ConvBench

A Comprehensive Convolution Performance Evaluation Benchmark

Lucas F. A. e Silva <lucas.silva@ic.unicamp.br>

Computer Systems Laboratory (LSC)
Institute of Computing (IC)
University of Campinas (UNICAMP)

February, 2024

Table of Contents

SECTION

Overview

Overview

- Main operation on most DL models
  - Largest share of a CNN execution (~90%).
  - Convolution is a computationally expensive operation.
- Different Convolution Algorithms have been developed:
  - Naive
  - IM2COL transformation + GEMM
  - Winograd
  - Direct Convolution: SConv, YAConv, and so on...

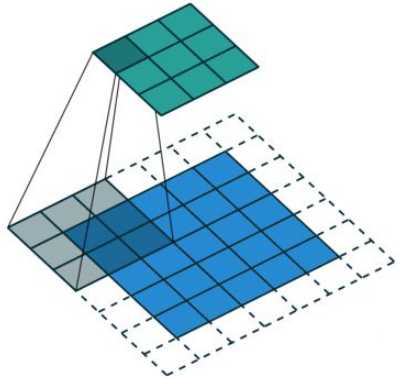


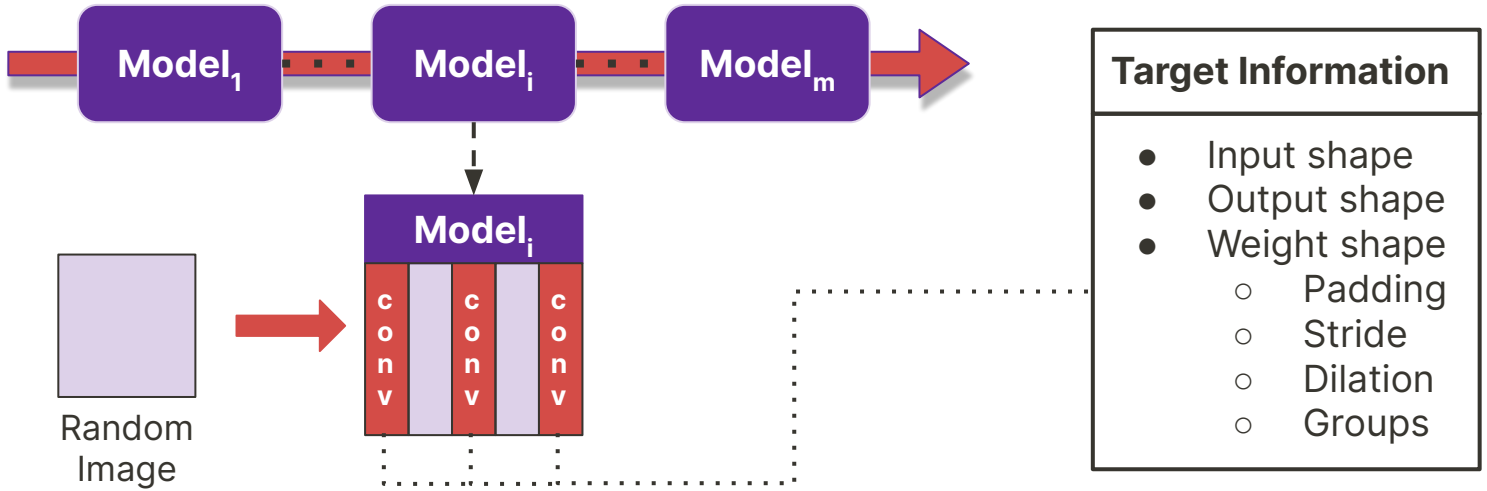
Figure: Convolution Animation, obtained from: https://github.com/vdumoulin/conv_arithmetic

SECTION

Operation Set Construction and Filtering

Operation set construction

- 1st step of the Convolution Benchmark: **Operation Set Construction**
- Hugging Face TIMM's DL model collection – 1017 models
  - CNNs, ViTs, and so on.



