



Deep Learning and GPU programming using OpenACC

14 – 17 July 2020



MODULE TWO: PROFILING

Dr. Volker Weinberg | LRZ | 16.07.2020





MODULE OVERVIEW

Topics to be covered

- Compiling and profiling sequential code
- Explanation of multicore programming
- Compiling and profiling multicore code



COMPILING SEQUENTIAL CODE



Ę

PGI COMPILER BASICS

pgcc, pgc++ and pgfortran

- The command to compile C code is 'pgcc'
- The command to compile C++ code is 'pgc++'
- The command to compile Fortran code is 'pgfortran'
- The -fast flag instructs the compiler to optimize the code to the best of its abilities

\$ pgcc -fast main.c
\$ pgc++ -fast main.cpp
\$ pgfortran -fast main.F90



PGI COMPILER BASICS -Minfo flag

- The Minfo flag will instruct the compiler to print feedback about the compiled code
- -Minfo=accel will give us information about what parts of the code were accelerated via OpenACC
- -Minfo=opt will give information about all code optimizations
- -Minfo=all will give all code feedback, whether positive or negative

\$ pgcc -fast -Minfo=all main.c
\$ pgc++ -fast -Minfo=all main.cpp
\$ pgfortran -fast -Minfo=all main.f90





Ę

OPENACC DEVELOPMENT CYCLE

- Analyze your code to determine most likely places needing parallelization or optimization.
- Parallelize your code by starting with the most time consuming parts, check for correctness and then analyze it again.
- Optimize your code to improve observed speed-up from parallelization.





Step 1: Run Your Code

Record the time it takes for your sequential program to run.

Note the final results to verify correctness later.

Always run a problem that is representative of your real jobs.

Terminal Window

```
$ pgcc -fast jacobi.c laplace2d.c
$./a.out
   0, 0.250000
 100, 0.002397
 200, 0.001204
 300, 0.000804
 400, 0.000603
 500, 0.000483
 600, 0.000403
 700, 0.000345
 800, 0.000302
 900, 0.000269
 total: 39.432648 s
```



Step 2: Profile Your Code

Obtain detailed information about how the code ran.

This can include information such as:

- Total runtime
- Runtime of individual routines
- Hardware counters

Identify the portions of code that took the longest to run. We want to focus on these "hotspots" when parallelizing.

! pgprof ./laplace

Jacobi relaxation Calculation: 4096 x 4096 mesh 0, 0.250000 100, 0.002397 200, 0.001204 300, 0.000804 400, 0.000603 500, 0.000483 600, 0.000403 700, 0.000345 800, 0.000302 900, 0.000269 total: 60.611229 s

======= CPU profiling result (bottom up): Time(%) Time Name 54.51% 32.43s calcNext 54.51% 32.43s | main 45.40% 27.01s __c_mcopy8_avx 0.05% 30ms swap 0.05% 30ms | main 0.03% 20ms __c_mcopy8

====== Data collected at 100Hz frequency



PROFILING SEQUENTIAL CODE Introduction to PGProf

- Gives visual feedback of how the code ran
- Gives numbers and statistics, such as program runtime
- Also gives runtime information for individual functions/loops within the code
- Includes many extra features for profiling parallel code

