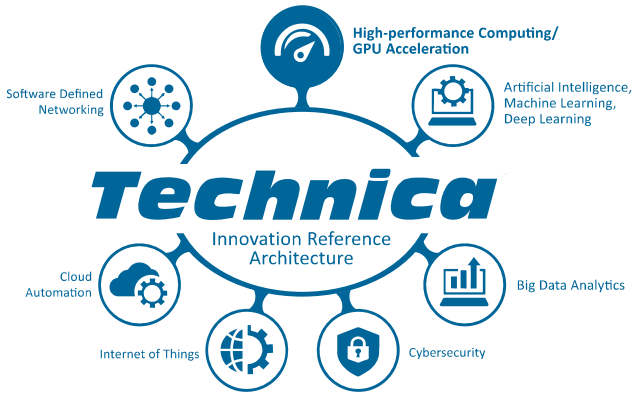
Technica™

General purpose computing on a GPU expands GPU implementation beyond its traditional usage in digital graphic rendering. Big Data is one area that tremendously benefits from this GPU-acceleration. This paper provides a high-level overview of GPU-acceleration and discusses how Technica utilized GPU for its innovative Big Data Products: FUNL and SQuadron.

The Technica Innovation Platform White Paper Series presents advanced topics that will drive competitive advantage for next-generation IT over the next three-to-five years.



GPU ACCELERATED COMPUTING

GPU-accelerated computing utilizes graphics processing units (GPUs) together with the Central Processing Unit (CPU) to accelerate scientific, analytics, engineering, consumer, and enterprise applications. GPU computing has demonstrated much progress with enhancing artificial intelligence performance with deep learning algorithms.

The performance experienced in GPU accelerated applications is often by orders of magnitude—10 to 100 times faster than CPU-only computing. GPU accelerated servers now power energy-efficient datacenters in government labs, universities, enterprises, and small-and-medium businesses around the world. GPUs are accelerating applications in platforms ranging from cars, to mobile phones and tablets, to drones and robots.

This paper presents key aspects of GPU computing and highlights innovations Technica has made in GPU accelerating its Big Data products: FUNL and SQuadron.

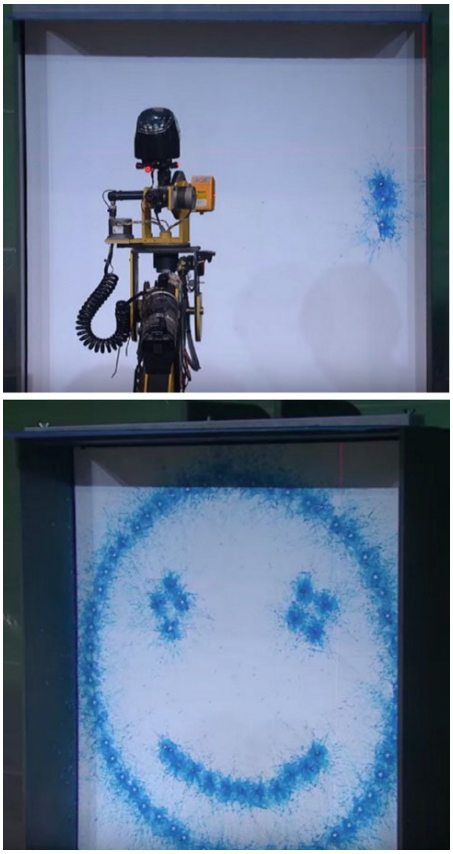


Figure 1 – Sequentially Painted Happy Face

GPU COMPUTING HISTORY

NVIDIA Corporation created the first GPUs in the early 2000s. GPU cards (printed circuit boards) were placed in desktop machines to speed up pixel processing for PC computer games. Instead of sequentially processing pixels with CPUs, GPUs allowed the parallel processing of numerous pixels simultaneously.

Figure 1 presents two progressive photos from a YouTube video filmed at a recent NVIDIA conference[1]. The photos portray a CPU working in sequentially to draw a face. The paint machine starts by outlining the drawing the circle of the face; then each eye; and finally, the smile is rendered. The process takes about a minute.

[1] Mythbusters Demo GPU versus CPU: (https://youtu.be/-P28LKWTzrI)

In contrast, a GPU renders a much more complex image, where each pixel is painted simultaneously. With parallel processing, the entire face of the Mona Lisa is completed in less than a second.