Operation set construction

- Unique keys identifying each convolutions
- **9011 different convolution operations (December/2023).**
  - 5481 elementwise convolutions.
  - 3530 not elementwise.
    - 2269 grouped convolution
    - 17 dilated convolution
    - 93 rectangular convolution (only filters).
    - 2 rectangular convolution (filter and input)
    - 1149 regular convolution

Operation set construction

Operation Set Description:

- Input spatial size: **4×4 up to 1024×1024**
- Output spatial size: **1×1 up to 400×400**
- Channels: **3 up to 2048**
- Kernel number of filters: **8 up to 4096**

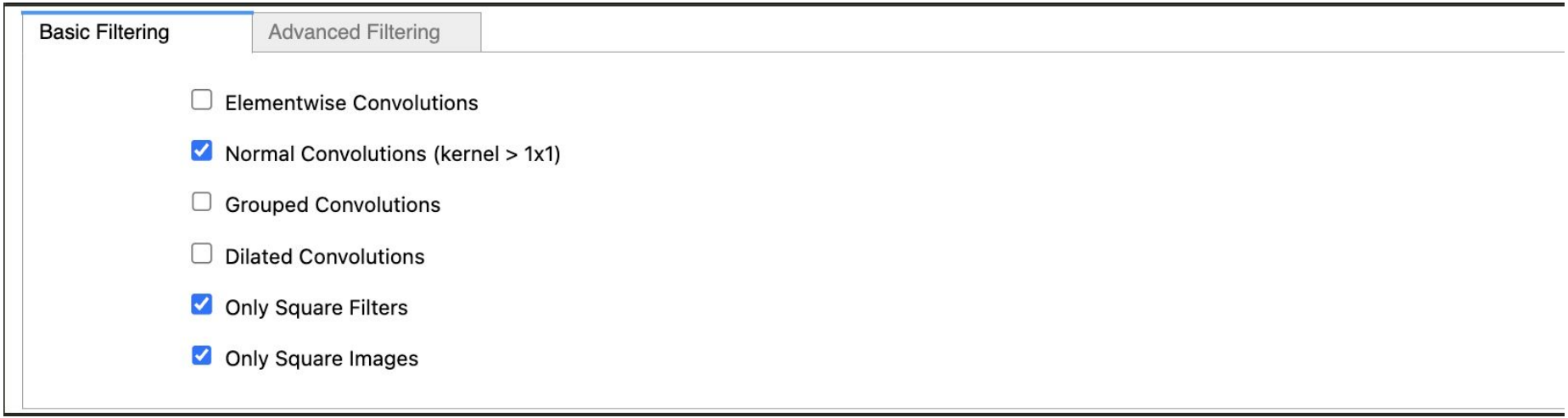
Number of Float Point Operations (#FLOPS):

- ranges from **903 #MFLOPS** up to **43 #GFLOPS**

Operation set construction

Filtering convolution operation set.

