

TRAINING & BEST PRACTICES TO DEVELOP PORTABLE YET PERFORMANT CODE

Sunita Chandrasekaran

Assistant Professor, University of Delaware

Dept. of Computer & Information Sciences

schandra@udel.edu



LEADERSHIP
COMPUTING
FACILITY



SIAM PP MS42: Feb 14, 2020

CREATE BETTER SOFTWARE!

- Programmer Productivity
- Software Sustainability
- Facilitate Reproducibility of Computational Results
- Incremental Code Improvement
- Re-usability of code
- Reduced development effort



image credit: <https://www.pnas.org/content/115/20/5042>

BEST PRACTICES - 7 OF MANY!

- Best Practice #1 Profiling
- Best Practice #2: Systematic Testing
- Best Practice #3: Report bugs
- Best Practice #4: Automate
- Best Practice #5: Document
- Best Practice #6: Pair Programming
- Best Practice #7: Open Source but...



My group in action :-)



Computational
Research and
Programming Lab

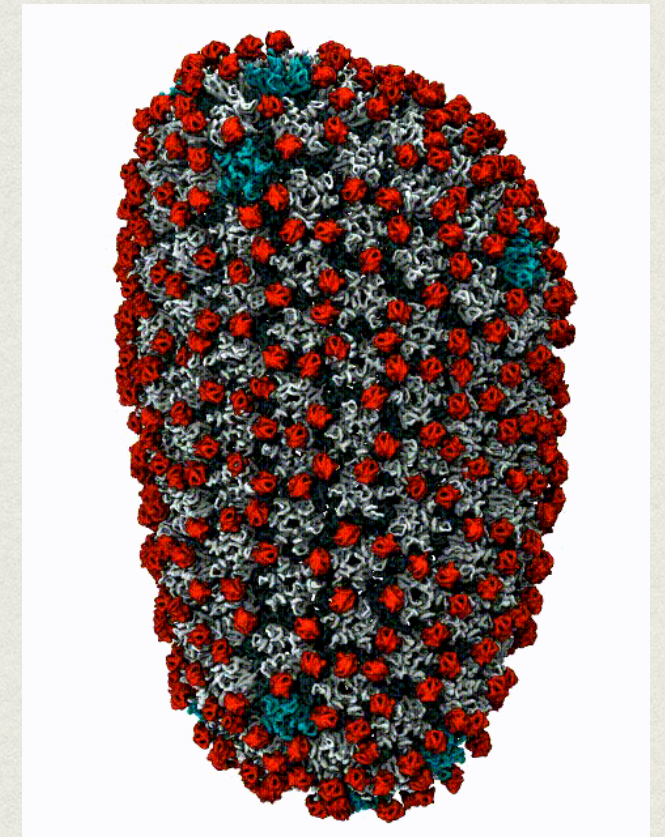
BEST PRACTICE #1: PROFILING

ACCELERATING A BIOPHYSICS PROBLEM ON GPUS

- Nuclear Magnetic Resonance (NMR) is a vital tool in structural biology and biochemistry
- NMR spectroscopy measures chemical shifts
- Predicting chemical shift has important uses in scientific areas such as drug discovery

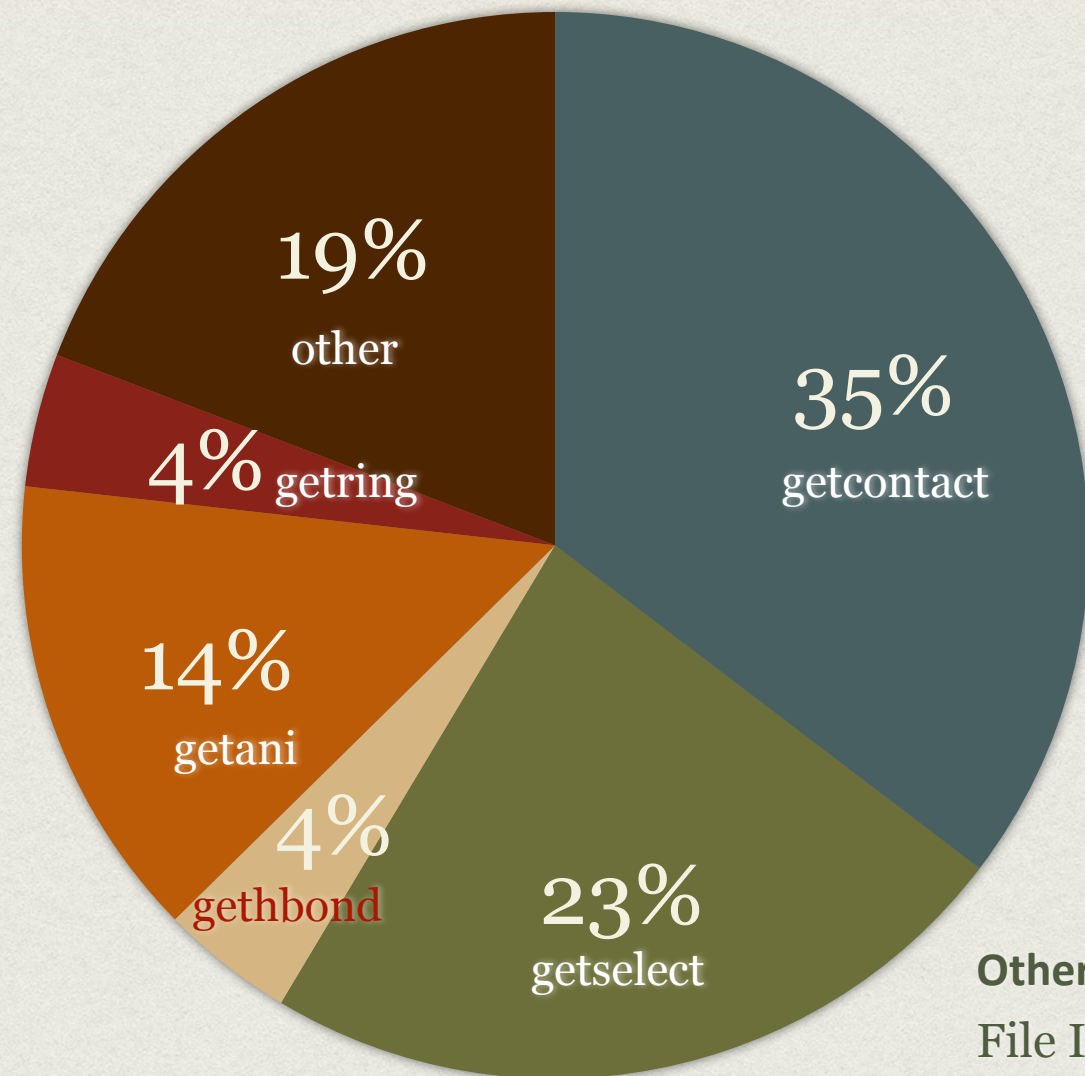
Our goal:

