Lecture 12 - GPGPU (The Easy Parts) Part 2

DSE 512

Drew Schmidt
2022-03-03

From Last Time

- Homework is out --- due Saturday
- New homework "soon"
- Getting a GPU on ISAAC
  - Have not tested NVBLAS
  - Assume this works fine with R, not Python
  - More on ISAAC GPUs later
- Questions?

GPU Interfaces

Implicit (NVBLAS)

- Pros
  - Easy to use
  - No code changes
  - Multi-GPU
- Cons
  - Not suitable for all problems
  - Weird segfaults?

Explicit (CuPy, fmlr)

- Pros
  - Data stays on GPU
  - Optimized for many operations
  - Small(er) problems can work
- Cons
  - Harder to use
  - Code changes
  - Multi-GPU isn't guaranteed

CuPy

CuPy

- GPU accelerated NumPy and SciPy
- Uses NVIDIA CUDA
- Can be tricky to install…
- Know NumPy? Know CuPy.
- API reference
  https://docs.cupy.dev/en/stable/reference/index.html

CuPy

Installing

In the Cloud

- conda
- pip binaries
- Docker container
- ~~Build from source~~

On ISAAC

- ~~conda~~
- ~~pip binaries~~
- Singularity container
- Build from source

Problem with Conda (binaries generally)

```
conda create --name mce cupy
# ...
conda activate mce
python
```

```
import cupy as cp
```

```
ImportError: libcuda.so.1: cannot open shared object file: No such file or directory
```

Installing

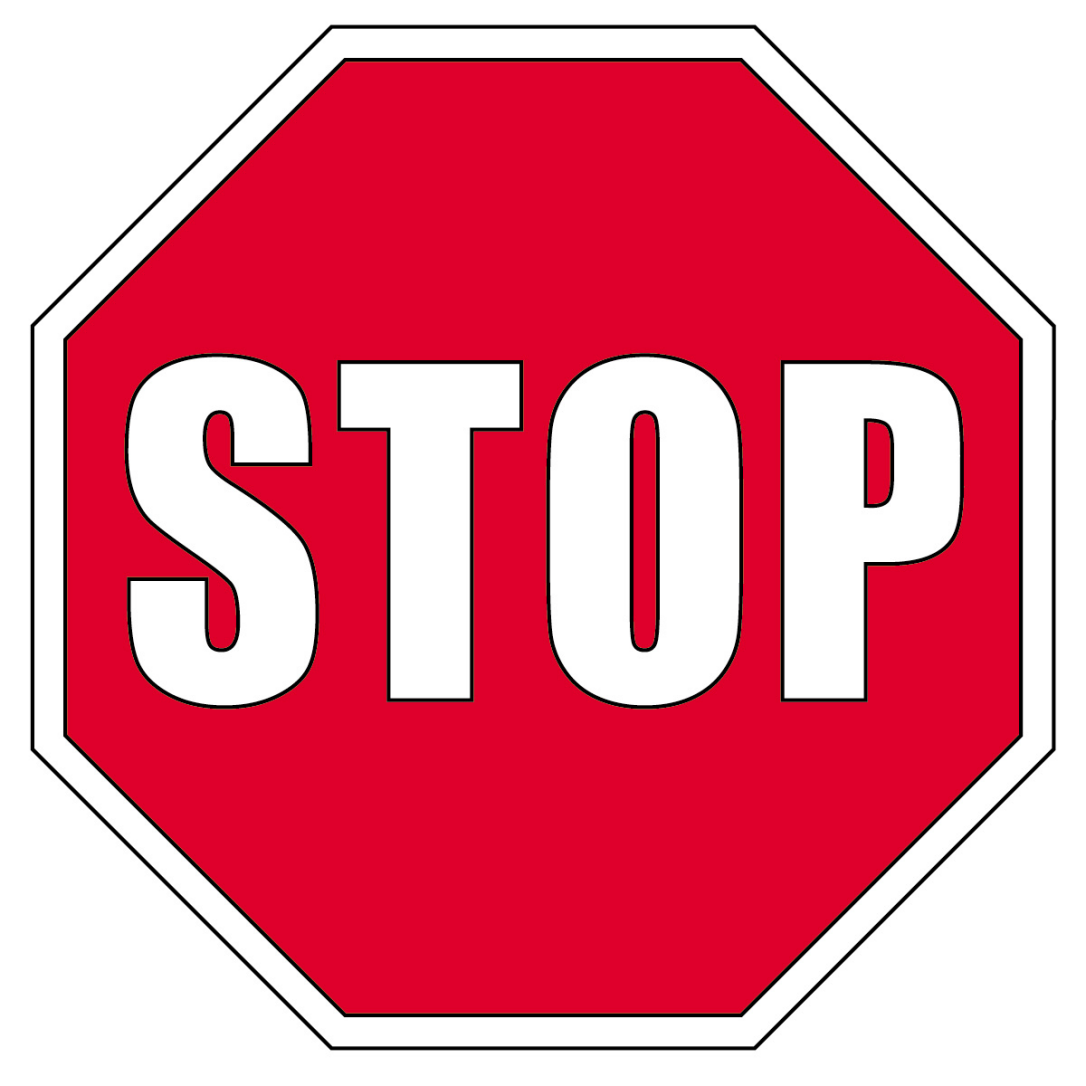
In Theory

```
pip install cupy
```

In Reality

```
module unload PE-Intel
module load cuda
module load gcc
export CUDA_HOME=/sw/isaac/applications/cu
CC=gcc NVCC=${CUDA_HOME}/bin/nvcc \
  CUDA_PATH="${CUDA_HOME}" \
  LDFLAGS=${CUDA_HOME}/lib64 \
  CFLAGS="-I${CUDA_HOME}/include" \
  pip install cupy-cuda114
```

AND IT STILL DOESN'T WORK



Running CuPy on ISAAC

- I give up
- Just use singularity

```
singularity exec --nv cupy.simg python
```

- Images
  - cuda https://hub.docker.com/r/nvidia/cuda/
  - CuPy https://hub.docker.com/r/cupy/cupy

Building a CuPy Singularity Image

```
MY_CONTAINER="cupy/cupy"
mkdir -p /tmp/d2s
docker run \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /tmp/d2s:/output \
  --privileged -t --rm \
  singularityware/docker2singularity ${MY_CONTAINER}
```

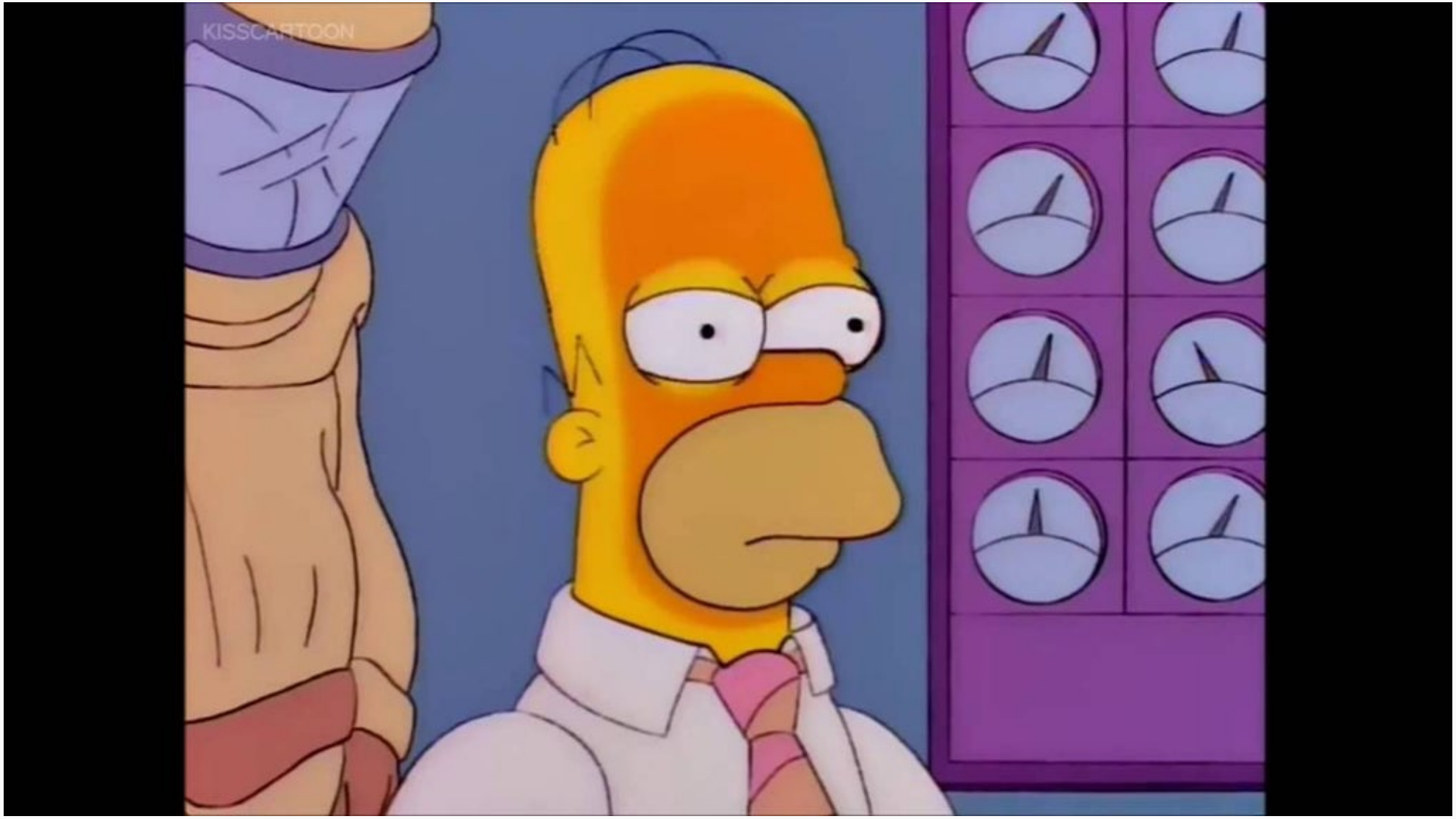
`/lustre/isaac/proj/UTK0188/cupy.simg`

Running the Image

```
singularity exec --nv cupy.simg python3
```

```
INFO:    Could not find any nv files on this host!
```

That's a Real Pickle



Getting a GPU on ISAAC

```
PROJECT="ACF-UTK0011"
RUNTIME="0:15:00"

srun --account=${PROJECT} --time=${RUNTIME} \
    --qos=campus-gpu --partition=campus-gpu \
    --pty bash -i
```

```
srun: job 21181 queued and waiting for resources
srun: job 21181 has been allocated resources
```

```
singularity exec --nv ~/cupy.simg python3
```

```
Python 3.8.10 (default, Jun  2 2021, 10:49:15)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cupy as cp
>>>
```

CuPy Basics

```
import numpy as np
import cupy as cp

m = 10000
n = 500
np.random.seed(1234)
x_cpu = np.random.rand(m, n)
cp.random.seed(1234)
x_gpu = cp.random.rand(m, n)
```

CuPy Norm

```
np.linalg.norm(x_cpu)
```

```
1290.7480024767005
```

```
x_gpu = x_gpu.astype(np.float32)
cp.linalg.norm(x_gpu)
```

```
array(1291.5363, dtype=float32)
```

CuPy SVD

CPU (float64)

```
import time

t0 = time.perf_counter()
svd = np.linalg.svd(x_cpu)
t1 = time.perf_counter()
print(t1 - t0)
```

```
14.223893312038854
```

GPU (float32)

```
import time

t0 = time.perf_counter()
svd = cp.linalg.svd(x_gpu)
t1 = time.perf_counter()
cp.cuda.stream.get_current_stream().synch
print(t1 - t0)
```

```
4.236048500053585
```

```
# run it again
```

```
0.22288543893955648
```

fmlr

fmlr

- fused matrix library for R
- Novel matrix framework
- Includes a GPU backend (NVIDIA CUDA)
- Very different from base R
- Documentation https://fml-fam.github.io/fmlr/html/index.html

Installation

```
options(repos=structure(c(
  HPCRAN="https://hpcran.org/",
  CRAN="https://cran.rstudio.com/"
)))

install.packages("fmlr")
```

Installation with GPU Support

```
install.packages("fmlr", configure.args="--enable-gpu")
```

fmlr on ISAAC?

```
CUDA_HOME = "/sw/isaac/applications/cuda/11.4.2/rhel8_binary/"
cfargs = paste0("--enable-gpu --with-cuda=", CUDA_HOME)
install.packages("fmlr", configure.args=cfargs)
```

```
/usr/bin/ld: cannot find -lnvidia-ml
```

fmlr on ISAAC

```
NVML_PATH="/sw/isaac/applications/cuda/11.4.2/rhel8_binary/targets/x86_64-linux/lib/stubs
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${NVML_PATH}
LDFLAGS="-L${NVML_PATH}" R
```

```
CUDA_HOME = "/sw/isaac/applications/cuda/11.4.2/rhel8_binary/"
cfargs = paste0("--enable-gpu --with-cuda=", CUDA_HOME)
install.packages("fmlr", configure.args=cfargs)
```

```
/usr/bin/ld: cannot find -lnvidia-ml
```

Running fmlr on ISAAC

- I give up
- Just use singularity

```
singularity exec --nv r.simg R
```

- Images
  - cuda https://hub.docker.com/r/nvidia/cuda/

fmlr Basics

- Uses R6 "reference classes"
- Objects have to be declared (usually)
- Default type is `double`

```
x = cpumat(3, 2)
x$info()
```

```
# cpumat 3x2 type=d
```

```
x$fill_linspace()
x
```

```
1.0000 4.0000
2.0000 5.0000
3.0000 6.0000
```

fmlr and GPUs

```
c = card()
c
```

```
## GPU 0 (NVIDIA GeForce GTX 1070 Ti) 1011/8116 MB - CUDA 11.4 (math mode: default)
```

```
x = gpumat(c, 3, 2)
x$fill_linspace()
x$info()
```

```
# gpumat 3x2 type=d
```

```
x
```

```
1.0000 4.0000
2.0000 5.0000
3.0000 6.0000
```

SVD

```
v = gpuvec(c)
linalg_svd(x, v)
v
```

```
9.5080 0.7729
```

```
x
```

```
-3.7417 -8.5524
0.4218 1.9640
0.6327 0.8598
```

Covariance Matrices

```
x$fill_runif(1234)
x
```

```
0.5823 0.9964
0.4636 0.1182
0.6156 0.2672
```

```
cp = linalg_crossprod(1.0, x)
cp$info()
```

```
# gpumat 2x2 type=d
```

```
cp
```

```
0.9330 0.7995
0.7995 1.0781
```

```
cp = stats_cov(x)
cp
```

```
0.0064 0.0171
0.0171 0.2208
```

What Happens to the Input Data?

```
x
```

```
-0.2519 -0.7767
0.7986 0.1702
-0.5466 0.6065
```

```
dimops_colmeans(x)
```

```
0.0000 -0.0000
```

PCA

```
x$fill_runif(1234)
rot = gpumat(c)
stats_pca(TRUE, FALSE, x, v, rot)
v
```

```
0.4714 0.0709
```

```
rot
```

```
-0.0790 0.9969
-0.9969 -0.0790
```

SVD Timing

CuPy

```
import cupy as cp

m = 10000
n = 500
cp.random.seed(1234)
x_gpu = cp.random.rand(m, n).astype(np.fl

import time
t0 = time.perf_counter()
svd = cp.linalg.svd(x_gpu)
t1 = time.perf_counter()
cp.cuda.stream.get_current_stream().synch
print(t1 - t0)
```

```
4.236048500053585
0.22288543893955648
```

fmlr

```
suppressMessages(library(fmlr))
c = card()
m = 10000
n = 500
x = gpumat(c, m, n, type="float")
x$fill_runif(1234)

s = gpuvec(c, type="float")
u = gpumat(c, type="float")
vt = gpumat(c, type="float")
system.time({
  linalg_svd(x, s, u, vt)
  c$synch()
})
```

```
user  system elapsed
0.158   0.027   0.185
```

Matrix/Vector Operations

Arithmetic

- Dot product
- Addition
- Multiplication
- Crossproducts $x^T x$ and $x x^T$

Matrix Factorizations

- SVD
- Eigendecomposition
- QR/LQ
- Cholesky
- LU

Dimensional Operations

- Row/col-sums/means
- Arithmetic row/col-sweep
- Scale

Matrix Methods

- Transpose
- Inversion
- System solving
- Determinant
- Trace
- Various norms

Singular Value Decomposition

- Four main implementations
  - Divide-and-conquer (`gesdd()`)
  - QR/LQ-based SVD
  - Crossproduct SVD (uses $X^T X$ or $X X^T$)
  - Randomized SVD (Halko et al, *Finding Structure with Randomness*)
- Compute values-only or values and left/right vectors

```
fmlr::linalg_svd(x, s, u = NULL, vt = NULL)
fmlr::linalg_qrsvd(x, s, u = NULL, vt = NULL)
fmlr::linalg_cpsvd(x, s, u = NULL, vt = NULL)
fmlr::linalg_rsvd(seed, k, q, x, s, u = NULL, vt = NULL)
```

fmlr Core Features

- Dense matrix and vector types
  - CPU
  - GPU (CUDA)
  - MPI (2-dimensional block cyclic)
- `float` and `double` fundamental types (some support for `__half` for GPU)
- All data held externally to R
- Single interface covers all types and backends - it Does The Right Thing (TM)

craze

- Syntax like base R
- fmlr object backend
- Same problems as base R
  - implicit copies
  - S4 overhead
  - ...

```
library(craze)

x = matrix(as.double(1:9), 3)
x_gpu = fmlmat(x, backend="gpu")
x_gpu
```

```
## # gpumat 3x3 type=d
## 1.0000 4.0000 7.0000
## 2.0000 5.0000 8.0000
## 3.0000 6.0000 9.0000
```

```
x_gpu %*% x_gpu
```

```
## # gpumat 3x3 type=d
## 30.0000 66.0000 102.0000
## 36.0000 81.0000 126.0000
## 42.0000 96.0000 150.0000
```

Questions?