NODULE TWO: PROFILING

Dr. Volker Weinberg | LRZ





MODULE OVERVIEW

Topics to be covered

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



COMPILING SEQUENTIAL CODE



NVIDIA'S HPC COMPILERS (AKA PGI) NVIDIA Compiler Names (PGI names still work)

- nvc The command to compile C code (formerly known as 'pgcc')
- nvc++ The command to compile C++ code (formerly known as 'pgc++')
- nvfortran The command to compile Fortran code (formerly known as pgfortran/pgf90/pgf95/pgf77)
- The -fast flag instructs the compiler to optimize the code to the best of its abilities

\$ nvc -fast main.c
\$ nvc++ -fast main.cpp
\$ nvfortran -fast main.F90

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



NVIDIA'S HPC COMPILERS (AKA PGI) -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

\$ nvc -fast -Minfo=all main.c
\$ nvc++ -fast -Minfo=all main.cpp
\$ nvfortran -fast -Minfo=all main.f90



PROFILING SEQUENTIAL CODE



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.





PROFILING SEQUENTIAL CODE

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

```
$ nvc -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
```



PROFILING SEQUENTIAL CODE

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.





PROFILING WITH NSIGHT SYSTEM AND NVTX



Nsight Product Family





Timeline View *							
a Al fard blann -	5s +850ms +900ms	+950ms 6s	*50ms +100ms	+150ms +200ms	+250ms +300ms	+350ms	+400mis
System				A THEFT A THE DAY THE			📖 🔟
CUDA API	11 1.0 .000 0 101 00000 0. 000	. 46010100164000	I hread/core	1 0x1 : 41 - 41 - 11 - 11 - 10 - 10 - 14			
cuBLAS		A second second second		and the state of the state	م من م م	1 440 Albertan	I
Profile: overhead			migration				
→ 🗹 [178] python +			Ingradion	ALLES A REAL SAME. S		والمستعمل المراجع المراجع المتراكي المتكار	L
System	Processes	pt	التقريبا والمنفوط والمتعالية				
CUDA API		@2-101158781-401001-66618.	200002 40 4- 0.000-CD) 04	64 1044 1 1 1 d 0 0 4 1 0 600 1		(CERT 1) - 1 - 0000	184
cuDNN cuBLAS	and	other Betrater security .	all the second s	ka k		1 i.i.	
Profile: overhead					Thread state		
→ [165] pythen +	threads		ي المراجعة المراجعة المراجع المراجع المراجعة المراجعة المراجعة المراجعة المراجعة المراجعة المراجعة ا			الشاكر إلى المحا	L. J
System	state balance bring i wall Detr.	I a		10. 0. 1. 10. 00. 10. 00. 10. 00.		11000000	JINI SPALL
CUDA API			10000-0. 0.4 datal. (600-60.4 D	1.44.41.04 (201 1.986) 088			
cuONN cuBLAS		and a service of the second second	and a difference of the	the second second	Een.	a 11 f)) (r)
Profiler overhead	CUDA and						
			ا الا الكراخ الم	i Kili Kata i su hati i su h	L () L ()	1	
System	OpenGL API trace]] _pth_][p_]] [p.
CUDA API			10-11 4 84-4 81,000,04 - 4	1 a 84.14 a 66 8 4 a 1863 8 a	0101 4		
