Accelerating CNN Inference Using SIMD Techniques

Group 40

312553049 單綿恆 312581031 張清華 31351007 陳晶

Outline

1. Abstract
2. Method
3. Experiments

Outline

1. **Abstract**
   - **Motivation**
   - **AlexNet**
2. **Method**
3. **Experiments**

Outline

1. Abstract
    - Motivation
    - AlexNet
2. Method
3. Experiments

Abstract – Motivation

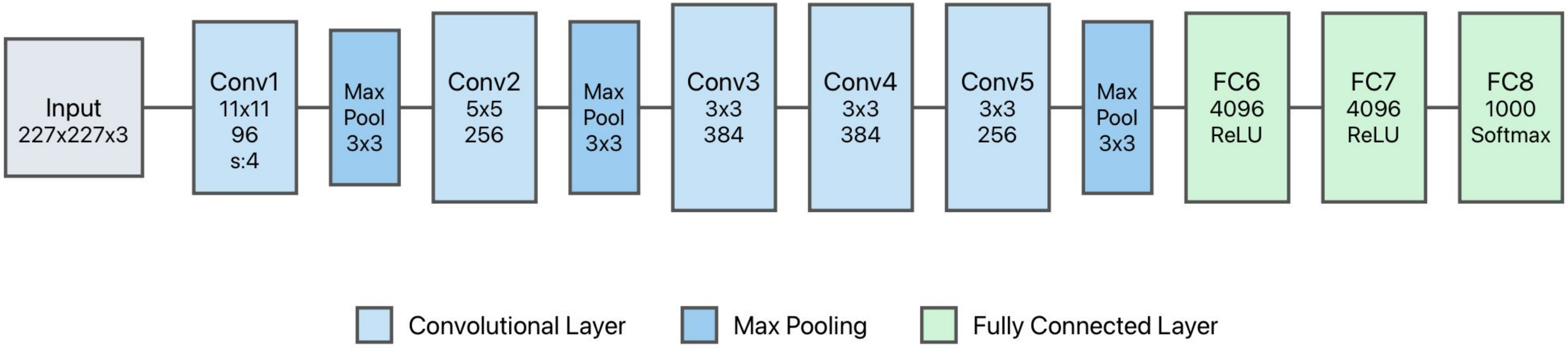
1. CNNs are **widely used** in various scenarios, with AlexNet being one of the most representative examples.
2. Small **IoT devices lack GPU** capabilities.
3. Dive into SIMD instructions for low-level optimization.
4. Apply SIMD for AI acceleration and inference optimization.