- Accelerate the prediction of chemical shift
 - To enable execution of multiple chemical shift predictions repeatedly
 - To allow chemical shift predictions for larger scale structures



SERIAL PROFILE VISUAL

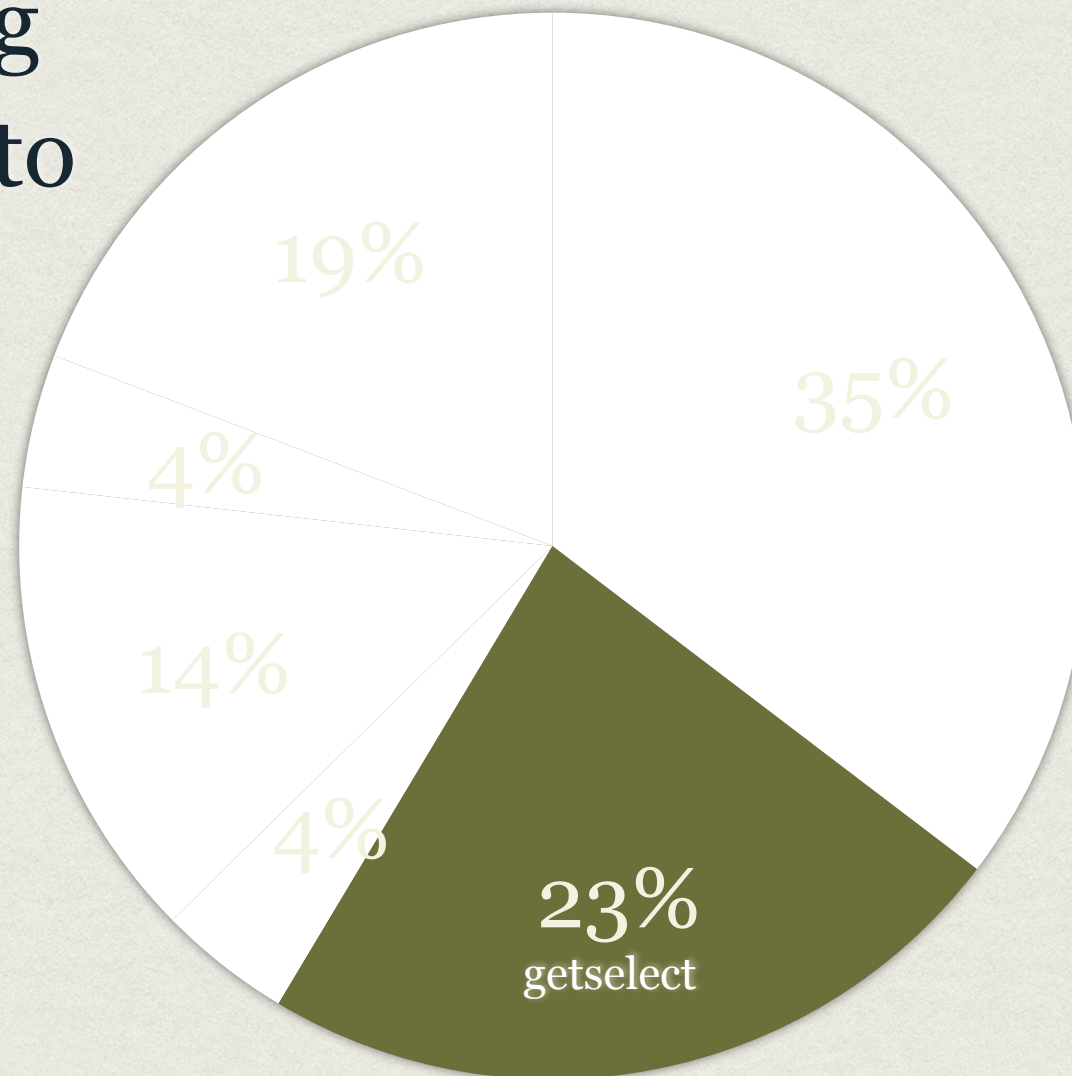
- Serial code profiling
NVProf
- Obtained large overview
without needing to read
thousands of lines of
code
- Identified hotspots
within the code
- 2 undergrads - 1 year
project



