GPU accelerated computing towards a fast and scalable seismic wave modelling in SEISCOPE SEM46 code

Jian Cao[1*], Romain Brossier[1], Eduardo Cabrera[3], Josep de la Puente[3], Ludovic Métivier[1,2] and Alizia Tarayoun[1]

[1] Univ. Grenoble Alpes, ISTerre, F-38058 Grenoble, France
[2] CNRS, Univ. Grenoble Alpes, LJK, F-38058 Grenoble, France
[3] Barcelona Supercomputing Center, Geophysical Applications Group, Barcelona, Spain

Monday 30th May, 2022

Main objectives

This study aims at the development of GPU-accelerated code for a fast and scalable seismic wave modelling using the spectral element method (SEM) within the framework of our full waveform modelling and inversion code SEM46. Overall, it contains the single GPU algorithm investigation to explore the computational efficiency that stems from the application of Cartesian-based structured meshes and the multi-GPU implementation based on the domain-decomposition strategy.

New aspects covered

(1) Three types of parallel prototypes of SEM-based CUDA kernel are presented and compared in terms of modelling accuracy and computational efficiency. (2) To benefit from the Cartesian-based structured mesh, an element-wise parallelization with odd-even mesh coloring is proposed which achieves a significant speedup over the equivalent serial CPU reference code. (3) With the help of CUDA-AWARE MPI, an excellent scaling is obtained in the domain-decomposition-based multi-GPU implementation, which boosts its applicability on large-scale realistic problems.

Summary (200 words)

Modelling of seismic wave propagation is widely used in the study and imaging of the Earth's interior. Especially for the techniques of reverse time migration and full waveform inversion, an accurate and efficient seismic wave modelling solver plays a key role in handling complex and large-scale problems. In addition to developing novel modelling algorithms from a mathematical standpoint, it is necessary to study how to implement existing methods on modern heterogeneous high-performance computing (HPC) platforms, including at least one type of accelerator. Utilizing GPUs as accelerators has been shown to be attractive in the geophysical applications. To benefit from this multi-threaded architecture, we explore its implementation in the spectral element modelling (SEM) engine of our full waveform modelling and inversion code SEM46. Based on the features of SEM algorithm and the Cartesian-based structured mesh in SEM46, we investigate GPU kernels with three different parallel prototypes with particular focus on the modelling accuracy and computational efficiency. The memory constraint in the 3D implementation is addressed by domain decomposition using multiple GPUs with CUDA-AWARE MPI to achieve direct GPU-to-GPU communications. The resulting GPU solver exhibits a high speedup and excellent scaling over multi-GPUs, making it promising for large-scale realistic 3D problems.

Introduction

In seismology and exploration geophysics, seismic wave modelling serves as the foundation for understanding and interpreting geophysical data in the Earth's interior investigation. Full waveform inversion (FWI), as an efficient seismic imaging tool, has shown the potential of extracting high-resolution quantitative medium parameters of the subsurface using the full seismic data information (Virieux and Operto, 2009). Many successful FWI applications have been reported in the oil & gas community, especially for marine surveys. As the exploration targets are becoming larger (industry-sized 3D problems) and more complex (e.g. the consideration of elastic effects for reservoir characterization), this technique relies heavily on high-performance computing (HPC) for its success. Over the past two decades, using GPUs as accelerators is getting more and more popular in the speedup of key kernels for seismic imaging and inversion applications, such as their modelling kernels (Micikevicius, 2009; Komatitsch et al., 2010), since it has a wider memory bandwidth and light-weight parallelism to launch thousands of threads concurrently. In this paper, we investigate the GPU parallelization of spectral-element-based seismic wave modelling within the framework of our full waveform modelling and inversion code SEM46, thereby extending it to be executable on multi-GPU clusters with a high speedup and scalability.