Outline

1. Abstract
    - Motivation
    - AlexNet
2. Method
3. Experiments

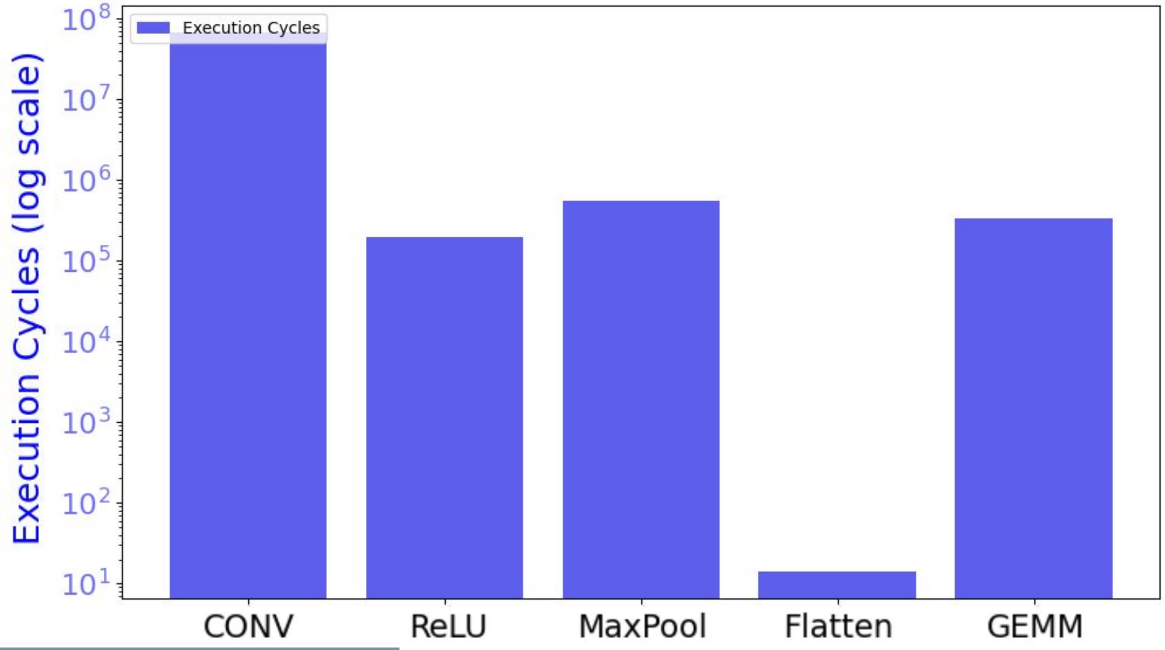
Abstract – AlexNet

Architecture



Abstract – AlexNet

Time Consumption Analysis



Outline

1. Abstract
2. Method
    - Hardware
    - Software
3. Experiments

Outline

1. Abstract
2. Method
    - Hardware
    - Software
3. Experiments

Method – Hardware

Adding SIMD ISA Extension

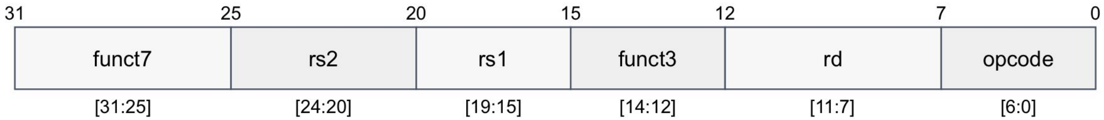
Four instruction type:

