

AutoGEMM

Installation and user guide

Dec 15, 2022

Contents

1	Description	1
2	Package structure	1
3	Installation steps	2
3.1	Prerequisites and supported platforms	2
3.2	Creation of a virtual environment	2
3.3	Additional software prerequisites	3
3.4	Installation of Apache TVM	3
4	Driver configuration and usage	3
4.1	Configuration files	3
4.1.1	Execution configuration file (<code>exec.cfg</code>)	3
4.1.2	Machine configuration file (<code>machine.cfg</code>)	4
4.1.3	Driver execution	4
4.1.4	Parallel executions	5
5	Prototypes of the generators	5

1 Description

AutoGEMM is a software package that offers a family of Python code generators for high-performance GEMM (General Matrix-Matrix Multiplication) implementations based on Apache TVM.

This guide complements the submitted paper *Automatic Generators for a Family of Matrix Multiplication Routines with Apache TVM* and includes basic setup and execution steps for the package.

The provided software package includes all the generators described in the paper (listed in Sections 5, 6, and 7 of the manuscript) and used to extract the performance plots in Section 8, together with a simplified high-level *driver program* to easily use them.

2 Package structure

The software bundle consists on two main directories:

- `doc/`

File name	Description
<code>userManual.pdf</code>	This documentation file (PDF format)
<code>userManual.html</code>	This documentation file (HTML format)

- `src/`: Main code directory containing the driver and configuration files prepared to conduct experiments.

File name	Description
<code>driver.py</code>	Simplified driver to conduct specific performance experiments using different configuration parameters and code generators
<code>exec.cfg</code>	Configuration file to define the general experimental parameters
<code>machine.cfg</code>	Configuration file to define the specific features of the target architecture
<code>generators/</code>	Directory containing code generators, described below

Within the `src/` directory, the Python scripts for the generators described in the paper can be found in the `generators` subdirectory:

- `src/generators/`: Directory containing Python generators leveraging Apache TVM to generate architecture-specific GEMM codes. Specifically, the included files mimic the code generators listed in the manuscript, namely:

File name	Description	Listing
<code>basic_GEMM_B3A2C0.py</code>	TVM generator for the basic GEMM	5
<code>blocked_GEMM_B3A2C0.py</code>	TVM generator for GEMM mimicking the blocking scheme of the baseline algorithm	6
<code>packed_GEMM_B3A2C0.py</code>	TVM generator for GEMM mimicking the (blocking scheme and) packing of the baseline algorithm	
<code>packed_GEMM_B3A2C0_ukernel.py</code>	TVM generator for GEMM mimicking (the blocking and packing of) the baseline algorithm, and integrating the optimized micro-kernel with C-resident.	8
<code>opt_GEMM_B3A2C0_ukernel.py</code>	TVM generator for GEMM mimicking (the blocking and packing of) the baseline algorithm, integrating both the optimized micro-kernel with C-resident and fine-grain optimizations	9
<code>opt_GEMM_B3A2C0_ukernel_parallel.py</code>	TVM generator for GEMM mimicking (the blocking and packing of) the baseline algorithm, integrating both the optimized micro-kernel with C-resident and fine-grain optimizations and a loop-level parallelization of loop <code>ic</code>	9
<code>opt_GEMM_A3C2B0_ukernel.py</code>	TVM generator for GEMM mimicking the blocking and packing schemes of the A3C2B0 algorithm	11
<code>opt_GEMM_C3A2B0_ukernel.py</code>	TVM generator for GEMM mimicking the blocking and packing schemes of the C3A2B0 algorithm	12

3 Installation steps

3.1 Prerequisites and supported platforms

AutoGEMM relies on two software prerequisites: any installation of Python 3, and Apache TVM. The prerequisites for the latter are not considered in this documentation, but will be automatically managed and installed by the following installation steps. To ease the installation process, the following instructions are based on the creation of a virtual environment via Python.

