Product Review : Jetson Nano

Bill brouwer

03/2021

Introduction

In 2014 at or around Nvidia GTC, the first Jetson was released, the TK1 : https://developer.nvidia.com/blog/jetson-tk1-mobile-embedded-supercomputer-cuda-everywhere/

It was a powerful entry into the embedded computing arena, one that was quickly superseded by the TX devices and then Xavier, the latter equipped with a Volta GPU. All Jetson devices have a low power Tegra (System on Chip or SoC) processor, which contains ARM CPU cores and the Nvidia GPU. Figure 1 gives a side by side physical comparison of the TK1 and Nano developer boards.

Figure 1 : Nvidia Jetson TK1 (left) and Nano (right). Camera dangling off the table edge not included 😊

The initial impression you'll receive with the Nano is that it has a smaller form factor than its cousins (the TK1 and possibly TX1 have been discontinued). As you might expect from the space savings and can see from table 1 shortly, there are less GPU cores, hardware accelerated video options and peripherals. There are however many more USB ports than the original TK1, and the Nano is a 64 versus 32 bit architecture. Setup of the device simply involves burning an Ubuntu 18 image to a microSD card and then booting with card inserted under the SoC. Key software like the Nvidia SDK and OpenCV come preinstalled, as components of 'Jetpack'. In the intervening years since the TK1 was released, we've seen the rapid growth of deep learning; cuDNN is also included. All that to say, it's become easier and more economical to work with the Jetson over time.

Hardware Details

As alluded to, the Nano is essentially a stripped-down version of the TX1, touted as targeting robotics development for example, versus more sophisticated applications such as autonomous vehicles. The following table is taken from : https://www.fastcompression.com/blog/jetson-benchmark-comparison.htm

Table 1: Jetson product family details

Hardware feature	Jetson Nano	Jetson TX1	Jetson TX2/TX2i	Jetson AGX Xavier
CPU (ARM)	4-core ARM A57 @ 1.43 GHz	4-core ARM Cortex A57 @ 1.73 GHz	4-core ARM Cortex-A57 @ 2 GHz, 2-core Denver2 @ 2 GHz	8-core ARM Carmel v.8.2 @ 2.26 GHz
GPU	128-core Maxwell @ 921 MHz	256-core Maxwell @ 998 MHz	256-core Pascal @ 1.3 GHz	512-core Volta @ 1.37 GHz
Memory	4 GB LPDDR4, 25.6 GB/s	4 GB LPDDR4, 25.6 GB/s	8 GB 128-bit LPDDR4, 58.3 GB/s	16 GB 256-bit LPDDR4, 137 GB/s
Storage	MicroSD	16 GB eMMC 5.1	32 GB eMMC 5.1	32 GB eMMC 5.1
Tensor cores	--	--	--	64
Video encoding	(1x) 4Kp30, (2x) 1080p60, (4x) 1080p30	(1x) 4Kp30, (2x) 1080p60, (4x) 1080p30	(1x) 4Kp60, (3x) 4Kp30, (4x) 1080p60, (8x) 1080p30	(4x) 4Kp60, (8x) 4Kp30, (32x) 1080p30
Video decoding	(1x) 4Kp60, (2x) 4Kp30, (4x) 1080p60, (8x) 1080p30	(1x) 4Kp60, (2x) 4Kp30, (4x) 1080p60, (8x) 1080p30	(2x) 4Kp60, (4x) 4Kp30, (7x) 1080p60	(2x) 8Kp30, (6x) 4Kp60, (12x) 4Kp30
USB	(4x) USB 3.0 + Micro-USB 2.0	(1x) USB 3.0 + (1x) USB 2.0	(1x) USB 3.0 + (1x) USB 2.0	(3x) USB 3.1 + (4x) USB 2.0
PCI-Express lanes	4 lanes PCIe Gen 2	5 lanes PCIe Gen 2	5 lanes PCIe Gen 2	16 lanes PCIe Gen 4