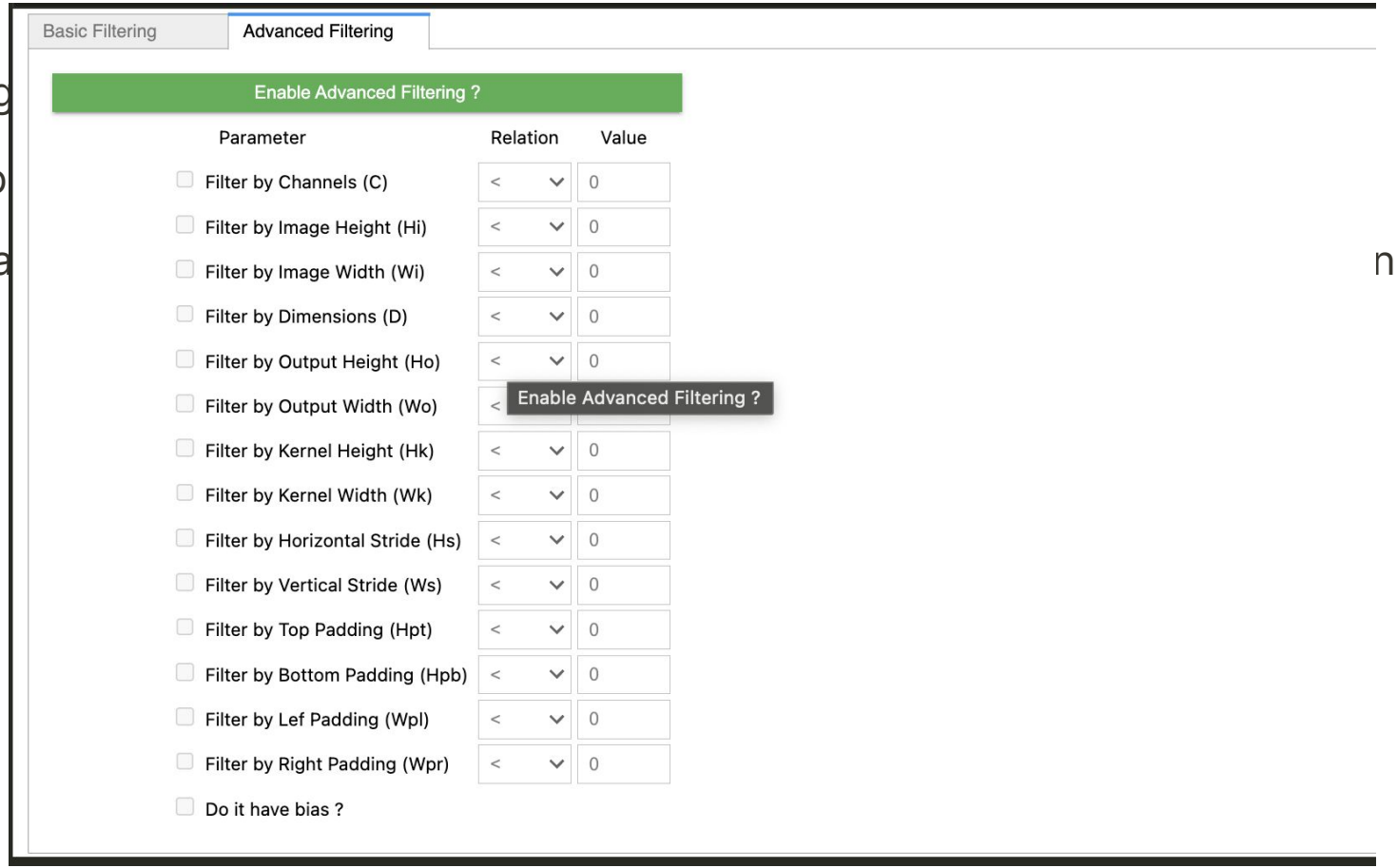
- ✓ Operation set organized saved as pickle file.
- ✓ Transform to pandas Dataframe and **Filter** by a just composed filtering string.



Operation set construction

Filtering

- ✓ Op
- ✓ Tra ng.



SECTION

Convolution Algorithm Definition and ConvBench Structure

ConvBench Structure

ConvBench was designed as a including **Header file.**

- It Inherits from a *timing.h* class;
- And implements all the benchmark methods in a templated class**, ConvBench**.
  - Instances to be benchmarked must **inherit** from ConvBench class and:
    - Properly **override** the *convolution* function.
    - Properly **override** the *convolution_baseline* function.
    - And **Implement** a simple *main* function.
  - ConvBench *convset_exec()* takes care of the convolution algorithm assessment

ConvBench Structure: Timing class

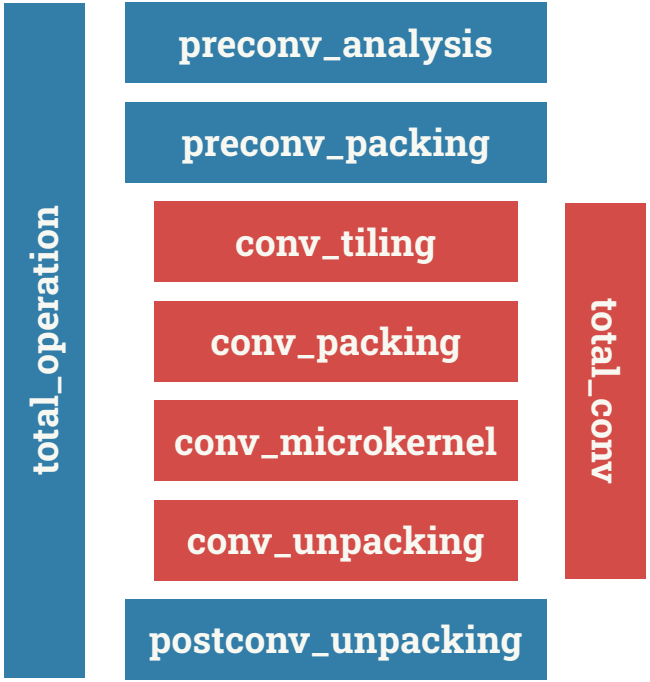
Standardized timing nomenclature.

Just enclose code snippets between the:

<time_name>_start()

<time_name>_update()

And you are ready to go! At the end specialized plots will be available.



ConvBench Structure: ConvBench class

ConvBench class is a templated class by *T* (Theoretically, accepting any kind of data type).

- ***void convset_load(csv_filename)***:
  - Load the filtered convolution operation set.
- ***void convdata_gen(datagen_strategy)***:
  - Populate the data buffers (**Random/Constant**).
- ***void convset_exec(running_strategy)***:
  - The main execution procedure (**Correctness/Direct/Baseline**).
- ***virtual void convolution(args ...)***:
  - The main convolution algorithm to be assessed.
- ***virtual void convolution_baseline(args ...)***:
  - The baseline convolution algorithm to be compared.

ConvBench Structure • Code example


```
#include “convbench.h”

template <T>
class Inherited_Conv : public ConvBench<T>{

      //constructors and destructors
      Inherited_Conv();
      ~Inherited_Conv();

      //Conv algorithms
      void convolution(args ... ) override {
            // Main convolution algorithm implementation
      }

      void convolution_baseline(args ... ) override {
            // baseline convolution algorithm implementation
      }

      //Some other stuff if needed

}

int main() {
      // Object instantiation
      Inherited_Conv<T> bench = Inherited_Conv<T>();
      // Benchmark execution
      bench.convset_exec(args ... );
      return 0;
}
```

ConvBench Structure • Code example


```
#include “convbench.h”

template <T>
class Inherited_Conv : public ConvBench<T>{

      //constructors and destructors
      Inherited_Conv();
      ~Inherited_Conv();

      //Conv algorithms
      void convolution(args ... ) override {
            // Main convolution algorithm implementation
      }

      void convolution_baseline(args ... ) override {
            // baseline convolution algorithm implementation
      }

      //Some other stuff if needed

}

int main() {
      // Object instantiation
      Inherited_Conv<T> bench = Inherited_Conv<T>();
      // Benchmark execution
      bench.convset_exec(args ... );
      return 0;
}
```

ConvBench Structure • Code example


```
#include “convbench.h”

template <T>
class Inherited_Conv : public ConvBench<T>{

      //constructors and destructors
      Inherited_Conv();
      ~Inherited_Conv();

      //Conv algorithms
      void convolution(args ... ) override {
            // Main convolution algorithm implementation
      }

      void convolution_baseline(args ... ) override {
            // baseline convolution algorithm implementation
      }

      //Some other stuff if needed

}

int main() {
      // Object instantiation
      Inherited_Conv<T> bench = Inherited_Conv<T>();
      // Benchmark execution
      bench.convset_exec(args ... );
      return 0;
}
```

