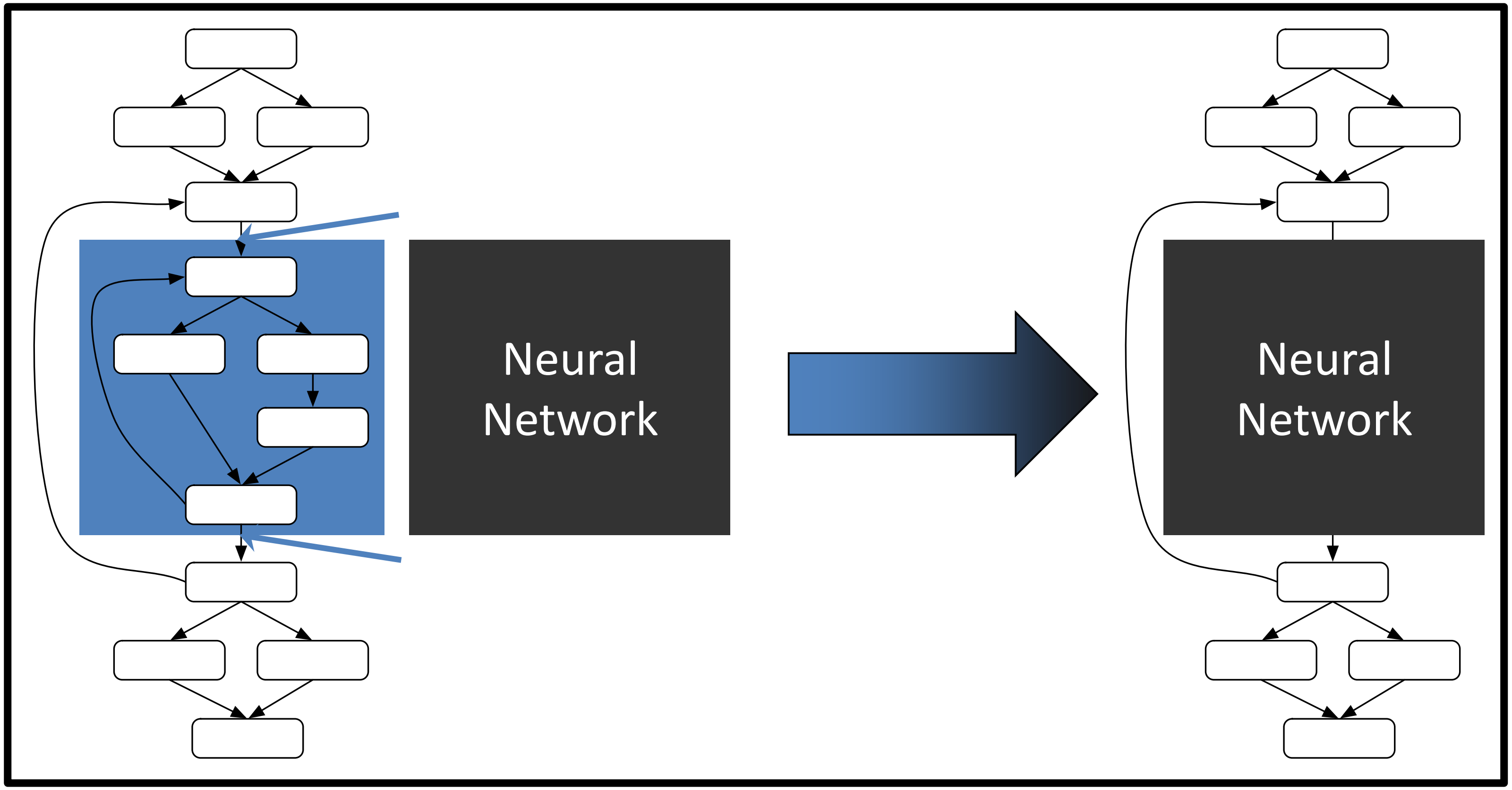
Multimedia

Opportunity

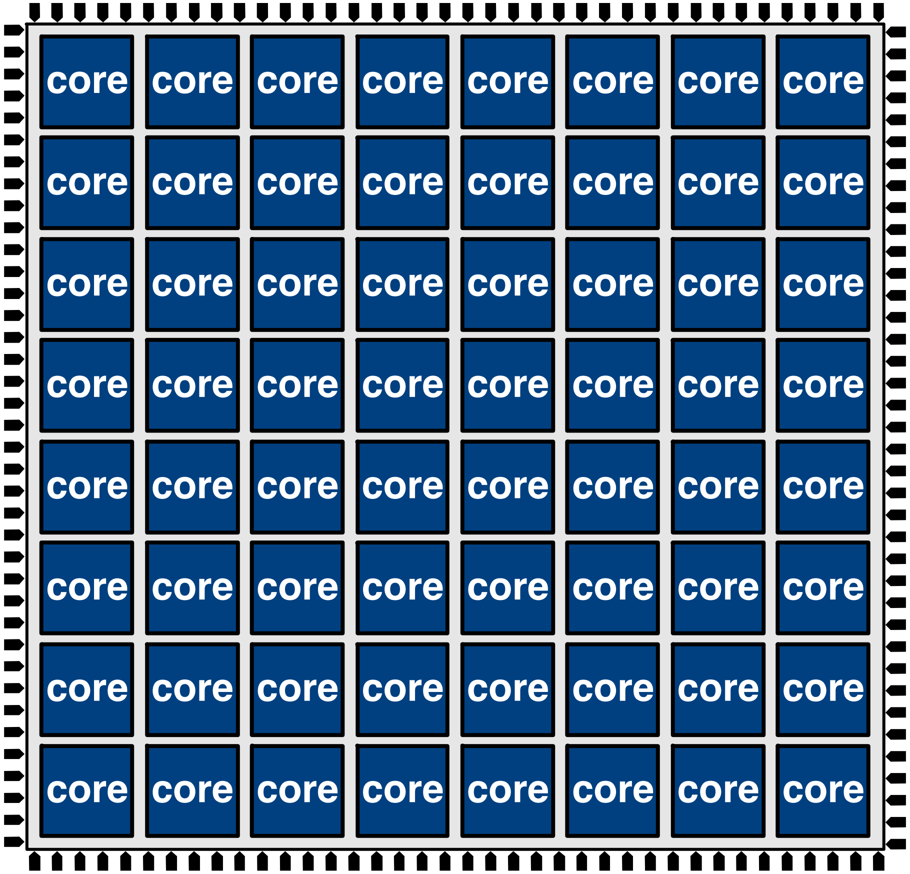


More than **55%** of application **runtime** and **energy** is in **neurally approximable regions**

Neural Transformation for GPUs

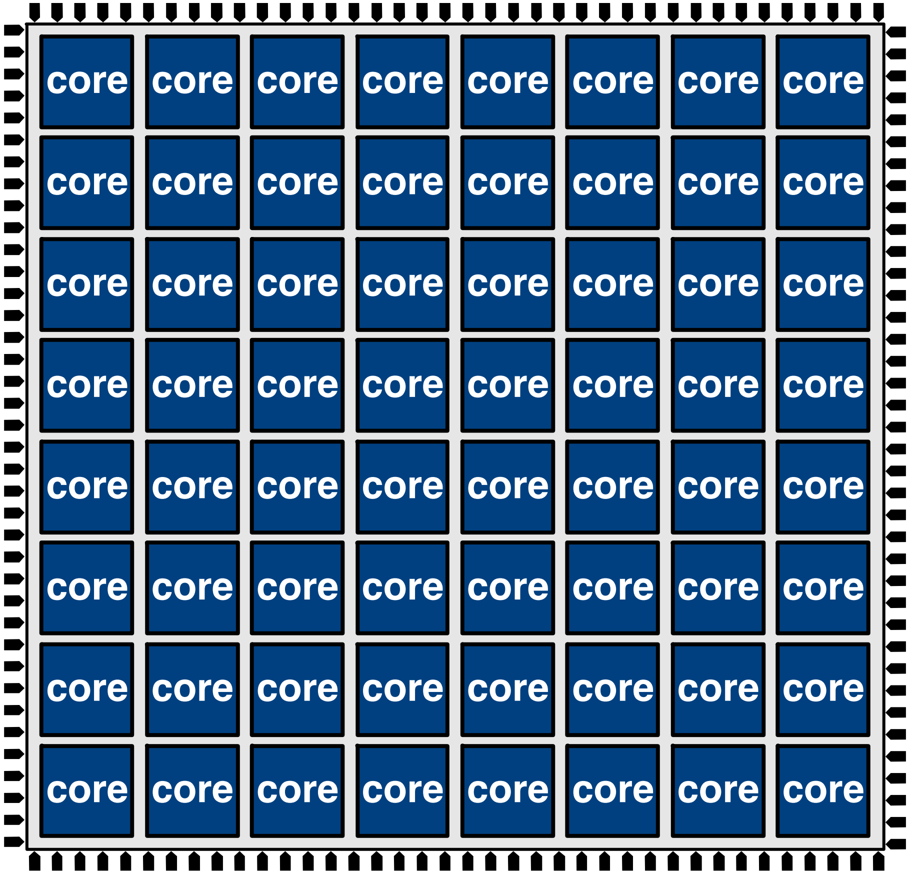


Challenges

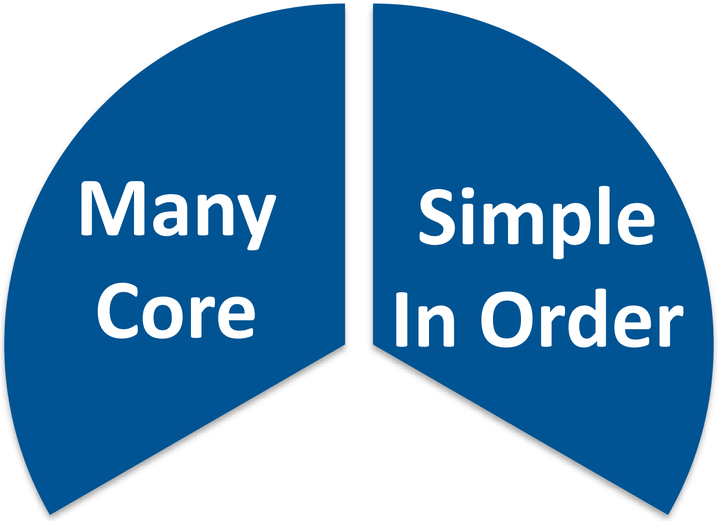

core core core core core core core core
core core core core core core core core
core core core core core core core core
core core core core core core core core
core core core core core core core core
core core core core core core core core
core core core core core core core core
core core core core core core core core

Many Core

Challenges



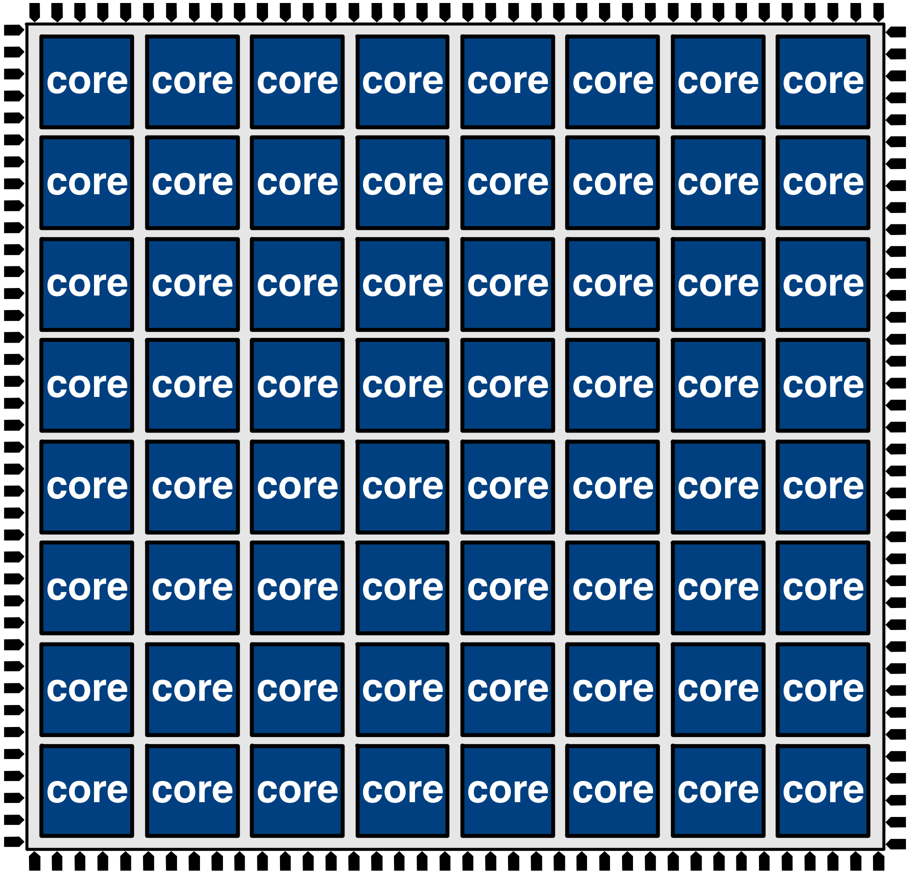
core	core	core	core	core	core	core	core
core	core	core	core	core	core	core	core
core	core	core	core	core	core	core	core
core	core	core	core	core	core	core	core
core	core	core	core	core	core	core	core
core	core	core	core	core	core	core	core
core	core	core	core	core	core	core	core
core	core	core	core	core	core	core	core

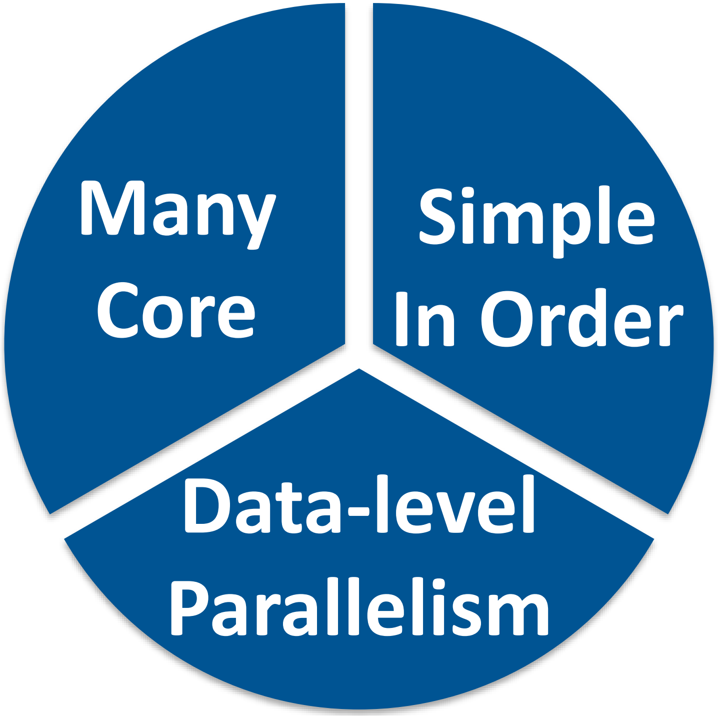


Many Core

Simple In Order

Challenges


core core core core core core core core
core core core core core core core core
core core core core core core core core
core core core core core core core core
core core core core core core core core
core core core core core core core core
core core core core core core core core
core core core core core core core core