SEM46: Spectral-element based full waveform modelling and inversion code

SEM46 ("Spectral Element Method for Seismic Imaging at eXploration scale") is a code developed within the SEISCOPE project to perform 3D full waveform modelling and inversion for a wide range of exploration scenarios: acoustic media, anisotropic (visco)elastic media and fluid-solid coupled media (Trinh et al., 2019; Cao et al., 2022). Its modelling engine employs a classical hexahedra-based SEM frame (Komatitsch and Tromp, 1999) for solving the wave equations in the second-order formulation. Here we consider an elastic wave modelling scenario. The elastodynamics equation is given as

$$\rho \partial_{tt} \mathbf{u} = \nabla \cdot \boldsymbol{\sigma} + \mathbf{f}, \qquad \boldsymbol{\sigma} = \mathbf{C} : \boldsymbol{\varepsilon}, \qquad \boldsymbol{\varepsilon} = \frac{1}{2}\left[\nabla \mathbf{u} + (\nabla \mathbf{u})^T\right], \tag{1}$$

where $\mathbf{u}$ and $\mathbf{f}$ are the displacement and force vectors, $\boldsymbol{\sigma}$ and $\boldsymbol{\varepsilon}$ are the stress and strain tensors, and ρ and $\mathbf{C}$ are the medium density and elasticity tensor. In SEM, the weak form of PDE (1) is established by using high-order Lagrange polynomials as basis functions and Gauss–Lobatto–Legendre (GLL) quadrature for each discrete element. After the assembly, the resulting global semi-discretized system is

$$\mathbf{M}\partial_{tt}\mathbf{u} = -\mathbf{K}\mathbf{u} + \mathbf{F}, \tag{2}$$

where $\mathbf{M}$ and $\mathbf{K}$ denote the global mass and stiffness matrices, and vectors $\mathbf{u}$ and $\mathbf{F}$ correspond to the displacement field and source term in the entire computation domain. This semi-discretized linear system is solved using the second-order Newmark time scheme in a prediction-correction format (Komatitsch and Tromp, 1999). To get an intuitive view of global matrices $\mathbf{M}$ and $\mathbf{K}$, we illustrate them in symbolic form for a 1D case with three elements using second-order polynomials as basis functions (three points involved in GLL quadratures)

$$\mathbf{M} = \begin{pmatrix} M_1^{(1)} \\ M_2^{(1)} \\ \left[M_3^{(1)} + M_1^{(2)}\right] \\ M_2^{(2)} \\ \left[M_3^{(2)} + M_1^{(3)}\right] \\ M_2^{(3)} \\ M_3^{(3)} \end{pmatrix}, \mathbf{K} = \begin{pmatrix} K_{1,1}^{(1)} & K_{1,2}^{(1)} & K_{1,3}^{(1)} & & & & & \\ K_{2,1}^{(1)} & K_{2,2}^{(1)} & K_{2,3}^{(1)} & & & & & \\ K_{3,1}^{(1)} & K_{3,2}^{(1)} & \left[K_{3,3}^{(1)} + K_{1,1}^{(2)}\right] & K_{1,2}^{(2)} & K_{1,3}^{(2)} & & & \\ & & K_{2,1}^{(2)} & K_{2,2}^{(2)} & K_{2,3}^{(2)} & & & \\ & & K_{3,1}^{(2)} & K_{3,2}^{(2)} & \left[K_{3,3}^{(2)} + K_{1,1}^{(3)}\right] & K_{1,2}^{(3)} & K_{1,3}^{(3)} \\ & & & & K_{2,1}^{(3)} & K_{2,2}^{(3)} & K_{2,3}^{(3)} \\ & & & & K_{3,1}^{(3)} & K_{3,2}^{(3)} & K_{3,3}^{(3)} \end{pmatrix}. \tag{3}$$

It can be clearly seen that the global mass and stiffness matrices contain the sum of the values at the border between two elements, which implies that those nodes need to be carefully treated to avoid "race" problems in parallel executions. The existing HPC implementation of SEM46 is based upon a two-level MPI parallelization. The inner level performs a domain decomposition over a Cartesian-based structured mesh which is easily partitioned and accurately conforms to complex geological surfaces through vertical deformation of elements (Figure 3a). The outer level manages multi-sources in parallel, which is embarrassingly parallel, to handle computation with a large number of shots in FWI applications.

Single- and multi-GPU implementations and examples

We use CUDA (Compute Unified Device Architecture) from Nvidia in our implementation and focus on speeding up the inner level of SEM46, since the outer level generally satisfies theoretical efficiency.

