Evaluation of Programming Models and Performance for Stencil Computation on GPGPUs

Baodi Shan[1,2], Mauricio Araya-Polo[1]

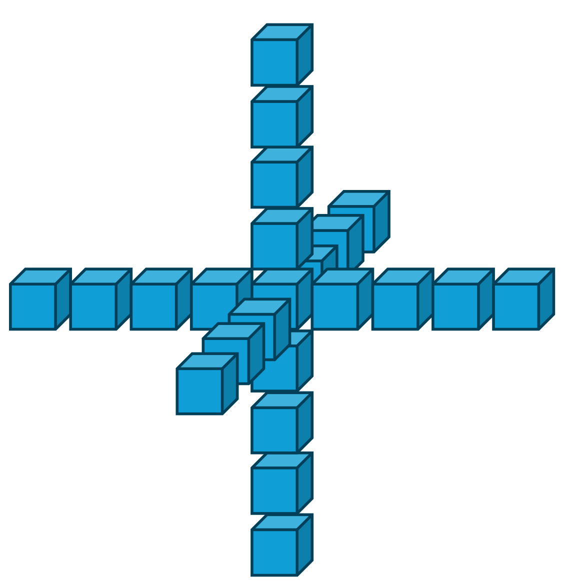
[1]TotalEnergies EP Research & Technology US, LLC. [2]Stony Brook University

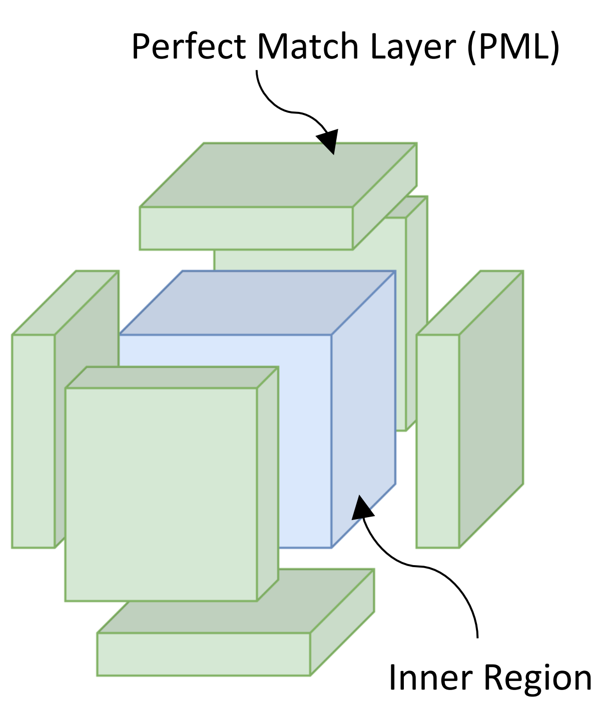
Introduction

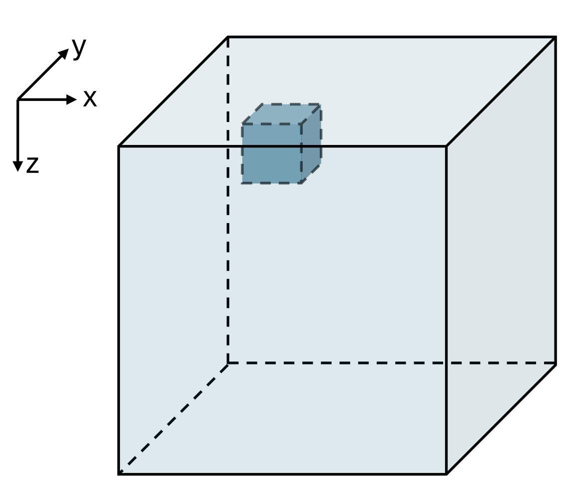
GPGPUs are widely used in HPC. Therefore, it is crucial to experiment and discover how to better utilize their latest generations of relevant applications. We introduce highly tuned stencil-based kernels for NVIDIA A100 and H100(inside Grace Hopper Superchip) GPGPUs. Performance results yield useful insights into the behavior of this type of computation for these new accelerators. This knowledge can be leveraged by many scientific applications which involves stencil computations. Further, evaluation of three different programming models: CUDA, OpenACC, and OpenMP target offloading is conducted on aforementioned accelerators.