Other Contains:
File I/O
PDB Structure
Initialization

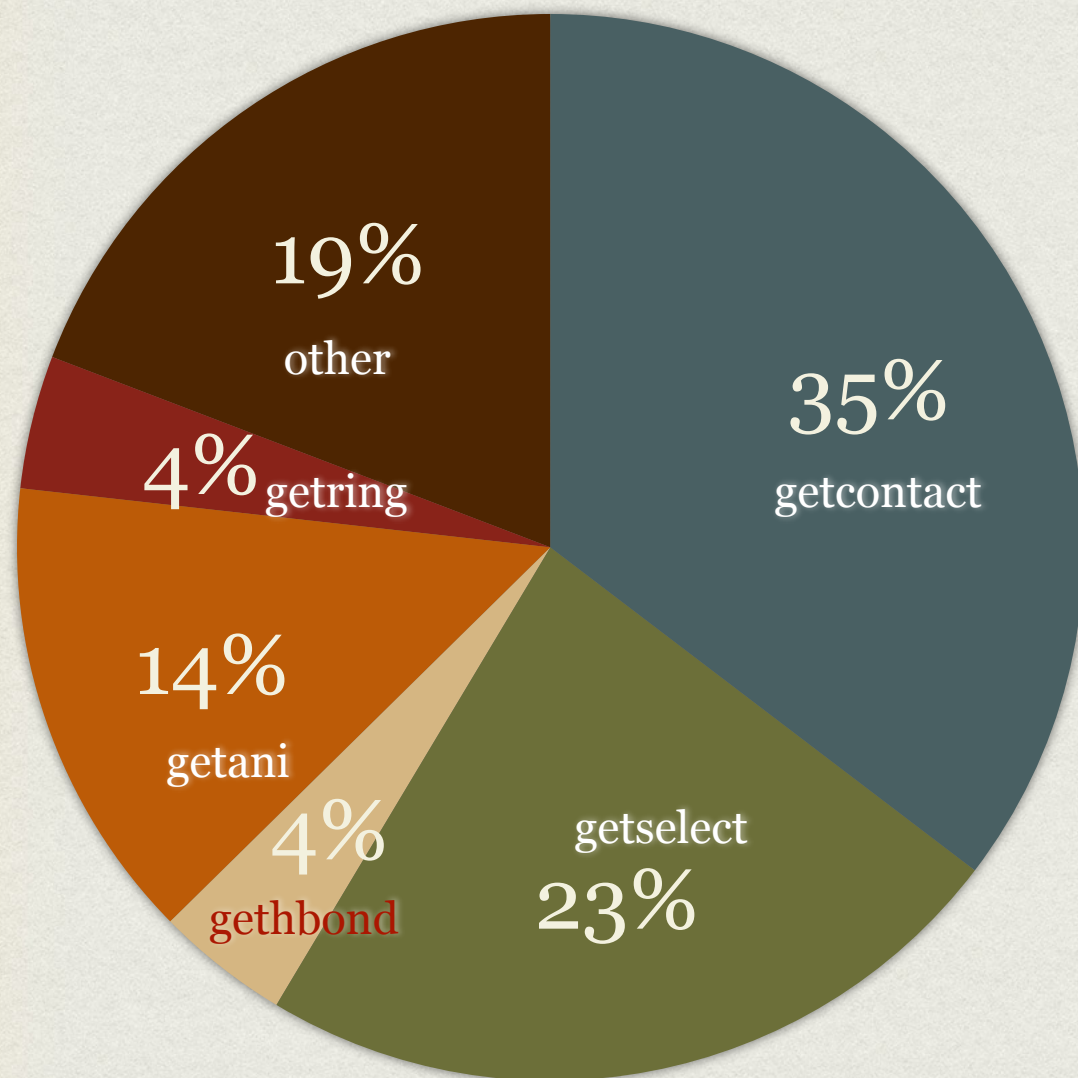
SERIAL CODE CLEAN UP!!

- `getselect()`
- Looking into cleaning the serial code prior to parallelizing it

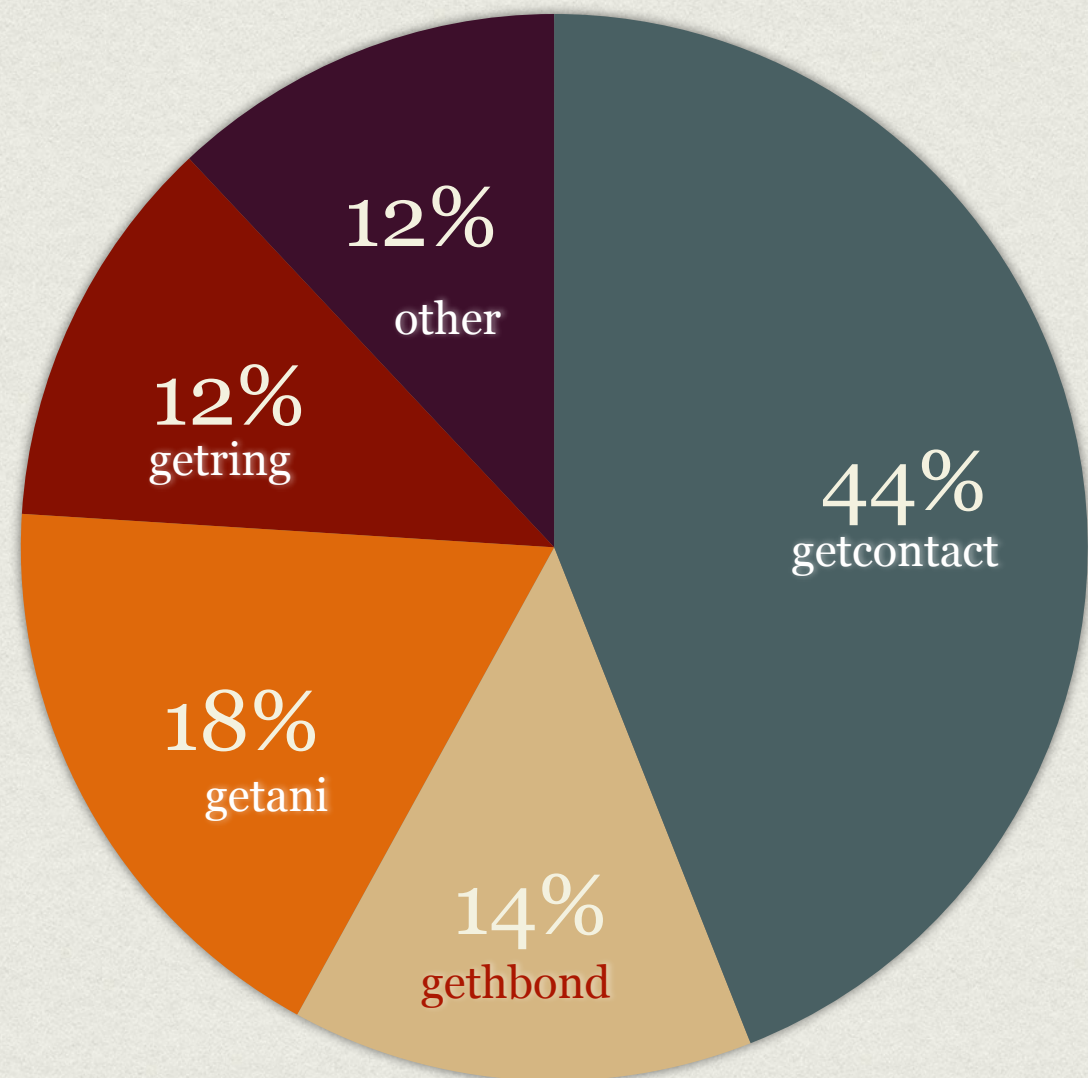


PROFILER MAGIC!

