Implementierung und Optimierung von CNNs in C++ mit pragmabasierten Ansätzen

11. Februar 2022

Friedrich-Alexander-Universität Erlangen-Nürnberg

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Übersicht

1. Überblick
2. OpenMP
3. OpenACC
4. Evaluation

Überblick

Implementierung: Überblick

```
ConvLayer<1, in_dim_x, in_dim_y, 3, 3> c0;
LeakyReLuLayer3D<c0.num_filter, c0.out_dim_x, c0.out_dim_y> c1;
ConvLayer<c1.dim_x, c1.dim_y, c1.dim_z, 3, 3> c2;
LeakyReLuLayer3D<c2.num_filter, c2.out_dim_x, c2.out_dim_y> c3;
FlattenLayer<c3.dim_x, c3.dim_y, c3.dim_z> c4;
FullyConnectedLayer<c4.out_dim, num_classes> c5;
XEntropyLoss<num_classes> c6;
```

- template-basierte Klassen pro Layer
- c-style Arrays für Gewichte, Bias, Eingabe, ...
- jede Klasse speichert Ausgabe des Layers
  $\Rightarrow$ als Input für das darauffolgende
  $\Rightarrow$ keine Speicherkopien / doppelt genutzter Speicher!

OpenMP

OpenMP

The OpenMP API supports multi-platform shared-memory parallel programming in C/C++ and Fortran. The OpenMP API defines a portable, scalable model with a simple and flexible interface for developing parallel applications on platforms from the desktop to the supercomputer.

`openmp.org`

OpenMP

> The OpenMP API supports multi-platform shared-memory parallel programming in C/C++ and Fortran. The OpenMP API defines a portable, scalable model with a simple and flexible interface for developing parallel applications on platforms from the desktop to the supercomputer.
>
> `openmp.org`

fork-join model of parallel execution:
Multiple threads of execution perform tasks defined (implicitly or explicitly) by OpenMP directives can produce different results if different addition associations occur.

- Aufpassen bei Sichtbarkeit von shared Variablen in Schleifen
- `reduction` directives vor Schleifen

OpenMP: Optimierung

Optimierung:

- Nutzung von pragmas / directives um parallele Regionen zu definieren
- überlasse den Rest dem Compiler

OpenMP: Optimierung

Optimierung:

- Nutzung von pragmas / directives um parallele Regionen zu definieren
- überlasse den Rest dem Compiler

CNN Implementierung besteht aus vielen Schleifen
$\Rightarrow$ OpenMP loop directive wo möglich

- loop directive ist parallel region
- Region wird mit aktueller Menge an Threads ausgeführt (`OMP_NUM_THREADS`)

OpenACC

OpenACC

OpenACC supports offloading of both computation and data from a host device to an accelerator device. In fact, these devices may be the same or may be completely different architectures, such as the case of a CPU host and GPU accelerator.

`openacc.org`

OpenACC

> OpenACC supports offloading of both computation and data from a host device to an accelerator device. In fact, these devices may be the same or may be completely different architectures, such as the case of a CPU host and GPU accelerator.
>
> `openacc.org`

⇒ platformunabhängige Programmierhochsprache für Beschleuniger

+ ein Source Code für viele Geräte + gute Performance bei allen
- kostet Performance: manche Optimierungen nur in lower-level Programmiermodellen (CUDA, OpenCL, …)

OpenACC: Optimierung

Basic directives:

- `#pragma acc kernels` um zu beschleunigenden Bereich
- `#pragma acc parallel loop` als Alternative

OpenACC: Optimierung

Basic directives:

- `#pragma acc kernels` um zu beschleunigenden Bereich
- `#pragma acc parallel loop` als Alternative

Weitere directives:

`... reduction(+:sum)` etc.

`... collapse(n)`

`... independent`

OpenACC: Optimierung

Basic directives:

- `#pragma acc kernels` um zu beschleunigenden Bereich
- `#pragma acc parallel loop` als Alternative

Weitere directives:

`... reduction(+:sum)` etc.

`... collapse(n)`

`... independent`

Zur Überprüfung des generierten Codes / Schleifen / ... :

```
nvc++ -Minfo=accel
```

OpenACC: Speicherverwaltung

In the case that the two devices have different memories the OpenACC compiler and runtime will analyze the code and handle any accelerator memory management and the transfer of data between host and device memory.