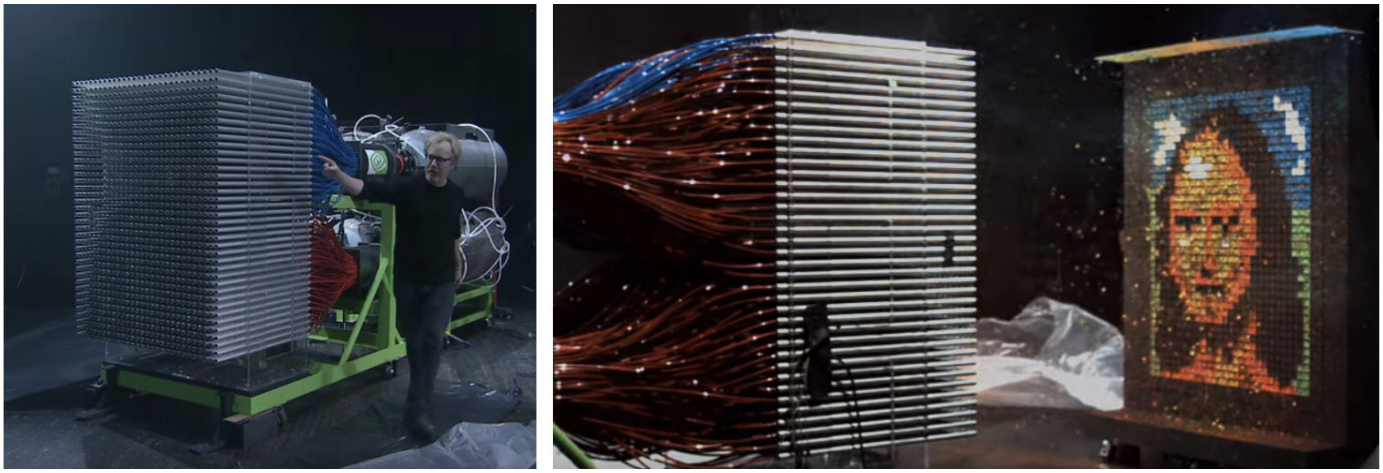


Figure 2 – Parallel Painted Happy Face

In 2007, NVIDIA took steps to move GPUs beyond gaming applications with the first release of CUDA (Compute Unified Device Architecture). CUDA is a parallel computing platform and application programming interface (API) architecture that allows developers to leverage the massively parallel processing capability of the GPU. While CUDA is proprietary to NVIDIA GPUs, the OpenCL (Open Computing Language) API has been developed as an open alternative.

CPU VS. GPU

The introduction of CUDA allowed programmers to take advantage of the power of parallel processing for lower level tasks. With APIs like CUDA and OpenCL, programmers are able to improve performance with a mix of CPU and GPU technology, using the best processor best suited for specific tasks.

Since 2007, both types of processors have grown in processing power. However, the performance of GPUs in general and NVIDIA GPUs specifically have increased at a faster rate as portrayed in **Figure 3**. The blue line represents the teraflop processing power of various NVIDIA GPU products, compared to top class CPUs.