File View Window	<u>R</u> un Help												
📫 🗟 🖳 📑 🖷	್ರ್ . (ಈ	⊇ 2 F	Γ Γ [\$ 1	7							
conjugate_gradien	t.nvvp 🛙											- 0	
		0.69 s	0.695	s 0.	7 s	0.705 s		0.71 s	0.715 s	0.72 s	0.725 s	0.73 s	
Process "cg.x" (5127)													1
Thread 1601578816	i												
└ OpenACC	acc	_update@ve	ector.h	acc_compu acc_v	te_con vait@v	istruct@vecto vector.h:33	r	acc_com ac	pute_construc cc_wait@vecto	t@vector r.h:33	acc_compu acc_v	te_construct(vait@vector.h	
L Driver API				cuStr	eamSy	ynchronize		cu	JStreamSynchr	onize	cuStr	eamSynchror	
Profiling Overhead													
🖃 [0] Quadro GP100													
Context 1 (CUDA)													
🗆 🍸 MemCpy (Htol	D)												
🗆 🍸 MemCpy (Dtol	H)												
Compute				_Z6matvec	RКбта	ltrixRK6vector	S	_Z6matv	vecRK6matrixR	KóvectorS	_Z6matvec	RK6matrixRK6	
└ 🍸 89.0% _Z6m	natvecR			_Z6matvec	RКбта	ltrixRK6vector	S	_Z6matv	vecRK6matrixR	KóvectorS	_Z6matvec	RK6matrixRK6	
└ 🍸 7.2% _Z6wa	xpbydR												
└ 🍸 2.4% _Z3do	tRK6ve										1		
└ 🍸 1.3% _Z3do	tRK6ve												
Streams							_						U
Stream 13				_Z6matvec	RK6ma	ltrixRK6vector	S	_Z6matv	vecRK6matrixR	KóvectorS	_Z6matvec	RK6matrixRK6	J
	(•(C						<u>) Þ</u>	
🗔 Analysis 🔤 GPU De	etails 🛛 🛅	CPU Details	📮 Conso	le 🗔 Settir	ngs	- 6	3	Propert	ies 🖾				
						E 🙏 🛆 🕚	~	Stream 13	3				
Name	Start Time	Duration	Grid Size	Block Size	Regs	Static SMem	A	Duration	n				
Memcpy HtoD [async]	307.719 ms	2.08 µs	n/a	n/a	n/a	n/a		Session	n				
Memcpy HtoD [async]	308.386 ms	1.344 µs	n/a	n/a	n/a	n/a							
Memcpy HtoD [async]	310.385 ms	1.281 ms	n/a	n/a	n/a	n/a							
Memcpy HtoD [async]	312.464 ms	1.356 ms	n/a	n/a	n/a	n/a							
Memcpy HtoD [async]	313.983 ms	2.848 µs	n/a	n/a	n/a	n/a							
Memcpy HtoD [async]	314.374 ms	282.264 µs	n/a	n/a	n/a	n/a							
Memcpy HtoD [async]	316.287 ms	1.351 ms	n/a	n/a	n/a	n/a							
Memcpy HtoD [async]	318.214 ms	1.351 ms	n/a	n/a	n/a	n/a	5						
(4())		(4)					Þ



PROFILING SEQUENTIAL CODE CPU Details

Please make sure to choose "Profile current process only" from the dropdown instead of "Profile child process". The CPU details will not be displayed in the profiler otherwise.

PGI	Create New Session	+ = ×
Executable Prop	erties	
Set executable pro	perces	
Connection:	Local	Manage connections
Toolkit/Script:	CUDA Toolkit 10.1 (/opt/pgi/linux86-64-llvm/2019/cuda/10.1/	bin/) Manage
File:	/home/openacc/labs/module2/English/C/laplace	Browse
Working directory:	Enter working directory [optional]	Browse
Arguments:	Enter command-line arguments	
	Profile child processes	▼
Environment:	Profile child processes	Add
	Profile current process only	Delete
	< Back Next > Cancel	Einish



First sight when using PGPROF

- Profiling a simple, sequential code
- Our sequential program will run on the CPU
- To view information about how our code ran, we should select the "CPU Details" tab

File View Window <u>R</u> un H	ielp • • • • F	r 📕 🚊 🚊 🔥	⊽		
💺 laplace.nvvp 🖾					- 8
	0 s	50000000 s	100000000 s	150000000 s 20000000 s	250000000 s
🖬 Analysis 🗱 📑 GPU Details	📰 CPU Details	🚽 Console 🗔 Settings	»	□ Properties 🛛	
				Select or highlight a single interval to see properties	
The guided analysis syste through the various analy you understand the optim opportunities in your app become familiar with the process, you can explore analysis stages in an ungu optimizing your application fully utilize the compute a capabilities of the GPU. Th look at your application's as well as the performant					



PROFILING SEQUENTIAL CODE CPU Details

- Within the "CPU Details" tab, we can see the various parts of our code, and how long they took to run
- We can reorganize this info using the three options in the top-right portion of the tab
- We will expand this information, and see more details about our code

Applications S PGI Profiler					11:5	1 ro
91		PGI Profiler			• =	đ
Eile <u>V</u> iew <u>W</u> indow <u>R</u> un <u>H</u> elp						
े 🖬 🖳 🖬 🖏 • (२, २, १) ि (२ ४ ()						
*NewSession1 🛛						- 6
0 s	0.2 s	0.5 s		0.8 s		1
Analysis GPU Details (Summary) E CPU Details 23	OpenACC Details	🗑 OpenMP Details 🗟 Console 📑 Sc	ttinos 🔍 🗖	Properties S	ngle interval to s	ee
Event	%			properties		
			1			
▼ /home/openacc/labs/module2/English/C/laplace2d.c	54.116%		33.2 s 33.2 s			
▶ calcNext	54.099%		33.2 s 33.2 s			
Þ swap	0.016% 10	ms 10 ms				
✓ /opt/pgi/linux86-64-llvm/19.10/lib/libpgc.so	45.607%	28 s	28 s			
▼ Unknown Filename	45.607%	28 s	28 s			
▶c_mcopy8_avx	45.607%	28 s	28 s			



