Performance of OpenMP loop transformations for the acoustic wave stencil on GPUs

Jaime Freire de Souza, Leticia Suellen Farias Machado, Edson Satoshi Gomi, Claude Tadonki, Simon Mcintosh-Smith, Hermes Senger

To cite this version:

Jaime Freire de Souza, Leticia Suellen Farias Machado, Edson Satoshi Gomi, Claude Tadonki, Simon Mcintosh-Smith, et al.. Performance of OpenMP loop transformations for the acoustic wave stencil on GPUs. SC22 The International Conference for High Performance Computing, Networking, Storage, and Analysis, Nov 2022, Dallas, United States. . hal-03888100

HAL Id: hal-03888100

https://hal-mines-paristech.archives-ouvertes.fr/hal-03888100

Submitted on 29 Dec 2022

Performance of OpenMP loop transformations for the acoustic wave stencil on GPUs

J.F.D Souza [1,2]   L.S.F. Machado [1]   E. Gomi [2]   C. Tadonki [3]   S. McIntosh-Smith [4]   H. Senger [1,2]

[1]Universidade Federal de São Carlos (UFSCar), Brazil   [2]Universidade de São Paulo, Brazil   [3]Mines Paristech/PSL, France   [4]University of Bristol, UK

OpenMP and heterogeneous architectures

- The support for heterogeneous architectures was introduced in OpenMP 4.0 and OpenMP 4.5.
- OpenMP 5.1 introduced *unroll* and *tiling* loop transformations. Code offloading for these transformations is supported in Clang 13.
- Despite being around for decades, the availability of these transformations for portability across compilers in OpenMP is relatively new. And we exercise it.

The application kernel

Kernel of seismic applications such as in full-waveform inversion (FWI) and reverse-time migration (RTM), the propagation of acoustic waves can be be modeled as follows:

$$\frac{1}{v_p^2}\frac{\partial^2 p(\mathbf{x},\mathbf{y},\mathbf{z},t)}{\partial t^2} - \nabla^2 p(\mathbf{x},\mathbf{y},\mathbf{z},t) = f(\mathbf{x},\mathbf{y},\mathbf{z},t) \tag{1}$$

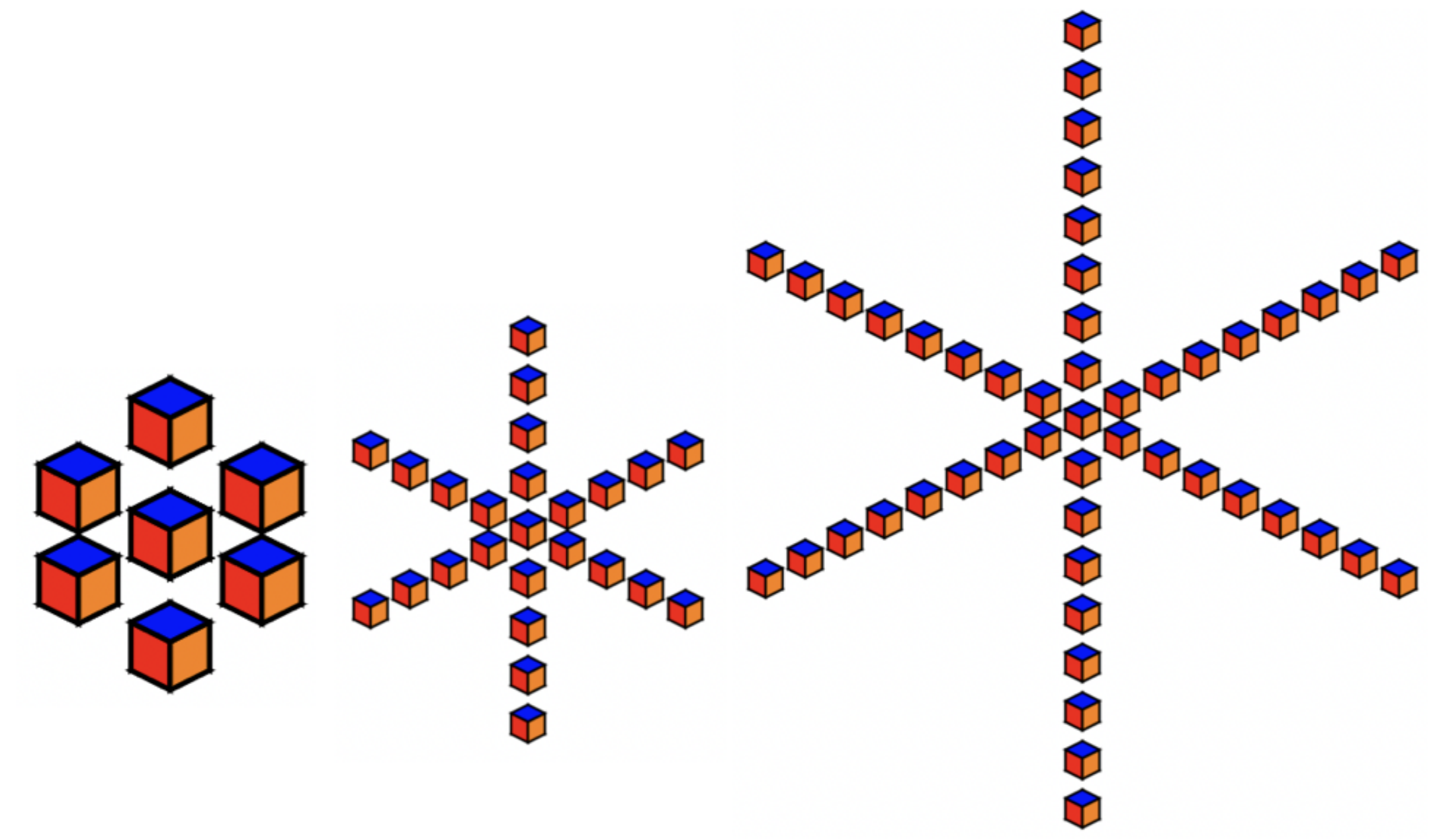
where v_p is the velocity, $p(\mathbf{x},\mathbf{y},\mathbf{z},t)$ is the pressure field, and $f(x,y,z,t)$ is the source. This PDE is solved by finite differences on a 3D grid spaced by distances Δx, Δy, and Δz. By using second-order central differences , we get the following discretized equation (for 2nd-spatial order), where Δt is the time increment:

$$\begin{aligned} p_{i,j,k}^{(n+1)} = {} & 2p_{i,j,k}^{(n)} - p_{i,j,k}^{(n-1)} \\ & + 2\Delta t^2 . v^2 \Bigg(\frac{p_{i+1,j,k}^{(n)} - 2p_{i,j,k}^{(n)} + p_{i-1,j,k}^{(n)}}{\Delta x^2} \\ & + \frac{p_{i,j+1,k}^{(n)} - 2p_{i,j,k}^{(n)} + p_{i,j-1,k}^{(n)}}{\Delta y^2} \\ & + \frac{p_{i,j,k+1}^{(n)} - 2p_{i,j,k}^{(n)} + p_{i,j,k-1}^{(n)}}{\Delta z^2} \Bigg) \end{aligned} \tag{2}$$

A major performance issue with stencils is their high demand for memory access.

```
for(int t = 0; t < time_steps; t++) {
  #pragma omp target teams distribute parallel for \
 collapse(3)
  for(int i = radius; i < d1 - radius; i++){
    for(int j = radius; j < d2 - radius; j++){
      for(int k = radius; k < d3 - radius; k++){
        ...
        for(int ir = 1; ir <= radius; ir++){
          // stencil point calculation
```

Listing 1: The baseline strategy for the wave equation on GPUs.



a) space order 2   b) space order 8   c) space order 16

Figure 1: Shapes of 3D stencils used in the experiments.

Setup of Experiments

- Experiments on three GPU architectures (see Table 1)
- Discretized 2nd time order, space orders of 2, 8, and 16;
- Grid sizes: 256^3, 512^3, and 1024^3 points with 400, 800, and 1600 time steps;
- Float precision FP32, and FP64;
- Four strategies: collapse, unroll, tile, tile+unroll. For tilling, best block shapes were obtained via auto-tuning.

Table 1: GPUs architecture specifications.