3.2 Creation of a virtual environment

```
# Installation of Python3 virtualenv (considering a Debian-based Linux distribution)
user@machine:~$ apt-get install python3-virtualenv
# Creation of a virtual Python environment with name .tvm
user@machine:~$ python3 -m venv .tvm
# Activation of the virtual environment
user@machine:~$ source .tvm/bin/activate
(.tvm) user@machine:~$
```

3.3 Additional software prerequisites

```
# Installation of additional software requisites via pip
(.tvm) user@machine:~$ pip install typing_extensions pytest
```

3.4 Installation of Apache TVM

```
# Installation of Apache TVM and dependences
(.tvm) user@machine:~$ pip install apache-tvm
```

4 Driver configuration and usage

4.1 Configuration files

4.1.1 Execution configuration file (`exec.cfg`)

This configuration file includes the specific conditions for the execution of one experiment. The file is structured in sections, each one including a number of mandatory parameters (we consider next the operation with matrices input matrices A and B , and output matrix C , $C := A * B$):

- Section [GENERAL]:
 - *repeats*: number of repetitions of the test (boolean).
 - *assembly*: generation of the assembly code for further analysis (boolean).
- Section [EXPERIMENT]:
 - *variant*: algorithmic variant for GEMM (string). Supported values are B3A2C0, A3C2B0 and C3A2B0.
 - *opt_level*: optimization level of the TVM generator (string). Accepted values for variant B3A2C0 are basic, blocked, packed, packed_ukernel, opt_ukernel and opt_ukernel_par. Variants A3C2B0 and C3A2B0 only accept the opt_ukernel optimization level.
 - *dtype*: datatype to use (string).
- Section [PROBLEM]:
 - *M*: number of rows of matrix C .
 - *N*: number of columns of matrix C .
 - *K*: number of columns of matrix A /rows of matrix B .
- Section [BLOCKSIZES]:
 - *mr*: register blocksize m_r (integer).
 - *nr*: register blocksize n_r (integer).
 - *kr*: register blocksize k_r (integer).
 - *MC*: cache blocksize m_c (integer).
 - *NC*: cache blocksize n_c (integer).
 - *KC*: cache blocksize k_c (integer).
 - *ls*: lanesize, number of lanes in vector registers of the target architecture (integer).

Example of configuration file `exec.cfg`:

```
#####
[GENERAL]
repeats = 1
assembly = 1

[EXPERIMENT]
variant = B3A2C0
opt_level = opt_ukernel
dtype = float32

[PROBLEM]
M = 4096
N = 4096
K = 4096
```

```
[BLOCKSIZES]
mr = 4
nr = 16
kr = 4

MC = 256
NC = 1280
KC = 128

ls = 16
#####
```

4.1.2 Machine configuration file (`machine.cfg`)

This configuration file includes the specific characteristics of the target architecture. In this version, this information includes a label describing the architecture, and the `llvm` target used to generate code, as follows:

- Section `[MACHINE]`:
 - `name`: label to identify the experiment, under user’s choice (string).
 - `target`: `llvm` target identifier (string).

Example of configuration file `machine.cfg` (in this example, targeting an Intel Icelake architecture):

```
#####
[MACHINE]
name = icelake

target = llvm -mcpu=icelake-server
#####
```

Some examples for *target* used in the manuscript include:

Architecture	Target
Basic (native wihtout optimization)	<code>llvm</code>
ARMv8a (8.2) with NEON	<code>llvm -device=arm_cpu -mattr=+v8.2a,+fp-armv8,+neon</code>
ARMv8a (8.2) with NEON and FP16	<code>llvm -device=arm_cpu -mattr=+v8.2a,+fp-armv8,+neon,+fp16fml</code>
AMD Zen2 with AVX2	<code>llvm -mcpu=znver2</code>
Intel Icelake with AVX512	<code>llvm -mcpu=icelake-server</code>

4.1.3 Driver execution

The file `driver.py` accepts two mandatory arguments that indicate the path to the configuration files. A detailed usage guide can be extrated upon execution with the `-h` argument:

```
(.tvm) user@machine:~$ python3 driver.py -h
## Test driver for BLAS generators
usage: Test driver. [-h] -c EXECCONFIG -m MACHINECONFIG
```