cuDNN cuDNN		and the second	Hall the state of the state	all a shaka a sa a s	and the first of a second s	· Le a	1.4
v I fistation .			t t file, salta 1990) i tabi	and the second			
Suter							
CUDA API			04 Jt (mm.)) 1, 04 0	441	1.015		
cuDNN		and the second second	1.4 · 1 · (4.1)	يدير جايد و الجالي	1	a 1	34
cuBLAS 33 threads hidden	CUBLAS Irace						
CUDA (Tesla P100-SXM2-1668)		A REAL PROPERTY AND A REAL	ia Land, She addresida, Sheka, et Jinahi in Dagarini Madada eta Jin			1 / / / / / / · · · ·	
✓ Stream 174 > Memory			na na maaanaa . In na in ha himmah la na hi	to whe is one have it is to be obtained at	a la bas la bas la bismada la	alar in bern in beinder	
✓ Kernels	ILO 010 II 0111 II		IFT OF CONTRACT IN ALL IN ALL INCOMENT IN A REAL	I A MAD IN MAR IS MAR IN MAR IN MAR IN	IN IN JUL IN JULY IN JUCKEDING O		1. 100
> maxwell_fp16_scudnn_fp16_128x128_strided8_s	gliff, interior, nn	l and mamony					
 maxwel_fp16_scudnit_fp16_t28s128_strided8_it 		erand memory	11111		11 11 11	0 1	
> dgrad_engine	10 trop	ofor optivition				21 <u>1</u> 21	
> cudnn_maxwell_gcgemm_6bb64_tn_batched 20 janual conversion builden	เเลก	sier activities					
* Stream 12	Last BR Shifts		بالمتعارية العارية العارية بالمراجعا والمتحاد	بالأبلاب الالتراب للأراب الأرابي المراجع والمتعاقبات	ولاحد بالاقتدام معتقد فالمتقادم	4 + 148- M 10	
← Kernels	And 00 and store and store and	· · · · · · · · · · · · · · · · · · ·	والمراجع والأعاد الأعاد المتحد والمتعادية والمتعادية والمتعادية	սեղիկում պետուլինու ին տեղել է հետե	بناهم الجاويين فيلت السابية الم	Danis and a DOLLAR	10.1 0 1.00.
AlReduceKernel AlReduceKernel			a state of the sta	and the state of a state of the	and a second		
1 kernel group(s) hidden							
67 stream(s) hidden							
 CUDA (Testa P100-SXM2-16G8) Stream 173 							
> Stream 20	Multi-GPU		consists and other or all strength of sides for a solution	a had been also de de la ser de la ser de la facilitat de la des de la	a chair a she ar an the second second	B B. MARTINE MARTIN	March March MMAL
 bit stream(s) hidden CUDA (Tesla P100-SXM2-16GB) 		THE REAL PROPERTY AND ADDRESS OF A DREAM AND AND A DREAM AND A DREAM AND AND A DREAM AND AND A DREAM AND AND A DREAM AND	THE RESIDENCE AND REAL PROPERTY AND IN		a anna anna anna		
CUDA (Tesla P100 SXM2-16GB)		- MARKET A COME IN THE REPORT OF A DESCRIPTION OF A DESCR		ALAMA ANA ALAMA A A A A A A A A A A A A A A A A			
 CUDA (Tesla P100-SXM2-1668) CUDA (Tesla P100-SXM2-1668) 			ومرجعي الجميعية والمحمد والمحمد				
 CUDA (Tesla P100-SXM2-16G8) CUDA (Tesla P100-SXM2-36G8) 		 Providely, Reference of the second sec	an in a state and a state, in the part of provident state of the state		n andra a and a second s		
- CODA (TIESE P 100-SAME- 1968)	4		This materia	al is released by NVIDIA Corporation	under the Creative Commons	Attribution 4.0 Internation	al (CC BY 4.0)