Power	5W / 10W	10W	7.5W / 15W	10W / 15W / 30W

There is a second Nano with memory of 2Gb, sold for a cost of roughly 60 USD versus 120 USD for the 4Gb model, both extremely attractive price points. Here I've restricted attention to the 4Gb model, revision BO1, which also has two camera ports versus one. The microSD card will need to be purchased separately (I recommend a card with 64Gb memory minimum, particularly for deep learning applications), much more setup detail here : https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit

The Jetson Nano has an older and slower GPU architecture than TX2 or Xavier, and some of the power savings come from the lower GPU core count over the TX1. I also found when using the board that to achieve 5W power consumption, it appears two of the four CPU cores are also disabled. With any Jetson you can run **tegrastats** to see relevant hardware statistics eg., output when running in max power (~10W) mode:

```
RAM 1442/3964MB (lfb 250x4MB) SWAP 0/1982MB (cached 0MB) CPU
[12%@102,3%@102,0%@102,1%@102] EMC_FREQ 0% GR3D_FREQ 0% PLL@32C CPU@37C
PMIC@100C GPU@34C AO@40C thermal@35.5C
```

and 5W:

```
RAM 1438/3964MB (lfb 246x4MB) SWAP 0/1982MB (cached 0MB) CPU
[10%@102,4%@102,off,off] EMC_FREQ 0% GR3D_FREQ 0% PLL@32.5C CPU@37C PMIC@100C
GPU@34.5C AO@40C thermal@35.75C
```

As we'll see shortly, this feature has limited impact on a workflow that can take good advantage of GPU and hardware acceleration. If you do expect to run at full power, I can attest to the fact that you will need to use a proper >= 2.5A supply, for the micro-USB input cable.

One very nice addition to the Nano over earlier releases like TK1 is a single or double RPi camera ports, versus the more sophisticated MIPI-CSI connection of TX1/TX2. The latter allows for up to six cameras and requires more effort and know-how, whereas the inclusion of RPi ports certainly appeals more to the hobbyist and makes for easier development. The ports and a single ribbon connected camera are visible in the first image, available for around 20USD. Furthermore, the Jetson nano ships with drivers for IMX 219 devices like the waveshare pictured above, and creating a video stream can be as simple as :

```
DISPLAY=:0.0 gst-launch-1.0 nvarguscamerasrc ! 'video/x-raw(memory:NVMM),
width=1920, height=1080, format=(string)NV12, framerate=(fraction)20/1' !
nvoverlaysink -e
```

With regards to processing hardware, as mentioned, the Nano has a 128 core Maxwell device, which replaced Kepler in or around 2014. Maxwell provides a larger L2 cache (2M) than Kepler and overall smaller power consumption. Single precision FLOPs were increased per core versus Kepler, though

double precision FLOPs were less than Kepler. Maxwell does boast better video and audio codec performance over Kepler, and we'll look at a specific result shortly.

The SoC Tegra chip also contains four Cortex-A57 cores, maintaining the RISC architecture and low power consumption that ARM is now famous for. The A57 is specifically based on AArch64, with the Advanced SIMD (Neon) extension. Neon supports up to 64-bit integer and single-precision (32-bit) floating-point data and operations, with 128bit wide vector hardware and associated instructions. Buried deep down in the wiki article for Advanced SIMD is a comment to the effect that GCC doesn't consider Neon safe on AArch64 for ARMv8.

Before we get to benchmarks, a couple more remarks on peripherals. As the table suggests, there are indeed 4 PCIe lanes available, albeit accessible via an M.2 expansion slot below the Tegra processor. This implies that the processor needs to be removed for access from the SODIMM, and obviously space is limited. Though wifi is missing, the Nano comes with an ethernet port, which unfortunately is potentially buggy eg., see https://forums.developer.nvidia.com/t/eth0-is-missing/76077

Finally, there is a generous expansion header and multiple power supply alternatives to the micro USB; for much more information please consult the Jetson Nano user guide.

