COMPUTATIONAL AND DATA-INTENSIVE (CDI) RESEARCH -BEST PRACTICES

Sunita Chandrasekaran

Assistant Professor, University of Delaware

Dept. of Computer & Information Sciences

CRPL Suita Chardrasekaran

ELAWARE,

Nemours. **OpenACC**

schandra@udel.edu

Xpert Network, July 15, 2021

National Laboratory





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...

BEST PRACTICE #1: PROFILING





ACCELERATING A BIOPHYSICS PROBLEM ON GPUS (in collaboration with Prof. Juan R. Perilla, UD)

- 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

RPL

 2 undergrads - 1 year project





OPENACC-GPU PERFORMANCE RESULTS

	100K atoms	1.5M atoms	5M atoms	6.8M atoms	11.3M atoms
Serial (Unoptimized)	167.115	572.01 S	3547.07s	7 hours	14 hours (estimate)
Serial (Optimized)	53.57s	196.125	2003.6s	1510.71s	2614.4 5
Multicore	4.67s	32.82s	116.66s	153.8s	146.06s
NVIDIA P40	3.47s	17.15s	56.2s	7 8.5 7s	72.55s
NVIDIA V100	3.115	13.62s	39.79s	49.63s	47.71s

PLOS Computational Biology journal: https://journals.plos.org/ploscompbiol/article?i d=10.1371/journal.pcbi.1007877





BEST PRACTICE #2: SYSTEMATIC TESTING





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

• LOC: 30-40K Lines of Code

- Team: 4 Computer Scientists and 2 Solar physicists
- Tools used: Valgrind, GDB, Python notebooks, PCAST (an NVIDIA/PGI validation tool)
- Test bed: 3 different CPU-GPU compilers and 2 different platforms (NCAR & MaxPlank)



Comprehensive model of entire life cycle of a solar

flare



 Version Control: GitLab The Daniel K. Inouye Solar Telescope (DKIST), a ~\$300M NSF
 TinNCAR ent: 8 months and the clock was observational solar physics by an order of magnitude.
 StRPLicking!!!!



ERROR IN DENSITY CALCULATION



Difference in density between reference and the test runs





LESSONS LEARNT

- Work in small steps: Make incremental changes
- **Testing:** AFTER every commit BEFORE optimizing/scaling
- DOCUMENT!!! PLEASE :-)
- **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 or the base version. VERIFY again
- Verification: Build Jupyter notebooks to capture and record divergence of results between serial/multi-core CPU and GPU



YOOHOO - PASC PUBLICATION!

Sunita Chandrasekaran @sunitachandra29 · Dec 14, 2020
So proud of my project team!
Paper submitted
3 yrs
4 talks
1 PhD prelims
1 MASSIVE bug took 10 months to fix
RIGOROUS testing
Jupyter npbs
GitHub
MANY intense
w/ scientists
#InterdisciplinaryScience

The JOY u get doing spell check 4 hrs b4 the deadline!







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



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 directiversity of Time critical bugs - communicate with the developers directiversity of FIAWARE



BEST PRACTICE #4: DOCKER



COLLABORATIVE SOFTWARE DEVELOPMENT

Tools/Platforms such as Docker, Container, GitHub

• Dramatically reduces barrier to collaboration

NVIDIA, NGC ACCELERATED SO

CELERATED SOFTWAR

Google slides

								SETUP	[la pa
		RATED SOFTW	/ARE						m
		< Open	ACC T	raining Ma	aterials				th
				3					Тс
	SETUP			Publisher	Built By	Latest Tag	Modified		
				NVIDIA	PGI Compile	20.1.1	February 5, 2		
				Description					
				These training n	naterials have been devel	oped as a collaborat	ion between the University of	f	
			•	NVIDIA Corpora	tion and are provided fre	e of charge by Open/	ACC.org.		τī
			;	Labels					re
				HPC High Pe	erformance Computing	penACC PGI Sup	percomputing		be La
									_
				Pull Command					R
									IT
				docker pull	nvcr.io/hpc/openacc-tra	ining-materials:20.	.1.1	⑦ Documentation ^{IP}	de
								S User Forum 🗗	th
		Overview	Tags					NGC Version: 2 22 0	
								NGC 70131011, 2.22.0	
	⑦ Documentation ^{CP}			0		in a Matania			
R	දි. User Forum 🗗			OpenAC	official frain	ing Materia	als -		
1	<< Collapse			These training n	naterials have been devel	oped as a collaborati	ion between the University of	f Delaware and	
	NGC Version: 2.22.0			NVIDIA Corpora	tion and are provided fre	e of charge by Open	ACC.org. Please see CONTRIB	UTING.mt (

IWARE	SIGN IN CREA	ATE AN ACC
	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 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:	
	<pre>\$ docker rungpus all -itrm -p 8000:8000 nvcr.io/hpc/openacc-training-materials:20.1.1</pre>	
	LINIVERSI	TYOF

BEST PRACTICE #4: AUTOMATE



OPENMP VALIDATION & VERIFICATION TESTSUITE



ECP SOLLVE OPENMP V&V







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!