Before



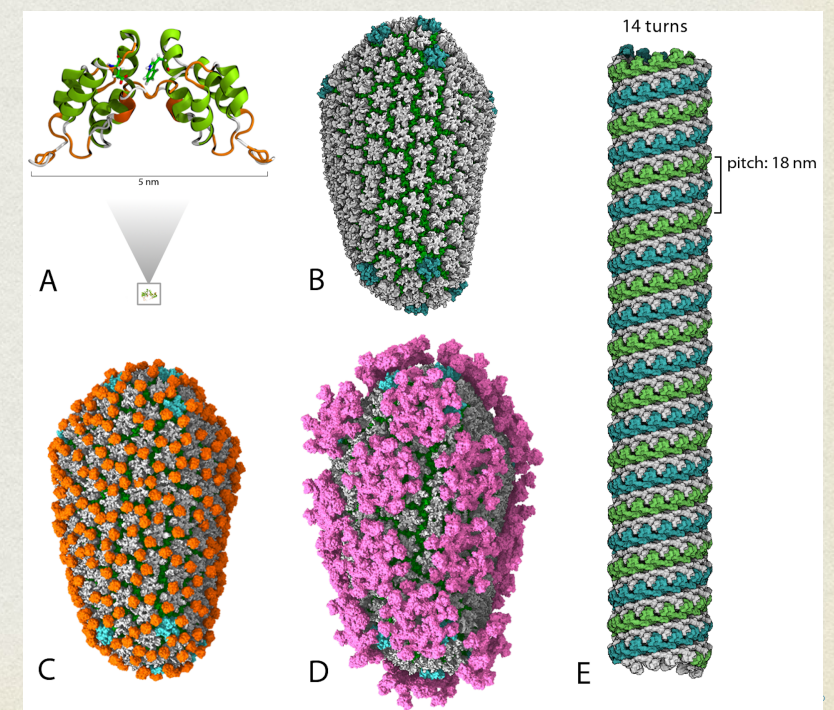
After



OPENACC-GPU PERFORMANCE RESULTS

	100K	1.5M	5M atoms	6.8M	11.3M
Serial (Unoptimized)	167.11s	572.01s	3547.07s	7 hours	14 hours (estimate)
Serial (Optimized)	53.57s	196.12s	2003.6s	1510.71s	2614.4s
Multicore	4.67s	32.82s	116.66s	153.8s	146.06s
NVIDIA P40	3.47s	17.15s	56.2s	78.57s	72.55s
NVIDIA V100	3.11s	13.62s	39.79s	49.63s	47.71s

https://github.com/UD-CRPL/ppm_one
 Biorxiv: <https://www.biorxiv.org/content/10.1101/2020.01.12.903468v1.full.pdf>
 [Under Review]



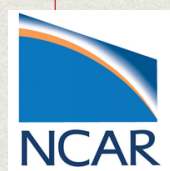
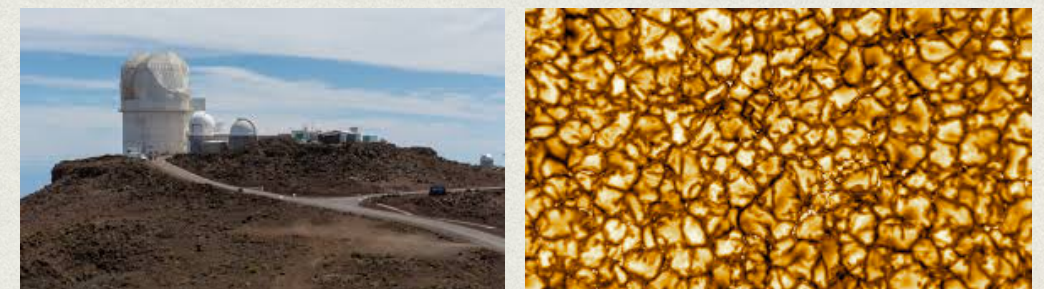
BEST PRACTICE #2: SYSTEMATIC TESTING

PREPARE MURAM (MAX PLANCK UNIVERSITY OF CHICAGO RADIATIVE MHD) FOR NEXT-GENERATION SYSTEMS

- Primary solar model for simulations of upper convection zone, photosphere, and corona
- Typical runs across ~10,000 CPU cores, runs ~100x slower than real-time
- New physics will also require more computation power
- **Goals: Port and Accelerate MURaM to GPU, achieve real-time simulation, prepare for new physics**



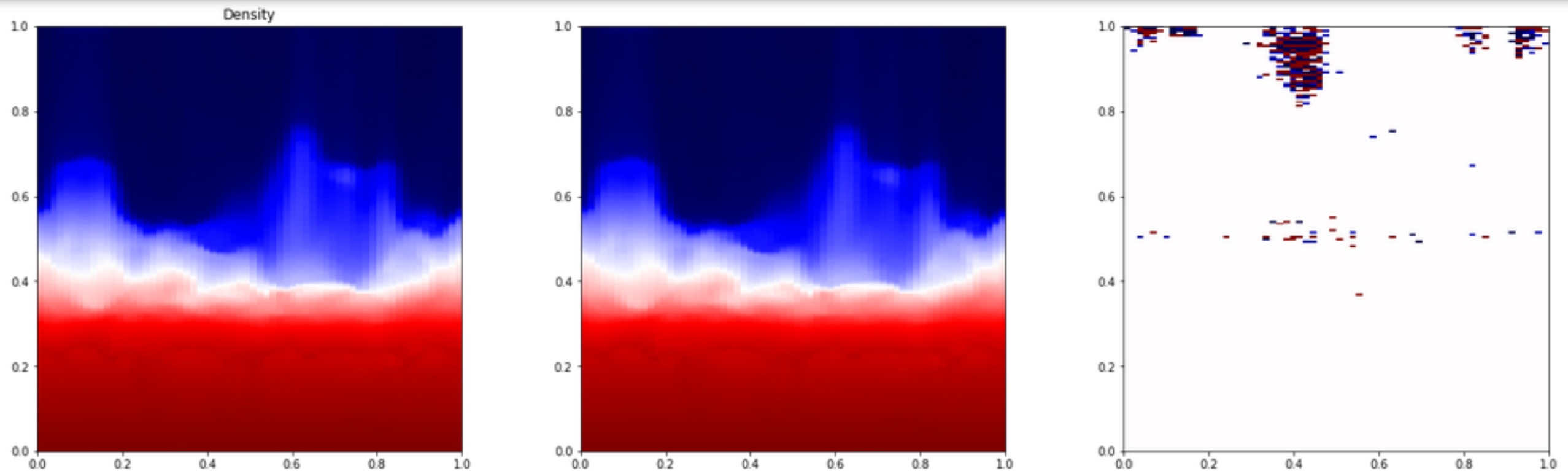
*Comprehensive model of entire life cycle of a solar flare
(Cheung et al 2018)*