Benchmarks

For the following results, I compared key tasks run on the Jetson Nano versus my Lenovo Ideapad Flex, equipped with an Intel i7-8565U CPU : https://ark.intel.com/content/www/us/en/ark/products/149091/intel-core-i7-8565u-processor-8m-cache-up-to-4-60-ghz.html

Important points to note include the configurable Thermal Design Power (TDP). At the lower end, the chip has a clock of 800Mhz, most commonly runs at 1.8GHz and (while I haven't tried), can apparently run at up to 4.6 GHz. The Ideapad Flex also comes equipped with a CUDA-ready GeForce MX230 Nvidia GPU : https://www.nvidia.com/en-us/geforce/gaming-laptops/geforce-mx230/features/ This is smaller and less powerful than the Quadro P5200 in my Dell workstation laptop, but very similar to the GPU available in the TX2 and still packs a punch.

Video Decoding

During this comparison, I simply timed decoding a short h264 / 1080p format video with 650 frames to (420p) YUV output. On the Ideapad, I compiled and used an example application based on the openh264 library, and ran as :

```
> bin/dh264 data/welsdec.264
```

On the jetson, I compiled and used a sample application that uses hardware acceleration, and ran as :

> jetson_multimedia_api/samples/00_video_decode/video_decode H264 -f 2 --stats --max-perf --disable-rendering --report-metadata --dbg-level 3 data/welsdec.264

Table 2: Video decoding performance test results

Method	Decode Time (s)	Average FPS
openh624 (Intel i7-8565U)	4.58	141.9
Jetson Nano	2.66	243.69

This pleasing outcome (roughly 1.7x) was independent of whether the Nano was operating at 5W or 10W, and here the Ideapad was powered.

Key Algorithms

For the main suite of tests, I assembled 21 routines into a single driver, pulling from some of the most important numerical and computer vision projects of the last decade, including OpenCV, Armadillo C++, Eigen C++, cublas, cufft and cusolve. I also tested disk read / write, and ran the tests at maximum and minimum power for both machines, for a total of 42 benchmarks each, run as:

> bin/bench data/frac.mtx 1024 data/wells1.png

Both host and device code were compiled with O3, and nvcc calls GNU 9.3 (Ideapad / Ubuntu 20) or GNU 7.5 (Nano / Ubuntu 18) for host code compilation. Full output is available in the Appendix, here I'll just focus on a few highlights.

First the failures; double prevision SVD, two dimensional FFT and the Radon transform appear to fail or perform poorly relative to CPU on the Nano versus the Pascal card in the Ideapad. For example, on the Ideapad, cufft is roughly 6-7x faster than the equivalent routine, while on the Nano, GPU and CPU routines take roughly the same time.

With regards to the Radon transform, there is a clue in the PTXAS output from the nvcc compiler :

```
ptxas info    : Compiling entry function '_Z14radon_gpu_finePfS_S_fff' for
'sm_53'
ptxas info    : Function properties for _Z14radon_gpu_finePfS_S_fff
    48 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
ptxas info    : Used 41 registers, 4104 bytes smem, 356 bytes cmem[0], 104
bytes cmem[2]
```

Maxwell devices like the Nano have 32k registers available per thread block, and in this implementation of the Radon transform optimized for another architecture, each block runs 1k threads. Therefore, the runtime requirement of 41 registers per thread exceeds those available on the Nano (32) and the kernel needs to be refactored and reoptimized; something to keep in mind for your custom kernels as well. This may also be the issue with the FFT2 performance, where (slower) global memory might be used in lieu of much faster thread registers. The Jacobi SVD routine from cusolve failed to work when the Jetson operates at 5W.

On the topic of the SVD, using a random 1kx1k matrix and the Eigen C++ routine compiled for Intel versus ARM appears 4x faster, however, cublas versions of the same algorithm are 10x faster than their Eigen (host/CPU) versions. Eigen routines frankly seemed suspiciously slow, and should be checked against Intel MKL SVDj (for example). I did notice that using g++ directly on host code versus trusting that nvcc calls g++ with all the flags specified produced faster GEMMs.

Disk read/write doesn't appear dramatically worse using a microSD versus the SSD of the Ideapad. I didn't test random read/writes, which obviously are best avoided in a performance critical context anyway. Finally, single precision GEMM is fast on either GPU versus the CPU result, as well as Gaussian Mixture Modeling (GMM) from Armadillo and OpenCV tasks like circle detection (based on Hough) and image interpolation. The change in power operating modes had little impact on GPU executed routines; table 3 summarizes several results.