We extensively study the performance and portability of various kernels under each programming model and provide corresponding optimization recommendations. Furthermore, we compare the performance of different programming models on the mentioned architectures. Up to 58% performance improvement was achieved against the previous GPGPU generation for a highly optimized kernel of the same class, and up to 42% for all classes. In terms of programming models, and keeping portability in mind, optimized OpenACC implementation outperforms OpenMP implementation by 33%. If portability is not a factor, the best CUDA implementation outperforms the optimized OpenACC one by 2.1x.

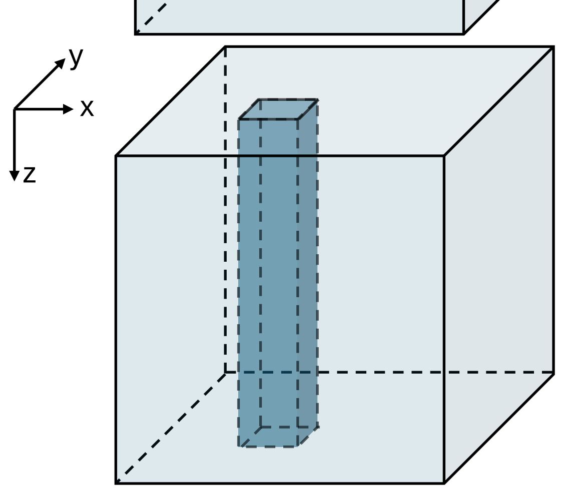
Stencil Computation







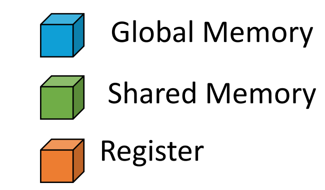
3D Blocking, each data region is segmented into 3D blocks aligned with the axis.



2.5D Blocking, we divide the data domain along the inner two dimensions, X and Y, and conduct a streaming computation along the outermost Z dimension.

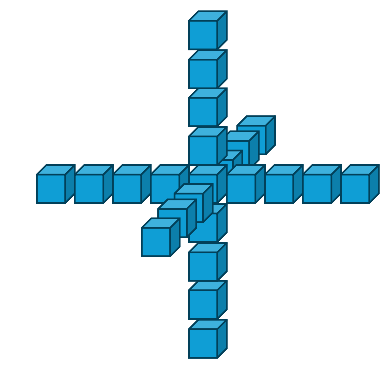
CUDA Kernels

Grid Size: 1024³, 1000 Steps, Unit: second



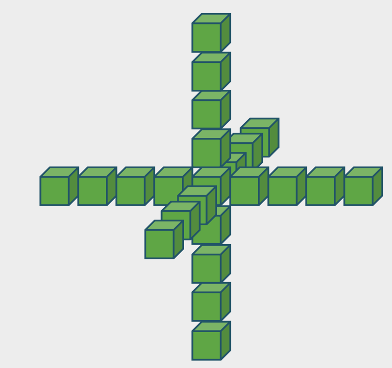
GMEM: Each thread fetches all array elements it needs to compute the stencil at one point in the domain directly from global memory.

A100: 27.058
H100 w/o cluster: 11.710
H100 w/ cluster: 11.315



SMEM: Threads in a block collaboratively fetch array elements from global memory, store them into shared memory, synchronize, and perform the stencil computation on array elements fetched from shared memory.

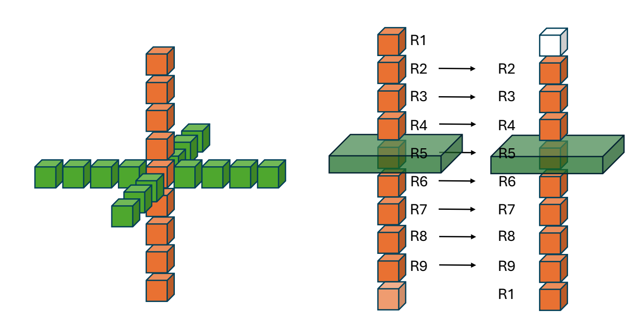
A100: 23.740
H100 w/o cluster: 11.635
H100 w/ cluster: 11.751



St_Reg_Fixed: Maintain array elements for an "active plane" in shared memory and points for the streaming dimension in registers. At each step in the computation, each thread replaces a trailing element in a register with a leading element.