PROFILING SEQUENTIAL CODE CPU Details

- We can see that there are two places that our code is spending most of its time
- 33.2 seconds in the "calcNext" function
- 28 seconds in a memcpy function
- The c_mcopy8_avx that we see is actually a compiler optimization that is being applied to our "swap" function

					11:34	TOL
		PGI Profiler			• =	P.
ile <u>V</u> iew <u>W</u> indow <u>R</u> un <u>H</u> elp						
1 🖩 🖳 🖷 👒 • 1 🕁 🗠 🔍 1 E F K 🖺 🖁	- A -					
*NewSession1 🛙					-	
0 s	0.2 s	0.5 s		0.8 s		1,5
🗄 Analysis 📗 GPU Details (Summary) 🧮 CPU Details 🛱 📷	OpenACC Det	ails 🛅 OpenMP Details 📮 Console 🔚 Settings		Properties	-	
All threads 🔻 0 max				Select or highlight a sin properties	ngle interval to see	e
Event	%					
W /bama/apapace/labe/medulo2/English/C/laplaco2d e	54 11 69/	22.2	22.2 c			
 /nome/openacc/rabs/module2/English/C/rapiace20.C 	54.110%	53.2 5	33.2.5			
▶ calcNext	54.099%	33.2 s	33.2 s			
Þ swap	0.016%	10 ms 10 ms				
· · · · · · ·						
/opt/pgi/linux86-64-llvm/19.10/lib/libpgc.so	45.607%	28 s 28 s				
	45.607%	28 s 28 s				
▶c_mcopy8_avx	45.607%	28 s 28 s				
1	1		•			



- We are also able to select the different elements in the CPU Details by double-clicking to open the associated source code
- Here we have selected the "calcNext" element, which opened up the source file in the top part of the windo





Step 2: Profile Your Code

Obtain detailed information about how the code ran.

This can include information such as:

- Total runtime
- Runtime of individual routines
- Hardware counters

Identify the portions of code that took the longest to run. We want to focus on these "hotspots" when parallelizing.





}

Step 3: Identify Parallelism

Observe the loops contained within the identified hotspots

Are these loops parallelizable? Can the loop iterations execute independently of each other? Are the loops multi-dimensional, and does that make them very large?

Loops that are good to parallelize tend to have a lot of iterations to map to parallel hardware. void pairing(int *input, int *output, int N){

```
for(int i = 0; i < N; i++)
    output[i] = input[i*2] + input[i*2+1];</pre>
```





PLEASE START LAB NOW!



- To get started, follow these steps:
- Create an NVIDIA Developer account at <u>http://courses.nvidia.com/join</u> Select "Log in with my NVIDIA Account" and then "Create Account" (done yesterday)
- Visit <u>http://courses.nvidia.com/dli-event</u> and enter the event code

HLRS_OACC_AMBASSADOR_JUL20



	■ https://courses.nvidia.com/courses/course-v1%3ADLI%2BC-AC-03%2BV1/course/		⊘ ☆	± III\ €
0	NVIDIA.		Courses 📃 Volker_Weinberg_Test 🗸	•
	Home Course Progress			
	Fundamentals of Accelerated Computing with OpenACC		n the course Search Resume Course	
	Fundamentals of Accelerated Computing with OpenACC		Course Tools	
	Click here to get started	esume Course 🔮	Bookmarks	
	Feedback		Today is Jun 15, 2020 17:20 CEST	
			Course Handouts No Course Handouts	

Fundamentals of Accelerated Computing with OpenACC

Bookmark this page



Please wait 5 - 10 minutes while your interactive GPU enabled environment loads. When the "Launch" button appears, click it to get started.





PROFILING MULTICORE CODE What is multicore?