	RTX 2080 Super	V100	A100
GPU Architecture	Turing	Volta	Ampere
SMs	48	80	108
CUDA cores / GPU	3072	5120	6912
Peak FP64 TFLOPS	0.35	7.8	9.7
Peak FP32 TFLOPS	11.2	15.7	19.5
Memory Size	8 GB	32 GB	40 GB
Memory Bandwidth	496 GB/s	900 GB/s	1555 GB/s
Shared Memory / SM	64 KB	96 KB	164 KB
L2 Cache Size	4 MB	6 MB	40 MB

```
for(int t = 0; t < time_steps; t++) {
  #pragma omp target teams distribute parallel for \
 collapse(3)
  #pragma omp tile sizes(BLOCK1,BLOCK2,BLOCK3)
  for(int i = radius; i < d1 - radius; i++){
    for(int j = radius; j < d2 - radius; j++){
      for(int k = radius; k < d3 - radius; k++){
        ...
        #pragma omp unroll full
        for(int ir = 1; ir <= radius; ir++){
          // stencil point calculation
```

Listing 2: Using tiling and unroll.

Table 2: Tile sizes applied to 2nd spatial order with FP64.

GPU	Grid size	Best tile sizes		
		Axis 1	Axis 2	Axis 3
RTX 2080	256^3	32	1	4
	512^3	16	1	4
V100	256^3	1	1	4
	512^3	8	1	2
	1024^3	8	1	2
A100	256^3	2	2	4
	512^3	2	1	4
	1024^3	8	1	4