Best in H100

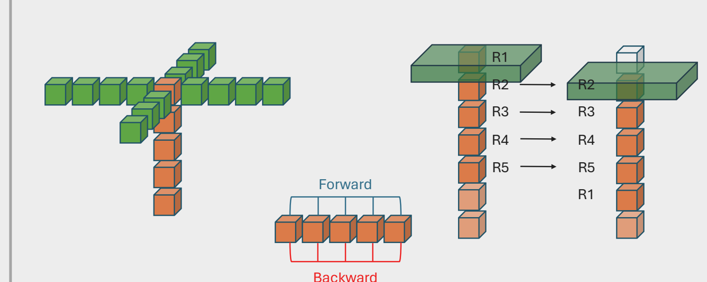
A100: 19.908
H100 w/o cluster: 9.445
H100 w/ cluster: 9.434



St_Semi: Compute the stencil in two steps rather than one, saving partial results in shared memory, which changes the balance of loads and stores.

Best in A100

A100: 16.154
H100 w/o cluster: 9.654
H100 w/ cluster: 10.435

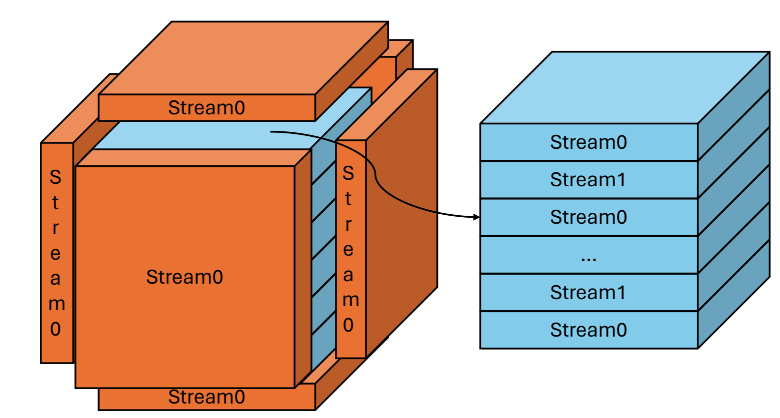


Kernel	Device & Configure	Memory Throughput(GB/s)	Device Memory Read/Write(GB)
St_Reg_Fixed	A100	621.87	34.40
	H100 w/o cluster	1110(78.49%)	23.55(-31.54%)
	H100 w/ cluster	1100(76.89%)	23.06(-32.97%)
St_Semi	A100	519.70	26.12
	H100 w/o cluster	950.54(82.90%)	22.39(-14.28%)
	H100 w/ cluster	935.73(80.05%)	21.91(-16.12%)

- According to the table, for these two kernels, when the thread block cluster is adopted, the read and write volume of the GPU memory significantly decreases, while the drop in throughput is not as severe as that in GPU memory read and write volume.
- For *St_Reg_Fixed*, the upgrade of the H100 over the A100 resulted in a significant reduction in memory reads and writes, a reduction that had a relatively weak effect on *St_Semi*.

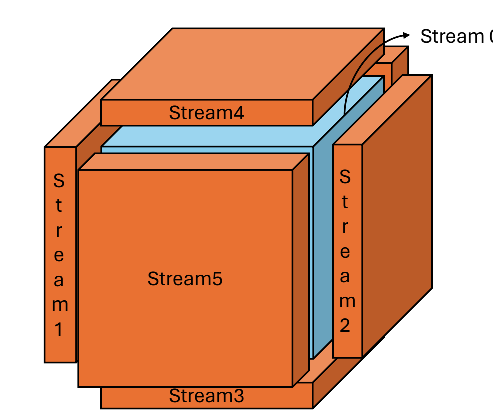
Directive-based Programming Model

Grid Size: 1024³, 1000 Steps, Unit: second



OpenACC
A100: 53.188
H100: 23.196

OpenACC-async
A100: 44.222
H100: 19.229



OpenMP-nowait
A100: 58.568
H100: 29.527

Comparison of Different Programming Models

	T4	A100	H100
CUDA	31.775s	5.706s	3.364s
OpenACC	81.718s	11.260s	6.276s
OpenMP	139.358s	18.012s	9.651s
(OpenACC - CUDA)/OpenACC	0.611	0.493	0.464
(OpenMP - CUDA)/OpenMP	0.772	0.683	0.651
(OpenACC - OpenMP)/OpenMP	0.414	0.375	0.349

As the generations of GPGPUs continue to update, the gap between the three models is gradually narrowing.

Full paper: Abstract: Poster: