Why GPU parallelism is a thing.

David Celný

Department of Physical Chemistry, UCT Prague

celnyd@vscht.cz

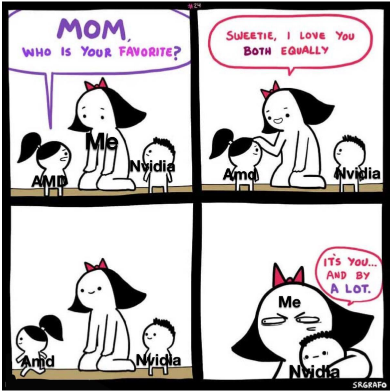
September 2, 2021

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Overview

What is GPU.
Versus CPU?
Why GPU?
How GPU?
Conclusion



What is GPU?

Macmillan.com

"The part inside a computer that changes information into images."

PCmag.com

"Graphics Processing Unit is a programmable processor specialized for rendering all images on the computer's screen."

Wikipedia.org

"A graphics processing unit (GPU) is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device."

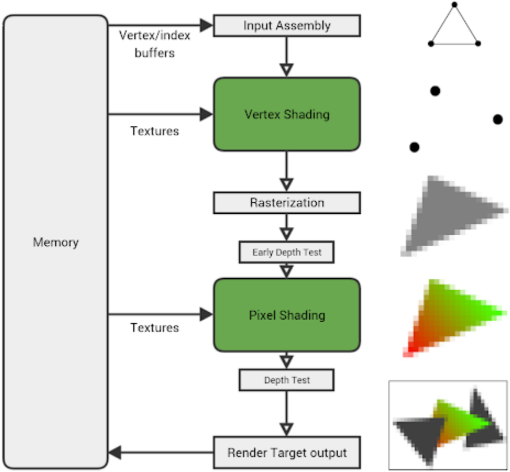


Figure: How the GPu renders in image. © fragmentbuffer.com

This will be the GPU hardware used for comparison.

Figure: Nvidia RTX 3090 GPU featuring Ampere micro-architecture chip.
@ amazon.com

Design comparison

CPU	GPU
1. General tasks *(implicitly serial)*	1. Rendering graphics *(implicitly parallel)*
2. High frequency *($\leq$ 4.5 GHz)*	2. Lower frequency *($\leq$ 1.6 GHz)*
3. Large cache size *(reg, L1, L2, L3)*	3. Reduced cache size *(reg, L1, *L2)*
4. Small # of cores *(2-64)*	4. Huge # of cores *(100 - 10000)*
5. Versatile units *(if,exp)*	5. Specialized cores *(exp, sp/dp, tc)*
6. Latency focused	6. Throughput focused

Chip comparison

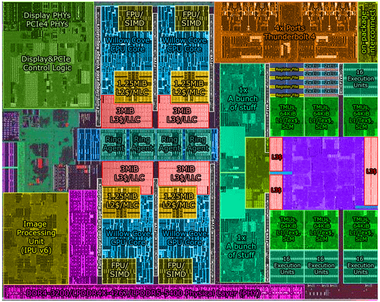


Figure: Intel Tiger Lake 2020
@ Tom's Hardware

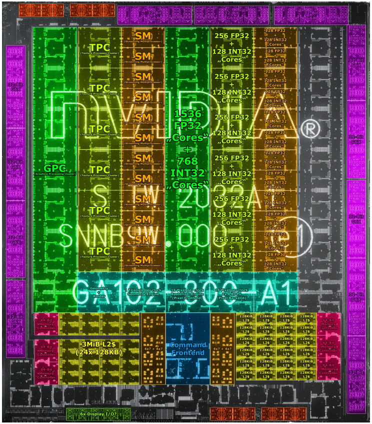


Figure: Nvidia Ampere GA102 2020
@ Wccftech

Consequences

+ additional computation unit *(1)*
+ faster execution on parallel tasks *(4,6)*
+ significant **speedup** for fitting tasks *(all)*
+ lower energy consumption per flops

? demands into algorithm structure *(1)*

- need to synchronize/ wait for clock adjust *(2)*

? manage data memory traversal *(3)*

- demand **parallel** problem *(4)*
- adjust the according to used hardware *(5)*