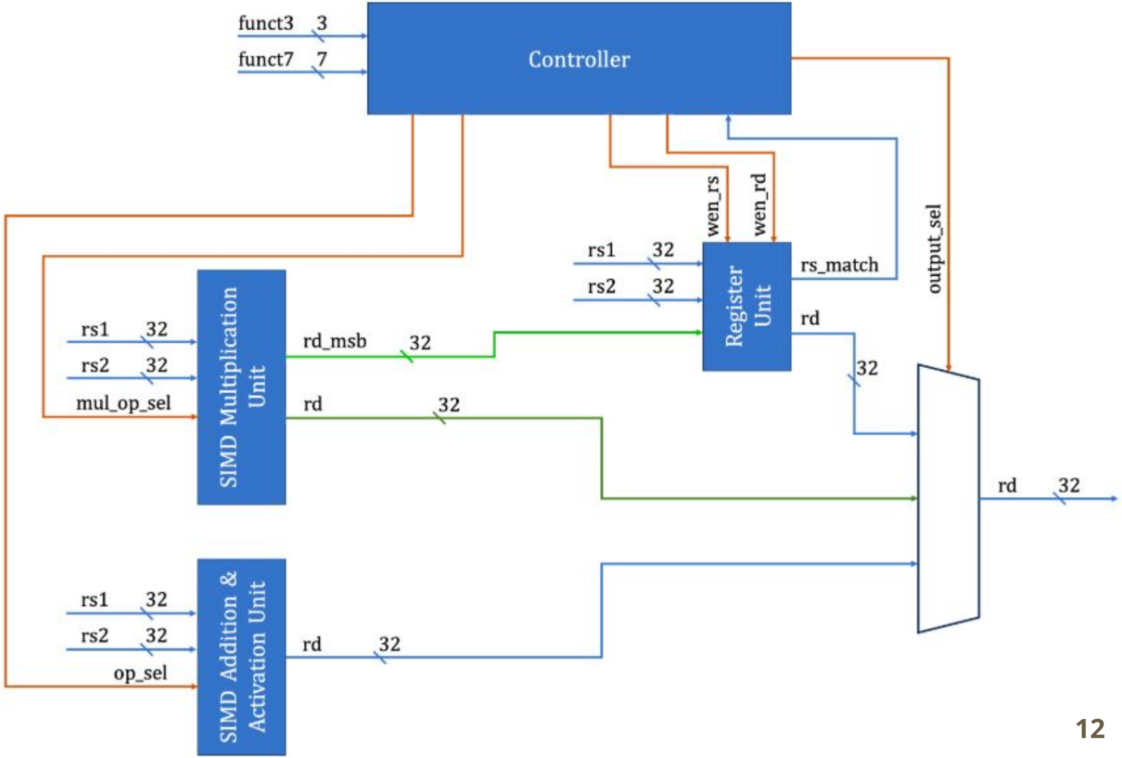
- Add
- Substract
- Multiply
- Quantize

Type	Vector-Vector	Vector-Scalar
ADD	`sADDI8I8S.vv`	`sADDI8I8S.vx`
	`sADDI16I16S.vv`	`sADDI16I16S.vx`
SUB	`sSUBI8I8S.vv`	`sSUBI8I8S.vx`
	`sSUBI16I16S.vv`	`sSUBI16I16S.vx`
PMUL	`sPMULI8I16S.vv.L`	`sPMULI8I16S.vx.L`
	`sPMULI8I16S.vv.H`	`sPMULI8I16S.vx.H`
AMUL	`sAMULI8I8S.vv.NQ`	`sAMULI8I8S.vx.NQ`
	`sAMULI8I8S.vv.AQ`	`sAMULI8I8S.vx.AQ`
QNT	`sQNTI16I8S.vv.NQ`	
	`sQNTI16I8S.vv.AQ`	

Method — Hardware

Custom Function Unit — SIMD Execution Engine

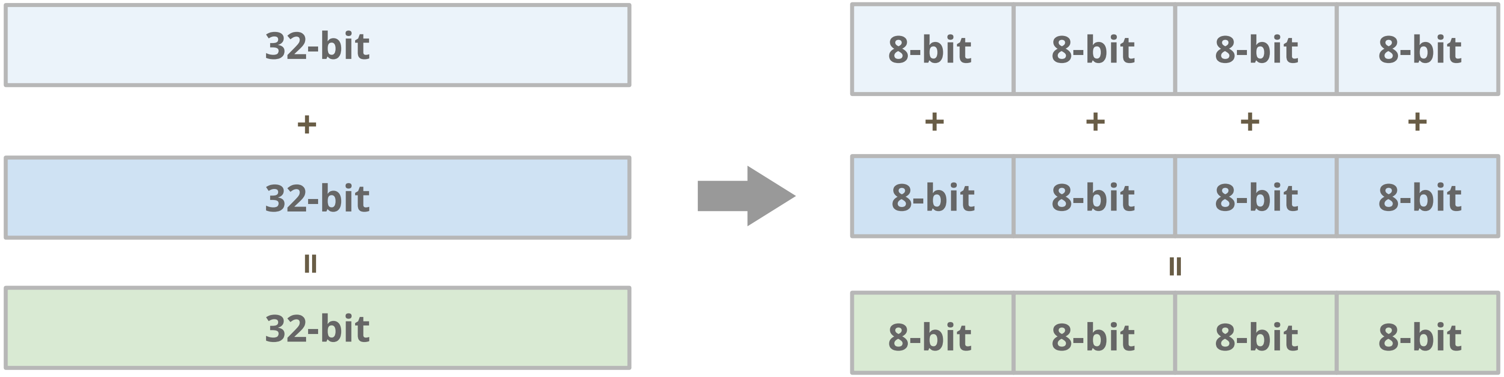




Method – Hardware

31 25 20 15 12 7 0

	funct7	rs2	rs1	funct3	rd	opcode
	[31:25]	[24:20]	[19:15]	[14:12]	[11:7]	[6:0]
s**ADD**I8I8S.vx	7b1000000			3b000		SIMD
s**ADD**I16I16S.vx	7b1000000			3b001		SIMD



Method – Hardware

ALU Implementation

```
// 8-bit wire assignment
for (i <- 0 until 4) {
    rs1ByteArray(i) := io.rs1(8 * i + 7, 8 * i)
    rs2ByteArray(i) := io.rs2(8 * i + 7, 8 * i)

    rdByteArray(i) := MuxLookup(
        io.opSel.asUInt,
        DontCare,
        Seq(
            AddSubActivationOp.ADDI8I8S_VV.asUInt -> (rs1ByteArray(i).asSInt + rs2ByteArray(i).asSInt).asUInt,
            AddSubActivationOp.SUBI8I8S_VV.asUInt -> (rs1ByteArray(i).asSInt - rs2ByteArray(i).asSInt).asUInt,
            AddSubActivationOp.ADDI8I8S_VX.asUInt -> (rs1ByteArray(i).asSInt + rs2ByteArray(0).asSInt).asUInt,
            AddSubActivationOp.SUBI8I8S_VX.asUInt -> (rs1ByteArray(i).asSInt - rs2ByteArray(0).asSInt).asUInt,
            AddSubActivationOp.SCMPLE8.asUInt -> Mux(
                rs1ByteArray(i).asSInt <= rs2ByteArray(i).asSInt,
                rs2ByteArray(i),
                rs1ByteArray(i)
            )
        )
    )
}
```