- Multicore refers to using a CPU with multiple computational cores as our parallel device
- These cores can run independently of each other, but have shared access to memory
- Loop iterations can be spread across CPU threads and can utilize SIMD/vector instructions (SSE, AVX, etc.)
- Parallelizing on a multicore CPU is a good starting place, since data management is unnecessary





Using a multicore CPU with OpenACC

- OpenACC's generic model involves a combination of a host and a device
- Host generally means a CPU, and the device is some parallel hardware
- When running with a multicore CPU as our device, typically this means that our host/device will be the same
- This also means that their memories will be the same





Compiling code for a specific parallel hardware

- The '-ta' flag will allow us to compile our code for a specific, target parallel hardware
- 'ta' stands for "Target Accelerator," an accelerator being another way to refer to a parallel hardware
- Our OpenACC code can be compiled for many different kinds of parallel hardware without having to change the code

```
$ pgcc -fast -Minfo=accel -ta=multicore laplace2d.c
calcNext:
    35, Generating Multicore code
    36, #pragma acc loop gang
```



- The first difference we see in this multicore profile is that there is now a "timeline"
- This timeline will show when our parallel hardware is being used, and how it is being used
- Each of the blue bars represent a portion of our program that was run on the multicore CPU





PROFILING MULTICORE CODE CPU Details

- Looking at our CPU Details, we can see that there is a lot more happening compared to our sequential program
- For the most part, these extra details revolve around the need for the CPU cores to communicate with each other

Applications PGI Profiler				15:0					
PGI	PGI	Profiler		+ = = 2					
File View Window Help	ie ⊼iew Willingom Fielb								
🖉 🔤 Analysis 🖿 GPU Details (Summary) 🎟	CPU Details & CopenACC Detail	s 🧰 OpenMP Details 🗏 Cons	ole 🥅 Settings	1. at 5: - a					
	max		ore La Secondo	+4 14 -4					
Event	%								
initialize_1F1L48	42.736%								
c_mcopy8_avx	38.511%								
> sched_yield	9.698%								
kmp_hardware_timestamp	4.024%								
<pre>kmp_join_barrier(int)</pre>	3.179%								
> 777	0.644%								
≻kmp_yield	0.121%								
> _mp_bcs_nest	0.04%								



View of computational threads

 You can see statistics for all threads or select a specific thread in the box on the top left of the CPU Details tab.





View of all computational threads

- When moving the mouse on the % value, one can see
 - Mean across all threads
 - Total across all threads
 - Total as a percentage of all the time spent on one / all threads.

Function: calcNext	511%
Management all threads 12 a	Function:c_mcopy8_avx
Mean across all threads: 13 s	Mean across all threads: 9.9 s
Total across all threads: 13 s	Total across all threads: 39.8 s
Percentage of all the time spent on Thread 0: 52.193	% Percentage of all the time spent on Thread 0: 38.511%