PROFILING SEQUENTIAL CODE

Using Command Line Interface (CLI)

NVIDIA Nsight Systems CLI provides

- Simple interface to collect data
- Can be copied to any system and analysed later
- Profiles both serial and parallel code
- For more info enter nsys --help on the terminal

To profile a serial application with NVIDIA Nsight Systems, we use NVIDIA Tools Extension (NVTX) API functions in addition to collecting backtraces while sampling.



PROFILING SEQUENTIAL CODE

NVIDIA Tools Extension API (NVTX) library

What is it?

- A C-based Application Programming Interface (API) for annotating events
- Can be easily integrated to the application
- Can be used with NVIDIA Nsight Systems

Why?

- Allows manual instrumentation of the application
- Allows additional information for profiling (e.g: tracing of CPU events and time ranges)

How?

- Import the header only C library nvToolsExt.h
- Wrap the code region or a specific function with nvtxRangePush() and nvtxRangPop()





Open the generated report files (*. nsys-rep) from command line in the Nsight Systems profiler.

File -> Open





Navigate through the "view selector".

"Analysis summary" shows a summary of the profiling session. To review the project configuration used to generate this report, see next slide.

"Timeline View" contains the timeline at the top, and a bottom pane that contains the events view and the function table.

 X
 Image: Summary

 Image: Summary
 0s

 Analysis Summary
 0s

 Symbol Resolution Logs
 Files

 Files
 CPU (80)

 *
 Threads (3)

 *
 [12108] laplace

Read more: <u>https://docs.nvidia.com/nsight-systems</u>



USI	ng Nsight Systems	 CPU (80) Threads (3) (2) [12108] laplace + 				
laplace3.qdrep X						
Profiling session of	uration: 00:55.623	NVTX	-	while (55.470 g)		
Total number of threads	3					
Number of events collected	70,773	Profiler overhead				
Report size	85122 KB					
Report capture date	19 March 2020 08:01:16	2 threads hidden				
Host computer	pm-dgs-28					
Profiling stop reason	Stopped by user					
Imported from	home/mochgank/Code/openacc-training-materials/labs/module2/English/C/taplace3.qdstrm					
Import host computer	pm-ogr_cos					
Show report file in folder						
prm-dgx-28 (0:1)				Time alling a view		
Target				i imeline view		
Target name	pm-dgx-28			(charts and the hierarchy on the ton nane)		
Target name Platform	prm-dgx.28 Linux			(charts and the hierarchy on the top pane)		
Target name Platform OS	prm-dgx-28 Linux Uburitu 18.04.3 LTS			(charts and the hierarchy on the top pane)		
Target name Platform OS Hardware platform	pm-dgs-28 Linux Uburtu 18:04.3.LTS - x60_64			(charts and the hierarchy on the top pane)		
Target name Platform OS Hardware platform Serial number	prm-dgx-28 Linux Uburnt 16 43 LTS x86_64 Local (CL1)			(charts and the hierarchy on the top pane)		
Target name Platform OS Hardware platform Serial number CPU description	prm-dgs-28 Linux Ubunti 18 04 J LTS x86_64 Local (CL) Intel(R) Xeon(R) OPU E5-2658 v4 @ 2.20GHz			(charts and the hierarchy on the top pane)		
Target name Platform OS Hardware platform Serial number CPU description CUDA driver version	pm dgs 28 Linax Unanta 16 04 LTS x86_64 Local (CL) Invik(Y) Xeou(X) CPUE 52689 vf @ 2 200Hz 10.2			(charts and the hierarchy on the top pane)		
Target name Platform OS Hardware platform Serial number CPU description CUDA driver version NVIDIA driver version	pm dgs.28 Ukur Uburtu 18.43.11'S x86_64 Local (CL) Intel(P, Xour(R) CPU E5-2698 v4 @ 2.20GHz 10.2			(charts and the hierarchy on the top pane)		
Tagel name Platform OS Hardware platform Serial number CPU description CUDA driver vension NVIDIA driver vension GPI rontert exelect	pm-dgs-28 Ukar Uberha 1843.173 346_64 Local (CL) Imaily Second OU ES-2599 v4 @ 2.200Hz 10.2 44433.01 suenostad	1 1 Maria 1 Maria 2 + 1 Maria 1 Maria		(charts and the hierarchy on the top pane)	Duration TO 5 133.02 ev 12306 55.070 12306	itart 0.008 0.141

Timeline view (event view and function table on the bottom pane)



NVIDIA Nsight Systems 2021.2.1 × Eile View Tools Help **Project Explorer** Project 1 | laplace.qdrep Project 1 0 x² 1x 0 Timeline View § 1 error, 10 messages laplace.qdrep D 05 10s 40s 50s 60s 20\$ 30s CPU (8) Threads (3) ▼ √ [316] laplace + while [66,990 s] NVTX Profiler overhead Filter and Reorder Shift+F 3 0 2 threads hidden... Filter and Zoom in Zoom into Selection right click in selected region and Zoom into selection! **OpenACC**

Enlarge view!

I Timeline View	-					$0 \ \overline{x^2}$ 1	x	s <u>1 error, 10</u>	<u>0 message</u>
	9s	Oms	+300ms	+350ms	+400ms	+450ms	+500ms	+550ms	
 CPU (8) 									
 Threads (3) 									
▼ ✓ [316] laplace →									
NVTX		swap [30,4) c	alc [36,252 ms] swa	ıp (30,3) calc (36,083 r	while [66, ns] swap [30,4 calc [3	990 s] (6,042 ms] swap [30,4.	calc [36,002 ms] swa	o [30,5 calc [36,064 m	ns] [sw
Profiler overhead									-
2 threads hidden	ag o								
		4							-
Events View									



PROFILING SEQUENTIAL CODE

Viewing captured NVTX events and time ranges via Nsight Systems GUI

From the Timeline view, right click on the "NVTX" from the top pane and choose "Show in Events View".

From the bottom pane, you can now see name of the events captured with the duration.