Outline

1. Abstract
2. Method
    - Hardware
    - Software
3. Experiments

Method – Software

Convolution Implementation

```
void Conv::execPerLayerNaiveQuant() {
    int16_t tempINT16_Buffer[100000] = {0};
    int16_t temp_C[4] = {0}, temp_D[4] = {0};
    int8_t temp_A[4] = {0}, temp_B[4] = {0}, temp_E[4] = {0};

    for (int n = 0; n < info->kernel.N; n++) {
        for (int oh = 0; oh < output->H; oh++) {
            for (int ow = 0; ow < output->W; ow++) {
                output->data[n * output->H * output->W + oh * output->W + ow] = info->bias.data[n];
                for (int c = 0; c < info->kernel.C; c++) {
                    for (int kh = 0; kh < info->kernel.H; kh++) {
                        for (int kw = 0; kw < info->kernel.W; kw += 4) {
                            // ... (rest of the code is cut off)
                        }
                    }
                }
            }
        }
    }
}
```

Method – Software

Convolution Implementation

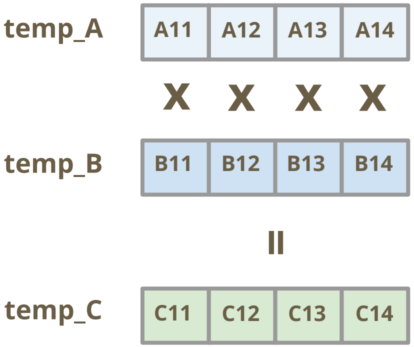
temp_A	A11	A12	A13	A14
temp_B	B11	B12	B13	B14

```
for (int kw = 0; kw < info->kernel.W; kw+=4) {
    for (int i = 0; i < 4; i++) {
        if (kw + i >= info->kernel.W) {
            temp_A[i] = 0;
            temp_B[i] = 0;
        }else{
            temp_A[i] = input->data[c * input->H * input->W
                                    + (oh + kh) * input->W
                                    + (ow + kw + i)];
            temp_B[i] = info->kernel.data[((n * info->kernel.C + c) * info->kernel.H + kh) * info->kernel.W
                                          + kw
                                          + i];
        }
    }
    sPMULI8I16S_vv_L(temp_C, temp_A, temp_B);
    sPMULI8I16S_vv_H(temp_C+2, temp_A, temp_B);
    for (int i = 0; i < 4; i++)
        tempINT16_Buffer[n * output->H * output->W + oh * output->W + ow] += temp_C[i];

}
```

Method – Software

Convolution Implementation

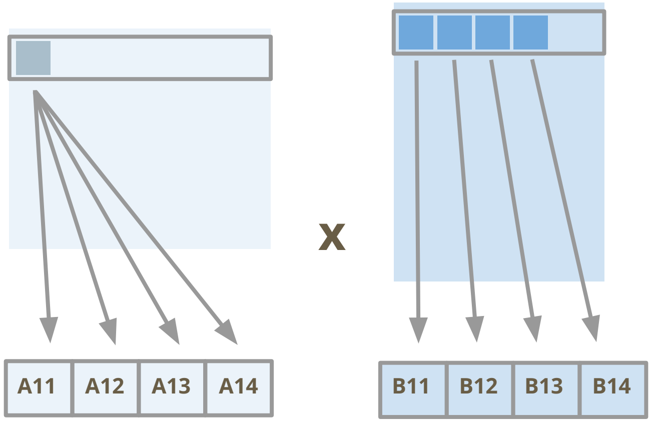



```
for (int kw = 0; kw < info->kernel.W; kw+=4) {
    for (int i = 0; i < 4; i++) {
        if (kw + i >= info->kernel.W) {
            temp_A[i] = 0;
            temp_B[i] = 0;
        }else{
            temp_A[i] = input->data[c * input->H * input->W
                                    + (oh + kh) * input->W
                                    + (ow + kw + i)];
            temp_B[i] = info->kernel.data[((n * info->kernel.C + c) * info->kernel.H + kh) * info->kernel.W
                                          + kw
                                          + i];
        }
    }
    sPMULI8I16S_vv_L(temp_C, temp_A, temp_B);
    sPMULI8I16S_vv_H(temp_C+2, temp_A, temp_B);
    for (int i = 0; i < 4; i++)
        tempINT16_Buffer[n * output->H * output->W + oh * output->W + ow] += temp_C[i];

}
```

