Time Blocking

Time tiling/blocking is a technique for increasing data locality. Data that are already loaded in a local memory (shared memory for the GPU) are reused to calculate multiple time steps. Intra-kernel, thread-block communications are not feasible on GPUs, so additional ghost zone has to be read from the thread-blocks. Larger ghost zone leads to more redundant computations and more requested shared memory, potentially compromising performance.

Previous Research

[2] conducts a survey on the optimal ghost zone size used for time blocking on various stencils, whereas [1] uses time blocking as an optimization in their automatic stencil generator framework. Both papers study CUDA enabled GPUs.

Results show that time tiling is most suitable for 1d and 2d stencils. [1] achieves at least 3x for 1d stencils and 1-3x for 2d. Similarly [2] achieves better performance with time tiling for 1d and 2d stencils. In both papers, 3d stencils (Jacobi 3-D, Cell) don't benefit at all from time blocking.

Since, thread-blocks require all the stencil data to be available, a 3d block must be fully loaded into shared memory. Three dimensional matrices use a lot of memory, even for smaller stencils

Time augmented stencil (?)

Our cuda 3d stencil implementation loads only a 2d tile of data and computes its effect to neighboring cells. This method does not allow the use of the aforementioned technique, as it demands a holistic approach.

The idea behind time augmented stencil, is to combine multiple time steps and create a larger stencil. Depending on its density and size, the new number of computations can be more than double, but the effect on shared memory is the same as the time tiling on 2d stencils.

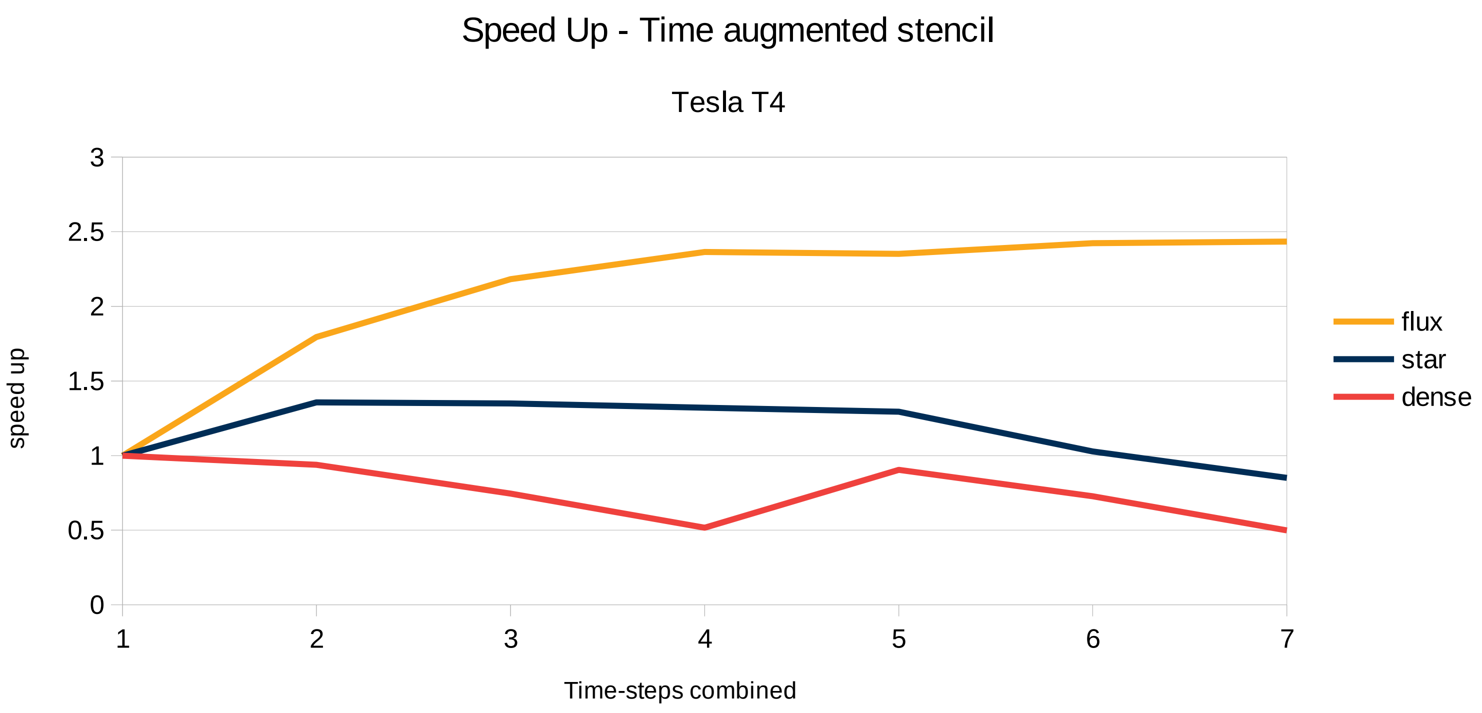
The augmented stencil can be created by convoluting the stencil with itself or by substituting the stencil formula for *t+1* to the formula for *t+2,* therefore , obtaining a formula for *t+2* by using the data at *t.* The second method was chosen for the framework as it leaves zeros where there are not calculations. The library for the convolution had floating point errors at zero valued cells and that could be mistaken for computations by the code generation system.

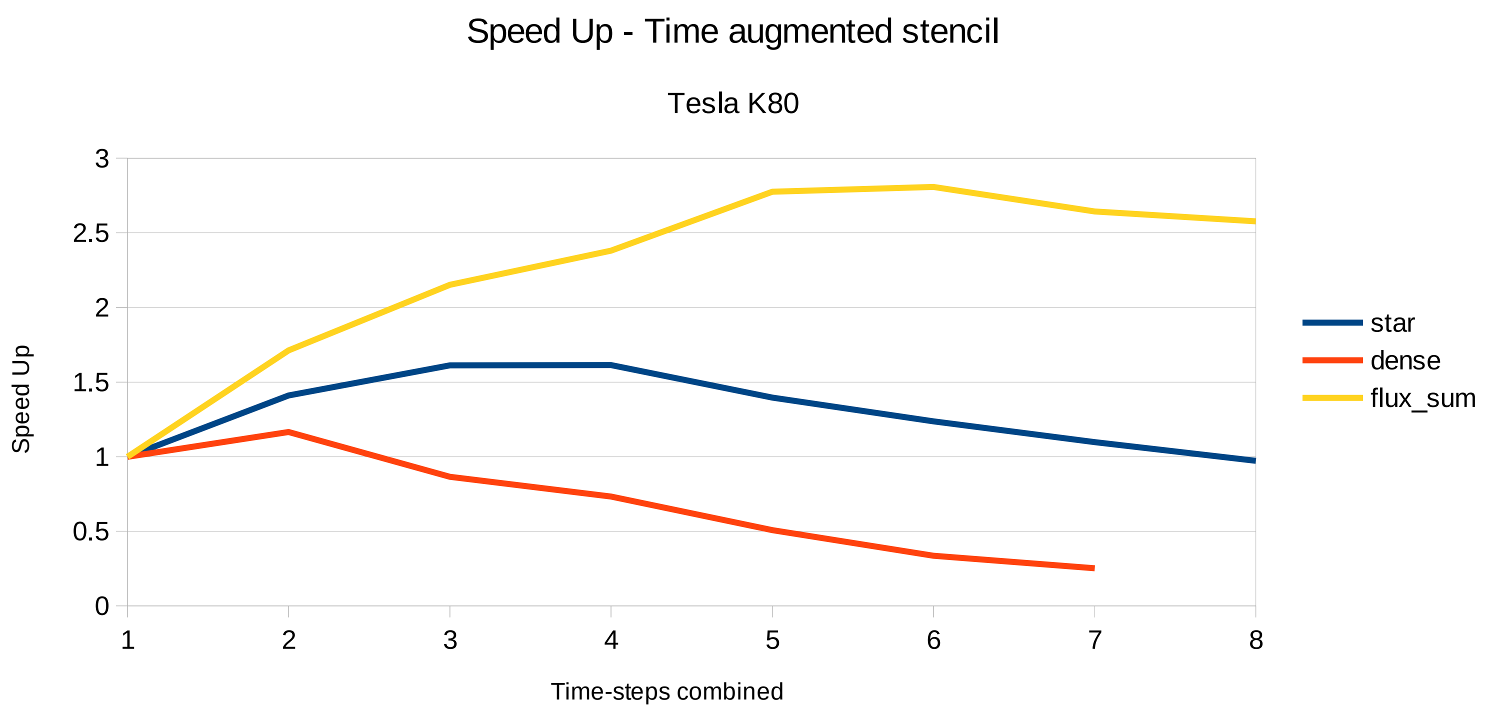
Benchmarks

3 types of stencil were benchmarked with different densities:

• [3] flux summation	4-points	0.14 density
• star	9-points	0.33 density
• dense	27-points	1 density

Flux summation can be considered as half a star stencil, extending only to positive indices from the center. The radius of the stencils tested was equal to 1, meaning they could be encompassed by a 3x3x3 block.





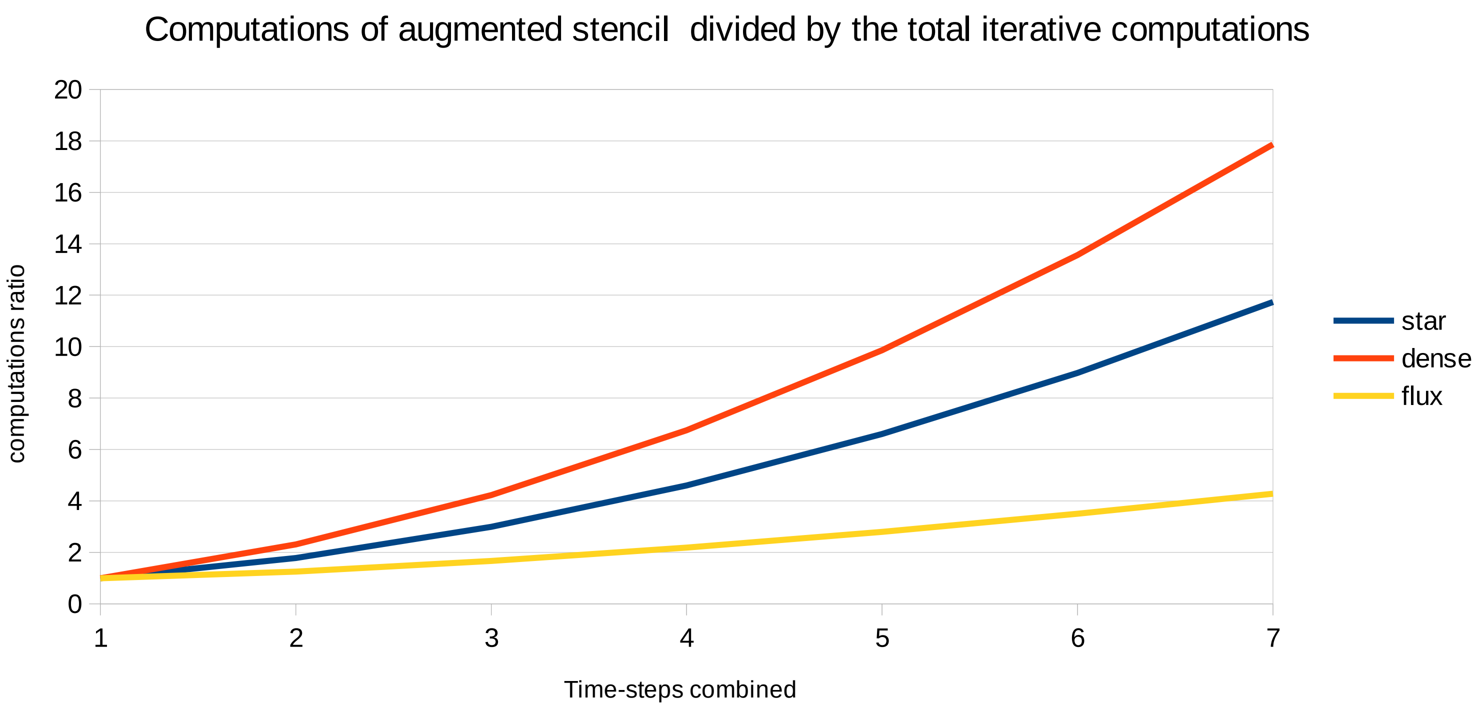
Dense stencil benefits the least from the time augmented stencil technique. However, for 2 time-steps on the Tesla K80, t. augmented stencil is x1.16 faster.
Star stencil achieves 1.35x and 1.6x speed up for 4 time-steps and **flux summation** is 2.8x faster for 6 time-steps on Tesla K80 and 2.43x on Tesla T4 with a thin room for further improvement.

The success of this method is apparent by looking at the star stencil with radius = 1, identical to the Jacobi-3D [1] and similar to Cell [2] (See Appendix).

The problem with dense stencil is the polynomial increase in computations for each time-step combination.

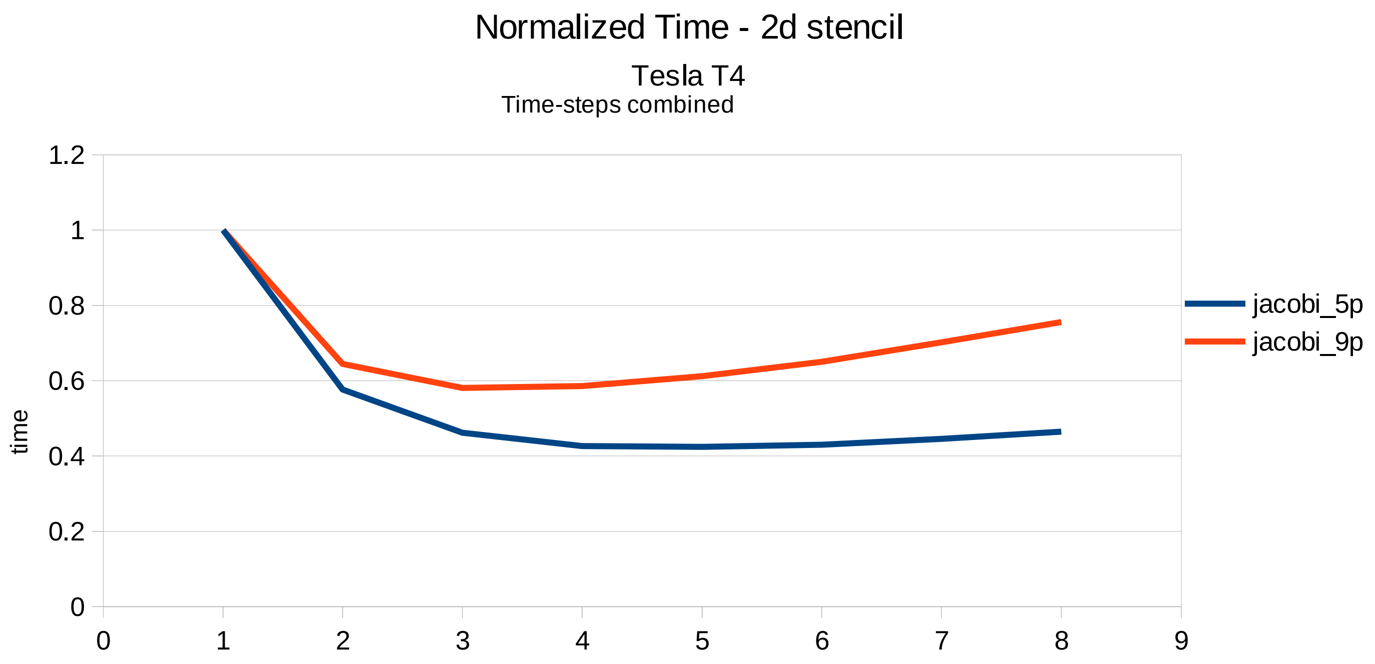
Denoting *c* is the computations of the initial stencil and *f(t)* is the number of computations for the time augmented stencil for *t* combined time-steps. The following graph shows the ratio $f(t) / t * c$

that could be interpreted as the ratio of "redundant" computations.



Although time augmented stencil does more than double computations, the global memory accesses are minimized (like time-tiling) but also the shared memory accesses and synchronization points (unlike time-tiling). The most important factor for the speed up, seems to be high occupancy with efficient usage of 2d shared memory.

The same 3d stencil framework can work on 2d stencils. Below are the results for 2-d Jacobi with radius 1 and 2.



The framework is not optimized for 2D stencil. The input was given as a 3d array with z dimension equal to 1. However, the results are comparable to [1] (See Appendix.)

Conclusion

Time blocking for 3d stencils is not suitable. Request for large shared memory negatively impacts occupancy, and therefore performance. Also it may exceed available resources for big stencils and/or many time-steps. The time augmented stencil is efficient on shared memory and postpones hitting the resources limit by many time-steps. Denser stencils still see no significant performance because of the “exploding” number of computation and register usage. Less dense stencil see a noteworthy speed up.

For dense stencils, other methods of optimizations can be used, such [4] frequency domain convolution.

References

[1] J. Holewinski, L.-N. Pouchet, and P. Sadayappan, “High-performance code generation for stencil computations on GPU architectures,” in *Proceedings of the 26th ACM international conference on Supercomputing*, New York, NY, USA, Jun. 2012, pp. 311–320, doi: 10.1145/2304576.2304619.

[2] J. Meng and K. Skadron, “A Performance Study for Iterative Stencil Loops on GPUs with Ghost Zone Optimizations,” *Int J Parallel Prog*, vol. 39, no. 1, pp. 115–142, Feb. 2011, doi: 10.1007/s10766-010-0142-5.

[3] Brandvik, isc-20111, https://www.nvidia.com/content/PDF/isc-2011/Brandvik.pdf

[4] G. Papamakarios, G. Rizos, and N. P. Pitsianis, “FLCC: A Library for Fast Computation of Convolution and Local Correlation Coefficients,” p. 8.

Appendix

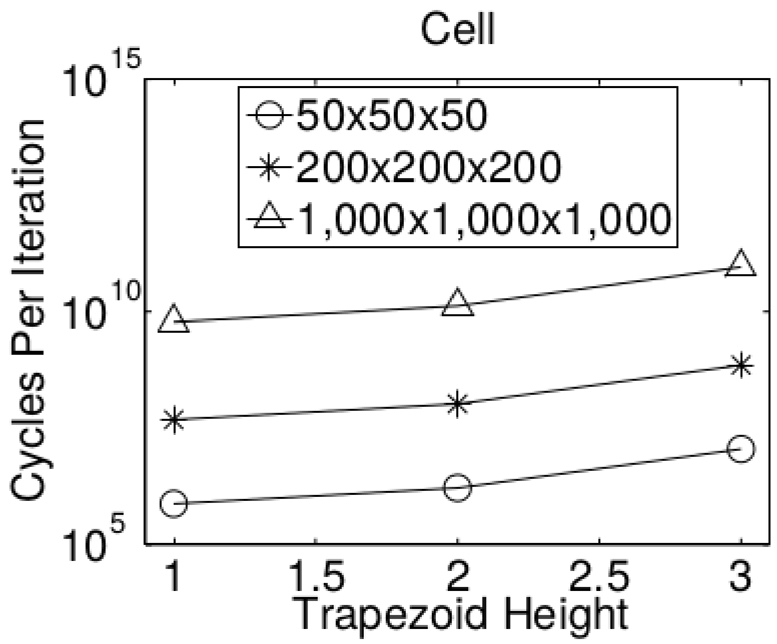


Figure from [2]. Cell is a 3d kind of stencil (from Conway's game of life), that is similar to the star stencil. Trapezoid height shows the number of combined time-steps.