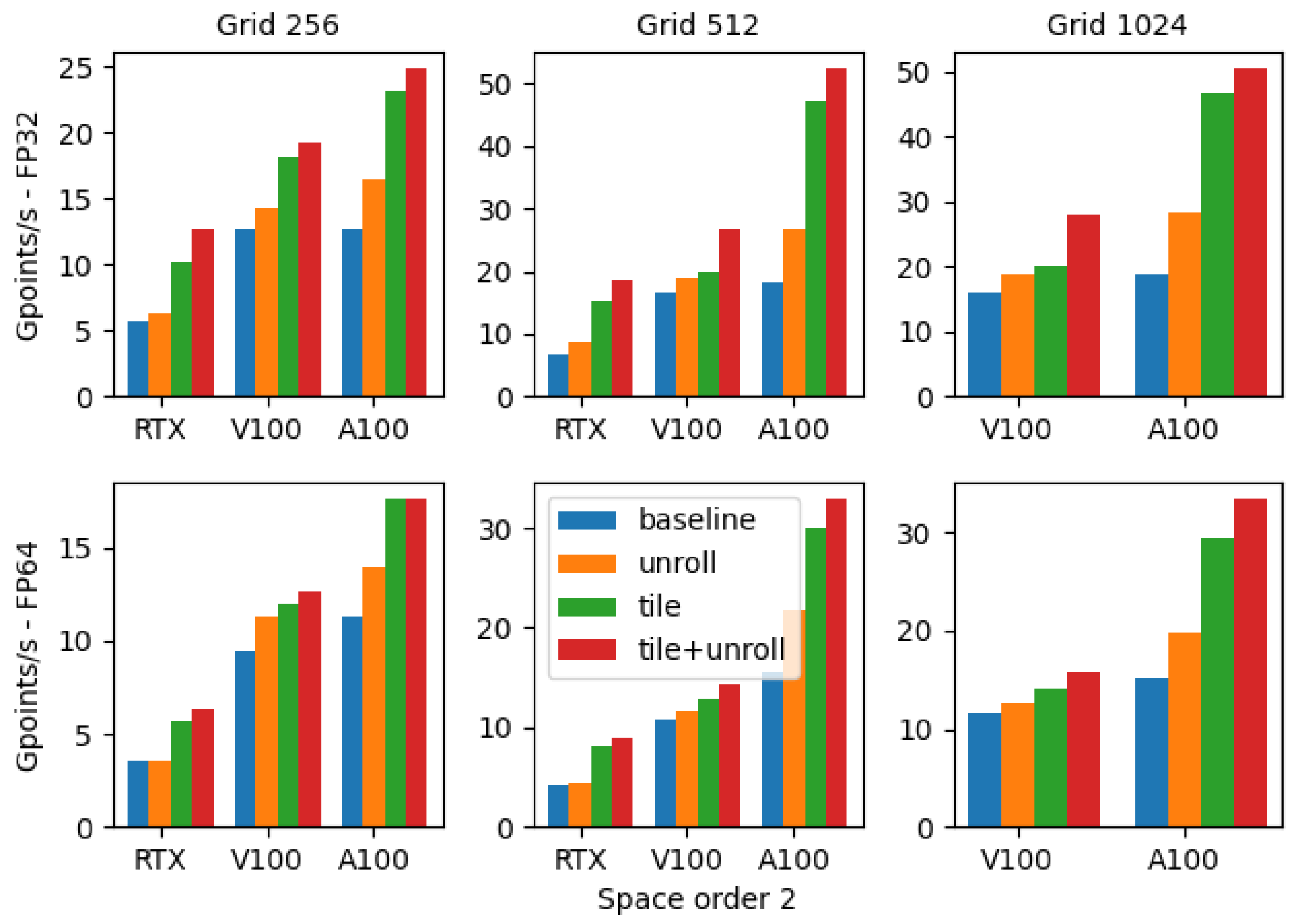


Figure 2: Gpoint/s for space order 2.

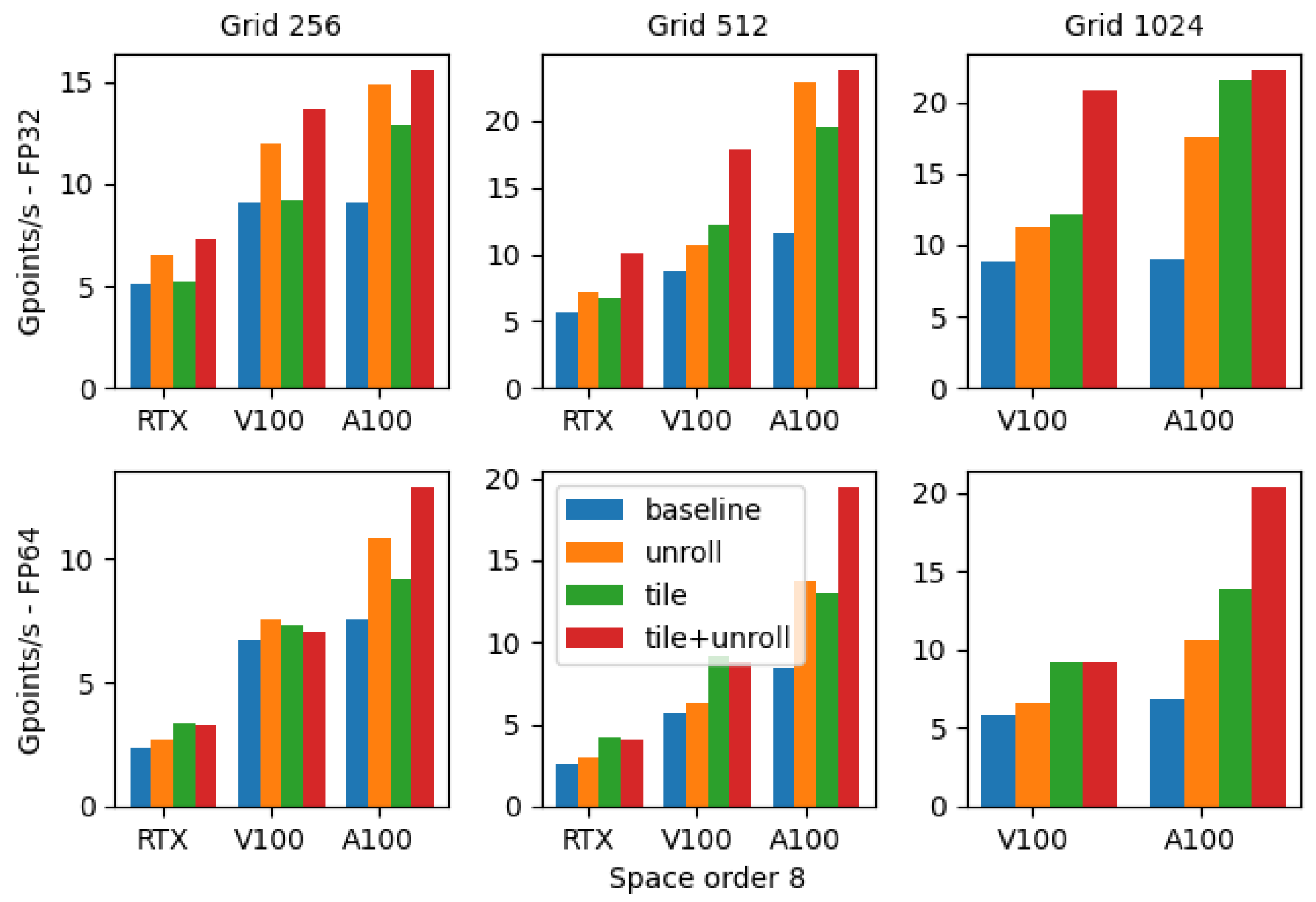


Figure 3: Gpoint/s for space order 8.