Method – Software

GEMM Implementation



temp_A

temp_B

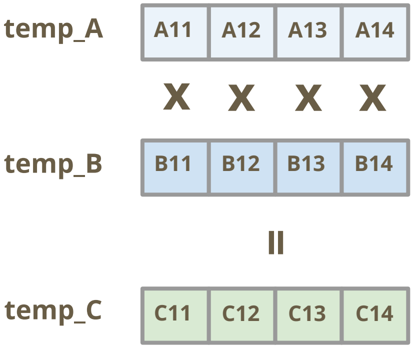
```
void Gemm::execPerLayerAdvanceQuant() {
    int16_t tempINT16_Buffer[100000] = {0};
    int index_A, index_B, index_C;
    int8_t temp_A[4] = {0}, temp_B[4] = {0};
    int16_t temp_C[4] = {0}, temp_D[4] = {0};

    for (int m = 0; m < input->H; m++) {
        index_A = m * input->W;    // M * K
        index_C = m * output->W;   // M * N
        for (int k = 0; k < input->W; k++) {
            index_B = k * info->weight.W;   // K * N
            for (int n = 0; n < info->weight.W; n += 4) {
                for (int i = 0; i < 4; ++i){
                    if (n + i >= info->weight.W)
                        temp_A[i] = 0;
                    else
                        temp_A[i] = input->data[index_A + k];
                }
                *(int32_t*)temp_B = *(int32_t*)&(info->weight.data)[index_B + n];
                int output_index = index_C + n;
                sPMULI8I16S_vv_L(temp_C, temp_A, temp_B);
                sPMULI8I16S_vv_H(temp_C + 2, temp_A, temp_B);
                sADDI16I16S_vv(tempINT16_Buffer + output_index,
                               tempINT16_Buffer + output_index,
                               temp_C);
                sADDI16I16S_vv(tempINT16_Buffer + output_index + 2,
                               tempINT16_Buffer + output_index + 2,
                               temp_C + 2);

            }
        }
        for (int n = 0; n < info->weight.W; n++)
            tempINT16_Buffer[index_C + n] += info->bias.data[index_C + n];
    }
}
```

Method – Software

GEMM Implementation




```
void Gemm::execPerLayerAdvanceQuant() {
    int16_t tempINT16_Buffer[100000] = {0};
    int index_A, index_B, index_C;
    int8_t temp_A[4] = {0}, temp_B[4] = {0};
    int16_t temp_C[4] = {0}, temp_D[4] = {0};

    for (int m = 0; m < input->H; m++) {
        index_A = m * input->W;    // M * K
        index_C = m * output->W;   // M * N
        for (int k = 0; k < input->W; k++) {
            index_B = k * info->weight.W;   // K * N
            for (int n = 0; n < info->weight.W; n += 4) {
                for (int i = 0; i < 4; ++i){
                    if (n + i >= info->weight.W)
                        temp_A[i] = 0;
                    else
                        temp_A[i] = input->data[index_A + k];
                }
                *(int32_t*)temp_B = *(int32_t*)&(info->weight.data)[index_B + n];
                int output_index = index_C + n;
                sPMULI8I16S_vv_L(temp_C, temp_A, temp_B);
                sPMULI8I16S_vv_H(temp_C + 2, temp_A, temp_B);
                sADDI16I16S_vv(tempINT16_Buffer + output_index,
                               tempINT16_Buffer + output_index,
                               temp_C);
                sADDI16I16S_vv(tempINT16_Buffer + output_index + 2,
                               tempINT16_Buffer + output_index + 2,
                               temp_C + 2);
            }
        }
        for (int n = 0; n < info->weight.W; n++)
            tempINT16_Buffer[index_C + n] += info->bias.data[index_C + n];
    }
}
```