Table 3: Execution time in milliseconds for several algorithms run on an Ideapad CPU/GPU and Jetson Nano CPU/GPU. Dense matrices are 1kx1k, the test sparse matrix was 4795x4795 with 90081 non-zeros. GMM was applied to random 5D data modeled with 2 Gaussians, and the OpenCV routines were tested on a 595x768 image.

Method	Ideapad – power	Ideapad - battery	Nano - 10W	Nano - 5W
dp SVDj (Eigen)	106859	102346	398237	470872
dp SVDj (cusolve)	10252	10250	40744	N/A
SGEMM (Eigen)	265	265	1396	2112
SGEMM (cublas)	3	3	56	59
GMM (arma)	6	6	42	66
sp sparseLU fact. (Eigen)	34	34	137	214
OpenCV circ. Detect	5	4	14	22
OpenCV interp.	4	14	23	74

Stream Processing

For this test I put together a simple stream processing application, that accepts a (decoded) video stream, analyzes frames and outputs numerical information, in this case the tilt or pitch angle of a solar panel. The driver has a producer / consumer pattern, where the main thread pushes frames on a queue for processing by worker threads. Given the failure of the Radon transform kernels on the Jetson Nano and relative success of SGEMM, I used a basic SVD based approach for image feature extraction instead; three columns/rows from a single U/V pair are applied to each image for dimension reduction. The following figure shows the three (quasi-)singular values extracted per frame from a stream where all we monitor is one angle. Since we're only interested in quantifying a single degree of freedom, one singular value should be sufficient, more is probably overkill but can help quantify other degrees of freedom like yaw and roll. Image features extracted, I use the unsupervised technique of kd-tree to find the nearest neighbor feature index that in a real scenario maps to a particular angle.