ConvBench Structure • Code example


```
#include “convbench.h”

template <T>
class Inherited_Conv : public ConvBench<T>{

      //constructors and destructors
      Inherited_Conv();
      ~Inherited_Conv();

      //Conv algorithms
      void convolution(args ... ) override {
            // Main convolution algorithm implementation
      }

      void convolution_baseline(args ... ) override {
            // baseline convolution algorithm implementation
      }

      //Some other stuff if needed

}
```

```
int main() {
      // Object instantiation
      Inherited_Conv<T> bench = Inherited_Conv<T>();
      // Benchmark execution
      bench.convset_exec(args ... );
      return 0;
}
```

SECTION

Results Summary, Dataframe Visualization and Plotting

Results Summary, Dataframe visualization and Plotting

ConvBench returns a CSV file of timing measurements.

Thus, you can:

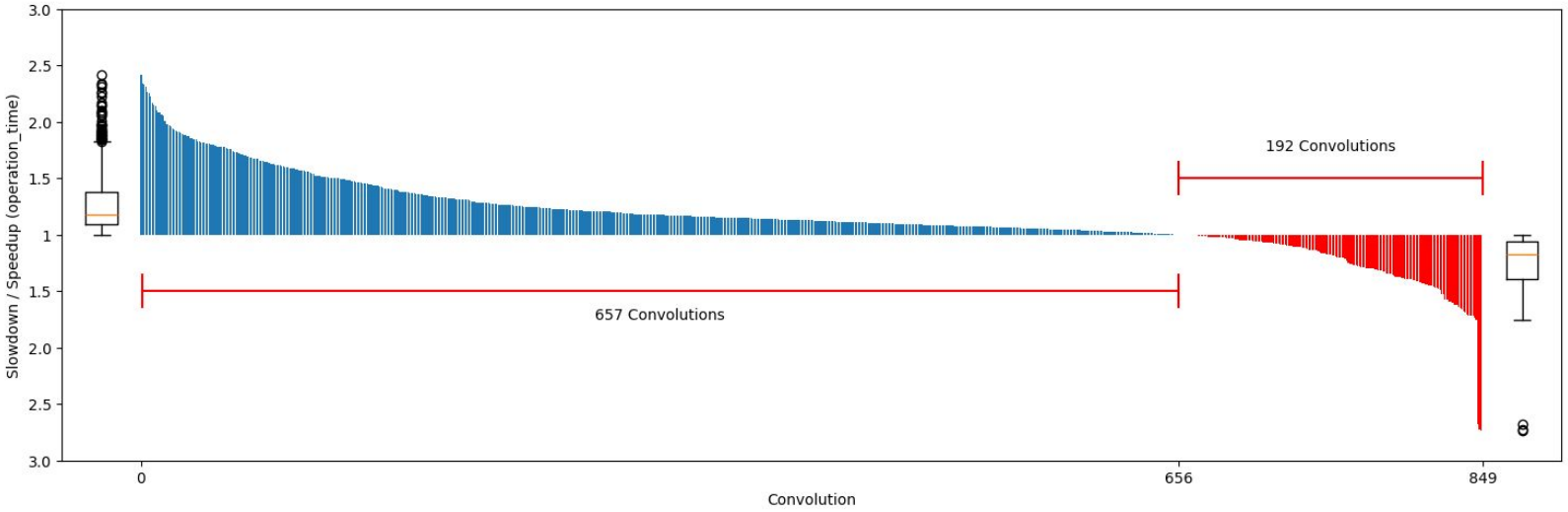
- ✓ Simply open it with some spreadsheet software;
- ✓ Open as a **pandas dataframe**;
- ✓ Filter, reshape, inspect.
- ✓ **Plot the results** with matplotlib.
- ✓ And so on ....

