Report on "DiffTaichi: Differentiable Programming for Physical Simulation" by Hu et al.

DEEPANSHU

DiffTaichi is a new differentiable programming language designed specifically for building high-performance differentiable physical simulators.

Existing differentiable programming tools for deep learning are not suitable for physical simulation due to their lower arithmetic intensity and lack of support for imperative programming and flexible indexing.

We discussed about the need to have a GPU based infrastructure. Existing methods in python were based out of Tensorflow/PyTorch. DiffTaichi takes the best of both worlds and have a language that directly runs on GPU (speeds similar to the CUDA framework) and coded up in python (ease of implementation).

DiffTaichi uses a two-scale automatic differentiation (AD) system. It uses source code transformations to differentiate within kernels to preserve parallelism and arithmetic intensity. It also uses a light-weight tape to record kernel launches for end-to-end differentiation.

One can think of the source code transformation as essentially a function that mapps the space of code in one langugage to the DiffTaichi framework. Similar to CUDA programming, there are decorators for parallel frameworks and restrictions within the loop. Since there is an option of flexible indexing (discussed later), it becomes easier to deal with graphics problems withim the restriction of DiffTaichi on the loop.

To make AD well-defined in an imperative setting where global tensors can be freely modified, DiffTaichi imposes Global Data Access Rules. These rules ensure that global tensor elements are written correctly, avoiding conflicts. In cases where memory consumption becomes an issue due to the history of tensor values, checkpointing can be employed to manage memory efficiently.

The presentation also discussed about methods of implementing the language in a memory efficient way. The compilation uses a Directed Acyclic Graph or a DAG to store the intermediate calculations and fixes a total order on the causality of command execution. This makes the language deterministic.

This is done by generating an adjoint kernel that stores the node data of the DAG and the edges connection determines the flow of code. This also means that there are different checkpointing methods deployed.

Storage control of adjoint tensors is also provided, allowing users to specify storage using the Taichi data structure description language. This feature enables users to manage adjoint tensors effectively.

In practical applications, DiffTaichi has been used to implement and automatically differentiate ten physical simulators, including rigid bodies, deformable objects, and fluids. This demonstrates its suitability for developing complex and high-performance differentiable physical simulators, potentially with neural network controllers.

It uses a "megakernel" approach, allowing the fusion of multiple computation stages into a single kernel. This approach is differentiated using source code transformations and just-in-time compilation. Compared to traditional linear algebra operators in TensorFlow and PyTorch, DiffTaichi kernels offer higher arithmetic intensity, making them more efficient for physical simulations.

Author's address: Deepanshu.

Unlike functional array programming languages commonly used in deep learning, DiffTaichi adopts an imperative approach. It provides support for imperative constructs like parallel loops and control flows (e.g., "if" statements). This approach simplifies tasks such as handling collisions, boundary conditions, and iterative solvers, making it easier to port existing physical simulation code to DiffTaichi.

DiffTaichi allows direct manipulation of array elements via arbitrary indexing. This flexibility enables partial updates of global arrays and natural expression of common simulation patterns. Unlike existing systems that rely on unintuitive scatter/gather operations for such patterns. This flexible indexing is particularly useful during various simulations in Computer Graphics problems.

DiffTaichi was also benchmarked against various simulation cases from rigit body dynamics to water simulation. The paper also addressed the period of collision and fixing gradient at the time of impact.

DiffTaichi can be used to implement a variety of differentiable physical simulators like continuum mechanics simulators, liquid simulators, incompressible fluid simulators, rigid body simulators, etc.

In deep learning, existing tools are optimized for large data blobs, such as convolutional layers in neural networks, with high-level operations like tensor addition and multiplication. These operations are highly efficient due to their high arithmetic intensity, making the most of hardware capabilities. However, these high-level operations are inflexible and cannot be customized easily, leading users to compose their desired high-level operations using low-level operations, resulting in temporary buffers and excessive GPU kernel launches.

DiffTaichi, on the other hand, is tailored for differentiable physical simulations, offering features specifically designed for this domain. The table in the text provides a detailed comparison of DiffTaichi with other existing programming tools. It emphasizes that while tools like PyTorch and TensorFlow have been successful in deep learning, they may not have been designed for differentiable physical simulation, and their operator fusion capabilities may not meet the requirements of simulation tasks.

The core of DiffTaichi's differentiation system revolves around primal and adjoint kernels. Primal kernels are operators that take multiple input tensors and produce output tensors, executing uniform operations on them. In the context of differentiable programming, a loss function is defined on the output tensors, and the gradients of this loss function with respect to each tensor are computed and stored in adjoint tensors, which are denoted as Xijk. The automatic differentiation (AD) system transforms primal kernels into adjoint kernels, allowing the computation of gradients efficiently.

The process of differentiating within kernels using the "make adjoint" pass, which is part of the reverse-mode AD. This pass transforms forward evaluation (primal) kernels into gradient accumulation (adjoint) kernels. It operates on the hierarchical intermediate representation (IR) of Taichi and can handle multiple outer for loops in primal kernels. During this pass, local adjoint variables are allocated for gradient contribution accumulation, and the compiler traverses statements in reverse order to accumulate gradients.

The differentiable simulators built with DiffTaichi are more concise, faster and easier to develop compared to other approaches like TensorFlow or manually written CUDA kernels. The paper also compared the lines of code required to implement various techniques.

DiffTaichi optimised on the differences in workload and requirements between deep learning and differentiable physical simulations. It emphasizes that while existing tools are optimized for deep learning tasks, DiffTaichi is designed to meet the unique challenges of differentiable physical simulations, offering features like megakernels, imperative programming, and flexible indexing. It also provides insight into the core of DiffTaichi's differentiation system, which transforms primal kernels into adjoint kernels, enabling efficient gradient computation.

As discussed multiple times in the COV discussions, the differential physical simulators not always give useful gradients. Issues like discontinuities, singularities and initialization can cause gradients to be misleading. One of the main tasks in approximating the collision forces and impulses is to determine the effect on the body on that short duration of time. Techniques like time of impact and proper initialization can help improve gradient quality.

In summary, we discussed DiffTaichi in light of GPU architecture, graphics use cases and programming langugage implementation. DiffTaichi is an example of an efficient product made using knowledge from various fields of computer science and is possible due to extensive collaborative options available in the research domain.