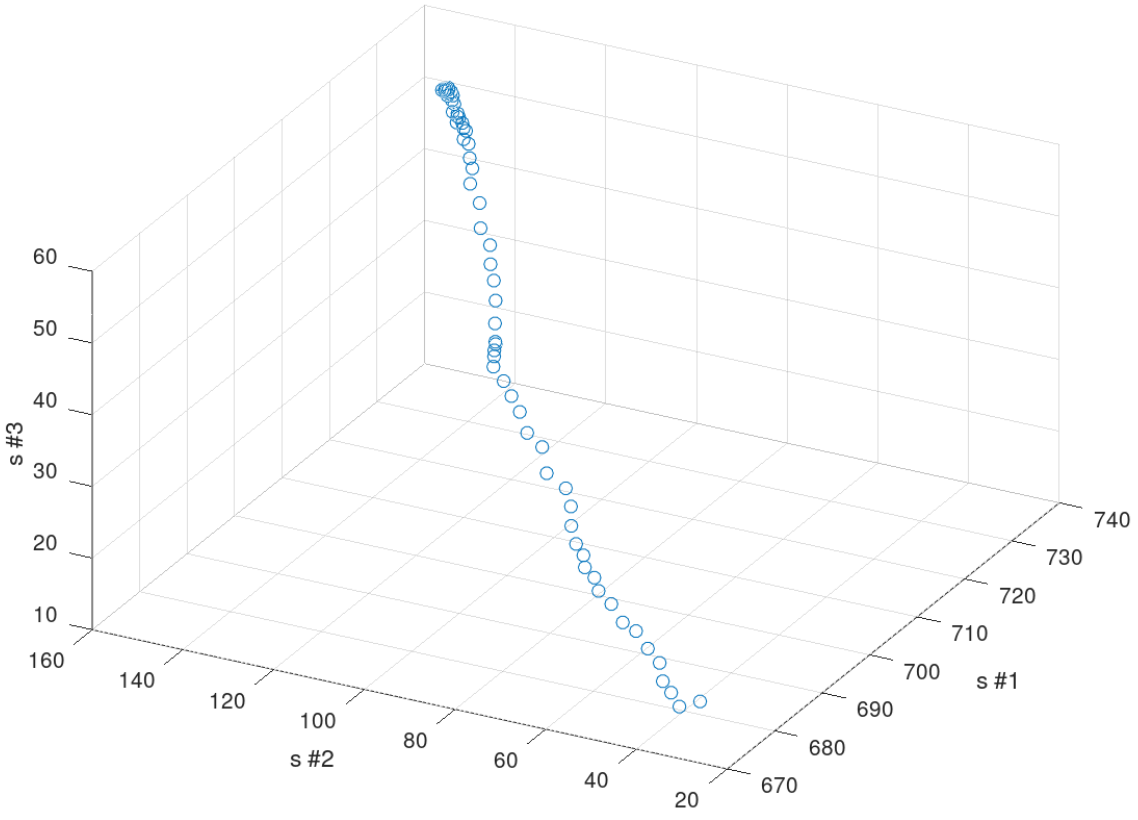

Figure 2 : First three (quasi-)singular values for image frames that display a pitch angle in the range of interest.

In order to provide the same stream input for consistency between tests, I simply piped a decoded file of data/solar_tilt.h264 to the application, loading required ml files specified at the command line:

```
> cat decoded.yuv | bin/tilt --u data/U.mat --v data/V.mat --m
data/model.mat --dt 1
```

On both devices, streaming from disk is faster than realtime (30 fps) and certainly the processing is the rate limiting step, results in the following table. Despite the fact that the code was naïve and unoptimized (eg., no overlapping computation / communication with GPU), the Nano can process a high definition video stream (1920x1080 pixel frames , 30fps) in near realtime at low power.

Table 4: Streaming application results; Ideapad allows for better than realtime processing, as does the Jetson running at 10W. In all cases, two worker threads were used.

Feature	Ideapad - power	Nano - 10W (GPU)	Nano - 5W (GPU)	Nano - 5W (CPU)
FPS	68.42	40.15	24.88	17.33
pixels / s	1.40E+08	8.33E+07	5.15E+07	3.59E+07

Summary

As I hope you can appreciate from this exhaustive article, overall the Jetson Nano is very easy to develop with, and for certain tasks, provided superior results. For example, hardware accelerated codec operations handily outperform similar operations using the Cisco openh264 library code compiled on Intel. The GPU has half the cores and registers, less memory than it's later (Pascal) relative (eg., the just released TX2 NX), which certainly impacts intensive routines like Radon. However, the reality is that routines like these where rigor is needed in order to avoid artifacts (eg., computed tomography / identifying cancerous tissue) aren't really the domain of small devices like the Jetson Nano, and standard computer vision algorithms eg., Hough and single precision GEMM perform well enough. It's also unlikely that you would want or need to perform real-time processing on a high-def feed, though as I lazily tried to demonstrate, you can come close even at low power. The Nano has a couple of deficiencies including a buggy ethernet and hard to access PCIe expansion. The reported memory bandwidth looks low too, perhaps another bug. However, if you can do without the extra camera and 2Gb or memory, the cheaper 60 USD priced Nano is probably the perfect fit for simple edge and robotics applications.

Happy parallel computing! Bill b

Appendix – Benchmark output results

Ideapad (powered)

```
Inputs:
 sparse matrix:         data/frac.mtx
 dense mat side:        1024
 FFT2 size:             2048
 test image:            data/wells1.png

GPU device(s):
 clock rate:            1531000
 Major/minor:           6/1
 SMP count:             2
 Global mem:            2099904512
 Sh mem / block:        49152
 Name:                  GeForce MX230
 Max threads / block:   1024
 registers / block:     65536
 Peak mem Gb/s:         56.064
 Cuda cores:            256
 Arch:                  Pascal

Benchmarks (ms):
 eigen dbl SVDj:        106859
 cusolve dbl SVDj:      10252
 16Mb disk wr:          103
 16Mb disk rd:          18
 arma DGEMM:            676
 arma SGEMM:            588
 eigen SGEMM:           265
 cublas SGEMM:          3
 arma GaussMM:          6
 cufft FFT2:            61
 arma FFT2:             435
 cpu radon:             22969
 cuda radon:            5124
 eigen dbl spLU anlys:  14
 eigen dbl spLU fact:   30
 eigen flt spLU anlys:  14
 eigen flt spLU fact:   34
 eigen dbl BiCG cmpt:   0
 opencv circ detect:    5
 opencv cubic interp:   4
 opencv line detect:    5
```