Outline

1. Abstract
2. Method
3. Experiments
    - Environment
    - Serial vs SIMD
    - Summary

Outline

1. Abstract
2. Method
3. Experiments
    - Environment
    - Serial vs SIMD
    - Summary

Experiments – Enviroments

CFU (Custom Function Unit) Playground

System-on-Chip (SoC) equipped with a 32-bit RISC-V CPU + custom function unit

Tools

HDL: Chisel3

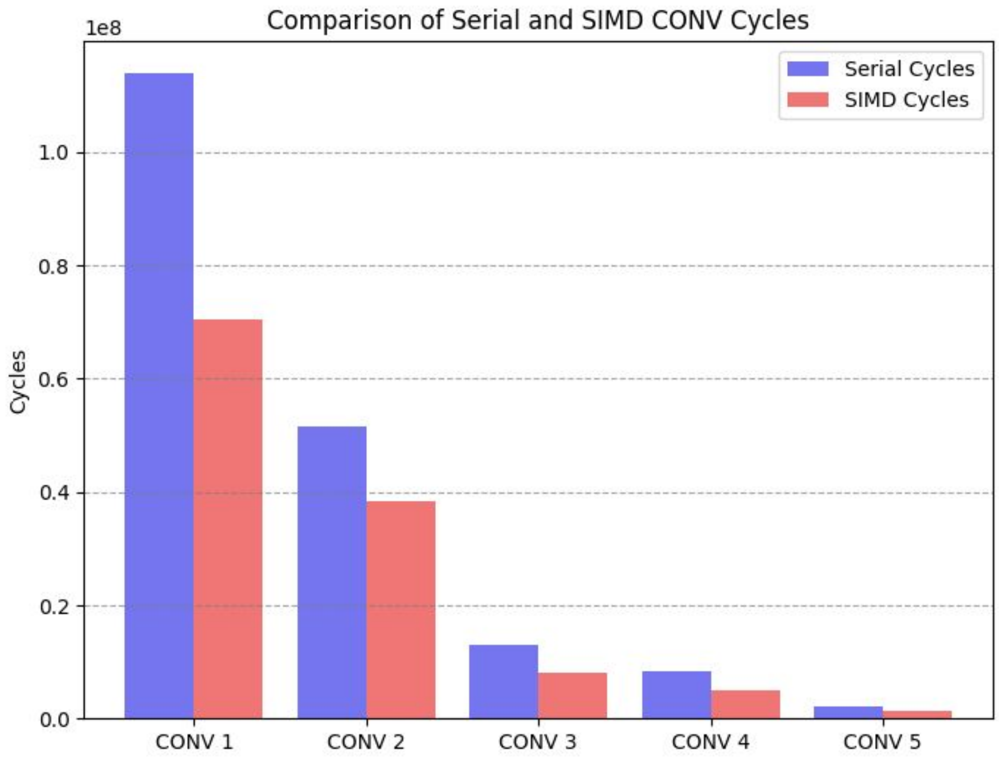
SW: C/C++

Outline

1. Abstract
2. Method
3. **Experiments**
    - Environment