Project 2 🔲 laplac	e.qdrep 🛛									
Timeline View	-						0 x ² 1x]	\$ <u>1 error</u>	, 10 message
	18s 🛛	+640ms	+660ms	+680ms +	⊧700ms +7	20ms +740m	ns +760ms	+780ms	+800ms	+820ms
CPU (8)										_
 Threads (3) 										
▼ √ [113] Japlar	ΓAΠ									
NVTX					· · · · · ·	while [62,797 s]	Y			
		c swap [19,72	3 C	alc [40,443 ms]	swap [19,701	calc [40,366	ms] swap [19,	767	calc [40,423 ms]	swa
Profiler over	head									
2 threads hidd	len 🥜 o				-					
										-
		•								•
Events View	•									
							Ν	ame 🔻		Q
#	Name			Start	Duration	n TID	Category	Desc	cription:	
1	🝷 🛛 whil	e		0,193229	s 62,797 s	s 113		whi	le	
2		calc		18,5981s	40,468	ms 113		Beg	ins: 0,193229s	07 c)
3		swap		18,6386s	19,723	ms 113		Thre	ad: 113	57 5)
4		calc		18,6583s	40,443	ms 113				
5		swap		18,6988s	19,701	ms 113				
☐ 6		calc		18,7185s	40,366	ms 113				
7		swap		18,7589s	19,767	ms 113		-		
4				1		I	1			

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"
- Visit <u>http://courses.nvidia.com/dli-event</u> and enter the event code

LRZ_OPENACC_AMBASSADOR_NO23



OpenAC

Erstelle deinen Accou	nt
Email	
volker.weinberg@lrz.de	
Anzeigename	
V Weinberg	
Geburtsdatum	
	ō
Passwort	
	Stark
Kennwort bestätigen	
	-
Angemeldet bleiben <u>Mit Sicherheit</u>	<u>sgerät anmelden</u> ()
2 (Ø	
Ich bin ein Mensch Captcha Privatsphäre - Redingungen	
Indem ich fortfahre, stimme ich den <u>NVIDL</u> Nutzungsbedingungen und Datenschutzer	<u>A-Konto-</u> klärung
KONTO ERSTELLEN	
Weitere Anmeldeoptionen	
Weitere Anmeldeoptionen	



	\leftrightarrow \rightarrow C \triangle \triangleq courses.n	vidia.com/dli-event/		९ ☆ 8
	Apps 📷 Linux Academy 📷 Linux Ac	ademy Weather 🛛 🕺 DLI Event Dee	p Learning I 🤾 Portal Axis Security	
				Volker 🗸
	Deep Learning Institute	Online Courses Instru	ctor-Led Workshops	
	DLI Event			
	•			
	Event Code			
	LRZ_OPENACC_AMBASSAD	Enter your event code.		
	ENROLL			
	DLI Solutions	Products	Technologies	Resources
OpenACC More Science, Less Programming				

🚳 NVIDIA.

undamentals of Accelerated Computing with OpenACC	Sear	ch the course	Search	Start Course
Welcome to Fundamentals of Accelerated Computing with OpenACC.	×	Course Tools Bookmark: Updates	S	
Head over to the "Course" tab to get started!		Course Hand No Course Ha	outs andouts	

OpenACC

Eagdback

🕺 NVIDIA.

VOLKER 🗸

se progre	SS				
rse > Funda	nentals of Accelerated Computin	g with > Click here to get started	> Fundamentals of Accelerated	Computing with	
Previous					Next >
undamen Bookmark this	tals of Accelerated	Computing with Ope	nACC		
undamen Bookmark this	tals of Accelerated	Computing with Ope	nACC		

This will launch a pre-configured GPU workstation, it may take 5-10 minutes.



	ARNING INSTITUTE	ONLINE COURSES	INSTRUCTOR-LED WORKSHOPS	EDUCATOR PRO	OGRAMS	ENTERPRISE S	OLUTIONS
urse > Fundar	ess mentals of Accelerated Computing wit	h > Click here to get s	tarted > Fundamentals of Accelerated	Computing with			
< Previous			Ð				Next 🔰
undamen	tals of Accelerated Co	mputing with (DpenACC				
BOOKMARK THIS	page						







PROFILING MULTICORE CODE



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





PROFILING MULTICORE CODE 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





PROFILING MULTICORE CODE

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

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



PROFILING MULTICORE CODE

Compiling code for a specific parallel hardware

nsys profile -t nvtx --stats=true --force-overwrite true -o laplace_parallel ./laplace_parallel

NVTX Push-Pop Range Statistics:

Range	Maximum	Minimum	Average	Instances	Total Time (ns)	Time(%)
while	24908340742	24908340742	24908340742.0	1	24908340742	49.9
calc	52044034	9986457	13167317.0	1000	13167317033	26.4
swap	62627309	8693117	11711313.3	1000	11711313301	23.4
init	175394843	175394843	175394843.0	1	175394843	0.4

Report file moved to "/home/openacc/labs/module2/English/C/laplace_parallel.qdrep" Report file moved to "/home/openacc/labs/module2/English/C/laplace_parallel.sqlite"



PARALLEL VS SEQUENTIAL Compiler feedback

Have a close look at the compiler feedback for both sequential and parallel implementation of the application.

