GPHUngarian

An efficient GPU implementation of the Hungarian algorithm for shape matching problems

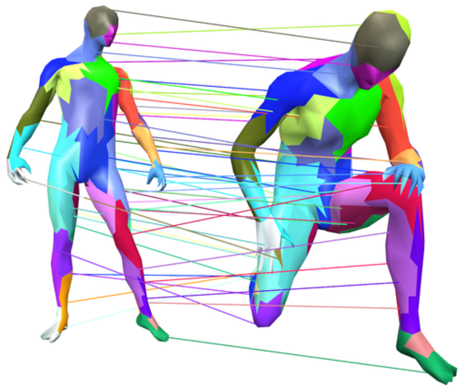
Daniele Solombrino

March 2021

Sapienza
Università di Roma

3D Shape Matching

- **3D Computer Vision** domain
- Find a **match** between **similar parts** in two distinct 3D objects
- **Not easy**
  - 3D shapes are **deformable**
  - Task **success unrelated** to object **pose**
- Part of **other** important **tasks**:
  - Object Recognition
  - Object Classification



The Hungarian Algorithm

- **3D Shape Matching** can be solved via the **Linear Assignment Problem**
- Lots of Linear Assignment solvers
  - **The Hungarian Algorithm**
  - Auction Algorithm
  - Linear programming

The challenges

- Hungarian Algorithm **not directly** portable to GPU as is
- **Slow** execution times on big inputs

Scientific goals

- Make the Hungarian Algorithm **GPU implementable**
- Provide GLADIA Sapienza Lab with a **faster** 3D Shape Matching solver

Personal motivations

- Learn how to "**accelerate**" algorithms
- **GPUs** rapidly gaining popularity:
  - Heavy computations solved in relatively short time (Machine Learning, Deep Learning…)

The journey
Basic GPU programming (1/3)

- **GPU programs** are called **kernels**
- **Data** used by GPUs must be stored in **Buffers**
- **GPUs** have their own **memory** (called **VRAM**)

The journey
Basic GPU programming (2/3)

- **GPUs** are called **devices**
- **CPU** is the **host**
- “**Host manages** devices”:
  - Kernel
    - Execution
    - Synchronization (if needed)
  - Buffers
    - Creation and deletion
    - Initialization
    - Location (VRAM or RAM)
    - Permissions (Read Only, Write Only or Read and Write)

The Journey

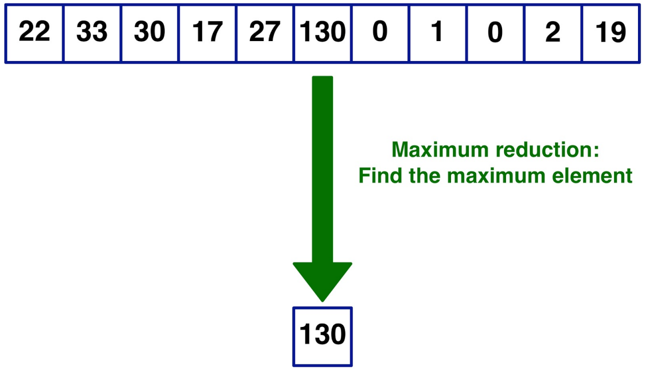
Basic GPU programming (3/3)

- **Single Instruction** simultaneously executed on **Multiple Data** (**SIMD**)
- Highly **specialized** and **optimized** hardware **chips**
- Multiple simultaneous **memory** and **arithmetic** operations

The journey
Advanced GPU programming (1/4)

Reduction design pattern:

- X **input** items **reduced** to Y output elements
- **Commutative** and **associative** Reduction **function**
  - Applied in parallel to multiple elements



The journey

Advanced GPU programming (2/4)

- **Memory access alignment** influences execution time:
  - **Good Alignment**
    - Fully parallelized memory accesses
  - **Bad Alignment**
    - Underutilized hardware
  - **Conflicting Alignment**
    - Fully serialized memory accesses

The journey

Advanced GPU programming (3/4)

Saturation programming pattern:

- **Efficient** Reduction implementation
- Total **control** of memory access alignments

