crow Documentation

Release 0.1

Andrej Copar

Sep 11, 2018

Contents:

Crow is a multi-GPU and multi-CPU implementation of non-negative matrix tri-factorization.

CHAPTER 1

Setup guide

We recommend building and running CROW with docker to avoid recompiling libraries manually. If you want to install on host server directly, follow the *Manual setup*.

Before you use `make` you need to clone the repository crow git repository.

```
git clone https://github.com/acopar/crow && cd crow
```

The following command will install `docker` and copy `crow` executables. Depending on whether you have CUDA-enabled GPUs, it will also install nvidia-docker and CUDA toolkit.

```
make install
```

In case of any problems, check your dependencies and follow the *requirements setup guide*.

CHAPTER 2

Docker containers

Start the container with provided `crow-start` command:

```
crow-start
```

You can force CPU-only version on a system with GPUs available with `crow-start cpu`. Container will run in the background. You can check if the container is running with `docker ps`. You can stop the container with `crow-stop` command, or use specific `docker` commands (Tutorial).

2.1 Connect to a running container

Then connect to a container with either `crow-exec`, like this:

```
crow-exec
```

Alternatively, you can ssh into the container. For login, `crow-ssh` uses dedicated identity files that are located at ~/.ssh/crow and ~/.ssh/crow.pub. The keys are generated during installation added to container's authorized keys automatically.

```
crow-ssh
```

You can login into container as admin like this:

```
crow-sudo
```

2.2 Docker volumes

Crow docker images makes use of the following external volumes.

- crow: path to the crow source code (for development versions).
- data: path to directory with data, mounted read-only.

- results: this is where the factorized data will be stored.
- cache: path to directory, where the application stores intermediate files. Starting with empty folder, the cache can take several gigabytes, depending on your data so make sure that you have enough space on the partition. You can safely clean this folder, but note that it may take some time to process the data again.

By default, mount points for each volume points to a folder in the current working directory. Refer to the *data section* for more information on how to manage folders shared with the container. Any data not stored in shared folders will disappear when you stop the container.

Now, you can proceed to the *Tutorial*.

2.3 Non-sudo users

Users without administrator privileges need to be in `docker` group. Also, CUDA and nvidia-docker2 need to be installed on the host. You can manage containers with executables located in `scripts` folder (add it to your PATH). Once inside docker the container, follow the *Tutorial* on how to factorize your data.

2.4 Test run

After you have the environment up and running, you can use this command to test if everything works correctly.

```
crow-test -g
```

This command generates a small random dataset and tries to factorize it. The `-g` switch tells the test to use the GPUs.

Documentation for more in-depth benchmarks is found here *Benchmark*.

2.5 Supported platforms

This software has been developed and tested for Linux platform. The provided docker images should also work on other platforms. Follow the official instructions on how to set up docker environment for your platform.

CHAPTER 3

Host requirements

Docker install requirements for all environments:

- docker-ce >= 17.03

Requirements for GPU environments:

- CUDA >= 8.0
- nvidia-docker >= 2.0

If you don't have these programs on the system, you can install them using `make` (see below) or install them manually. If you install dependencies manually, run `scripts/install-crow.sh` after to install `crow` executables.

3.1 Quick setup

You can use the provided install scripts to setup requirements automatically. Currently Ubuntu-based systems are supported through helper scripts, otherwise, use manual instructions below. Clone the repository to access install scripts.

```
git clone https://github.com/acopar/crow && cd crow
```

Install the framework and its dependencies (Ubuntu users):

```
make install
```

3.2 CUDA

If you do not have nvidia driver and CUDA on your system, refer to this section: *CUDA install guide*.

3.3 Docker engine

Here are instructions for Ubuntu Linux. For other Linux distributions and other platforms refer to the Official docker install guide.

```
sudo apt-get install apt-transport-https ca-certificates curl software-properties-
↪common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
↪$(lsb_release -cs) stable"
sudo apt-get update
sudo apt-get install docker-ce
```

3.4 Nvidia-docker

Download and install nvidia-docker. These instructions will work on most Linux distributions. Alternatively, Ubuntu and CentOS users can also use the provided deb and rpm packages. This software is for GPU hosts only.

```
curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add -
curl -s -L https://nvidia.github.io/nvidia-docker/ubuntu16.04/amd64/nvidia-docker.
↪list | sudo tee /etc/apt/sources.list.d/nvidia-docker.list
sudo apt-get update

# Install nvidia-docker2 and reload the Docker daemon configuration
sudo apt-get install -y nvidia-docker2
sudo pkill -SIGHUP dockerd
```

3.5 Add user to docker group

In order to execute docker commands as a regular user, you need to have docker privileges:

```
sudo gpasswd -a $(whoami) docker
```