Ideapad (battery)

```
Inputs:
 sparse matrix:         data/frac.mtx
 dense mat side:        1024
 FFT2 size:             2048
 test image:            data/wells1.png

GPU device(s):
 clock rate:            1531000
 Major/minor:           6/1
 SMP count:             2
 Global mem:            2099904512
 Sh mem / block:        49152
 Name:                  GeForce MX230
 Max threads / block:   1024
 registers / block:     65536
 Peak mem Gb/s:         56.064
 Cuda cores:            256
 Arch:                  Pascal

Benchmarks (ms):
 eigen dbl SVDj:        102346
 cusolve dbl SVDj:      10250
 16Mb disk wr:          47
 16Mb disk rd:          9
 arma DGEMM:            678
 arma SGEMM:            592
 eigen SGEMM:           265
 cublas SGEMM:          3
 arma GaussMM:          6
 cufft FFT2:            64
 arma FFT2:             437
 cpu radon:             23103
 cuda radon:            5145
 eigen dbl spLU anlys:  14
 eigen dbl spLU fact:   30
 eigen flt spLU anlys:  14
 eigen flt spLU fact:   34
 eigen dbl BiCG cmpt:   0
 opencv circ detect:    4
 opencv cubic interp:   14
 opencv line detect:    5
```

Jetson Nano (10W)

```
Inputs:
 sparse matrix:         data/frac.mtx
 dense mat side:        1024
 FFT2 size:             2048
 test image:            data/wells1.png

GPU device(s):
 clock rate:            921600
 Major/minor:           5/3
 SMP count:             1
 Global mem:            4148305920
 Sh mem / block:        49152
 Name:                  NVIDIA Tegra X1
 Max threads / block:   1024
 registers / block:     32768
 Peak mem Gb/s:         0.204
 Cuda cores:            128
 Arch:                  Maxwell

Benchmarks (ms):
 eigen dbl SVDj:        398237
 cusolve dbl SVDj:      40744
 16Mb disk wr:          177
 16Mb disk rd:          35
 arma DGEMM:            2318
 arma SGEMM:            1823
 eigen SGEMM:           1396
 cublas SGEMM:          56
 arma GaussMM:          42
 cufft FFT2:            1267
 arma FFT2:             1256
 cuda_radon::radon_gpu_fine failed
 eigen dbl spLU anlys:  61
 eigen dbl spLU fact:   146
 eigen flt spLU anlys:  61
 eigen flt spLU fact:   137
 eigen dbl BiCG cmpt:   0
 opencv circ detect:    14
 opencv cubic interp:   23
 opencv line detect:    78
```

Jetson Nano (5W)

```
Inputs:
 sparse matrix:         data/frac.mtx
```

```
 dense mat side:        1024
 FFT2 size:             2048
 test image:            data/wells1.png

GPU device(s):
 clock rate:            921600
 Major/minor:           5/3
 SMP count:             1
 Global mem:            4148305920
 Sh mem / block:        49152
 Name:                  NVIDIA Tegra X1
 Max threads / block:   1024
 registers / block:     32768
 Peak mem Gb/s:         0.204
 Cuda cores:            128
 Arch:                  Maxwell

Benchmarks (ms):
 eigen dbl SVDj:        470872
 16Mb disk wr:          320
 16Mb disk rd:          47
 arma DGEMM:            2946
 arma SGEMM:            2595
 eigen SGEMM:           2112
 cublas SGEMM:          59
 arma GaussMM:          66
 cufft FFT2:            1949
 arma FFT2:             1853
 cuda_radon::radon_gpu_fine failed
 eigen dbl spLU anlys:  95
 eigen dbl spLU fact:   228
 eigen flt spLU anlys:  94
 eigen flt spLU fact:   214
 eigen dbl BiCG cmpt:   0
 opencv circ detect:    22
 opencv cubic interp:   74
 opencv line detect:    135
```