Why is my algorithm so slow?
Parallel computing on the GPU
Evolution of GPGPU

GPU programming

Hi!

Iván Eichhardt

- ivan.eichhardt@sztaki.mta.hu

Course materials

- cg.elte.hu/~gpgpu/opencl

Topics

- Introduction to parallel computing
- OpenCL
- Parallel computing with a theoretical approach

Why is my algorithm so slow?!

Why is my algorithm so slow?

- Is light too slow?!
- Too few trasistors?
- I guess I should optimize my program...

Light is indeed too slow...

- Think about it..
  - The speed of light is about 300.000 km/s.
  - Let's assume a PC with a CPU at 3,5 GHz...
    - ...and it can add two floating point numbers in 2 clock cycles
  - Attach a USB HDD with a cord of 1 meter
    - ...the data we'd like to process is stored on it.

How many additions could be issued while waiting for the data-to-be-processed to arrive?

Light is indeed too slow... (solution)

- CPU clock speed 3.5Ghz = $3.5*10^9$Hz
  - A cycle is $1/(3.5*10^9$Hz$)=2/7*10^{-9}$s ~ 285,7 psec (picosecond).
  - Two cycle (float addition) takes $4/7*10^{-9}$s ~ 571,8 psec time.
- Meanwhile on the cord, the light travels („s=v*t”)
  c$*4/7*10^{-9}$s ~ $3*10^8$m/s * $4/7*10^{-9}$s = $12/7*10^{-1}$m ~ 0.17m
  - The cord is 1 m long. That’s 1m/($12/7*10^{-1}$m) ~ 6 operations.
- So the CPU was doing nothing in the meantime.
- and there are other slowing factors...
  - E.g. the typical HDD access time/latency is around 10 msec...
    ...so the CPU can just go and take a vacation.

Units of time

- 1 ns = 10^-9 seconds
- 1 us = 10^-6 seconds
  - = 1,000 ns
- 1 ms = 10^-3 seconds
  - = 1,000 us
  - = 1,000,000 ns

Some further insights

- Is light too slow?
  - Make data travel less!
  - Move more data in parallel! (32, 64, ...)
  - Use intermediate stores! (e.g. cache)

type	latency	comments
L1 cache reference	0.5 ns	
Branch mispredict	5 ns	
L2 cache reference	7 ns	14x L1 cache
Mutex lock/unlock	25 ns	
Main memory reference	100 ns	20x L2 cache, 200x L1 cache
Disk seek	10,000,000 ns	

Latency comparison ~2012

type	latency	comments
L1 cache reference	0.5 ns	
Branch mispredict	5 ns	
L2 cache reference	7 ns	14x L1 cache
Mutex lock/unlock	25 ns	
Main memory reference	100 ns	20x L2 cache, 200x L1 cache
Read 4K randomly from SSD*	150,000 ns	~1GB/sec SSD
Read 1 MB sequentially from memory	250,000 ns	
Round trip within same datacenter	500,000 ns	
Read 1 MB sequentially from SSD*	1,000,000 ns	~1GB/sec SSD, 4X memory
Disk seek	10,000,000 ns	20x datacenter roundtrip
Read 1 MB sequentially from disk	20,000,000 ns	80x memory, 20X SSD

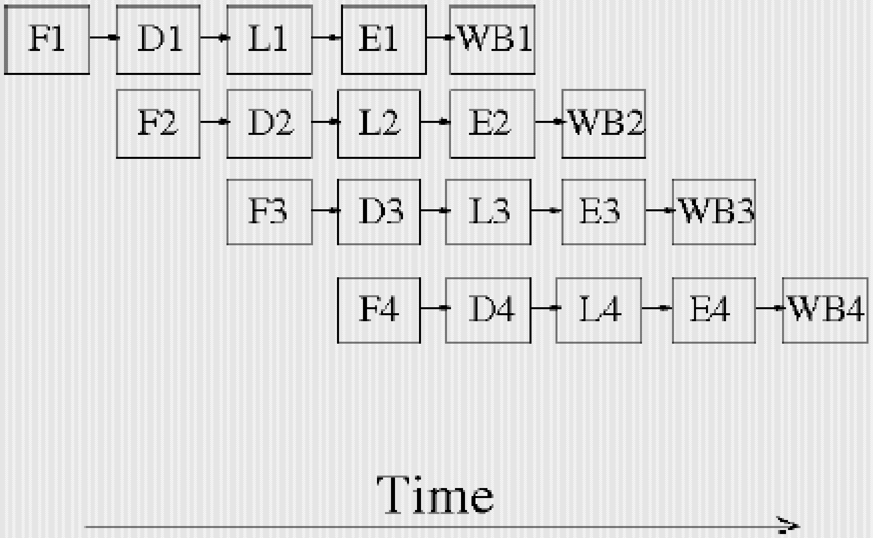
Too few transistors…

- Assume our algorithm is designed sequentially.
- Independednt steps could be executed in „at the same time"…
  - The key is: parallelism.
- But everyhing has its cost.

Too few transistors…

- Too few transistors?
  - Parallel architectures!

The assembly line principle: parallel execution of subtasks



Too few transistors… (approaches)

- Kevés a tranzisztor?
  - Párhuzamos architektúrák!

- **SISD** – Single Instruction Single Data
  - An instruction is only considered with its data.
- **MIMD** – Multiple Instruction Multiple Data
  - Multiple instructions work on various data.
    - Multiple processors
    - Multiple threads..
- ~~MISD – Multiple Instruction Single Data~~
  - … for robustness.
- **SIMD** – Single Instruction Multiple Data
  - The same instruction operated on multiple data.

Outlook: miniaturization

- Make more transistors fit the same chip area.
- e.g. ~14 nm

- Limits:
  - At atomic scales: leaking current..
    (atomic width ~ 10 – 100 picometer)
  - The picometre is one thousandth (1/1000 × nm).

GPU architectures

- They have undergone great development.
  - ~~Specialized, non programmable hardware.~~
  - …
  - Programmable hardware for general purpose computing.

GPU architectures

- They have undergone great development.

Parallel computing using the GPU

GPGPU – ?

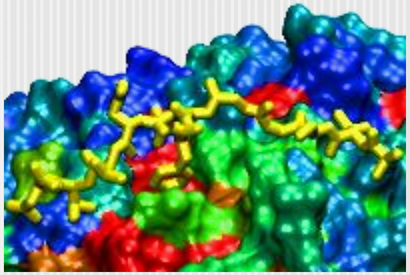
- GPGPU
  - **General-purpose computing on Graphics Processing Units**

- For the average user, the most computational power can be accessed through a GPU.

- Its strength lies in parallelism.

A few GPGPU usecases

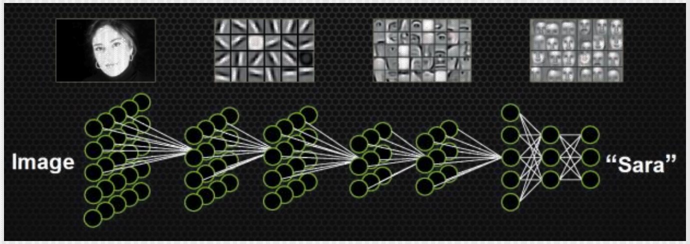
- Simulation of protein folding
  - Folding@home
  - H1N1 simulation



- The lost video of Apollo 11
  - Input: Overwritten video, recording from a monitor playing video, partial copies
  - 100x speedup using GPU

Lowry Digital has started to recover the lost Apollo 11 video, thanks to some difficult digital image processing.

- Training neural networks
  - Deep Learning, etc.



Another comparison

- FPGA, GPGPU, CPU
  - Field-programmable gate array (FPGA)
  - DES decryptin
    „Data Encryption Standard"
    - CPU: 16 million keys / s
    - GPU: 250 million keys / s (GTX-295)
    - FPGA: ~1.8 billion keys / s

