

Optimizing Stencil Operations in OpenACC

Ronald M. Caplan and Jon A. Linker

Predictive Science Inc.

9990 Mesa Rim Rd. Suite 170
San Diego, CA 92121
www.predsci.com

Funding provided by:
NASA SBIR-STTR
America's Seed Fund

INTRODUCTION



Stencil operations are used widely in HPC applications and are an optimization challenge on both CPUs and GPUs due to low flop-per-data ratios and non-contiguous memory access patterns.

When using low-level APIs on GPUs (such as CUDA), sophisticated fine-tuned optimizations can be formulated.

Many legacy/large HPC codes prefer higher-level and more portable APIs such as OpenACC, inhibiting fine-tuning.

However, OpenACC does allow some tuning options through various parallelization constructs and loop directives.



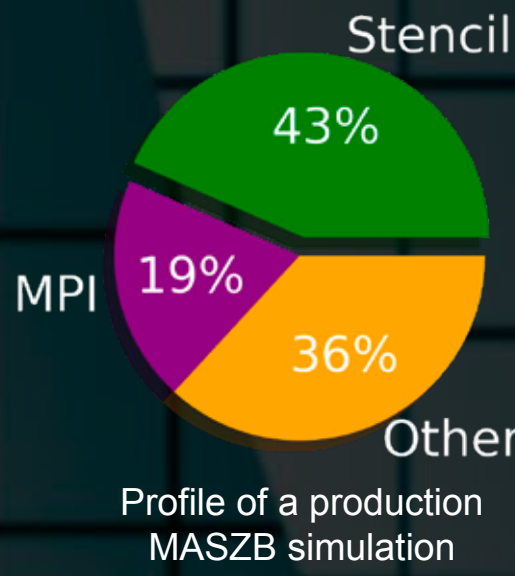
In any optimization venture, it is important to balance effort/time with expected results.

APPLICATION



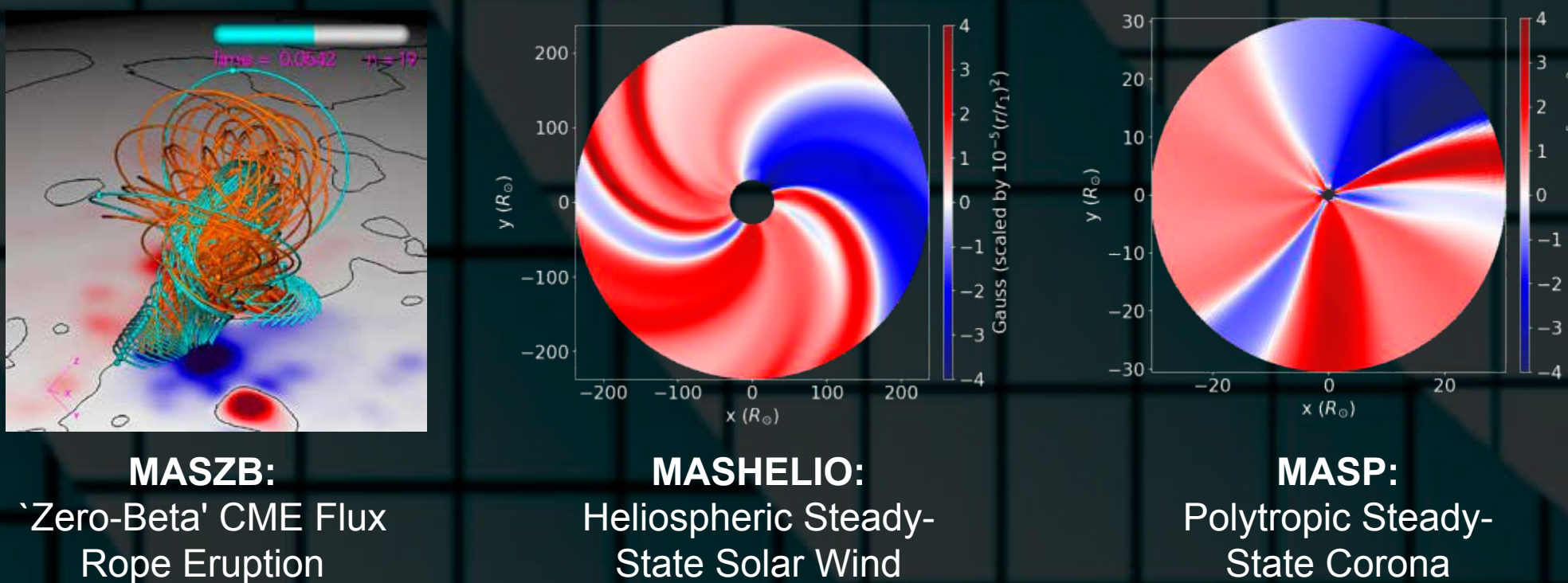
The application we investigate here is the Magnetohydrodynamics Around a Sphere (MAS) code. MAS is used extensively in solar physics research and uses OpenACC for GPU-acceleration.