The Daniel K. Inouye Solar Telescope (DKIST), a ~\$300M NSF investment, is expected to advance the resolution of ground based observational solar physics by an order of magnitude.



ERROR IN DENSITY CALCULATION



Difference in density between reference and the test runs

UNRESOLVABLE BUG

- **LOC:** 20-30K Lines of Code
- **Team:** 4 Computer Scientists and 2 Solar physicists
- **Tools used:** Valgrind, GDB, Python notebooks, PCAST (PGI tool)
- **Test bed:** 3 different compilers and 2 different platforms
- **Version Control:** GitHub
- **Time spent:** 3 months and the clock is still ticking!!!!

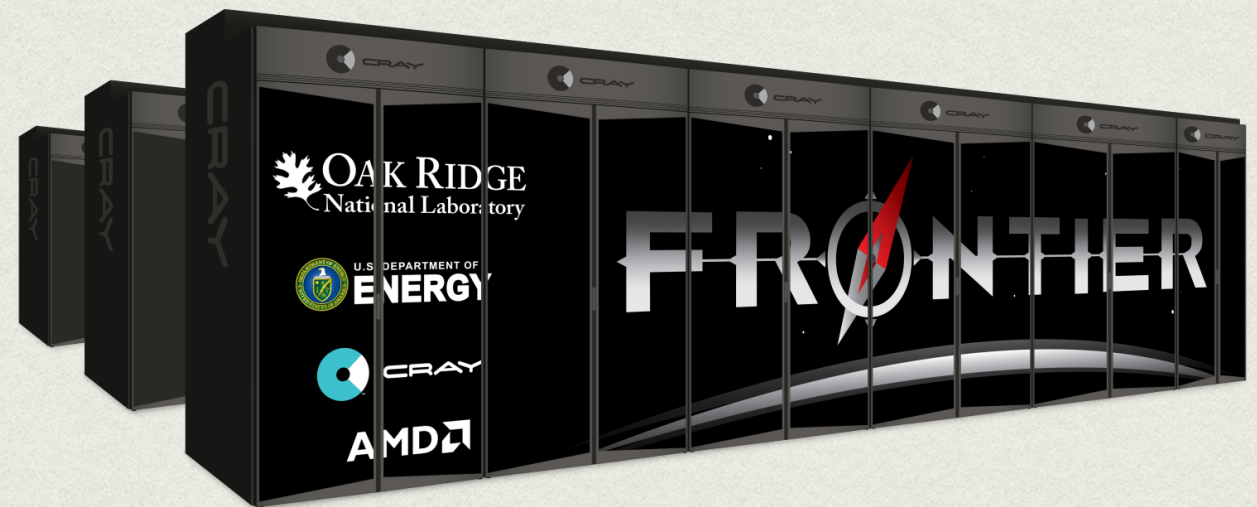
LESSONS LEARNT

- **Work in small steps:** Make incremental changes
- **Testing:** Weekly/bi-weekly testing
- **DOCUMENT!!!**
- **Test bed:** 3 diff. compilers, more than 2 hardware setup, more than 2 versions of a compiler (perhaps?)
- **Optimization:** ONLY after a thorough verification of the serial version
- **Verification:** Use tools to verify divergence of results between CPU and GPU