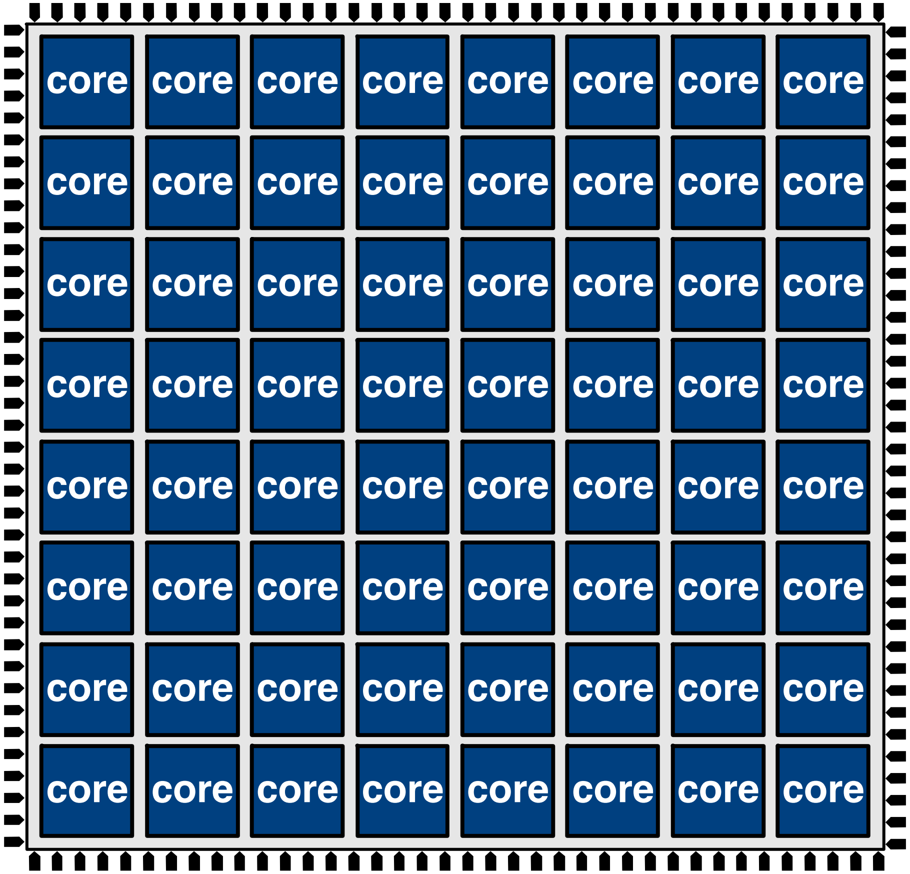


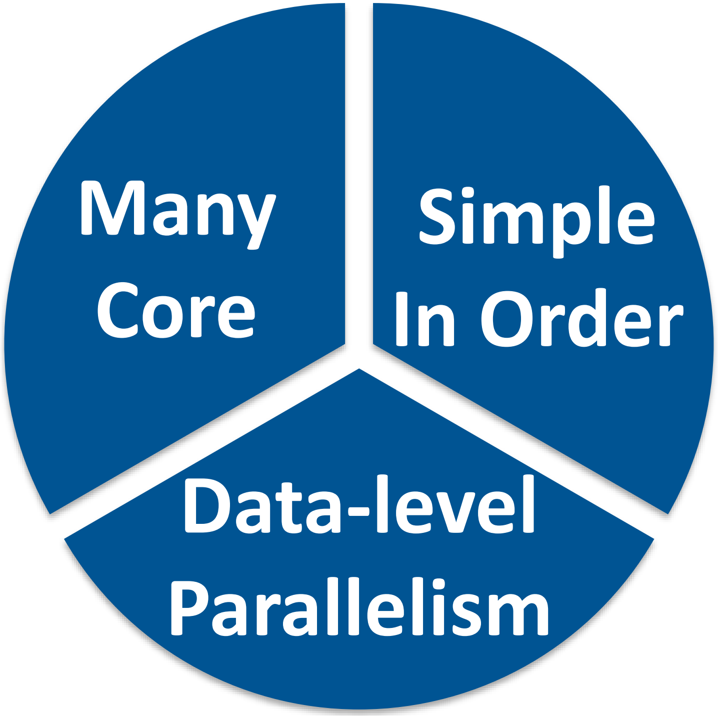
Many Core

Simple In Order

Data-level Parallelism

Challenges


core core core core core core core core
core core core core core core core core
core core core core core core core core
core core core core core core core core
core core core core core core core core
core core core core core core core core
core core core core core core core core
core core core core core core core core



Many Core

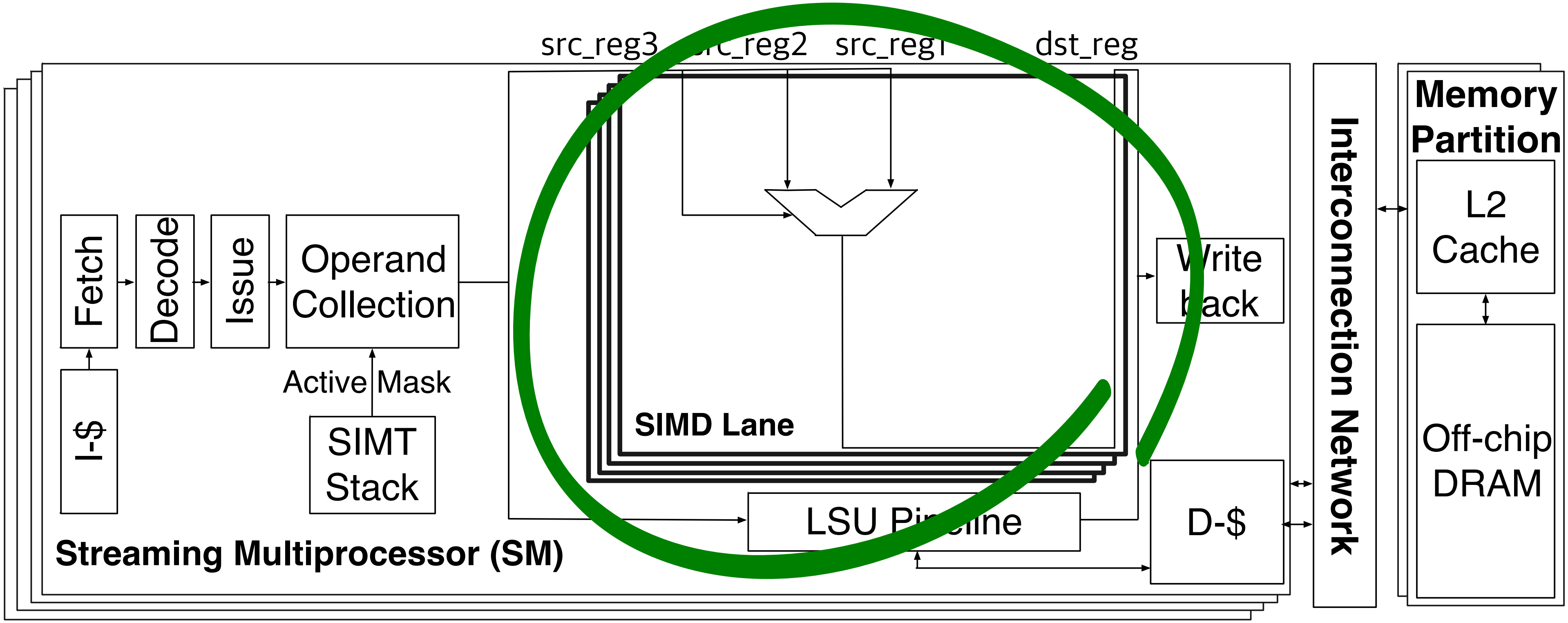
Simple In Order

Data-level Parallelism

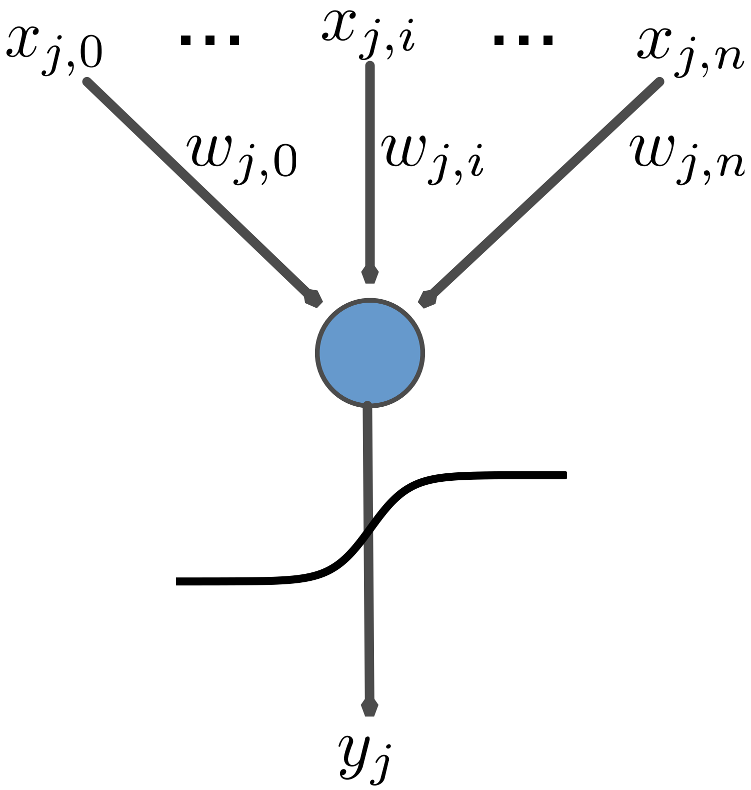
Augmenting the CPU based neural processing units to each SIMD lane imposes **31.2%** area overhead

NGPU

Neurally-Accelerated GPU Architecture



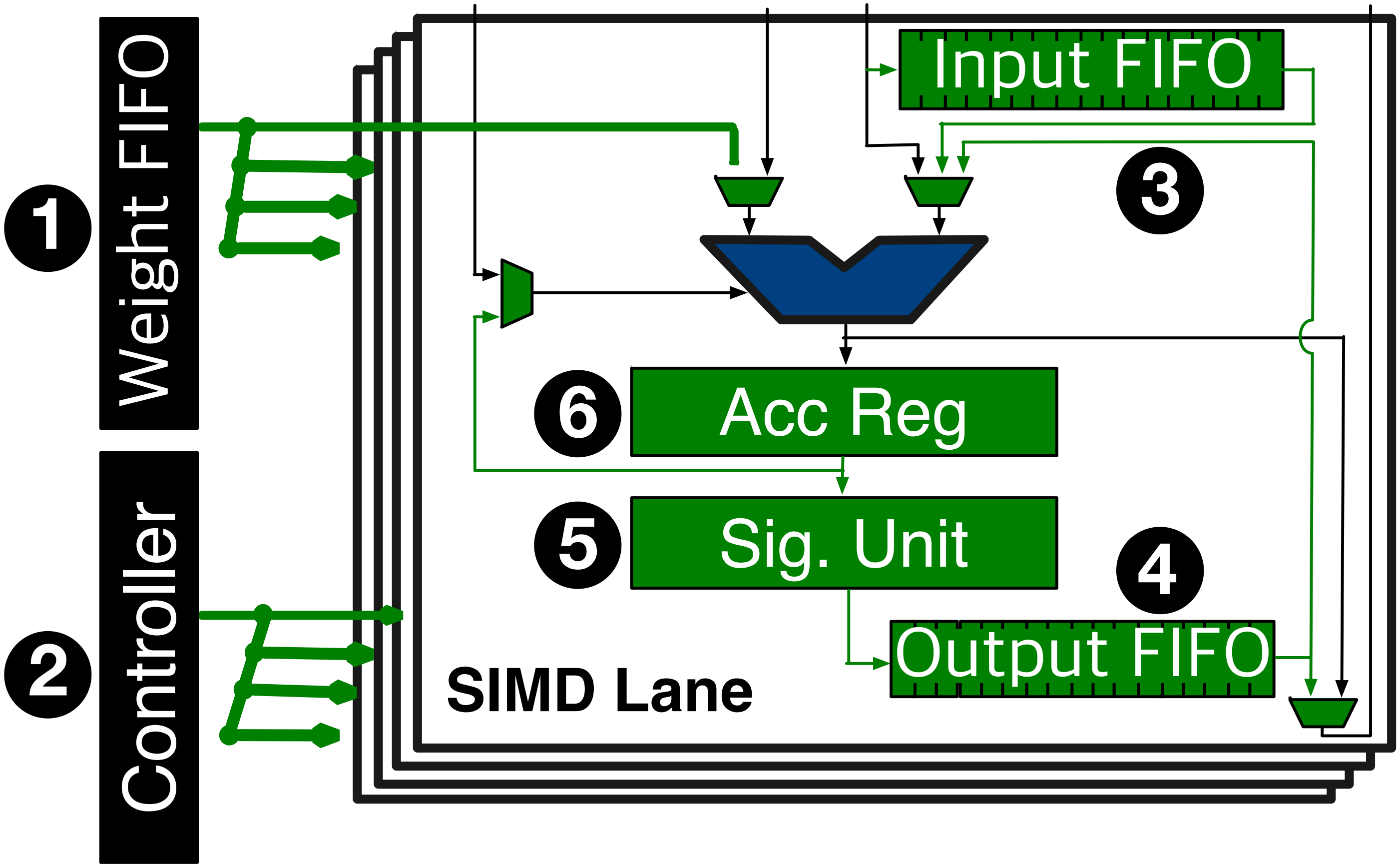
Neuronal Network Operations



$$
\begin{aligned}
y_j = \; & \\
& sigmoid(\\
& \quad w_{j,0} \times x_{j,0} + \\
& \quad \ldots \\
& \quad w_{j,i} \times x_{j,i} + \\
& \quad \ldots \\
& \quad w_{j,n} \times x_{j,n} + \\
&)
\end{aligned}
$$