For more information, visit: www.predsci.com/mas



For large production runs, over 40% of the run-time is located in a stencil operation applied to a 3-component staggered velocity vector.

Here, we try to optimize the stencil routine by exploring various directive options and comparing their performance. We test three distinct real-world model/problem types, each with three increasing resolutions:



	SMALL	MEDIUM	LARGE
MASZB	51x51x51	101x101x121	161x161x161
MASHLIO	51x51x51	141x111x129	161x161x161
MASP	51x51x51	101x101x121	161x161x161

The "large" problem size is closer to a production run, while the "small" problem size is important for MPI scaling considerations.

STENCIL CODE

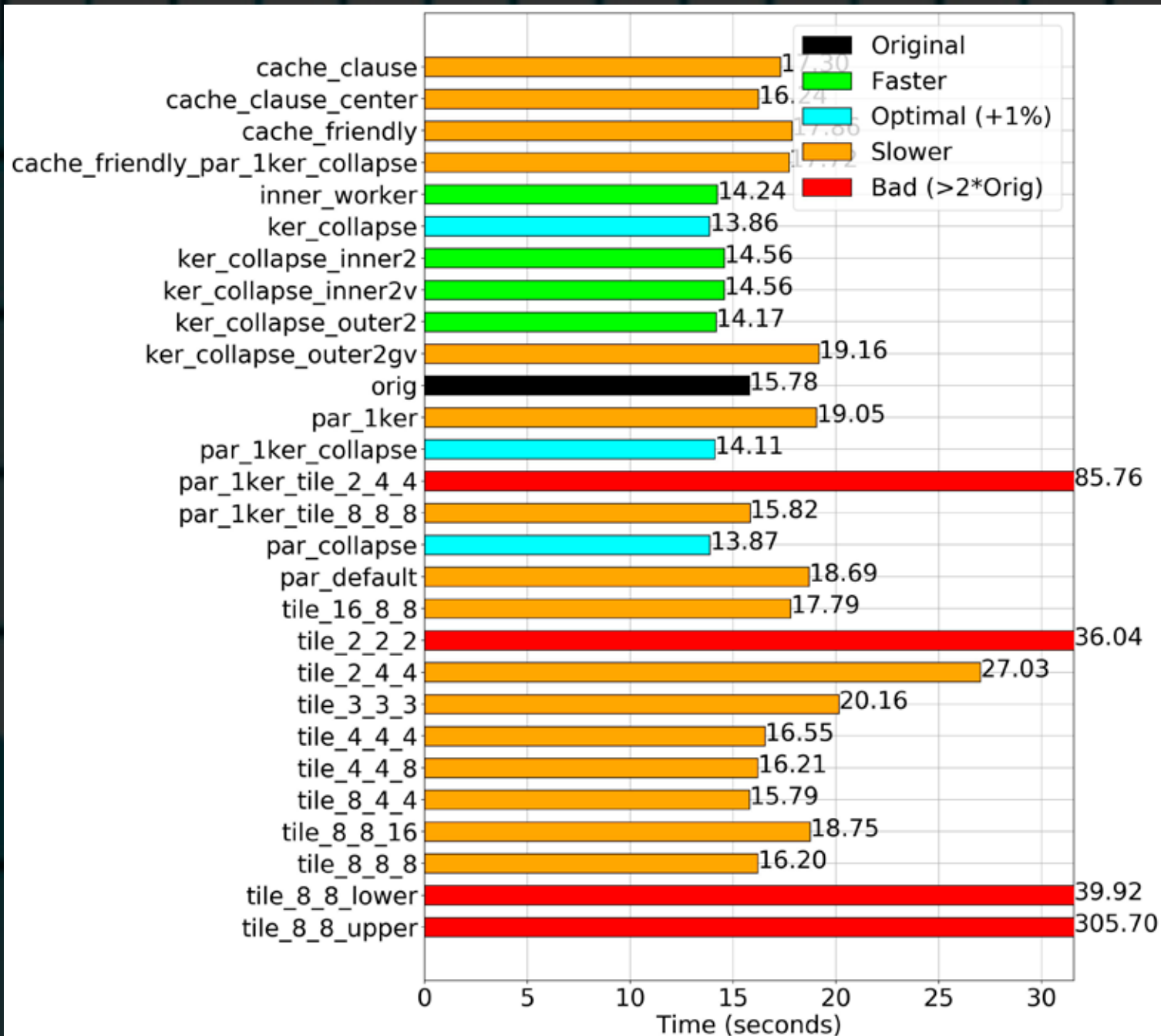
NOTE: All arrays are already resident on the device and "default(present)" is used for all loops

```
[A1]
[L1]
do k=2,npm1
[L2]
do j=2,ntm1
[L3]
do i=2,nrm-1
[C1]
ii=(ntm2*(nrm-2)*(k-2)+(nrm-2)*(j-2)+(i-1))
q(ii)=a_r(i,j,k,1)*pskr(i,j,k-1)+a_r(i,j,k,2)*pskr(i,j,k-1,k)
+a_r(i,j,k,3)*pskr(i,j,k-1,k)+a_r(i,j,k,4)*pskr(i,j,k-1,k)
+a_r(i,j,k,5)*pskr(i,j,k-1,k)+a_r(i,j,k,6)*pskr(i,j,k-1,k)
+a_r(i,j,k,7)*pskr(i,j,k-1,k)+a_r(i,j,k,8)*pskr(i,j,k-1,k)
+a_r(i,j,k,9)*pskr(i,j,k-1,k)+a_r(i,j,k,10)*pskr(i,j,k-1,k)
+a_r(i,j,k,11)*pskr(i,j,k-1,k)+a_r(i,j,k,12)*pskr(i,j,k-1,k)
+a_r(i,j,k,13)*pskr(i,j,k-1,k)+a_r(i,j,k,14)*pskr(i,j,k-1,k)
+a_r(i,j,k,15)*pskr(i,j,k-1,k)
enddo
enddo
enddo
[E1]
[A2]
[L1]
do k=2,npm1
[L2]
do j=2,ntm1
[L3]
do i=2,nrm1
[C2]
ii=(npm2*ntm2*(nrm-2))
+(ntm2*nrm2*(k-2)+nrm2*(j-2)+(i-1))