Description

```
optional arguments:
  -h, --help            show this help message and exit
  -c EXECCONFIG, --execconfig EXECCONFIG
                        Execution configuration file
  -m MACHINECONFIG, --machineconfig MACHINECONFIG
                        Machine configuration file
```

An example execution of the driver is illustrated next:

```
(.tvm) user@machine:~$ python3 driver.py -c exec.cfg -m machine.cfg
## Test driver for BLAS generators
icelake,B3A2C0_opt_ukernel_par,4096,4096,4096,256,1280,128,4,16,4,465.45
```

Note that the CSV output includes the execution information, namely:

- Machine name.
- Execution configuration in the format *variant_optimizationlevel*.
- Matrix dimensions (M, N, K).
- Cache blocksizes (MC, NC, KC).
- Register blocksizes (mr, nr, kr).
- Performance (in terms of GFLOPS).

If the assembly file has been requested in the execution configuration file, it will be placed in the `assembly` folder with name *variant_MxNxK_mrxnr.s* (e.g. *B3A2C0_256x1280x128_4x16.s*).

4.1.4 Parallel executions

For codes generated with support for multithreading, the environment variable `TVM_NUM_THREADS` controls the number of threads deployed for execution. As an example, the following execution:

```
(.tvm) user@machine:~$ TVM_NUM_THREADS=1 python3 driver.py -c exec.cfg -m machine.cfg
## Test driver for BLAS generators
icelake,B3A2C0_opt_ukernel_par,4096,4096,4096,256,1280,128,4,16,4,32.46
```

would execute a sequential version of the corresponding GEMM code, whereas

```
(.tvm) user@machine:~$ TVM_NUM_THREADS=64 python3 driver.py -c exec.cfg -m machine.cfg
## Test driver for BLAS generators
icelake,B3A2C0_opt_ukernel_par,4096,4096,4096,256,1280,128,4,16,4,464.63
```

would execute a parallel version deploying 64 threads (observe the difference in performance reported as the last field of the comma-separated output). By default, TVM deploys as many threads as cores are available in the system.

5 Prototypes of the generators

- Prototypes:

```
def basic_GEMM_B3A2C0(m, n, k, mc, nc, kc, mr, nr, kr, lanesize, dtype, target)
def blocked_GEMM_B3A2C0(m, n, k, mc, nc, kc, mr, nr, kr, lanesize, dtype, target)
def packed_GEMM_B3A2C0(m, n, k, mc, nc, kc, mr, nr, kr, lanesize, dtype, target)
def opt_GEMM_B3A2C0_ukernel(m, n, k, mc, nc, kc, mr, nr, kr, lanesize, dtype, target)
def opt_GEMM_B3A2C0_ukernel_parallel(m, n, k, mc, nc, kc, mr, nr, kr, lanesize, dtype, target)
def opt_GEMM_C3A2C0_ukernel(m, n, k, mc, nc, kc, mr, nr, kr, lanesize, dtype, target)
def opt_GEMM_A3C2B0_ukernel(m, n, k, mc, nc, kc, mr, nr, kr, lanesize, dtype, target)
```

- Parameters:

Parameter	Type	Description
m	integer	Number of rows of C (rows of A)
n	integer	Number of columns of C (columns of A)
k	integer	Number of columns of A (rows of B)
mc	integer	Cache parameter. Number of rows of Ac
nc	integer	nc cache parameter. Number of columns of Bc
kc	integer	kc cache parameter. Number of columns of Ac (rows of Bc)
mr	integer	Micro-kernel parameter. Number of rows of Cr (rows of Ar)
nr	integer	Micro-kernel parameter. Number of columns of Cr (columns of Br)

Parameter	Type	Description
kr	integer	Micro-kernel parameter. Number of rows of each micro-panel in Bc (not used in variant B3A2C0)
lanesize	integer	Number of elements (lanes) in a vector register
dtype	Numpy datatype	Datatype for matrix elements
target	TVM target	Target description, following the convention in https://tvm.apache.org/docs/reference/api/python/target.html

- Return value:
func: Code module for the target device