BEST PRACTICE #3: REPORT BUGS

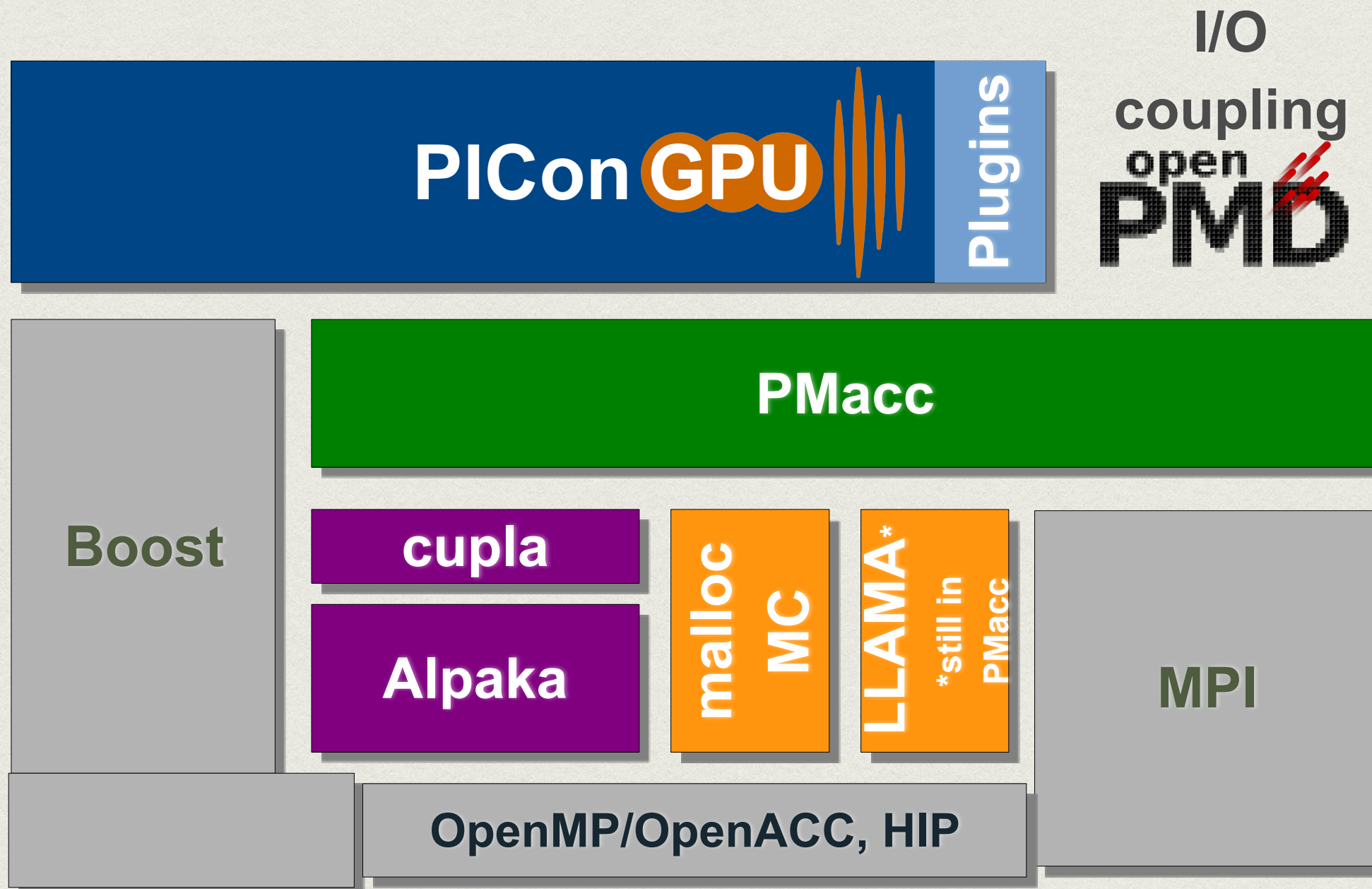
CAAR-ORNL-PICONGPU PROJECT

- Preparing PIConGPU, a plasma Physics application for the upcoming exascale system - Frontier



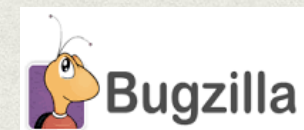
CAAR Project in Collaboration with COE
(AMD + Cray) developers

OUR OPEN HPC SOFTWARE STACK



REPORT BUGS!

- **Code Review:** author of a PR cannot merge his/her own PR)
- **Report bugs:** help improve compilers
- **Reproducible code:** Useful to debug
- **REPORT:** Workarounds OK but "REPORT" bugs
- Report bugs via a ticket system (say Trac – wiki + issue tracker) and not via email – **PLEASE!** ☺
 - The bug and its fix got to be recorded
 - Documented
 - Code changes to be tracked
- **Time critical bugs** - communicate with the developers directly



BEST PRACTICE #4: DOCKER

COLLABORATIVE SOFTWARE DEVELOPMENT

- Tools/Platforms such as Docker, Container, GitHub
- Dramatically reduces barrier to collaboration
- Google slides

The screenshot displays the NVIDIA NGC Accelerated Software interface. The main content area shows the 'OpenACC Training Materials' page. The page includes a table with columns for Publisher, Built By, Latest Tag, and Modified. The description states that these materials were developed as a collaboration between the University of NVIDIA Corporation and OpenACC.org. The page also features a 'Pull Command' section with a Docker command: `docker pull nvcr.io/hpc/openacc-training-materials:20.1.1`. The right sidebar shows 'Module 6 - Loop Optimizations with OpenACC', which is the last 'core' module. It describes the module's focus on loop clauses and provides a list of topics to be covered: Seq/Auto clause, Independent clause, Reduction clause, Collapse clause, Tile clause, and Gang Worker Vector. The bottom of the page shows the 'Overview' tab selected, with a description of the training materials and a link to the 'CONTRIBUTING.md' file.

NVIDIA NGC | ACCELERATED SOFTWARE

OpenACC Training Materials

Publisher	Built By	Latest Tag	Modified
NVIDIA	PGI Compile...	20.1.1	February 5, 2...

Description

These training materials have been developed as a collaboration between the University of NVIDIA Corporation and are provided free of charge by OpenACC.org.

Labels

HPG High Performance Computing OpenACC PGI Supercomputing

Pull Command

```
docker pull nvcr.io/hpc/openacc-training-materials:20.1.1
```

Module 6 - Loop Optimizations with OpenACC

[labs/module6](Module 6) is the last "core" module. After Module 6, we expect students to be able to begin parallelizing their own personal code with OpenACC with a good amount of confidence. The remaining modules after this point are considered to be "advanced" modules, and are optional, and some may only be applicable to specific audiences. Module 6 is all about loop clauses. This module is meant to be very visual, so that students can get a good sense of exactly how each clause is affecting the execution of their loop.

Topics that will be covered are as follows:

- Seq/Auto clause
- Independent clause
- Reduction clause
- Collapse clause
- Tile clause
- Gang Worker Vector

This module touches on each of the loop clauses, show how they look within code, and give a visual representation of it. The gang/worker/vector will most likely be the lengthiest section in this module, just because it is the most complex. Also, in the lab section of Module 6, we will make our final optimization to our Laplace code by utilizing loop optimizations and gang/worker/vector.

Running the Docker container

The code labs have been written using Jupyter notebooks and a Dockerfile has been built to simplify deployment. In order to serve the docker instance for a student, it is necessary to expose port 8000 from the container, for instance, the following command would expose port 8000 inside the container as port 8000 on the lab machine:

```
$ docker run --gpus all -it --rm -p 8000:8000 nvcr.io/hpc/openacc-training-materials:20.1.1
```

