Jupyter-Based Self-Service Training in OOD for Novel Compute Architectures

A. Jezghani
Georgia Institute of Technology, Atlanta, GA

Georgia Tech

Who We Are

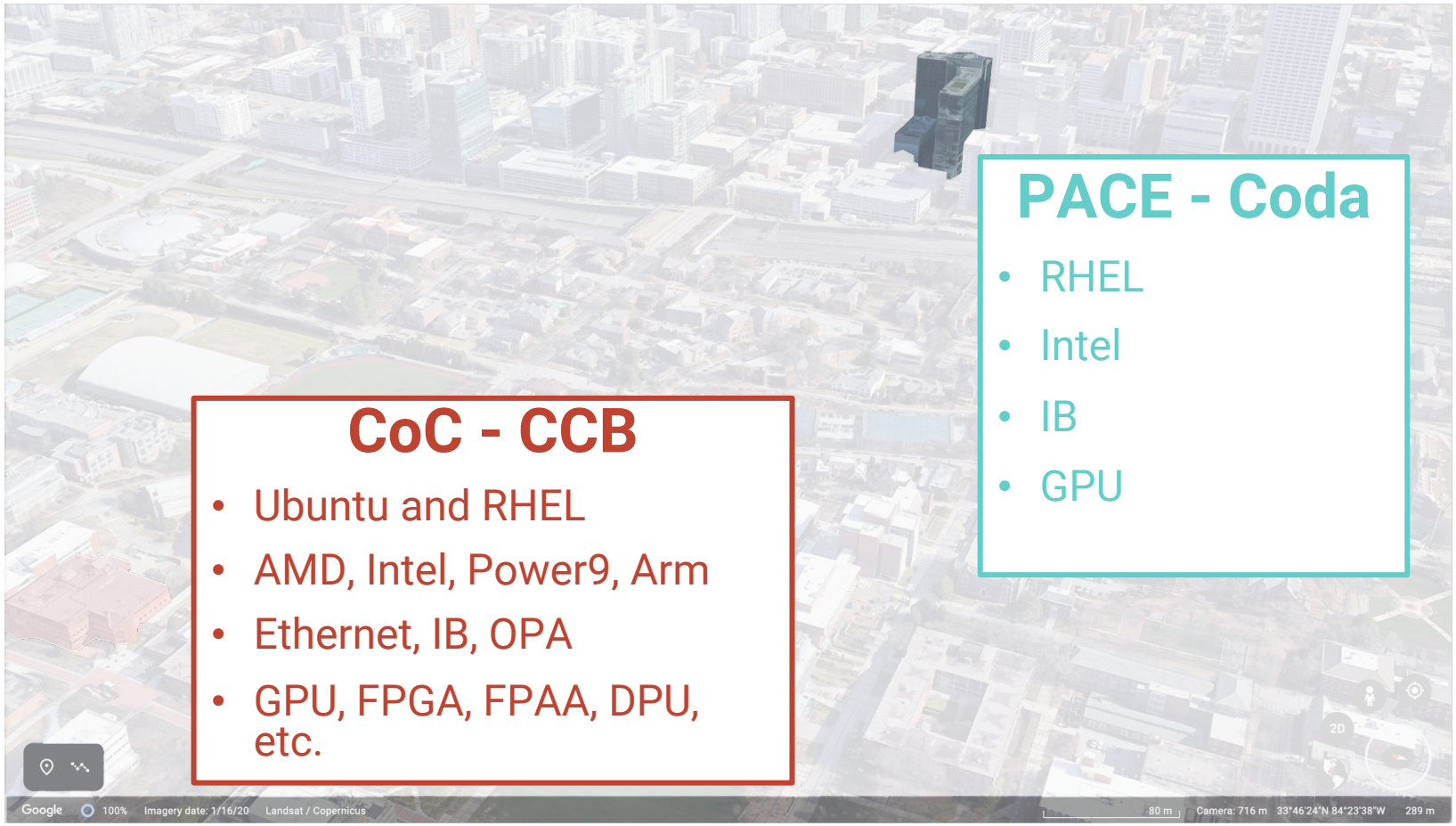
Aaron Jezghani
PACE

Kevin Manalo
PACE (former)