OpenACC: Speicherverwaltung

> In the case that the two devices have different memories the OpenACC compiler and runtime will analyze the code and handle any accelerator memory management and the transfer of data between host and device memory.

Massiver Overhead für Datentransfer von/zu Host zu/von Gerät

$$2 \cdot (60000 + 10000) \cdot (7 \cdot 2) = 1960000 \text{ Kopiervorgänge}$$

OpenACC: Speicherverwaltung

> In the case that the two devices have different memories the OpenACC compiler and runtime will analyze the code and handle any accelerator memory management and the transfer of data between host and device memory.

Massiver Overhead für Datentransfer von/zu Host zu/von Gerät

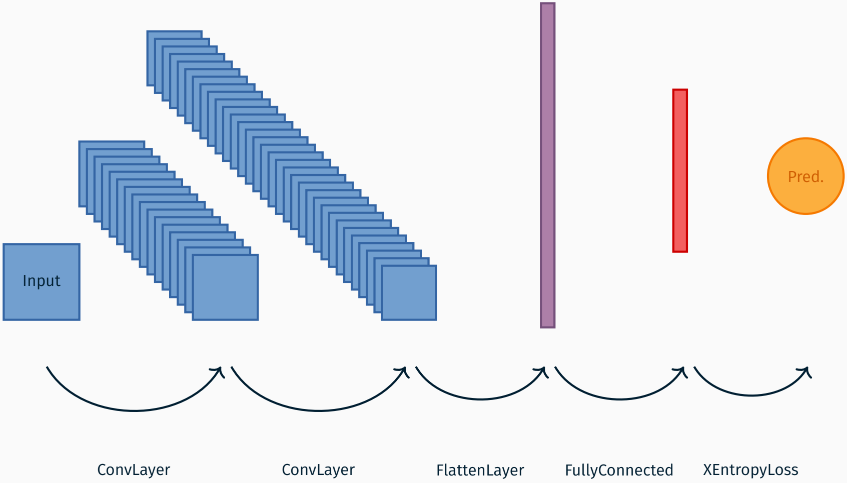
$$2 \cdot (60000 + 10000) \cdot (7 \cdot 2) = 1960000 \text{ Kopiervorgänge}$$

Lösung: mehr directives!

- `enter data`
- `exit data`
- `present`
- `update`

Evaluation

Netzwerklayout



Evaluationsumgebung

Evaluationsparameter:

- mit/ohne Optimierungen:
  - ’normales’ Programm
  - OpenMP (`-fopenmp`)
  - OpenACC (`-acc=gpu`)
- Skalierungsfaktor für x und y Dimensionen (1, 2)
- Filtergrößen (aus 2, 3, 4, 8, 16, 32)
- Kernelgröße (3x3, 5x5)

Evaluationsumgebung

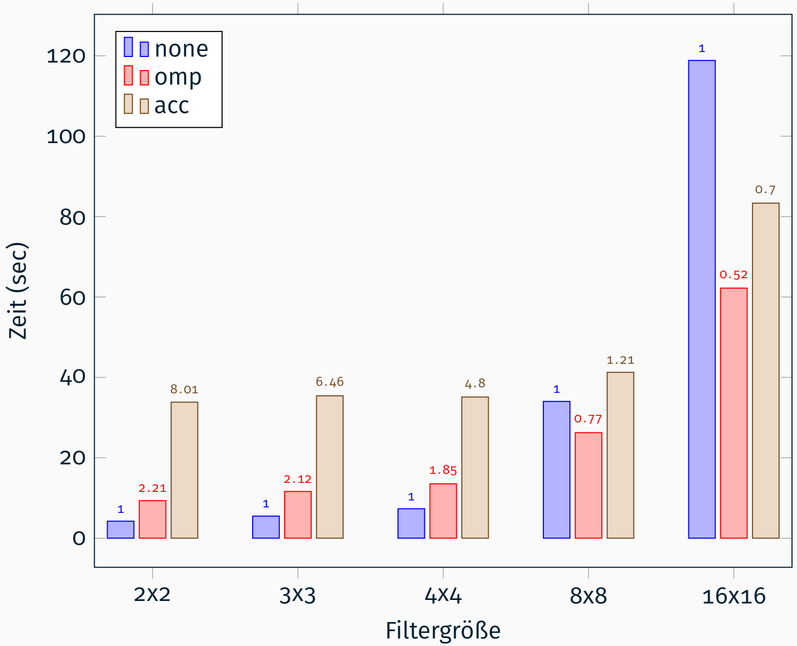
Evaluationsparameter:

- mit/ohne Optimierungen:
  - 'normales' Programm
  - OpenMP (`-fopenmp`)
  - OpenACC (`-acc=gpu`)
- Skalierungsfaktor für x und y Dimensionen (1, 2)
- Filtergrößen (aus 2, 3, 4, 8, 16, 32)
- Kernelgröße (3x3, 5x5)

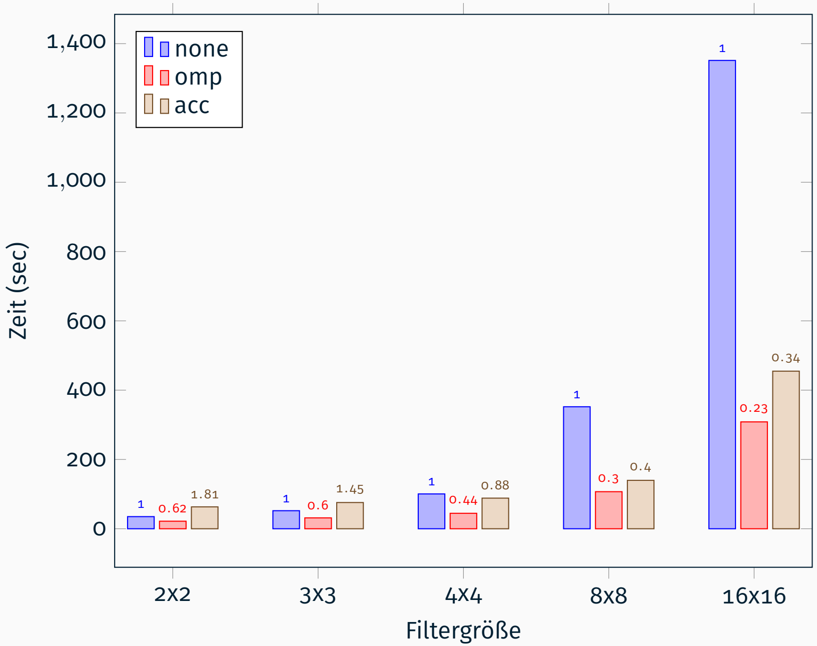
Umgebung: CIP3 @ FAU

- Intel(R) Core(TM) i7-8700 @ 3.20GHz, 6 cores, 32GB RAM
- Nvidia GeForce RTX 2080
- Debian 11 / bullseye, Kernel version 5.10.69
- nvc++ 21.9-0, C++17, OpenMP 201511, OpenACC 2.7

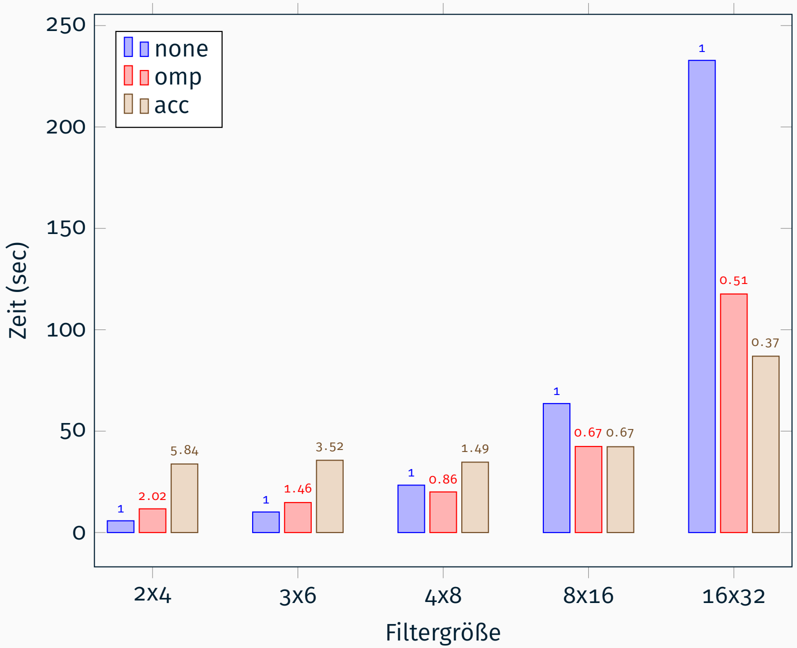
Ergebnisse: Gleich große Filter, Kernelgröße = 3, Scaling = 1