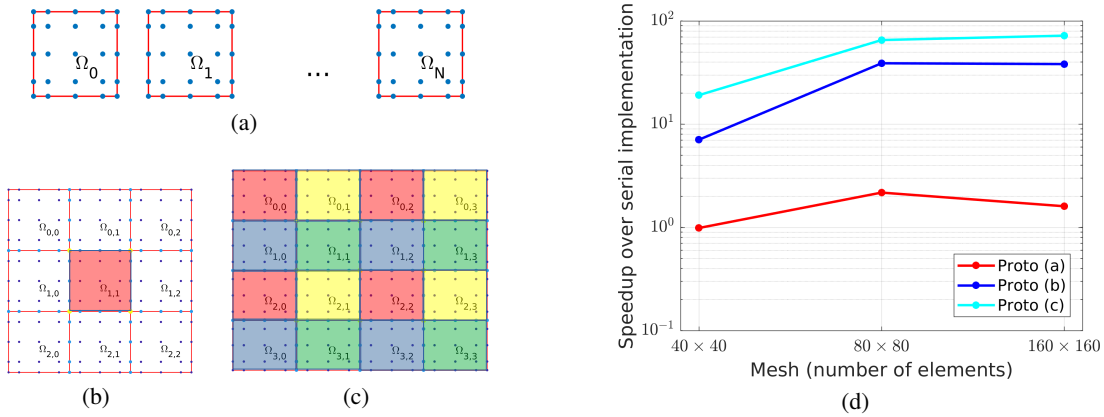

Figure 1: Schematic of three GPU parallel prototypes (a, b and c) in 2D and speedups on one GPU (Quadro RTX 4000) over a serial reference code on CPU (Intel Core i9-9880H @3.2GHz) (d). (a) GLL-quadrature-wise parallelization, (b) element-wise parallelization with neighboring and (c) element-wise parallelization with odd-even mesh coloring. The red lines delineate element borders in those figures. For (b) and (c), the element border points in blue and yellow indicate two and four cumulative sums are required in the SEM assembly, respectively.

By following the SEM algorithm given in the previous section, we can create three separated CUDA kernels in the Newmark time-marching loop: the first kernel is the prediction process to update the current displacement vector **u** using the wavefield variables at the previous time step, the second kernel computes the matrix-vector product of the stiffness matrix **K** and updated **u** vector on the right hand side of Eq. (2), and the last kernel is the correction process to obtain the updated acceleration and velocity vectors. Except for the second kernel, the other two kernels perform point-to-point manipulations which is trivially parallel in the GPU implementation. Therefore, we focus on the design of the second kernel that is also the bottleneck of the SEM modelling (about 70% computational cost). Figures 1a-1c illustrate three alternative GPU parallel prototypes to achieve the matrix-vector product for the second kernel in 2D SEM modelling. Their corresponding CUDA algorithms are illustrated in Table 1. Prototype (a) is a straightforward implementation following the procedure of discrete system building in SEM, namely parallelizing the involved GLL quadrature for each independent element and assembling them globally over the mesh in a sequential manner (Algorithm 1). This prototype is easy to implement with CUDA and can be used in both structured and unstructured meshes (Komatitsch et al., 2010). Prototypes (b) and (c) are both based on the element-wise parallelization (one thread per element) but use different ways to avoid the race problem related to the sum of the values at the element borders. As shown in Figure 1b, the strategy adopted in Prototype (b) is to compute the matrix-vector products of both the element assigned for the thread and the elements which share the same border with it, which involves extra computations (8x in 2D and 26x in 3D) for each thread (Algorithm 2). To circumvent those computations, an odd-even mesh coloring strategy is introduced in Prototype (c), thanks to the property of Cartesian-based structured mesh (the spatial position of each element can be directly associated to the CUDA thread IDs). This prototype contains two key points: 1) coloring different element groups according to their assigned thread IDs in the block being odd or even, which makes the elements in the same group (even/odd coloring) have no dependencies, 2) performing a color-based assembly hierarchically in the shared memory to benefit from fast on-chip memory. The modelling tests on a truncated Marmousi-II model in Figure 2 shows that GPU codes with all these prototypes can produce accurate SEM modelling results with negligible differences with respect to the serial CPU reference code. However, their performance is quite different as shown in Figure 1d. GPU speedups with the element-wise parallelization (Prototypes b and c) are superior to the GLL-quadrature-wise parallelization (Prototype a), since the way of assigning one thread per element allows for a higher count of simultaneous active CUDA cores. The comparison of both element-wise GPU codes indicates a higher speedup (70x) can be obtained from Prototype (c), owing to the odd-even mesh coloring strategy and fast IO on shared memory.