William Powell
CSE

Jeffrey Valdez
IC

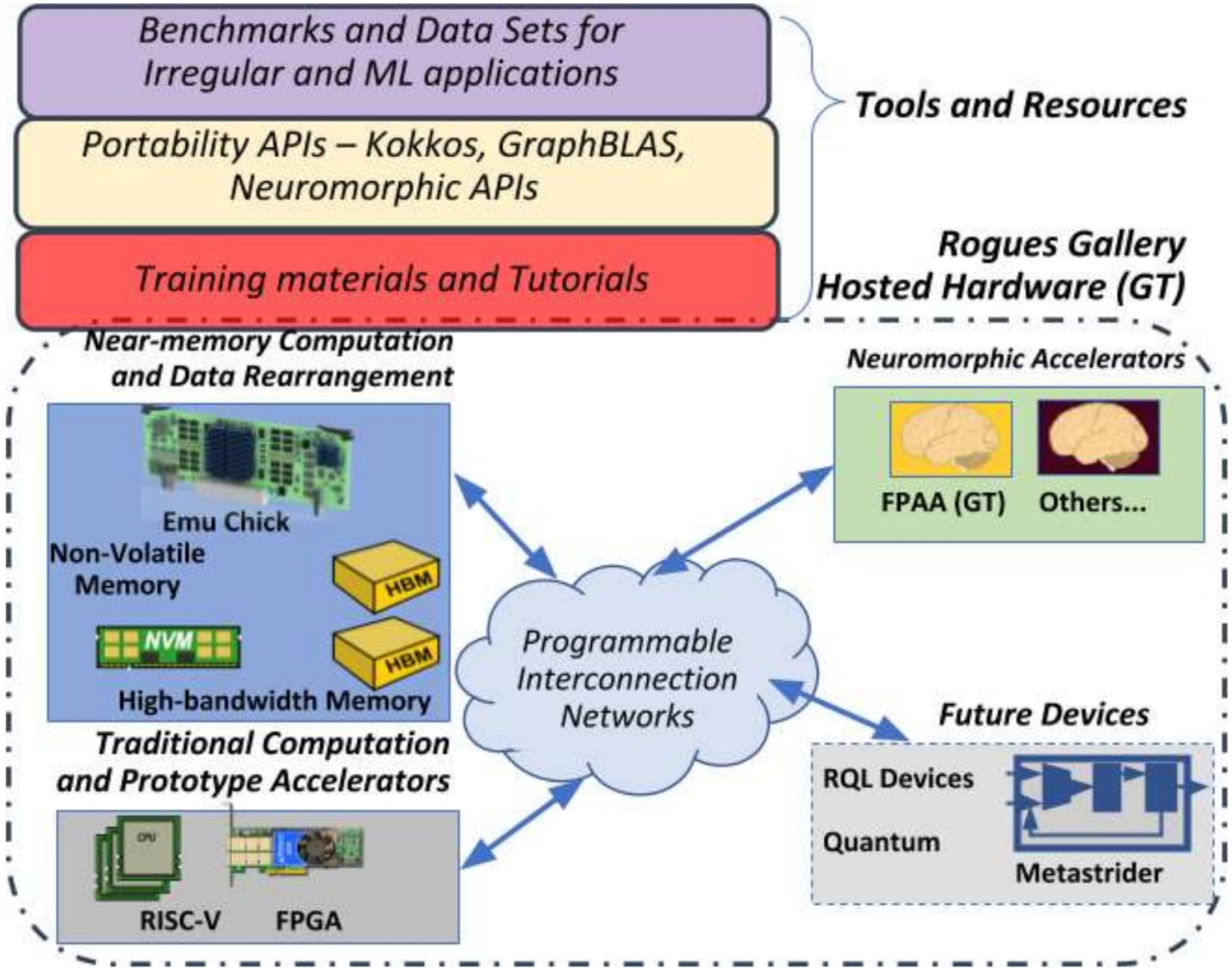
Jeffrey Young
CS



The Rogues Gallery

***NSF MRI award #1828187**: "MRI: Acquisition of an HPC System for Data-Driven Discovery in Computational Astrophysics, Biology, Chemistry, and Materials Science."*

***NSF award #2016701**: "CCRI: Medium: Rogues Gallery: A Community Research Infrastructure for Post-Moore Computing"*



Georgia Tech

The Hurdles of New Technologies and Users

- As exciting as algorithm exploration and software porting might be for veteran users…
  - The “tried-and-true” tools often take several cycles to adapt
    - This happens with updates to traditional architectures too
    - Vendors contribute heavily to upstream, but it can still take time to trickle down
  - Vendor-provided tools are great to use!
    - …but even as a port of something familiar, there’s still a learning curve
    - And these tools may be proprietary!
- Meanwhile, new users are constantly inbound
  - Remote command line access can be intimidating (and feel prohibitive)
  - Even for experienced users, interactive server workflows can be problematic

These Aren't New Issues! But...