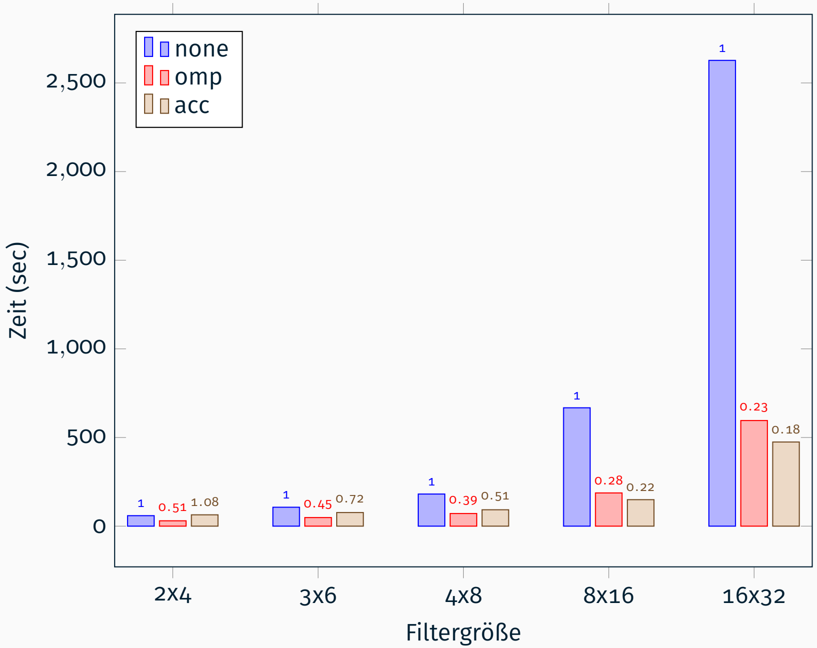
Ergebnisse: Gleich große Filter, Kernelgröße = 5, Scaling = 2



Ergebnisse: zweiter Filter größer, Kernelgröße = 3, Scaling = 1



Ergebnisse: zweiter Filter größer, Kernelgröße = 5, Scaling = 2



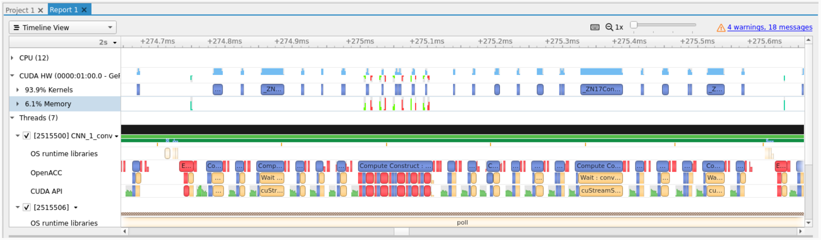
Ergebnisse: genauere Betrachtung

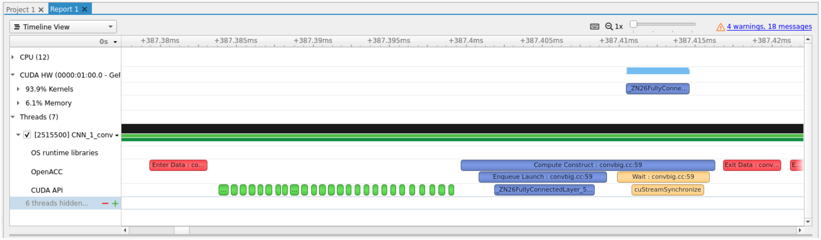
Analyse mithilfe von `nvprof`:

- `./CNN_1_convbig_16_16_5_acc`: Genauigkeit von 0.980900
  - Gesamtlaufzeit: 141.187s
  - 44.27% (44.5685s) der GPU-Aktivität in einem Kernel des backwards im zweiten Conv-Layer
  - 39.16% (48.1313s) der OpenACC-Routinen in `acc_wait@convbig.cc:115`
  - 86.07% (101.412) der API-Calls in `cuStreamSynchronize`

- `./CNN_1_convbig_16_32_5_acc`: Genauigkeit von 0.983400
  - Gesamtlaufzeit: 145.572s
  - 43.48% (45.2223s) der GPU-Aktivität in einem Kernel des backwards im zweiten Conv-Layer
  - 38.42% (48.8038s) der OpenACC-Routinen in `acc_wait@convbig.cc:115`
  - 86.26% (104.623s) der API-Calls in `cuStreamSynchronize`

Ergebnisse: genauere Betrachtung mit nsight-sys



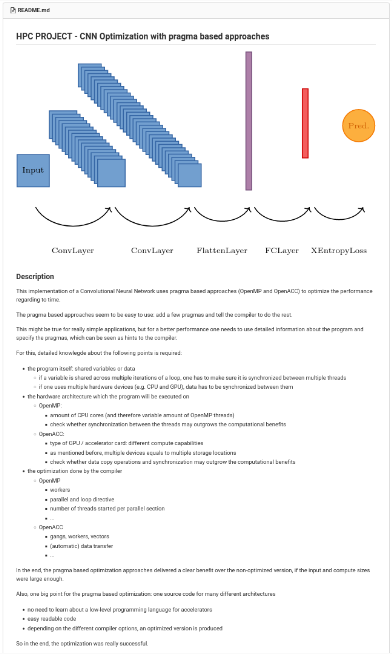


Zusammenfassung

Wann macht Beschleunigung mit OpenMP und OpenAcc Sinn?

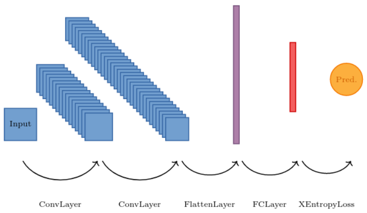
- kostengünstige Parallelisierung von nicht-kritischen Funktionalitäten (Code muss nicht/kaum umgeschrieben werden)
- Portierbarkeit (Compiler übernimmt Abbildung auf vorhandene Zielgeräte)
- Entwicklung von Prototypen von Parallelenanwendungen (Ausprobieren von Parallelisierbarkeit)

https://gitlab.cs.fau.de/hpc-project-ss21/cnn-openmp



README.md

HPC PROJECT - CNN Optimization with pragma based approaches



Description

This implementation of a Convolutional Neural Network uses pragma based approaches (OpenMP and OpenACC) to optimize the performance regarding to time.

The pragma based approaches seem to be easy to use: add a few pragmas and tell the compiler to do the rest.

This might be true for really simple applications, but for a better performance one needs to use detailed information about the program and specify the pragmas, which can be seen as hints to the compiler.

For this, detailed knowlegde about the following points is required:

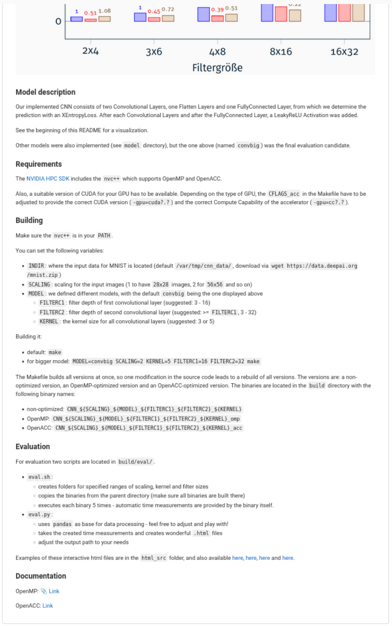
- the program itself: shared variables or data
  - if a variable is shared across multiple iterations of a loop, one has to make sure it is synchronized between multiple threads
  - if one uses multiple hardware devices (e.g. CPU and GPU), data has to be synchronized between them
- the hardware architecture which the program will be executed on
  - OpenMP:
    - amount of CPU cores (and therefore variable amount of OpenMP threads)
    - check whether synchronization between the threads may outgrows the computational benefits
  - OpenACC:
    - type of GPU / accelerator card: different compute capabilities
    - as mentioned before, multiple devices equals to multiple storage locations
    - check whether data copy operations and synchronization may outgrow the computational benefits
- the optimization done by the compiler
  - OpenMP
    - workers
    - parallel and loop directive
    - number of threads started per parallel section
    - ...
  - OpenACC
    - gangs, workers, vectors
    - (automatic) data transfer
    - ...