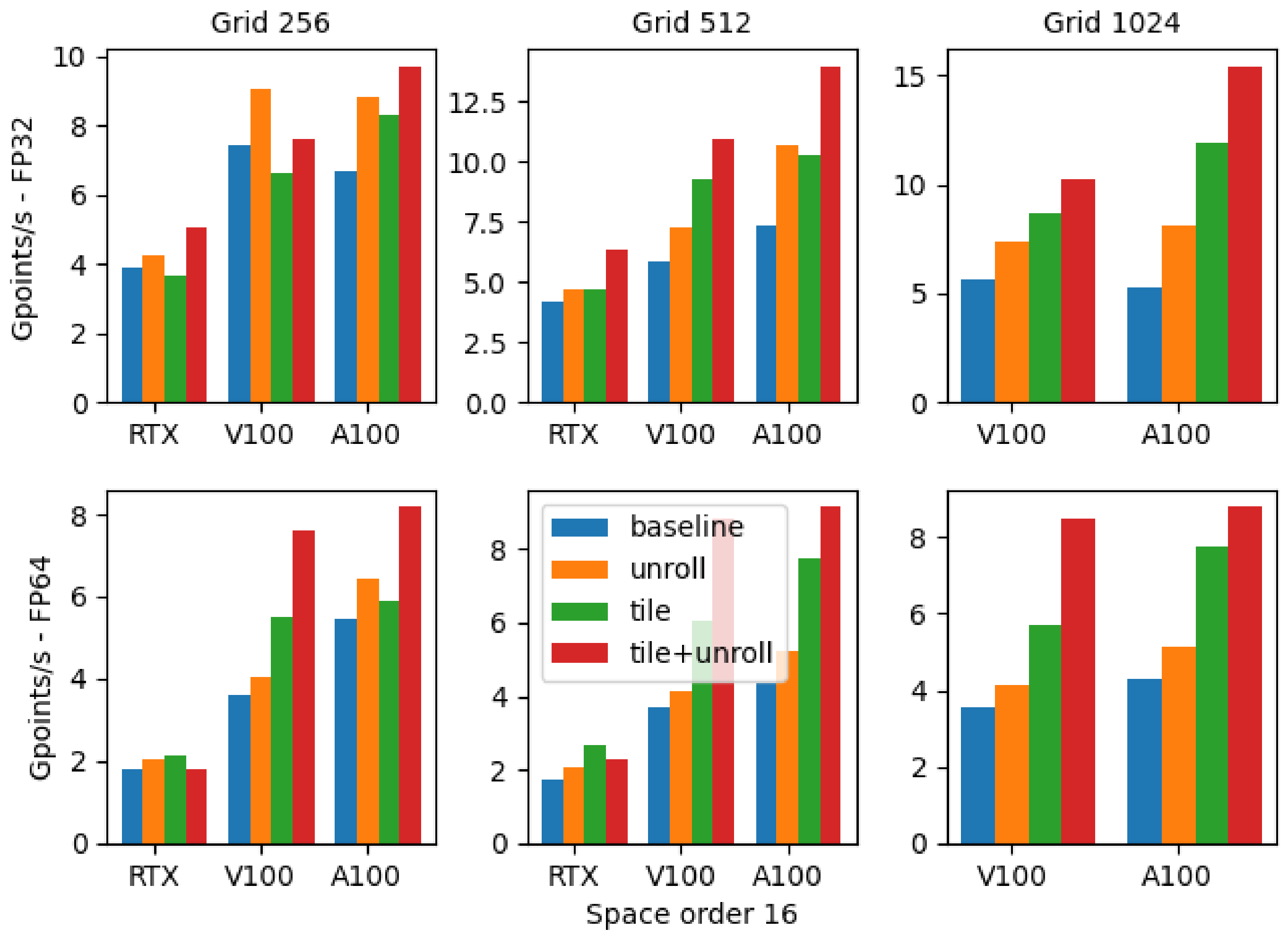


Figure 4: Gpoint/s for space order 16.