- Knowledge transfer can be a difficult process
  - Documentation relies on the "Goldilocks principle"...on a user-by-user basis.
  - Trainings are great but might not be convenient for the users.
  - Command-line self-service tutorials may not include the instructive element available in other forms.
- Many solutions are less than elegant
  - Performance problems (X11 forwarding)
  - Scalability issues (FastX licensing, X2Go resource needs)
  - Less-than intuitive interfaces (SSH tunneling)
  - Client application requirements (e.g. VNC)
  - Local security rules can be an impediment (Port blocking)

Configuring Compute Nodes for OOD

- Not surprisingly, it worked out of the box
  - (since our OOD VM is x86)
- Followed OSC templates that leverage xfce + TurboVNC
  - Aarch64 xfce is available from EPEL, but libturbojpeg needs to be manually built with the appropriate flag
- Everything else pretty much out of the box
  - Given the nature of the user community, the only modular software we've provided at the moment are compilers, math libraries, and MPI

How We Leverage Jupyter

- Apps directory set up by architecture
  - E.g. all Octavius-related apps prefixed with a64fx_*
  - Access to apps controlled via POSIX group, e.g. rg-arm-users
- We provide a dropdown that auto-populates available templates
  - E.g., containerized versions of TF and PyTorch
- Tutorials are merely a port of the standard Jupyter notebook
  - Resource request fixed to match the needs of the tutorial
    - 8 cores on fixed queue and fixed time
  - Job script copies a template notebook to session directory
  - Key with tutorial design is to keep code execution native to mimic typical workflow while leveraging the ipython capabilities for enhancement

Launching Containerized Notebooks

form.yml.erb

```
<%-
  require "csv"
  output= File.read('/net/netscratch/ondemand-jupyter/templates.txt')
  templates = CSV.parse(output)
-%>
# Batch Connect app configuration file
#
# @note Used to define the submitted cluster, title, description, and
#   hard-coded/user-defined attributes that make up this Batch Connect app.
---
title: "Jupyter"
cluster: "crnch"
attributes:
  modules: ""
  extra_jupyter_args: ""
  bc_num_cores:
    help: 'Number of CPUs/Cores.'
    id: bc_num_cores
    label: CPUs/Cores Per Node
    max: 48
    min: 1
    step: 1
    value: 1
    widget: number_field
    cacheable: false
```

```
  custom_template:
    label: "Starting Interface/Template"
    widget: select
    options:
      - [ "None - use Jupyter to select file to work on", ""]
      - [ "This isn't real, because why not", ""]
    <%- templates.each do |a| -%>
      - [ "<%= a[0] %>", "<%= a[1] %>" ]
    <%- end -%>
  custom_queue:
    widget: select
    id: bc_queue
    label: "Octavius Partition"
    cacheable: false
    options:
      - [ "Debug: up to 4 nodes and 1 hour", "rg-arm-debug"]
      - [ "Short: up to 16 nodes and 2 hours", "rg-arm-short"]
      - [ "Long: up to 16 nodes and 6 hours", "rg-arm-long"]
form:
  - extra_jupyter_args
  - modules
  - custom_queue
  - bc_num_hours
  - bc_num_slots
  - bc_num_cores
  #- bc_email_on_started
  - custom_template
```