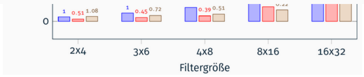
In the end, the pragma based optimization approaches delivered a clear benefit over the non-optimized version, if the input and compute sizes were large enough.

Also, one big point for the pragma based optimization: one source code for many different architectures

- no need to learn about a low-level programming language for accelerators
- easy readable code
- depending on the different compiler options, an optimized version is produced

So in the end, the optimization was really successful.





Model description

Our implemented CNN consists of two Convolutional Layers, one Flatten Layers and one FullyConnected Layer, from which we determine the prediction with an XEntropyLoss. After each Convolutional Layers and after the FullyConnected Layer, a LeakyReLU Activation was added.

See the beginning of this README for a visualization.

Other models were also implemented (see `model` directory), but the one above (named `convbig`) was the final evaluation candidate.

Requirements

The NVIDIA HPC SDK includes the `nvc++` which supports OpenMP and OpenACC.

Also, a suitable version of CUDA for your GPU has to be available. Depending on the type of GPU, the `CFLAGS_acc` in the Makefile have to be adjusted to provide the correct CUDA version (`-gpu=cuda?.?`) and the correct Compute Capability of the accelerator (`-gpu=cc?.?`).

Building

Make sure the `nvc++` is in your `PATH`.

You can set the following variables:

- `INDIR` : where the input data for MNIST is located (default `/var/tmp/cnn_data/` , download via `wget https://data.deepai.org/mnist.zip`)
- `SCALING` : scaling for the input images (1 to have `28x28` images, 2 for `56x56` and so on)
- `MODEL` : we defined different models, with the default `convbig` being the one displayed above
  - `FILTERC1` : filter depth of first convolutional layer (suggested: 3 - 16)
  - `FILTERC2` : filter depth of second convolutional layer (suggested: >= `FILTERC1` , 3 - 32)
  - `KERNEL` : the kernel size for all convolutional layers (suggested: 3 or 5)

Building it:

- default: `make`
- for bigger model: `MODEL=convbig SCALING=2 KERNEL=5 FILTERC1=16 FILTERC2=32 make`

The Makefile builds all versions at once, so one modification in the source code leads to a rebuild of all versions. The versions are: a non-optimized version, an OpenMP-optimized version and an OpenACC-optimized version. The binaries are located in the `build` directory with the following binary names:

- non-optimized: `CNN_${SCALING}_${MODEL}_${FILTERC1}_${FILTERC2}_${KERNEL}`
- OpenMP: `CNN_${SCALING}_${MODEL}_${FILTERC1}_${FILTERC2}_${KERNEL}_omp`
- OpenACC: `CNN_${SCALING}_${MODEL}_${FILTERC1}_${FILTERC2}_${KERNEL}_acc`

Evaluation

For evaluation two scripts are located in `build/eval/` .

- `eval.sh` :
  - creates folders for specified ranges of scaling, kernel and filter sizes
  - copies the binaries from the parent directory (make sure all binaries are built there)
  - executes each binary 5 times - automatic time measurements are provided by the binary itself.
- `eval.py` :
  - uses `pandas` as base for data processing - feel free to adjust and play with!
  - takes the created time measurements and creates wonderful `.html` files
  - adjust the output path to your needs

Examples of these interactive html files are in the `html_src` folder, and also available here, here, here and here.