? hiding memory latency *(6)*

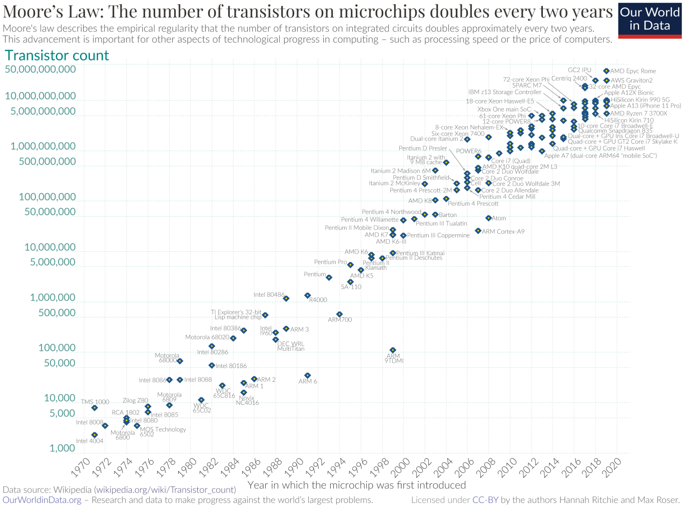


Figure: By Max Roser, Hannah Ritchie - https://ourworldindata.org/uploads/2020/11/Transistor-Count-over-time.png, CC BY 4.0, https://commons.wikimedia.org/w/index.php?curid=98219918

Where the CPU limits lie?

- The cap on maximal achievable frequency of CPU
  - related to operational current
  - heat generation during switching
- on the size
  - manufacturing technologies
  - quantum effects
- peripheries requirements and space efficiency
- standards and compatibility
- # of core-core connection on chip (UMA)

How much can parallelism help?

- not everything can be done in parallel
- type of problem determine parallel potential
- is it worth to invite more units to work
  - with regard to achievable speedup *(Amhdal)*
  - with regard to efficiency *(Gustafson-Barsis)*
- parallel is not always faster

Theorem (speedup definition)

$$speedup_{single\ PU} = \frac{t_{execution,A}}{t_{execution,B}} = \frac{t_A}{t_B} = s$$

Theorem (Amhdal's law)

For n_B number of B type PU's and α parallelized portion of code,

$$speedup_{multiple\ PU} = \frac{t_A}{(1-\alpha)t_B + \frac{\alpha \cdot s}{n_B}} = \frac{1}{(1-\alpha)s + \frac{\alpha \cdot s}{n_B}} = r$$

$$\lim_{n_B \to \infty} r = \frac{1}{(1-\alpha)s}$$

Question: How many cheaper PU type B with $s = 10$ should be bought for execution improvement of code when 90% of code can be run in parallel?

Theorem (sequential/parallel time)

For t_p total time code needs to run in parallel on n PU,

$$t_{sequential} = t_s = (1-\alpha) \cdot t_p + n \cdot \alpha \cdot t_p$$

Theorem (Gustafson-Barsis Law)

$$speedup = \frac{t_s}{t_p} = \frac{(1-\alpha) \cdot t_p + n \cdot \alpha \cdot t_p}{t_p} = (1-\alpha) + n \cdot \alpha$$

$$efficiency = \frac{speedup}{n} = \frac{1-\alpha}{n} + \alpha$$

Question: Note what Gustafson-Barsis law considers as baseline and discuss how does the law differ from Amhdal's.

Summary

- GPU draw images
- GPU is different than CPU
- CPU won't always improve
- to work with GPU one has to code differently
- there are two speed laws (Amhdal, Gustafson-Barsis)

References

Gerassimos Barlas (2015)
Multicore and GPU Programming: An Integrated Approach
Elsevier publishers ISBN: 978-0-12-417137-4

Thomas Sterling, Matthew Anderson & Maciej Brodowicz (2018)
High Performance Computing: Modern Systems and Practices
Elsevier publishers ISBN: 978-0-12-420158-3

Jason Sanders & Edward Kandrot (2011)
CUDA by Example: An introduction to General-Purpose GPU programming
Addison-Wesley ISBN-10: 978-0-13-138768-3

List of Nvidia graphics processing units (cited 2021)
https:\\en.wikipedia.org\wiki\List_of_Nvidia_graphics_processing_units

- overview image [Cit. 02.09.2021]. Available from https:\\www.reddit.com\r\pcmasterrace \comments\gcpgzd\in_terms_of_gpu
- until next time image [Cit. 02.09.2021]. Available from https:\\www.reddit.com\r\pcmasterrace \comments\6mhb7t\when_your_gpu_doesnt_know_how_good_its_life_is

Homework

1. Enable CUDA on your machine.
   - If you have Nvidia GPU then setup the CUDA toolkit.
   - If not set up remote access to school cluster.
2. test the setup by running the *deviceQuery*
   - get the code from nvidia toolkit
   - locate it: *..path_to_cuda/samples/1_Utilities/deviceQuery*
   - compile it there with call to *$make*
3. note down following parameters
   - Device name
   - CUDA capability
   - global memory[GB], L2 Cache size[MB], shared memory[kB], constant memory[kB]
   - maximum number of threads per multiprocessor, -——- per block
   - device ECC support

Until next time