



GPGPU for Embedded Systems Whitepaper

Dan Mor

1.	Introduction	3
2.	Existing Difficulties in Modern Embedded Systems.....	3
2.1.	Not enough of calculation power	3
2.2.	Power Consumption, Form Factor and Heat Dissipation.....	3
2.3.	Generic Approach	4
3.	GPGPU Benefits in Embedded Systems	4
3.1.	GPU Accelerating Computing.....	4
3.2.	CUDA and OpenCL Frameworks - Definitions	5
3.3.	CUDA Architecture and Processing Flow	5
3.4.	CUDA Benchmark on Embedded System.....	5
3.4.1.	“N-body” - CUDA N-Body Simulation.....	6
3.4.2.	SobolQRNG - Sobol Quasirandom Number Generator.....	8
3.5.	Switchable Graphics.....	8
3.5.1.	Why we need switchable graphics in embedded systems?.....	8
3.5.2.	NVIDIA® Optimus™ Technology.....	8
3.5.3.	AMD Enduro™ Technology	9
4.	Use Cases	9
5.	Aitech AI GPGPU Systems	10
5.1.	Aitech AI GPGU Product Line	10
5.2.	Aitech A178 Thunder - GPGPU Fanless AI Supercomputer	10

1. Introduction

Providing High Performance Embedded Computing (HPEC) Systems using General Purpose Computation on Graphic Processor Units (GPGPU).

In today's fast-growing computer industry, embedded systems must be head-to-head with the latest computing technology. The software applications used by embedded systems (consumer or military) are becoming more complex and are demanding a lot of computation power from the hardware.

Therefore, a deep understanding of the difficulties in modern embedded systems and a knowledge of existing technical solutions will help you to choose the best available product that will deliver optimal performance for the civilian and military market.

2. Existing Difficulties in Modern Embedded Systems

Below are the most common difficulties that we – Architects, Product Managers, Engineers – are dealing with in day-to-day situations.

2.1. Not enough of calculation power

While trying to reuse existing software applications, we are constantly adding new features and implementing new requirements. The code becomes more and more complex and the application becomes CPU “hangry”, so eventually, we are dealing with:

- Complex CPU Load Balancing – we are dancing on a “razor blade” in order to satisfy our SW application demands
- CPU choking – we are ending up with such slow OS response that we need to change the entire SW architecture and find the very fine line between acceptable response and getting the job done
- Upgrading and Overclocking – other solutions for adding additional computation power that can be costly (upgrading) or detrimental to component life (overclocking)

Nowadays, more and more applications are using CUDA libraries (GPU accelerated) in order to reduce development time and “squeeze” a maximum performance per watt from the computation engine.

All the reasons above are pushing embedded systems providers to consider using GPU instead of CPU as a main computing entity.

2.2. Power Consumption, Form Factor and Heat Dissipation

As we saw in the previous chapter, once we are in need of more calculation power, we are buying more powerful hardware or overclocking an existing one, leading to increased power consumption.

Looking at the defense industry, we can see that the form factor is playing an especially important role in product success. The footprint is constantly decreasing and market demand for SWaP (Size, Weight and Power) optimized systems is increasing. The embedded industry is not excluded and developers and manufacturers are trying to close the gap between consumer and embedded market by decreasing a footprint of new systems.

Increased power consumption and decreased footprint invite a new difficulty – heat dissipation. Imagine taking a powerful gaming PC and putting it in the same small factor “toaster” enclosure. You will end up with a burned-out system, instead of the high performance calculation job you need.

2.3. Generic Approach

While trying to secure a customer or program that typically have tight deadlines and milestones, embedded systems manufacturers can find themselves frequently compromising a generic approach versus a proprietary one. How many times do we get a system for integration and find out that some module of this system has a proprietary interface? Now, conversely, how smooth do you think the integration process will be, if all modules support a known generic interface? You right, it will work like a charm.

Zero Time Porting – Develop an entire solution on a Notebook or PC-based station with the same GPU architecture and then just copy it to embedded system.

3. GPGPU Benefits in Embedded Systems

3.1. GPU Accelerating Computing

GPU-accelerated computing is the use of a graphics processing unit (GPU) together with a central processing unit (CPU) to accelerate applications.

We have discussed the difficulties in modern embedded systems. Once we are using only a CPU as a main computing engine, eventually it will choke up. What if we could offload a portion of the compute-intensive application to the GPU, while the rest of the application runs on CPU?

This is exactly what GPU Accelerating Computing is doing - offloading some of the compute-intensive portion of application.

So, how is the GPU doing it faster than CPU?

The GPU has evolved into an extremely flexible and powerful processor because of:

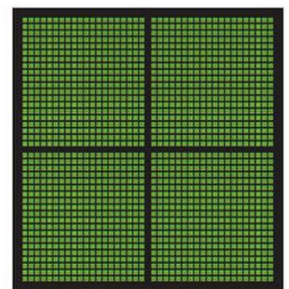
- Programmability
- Precision (Floating Point)
- Performance - thousands of cores to process parallel workloads
- GPUs are getting faster by thanks to the push from the giant gaming industry

NVIDIA® explains it very well:

A simple way to understand the difference between a CPU and GPU is to compare how they process tasks. A CPU consists of a few 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.



CPU
MULTIPLE CORES



GPU
THOUSANDS OF CORES

3.2. CUDA and OpenCL Frameworks - Definitions

CUDA by Wikipedia - Compute Unified Device Architecture (CUDA) is a parallel computing platform and an application programming interface (API) model created by NVIDIA®. It allows software developers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing – an approach known as GPGPU.

OpenCL by Wikipedia - Open Computing Language (OpenCL) is a framework for writing programs that execute across heterogeneous platforms consisting of central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs) and other processors.

3.3. CUDA Architecture and Processing Flow

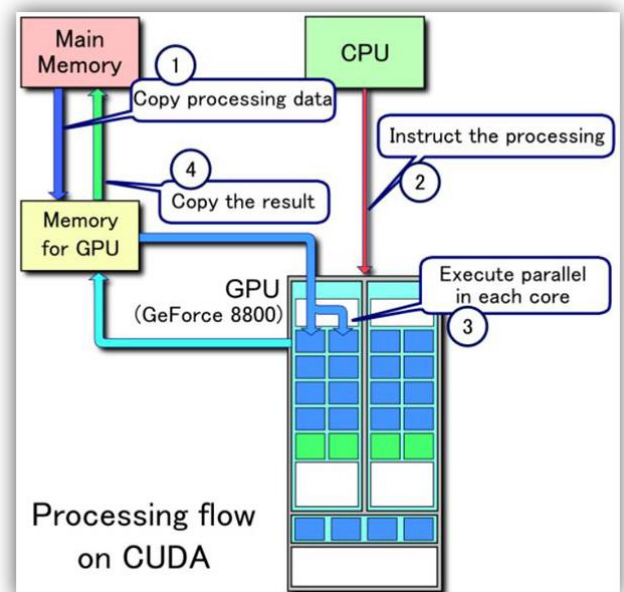
Unlike CPUs, GPUs have a parallel throughput architecture that executes many concurrent threads, rather than executing a single thread.

In the computer industry, GPUs are used not only for graphics rendering but also in game physics calculations (physical effects such as debris, smoke, fire, fluids).

CUDA has also been used to accelerate non-graphical applications in computational applications that use mathematical algorithms.

Example of a CUDA processing flow:

- Copy data from main mem to GPU mem
- CPU instructs the process to GPU
- GPU executes parallel in each core
- Copy the result from GPU mem to main mem



Since we can see that the processing flow involved both the CPU and GPU, we can declare that:

The faster this tandem (CPU and GPU) will be, the better computing performance we will get.

3.4. CUDA Benchmark on Embedded System

In order to compare computing performance between a CPU and GPU, we will use the NVIDIA® CUDA SDK 6.5 to run the same algorithm, once on the CPU and then on the GPU, then compare the results.

We want to do a “fair” comparison, so we picked up one of the latest Intel platforms – Haswell and NVIDIA GTX 770M graphic cards.

Both the boards and the HPEC GPGPU system are taken from [Aitech Systems](#) for the test’s purpose.

Hardware boards involved in this test:

- CPU – Intel Haswell i7 2.4GHz SBC (Manufacturer: Aitech Systems)
- GPU – Nvidia GTX 770M GPGPU board (Manufacturer: Aitech Systems)
- HPEC GPGPU System – A191 (Manufacturer: Aitech Systems)

3.4.1. “n-body” - CUDA N-Body Simulation

This sample demonstrates efficient all-pairs simulation of a gravitational n-body simulation in CUDA.

CUDA API:

`cudaGLSetGLDevice`, `cudaGraphicsMapResources`,
`cudaGraphicsUnmapResources`,
`cudaGraphicsResourceGetMappedPointer`,
`cudaGraphicsRegisterResource`, `cudaGraphicsGLRegisterBuffer`,
`cudaGraphicsUnregisterResource`

Key Concept:

Graphics Interop, Data Parallel Algorithms, Physically-based Simulation

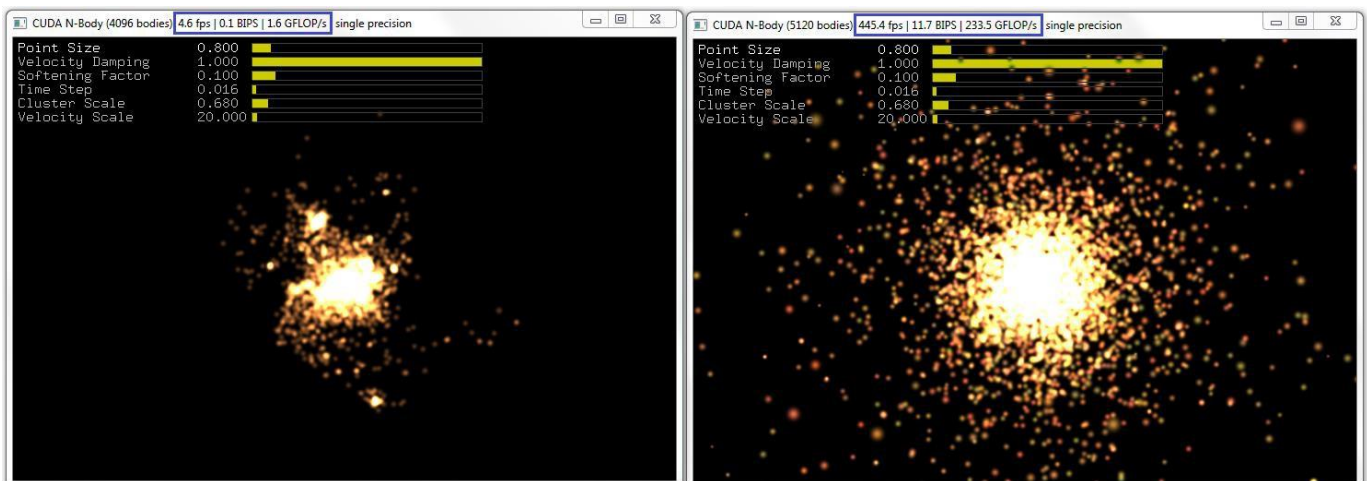


Figure 1: Running “n-body” on CPU

Figure 2: Running “n-body” on GPU

	CPU	GPU	FACTOR
FPS	4.6	445.4	X 100
GFLOP/s	1.6	233.5	X 145

Note:

FPS – Frame per Second

GFLOP/s – Giga Floating Point Operations per second

Imagine yourself offloading heavy duty calculations to the GPU and freeing the CPU for other mission critical tasks.

Now, we will increase the numbers of simulated n-bodies and compare the results between the CPU and GPU.

	CPU	GPU	FACTOR
n-body number	4096	4096	
Time for 10 iterations [msec]	2080.907	6.852	X 300
Interactions per second [billion ips]	0.081	24.486	
Single-precision GFLOP/s at 20 flops per interaction	1.612	489.728	
n-body number	8192	8192	
Time for 10 iterations [msec]	8318.286	27.261	X 300
Interactions per second [billion ips]	0.081	24.617	
Single-precision GFLOP/s at 20 flops per interaction	1.614	492.342	
n-body number	16384	16384	
Time for 10 iterations [msec]	33301.543	92.930	X 350
Interactions per second [billion ips]	0.081	28.886	
Single-precision GFLOP/s at 20 flops per interaction	1.612	577.716	

The graph below demonstrates the iteration latency of simulated, defined numbers of n-Bodies between CPU and GPU.

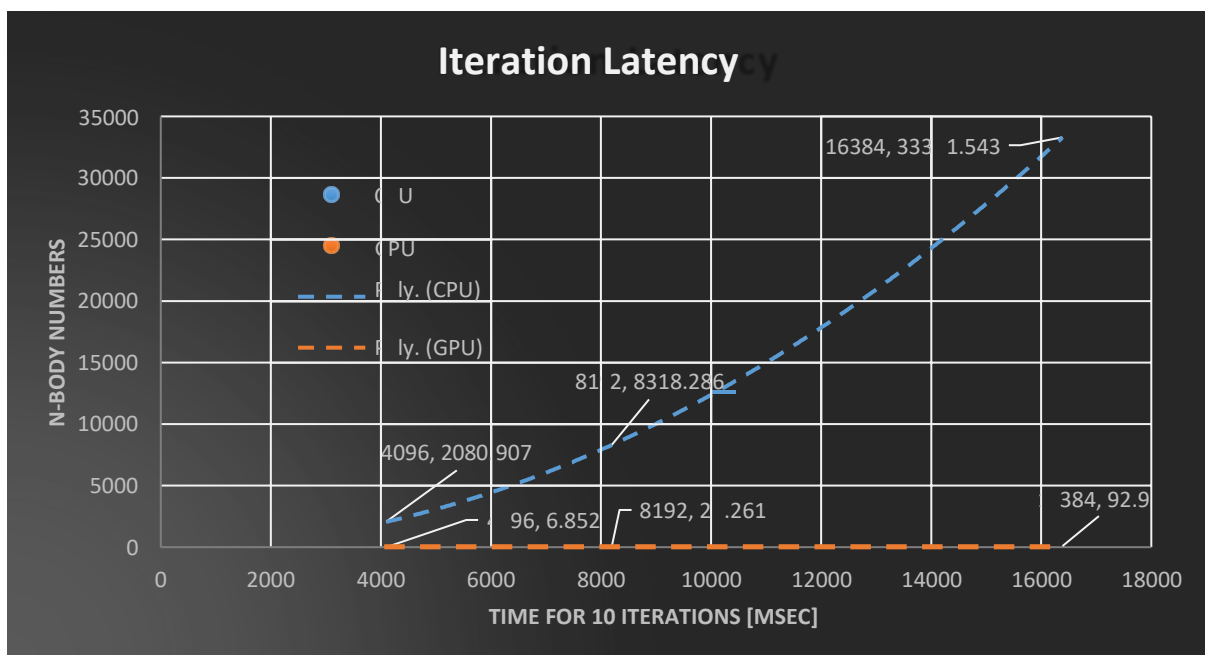


Figure 3: Iteration Latency between CPU and GPU

It takes the GPU 93 msec and the CPU 16 sec to perform a simulation on 16384 numbers.

3.4.2. SobolQRNG - Sobol Quasirandom Number Generator

This sample implements a Sobol Quasirandom Sequence Generator. Comparing this algorithm between the CPU and GPU, we see how many samples may be involved per second during the calculation process.

Number of vectors = 100000

Number of dimensions = 100

	CPU	GPU	FACTOR
Gsamples/s	0.230959	9.13413	X 40

3.5. Switchable Graphics

3.5.1. Why we need switchable graphics in embedded systems?

By using switchable graphics in embedded systems, we can actually enjoy both worlds of power consumption and performance. When needed, we can offload heavy duty tasks to the powerful GPU. On the other hand, when not needed, we can benefit from the internal IGP (integrated graphics processor) and save a lot of power, improving system power consumption and heat dissipation. So, what we have is:

- Full performance benefits of a discrete GPU
- Low power consumption of an integrated graphics solution
- Automatically optimized embedded systems that offer the best performance or power consumption

3.5.2. NVIDIA® Optimus™ Technology

NVIDIA® Optimus™ technology intelligently optimizes embedded systems, providing the outstanding graphics performance you need, when you need it, all the while optimizing power consumption.

NVIDIA®'s explanation of Optimus™ technology:

Using NVIDIA's Optimus technology, when the discrete GPU is handling all the rendering duties, the final image output to the display is still handled by the Intel integrated graphics processor (IGP). In effect, the IGP is only being used as a simple display controller, resulting in a seamless, flicker-free experience with no need to reboot.



When less critical or less demanding applications are run, the discrete GPU is powered off and the Intel IGP handles both rendering and display calls to conserve power and provide the highest possible battery life.