The journey

Advanced GPU programming (4/4)

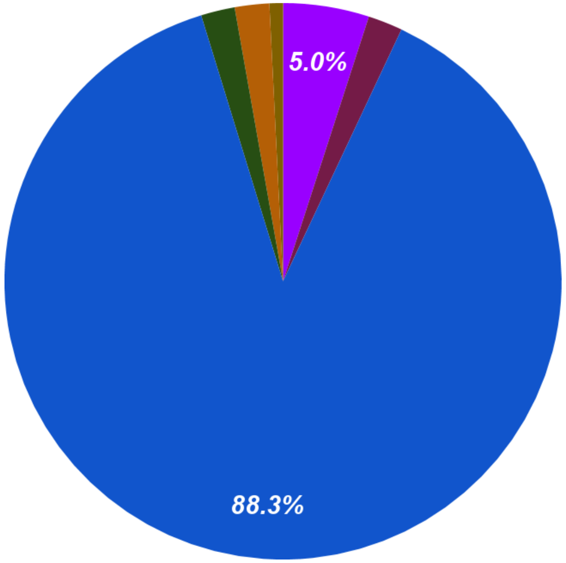
Multi-GPU adoption:

- **Same** kernel simultaneously executed on two or more GPUs
- **Data** (equally) **partitioned** between the different devices
- **Synchronization** between devices

The journey
A Hungarian study

Two **most contributing** steps of the Hungarian Algorithm

- **FiSuCoMi** and **FiBExSo** optimization leads to the **most tangible** speedup
  - Amdahl's law



The journey

Accelerating **FiSuCoMi**

- Easily **portable** to GPU
- First part is a **Reduction** via **Saturation**
- Second portion simply uses **SIMD**

The journey

Accelerating FiBExSo (1/2)

- Direct **porting** is **not possible**
  - Can not use Reduction
  - SIMD adoption results in **data race** conditions
- **Fully restructured**
  - **Data** and related **logic** entirely **reworked**
    - FiBExSo **split** in three steps:
      - **FiB**
      - **Ex**
      - **So**

The journey

Accelerating **FiBExSo** (2/2)