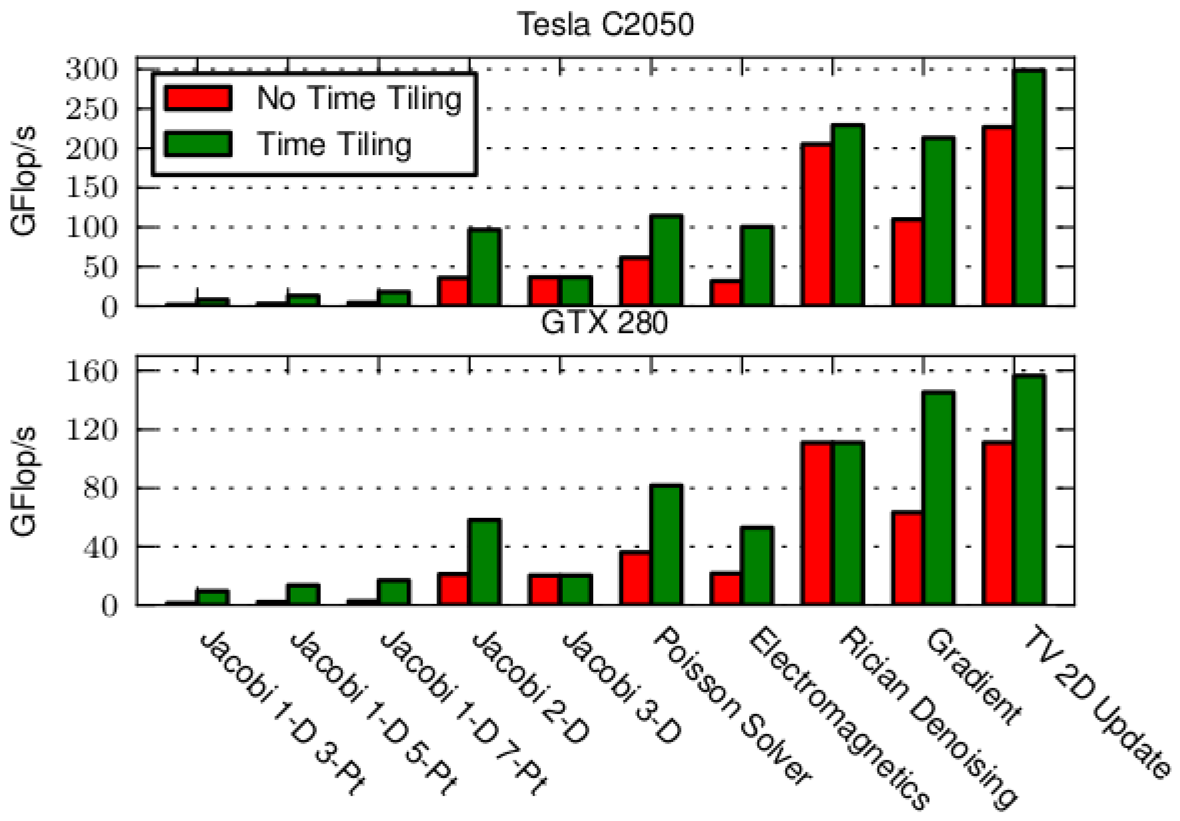


Figure from [1]. Jacobi 3-D is the only 3 dimensional stencil. No impact in performance with time tiling.

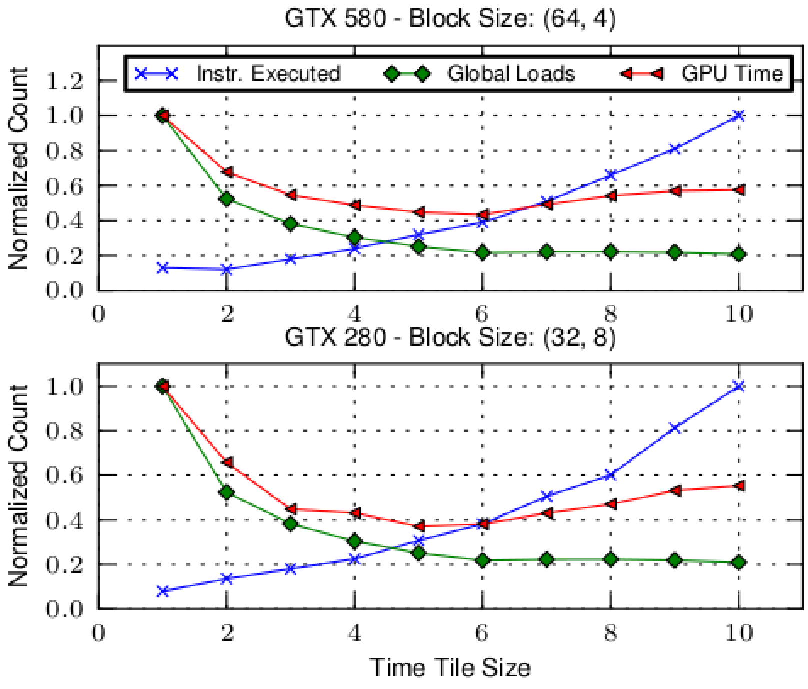


Figure from [1]. 5-points 2d Jacobi stencil. Normalized time reaches 0.4.