q(ii)=
a_t(i,j,k,1)*pskr(i,j,k)+a_t(i,j,k,2)*pskr(i,j,k)
+a_t(i,j,k,3)*pskr(i,j,k)+a_t(i,j,k,4)*pskr(i,j,k)
+a_t(i,j,k,5)*pskr(i,j,k)+a_t(i,j,k,6)*pskr(i,j,k)
+a_t(i,j,k,7)*pskr(i,j,k)+a_t(i,j,k,8)*pskr(i,j,k)
+a_t(i,j,k,9)*pskr(i,j,k)+a_t(i,j,k,10)*pskr(i,j,k)
+a_t(i,j,k,11)*pskr(i,j,k)+a_t(i,j,k,12)*pskr(i,j,k)
+a_t(i,j,k,13)*pskr(i,j,k)+a_t(i,j,k,14)*pskr(i,j,k)
+a_t(i,j,k,15)*pskr(i,j,k)
enddo
enddo
enddo
[E1]
[A3]
[L1]
do k=2,npm-1
[L2]
do j=2,ntm1
[L3]
do i=2,nrm1
[C3]
ii=(npm2*ntm2*(nrm-2))+(npm2*(ntm-2)*nrm2)
+ntm2*nrm2*(k-2)+nrm2*(j-2)+(i-1)
q(ii)=
a_p(i,j,k,1)*pskr(i,j,k)+a_p(i,j,k,2)*pskr(i,j,k)
+a_p(i,j,k,3)*pskr(i,j,k)+a_p(i,j,k,4)*pskr(i,j,k)
+a_p(i,j,k,5)*pskr(i,j,k)+a_p(i,j,k,6)*pskr(i,j,k)
+a_p(i,j,k,7)*pskr(i,j,k)+a_p(i,j,k,8)*pskr(i,j,k)
+a_p(i,j,k,9)*pskr(i,j,k)+a_p(i,j,k,10)*pskr(i,j,k)
+a_p(i,j,k,11)*pskr(i,j,k)+a_p(i,j,k,12)*pskr(i,j,k)
+a_p(i,j,k,13)*pskr(i,j,k)+a_p(i,j,k,14)*pskr(i,j,k)
+a_p(i,j,k,15)*pskr(i,j,k)
enddo
enddo
enddo
[E2]
[W]
do k=2,npm-1
do j=2,ntm1
do i=2,nrm1
ii=(npm2*ntm2*(nrm-2))+(npm2*(ntm-2)*nrm2)
+ntm2*nrm2*(k-2)+nrm2*(j-2)+(i-1)
q(ii)=
a_p(i,j,k,1)*pskr(i,j,k)+a_p(i,j,k,2)*pskr(i,j,k)
+a_p(i,j,k,3)*pskr(i,j,k)+a_p(i,j,k,4)*pskr(i,j,k)
+a_p(i,j,k,5)*pskr(i,j,k)+a_p(i,j,k,6)*pskr(i,j,k)
+a_p(i,j,k,7)*pskr(i,j,k)+a_p(i,j,k,8)*pskr(i,j,k)
+a_p(i,j,k,9)*pskr(i,j,k)+a_p(i,j,k,10)*pskr(i,j,k)
+a_p(i,j,k,11)*pskr(i,j,k)+a_p(i,j,k,12)*pskr(i,j,k)
+a_p(i,j,k,13)*pskr(i,j,k)+a_p(i,j,k,14)*pskr(i,j,k)
+a_p(i,j,k,15)*pskr(i,j,k)
enddo
enddo
enddo
```

