The Sim Theory SDKs

Simulation Theory's team of engineers have harnessed their combined decades of experience in high-performance computing, software architecture, hardware architecture, multi-threaded applications, and performance optimization to revolutionize the way computers process work and help businesses fully use the computing power they already pay for.

Today, common solutions used to parallelize work across multi-core CPUs use methods developed when CPUs contained two to four physical cores and only one logical core per physical core. As a result, these solutions are most successful at assigning work on up to four cores. A study[1] done at Princeton in 2008 found that on a 32 core system, nearly 50% of the potential performance is lost. As CPU complexity and computing requirements continue to grow, the need to solve the problem becomes ever more pressing.

Simulation Theory has developed and released the Thunder and Lightning software development kits (SDKs) to directly address the fundamental challenge of scaling work on up to 128 threads[2] per scheduler instance. The SDKs are designed to be integrated into source code and complement, not replace, scheduling done by the operating system. Our custom, optimized libraries facilitate more efficient processing of data and scheduling of work so that all available hardware can be used to process work in parallel without needing a deep knowledge of threading and thread synchronization.

How do the Thunder and Lightning SDKs work?

On unoptimized systems, a chunk of work (A,B,C,D) is typically processed in serial from start to finish. Then another chunk of work (A,B,C,D) is started and finished until all of the work is done. As each of the A's, B's, C's, and D's can have dependencies on each other, work on a new chunk cannot be started before the prior chunk is completed, even if only one core is busy.

Using the Sim Theory SDKs, the work that was completed serially can be done in parallel. Using the proprietary Scheduler API, the work to be completed is described in a manner that maintains execution order and data dependencies. With this information, our Scheduler generates packets of execution that will be processed by the Sim Theory Scheduler. All available cores can be used to process thousands of A's simultaneously, then begin on thousands of B's, then C's and D's as soon as cores become available. Computational power is therefore used more efficiently, enabling work to be completed faster on the same hardware. This allows the customer to decide whether to transition to more cost-efficient hardware or add more work to be completed using the reclaimed computational power.

---

[1] Contreras, Gilberto; Martonosi, Margaret (2008). http://www.iiswc.org/iiswc2008/Papers/006.pdf

[2] Multiple scheduler instances can be run per process. As we are able to test on larger and more advanced CPUs, the allowed number of threads per scheduler will increase.

Which product is right for you?

Thunder SDK

Thunder is a software development kit that is designed for enterprise businesses that are using a hosted provider such as AWS or GCE or have on premises compute resources. It is designed for use on device, edge, server, and cloud infrastructure. The Thunder SDK supports Intel and AMD 64-bit CPUs running a Windows or Linux operating system and ARM64 CPUs using a Linux or Android operating system. Additional operating systems and CPU architecture support is on the development roadmap. If you need an OS or CPU architecture that is not yet supported, please contact us.

Lightning SDK

Lightning is a software development kit that is designed for use when developing a retail product such as a video game, media playback solutions, mobile applications, and more. The Lightning SDK supports Intel and AMD 64-bit CPUs running Windows or Linux operating systems and ARM64 CPUs running a Linux or Android operating system. Upcoming releases will support MacOS and iOS. Additional operating systems and CPU architecture support is on the development roadmap. If you need an OS or CPU architecture that is not yet supported, please contact us.

Professional Services

Simulation Theory offers custom licensing to support customers with more diverse needs. If you aren't sure which SDK is the best fit or need a bespoke solution, our team is ready to assist.

What problems can the SDKs solve?

When customers come to Simulation Theory looking to solve a problem or improve an existing solution, we most often find that their processing capability is throttled by at least one of the following five limitations.

CPU Underutilization

Modern CPUs are incredible feats of design and engineering. Sim Theory's passion and the focus of the Thunder and Lightning SDKs is unlocking the unused potential of hardware. Most software only effectively uses one to four CPU cores, while many modern CPUs contain eight or more cores. This means a lot of electricity, heat, and time are wasted due to the inefficient usage of hardware.

The Sim Theory SDKs allow developers to write efficient multi-threaded applications without requiring a deep knowledge of hardware architecture, drivers, or software features. When using more of the available cores in the CPUs we already own, we can reduce compute times, which saves time, money, and impact on the environment. Reducing compute times can also free up CPU time for other work loads such as data forming and compression to improve data transport times.