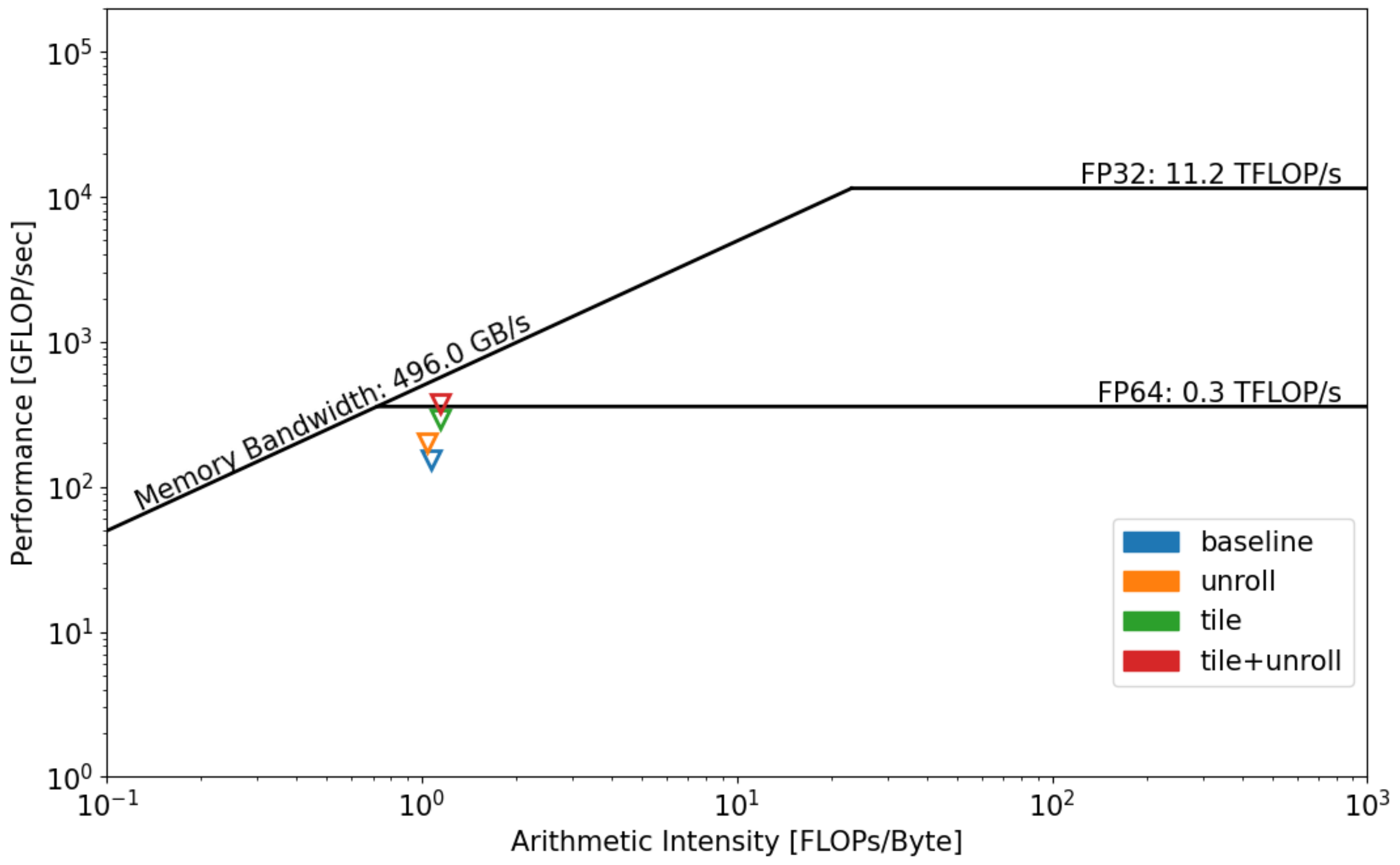


Figure 5: Roofline plot for FP32, 2nd space order on RTX 2080 Super.

Main Findings

- As a general remark, both loop transformations, unroll and tiling can yield significant improvements to the performance of the kernel evaluated on all GPUs evaluated.
- Performance gains ranged from 1.13x to 2.93x. In most scenarios, the best performance was achieved by combining unroll and tiling.
- The performance of tiling is highly sensitive to the choice of block size.