Launching Containerized Notebooks

submit.sh.erb

```
# Launch the Jupyter Notebook Server
<%- if context.custom_template == "TensorFlow.ipynb" -%>
  set -x
  export ONDEMAND_FILE="${odir}/<%=context.custom_template%>"
  export TEMPLATE_FILE="${sdir}/<%= context.custom_template %>"
  # Launch the Jupyter Notebook Server
  cp $ONDEMAND_FILE $TEMPLATE_FILE
  TEMPLATE_FILE=${TEMPLATE_FILE#*ondemand/}
  singularity run "${odir}"/tensorflow-arm.sif jupyter notebook --debug --
config="${CONFIG_FILE}" <%= context.extra_jupyter_args %> \
    --NotebookApp.default_url="/tree/ondemand/$TEMPLATE_FILE"
<%- elsif context.custom_template == "PyTorch.ipynb" -%>
  set -x
  export ONDEMAND_FILE="${odir}/<%=context.custom_template%>"
  export TEMPLATE_FILE="${sdir}/<%= context.custom_template %>"
  # Launch the Jupyter Notebook Server
  cp $ONDEMAND_FILE $TEMPLATE_FILE
  TEMPLATE_FILE=${TEMPLATE_FILE#*ondemand/}
  singularity run "${odir}"/pytorch-arm-neoverse-n1_r21.10-torch-1.9.0-openblas-jupyter.sif
jupyter notebook --debug --config="${CONFIG_FILE}" <%= context.extra_jupyter_args %> \
    --NotebookApp.default_url="/tree/ondemand/$TEMPLATE_FILE"
]
<%- else -%>
  set -x
  jupyter notebook --config="${CONFIG_FILE}" <%= context.extra_jupyter_args %>
<%- end -%>
```

Arm Compiler Tutorial

script.sh.erb

```
#!/usr/bin/env bash

# Benchmark info
echo "TIMING - Starting main script at: $(date)"

# Get Running Script Directory
get_script_dir () {
   SOURCE="${BASH_SOURCE[0]}"
   while [ -h "$SOURCE" ]; do
      DIR="$( cd -P "$( dirname "$SOURCE" )" && pwd )"
      SOURCE="$( readlink "$SOURCE" )"
      [[ $SOURCE != /* ]] && SOURCE="$DIR/$SOURCE"
   done
   $( cd -P "$( dirname "$SOURCE" )" )
   pwd
}
sdir="$(get_script_dir)"
export SDIR=$sdir
odir="/net/projects/notebook/ondemand-jupyter/"

# Set working directory to home directory
cd "${HOME}”

#
# Start Jupyter Notebook Server
#

<%- unless context.modules.blank? -%>
# Purge the module environment to avoid conflicts
module purge

# Load the require modules
module load <%= context.modules %>

# List loaded modules
module list
<%- end -%>

# Benchmark info
echo "TIMING - Starting jupyter at: $(date)"

# Launch the Jupyter Notebook Server
set -x
export ONDEMAND_FILE="${odir}/arm_compilers_evaluation.ipynb"
export TEMPLATE_FILE="${sdir}/arm_compilers_evaluation.ipynb"
cp $ONDEMAND_FILE $TEMPLATE_FILE
TEMPLATE_FILE=${TEMPLATE_FILE#*ondemand/}
jupyter notebook --config="${CONFIG_FILE}" <%= context.extra_jupyter_args %> \
  --NotebookApp.default_url="/tree/ondemand/$TEMPLATE_FILE"
```

Arm Compiler Tutorial

jupyter arm_compilers_evaluation (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [5]:
```
%%bash --err naive

ml arm21
cd arm-sve-tools/01_Compiler/01_Naive
for compiler in {arm,cray,gnu}
do
    make clean >/dev/null
    make run COMPILER=${compiler} 2>&1
done | tee /dev/stderr 2> >(sed -n 's&.*multiply took: \([0-9.]*\).*&\1&p' >&2)
```

```
------------------------------------------------------
armclang --version
Arm C/C++/Fortran Compiler version 21.1 (build number 1069) (based on LLVM 12.0.1)
Target: aarch64-unknown-linux-gnu
Thread model: posix
InstalledDir: /net/projects/tools/aarch64/rhel-8/arm-allinea/21.1/arm-linux-compiler-21.1_Generic-AArch64_RHEL-8_aarch64-linux/bin
armclang++ -Rpass=\(loop-vectorize\) -Rpass-missed=\(loop-vectorize\) -o mm_arm_def.exe mm.cpp
------------------------------------------------------
------------------------------------------------------
armclang --version
Arm C/C++/Fortran Compiler version 21.1 (build number 1069) (based on LLVM 12.0.1)
Target: aarch64-unknown-linux-gnu
Thread model: posix
InstalledDir: /net/projects/tools/aarch64/rhel-8/arm-allinea/21.1/arm-linux-compiler-21.1_Generic-AArch64_RHEL-8_aarch64-linux/bin
armclang++ -Rpass=\(loop-vectorize\) -Rpass-missed=\(loop-vectorize\) -Ofast -mcpu=native -o mm_arm_opt.exe mm.cpp
mm.cpp:42:13: remark: loop not vectorized [-Rpass-missed=sve-loop-vectorize]
            for (k= 0; k < m; ++k){
```