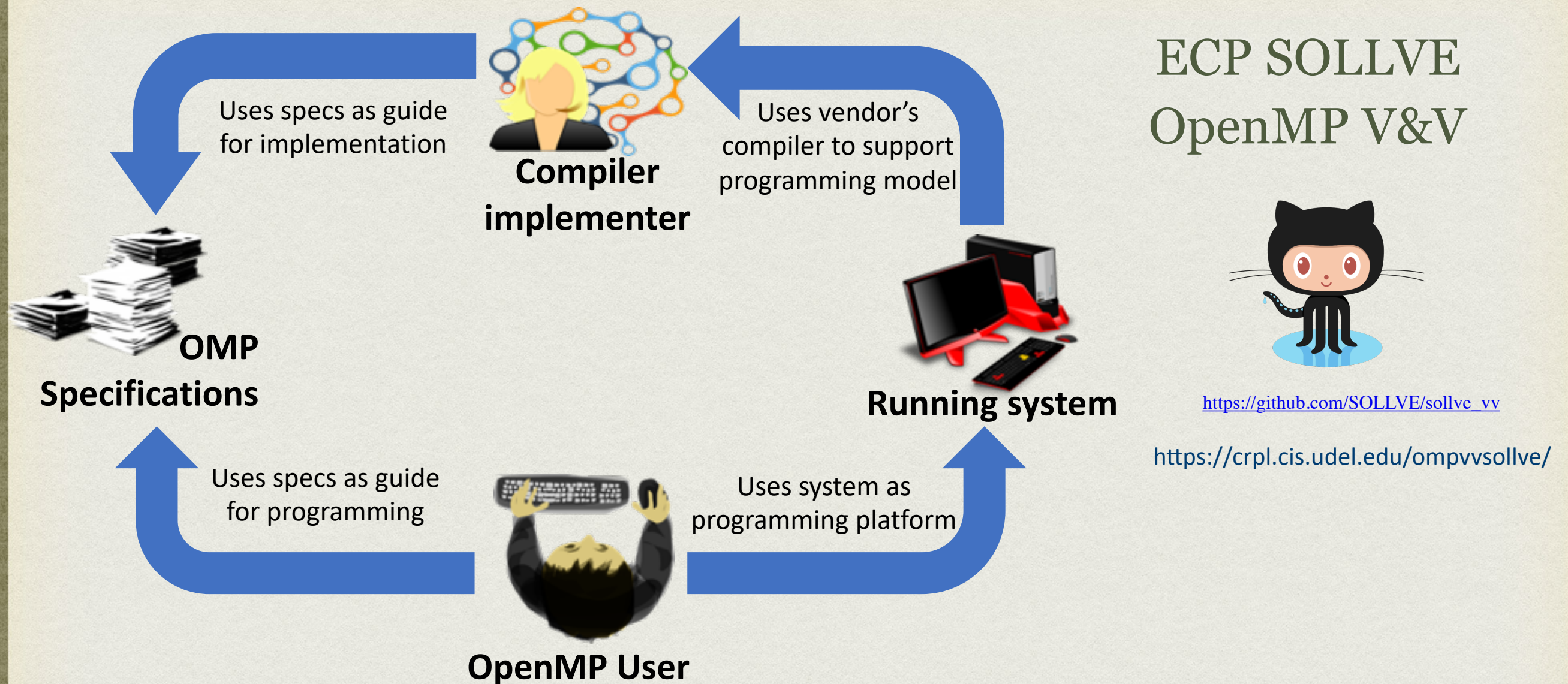
OpenACC Official Training Materials

These training materials have been developed as a collaboration between the University of Delaware and NVIDIA Corporation and are provided free of charge by OpenACC.org. Please see [CONTRIBUTING.md](#)