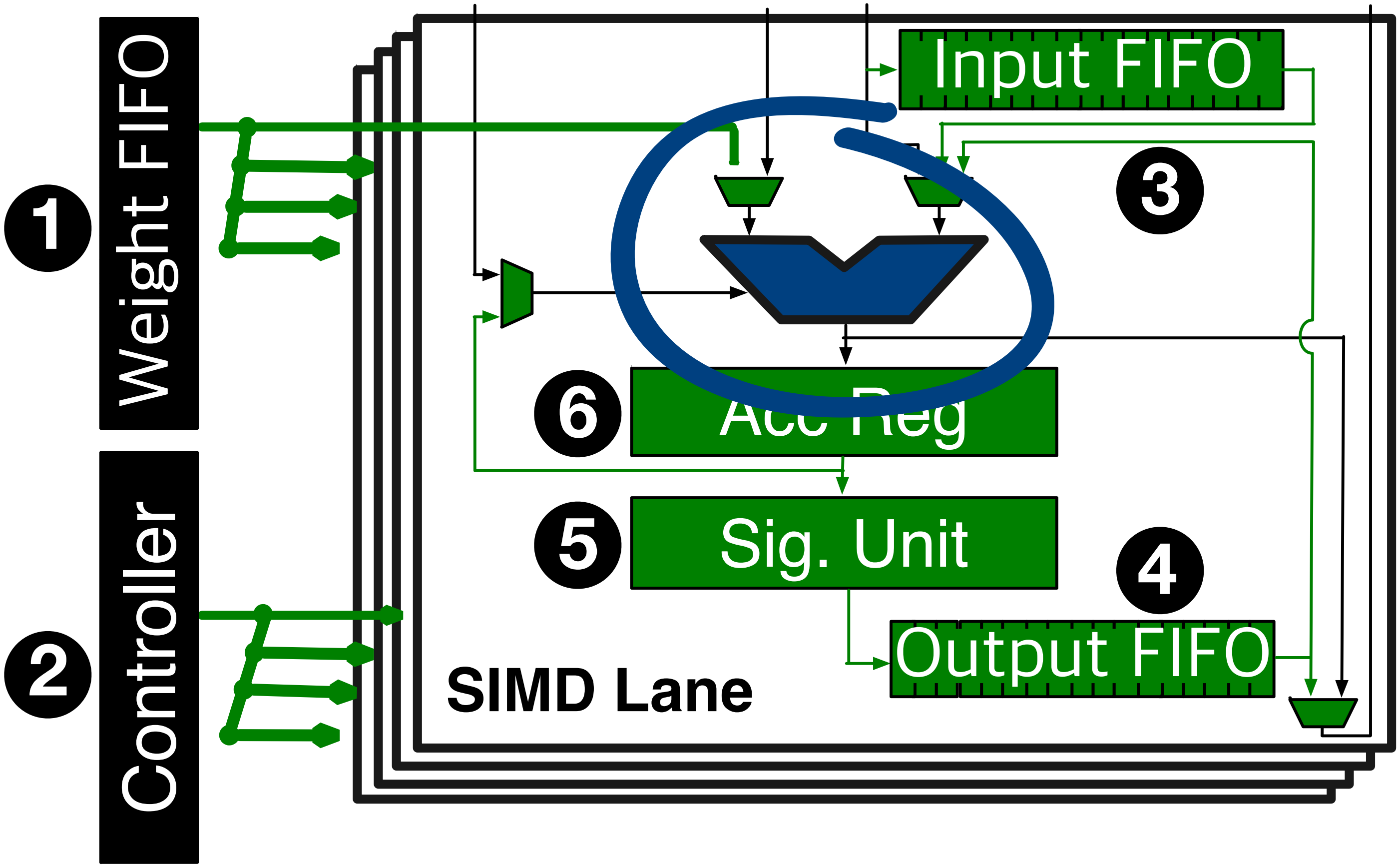
NGPU

Neurally-Accelerated GPU Architecture



NGPU

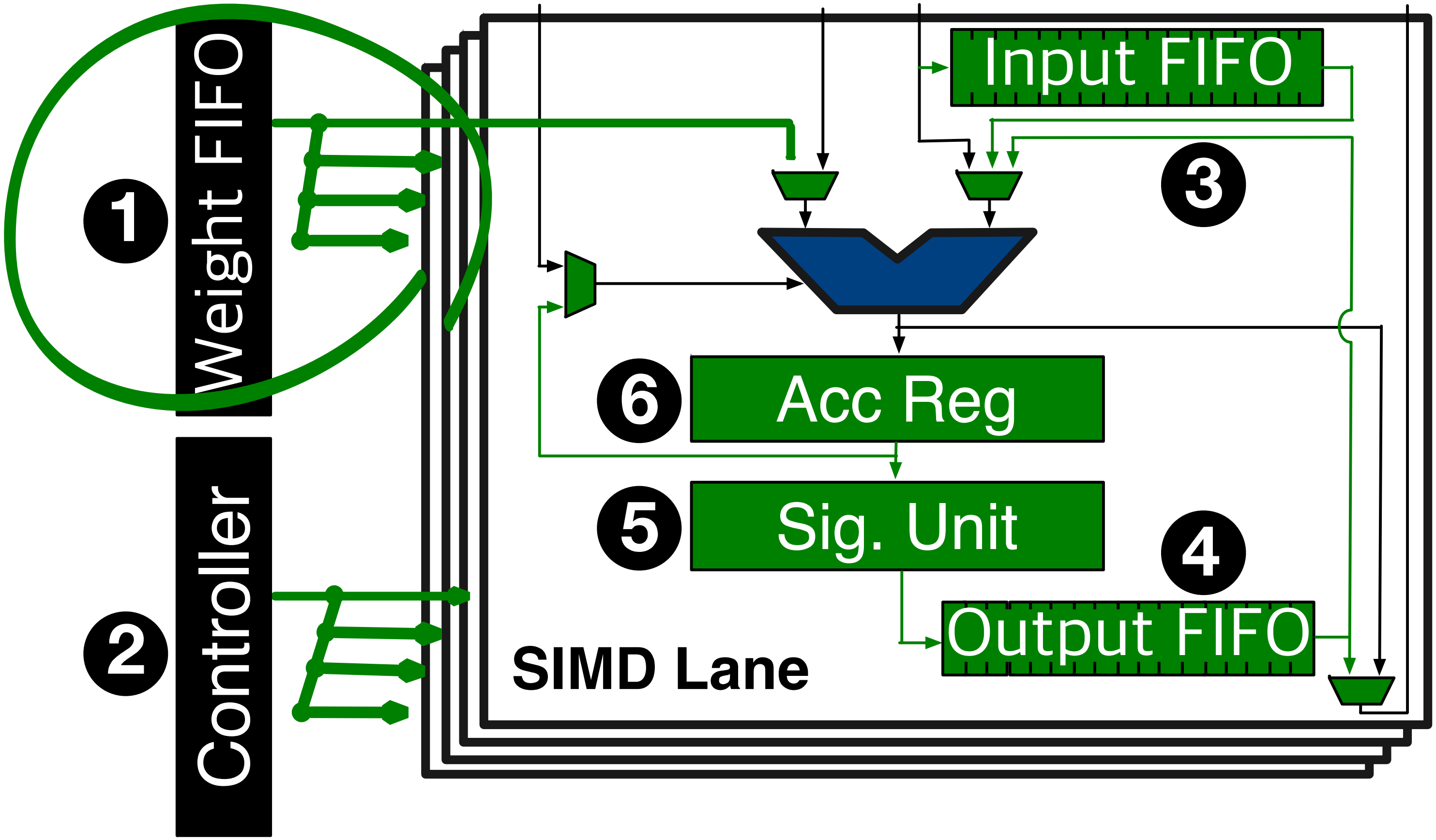
Neurally-Accelerated GPU Architecture



NGPU reuses the existing ALU in each SIMD lane

NGPU

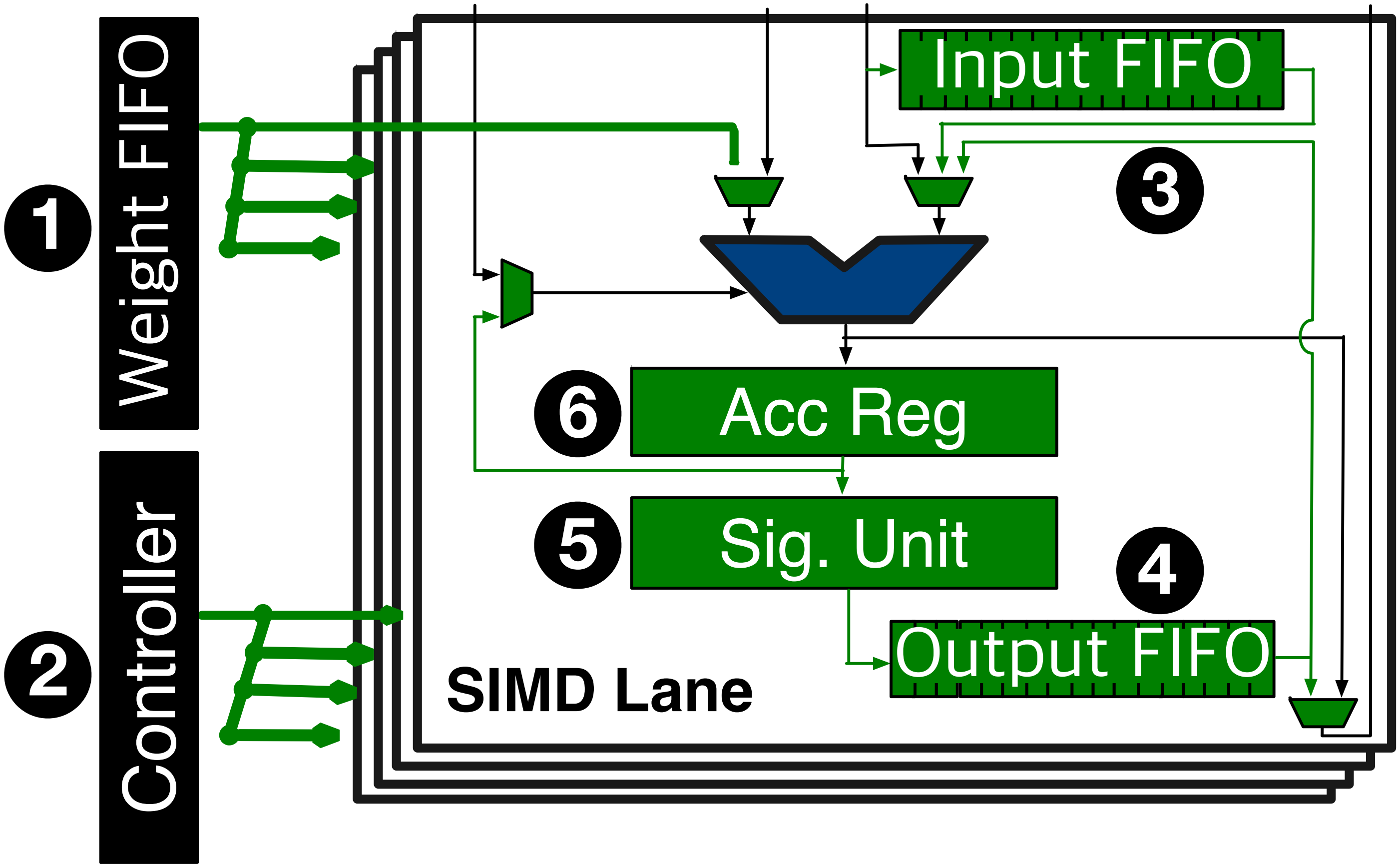
Neurally-Accelerated GPU Architecture



Weight FIFO is shared among all the SIMD lanes

NGPU

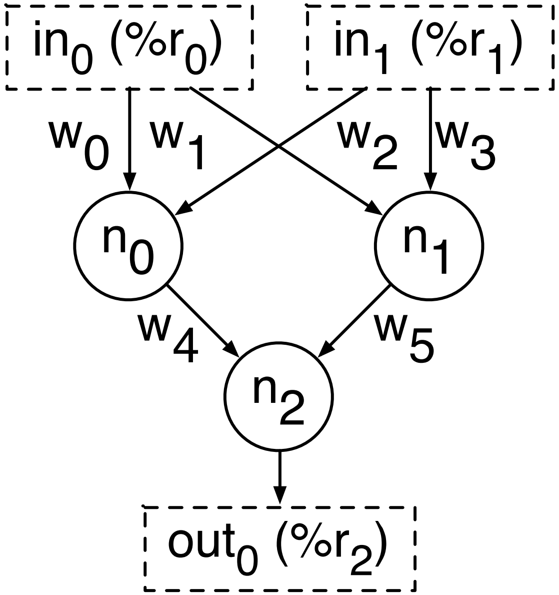
Neurally-Accelerated GPU Architecture



Overall **NGPU** has **≤1%** area overhead

NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;
st.global [addr2], %r2;
```

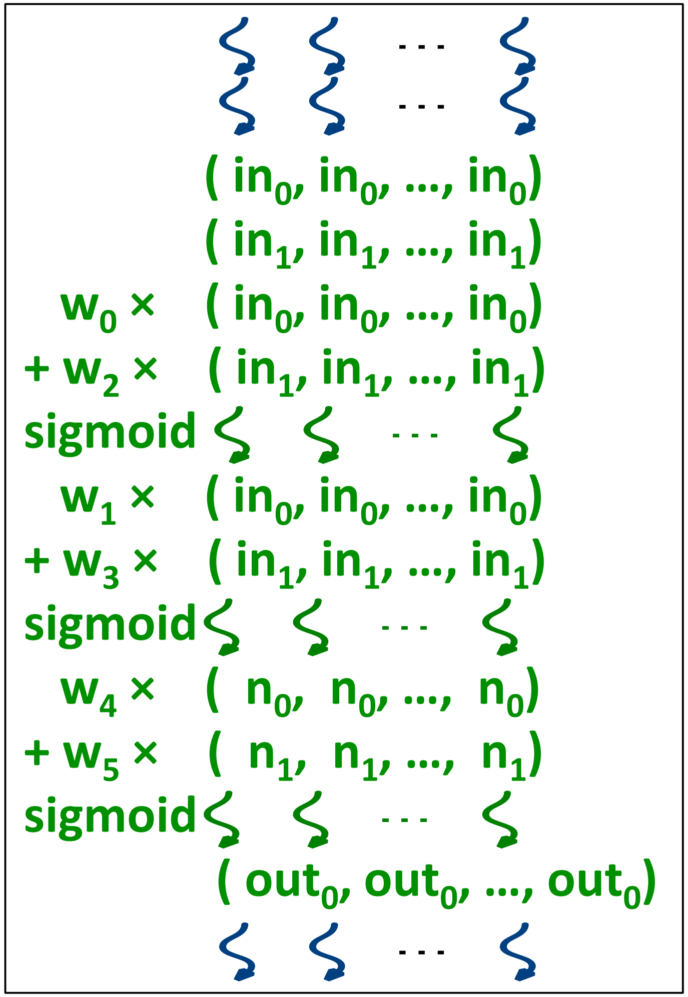


Neurally Accelerated GPU Application

Neural Network

NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;







st.global [addr2], %r2;
```



$(in_0, in_0, \ldots, in_0)$

$(in_1, in_1, \ldots, in_1)$

$w_0 \times (in_0, in_0, \ldots, in_0)$

$+ w_2 \times (in_1, in_1, \ldots, in_1)$

sigmoid

$w_1 \times (in_0, in_0, \ldots, in_0)$

$+ w_3 \times (in_1, in_1, \ldots, in_1)$

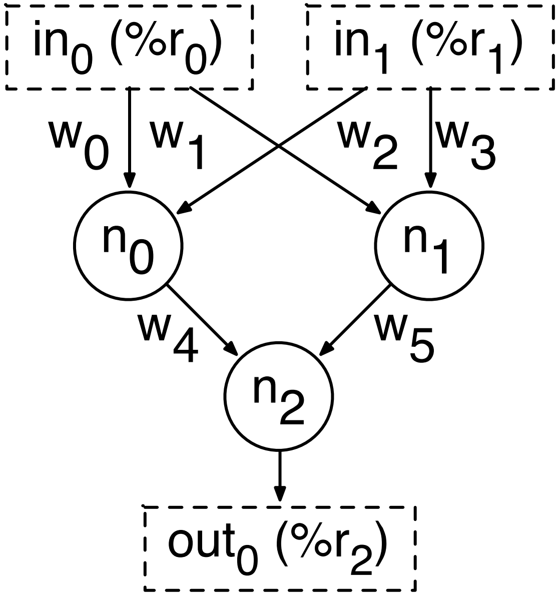
sigmoid

$w_4 \times (n_0, n_0, \ldots, n_0)$

$+ w_5 \times (n_1, n_1, \ldots, n_1)$

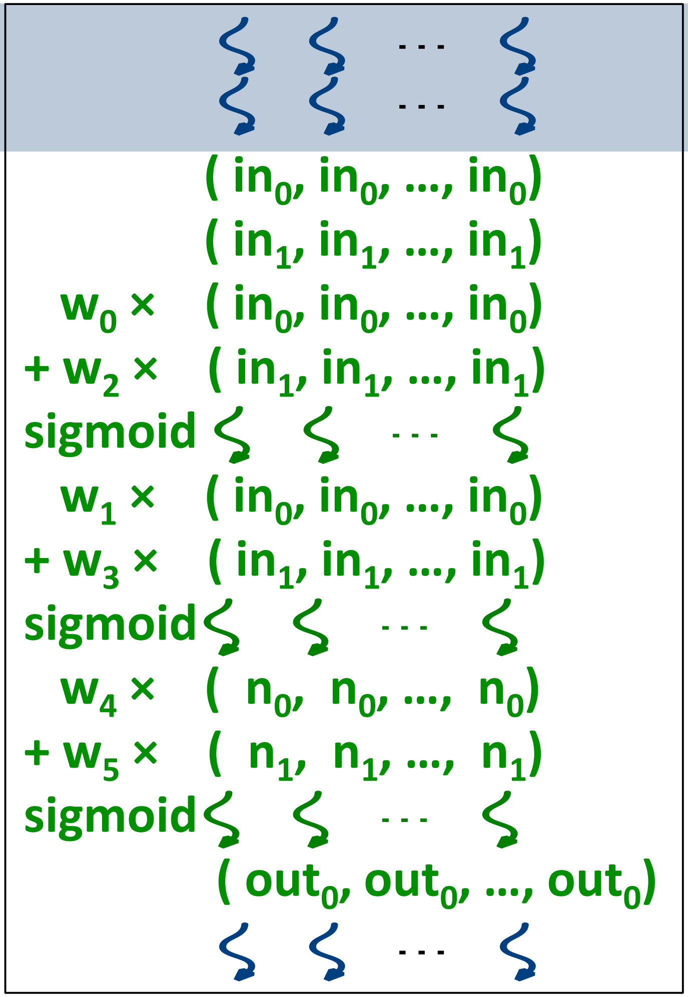
sigmoid

$(out_0, out_0, \ldots, out_0)$



NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;