PROCEDURE

- The stencil routine is timed by setting PGI_ACC_TIME=1 environment variable.
- For each problem-resolution combination, we average timings for each code option over three runs.

Example result (MASP LARGE):



SELECTED OPENACC DIRECTIVE OPTIONS

	[A1]	[L1]	[L2]	[L3]	[C1]	[E1]	[A2]	[C2]	[A3]	[C3]	[E2]	[W]
orig (i,j,k,ii)	kernels async(1)	loop independent	loop independent	loop independent		end kernels	kernels async(2)		kernels async(3)		end kernels	wait(1,2,3)
cache_clause	kernels async(1)	loop independent	loop independent	loop independent	cache (ps_r(i-1:i+1,j-1:j+1,k-1:k+1))	end kernels	kernels async(2)	cache (ps_r(i-1:i+1,j-1:j+1,k-1:k+1))	kernels async(3)	cache (ps_r(i-1:i+1,j-1:j+1,k-1:k+1))	end kernels	wait(1,2,3)
cache_clause_center	kernels async(1)	loop independent	loop independent	loop independent	cache (ps_r(i-1:i+1,j-1:j+1,k-1:k+1))	end kernels	kernels async(2)	cache (ps_r(i-1:i+1,j-1:j+1,k-1:k+1))	kernels async(3)	cache (ps_r(i-1:i+1,j-1:j+1,k-1:k+1))	end kernels	wait(1,2,3)
cache_friendly (ii,i,j,k)	kernels async(1)	loop independent	loop independent	loop independent		end kernels	kernels async(2)		kernels async(3)		end kernels	wait(1,2,3)
cache_f_par_1ker_collapse (ii,i,j,k)	parallel	loop collapse(3)				end kernels					end parallel	
inner_worker	kernels async(1)	loop independent	loop independent	loop independent worker		end kernels	kernels async(2)		kernels async(3)		end kernels	wait(1,2,3)
ker_collapse	kernels async(1)	loop independent collapse(3)				end kernels	kernels async(2)		kernels async(3)		end kernels	wait(1,2,3)
ker_collapse_inner2	kernels async(1)	loop independent collapse(2)				end kernels	kernels async(2)		kernels async(3)		end kernels	wait(1,2,3)
ker_collapse_inner2v	kernels async(1)	loop independent collapse(2) vector				end kernels	kernels async(2)		kernels async(3)		end kernels	wait(1,2,3)
ker_collapse_outer2	kernels async(1)	loop independent collapse(2)				end kernels	kernels async(2)		kernels async(3)		end kernels	wait(1,2,3)
ker_collapse_outer2gv	kernels async(1)	loop independent collapse(2) gang		loop independent vector		end kernels	kernels async(2)		kernels async(3)		end kernels	wait(1,2,3)
par_1ker	parallel	loop		loop		end parallel			parallel async(3)		end parallel	wait(1,2,3)
par_1ker_collapse	parallel	loop collapse(3)				end parallel			parallel async(3)		end parallel	wait(1,2,3)
par_1ker_tile_2_4_4	parallel	loop tile(2,4,4)				end parallel			parallel async(3)		end parallel	wait(1,2,3)
par_1ker_tile_8_8_8	parallel	loop tile(8,8,8)				end parallel			parallel async(3)		end parallel	wait(1,2,3)
par_collapse	parallel	loop collapse(3)				end parallel	parallel async(2)		parallel async(3)		end parallel	wait(1,2,3)
par_default	parallel async(1)	loop	loop	loop		end parallel	parallel async(2)		parallel async(3)		end parallel	wait(1,2,3)
tile_x_x_x	kernels async(1)	loop independent tile(x,x,x)				end kernels	kernels async(2)		kernels async(3)		end kernels	wait(1,2,3)
tile_8_8_lower	kernels async(1)	loop independent tile(8,8)				end kernels	kernels async(2)		kernels async(3)		end kernels	wait(1,2,3)
tile_8_8_upper	kernels async(1)	loop independent tile(8,8)				end kernels	kernels async(2)		kernels async(3)		end kernels	wait(1,2,3)