After rework FiB, Ex and So become effortlessly **portable** to GPU

- **FiB** and **So** → **Reduction** via **Saturation**
- **Ex** → **SIMD**

Final Result

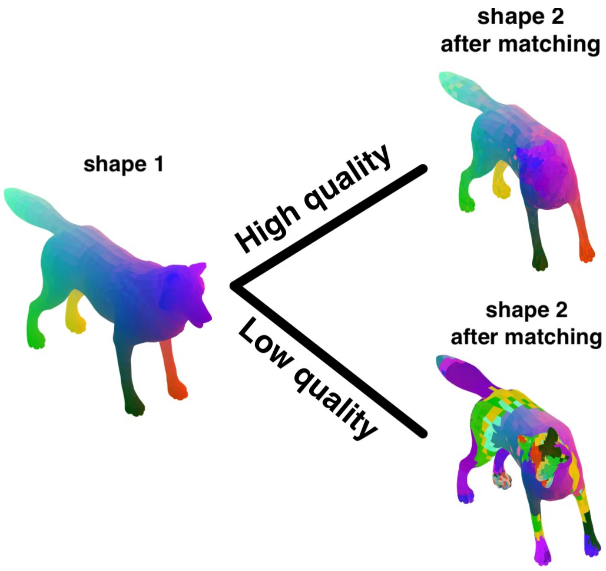
GPU output correctness

Numerical **equivalence** with CPU output

Final Result

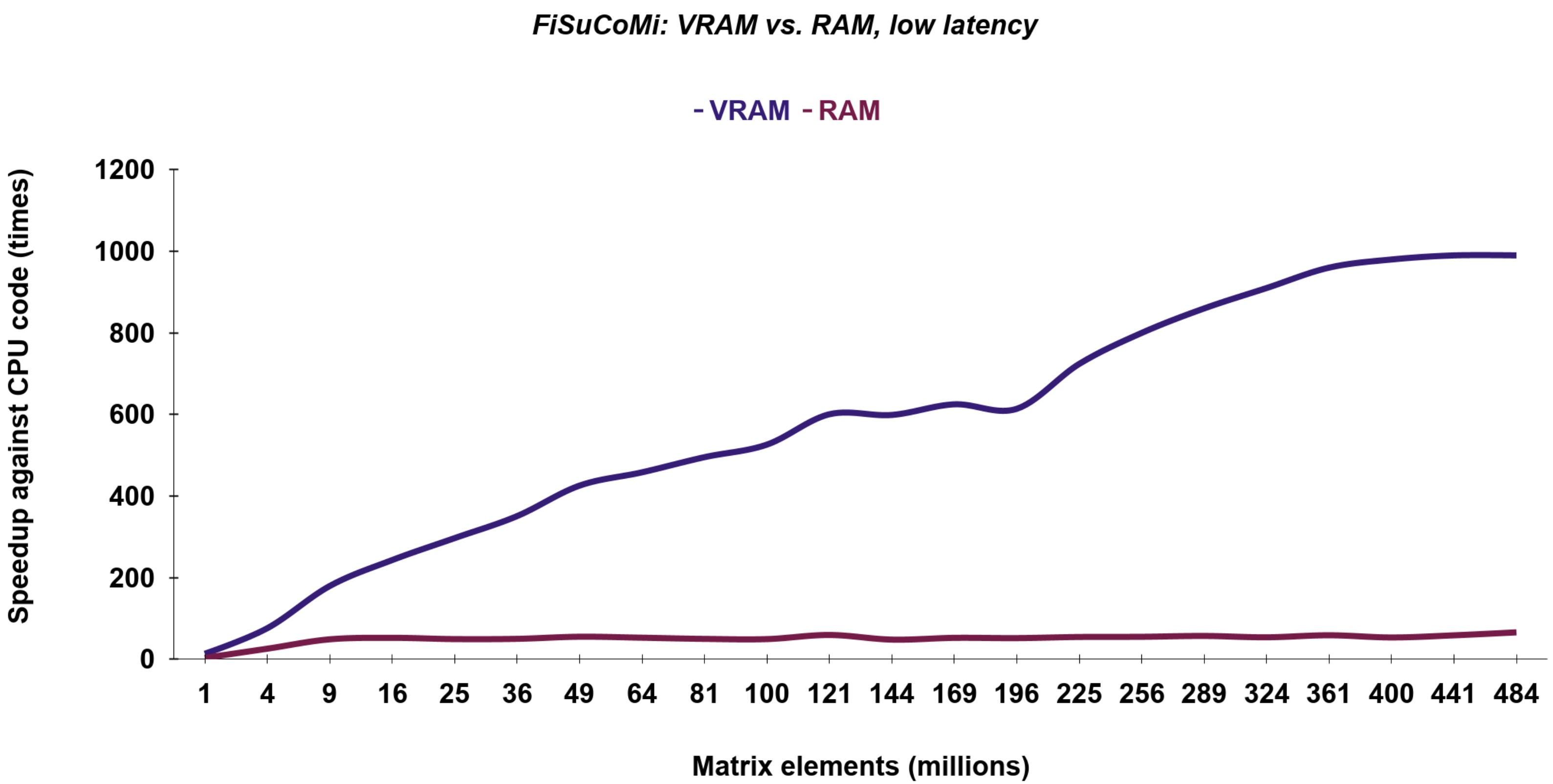
GPU output quality

- Adopts a **visual criteria**
- Uses the output matching to **transfer colors** from shape 1 to shape 2
- **Corresponding** parts colored the **same** way