st.global [addr2], %r2;
```



· · ·

· · ·

$(in_0, in_0, ..., in_0)$

$(in_1, in_1, ..., in_1)$

$w_0 \times (in_0, in_0, ..., in_0)$

$+ w_2 \times (in_1, in_1, ..., in_1)$

sigmoid · · ·

$w_1 \times (in_0, in_0, ..., in_0)$

$+ w_3 \times (in_1, in_1, ..., in_1)$

sigmoid · · ·

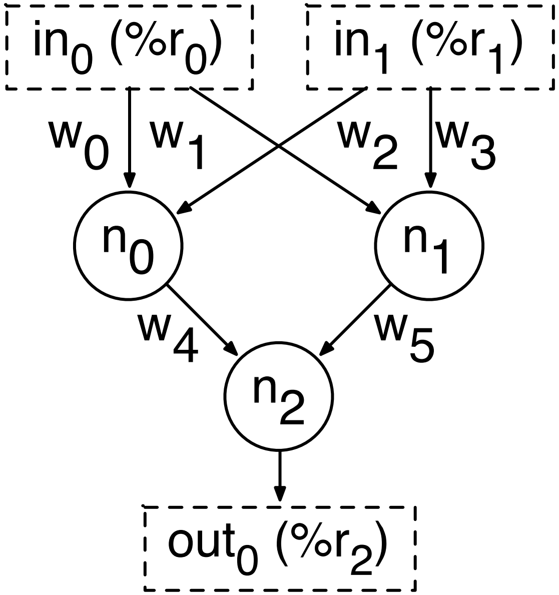
$w_4 \times (n_0, n_0, ..., n_0)$

$+ w_5 \times (n_1, n_1, ..., n_1)$

sigmoid · · ·

$(out_0, out_0, ..., out_0)$

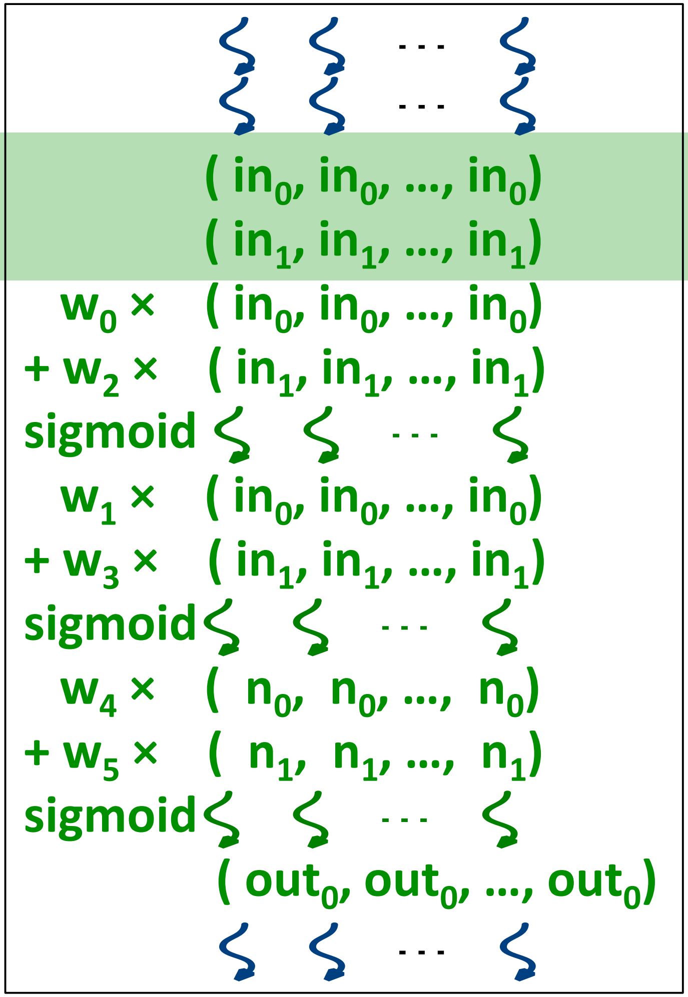
· · ·



SIMD lanes are in normal mode and performs precise computation

NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;









st.global [addr2], %r2;
```



$(in_0, in_0, \ldots, in_0)$

$(in_1, in_1, \ldots, in_1)$

$w_0 \times (in_0, in_0, \ldots, in_0)$

$+ w_2 \times (in_1, in_1, \ldots, in_1)$

sigmoid

$w_1 \times (in_0, in_0, \ldots, in_0)$

$+ w_3 \times (in_1, in_1, \ldots, in_1)$

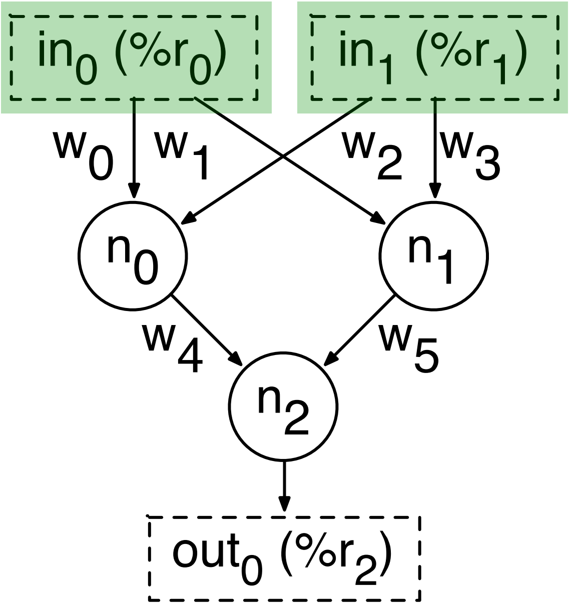
sigmoid

$w_4 \times (n_0, n_0, \ldots, n_0)$

$+ w_5 \times (n_1, n_1, \ldots, n_1)$

sigmoid

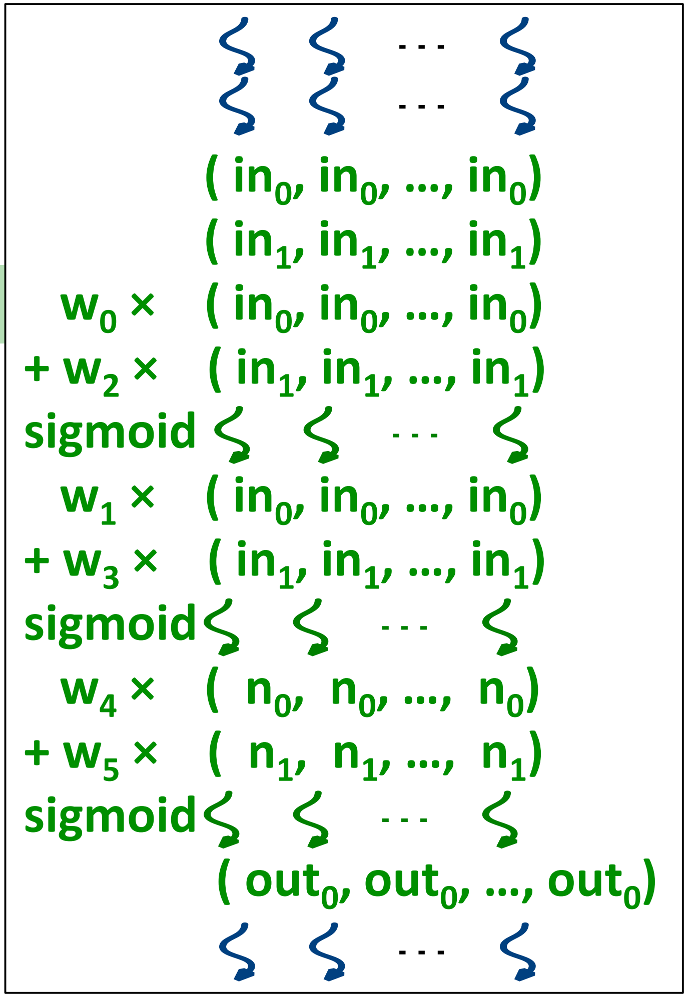
$(out_0, out_0, \ldots, out_0)$



SIMD lanes enter neural mode

NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;









st.global [addr2], %r2;
```



$(in_0, in_0, \ldots, in_0)$

$(in_1, in_1, \ldots, in_1)$

$w_0 \times (in_0, in_0, \ldots, in_0)$

$+ w_2 \times (in_1, in_1, \ldots, in_1)$

sigmoid

$w_1 \times (in_0, in_0, \ldots, in_0)$

$+ w_3 \times (in_1, in_1, \ldots, in_1)$

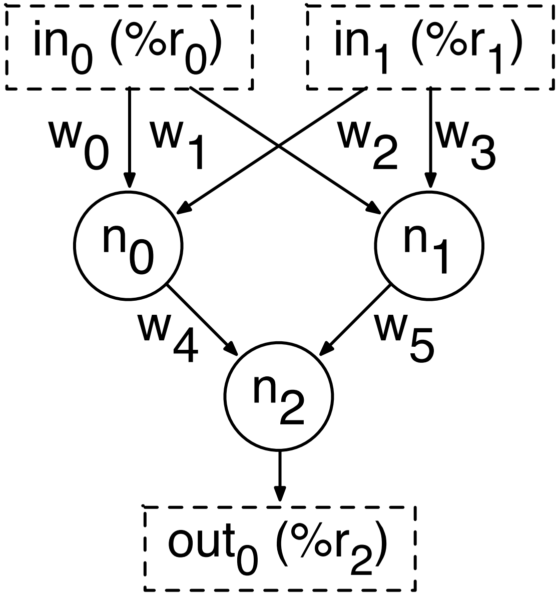
sigmoid

$w_4 \times (n_0, n_0, \ldots, n_0)$

$+ w_5 \times (n_1, n_1, \ldots, n_1)$

sigmoid

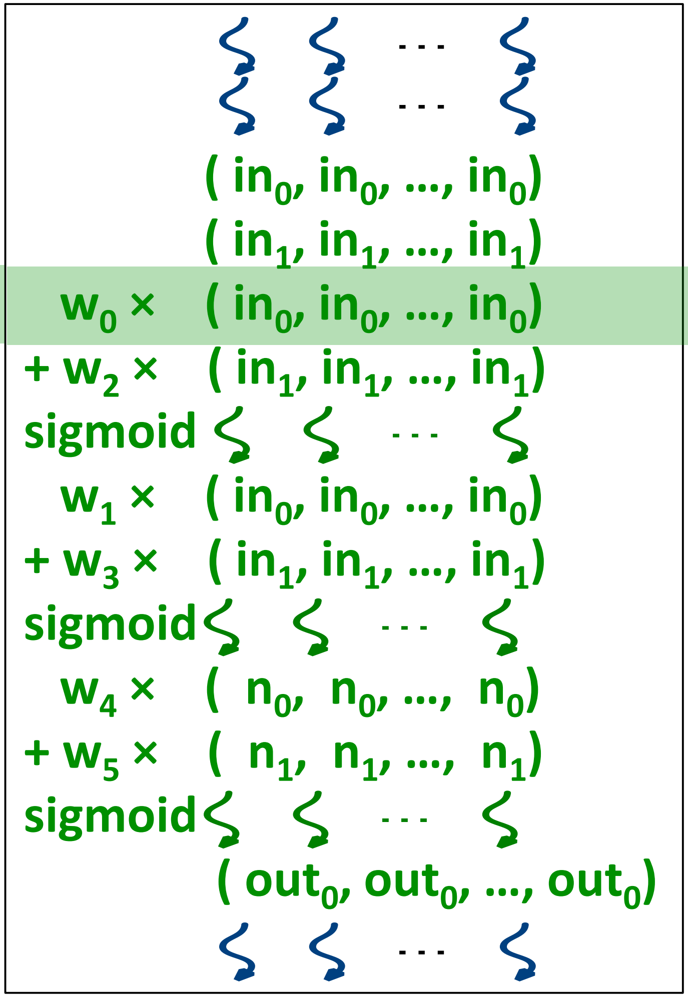
$(out_0, out_0, \ldots, out_0)$



SIMD starts the calculation of the neural network

NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;









st.global [addr2], %r2;
```



$(in_0, in_0, \ldots, in_0)$

$(in_1, in_1, \ldots, in_1)$

$w_0 \times (in_0, in_0, \ldots, in_0)$

$+ w_2 \times (in_1, in_1, \ldots, in_1)$

sigmoid

$w_1 \times (in_0, in_0, \ldots, in_0)$

$+ w_3 \times (in_1, in_1, \ldots, in_1)$

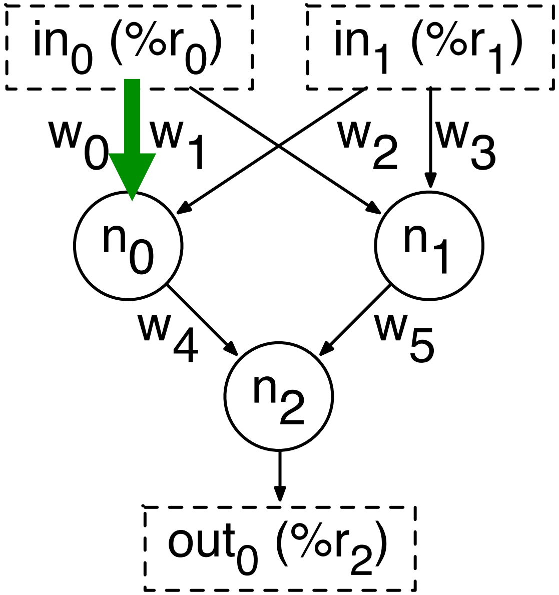
sigmoid

$w_4 \times (n_0, n_0, \ldots, n_0)$

$+ w_5 \times (n_1, n_1, \ldots, n_1)$

sigmoid

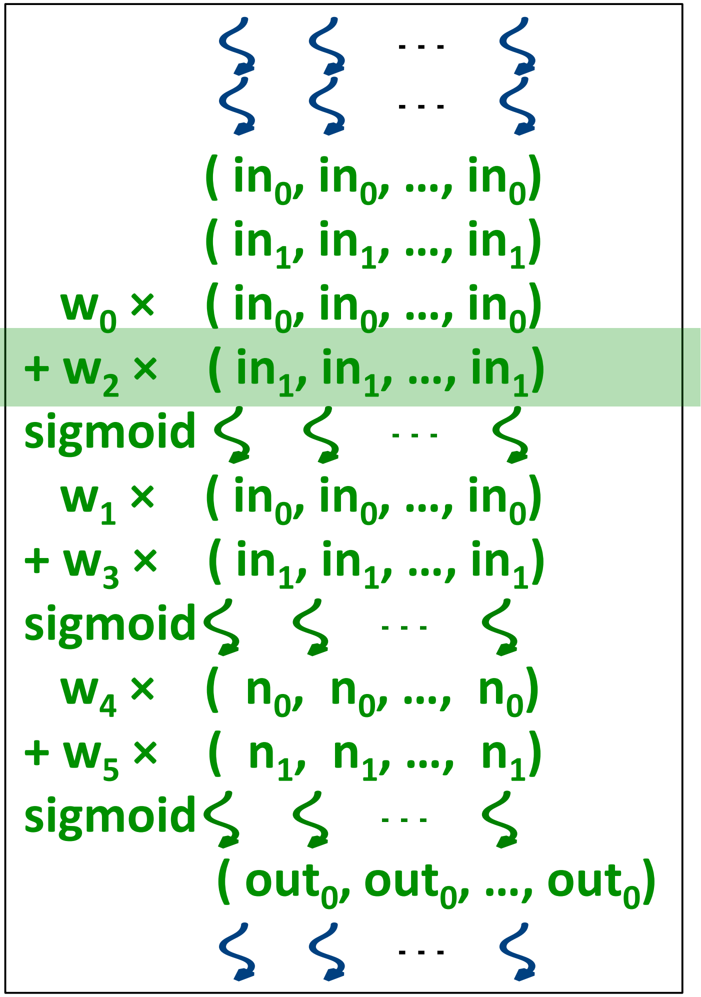
$(out_0, out_0, \ldots, out_0)$



The neurally accelerated SIMD lanes autonomously calculate the neural outputs in lock-step

NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;









st.global [addr2], %r2;
```



(in_0, in_0, …, in_0)

(in_1, in_1, …, in_1)

w_0 × (in_0, in_0, …, in_0)

+ w_2 × (in_1, in_1, …, in_1)

sigmoid

w_1 × (in_0, in_0, …, in_0)

+ w_3 × (in_1, in_1, …, in_1)

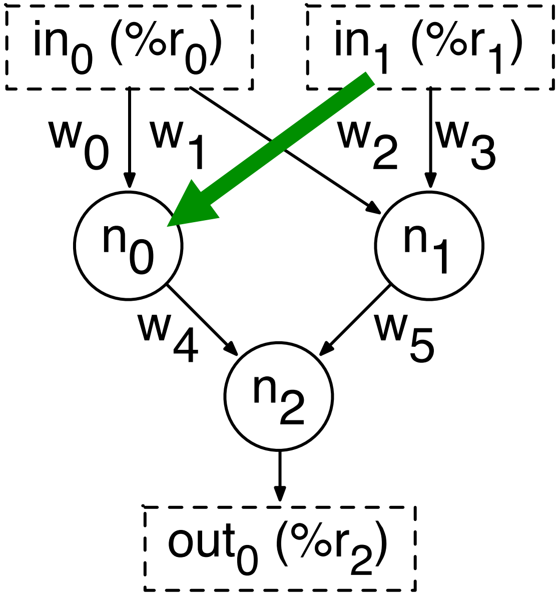
sigmoid

w_4 × (n_0, n_0, …, n_0)

+ w_5 × (n_1, n_1, …, n_1)

sigmoid

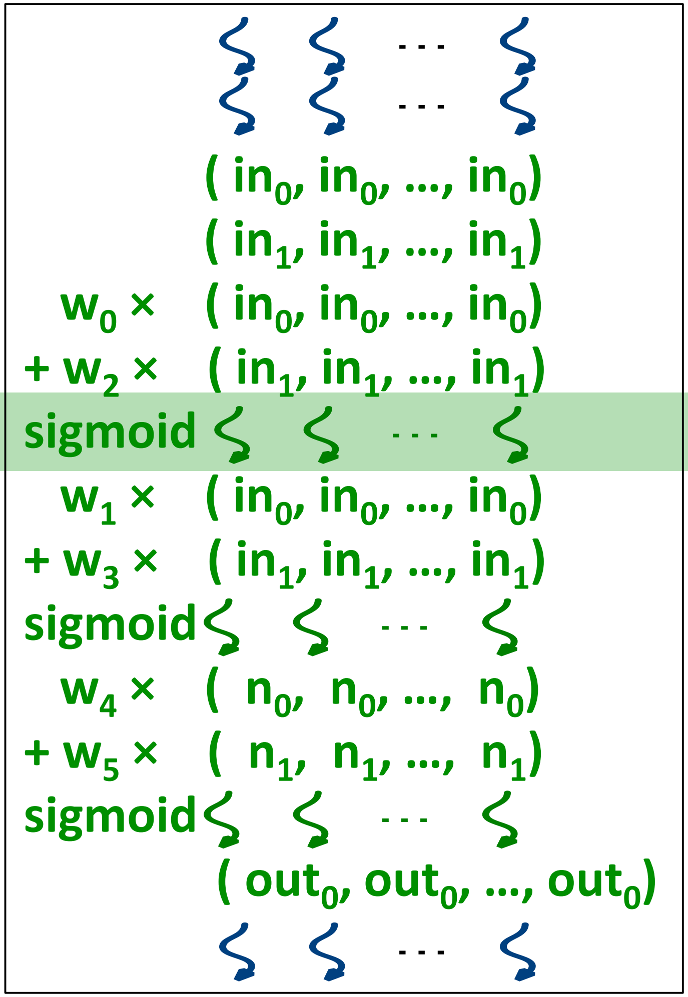
(out_0, out_0, …, out_0)



The accelerated SIMD lanes autonomously calculate the neural outputs in lock-step

NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;











st.global [addr2], %r2;
```



$(in_0, in_0, \ldots, in_0)$

$(in_1, in_1, \ldots, in_1)$

$w_0 \times (in_0, in_0, \ldots, in_0)$

$+ w_2 \times (in_1, in_1, \ldots, in_1)$

sigmoid

$w_1 \times (in_0, in_0, \ldots, in_0)$

$+ w_3 \times (in_1, in_1, \ldots, in_1)$

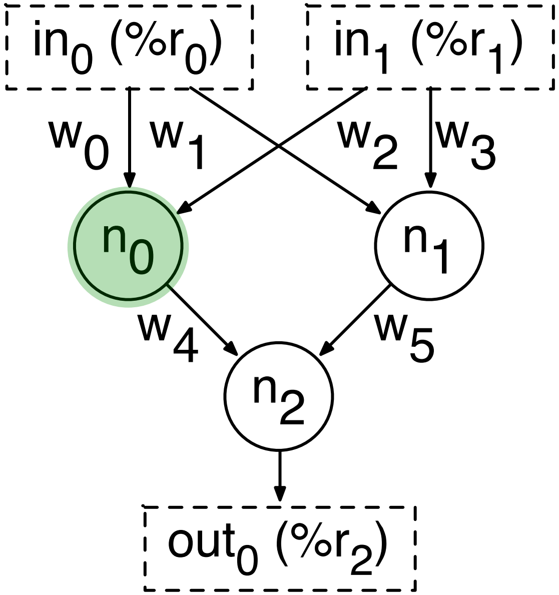
sigmoid

$w_4 \times (n_0, n_0, \ldots, n_0)$

$+ w_5 \times (n_1, n_1, \ldots, n_1)$

sigmoid

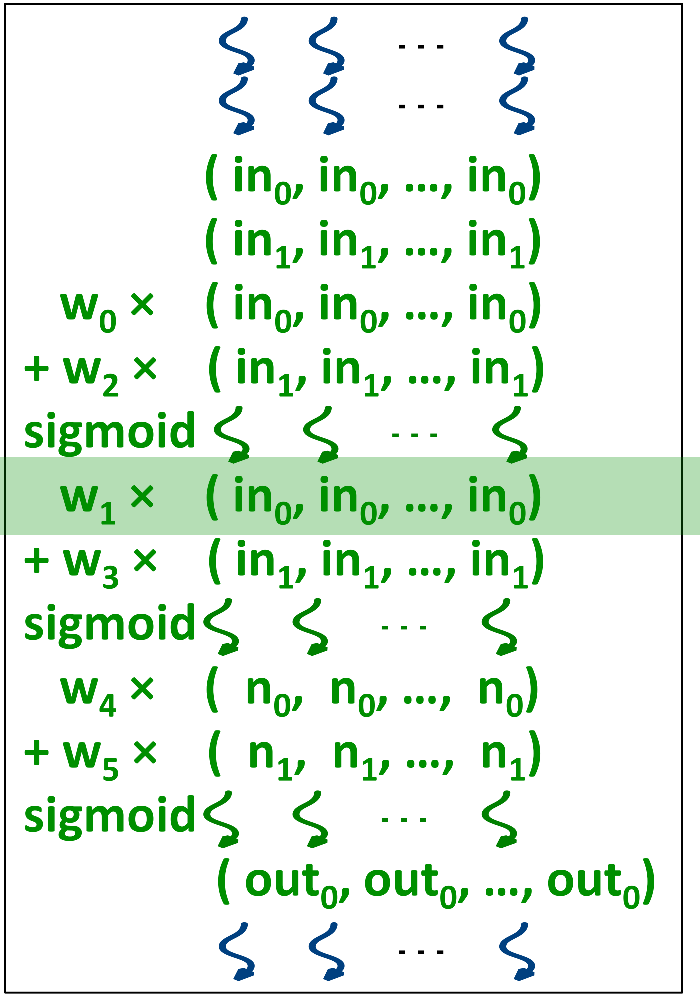
$(out_0, out_0, \ldots, out_0)$



The accelerated SIMD lanes autonomously calculate the neural outputs in lock-step

NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;











st.global [addr2], %r2;
```



$(in_0, in_0, …, in_0)$

$(in_1, in_1, …, in_1)$

$w_0 \times (in_0, in_0, …, in_0)$

$+ w_2 \times (in_1, in_1, …, in_1)$

sigmoid

$w_1 \times (in_0, in_0, …, in_0)$

$+ w_3 \times (in_1, in_1, …, in_1)$

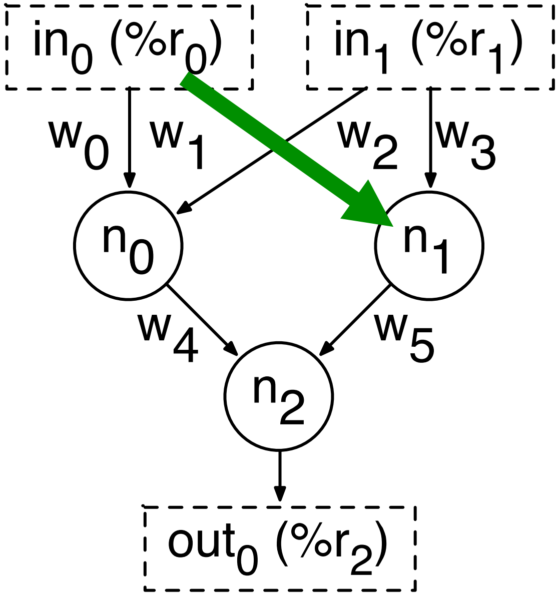
sigmoid

$w_4 \times (n_0, n_0, …, n_0)$

$+ w_5 \times (n_1, n_1, …, n_1)$

sigmoid

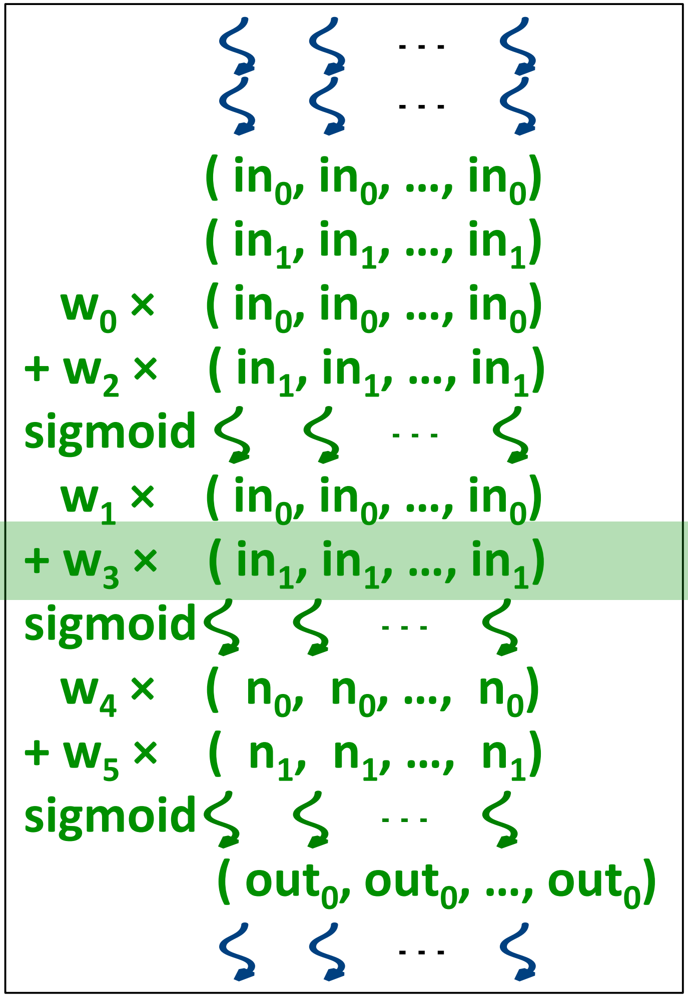
$(out_0, out_0, …, out_0)$



The accelerated SIMD lanes autonomously calculate the neural outputs in lock-step

NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;








st.global [addr2], %r2;
```



... ...

$(in_0, in_0, ..., in_0)$

$(in_1, in_1, ..., in_1)$

$w_0 \times (in_0, in_0, ..., in_0)$

$+ w_2 \times (in_1, in_1, ..., in_1)$

sigmoid ...

$w_1 \times (in_0, in_0, ..., in_0)$

$+ w_3 \times (in_1, in_1, ..., in_1)$

sigmoid ...

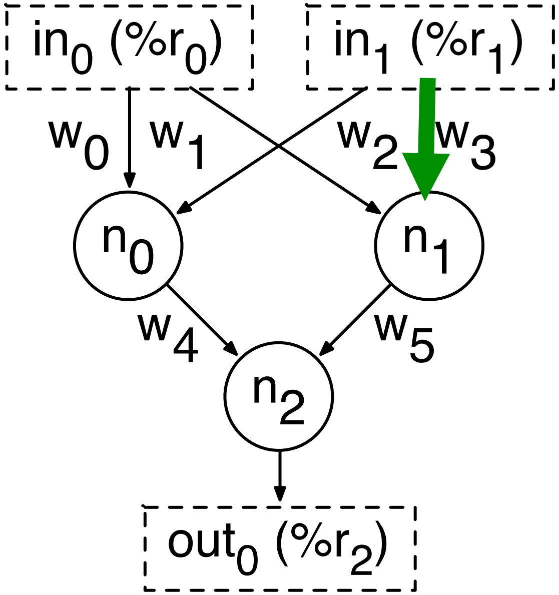
$w_4 \times (n_0, n_0, ..., n_0)$

$+ w_5 \times (n_1, n_1, ..., n_1)$

sigmoid ...