It provides information about how your program was optimized or why a particular optimization was not made.

Note: Adding -Minfo flag or -Minfo=accel or -Minfo=all when compiling, will enable compiler feedback messages, giving details about the parallel code generated.

jacobi.c	
laplace2	d.c:
alcNext	
38,	Accelerator restriction: size of the GPU copy of Anew, A is unknown
	Loop is parallelizable Generating Multicore code
38, 41,	-36, #pragma acc loop gang Generating implicit reduction(max:error) Loop is parallelizable
swap:	
54,	Accelerator restriction: size of the GPU copy of Anew, A is unknown Loop is parallelizable
	Generating Multicore code
57,	54, #pragma acc loop gang Loop is parallelizable Memory copy idiom, loop replaced by call toc_mcopy8

cop1.0	
place2	2d-multicore-parallel.c:
lcNext	
47	Generating implicit firstprivate(1, n, m)
.,,	Generating NV/TDTA GDU code
	40 = #nnage= acc loop gang /* blockIdy y */
	49, #pragma acc loop gang /* blocklux.x */
	Generating implicit reduction(max:error)
	52, #pragma acc loop vector(128) /* threadIdx.x */
47,	Generating implicit copyin(A[:]) [if not already present]
	Generating implicit copy(error) [if not already present]
	<pre>Generating implicit copyout(Anew[:]) [if not already present]</pre>
52,	Loop is parallelizable
ap:	
6 <mark>8</mark> ,	Generating implicit firstprivate(j,n,m)
	Generating NVIDIA GPU code
L	65, #pragma acc loop gang /* blockIdx.x */
	68, #pragma acc loop vector(128) /* threadIdx.x */
63,	Generating implicit copyout(A[:]) [if not already present]
	<pre>Generating implicit copyin(Anew[:]) [if not already present]</pre>
68,	Loop is parallelizable
rainv	<pre>@hawk-ai01:~/OpenACC/C/solutions></pre>
Thi	s material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BX 4



```
#include <math.h>
#include <stdlib.h>
#define OFFSET(x, y, m) (((x)*(m)) + (y))
void initialize(double *restrict A, double *restrict Anew, int m, int n)
{
    memset(A, 0, n * m * sizeof(double));
    memset(Anew, 0, n * m * sizeof(double));
    for(int i = 0; i < m; i++){</pre>
        A[i] = 1.0;
        Anew[i] = 1.0;
    }
}
double calcNext(double *restrict A, double *restrict Anew, int m, int n)
{
    double error = 0.0;
    #pragma acc parallel loop reduction(max:err)
    for( int j = 1; j < n-1; j++)</pre>
    {
        #pragma acc loop
        for( int i = 1; i < m-1; i++ )</pre>
        {
            Anew[OFFSET(j, i, m)] = 0.25 * ( A[OFFSET(j, i+1, m)] + A[OFFSET(j, i-1, m)]
                                           + A[OFFSET(j-1, i, m)] + A[OFFSET(j+1, i, m)]);
            error = max( error, fabs(Anew[OFFSET(j, i, m)] - A[OFFSET(j, i, m)]));
        }
    }
    return error;
}
void swap(double *restrict A, double *restrict Anew, int m, int n)
{
    #pragma acc parallel loop
    for( int j = 1; j < n-1; j++)</pre>
    {
        #pragma acc loop
        for( int i = 1; i < m-1; i++ )</pre>
        {
            A[OFFSET(j, i, m)] = Anew[OFFSET(j, i, m)];
        }
   }
}
void deallocate(double *restrict A, double *restrict Anew)
{
    free(A);
    free(Anew);
}
                           laplace2d.c
                                                                                                           "
               (Parallelised using OpenACC parallel
                                                                                                           "s
```

directives (pragmas highlighted)