Read the results into a dataframe

The values we stored in 'naive' are stored as a new-line delimited string, so we will use numpy.fromstring to read them into an array, reshape into a 3x3 2D array, transpose, and then create a dataframe with appropriate index and column labels. The restructuring of the data is so that we can take advantage of the "simpler" plotting calls in pandas versus matplotlib.

In [6]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

naive_df = pd.DataFrame(np.transpose(np.fromstring(naive,dtype=float,sep='\n').reshape(3,3)),
                        index=['Def','Opt','NoVec'],columns=['Arm','Cray','Gnu'])
print(naive_df)
```

```
         Arm   Cray    Gnu
Def    2.582  0.710  3.758
Opt    1.224  0.708  1.268
NoVec  1.225  0.710  1.269
```

jupyter arm_compilers_evaluation (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

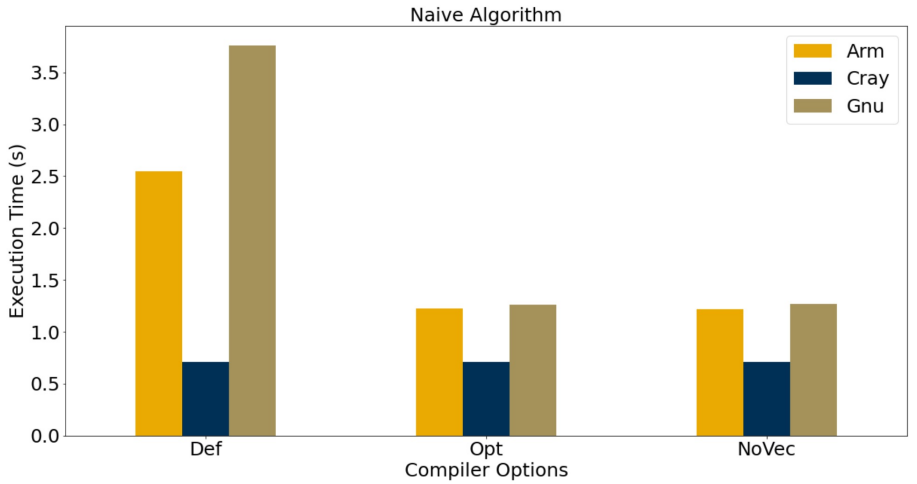
Run Markdown

```
         Arm   Cray    Gnu
Def    2.544  0.710  3.758
Opt    1.224  0.708  1.265
NoVec  1.215  0.710  1.268
```

Make a bar plot with the results

Given the nature of the data, we can summarize the results of the different compilers and options using a bar plot. Given a pandas dataframe, we can run the plot.bar function to handle each of our different combinations (and set the plot size and labels using pyplot calls).

In [4]:
```
naive_df.plot.bar(rot=0,figsize=(20,10),color=['#EAAA00','#003057','#A4925A'])
plt.title('Naive Algorithm',fontsize=25)
plt.legend(prop={'size': 25})
plt.ylabel('Execution Time (s)',fontsize=25)
plt.yticks(fontsize=25)
plt.xlabel('Compiler Options',fontsize=25)
plt.xticks(fontsize=25)
plt.show()
```



Conclusion

- Onboarding new users can be a challenge
- OOD is a powerful platform to lower the barrier to entry
  - Provides full cluster access while improving GUI/interactive experiences
- Can leverage applications and example code to produce self-service tutorials
- Similarly, can leverage the dynamic content capabilities to abstract complex workflows (such as containerized solutions)
- Expanding Rogues Gallery OOD capabilities and looking to grow the community
  - Encourage innovative contributions to A64FX using Octavius

Bored today/tomorrow? Check out the CRNCH Summit 2022 videos