When moving to the 3D implementation, the single-GPU size constraints of both shared memory and global memory need to be taken into consideration. Typically, the size of shared memory is 48 KB, which is not sufficient for assembling a 3D subdomain. Therefore, we flatten the 3D model to many 2D element tiles fitting the shared memory, which is also helpful to use the same Prototype (c) designed in the above 2D modelling. The global memory limitation can be solved by using multiple GPUs with the same domain decomposition as in the MPI-based CPU code. Figure 3b illustrates the speedup and

Algorithm 1 - Prototype a

```
Execution configuration:
dim3 dimBlock(8,8);
dim3 dimGrid(1,1);
Thread assign:
int tx=threadIdx.x;
int ty=threadIdx.y;
if (tx < NGLL && ty < NGLL) then
   for iel = 1 to nel1*nel2 do
      GLL_quadrature(iel,tx,ty,...)
   end for
end if
```

Algorithm 2 - Prototype b

```
Execution configuration:
dim3 dimBlock(8,8);
dim3 dimGrid((nel1+dimBlock.x-1)/dimBlock.x,
             nel2+dimBlock.y-1)/dimBlock.y);
Thread assign:
int iel1=blockIdx.x*blockDim.x+threadIdx.x;
int iel2=blockIdx.y*blockDim.y+threadIdx.y;
if (iel1 < nel1 && iel2 < nel2) then
   for iel2_local = iel2-1 to iel2+1 do
      for iel1_local = iel1-1 to iel1+1 do
         for igll2 = 1 to NGLL do
            for igll1 = 1 to NGLL do
               GLL_quadrature(iel1,iel2,igll1,igll2,...)
            end for
         end for
      end for
   end for
end if
```

Algorithm 3 - Prototype c

```
Execution configuration:
dim3 dimBlock(8,8);
dim3 dimGrid((nel1+dimBlock.x-1)/dimBlock.x,
             nel2+dimBlock.y-1)/dimBlock.y);
Thread assign:
int iel1=blockIdx.x*blockDim.x+threadIdx.x;
int iel2=blockIdx.y*blockDim.y+threadIdx.y;
int tx=threadIdx.x;
int ty=threadIdx.y;
Shared memory initialization:
Copy data from global memory to shared memory
__syncthreads();
if (iel1 < nel1 && iel2 < nel2) then
   if tx%2 == 0 && ty%2 == 0 then
      Loops for GLL quadrature within the element
   end if
   __syncthreads();
   if tx%2 != 0 && ty%2 != 0 then
      Loops for GLL quadrature within the element
   end if
   __syncthreads();
   if tx%2 == 0 && ty%2 != 0 then
      Loops for GLL quadrature within the element
   end if
   __syncthreads();
   if tx%2 != 0 && ty%2 == 0 then
      Loops for GLL quadrature within the element
   end if
   __syncthreads();
end if
```

Table 1: Algorithms of Prototypes a-c. "nel" and "NGLL" are number of elements and GLL points, respectively.

scalability of multi-GPU implementations with two different communication protocols: 1) peer-to-peer (GPU-to-GPU) transfer with CUDA-AWARE MPI and 2) GPU-to-CPU-to-GPU transfer with regular MPI. As expected, a direct GPU-to-GPU communication is more efficient than the one involving memory copied between GPU and CPU, although some ways of increasing the computational scale and using CUDA streams and asynchronous memory copies can be used to overlap those communications with computations (Micikevicius, 2009). The modelling accuracies are validated on a 3D homogeneous model with rough topography (Figure 4). The seismogram comparison shows an excellent agreement between the results of GPU and CPU codes, and the slight residuals come from the differences in manipulating math operations between the GPU and CPU architectures (Weiss and Shragge, 2013).

Conclusions

We present the GPU implementation of a SEM-based modelling engine within the framework of SEISCOPE SEM46 code. To address the computing bottleneck in SEM modelling, namely the product of stiffness matrix and displacement vectors, we develop and compare three parallel prototypes in the CUDA kernel design from aspects of modelling accuracy and efficiency. A significant speedup of 70x has been achieved by the CUDA kernel using an element-wise parallelization with odd-even mesh coloring in the 2D elastic modelling. The single-GPU memory limitation in the 3D modelling is solved by the domain-decomposition based multi-GPU implementation with a direct GPU-to-GPU communication provided by CUDA-AWARE MPI technique to achieve a linear scaling.