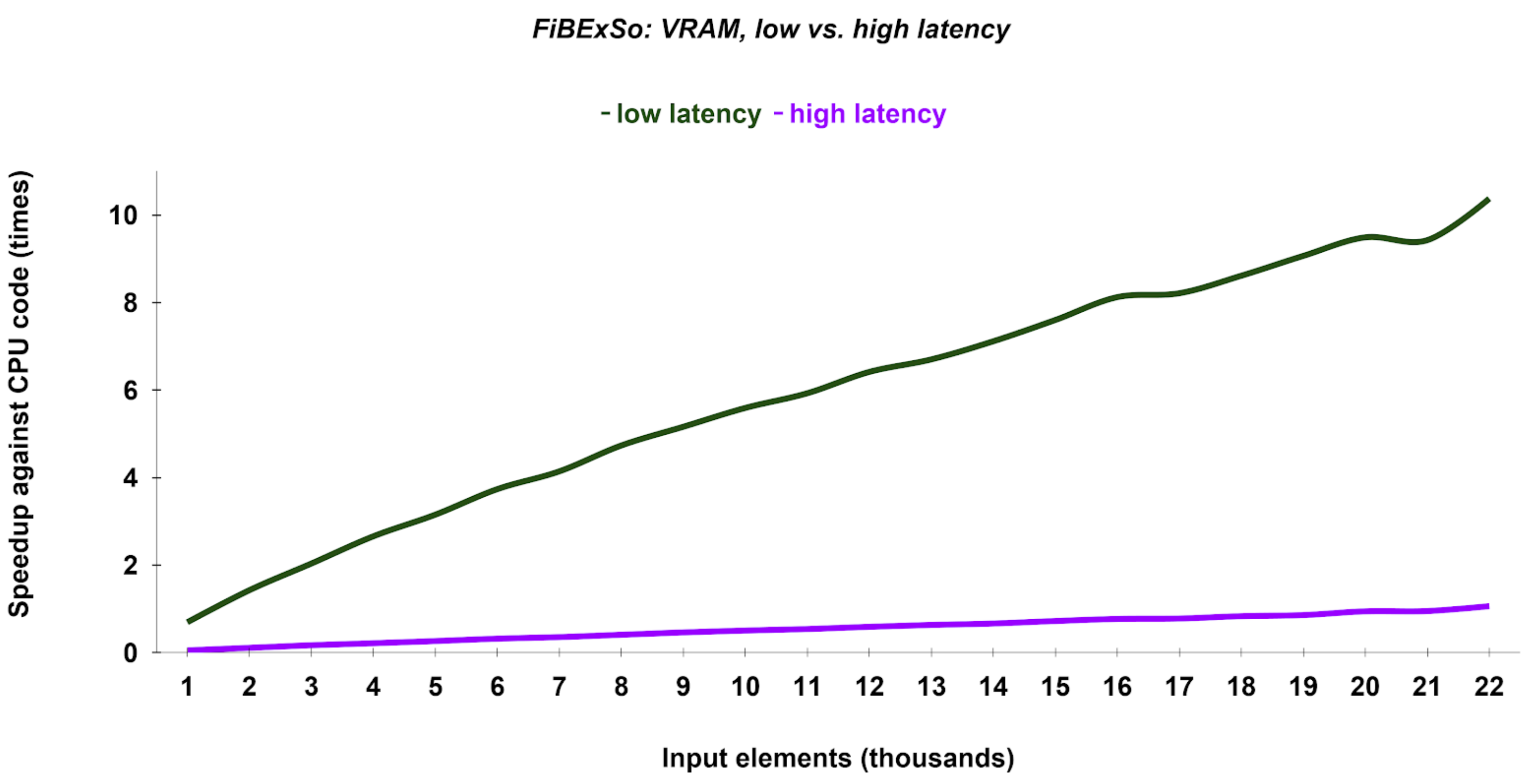
Final Result

Speedups: VRAM vs. RAM



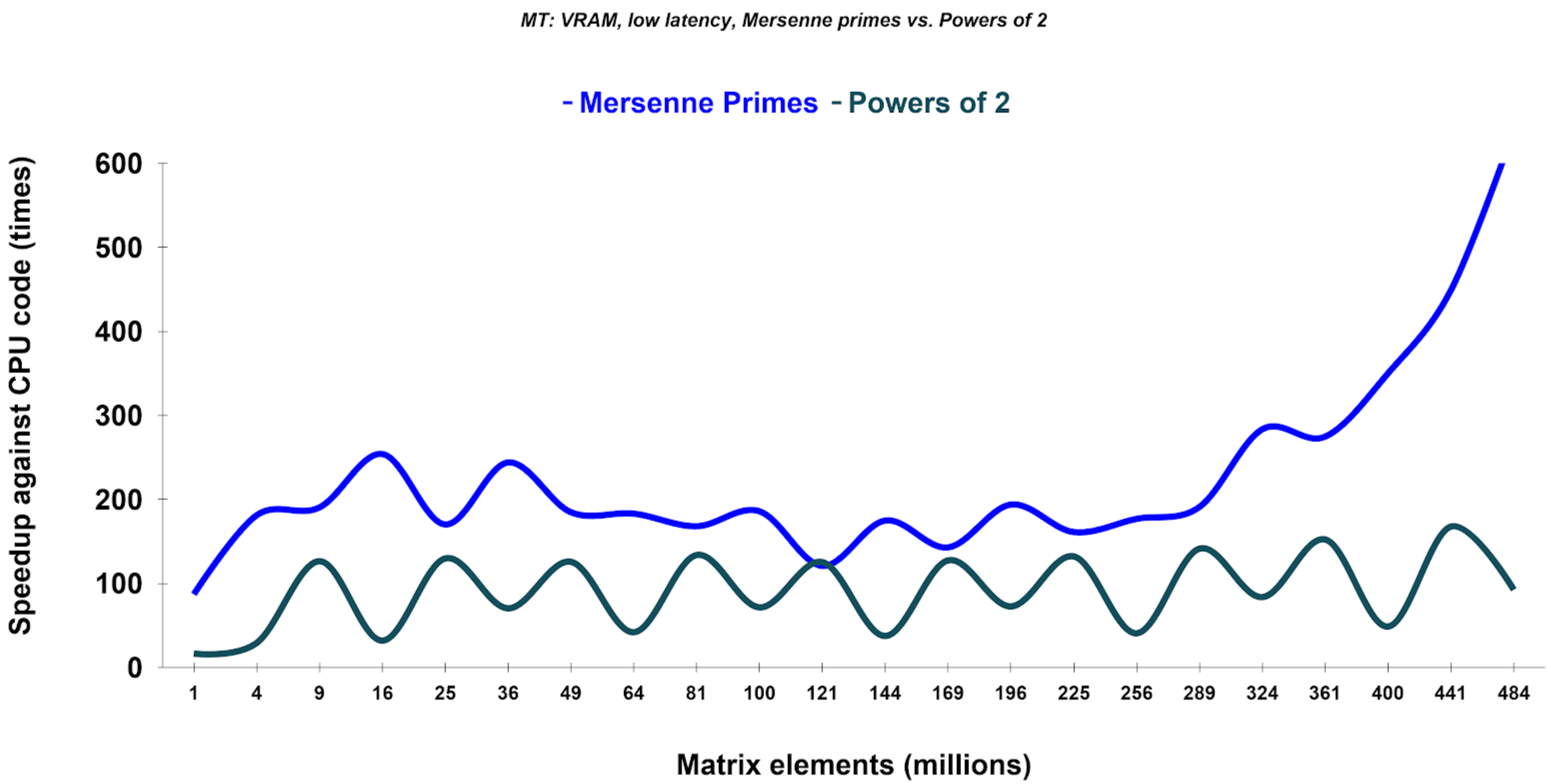
Final Result

Speedups: low vs. high latency



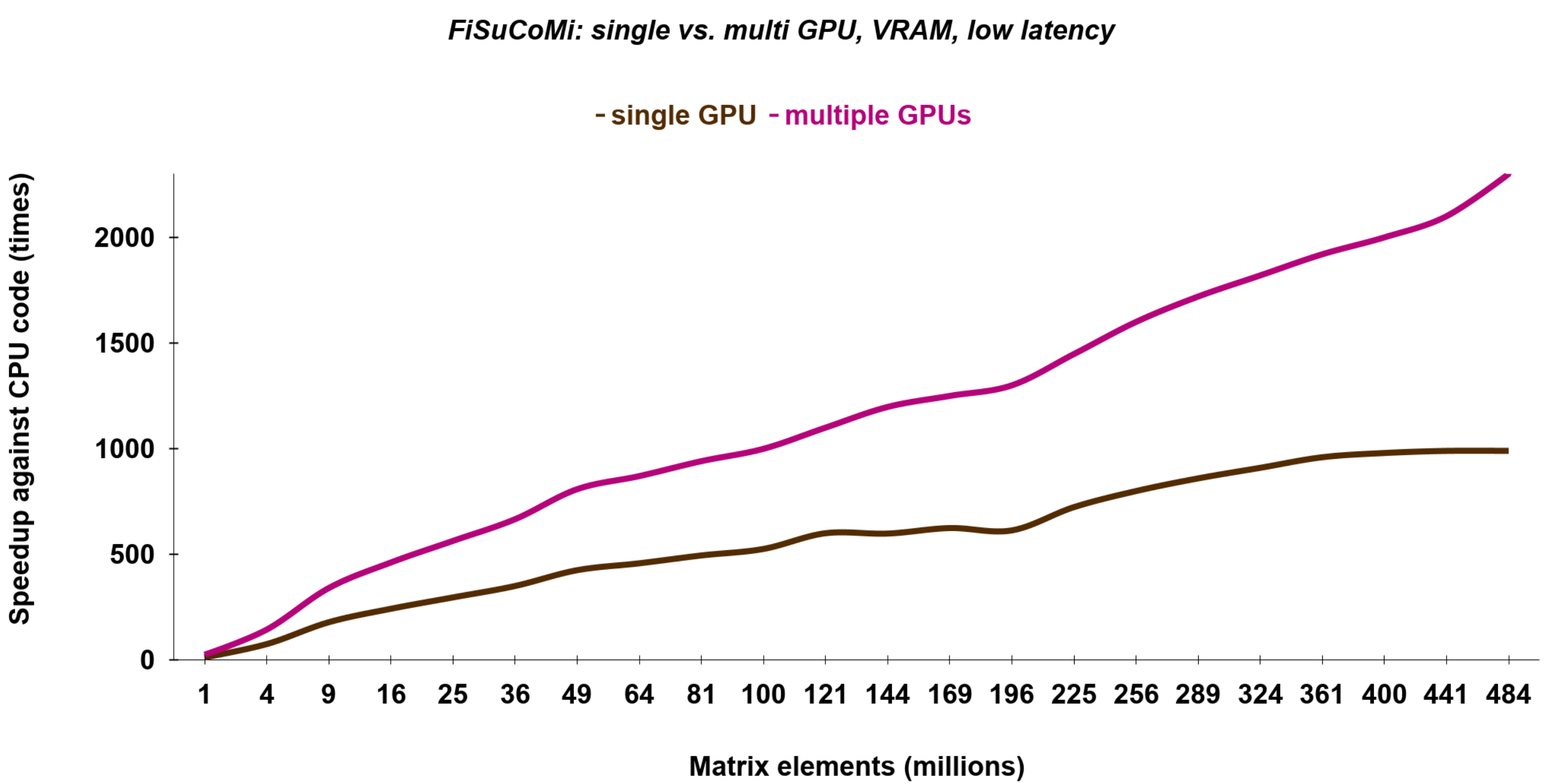
Final Result

Speedups: input data size rounding



Final Result

Speedups: single vs. multi GPU



Final Result
The library

- Includes all cited kernels and **extras**
- **Fast**
- **Open Source**
- **Scalable**
- **Portable**
- **Flexible**
- **Reliable**
- **Expandable**

Conclusions

- Hungarian Algorithm **not directly** portable to GPU
- **Reworked** it to comply with GPUs
- Resulting **GPU** code **faster** than CPU code
- Speedups heavily **influenced** by
  - Buffer location (**VRAM** vs. **RAM**)
  - Latency (**low** vs. **high**)
  - Input data size (**thousands** vs. **millions** of elements)
  - Input data size rounding (**Mersenne Primes** vs. **powers of 2**)
  - Number of devices (**single** vs. **multi** GPUs)

Future developments

- Deploy in “**local** host, **remote** devices” environments
  - Cloud computing
- Test support for **more** than **2** simultaneous **GPUs**
  - Clusters