Note: Variations in gang, worker, and vector sizes/topology were explored in [Caplan et al 2019 J. Phys. Conf. Ser. 1225 012012], and it was found to be optimal to use compiler defaults.

COMPUTE ENVIRONMENT



NASA NAS HECC
Pleiades Cluster



V100 SXM2 32GB



PGI v18.10 -"O3"



OpenMPI 2.1.2

"ta=tesla:cc70,cuda10.0"

RESULTS

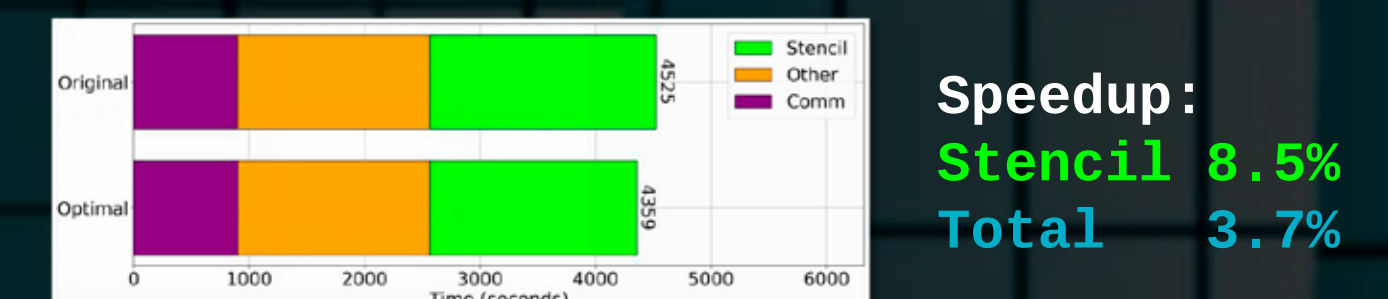
	SMALL			MEDIUM			LARGE		
	MASZB	MAS HELIO	MASP	MASZB	MAS HELIO	MASP	MASZB	MAS HELIO	MASP
orig	0.10	0.03	0.15	0.70	0.45	2.56	2.90	1.32	15.78
cache_clause	0.11	0.04	0.17	0.83	0.50	3.01	3.19	1.45	17.30
cache_clause_center	0.10	0.04	0.16	0.75	0.47	2.71	2.99	1.36	16.24
cache_friendly	0.10	0.04	0.16	0.78	0.49	2.82	3.28	1.50	17.86
cache_f_par_1ker_collapse	0.10	0.03	0.15	0.77	0.49	2.79	3.23	1.48	17.72
inner_worker	0.09	0.03	0.15	0.64	0.40	2.33	2.60	1.19	14.24
ker_collapse	0.09	0.03	0.14	0.62	0.39	2.27	2.53	1.16	13.86
ker_collapse_inner2	0.09	0.03	0.14	0.65	0.41	2.37	2.66	1.22	14.56
ker_collapse_inner2v	0.09	0.03	0.14	0.65	0.41	2.37	2.66	1.22	14.56
ker_collapse_outer2	0.09	0.03	0.14	0.64	0.40	2.33	2.59	1.19	14.17
ker_collapse_outer2gv	0.11	0.04	0.16	0.85	0.54	3.10	3.50	1.60	19.16
par_1ker	0.16	0.06	0.24	0.75	0.56	2.72	3.48	1.60	19.05
par_1ker_collapse	0.08	0.03	0.13	0.61	0.38	2.21	2.58	1.18	14.11
par_1ker_tile_2_4_4	0.14	0.05	0.21	1.70	1.04	5.99	15.70	7.18	85.76
par_1ker_tile_8_8_8	0.09	0.03	0.14	0.73	0.45	2.68	2.90	1.32	15.82
par_collapse	0.09	0.03	0.14	0.63	0.39	2.27	2.53	1.16	13.87
par_default	0.16	0.06	0.25	0.73	0.56	2.67	3.42	1.57	18.69
tile_16_8_8	0.12	0.04	0.18	0.84	0.50	3.03	3.25	1.49	17.79
tile_2_2_2	0.13	0.05	0.19	1.45	0.92	5.22	6.61	3.03	36.04
tile_2_4_4	0.11	0.04	0.17	0.82	0.49	2.95	4.91	2.26	27.03
tile_3_3_3	0.10	0.04	0.16	0.76	0.47	2.76	3.70	1.69	20.16
tile_4_4_4	0.10	0.03	0.15	0.73	0.45	2.64	3.02	1.39	16.55
tile_4_4_8	0.10	0.04	0.15	0.76	0.46	2.79	2.97	1.36	16.21
tile_8_4_4	0.09	0.03	0.15	0.70	0.43	2.53	2.88	1.32	15.79
tile_8_8_16	0.12	0.04	0.19	0.87	0.54	3.15	3.43	1.57	18.75
tile_8_8_8	0.11	0.04	0.16	0.78	0.46	2.84	2.96	1.36	16.20
tile_8_8_lower	0.17	0.06	0.26	0.79	0.79	2.89	7.28	3.34	39.92
tile_8_8_upper	0.42	0.15	0.64	11.85	7.92	43.01	55.82	25.60	305.70