Acknowledgements

This study has received fundings from HPC-Europa3 transnational access programme and European Union's Horizon 2020 research and innovation programme under the grant agreement No. 828947 (ENERXICO project) and No. 823844. This study was also partially funded by the SEISCOPE consortium (http://seiscope2.osug.fr), sponsored by AKERBP, CGG, CHEVRON, EXXON-MOBIL, GEOLINKS, JGI, PETROBRAS, SHELL, SINOPEC, SISPROBE and TOTALENERGIES. This study was granted access to the HPC resources provided by the GRICAD infrastructure (https://gricad.univ-grenoble-alpes.fr) and IDRIS/TGCC under the allocation 046091 made by GENCI.

References

Cao, J., Brossier, R., Górszczyk, A., Métivier, L. and Virieux, J. [2022] 3D multi-parameter full-waveform inversion for ocean-bottom seismic data using an efficient fluid-solid coupled spectral-element solver. *Geophysical Journal International*, **229**(1), 671–703.

Komatitsch, D., Göddeke, D., Erlebacher, G. and Michéa, D. [2010] Modeling the propagation of elastic waves using spectral elements on a cluster of 192 GPUs. *Computer Science-Research and Development*, **25**(1), 75–82.

Komatitsch, D. and Tromp, J. [1999] Introduction to the spectral element method for three-dimensional seismic wave propagation. *Geophysical Journal International*, **139**(3), 806–822.

Micikevicius, P. [2009] 3D finite difference computation on GPUs using CUDA. In: *Proceedings of 2nd workshop on general purpose processing on graphics processing units*. 79–84.

Trinh, P.T., Brossier, R., Métivier, L., Tavard, L. and Virieux, J. [2019] Efficient 3D time-domain elastic and viscoelastic Full Waveform Inversion using a spectral-element method on flexible Cartesian-based mesh. *Geophysics*, **84**(1), R75–R97.

Virieux, J. and Operto, S. [2009] An overview of full waveform inversion in exploration geophysics. *Geophysics*, **74**(6), WCC1–WCC26.

Weiss, R.M. and Shragge, J. [2013] Solving 3D anisotropic elastic wave equations on parallel GPU devices. *Geophysics*, **78**(2), F7–F15.

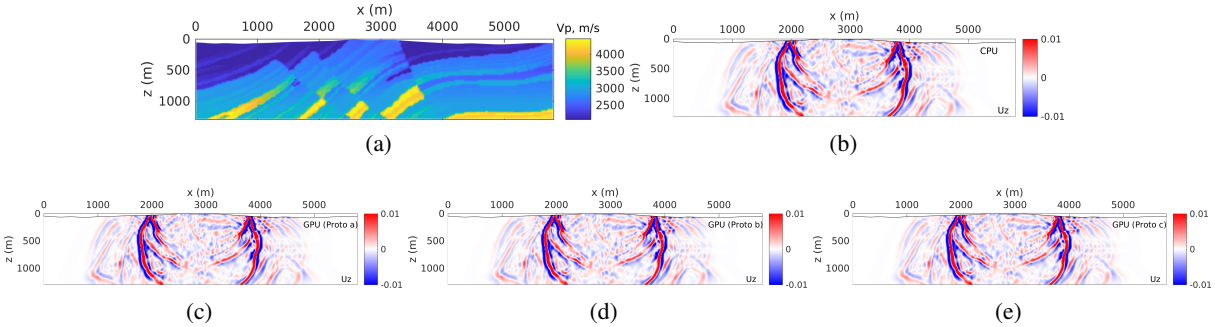

Figure 2: Modelling tests in 2D on a truncated Marmousi-II model. (a) P-wave velocity (V_p) of the test model, and the S-wave velocity (V_s) is defined by $V_s = V_p/\sqrt{3}$. Panels (b), (c), (d) and (e) correspond to wavefield snapshots of vertical displacement component (U_z) at time $t = 0.8$ s computed from a serial CPU reference code and three GPU codes with Prototypes (a), (b) and (c), respectively. All the codes are in single precision.