CHAPTER 4

Legacy nvidia-docker

Use of nvidia-docker v1.0 is deprecated, we recommend the new v2.0, which is supported in the latest version of `crow`. For backward compatibility, here are instructions on how to install the legacy version:

```
wget -P /tmp https://github.com/NVIDIA/nvidia-docker/releases/download/v1.0.1/nvidia-
↪docker_1.0.1_amd64.tar.xz
sudo tar --strip-components=1 -C /usr/bin -xvf /tmp/nvidia-docker*.tar.xz && rm /tmp/
↪nvidia-docker*.tar.xz
```

In order to use nvidia-docker, you need to run `nvidia-docker-plugin` at boot. If this was not enabled through nvidia-docker packages (deb, rpm), you can run this command on system boot.

```
sudo -b nohup nvidia-docker-plugin > /tmp/nvidia-docker.log
```

4.1 nvidia-docker-compose

Legacy versions also require `nvidia-docker-compose` library:

```
git clone https://github.com/eywalker/nvidia-docker-compose
sudo cp nvidia-docker-compose/bin/nvidia-docker-compose /usr/local/bin
```

This application depends on yaml python module, which can be installed with `pip`.

```
sudo pip install pyyaml
```

or using the packages from your distribution.

```
sudo apt-get install python-yaml
```

CHAPTER 5

Manual setup

Alternatively, you can install manually (refer to the dockerfile for details). Install requirements:

- Python == 2.7
- CUDA == 8.0
- openMPI >= 2.0 (make sure that the –with-cuda flag is set during configure time)
- openBLAS >= 1.12

5.1 Python

```
apt-get install python-dev python-pip
```

5.2 CUDA

If you have CUDA-enabled GPU devices, you have probably already installed nvidia drivers and CUDA. If not, you can install them from your package repository, like this:

```
sudo apt-get install nvidia-cuda-toolkit
reboot # nvidia driver requires reboot
```

This should also take care of installing current nvidia driver. If you want the latest version of CUDA or specific instructions for different platform, follow the Official CUDA installation guide.

5.3 openMPI

```
wget https://www.open-mpi.org/software/ompi/v2.0/downloads/openmpi-2.0.0.tar.gz
tar xzf openmpi-2.0.0.tar.gz
cd openmpi-2.0.0
./configure --with-cuda=/usr/local/cuda --prefix=/openmpi
make -j$(nproc) && make install
```

5.4 openBLAS

```
git clone https://github.com/xianyi/OpenBLAS
cd OpenBLAS && make -j$(nproc) FC=gfortran && make PREFIX=/OpenBLAS install
echo '/OpenBLAS/lib' > /etc/ld.so.conf.d/openblas.conf && ldconfig
```

5.5 NumPy

Compile numpy with openblas support

```
pip install cython
git clone https://github.com/numpy/numpy
printf "[openblas]\n" \
"libraries = openblas\n" \
"library_dirs = /OpenBLAS/lib\n" \
"include_dirs = /OpenBLAS/include\n" \
"runtime_library_dirs = /OpenBLAS/lib\n" >> numpy/site.cfg

cd numpy && python setup.py config && \
    python setup.py build -j $(nproc) && python setup.py install
```

5.6 Other system dependencies

Install libraries required for cuda-cffi module and zmq module.

```
apt-get install libffi-dev libzmq3-dev
```

5.7 Install crow and python dependencies

Set up environment variables, so it can find CUDA, OpenBLAS and openMPI libraries. To make changes permanent after reboot, you need to append these lines to /etc/profile or ~/.bashrc (depending on your configuration).

```
export PATH="/OpenBLAS/bin:/openmpi/bin:$PATH"
export LD_LIBRARY_PATH="/OpenBLAS/lib:/openmpi/lib:$LD_LIBRARY_PATH"
export CUDA_ROOT="/usr/local/cuda"
```

First download crow source from github.

```
git clone https://github.com/acopar/crow
cd crow
```

Install python requirements for CPU and GPU systems.

```
pip install -r requirements.txt
```

5.8 GPU module dependencies

Install python modules for GPU environment. Skip this step on computer without GPU devices.

```
pip install -r requirements-gpu.txt
```

You may need to get more recent version of scikit-cuda.

```
git clone https://github.com/lebedov/scikit-cuda
cd scikit-cuda && python setup.py install
```

5.9 Install crow module

Finally, you can install the package:

```
python setup.py install
```

You can also use pip to install current package.

```
pip install -e .
```

5.10 Test the configuration

You can check if the installed modules work correctly by importing them. Also check if your CUDA_ROOT and PATH environment variables are set correctly. If you use different distribution, path to CUDA can be different.

```
echo $CUDA_ROOT
ls $CUDA_ROOT
```

```
python -c 'import pycuda' # CUDA for python library
python -c 'from cuda_cffi import cusparse' # for sparse GPU operations
python -c 'import numpy; numpy.__config__.show()' # check blas info
```