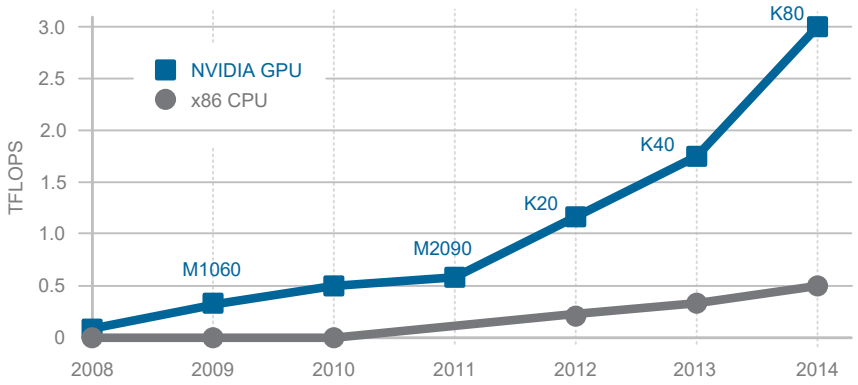


Figure 3 – CPU and GPU Peak Performance in Teraflops

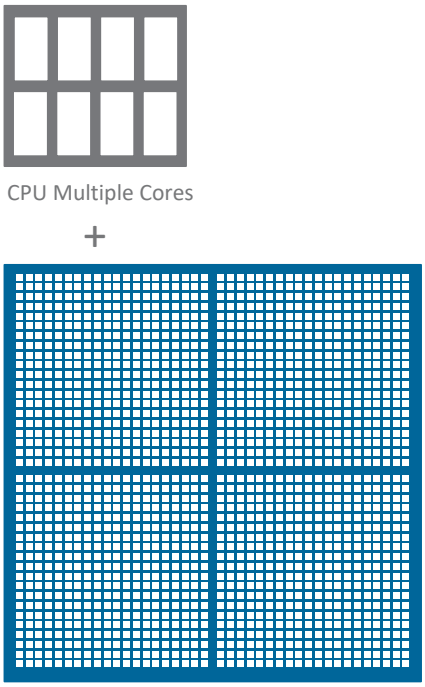


Figure 4 – CPU/GPU Core Comparison

One of the problems with today's modern CPU processors is they have hit a clock rate limit at around 4 GHz. At 4 GHz too much heat is generated for the current technology, and must be dissipated with elaborate and expensive cooling solutions.

Unable to increase the clock rate, manufacturers adopted a different approach to make forever-faster processors. The two main CPU manufacturers, Intel and AMD, have been forced to add more cores to the processors. Currently the fastest flagship processor from Intel, the Intel Core i7-6950X Processor Extreme Edition, has 10 cores. GPU manufacturers, like NVIDIA and AMD, have also followed this pattern. Due to less technical complexity needed in GPUs, the number of cores is in the thousands.

A CPU consists of a few (currently between four and ten) cores optimized for sequential serial processing while a GPU has a massively parallel architecture consisting of thousands of smaller, more efficient cores designed for handling multiple tasks simultaneously. Each core can be envisioned as a stand-alone processor. To leverage the power of multiple cores, programmers much master the art of parallel programming, i.e. splitting up tasks between processors.

PARALLEL PROGRAMMING & GPU ACCELERATION

There is a big challenge with parallel programming. It requires programmers to switch from their traditional serial, single-thread approach, to dealing with multiple threads all executing at once. Even considering only one CPU core, the programmer has to think about two, four, six, or eight program threads and how they interact and communicate with one another. Heterogeneous computing with CPUs and GPUs adds an additional layer of complexity. When GPU accelerating applications, programmers must understand what tasks are best suited to each type of processor as pictured in **Figure 5**.

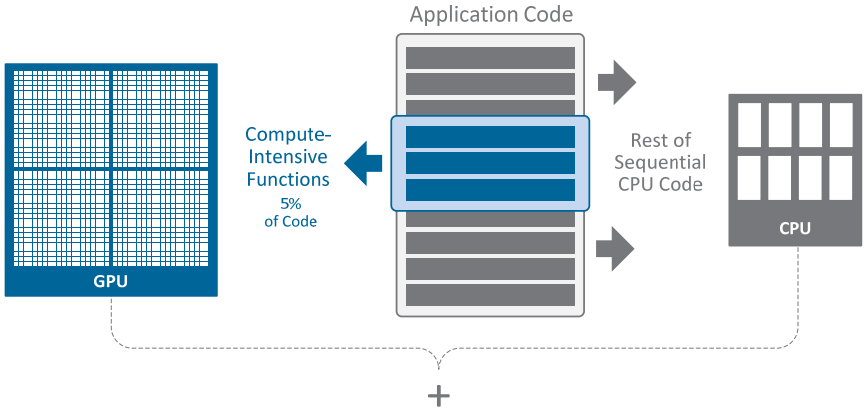


Figure 5 – How GPU Acceleration Works

There are three basic approaches to leveraging GPU acceleration for applications:

- Dropping in GPU-optimized Libraries
- Adding Compiler "Hints" to Auto-parallelize Code
- Using Extensions to Standard Languages like C, Fortran, Python, and Java

NVIDA offers all of these options for CUDA. **Figure 6** presents a standard Python program designed to perform a Monte Carlo simulation—an algorithm that allows risk to be modeled in quantitative analysis and decision making. The CPU-only Python code achieved its result for the simulation in 26 seconds.

```
def step(price, dt, c0, c1, noise):
    return price*np.exp(c0*dt+c1*noise)

def montecarlo(paths, dt, interest, volatility):
    c0 = interest – 0.5 * volatility ** 2
    c1 = volatility * np.sqrt(dt)

    for j in xrange(1, paths.shape[1]):
        prices = paths[:, j – 1]
        noises = np.random.normal(0., 1., prices.size)
        paths[:, j] = step(prices, dt, c0, c1, noises)
```