For example, we added the following automatically analysis:

Results Summary and Plotting: SConv

Normal convolution operations (filtering) speedup distribution (analysis)

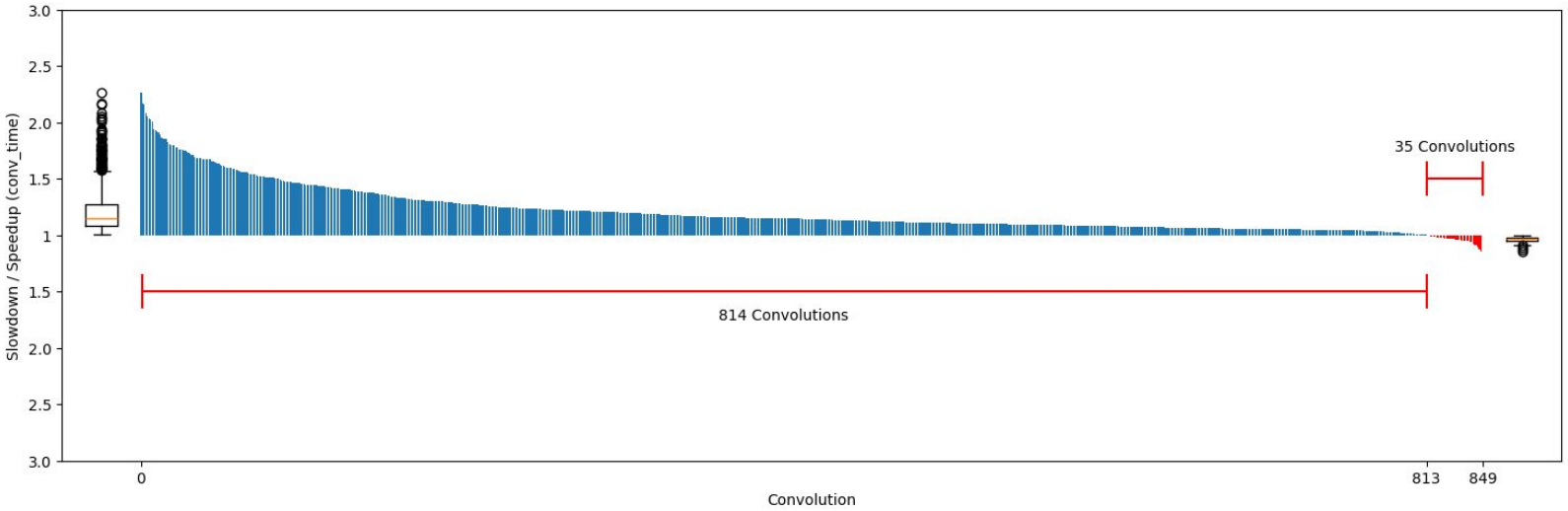
Total Operation Time speedup Analysis.



Results Summary and Plotting: SConv

Normal convolution operations (filtering) speedup distribution (analysis)