CHAPTER 6

Tutorial

The application reads the provided `coo` file into data matrix X (see *data manipulation* on how to structure such file). Then it the data into three non-negative latent factors, such that:

$$X \approx USV^T$$

6.1 Examples

Serial configuration using one CPU core where both factorization ranks are 20.

```
crow -k1 20 data.coo
```

Using single GPU, number of iterations is set to 200 and factorization ranks are 20x30.

```
crow -g -k1 20 -k2 30 -i 200 data.coo
```

Example usage for 4-GPU run with 2x2 block configuration and factorization rank 20.

```
crow -g -b 2x2 -k1 20 data.coo
```

When the process is finished, you should have the following files in `results` directory:

- U.npz - contains left factor matrix.
- V.npz - contains right factor matrix.
- S.npz - contains middle factor matrix.

6.2 Command line arguments

The following options can be set:

- -b: block configuration, for example 2x2.

- -e: calculate and print error function in each iteration.
- -g: use this argument to run on GPUs. By default, only CPU cores will be used.
- -i: maximum number of iterations, default is 100.
- -k1: left factorization rank. Defines number of latent vectors of matrix U.
- -k2: right factorization rank. Defines number of latent vectors of matrix V. By default, value of k1 is used.
- -o: impose orthogonality in factors U and V. By default non-orthogonal NMTF will be used.
- -p: parallelization degree, by default number of blocks equals to parallelization degree, but you can use parallelization degree smaller than the number of blocks. Useful to reduce memory requirements in GPU applications.
- -s: use sparse data structures. Do not use this if the matrix density is larger than 10%.
- -t: additional stopping criteria. By default, factorization will run for the number of iterations specified by -i. Available arguments are e4, e5, e6, e7. For example, when passing e6 parameter, the factorization stops after error function in two consecutive iterations changes for less than 10^-6.

Arguments:

- Single argument specifies path to data file. You can also provide basename of data files that exist in *data directory*.

CHAPTER 7

Docker volumes

Crow docker images makes use of the following external volumes.

- crow: path to the crow source code (for development).
- data: path to directory with data, mounted read-only.
- results: this is where the factorized data will be stored.
- cache: path to directory, where the application stores intermediate files. Starting with empty folder, the cache can take several gigabytes, depending on your data so make sure that you have enough space on the partition. You can safely clean this folder, but note that it may take some time to process the data again.

You can use symbolic links to connect path to your data, like this:

```
ln -s <path-to-your-data-folder> data
mkdir results
mkdir -p ../tmp/crow-cache
ln -s ../tmp/crow-cache cache
```

Alternatively, customize docker command and modify the volume paths to point to the desired locations:

```
docker run -d "$runtime" --name="crow" -p 22 -v $PWD/crow:/home/crow/crow -v $PWD/
↪data:/home/crow/data:ro -v $PWD/cache:/home/crow/cache -v $PWD/results:/home/crow/
↪results --rm acopar/crow
```

CHAPTER 8

Data manipulation

Before you can factorize your data, you first need to convert it into an appropriate format.

8.1 Input data

The data can be provided in coordinate list format (coo), which is a form of csv file, where each row describes one element in a matrix with row, column, value and header stores matrix dimensions. In header, we define matrix dimensions **n,m**. After that, each row of the file represents one non-zero value in the matrix. In each row, the first column represents index at first dimension **i**, second column index of second dimension **j** and third column represents the value of data matrix **X** at location X[i,j].

For example, consider this 2D matrix:

```
[[1, 0, 0], [5, 5, 0]]
```

Corresponding `coo` data file would look like this:

```
2,3
0,0,1
1,0,5
1,1,5
```

The following formats are used:

- coo - this is the default input data format, using coordinate list representation.
- npz - this method uses numpy.savez. Default format for resulting factors. It is also used to cache dense or csr sparse matrices internally.
- pkl - used to store other python data, such as dictionaries.
- csv - not used internally, but supported through conversion. Note that when converting back, there is no header line.