PROFILING MULTICORE CODE OpenACC Details

			PGI Profiler	(° - •
ile <u>V</u> iew <u>W</u> indow <u>H</u> elp				
🖡 *NewSession1 🛛 🗟 laplace	e2d.c 🔒 lapl	ace2d.c 🖾		
} return error; }				ŕ
⊖ void swap(double *restr	ict A, double	*restrict An	ew, int m, int n)	
<pre>for(int j = 1; j < {</pre>	n-1; j++)			
<pre>for(int i = 1;</pre>	i < m-1; i++)		
A[OFFSET(j, } }	i, m)] = Ane	w[OFFSET(j, i	, m)];	
⊖ void deallocate(double	*restrict A,	double *restr	ict Anew)	
<pre>void deallocate(double { free(A); free(Anew); }</pre>	*restrict A,	double *restr	ict Anew)	
<pre>void deallocate(double { free(A); free(Anew); } </pre>	*restrict A,	double *restr	ict Anew)	<u>~</u>
<pre>void deallocate(double { free(A); free(Anew); } </pre>	*restrict A,	double *restr	ict Anew) ACC Details & @ OpenMP Details @ Console 🕞 Settings P 🔲 Properties & acc_compute_cons	; □ □ □ truct@laplace2d.
<pre>void deallocate(double { free(A); free(Anew); } </pre>	*restrict A, mmar En CPU t	double *restr	ict Anew) ACC Details II DopenMP Details II Console II Settings II Properties II acc_compute_cons Start	▼ [truct@laplace2d. 5.62979 ≤
y void deallocate(double { free(A); free(Anew); } analysis GPU Details (Su ummary of OpenACC events o Name	*restrict A, mmar E CPU (on process: 384 %	double *restr.	ict Anew) ACC Details & OpenMP Details & Console Settings P Details & Console	truct@laplace2d. 5.62979 c 5.6403 s
y void deallocate(double { free(A); free(Anew); } analysis GPU Details (Su ummary of OpenACC events o lame aplace2d.c:36	*restrict A,	double *restr.	ict Anew)	truct@laplace2d. 5.62979 s 5.6403 s 10.51828
<pre>yoid deallocate(double { free(A); free(Anew); } analysis GPU Details (Su ummary of OpenACC events o Name laplace2d.c:36 acc_compute_construct</pre>	*restrict A, -	double *restr.	ACC Details 23 OpenMP Details Console Settings Competition Compared and Compared an	
<pre>void deallocate(double { free(A); free(Anew); } Analysis GPU Details (Su ummary of OpenACC events o Name i laplace2d.c:36 b acc_compute_construct i laplace2d.c:52</pre>	*restrict A, mmar	double *restr.	ACC Details X OpenMP Details Console Settings Console	
<pre>void deallocate(double { free(A); free(Anew); } analysis GPU Details (Su ummary of OpenACC events o Name laplace2d.c:36 b acc_compute_construct laplace2d.c:52 acc_compute_construct</pre>	*restrict A, mmar E: CPU I on process: 384 % 52.244% 52.244% 52.244% 47.334%	double *restr:	ict Anew) ACC Details II II OpenMP Details II Console II Settings II II Properties II acc_compute_cons II	truct@laplace2d. 5.62979 s 5.6403 s 10.51828 acc_comr unknown 201711
<pre>void deallocate(double { free(A); free(Anew); } </pre> <pre> free(Anew); </pre> <pr< td=""><td>*restrict A,</td><td>double *restr.</td><td>ict Anew) ACC Details & OpenMP Details Console Calls C</td><td>truct@laplace2d. 5.62979 e 5.6403 s 10.51828 acc_com unknown 201711 false</td></pr<>	*restrict A,	double *restr.	ict Anew) ACC Details & OpenMP Details Console Calls C	truct@laplace2d. 5.62979 e 5.6403 s 10.51828 acc_com unknown 201711 false

Applications PGI Profiler					13:01				
PGI		PGI Profi	ler		+ - • ×				
Elle View Window Help									
Save (Ctrl+S)									
*NewSession1 🕫 laplace2d.c 🕱									
}									
}									
edouble calcNext(double *rest	⊖double calcNext(double *restrict A, double *restrict Anew, int m, int n)								
<pre>double error = 0.0;</pre>									
<pre>for(int j = 1; j < n-1; </pre>	j++)								
for(int i = 1; i <	m-l; i++)								
{ Anew[OEESET(i, i	(m) = 0.25 * (A[OFFSET(i, i+1, m)] + A	[OFFSET(i, i-1, m)]						
incur(or iser()) i	,, ,	A[OFFSET(j-1, i, m)] + A	[OFFSET(j+1, i, m)]);						
error = fmax(er }	ror, fabs(Anew[OF	FSET(j, i, m)] - A[OFFSE	T(j, i , m)]));						
}									
return error; }									
woid cupp/double treatrict A	double treatric	t Aport int m int m)							
{	, double restric	t Anew, Int m, Int n)							
<pre>for(int j = 1; j < n-1; {</pre>	j++)				-				
					•				
🔚 Analysis 🔚 GPU Details (Summar	🖿 🛅 CPU Details 📷	OpenACC Details 없 🕞 Open	MP Details 📮 Console 📑 Settings 📮 🗖	□ Properties 🖾					
				acc_compute_constr	uct@laplace2d				
Summary of OpenACC events on proc	cess: 384			Start	5.62979 5				
Name %	Time	Calls		End	5.6403 s				
▼ laplace2d.c:36 52	.244% 9.74278	s 1000		Duration	10.51828				
 acc_compute_construct 52 	.244% 9.74278	s 1000		Event Kind	acc_com;				
▶ laplace2d.c:52 47	.334% 8.82718	s 1000		Parent Construct	unknown				
				Version	201711				
				Implicit	false				
				Device Type	acc_devic				



Ē

Introduction to lab code - visual

We will observe a simple simulation of heat distributing across a metal plate.

We will apply a consistent heat to the top of the plate.

Then, we will simulate the heat distributing across the plate.





Introduction to lab code - technical

The lab simulates a very basic 2-dimensional heat transfer problem. We have two 2-dimensional arrays, **A** and **Anew**.