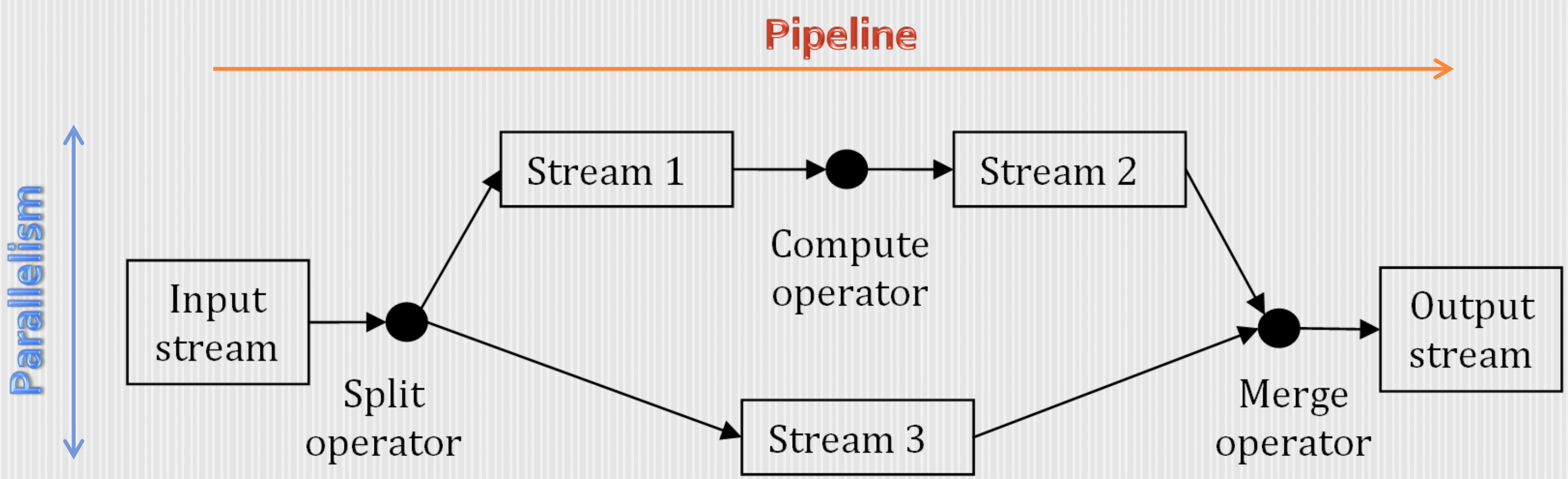
Sources

- H1N1 simultaion
  L. Barney - Studying the H1N1 virus using NVIDIA GPUs, Nov 2009.
  http://blogs.nvidia.com/ntersect/2009/11/studying-the-h1n1-virus-using-nvidia-gpus-.html
- Apollo 11
  R. Wilson - DSP brings you a high-definition moon walk, Sep 2009.
  http://www.edn.com/article/CA6685974.html
- DES decrypting
  Dr. Dobbs - Parallel algorithm leads to crypto breakthrough, Jan 2010.
  http://www.ddj.com/222600319
- The problems with GPGPU
  A. Ghuloum - The problem(s) with GPGPU, Oct 2007.
  http://blogs.intel.com/research/2007/10/the_problem_with_gpgpu.php