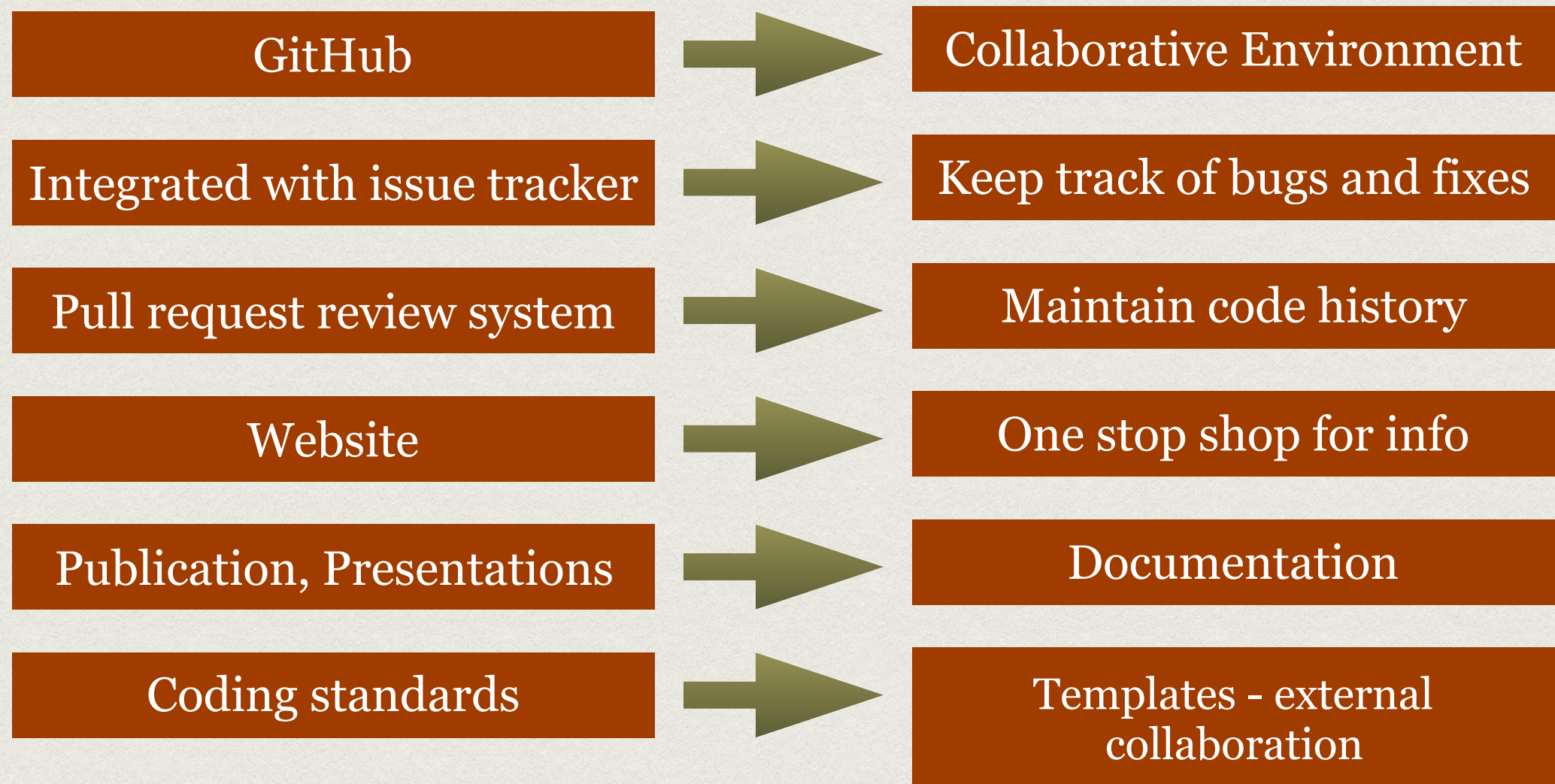


BEST PRACTICE #4: AUTOMATE

OPENMP VALIDATION & VERIFICATION TESTSUITE



ECP SOLOLVE OPENMP V&V



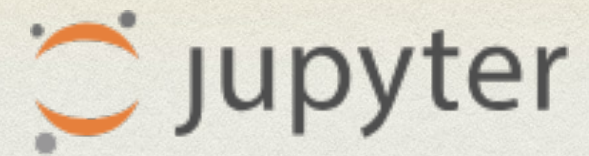
FURTHER IMPROVEMENTS

- Currently implementing a CI/CD ecosystem
 - Jenkins Automation System
 - Build, Deploy and Automate
 - Scale software quality
 - Save time
 - Prone to lesser errors
- More specifically to the V&V
 - Decoupling offloading tests from the rest of the code
 - Clear pass/fail messages

BEST PRACTICE #5: DOCUMENT (pretty pls! :-))

DOCUMENTATION

- IPython notebooks
- Embed code as part of documentation
- Probability of a programmer updating the documentation when the code changes is high!
- Useful for training and education

A screenshot of a Jupyter Notebook interface. The top bar shows "jupyter 2D-Heat" and "Last Checkpoint: 44 minutes ago (unsaved changes)". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations and execution. The notebook contains two code cells. The first cell (In [8]) contains a shell command to compile a C program:

```
In [8]: # To be sure we see some output from the compiler, we'll echo out "Compiled Successfully!"
#(if the compile does not return an error)
!make heat_0

pgcc -O3 heat.c -o heat_0
```

 The second cell (In [9]) contains a shell command to run the compiled program with timing information:

```
In [9]: # Execute our single-thread CPU-only Jacobi Iteration to get timing information. Make sure you compiled successfully in
# above cell first.
!./heat_0 1024 1024 20000 output.dat
```

 Below the code, the output shows the execution time: "Time for computing: 46.59 s". A text block states: "After each step, we will record the results from our benchmarking and correctness tests in a table like this one:". Below this is a table with the following data:

Step	Execution	ExecutionTime (s)	Speedup vs. 1 CPU Thread
1	CPU 1 thread	44.85	

 A "Profiling" section follows, with the instruction: "Run the following cell to profile and view the code:". The third code cell (In [*]) contains a shell command to run a profiler:

```
In [*]: %%bash
pgprof -o serial.prof --cpu-profiling-scope instruction ./heat_0 1024 1024 20000 output.dat
pgprof -i serial.prof
```

 The output of the profiler is displayed as a table:

```
===== CPU profiling result (bottom up):
Time(%)   Time   Name
99.72%    13.09s  runTest(int, char**)
99.72%    13.09s  | main
99.72%    13.09s  | ???
```

 The bottom of the interface shows a file browser with several files: "CC_2.5-c....zip", "BarbaOEGlobal2018.pdf", "2D-Heat.ipynb", "2018_SIAM_Huebl_J....pdf", and "science.bib".

BEST PRACTICE #6: PAIR PROGRAMMING

HACKATHONS

- Profile the code, understand performance bottlenecks
- Start with a smaller kernel
- Use a simple system to compile/execute the code for starters
- Identify a good starting point
- Debugging - “when in doubt, comment it out, re-test”
- <https://www.gpuhackathons.org/>



Chandrasekaran, Sunita, Guido Juckeland, Meifeng Lin, Matthew Otten, Dirk Pleiter, John E. Stone, Juan Lucio-Vega, Michael Zingale, and Fernanda Foertter. "Best Practices in Running Collaborative GPU Hackathons: Advancing Scientific Applications with a Sustained Impact." *Computing in Science & Engineering* 20, no. 4 (2018): 95-106.

BEST PRACTICE #7: OPEN SOURCE BUT...

OPEN SOURCE IS GREAT BUT....

- Not sustainable unless there is a community that will help with the sustainability
- Need help
 - From sponsors, funding organizations, interested vendors
 - To ensure continuity of software maintenance beyond the funded project period

SUMMARY

- Best Practice #1 Profiling
- Best Practice #2: Systematic Testing
- Best Practice #3: Report bugs
- Best Practice #4: Automate
- Best Practice #5: Document
- Best Practice #6: Pair Programming
- Best Practice #7: Open Source but...



My group in action :-)



Computational
Research and
Programming Lab

ACKNOWLEDGEMENTS

- **CRPL:** Sanhu Li, Mauricio Ferrato, Eric Wright, Mayara Gimenes, Fabian Mora, Matt Leinhauser, Thomas Huber, Matt Stack, Brad Atmiller, Hayden Carter, Kyle Friedline, Josh Davis, Jose Diaz
- **PPMONE:** Juan Perilla, Eric Wright, Mauricio Ferrato, Robert Searles, Alexander Bryer
- **MURAM :** Rich Loft, Matthias Rempel, Shiquan Su, Supreeth Suresh, Cena Miller, Raghu Kumar
- **CAAR:** Alex Debus, Thomas Kluge, Rene Widera, Sergei Bastrakov, Klaus Steiniger, Marco Garten, Matthias Werner, Jeff Kelling, Matt Leinhauser, Jeff Young, Josh Davis, Jose Diaz, Ronnie Chatterjee, Axel Huebl, Ronnie Chatterjee, Guido Juckeland, Michael Bussman
- **ECP SOLLVE:** Jose Diaz, Josh Davis, Thomas Huber, Swaroop Pophale, Oscar Hernandez, David Bernholdt