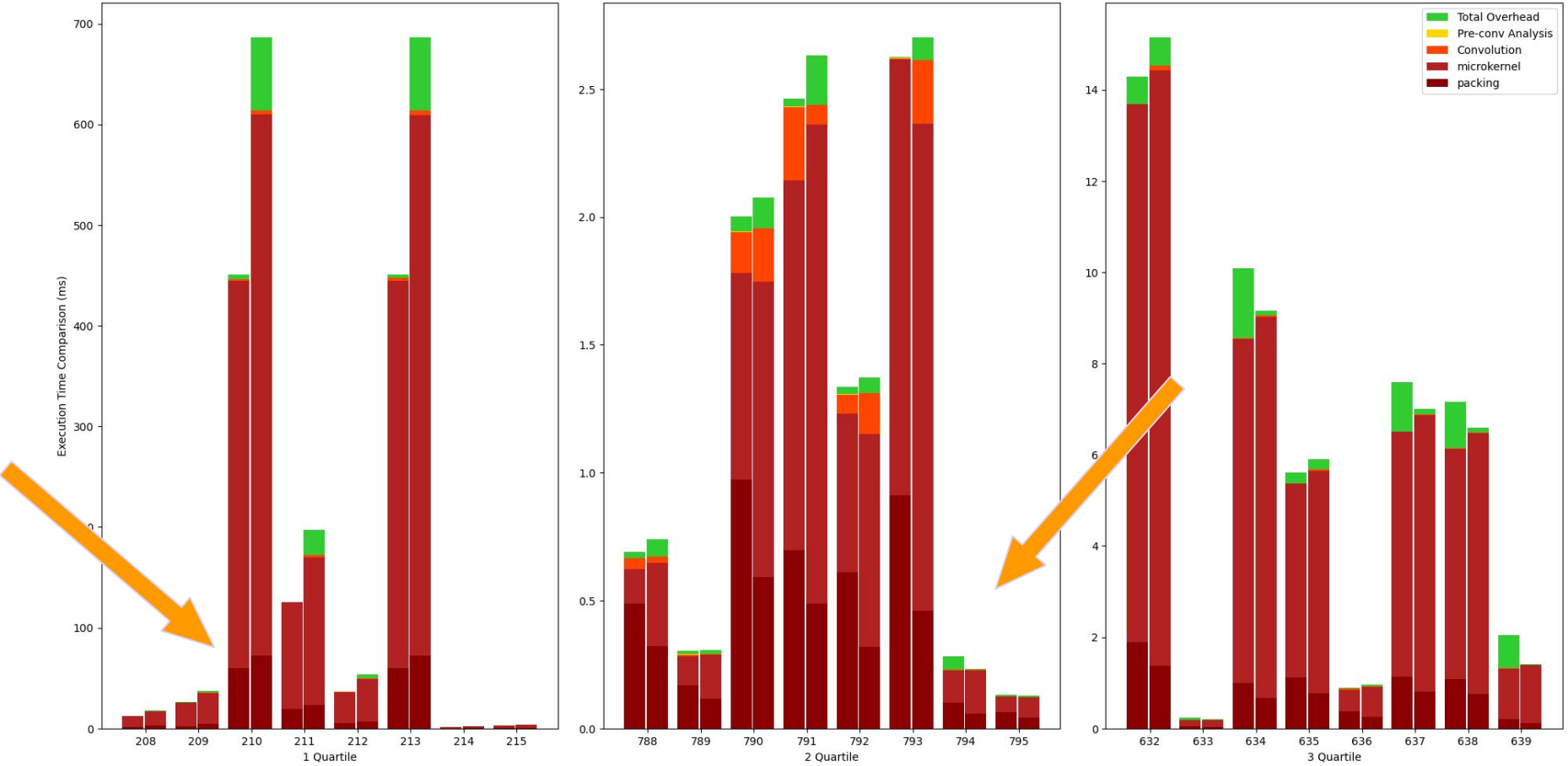
Total Convolution Time speedup analysis.



Results Summary and Plotting: SConv

Execution time breakdown graph.

Results Summary and Plotting: SConv



Results Summary and Plotting: SConv

Execution time breakdown graph.

The conv-packing step is **58,09%** faster than baseline on positive speedup cases.

However, on slowdown cases, the conv-packing step is **68,3%** slower than main algorithm.

SECTION

Conclusion

Conclusion

- ConvBench: an end-to-end platform for evaluating convolution algorithms.
- Fair comparison: timing class.
- A throughout assessment of 2D convolution operations (9K+ operations) + Filtering mechanism
- An automatic experimental procedure + result summarization in tables and plots

Thank You!

Email lucas.silva@ic.unicamp.br
GitHub https://github.com/LucasFernando-aes