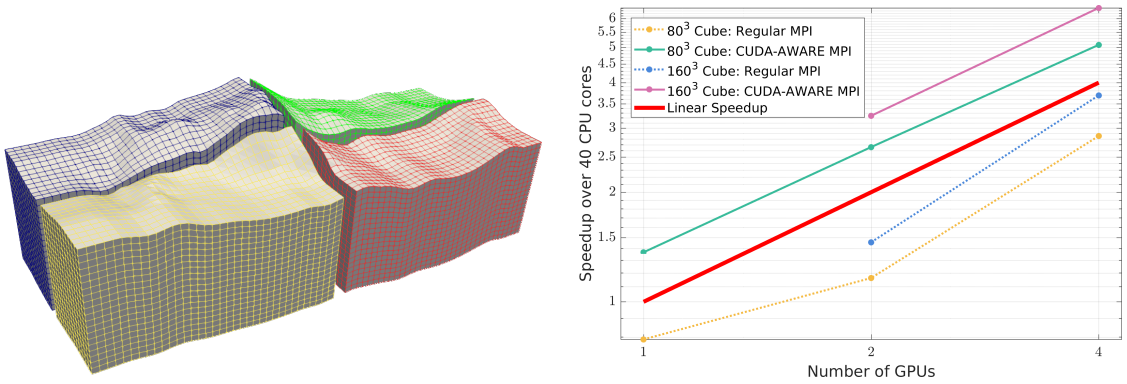

Figure 3: (a) Schematic of the domain decomposition on the Cartesian-based structured mesh in both CPU and GPU implementations. (b) Performance metrics in terms of speedup and scalability in the multi-GPU implementation for 3D problems, where data at the edge of each domain need to be shared with their neighbors using either CUDA-AWARE MPI (GPU-to-GPU transfer) or regular MPI (GPU-to-CPU-to-GPU transfer). Tests are run on the CTE IBM Power9 cluster from Barcelona Supercomputing Center: for each node, 2 × IBM Power9 8335-GTH CPU @2.4GHz (3.0GHz on turbo, 20 cores) and 4 × GPU NVIDIA V100 (Volta) with 16GB HBM2.

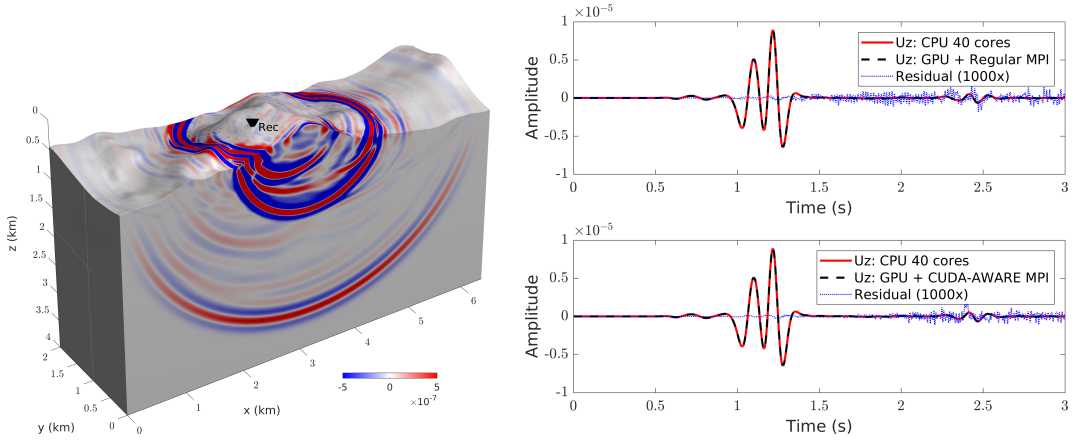

Figure 4: Multi-GPU modelling tests in 3D on a homogeneous elastic model with a rough topography using 4 GPUs. (a) Vertical component snapshot (U_z) at $t = 1.8$ s with CUDA-AWARE MPI, (b) seismogram comparison of two GPU codes and the existing CPU reference code on 40 CPU cores. All the codes are in single precision and run on the CTE IBM Power9 cluster from Barcelona Supercomputing Center: for each node, 2 × IBM Power9 8335-GTH CPU @2.4GHz (3.0GHz on turbo, 20 cores) and 4 × GPU NVIDIA V100 (Volta) with 16GB HBM2.