0	penACC
	More Science, Less Programming

Importing Saved repo Exporting	[=====================================	ome/mozhgank/ [=====	Code/openacc-training	g-materials	======100%] s/labs/module4/Er ======100%]	ıglish/C/solut	ions/parall	el/laplace-par.qd	lrep"		
Exported s /home/mozh	uccessfully to gank/Code/open	acc-training-	materials/labs/module	≘4/English,	/C/solutions/para	lle/laplace-	-par.sqlite				
Generating CUDA API S	CUDA API Stat	istics oseconds)									
Time(%)	Total Time	Calls	Average	Minimum	Maximum	Name					
93.1 2.2 1.8 1.6 0.6 0.4 0.2 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0	1107821913 25667004 21314088 18919205 6575301 4672638 1957950 1835300 1013630 498778 251874 9531 3936	4000 1 1 3000 1000 1000 1000 1000 1000 1	276955.5 25667004.0 2131408.0 6306.4 1955.0 1835.3 101363.0 166259.3 251874.0 9531.0 1968.0	2205 25667004 21314088 4141 5719 3898 1656 1685 1013630 7731 251874 9531 1136	63176580 25667004 21314088 38827 37100 23307 9112 15602 1013630 462241 251874 9531 2800	custreamsync cuMemHostAll cuMemAllocM cuLaunchkern cuMemcpyDtoł cuWemstD32A cuEventRecor cuEventSynch cuMemAlloch cuMemAlloch cuModuleLoac cuStreamCreat	thronize loc unaged iel iAsync_v2 isync d irronize sst_v2 /2 IDataEx ie	CUDA A statistic	API 25		
Time(%)	Total Time	Instances	Average	Minimum	Maximum	Name	ſ				
58.9 40.4 0.7	643260973 441862005 7350076	1000 1000 1000	643261.0 441862.0 7350.1	563448 434490 6880	63174901 452729 7936	calcNext_35_ swap_51_gpu calcNext_35_	_gpured	CUDA Ke statistic	ernel cs		
CUDA Memor	y Operation St	atistics (nan	oseconds)								-
Time(%)	Total Time	Operations	Average	Minimum	Maximum	Name				CUDA	
94.0 3.1 2.9	47346144 1567908 1447303	8191 1000 1000	5780.3 1567.9 1447.3	2688 1535 1407	71456 2816 2080	[CUDA Unifie [CUDA memcpy [CUDA memset	ed Memory men / DtoH] :]	mcpy HtoD]		Memory Operation statistics	
CUDA Memor	y Operation St	atistics (KiB)								
2	7.813 7.813 7.813 60072.000	1000 1000 8191	0.008 0.008 31.751		0.008 0.008 4.000	0.008 0.008 732.000	CUDA memc [CUDA memc] [CUDA mems [CUDA Unif	py DtoH] et] ied Memory memcpy			
Generating NVTX Push-	NVTX Push-Pop Pop Range Stat	Range Statis istics (nanos	tics econds)								
Time(%)	Total Time	Instances	Average	Minimum	Maximum	Range					
47.6 29.2 <u>18.3</u> 4.9	1199130726 734660957 461447578 122270944	1 1000 1000 1	1199130726.0 : 734661.0 : 461447.6 : 122270944.0	1199130726 627348 454563 122270944	1199130726 89307285 490929 122270944	while calc swap init	NVTX stati	(range istics			
alc" regic wap" regi	on (calcNe ion (swap total exe	xt function function) cution tim	า) takes 29.2% takes 18.3% of าe]				Open lapl with Nsight view t	ace-pai Systen he time	r.qdrep n GUI to line	

PARALLEL VS SEQUENTIAL SPEEDUP

Viewing captured NVTX events

Have a close look at the captured NVTX events for both serial and parallel implementations.

Time spent in "while" loop has significantly decreased.

0 1

3

init [122.271 ms]

while [1,199 s]

Achieved speedup: ~47



#	Name		Duration	TID	Start
1	▶ [] init [137.490 ms]	Sequential	137.490 ms	46133	0.00975564s
3	Iwhile [55,754 s]	ooquontiai	55.754 s	46133	0.147279s



PROFILING PARALLEL CODE Viewing timeline via Nsight Systems

Contents of the tree-like hierarchy on the left depend on the project settings used to collect this report.

If a certain feature has not been enabled, corresponding rows will not be shown on the timeline.

In this example, we chose to trace NVTX and OpenACC while sampling.

Note: Kernel launches are represented by blue and memory transfers are displayed in green.

OpenACC



LAB CODE



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

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	025	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 NVIDIA's HPC compiler
- Use Nsight Systems to understand where the program spends its time



THANK YOU