💭 jupytei	2D-Heat Last Checkpoint: 44 minutes ago (uns	aved changes)	Logout Control Pane
File Edit	View Insert Cell Kernel Widgets	Help	Trusted Python 3
+ %	A I → → H Run ■ C → Markdow	vn	
In [8]:	<pre># To be sure we see some output from #(if the compile does not return an e !make heat_0</pre>	the compiler, we'll echo out "Compiled proof)	Successfully!"
	pgcc -03 heat.c -o heat_0		
In [9]:	<pre># Execute our single-thread CPU-only d # above cell first. 1./heat_0 1024 1024 20000 output.dat</pre>	Jacobi Iteration to get timing informati	ion. Make sure you compiled successfully in
	Time for computing: 46.59 s		
	Time for computing: 46.59 s <i>After each step</i> , we will record the results from our	r benchmarking and correctness tests in a table like t	this one:
	Time for computing: 46.59 s After each step, we will record the results from our s	r benchmarking and correctness tests in a table like t tep Execution ExecutionTime (s) Speedup vs. 1 CP	this one: PU Thread
	Time for computing: 46.59 s After each step, we will record the results from our s	r benchmarking and correctness tests in a table like t tep Execution ExecutionTime (s) Speedup vs. 1 CP 1 CPU 1 thread 44.85	his one: PU Thread
	Time for computing: 46.59 s <i>After each step</i> , we will record the results from our s Profiling	r benchmarking and correctness tests in a table like t tep Execution ExecutionTime (s) Speedup vs. 1 CP 1 CPU 1 thread 44.85	his one: PU Thread
	Time for computing: 46.59 s <i>After each step</i> , we will record the results from our S Profiling ¶ Run the following cell to profile and view the code	r benchmarking and correctness tests in a table like t tep Execution ExecutionTime (s) Speedup vs. 1 CP 1 CPU 1 thread 44.85 :	this one: PU Thread
In [*]:	Time for computing: 46.59 s After each step, we will record the results from our Profiling Run the following cell to profile and view the code Name Pgprof -o serial.profcpu-profiling pgprof -i serial.prof	r benchmarking and correctness tests in a table like t tep Execution ExecutionTime (s) Speedup vs. 1 CP 1 CPU 1 thread 44.85 : -scope instruction ./heat_0 1024 1024 2	his one: ¹ U Thread 0000 output.dat
In [*]:	Time for computing: 46.59 s After each step, we will record the results from our Profiling Run the following cell to profile and view the code %%bash pgprof -o serial.profcpu-profiling pgprof -i serial.prof	r benchmarking and correctness tests in a table like t tep Execution ExecutionTime (s) Speedup vs. 1 CP 1 CPU 1 thread 44.85 : -scope instruction ./heat_0 1024 1024 2	his one: <u>PU Thread</u> 0000 output.dat
In [*]:	Time for computing: 46.59 s After each step, we will record the results from our Profiling Run the following cell to profile and view the code %%bash pgprof -o serial.profcpu-profiling pgprof -i serial.prof ======= CPU profiling result (bot Time(%) Time Name	r benchmarking and correctness tests in a table like t tep Execution ExecutionTime (s) Speedup vs. 1 CP 1 CPU 1 thread 44.85 : -scope instruction ./heat_0 1024 1024 2 :tom up):	his one: PU Thread 0000 output.dat
In [*]:	Time for computing: 46.59 s After each step, we will record the results from our Profiling Run the following cell to profile and view the code %%bash pgprof -o serial.profcpu-profiling pgprof -i serial.prof cpu-profiling result (bot Time(%) Time Name 99.72% 13.09s runTest(int, ch	r benchmarking and correctness tests in a table like t tep Execution ExecutionTime (s) Speedup vs. 1 CP 1 CPU 1 thread 44.85 :scope instruction ./heat_0 1024 1024 2 :tom up): har**)	his one: PU Thread 0000 output.dat

· · · · · · · · · · · · · · · · ·

• Useful for training and education

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.o
 rg/

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



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...





Computational Research and Programming Lab