Green: Faster
Yellow: Slower

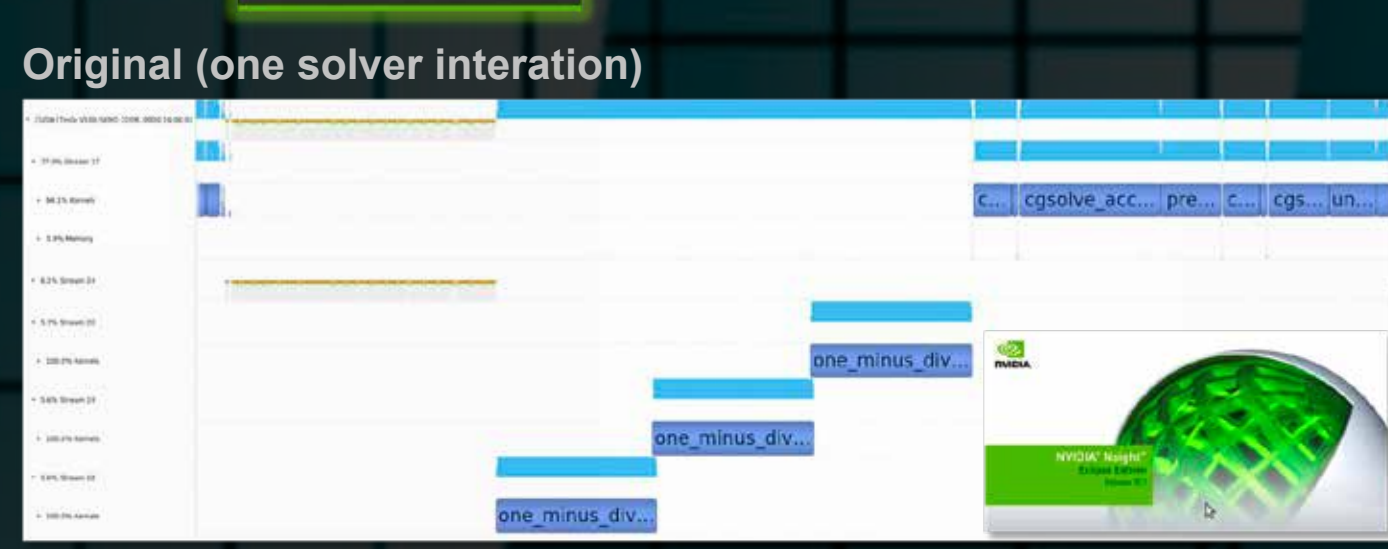
Cyan: Optimal
Red: Poor

PRODUCTION RUN

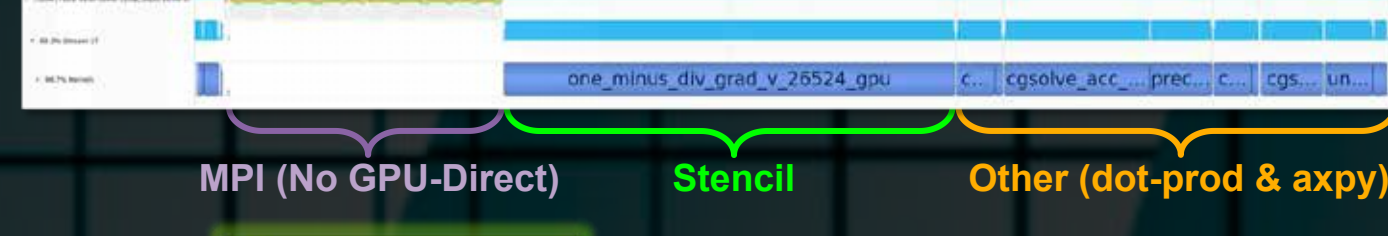
We compare the original and optimal codes on a full length MASZB production run (resolution 160x267x246)



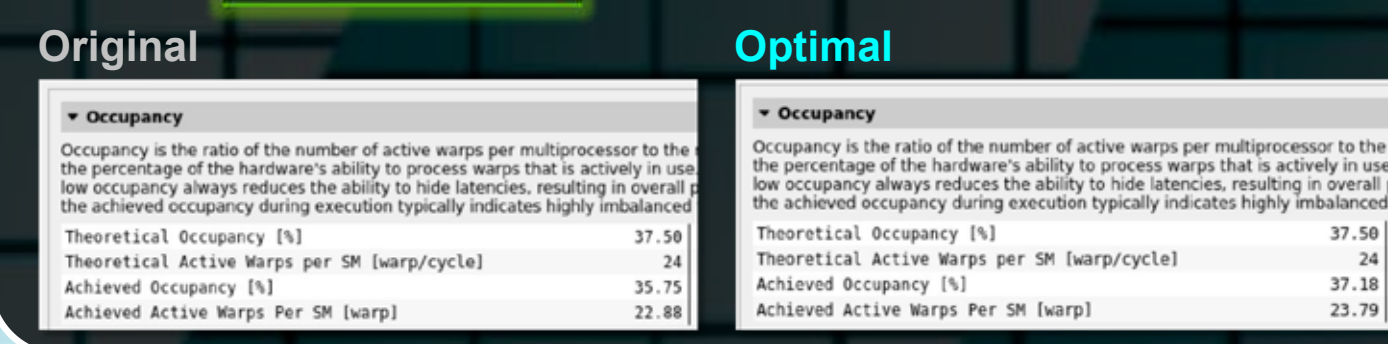
Using NVIDIA NSIGHT SYSTEMS we profile the stencil kernels:



Optimal (one solver iteration)



Using NVIDIA NSIGHT COMPUTE we analyze kernel occupancy:



SUMMARY

- Stencil operations are difficult to optimize
- OpenACC (hammer) vs. CUDA (scalpel)
- Many types of "hammer" possible due to directive options
- Optimal result using collapse not intuitive, but can yield over a 10% improvement in stencil time
- Results are size-dependent
- One can use a hammer the "wrong" way and get very bad results!
- Optimal result will be applied to other stencil-like loops in the code for further performance gains