Irregular Use of the GPU

Given the incredible compute power of modern GPUs, it is rare to be truly bound by compute or computational load on arithmetic logic units on the GPU. When Simulation Theory assists customers troubleshooting GPU concerns, we see that the root cause is most often being bound by the CPU or memory. When using the GPU for processing, the CPU is used to transform data for either the GPU or for recording compute commands for the GPU to process as one batch of work. When the CPU is not being used efficiently to push compute work to the GPU, it can easily be misinterpreted as being GPU bound.

Using the Sim Theory SDKs, developers can easily write code that conditions data, records compute commands in parallel, and uses all of the available cores on the CPU. Additionally, submitting command buffers of work for execution on the GPU can be efficiently pipelined to keep the GPU fully occupied. Integration of the Sim Theory SDKs can save companies the cost of keeping up with the newest and hardest to find GPUs without sacrificing performance.

Memory Access

Memory access is slow when compared to CPU speeds and cloud instances with large amounts of RAM are costly. Traditionally, to reduce compute load, operations and calculations are performed on the CPU or GPU and then the results are cached to memory or disk for look up later. We often find that when the Simulation Theory SDK is integrated and execution is parallelized, it is often faster and cheaper to recompute than it is to read cached results from memory. Recomputing rather than caching to memory allows Sim Theory customers to scale back RAM usage, improve execution speed, and use cheaper cloud instances.

File IO Speed

Even with modern SSD and M.2 drives, file IO is very slow compared to memory access speeds and CPU speeds. There are many strategies to improve File IO throughput, ranging from compression, efficient file formats, data organization, and more. The Simulation Theory SDK enables the mixing of functionally parallel and data parallel execution within the same scheduler as well as a growing number of asynchronous containers and algorithms to ease use and integration.

Complex or Hidden Performance Challenges

Some customers come to the Sim Theory team to find out what they don't know. Their cloud spend is uncontrolled or their application or service is running significantly slower than expected. Finding a quality profiling solution that identifies bottlenecks and narrows down the problem space is critical. Simulation Theory can provide some general recommendations using the support credit provided with an annual subscription to the Thunder SDK, or we can provide a bespoke solution with a custom support contract.

How do developers integrate the SDKs?

The Thunder and Lighting SDKs require integration with application or service source code. They provide a C API and C# bindings. The C API enables the SDKs to be called from multiple languages and used without requiring a direct binding, reducing the time required for integration. The C# bindings are provided to facilitate integration with the larger .NET community and with the Unity Engine. Future language bindings are on the development roadmap and will be implemented based upon the needs of our customers.

Depending on the complexity of the application or service's source code, integration can be done at a top level domain or in a more integrated manner. A typical top level integration takes anywhere from a few hours to a few days. A deeper, more complex integration generally results in greater performance gains, but the integration time is highly variable and dependent on the complexity of the source code and the performance goals.

Top Level Integration

To integrate either of the Sim Theory SDKs at a top level domain, developers must first assess the execution logic of their source code and define dependencies. The end result of this first step must then be used to assess the work that needs to be completed and developers must note which pieces can be done in parallel.

The collection of libraries, headers, and tools that make up the Sim Theory SDKs must be integrated into the existing build system. They complement, but do not replace what already exists. When jobs are kicked off using this new system, the proprietary Scheduler in the SDK builds out a new single-layer execution graph, parallelizing as much work as the execution logic allows.

Top level integrations bring significant performance improvements without requiring a lot of source code modification. In testing, Simulation Theory found that this type of integration can improve CPU utilization by about 40%. Please see our CNeRF case studies for more information.

Complex Integration

For source code that is more complex, or for situations where bringing out every last bit of available performance is critical, a deeper implementation of an SDK is necessary.

In addition to the first pass of work defining execution logic and dependencies detailed above, developers must assess any nested execution pipelines and define what can be further parallelized in the larger work chunks defined in the steps above. This results in a branching execution graph that allows the Scheduler to pass work to as many available cores as possible.

The integration process of the SDK into the build system, as described above, is largely the same. Developers, however, can choose to replace libraries they are using with our custom, optimized ones. Some examples include math, fileIO, virtual file system, and many more.

Using the more complex type of integration, Simulation Theory found that where roughly 0.0034GB of data were being processed per second on an standard implementation of a MACD algorithm, about 4.32GB of data could be processed per second using the same algorithm but optimized with the Sim Theory SDK. Please see our MACD case study for more information.

sales@simtheoryinc.com
simtheoryinc.com