8.2 Convert between different csv and coo formats

By default, data format used is coordinate list file (coo), and is currently used to represent input data for both sparse and dense matrices. However, dense matrices are often formated with a 2D csv table, separated by commas or tabulars. The following commands show examples how can you convert existing csv file into coo format. If there is a header line, you need to provide an option, so the converter knows to ignore the first row.

```
crow-conv --header 2d-table.csv coo-value-list.coo
```

Examples with explicit delimiter (comma by default) and no header.

```
crow-conv -d comma 2d-table.csv key-value.coo
crow-conv -d tab 2d-table.tab key-value.coo
```

8.3 Convert between npz and csv

Without command line arguments, the conversion is done based on file extension. If the file extension does not correspond to the content, use arguments to explicitly tell the input and output type. Arguments are checked with `crow-conv --help`.

```
crow-conv results/U.npz results/U.coo
crow-conv results/U.npz results/U.csv
```

8.4 Pickle to csv

There is also a convenience tool, which can convert pickled numpy array to csv format (dense).

```
crow-conv test.pkl test.csv
```

CHAPTER 9

Download data

To download preprocessed benchmark datasets, use the provided `get_datasets.sh` script.

```
scripts/get_datasets.sh
```

This script downloads five datasets that have already been converted into coordinate list format:

- ArrayExpress: 22283x5896, file size: 3.5GB
- TCGA-BRCA: 1222x60483, file size: 1.5GB
- Fetus: 25569x25608, file size: 622M
- Retina: 25823x25822, file size: 2.9GB
- Cochlea: 25824x25824, file size: 5.6GB
- TCGA-Methyl: 10181x485577, file size: 19GB
- TCGA-Methyl (cancer gene subset): 10181x14299, file size: 556MB

CHAPTER 10

Benchmark

When running benchmarks, you can read the time of iteration for each configuration from the terminal. If you wath to visualize, you can use the automated scripts, which use docker image.

10.1 Quick start

Download repository `crow-plot` benchmarking and plotting framework and run it. Note that for this to work, you need to have a running crow docker container.

```
git clone https://github.com/acopar/crow-plot
cd crow-plot
./speedup-bench.sh
```

If everything works correctly, you should have a few png and pdf files in img directory. The results should give you an idea on speedups you can get from your hardware configuration.

10.2 Benchmark list

Here is a list of available benchmarks:

- speedup-bench.sh: comparison of speedup on 1-4GPUs compared to a single processor configuration.
- efficiency-bench.sh: efficiency and communication overhead of multi-block configurations as described by percentage of linear speedup.
- rank-bench.sh: impact of factorization rank on speed.
- balanced-bench.sh: speedup of balanced partitioning compared to imbalanced partitioning on sparse datasets.

By default, GPU and multi-GPU configurations are compared against single-processor configuration. If you want to compare the speedup of multi-processor against a single-processor as well, add `-a` option to each benchmark script, for example:

```
./speedup-bench.sh -a
```

To test orthogonal NMTF, you need to provide `-o` option to the script:

```
./speedup-bench.sh -o
```

Note, that for orthogonal visualizations, you need to run visualization separately (see the next section).

10.3 Visualize results

The results that you see in the output of benchmark script, are also stored in the results folder in csv files. If you want to visualize the results, you can use the provided `scripts/crow-plot` script, like this:

```
scripts/crow-plot python /app/crowpl -a speedup ArrayExpress TCGA-BRCA fetus retina␣
↪cochlea
```

Option `-a` defines which type of visualization to perform. Available visualizations types are:

- speedup: visualize speedup compared to single-processor approach.
- efficiency: visualize efficiency on multi-block configurations.
- transfer: visualize percentage of communication overhead.
- k: visualize iteration times for different factorization ranks.
- balance: visualize speedup of balanced partitioning over imbalanced on sparse data.

To visualize results for orthogonal NMTF benchmarks, you also need to provide `-o` argument.

```
scripts/crow-plot python /app/crowpl -o -a speedup ArrayExpress TCGA-BRCA fetus␣
↪retina cochlea
```

10.4 Manual setup

To run benchmarks without visualization, it is not necessary to install the `crow-plot` package. You can run this command directly, provided that crow docker container is running.

```
python crowpl/benchmark.py ArrayExpress TCGA-BRCA fetus retina cochlea
```

Benchmark command line arguments:

- -a: run all experiments, including multi-processor.
- -c: run with disabled data transfer for communication overhead calculation.
- -i: run sparse datasets imbalanced, thus checking the partitioning speedup.
- -o: run orthogonal NMTF (Ding et al., 2006) instead of default non-orthogonal (Long et al., 2005) method.
- -r: benchmark factorization rank values.
- -u: update, configurations that were already computed are skipped.

To avoid the use of docker image for visualization, you can install `crow-plot` module manually. System dependencies of `crow-plot` are imagemagick and latex in addition to python modules listed in requirements.txt.

```
python setup.py install
```

After the installation is complete you can run crowpl module like this:

```
python crowpl -a speedup ArrayExpress TCGA-BRCA fetus retina cochlea
```

CHAPTER 11

Co-clustering example

11.1 Setup instructions

Before you run this example, you need to install CROW framework and start the container.

```
git clone https://github.com/acopar/crow-example
cd crow-example
pip install -r requirements.txt
```

11.2 Run Experiments

```
python main.py
```

CHAPTER 12

Reference

CHAPTER 13

Indices and tables

- genindex
- modindex
- search