Figure 6 – CPU-only Python Code

Contrast this with the code in **Figure 7** a combination of Python and the CUDA extension to CUDA via PyCUDA.

```
def step(price, dt, c0, c1, noise):
    price = price.astype(np.float32
    return price*cumath.exp(c0*dt+c1*noise)

def montecarlo(paths, dt, interest, volatility):
    c0 = interest – 0.5 * volatility ** 2
    c1 = volatility * np.sqrt(dt)

    prng = curand( )
    d_noises = gpuarray.empty(paths.shape[0], np.float32)
    d_curLast = gpuarray.to_gpu(paths[:,0].astype(np.float32))
    d_curNext = gpuarray.empty(paths.shape[0], np.float32)

    for j in xrange(1, paths.shape[1]):
        prng.fill_normal(d_noises)
        d_curNext = step(d_curLast, dt, c0, c1, d_noises)
        paths[:,j] = d_curNext.get( )
        d_curNext, d_curLast = d_curLast, d_curNext noises
```

Figure 7 – CPU and GPU Accelerated Python Code

With the additional eight lines of GPU acceleration code, the Monte Carlo simulation was performed in 1.5 seconds—a 17x speedup.

TECHNICA AND GPU COMPUTING

In 2013, Technica decided to focus its Independent Research and Development (IR&D) organization on GPU computing. The company foresaw that Big Data applications were a specific type of problem that benefitted greatly from the massively parallel processing capabilities of GPUs. The company devoted many in-house resources to learning the intricacies of CUDA. Technica then turned its attention to creating two products that leverage the power of GPU computing to perform certain Big Data processing tasks more cost effectively—Big Data on a budget-- and offer orders of magnitude improvement in cost and performance:

- **FUNL**—analytic engine that includes graph analytic, machine learning, and deep learning algorithms. Numerous GPU computing techniques were utilized to make FUNL a unique tool to process Big Data.

- **SQuadron**—GPU database with in-database analytics and integrated time series processing.

Figure 8 dramatically highlights the benefit of GPU computing, where SQuadron performs comparably to a CPU-only solution at 100x lower cost.

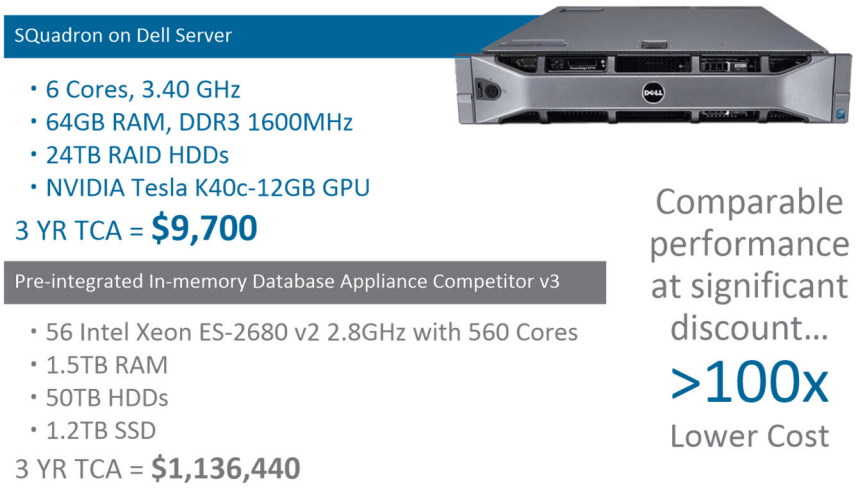


Figure 8 – SQuadron Cost Comparison

SUMMARY

GPU accelerated computing is one of the big drivers of the next generation of innovation, especially in Big Data applications like graph analytics and deep learning. The massively parallel architecture of GPUs allows orders of magnitude performance improvements at a fraction of the cost of CPU-only solutions. Technica has embraced GPU computing by becoming experts in CUDA and creating the innovative Big Data applications: FUNL and SQuadron.

Technica provides professional services, products, and innovative technology solutions to the Federal Government. We specialize in network operations and infrastructure; cyber defense and security; government application integration; systems engineering and training; and product research, deployment planning, and support.

Technica™
22970 Indian Creek Drive, Suite 500
Dulles, VA 20166
703.662.2000