    - **Serial vs SIMD**
    - Summary

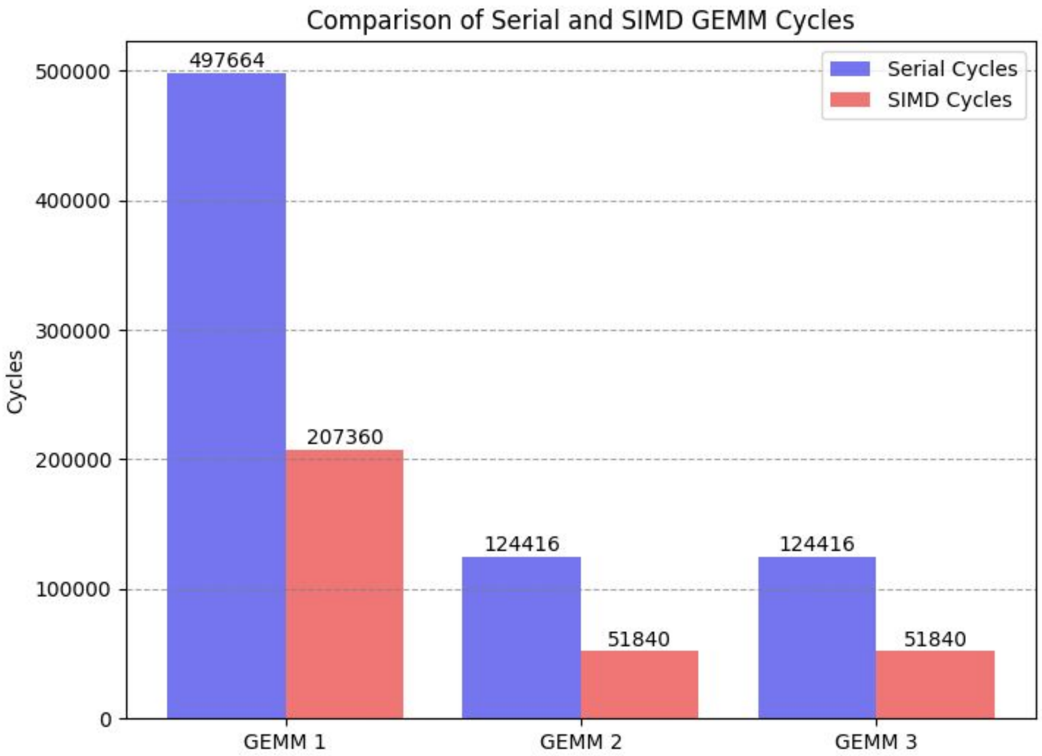
Experiments – Serial vs SIMD

Convolution



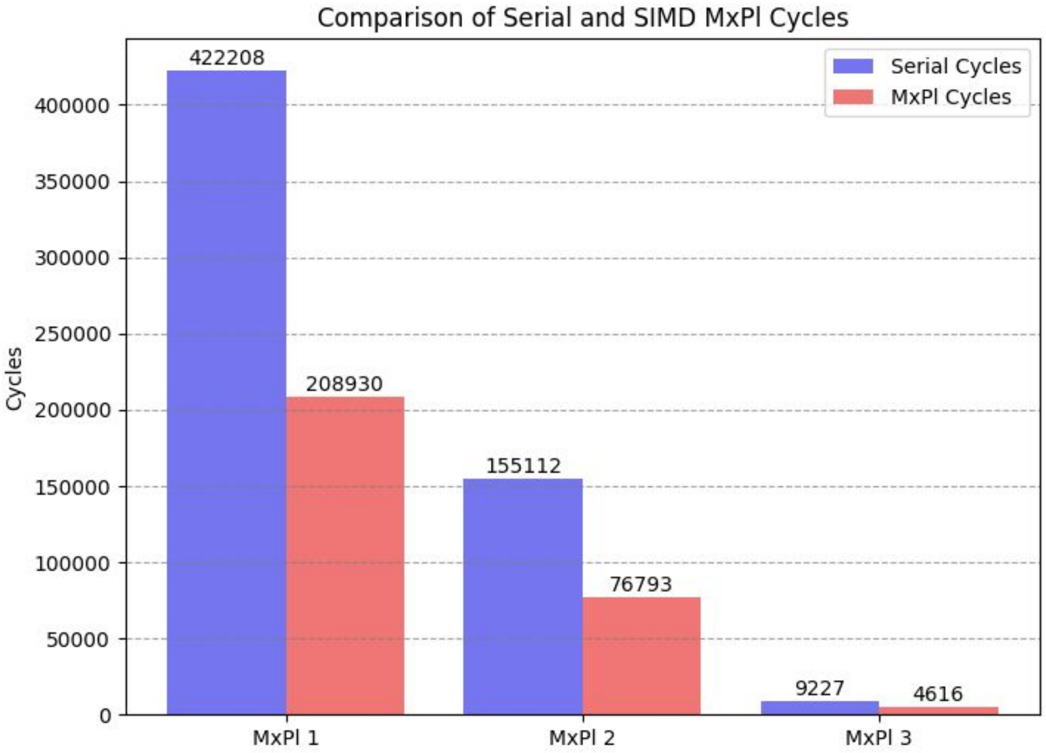
Experiments – Serial vs SIMD

GEMM



Experiments – Serial vs SIMD

Max Pooling



Outline

1. Abstract
2. Method
3. Experiments
    - Environment
    - Serial vs SIMD
    - Summary

Experiments – Summary

	Serial	SIMD	SpeedUp
CONV	188M	123M	**1.53**
GEMM	746k	311k	**2.4**
MxPl	586k	290k	**2.02**

Thanks