Read more here: <https://developer.nvidia.com/optimus>

3.5.3. AMD Enduro™ Technology

AMD Enduro technology optimizes your embedded system to give you an instant boost in graphics performance when you need it, consuming virtually zero watts of power when you don't.

This is how AMD explains the key benefits:

Seamlessly switch between your APU or integrated graphics and your AMD FirePro™ or AMD Radeon™ graphics, based on graphics workload, to allow you to get longer battery life and outstanding performance.



Intelligent implementation in AMD Catalyst™ drivers allows the system to find the best graphics option for your needs or enables you to configure it yourself for the best performance possible.

Read more here: <https://www.amd.com/en/technologies/enduro>

4. Use Cases

There are so many use cases for GPGPU technology. Actually, any application that involves mathematical calculation can be a very good candidate for this technology usage.

- Image Processing – enemy detection, vehicle detection, missile guidance, obstacle detection, object detection, classification, segmentation, etc.
- Radar
- Sonar
- Video encoding and decoding (NTSC/PAL to H.264)
- Data encryption/decryption
- Database queries
- Motion Detection
- Video Stabilization

We can benefit from GPGPU by developing our own application using CUDA and OpenCL high-level languages or we can run industrial GPU accelerated applications that are already optimized.

Hundreds of industry-leading applications are already GPU-accelerated. Find out if the applications you use are GPU-accelerated.

See more at: <https://www.nvidia.com/en-us/gpu-accelerated-applications/>






5. Aitech AI GPGPU Systems


After seeing all these benefits from AI GPGPU, are you wondering if there is an embedded system that leverages all these wonderful technologies?

There is such a system! Actually, an entire product line!

5.1. Aitech AI GPGPU Product Line

Below you can find a quick summary of Aitech's AI GPGPU product line, from HPEC systems based on Intel Xeon SBCs and NVIDIA powerful GPUs, like RTX3000, to Small Form Factor (SFF) systems based on the NVIDIA Jetson family.

Small FF	HPEC
 <p>A178</p> <ul style="list-style-type: none"> ▪ Small FF ▪ NVIDIA AGX Xavier ▪ 10GbE, USB3, DP ▪ Variety of I/O 	<ul style="list-style-type: none"> ▪ 2 x 3U VPX ▪ Intel CPU ▪ NVIDIA GPU ▪ Performance  <p>A191</p>
 <p>A176</p> <ul style="list-style-type: none"> ▪ Small FF ▪ NVIDIA TX2/TX2i ▪ HD-SDI & Composite IN ▪ Variety of I/O 	<ul style="list-style-type: none"> ▪ 4 x 3U VPX ▪ Intel CPU ▪ NVIDIA GPU ▪ Performance  <p>A196</p>
	<ul style="list-style-type: none"> ▪ 4 x 3U VPX ▪ Intel CPU ▪ NVIDIA GPU ▪ Performance+  <p>A195</p>










5.2. Aitech A178 Thunder - GPGPU Fanless AI Supercomputer

The A178 Thunder is the smallest and most powerful rugged GPGPU AI supercomputer, ideally suited for distributed systems, available with the powerful NVIDIA Jetson AGX Xavier System-on-Module.

Its Volta GPU with 512 CUDA cores and 64 Tensor cores reaches 32 TOPS INT8 and 11 TFLOPS FP16 at a remarkable level of energy efficiency, providing all the power needed for AI-based local processing right where you need it, next to your sensors. Two dedicated NVDLA (NVIDIA Deep-Learning Accelerator) engines provide an interface for deep learning applications.

With its compact size, the A178 Thunder is the most advanced solution for AI, deep learning, and video and signal processing for the next generation of autonomous vehicles, surveillance and targeting systems, EW systems, and many other applications.

Benefits	Aitech A178 System
AI & GPGPU	
Plenty of Calculation Power	
CUDA	
Power Consumption	
Form Factor and Heat Dissipation	
Generic Approach	
Zero Time Porting	

See more:

Aitech A178 Thunder: <https://aitechsystems.com/product/a178-thunder-gpgpu/>

Aitech AI GPGPU product line: <https://aitechsystems.com/mil-aero/gpgpu-mil-aero/>

Aitech video assets: <https://aitechsystems.com/resource-library/>