Documentation

OpenMP: 📎 Link

OpenACC: Link

Links zum genaueren Vergleichen

URLs:

`wwwcip.informatik.uni-erlangen.de/~na79zegy/dot/uni/hpcproject`

`https://wwwcip.informatik.uni-erlangen.de/~na79zegy/dot/uni/hpcproject/plot_eqfilter_scaling1.html`

`https://wwwcip.informatik.uni-erlangen.de/~na79zegy/dot/uni/hpcproject/plot_eqfilter_scaling2.html`

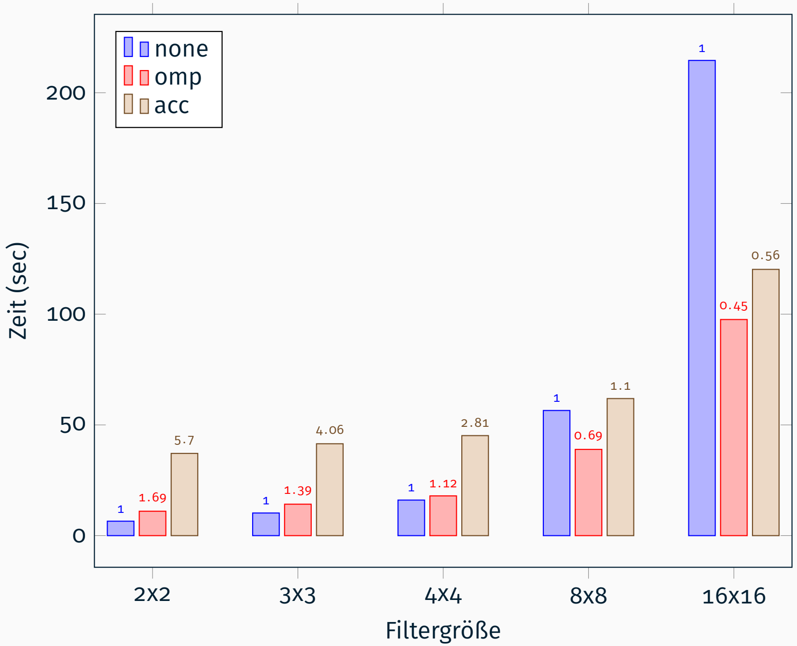
`https://wwwcip.informatik.uni-erlangen.de/~na79zegy/dot/uni/hpcproject/plot_noneqfilter_scaling1.html`

`https://wwwcip.informatik.uni-erlangen.de/~na79zegy/dot/uni/hpcproject/plot_noneqfilter_scaling2.html`

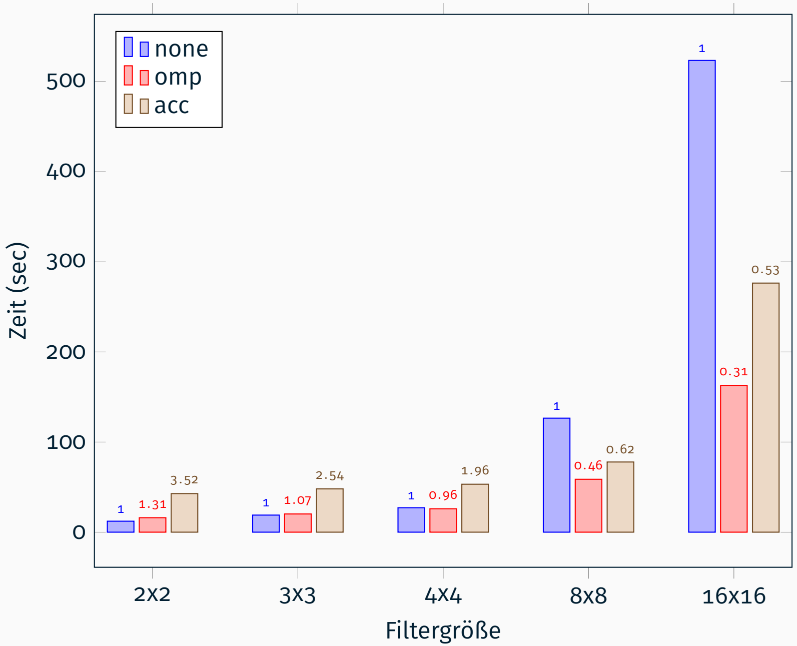
Vielen Dank fürs Zuhören!