$(out_0, out_0, ..., out_0)$

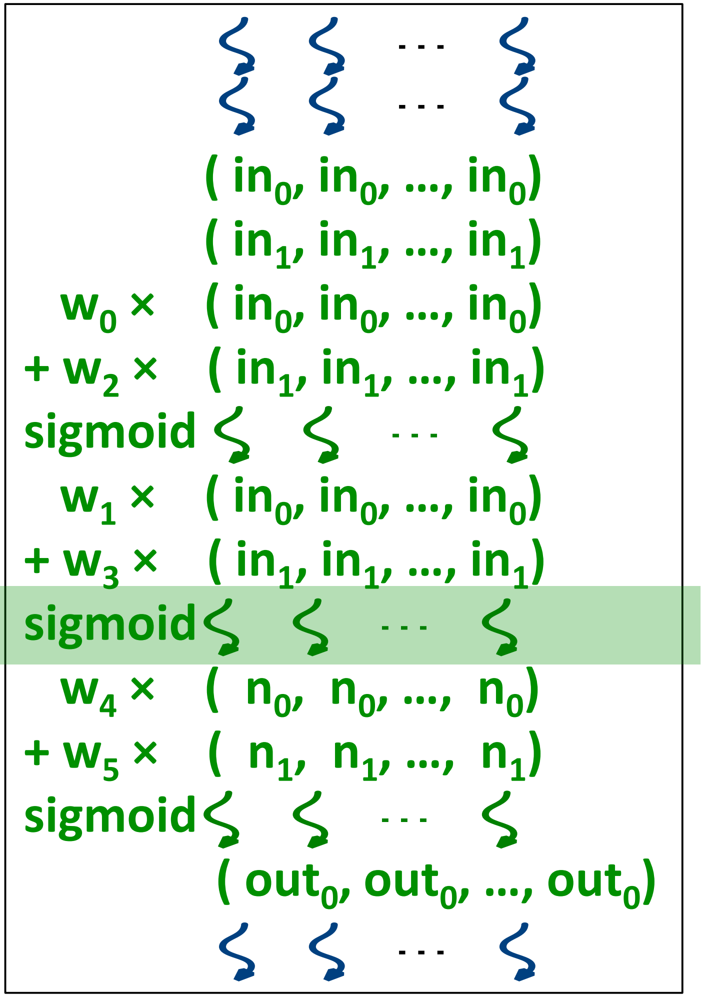
...



The accelerated SIMD lanes autonomously calculate the neural outputs in lock-step

NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;









st.global [addr2], %r2;
```



. . .

. . .

$(in_0, in_0, …, in_0)$

$(in_1, in_1, …, in_1)$

$w_0 \times (in_0, in_0, …, in_0)$

$+ w_2 \times (in_1, in_1, …, in_1)$

sigmoid . . .

$w_1 \times (in_0, in_0, …, in_0)$

$+ w_3 \times (in_1, in_1, …, in_1)$

sigmoid . . .

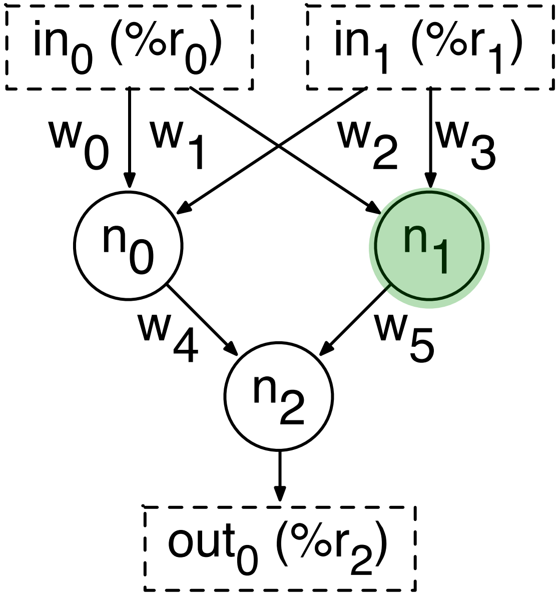
$w_4 \times (n_0, n_0, …, n_0)$

$+ w_5 \times (n_1, n_1, …, n_1)$

sigmoid . . .

$(out_0, out_0, …, out_0)$

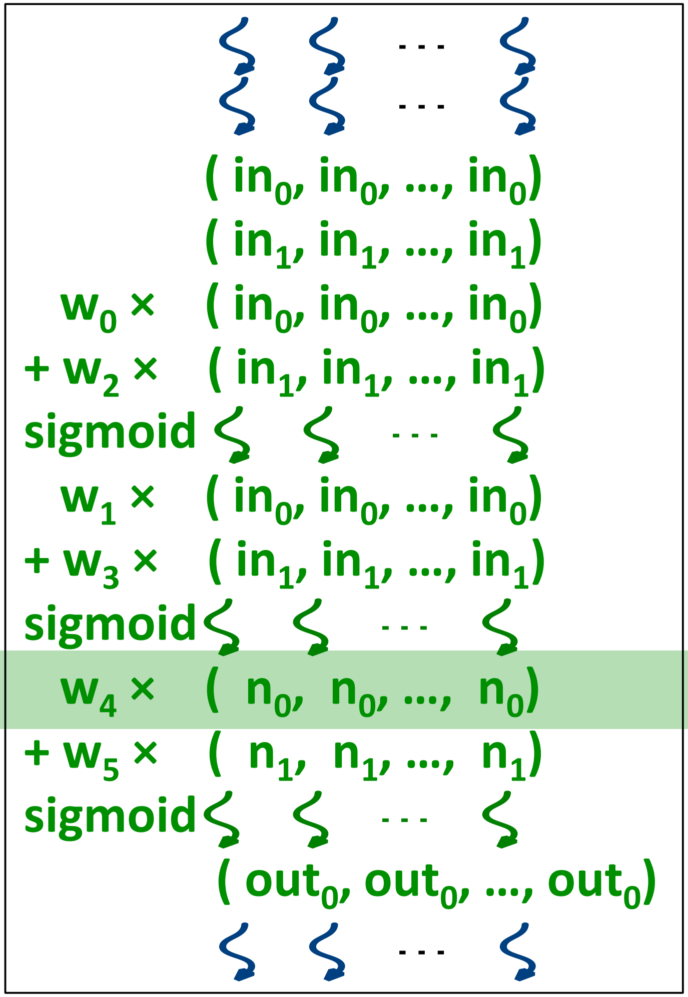
. . .



The accelerated SIMD lanes autonomously calculate the neural outputs in lock-step

NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;







st.global [addr2], %r2;
```



$(in_0, in_0, \ldots, in_0)$

$(in_1, in_1, \ldots, in_1)$

$w_0 \times (in_0, in_0, \ldots, in_0)$

$+ w_2 \times (in_1, in_1, \ldots, in_1)$

sigmoid

$w_1 \times (in_0, in_0, \ldots, in_0)$

$+ w_3 \times (in_1, in_1, \ldots, in_1)$

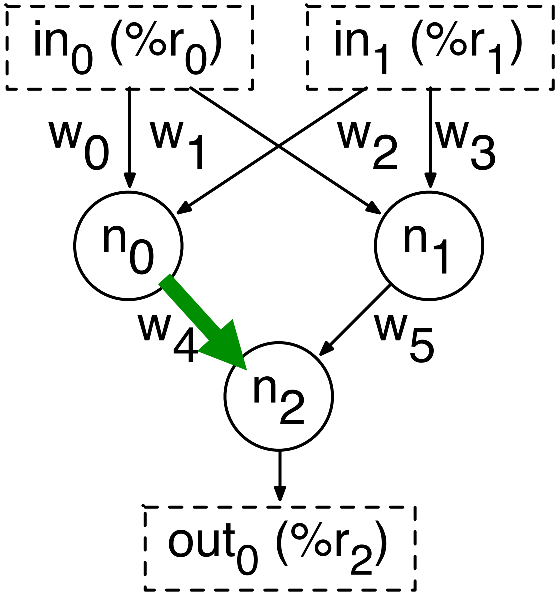
sigmoid

$w_4 \times (n_0, n_0, \ldots, n_0)$

$+ w_5 \times (n_1, n_1, \ldots, n_1)$

sigmoid

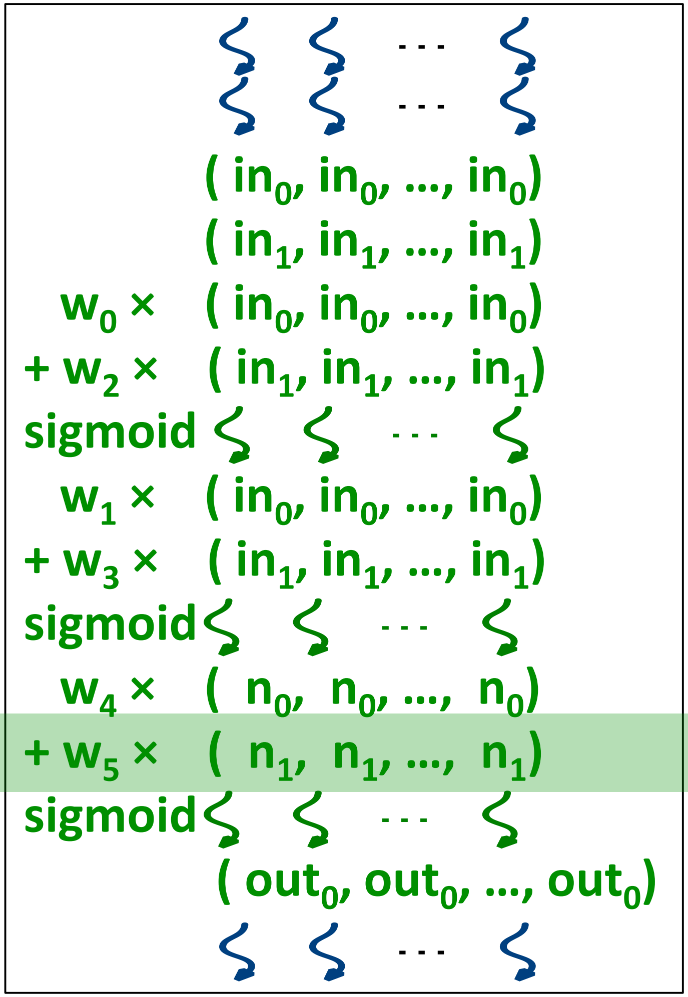
$(out_0, out_0, \ldots, out_0)$



The accelerated SIMD lanes autonomously calculate the neural outputs in lock-step

NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;









st.global [addr2], %r2;
```



$(in_0, in_0, \ldots, in_0)$

$(in_1, in_1, \ldots, in_1)$

$w_0 \times (in_0, in_0, \ldots, in_0)$

$+ w_2 \times (in_1, in_1, \ldots, in_1)$

sigmoid

$w_1 \times (in_0, in_0, \ldots, in_0)$

$+ w_3 \times (in_1, in_1, \ldots, in_1)$

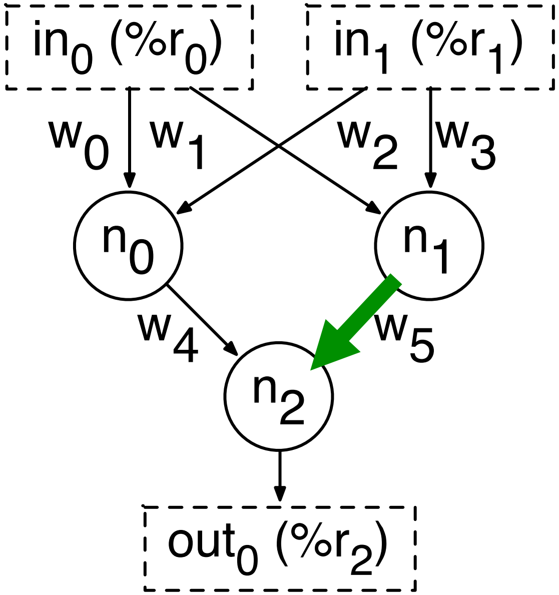
sigmoid

$w_4 \times (n_0, n_0, \ldots, n_0)$

$+ w_5 \times (n_1, n_1, \ldots, n_1)$

sigmoid

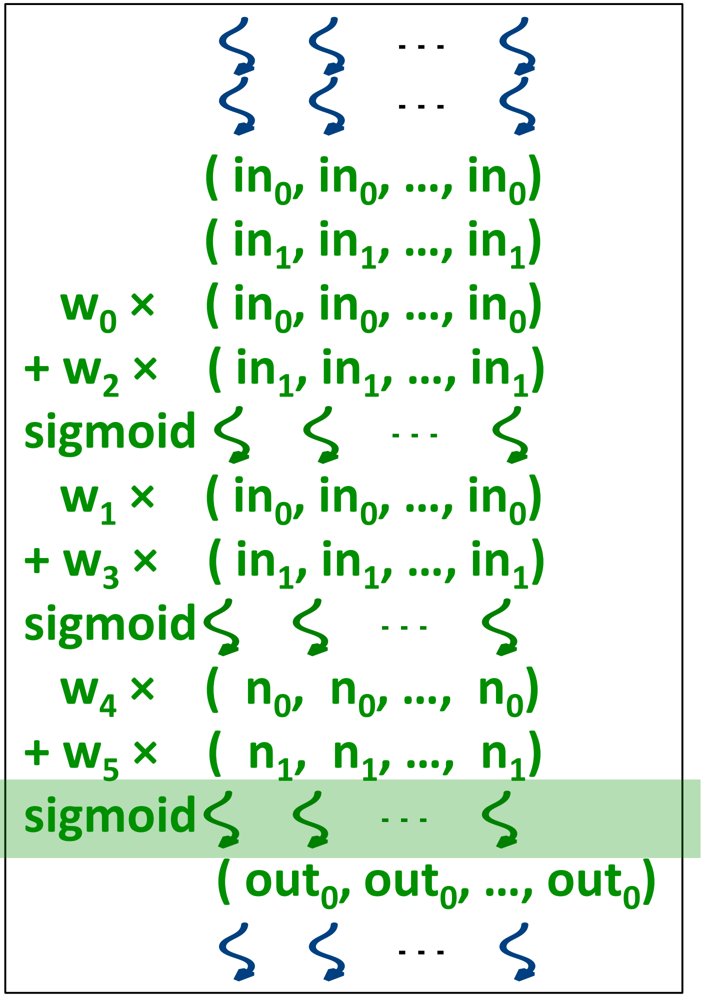
$(out_0, out_0, \ldots, out_0)$



The accelerated SIMD lanes autonomously calculate the neural outputs in lock-step

NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;









st.global [addr2], %r2;
```



$(in_0, in_0, \ldots, in_0)$

$(in_1, in_1, \ldots, in_1)$

$w_0 \times (in_0, in_0, \ldots, in_0)$

$+ w_2 \times (in_1, in_1, \ldots, in_1)$

sigmoid

$w_1 \times (in_0, in_0, \ldots, in_0)$

$+ w_3 \times (in_1, in_1, \ldots, in_1)$

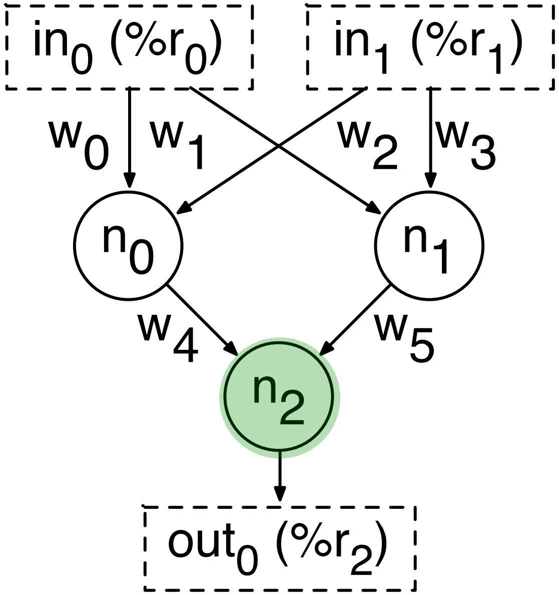
sigmoid

$w_4 \times (n_0, n_0, \ldots, n_0)$

$+ w_5 \times (n_1, n_1, \ldots, n_1)$

sigmoid

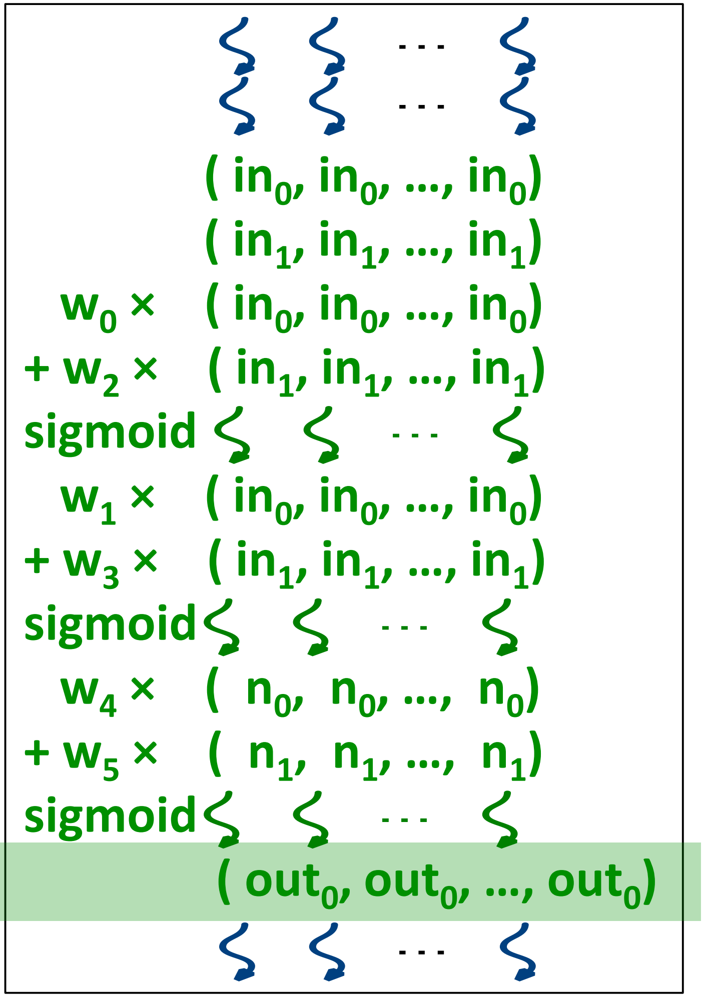
$(out_0, out_0, \ldots, out_0)$



The accelerated SIMD lanes autonomously calculate the neural outputs in lock-step

NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;

st.global [addr2], %r2;
```



$(\ in_0, in_0, \ldots, in_0)$

$(\ in_1, in_1, \ldots, in_1)$

$w_0 \times\ (\ in_0, in_0, \ldots, in_0)$

$+\ w_2 \times\ (\ in_1, in_1, \ldots, in_1)$

sigmoid

$w_1 \times\ (\ in_0, in_0, \ldots, in_0)$

$+\ w_3 \times\ (\ in_1, in_1, \ldots, in_1)$

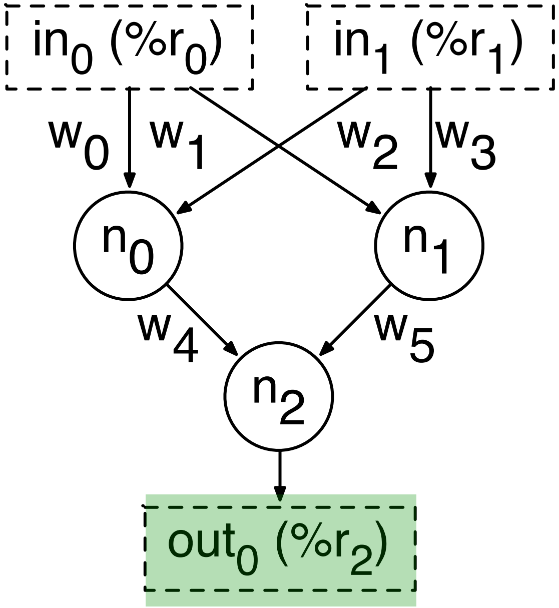
sigmoid

$w_4 \times\ (\ n_0, n_0, \ldots, n_0)$

$+\ w_5 \times\ (\ n_1, n_1, \ldots, n_1)$

sigmoid

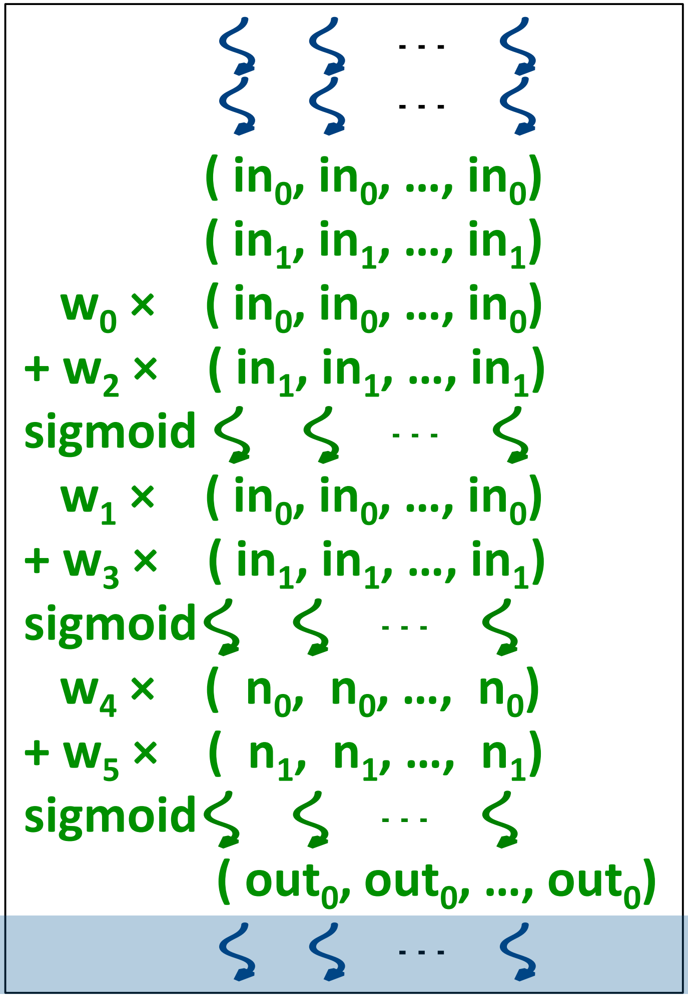
$(\ out_0, out_0, \ldots, out_0)$



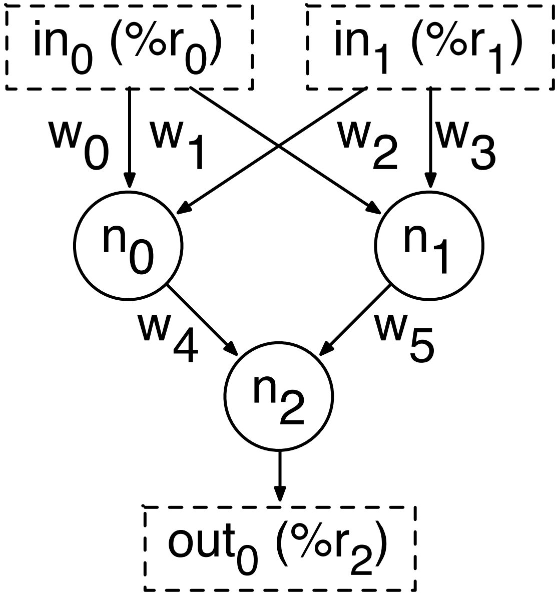
SIMD lanes exit neural mode

NGPU Execution Model

```
ld.global %r0, [addr0];
ld.global %r1, [addr1];
send.n_data %r0;
send.n_data %r1;
recv.n_data %r2;