Evolution of GPGPU

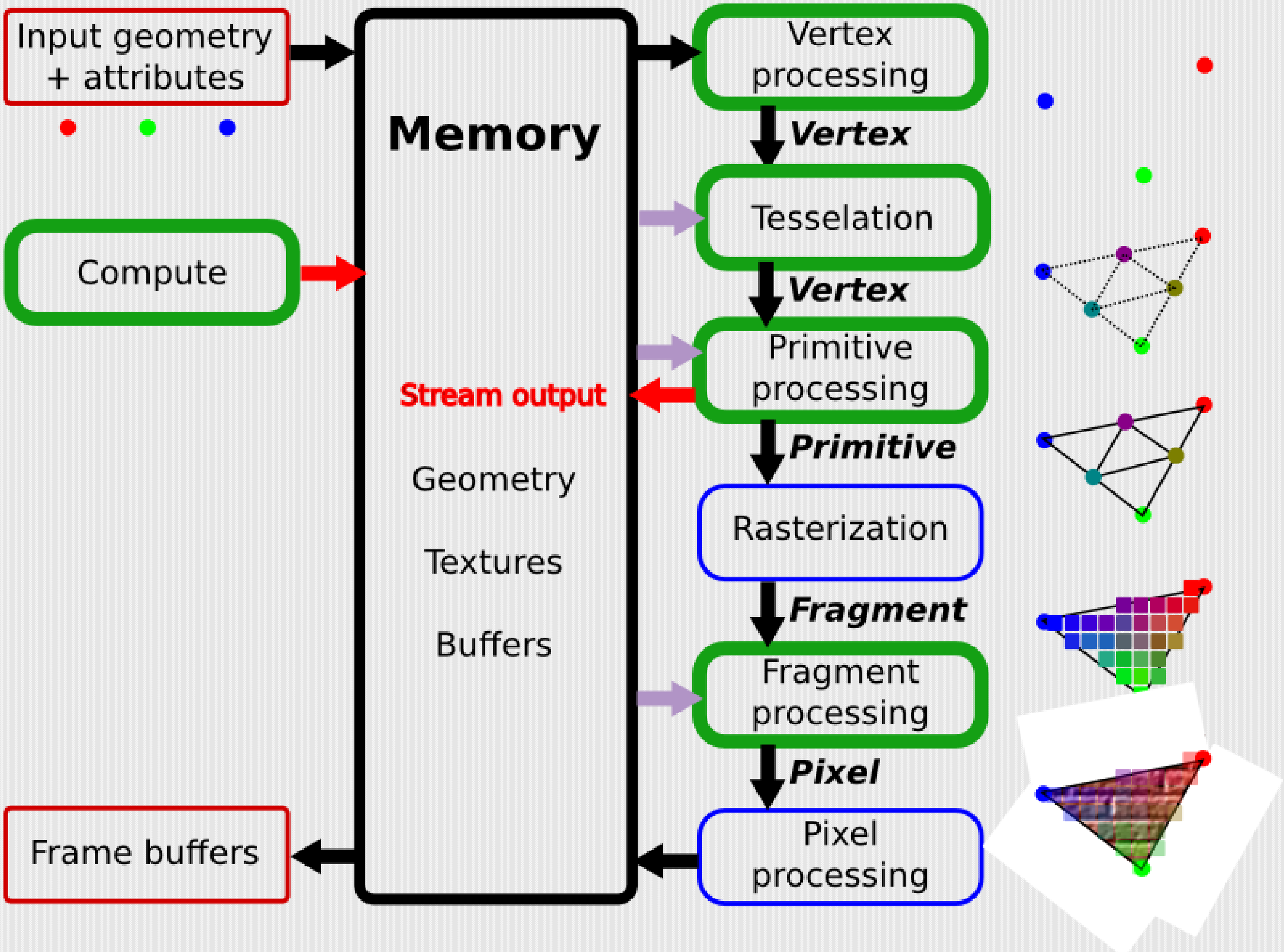
Stream-processing

- No synchronization or communication
- Applying a pipeline
- Parallelism



- Basic operations: Map, Amplify, Reduce, Sum

Stream processing in the graphics pipeline (on the GPU)



Vector-processing

- SIMD
  - GPU multiprocessor
    (e.g. Vertex attirbute streams)
  - CPU extensions (SSE*, 3DNow!, MMX, ...)
  - Data-centric, easy to parallelize
- Vectorization: the data is organized as vectors
  - E.g. *(vec_res, v1, v2 4×32 bit float vectors)*:

    ```
    vec_res.x = v1.x + v2.x;
    vec_res.y = v1.y + v2.y;
    vec_res.z = v1.z + v2.z;
    vec_res.w = v1.w + v2.w;
    ```
  - You could use a single instruction to perform all above...

Vector-processing

- Manhattan distance of 32-bit length binary strings

- Loop? (Sequential solution)

```
int bitcount_naive(int x)
{
    int count = 0;
    while (x != 0) {
        if ((x & 1) == 1) { count++; }
        x >>= 1;
    }
    return count;
}
```

Vector-processing

- Manhattan distance of 32-bit length binary strings
- „Parallel" solution

```
unsigned int bitcount(unsigned int x)
{
    x = (x & (0x55555555)) + ((x >> 1) & (0x55555555));
    x = (x & (0x33333333)) + ((x >> 2) & (0x33333333));
    x = (x & (0x0f0f0f0f)) + ((x >> 4) & (0x0f0f0f0f));
    x = (x & (0x00ff00ff)) + ((x >> 8) & (0x00ff00ff));
    x = (x & (0x0000ffff)) + ((x >> 16) & (0x0000ffff));
    return x;
}
```

Vector-processing

- Manhattan distance of 128-bit binary strings
- Use SIMD operations!

```
unsigned int bitcount_128(unsigned int4 x)
{
    const unsigned int4 a1(0x55555555, 0x55555555, 0x55555555, 0x55555555);
    const unsigned int4 a2(0x33333333, 0x33333333, 0x33333333, 0x33333333);
    const unsigned int4 a3(0x0f0f0f0f, 0x0f0f0f0f, 0x0f0f0f0f, 0x0f0f0f0f);
    const unsigned int4 a4(0x00ff00ff, 0x00ff00ff, 0x00ff00ff, 0x00ff00ff);
    const unsigned int4 a5(0x0000ffff, 0x0000ffff, 0x0000ffff, 0x0000ffff);

    x = (x & (a1)) + ((x >> 1) & (a1));
    x = (x & (a2)) + ((x >> 2) & (a2));
    x = (x & (a3)) + ((x >> 4) & (a3));
    x = (x & (a4)) + ((x >> 8) & (a4));
    x = (x & (a5)) + ((x >> 16) & (a5));
    return x.x + x.y + x.z + x.w;
}
```

Heterogeneous computing vs GPGPU

GPGPU

- GPU-s
  - Stream-processing
  - Compute Shader
  - CUDA
  - stb.
- (might be closer to hardware)

HETEROGENEOUS COMPUTING

- CPU, GPU, FPGA, etc..
- OpenCL standard
  - Open