Fragen?

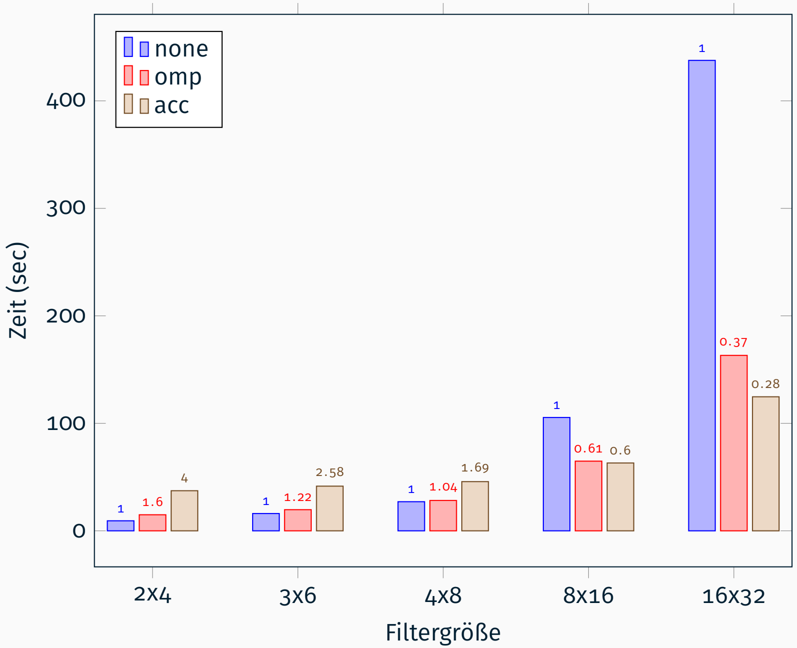
Ergebnisse: Gleich große Filter, Kernelgröße = 5, Scaling = 1



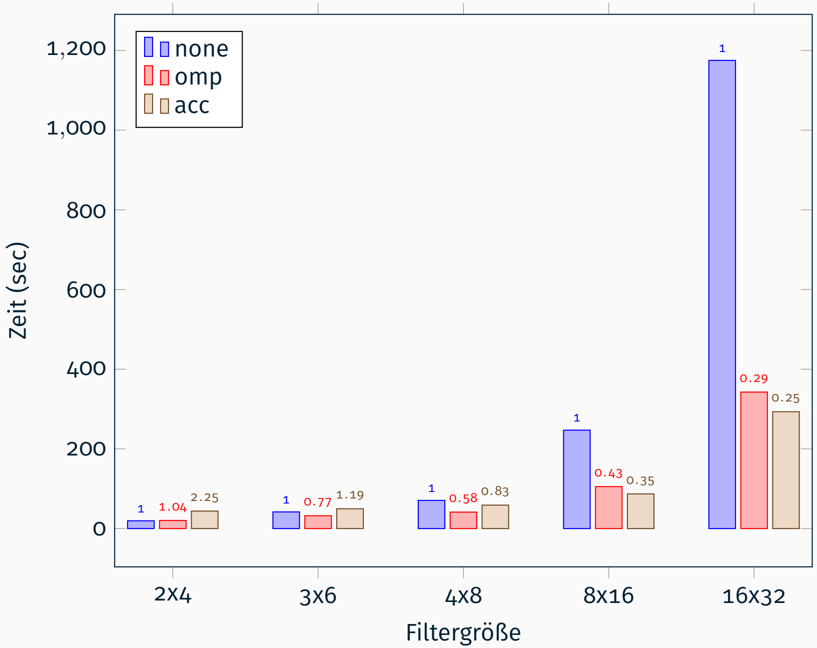
Ergebnisse: Gleich große Filter, Kernelgröße = 3, Scaling = 2



Ergebnisse: zweiter Filter größer, Kernelgröße = 5, Scaling = 1



Ergebnisse: zweiter Filter größer, Kernelgröße = 3, Scaling = 2


none
omp
acc
Zeit (sec)
2x4
3x6
4x8
8x16
16x32
Filtergröße