st.global [addr2], %r2;
```



. . .
. . .
(in_0, in_0, …, in_0)
(in_1, in_1, …, in_1)
w_0 × (in_0, in_0, …, in_0)
+ w_2 × (in_1, in_1, …, in_1)
sigmoid . . .
w_1 × (in_0, in_0, …, in_0)
+ w_3 × (in_1, in_1, …, in_1)
sigmoid . . .
w_4 × (n_0, n_0, …, n_0)
+ w_5 × (n_1, n_1, …, n_1)
sigmoid . . .
(out_0, out_0, …, out_0)
. . .



SIMD lanes are in normal mode

Experimental Setup

Machine Learning, Finance, Vision
3D Gaming, Medical Imaging
Numerical Analysis, Image Processing

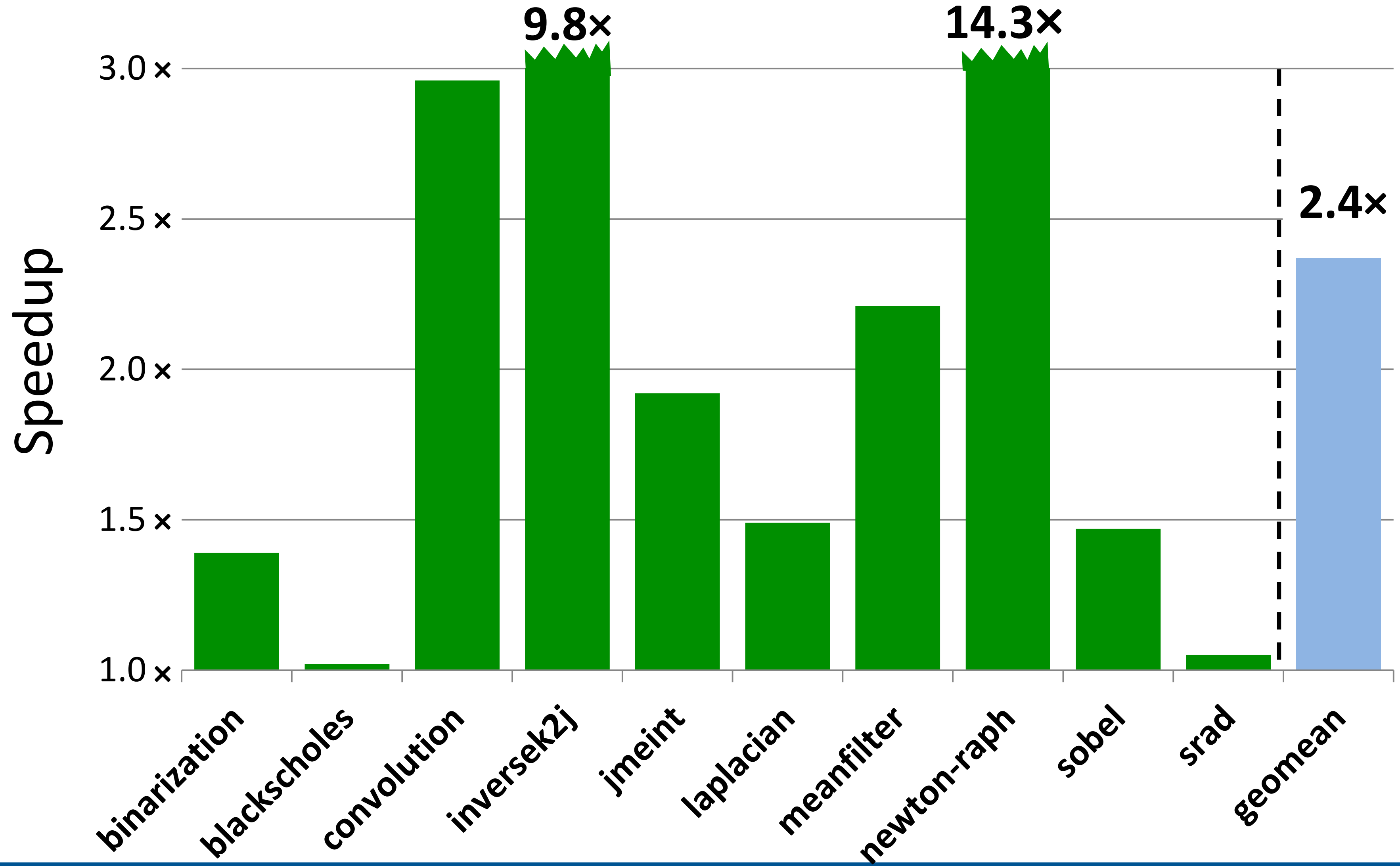
GPU Simulator

- GPGPUSim Cycle-Level Simulator
- Fermi-based GTX 480, Shader Core Frequency 1.4 GHz
- NVCC Compiler –O3

Power Model

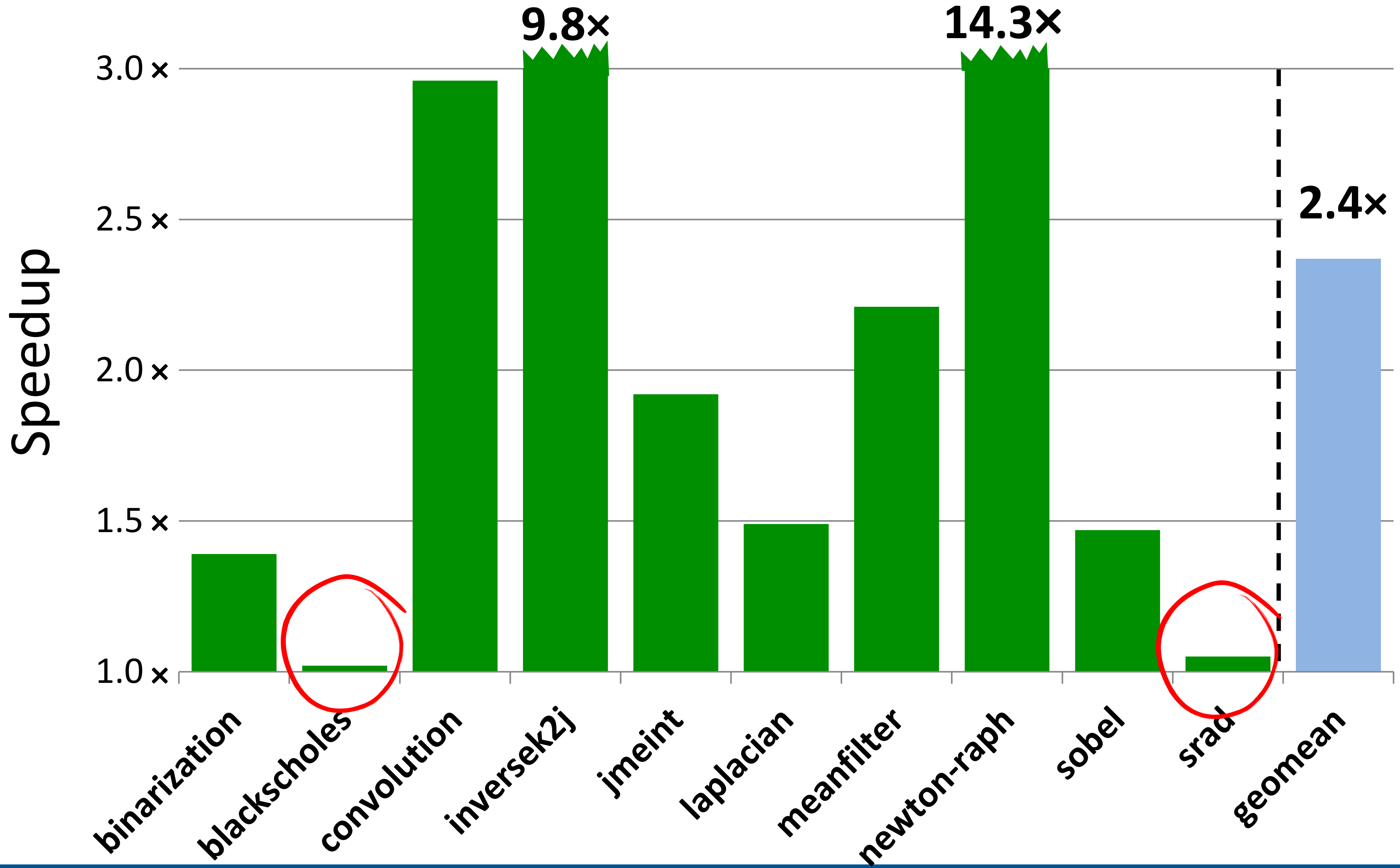
- Technology Node 40 nm
- GPUWattch
- McPAT and CACTI, Verilog

NGPU Speedup



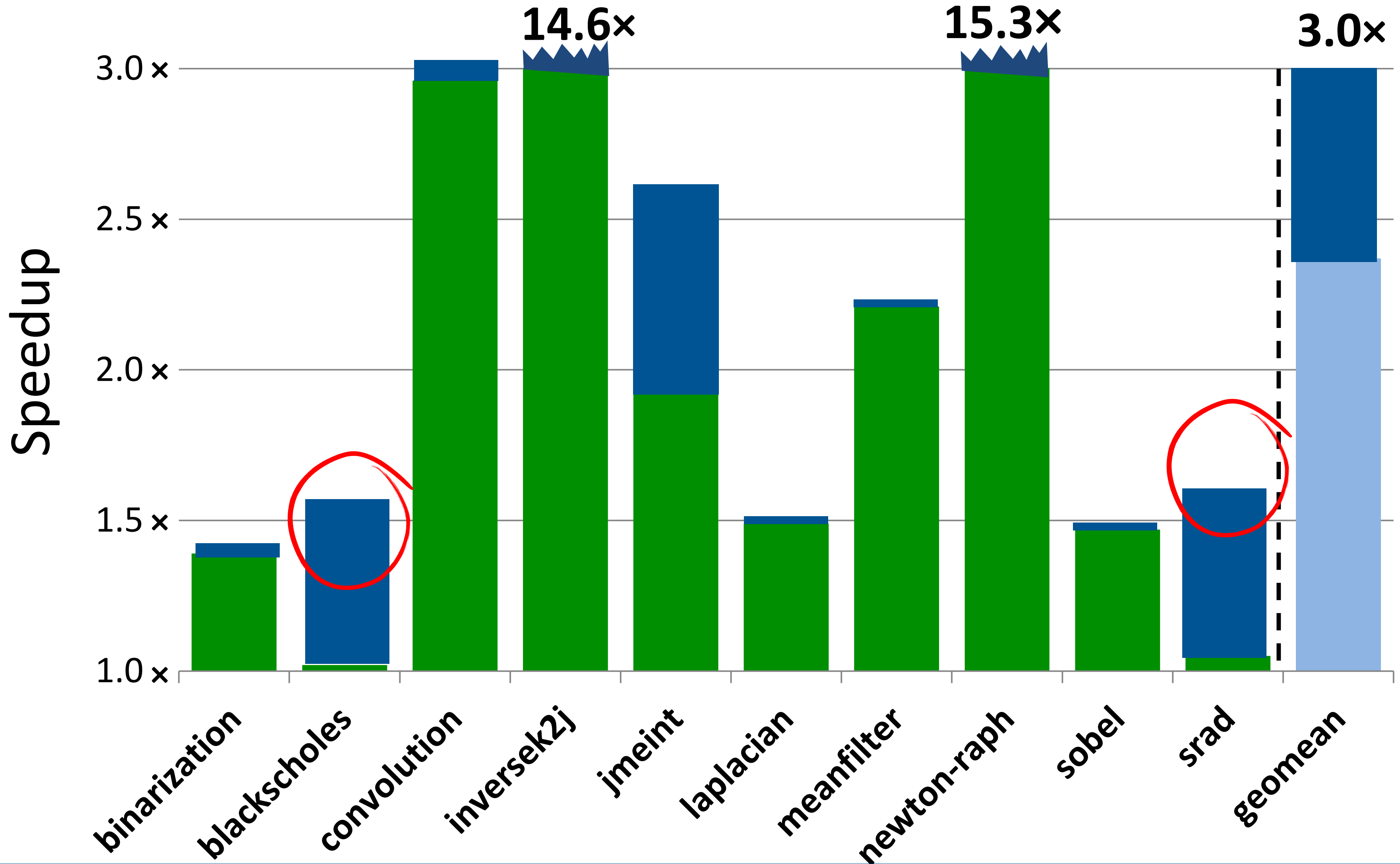
Most applications see **speedup** with **NGPU**

NGPU Speedup



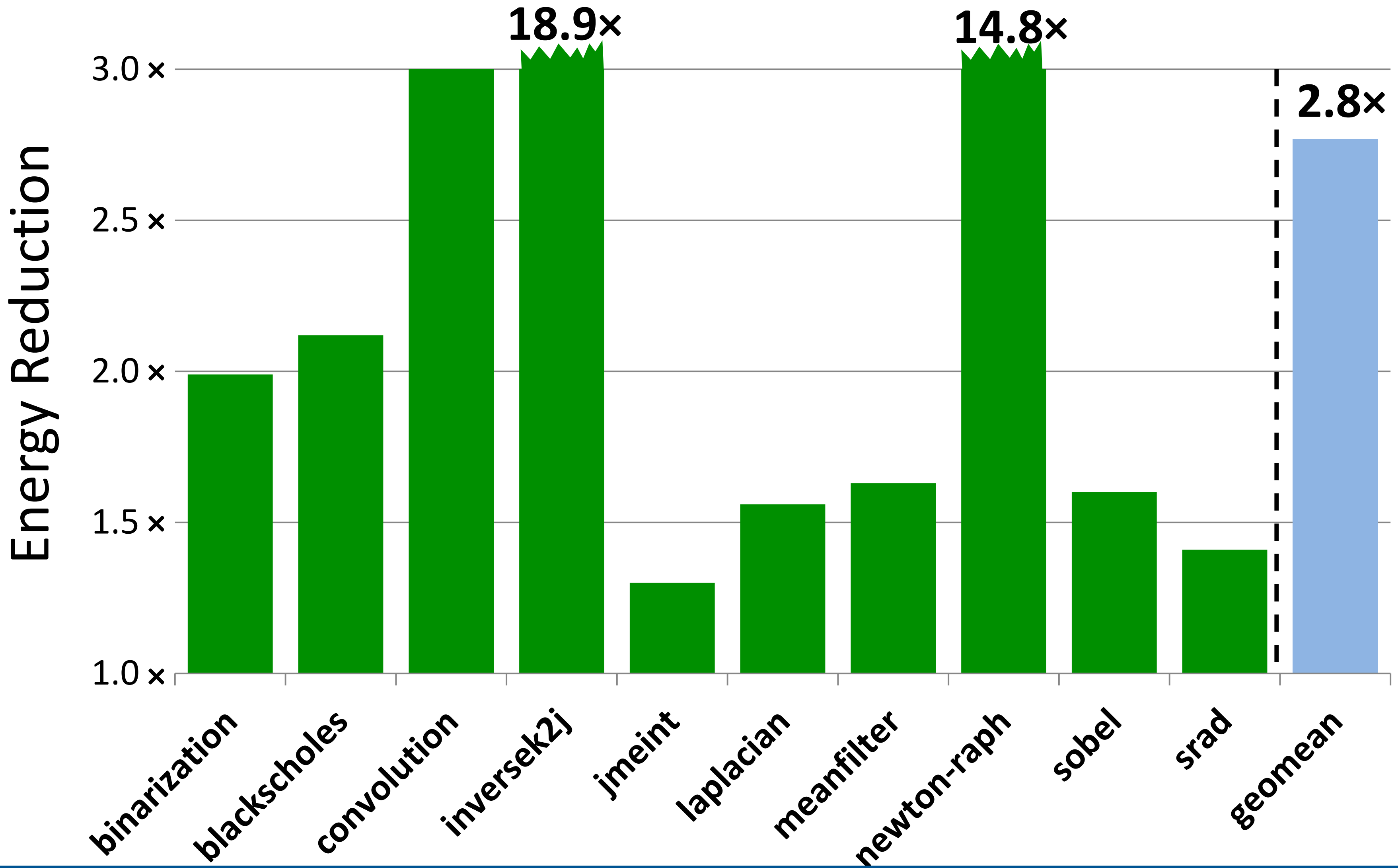
The speedup for **bandwidth-sensitive** applications is limited

NGPU Speedup with 2x Bandwidth



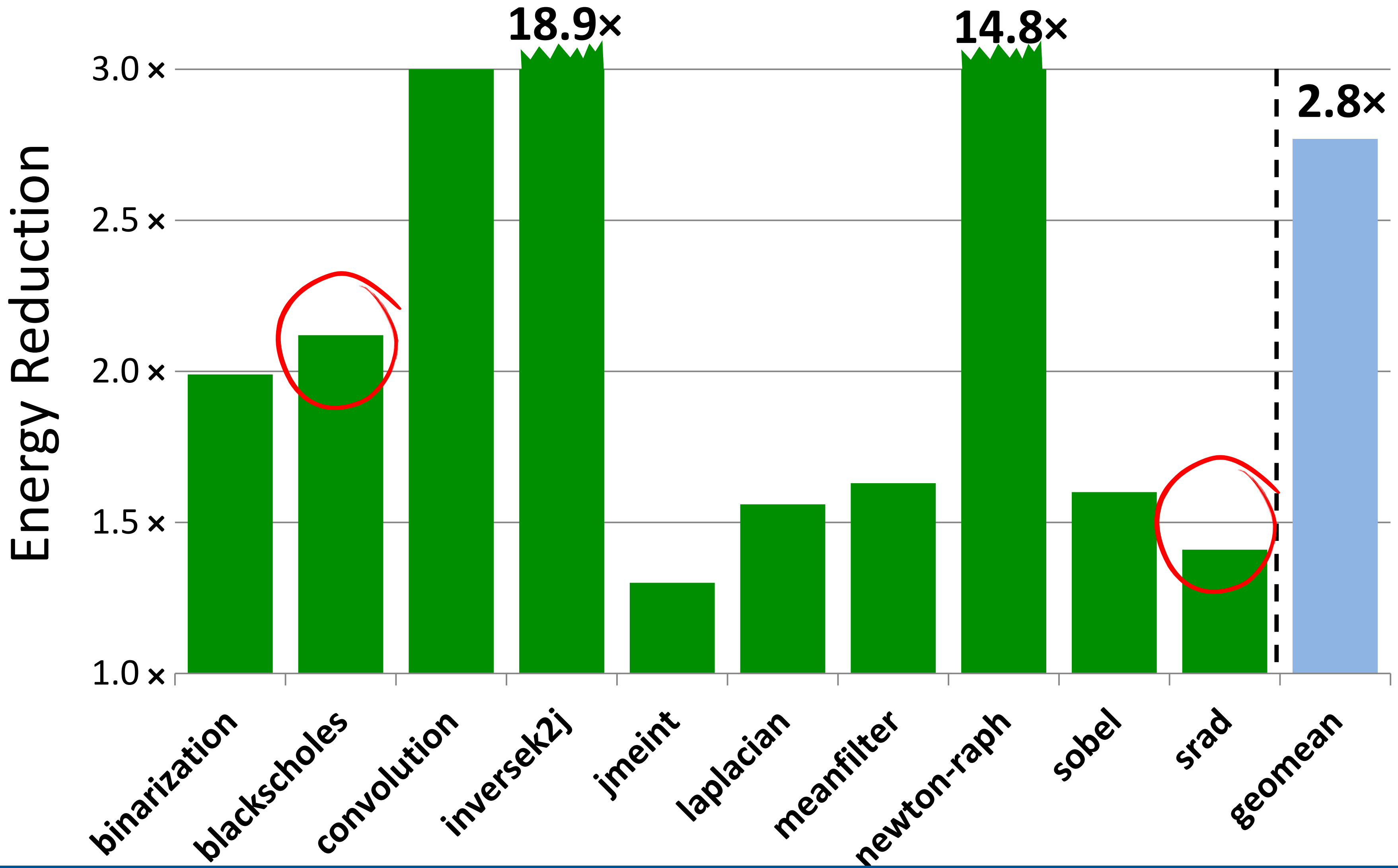
Bandwidth-sensitive applications see **speedup** with 2x bandwidth

NGPU Energy Savings with Baseline Bandwidth



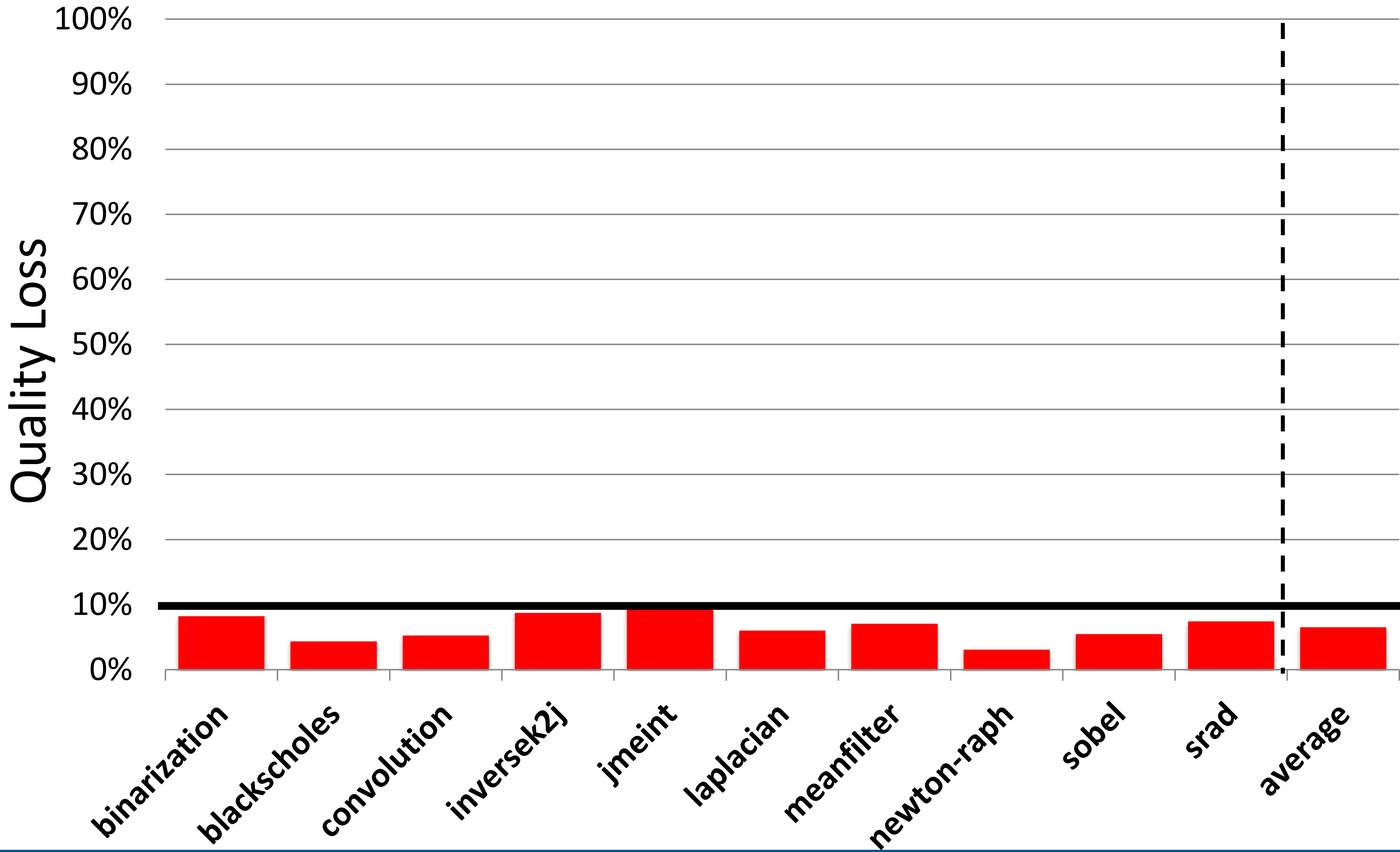
NGPU eliminates the von Neumann overhead which results in energy reduction

NGPU Energy Savings with Baseline Bandwidth



Even bandwidth-sensitive applications **see energy saving**

Application Quality Loss



Quality loss is **below 10%** in all cases

NGPU is a Fair Bargain

Overhead	**Area Overhead ≤ 1.0%**	
	Quality ≥ 97.5%	**Quality ≥ 90.0%**
Benefits	**1.9× Speedup 2.1× Energy Reduction**	**2.4× Speedup 2.8× Energy Reduction**