The arrays represent a 2dimensional, metal plate. Each element in the array is a **double** value that represents temperature.

We will simulate the distribution of heat until a **minimum change value** is achieved, or until we exceed a **maximum number of iterations.** OpenACC

 A

 0.0
 0.0
 0.0

 0.0
 0.0
 0.0

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

0.0

Anew

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

Introduction to lab code - technical

We initialize the top row to be a temperature of 1.0

The **calcNext** function will iterate through all of the inner elements of array A, and update the corresponding elements in Anew

We will take the average of the neighboring cells, and record it in **Anew.**

The **swap** function will copy the contents of Anew to A



Anew

0.0	0.0	0.0	0.0
0.0	023	0.73	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



Introduction to lab code

Α 1.0 1.0 1.0 1.0 0.0 0.25 0.25 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 Anew

1.0	1.0	1.0	1.0
0.0	0.25	0.25	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

The **swap** function will copy the contents of Anew to A



KEY CONCEPTS

In this module we discussed...

- Compiling sequential and parallel code
- CPU profiling for sequential and parallel execution
- Specifics of our Laplace Heat Transfer lab code



LAB GOALS In this lab you will do the following...

- Build and run the example code using the PGI compiler
- Use PGProf to understand where the program spends its time



THANK YOU



- To get started, follow these steps:
- Create an NVIDIA Developer account at <u>http://courses.nvidia.com/join</u> Select "Log in with my NVIDIA Account" and then "Create Account" (done yesterday)
- Visit <u>http://courses.nvidia.com/dli-event</u> and enter the event code

HLRS_OACC_AMBASSADOR_JUL20



		Courses Volker_Weir	nberg_Test 🔻
Home Course Progress			
Fundamentals of Accelerated Computing with OpenACC	Search th	ne course Search Resume Co	urse
Fundamentals of Accelerated Computing with OpenACC		Course Tools	
Click here to get started	Resume Course 🔮	Important Course Dates	
Feedback		Today is Jun 15, 2020 17:20 CEST	

Fundamentals of Accelerated Computing with OpenACC

Bookmark this page



Please wait 5 - 10 minutes while your interactive GPU enabled environment loads. When the "Launch" button appears, click it to get started.



Fundamentals of Accelerated Computing with OpenACC

Bookmark this page



1:56:47 REMAINING TIME LAUNCH TASK STOP TASK

Please wait 5 - 10 minutes while your interactive GPU enabled environment loads. When the "Launch" button appears, click it to get started.



Files Running Clusters	
Select items to perform actions on them.	Upload New -
	Name Last Modified File size
module2	vor 4 Tagen
module3	vor 4 Tagen
^C module4	vor 4 Tagen
^C module5	vor 4 Tagen
^C module6	vor 4 Tagen
START HERE.ipynb	vor 4 Tagen 1.36 kB



Welcome to the OpenACC labs

Please select the appropriate lab below.

- Module 2 Application Profiling with PGProf Lab This lab introduces students to application profiling using the PGProf profiler.
- Module 3 OpenACC Directives Basics This lab introduces OpenACC directives.
- Module 4 GPU Programming with OpenACC This lab introduces GPU programming with OpenACC.
- Module 5 Data Management with OpenACC This lab introduces OpenACC data management directives.
- Module 6 OpenACC Loop Optimizations This lab introduces students to loop optimizations in OpenACC.

Application Profiling with PGProf Lab

This lab is meant to accompany Module 2 of the OpenACC.org teaching materials. The purpose of this lab is to introduce students to application profiling using the PGProf profiler. Lab instructions and source code is available for C/C++ and Fortran.

Please see the following files to begin the lab:



PROFILING SEQUENTIAL CODE CPU Details

Please make sure to choose "Profile current process only" from the dropdown instead of "Profile child process". The CPU details will not be displayed in the profiler otherwise.

PGI	Create New Session	+ = ×			
Executable Prop	erties				
Set executable pro	perces				
Connection:	Local 🗸 M	anage connections			
Toolkit/Script:	CUDA Toolkit 10.1 (/opt/pgi/linux86-64-llvm/2019/cuda/10.1/bir	V) Manage			
File:	/home/openacc/labs/module2/English/C/laplace	Browse			
Working directory:	Enter working directory [optional]	Browse			
Arguments:	Enter command-line arguments				
	Profile child processes	[▼]			
Environment:	Profile child processes	Add			
	Profile current process only	Delete			
	-				
	< Back Next > Cancel	Finish			

