GPUs in gravitational wave research

Gergely Debreczeni
(Debreczeni.Gergely@wigner.mta.hu)

Head of
Gravitational physics Research Group,
& GPU Laboratory
Wigner RCP

Computing Coordinator
Virgo Experiment
Content

- Wigner GPU laboratory
- The Virgo experiment
- The way we work with GPUs
  - The Compute Wrapper, CUDA/OpenCL compute backends
- Analysis examples
  - Coalescing Compact Binaries
    - The matched-filter
  - Continuous Waves
    - The frequency Hough algorithm
- Plans with Tegra K1
- Conclusion
Content

- Wigner GPU laboratory
- The Virgo experiment
- The way we work with GPUs
  - The Compute Wrapper, CUDA/OpenCL compute backends
- Analysis examples
  - Coalescing Compact Binaries
    - The matched-filter
  - Continuous Waves
    - The frequency Hough algorithm
- Plans with Tegra K1
- Conclusion
The Wigner GPU laboratory

The Wigner GPU laboratory

http://gpu.wigner.mta.hu

- Small scale heterogenous cluster
- Serves as a playing ground for researchers in the Institute
- Support by experts: Nagy-Egri Mate Ferenc
- Various cards: Nvidias, AMDs, Phis
General Relativity

Observables

Experimentally prooved:
1. Perihelium shift of planets
2. Gravitational lensing
3. Time dilatation/contraction in strong/weak gravitational fields

4. ... and the existence of gravitational waves...
   - Gravitational waves are ripples/waves of space-time itself created by certain energetic acceleration of huge masses.
   - As of today, we have only indirect – but quite convincing – proof of their existence
A Hulse-Taylor pulsar

- The „Holy Grail” of gravitational wave researchers
- Discovered in 1974 (Russel Hulse, Joseph Taylor)
- 1993 Shared Nobel-prize

- Probably two neutron star
- 3.1 mm shrink for every orbit cycle
- 59 ms-os pulsating period
- 7.75 hour orbital period
Possible GW sources

- **Compact Coalescing Binaries**
- **Pulsars**
- **Supernovae**
- **Stochastic Waves**

**What powered the big bang?**

- Only gravitational waves can escape from the earliest moments of the Big Bang.
- Inflation (Big Bang plus $10^{-35}$ seconds?)
- Cosmic microwave background, distorted by seeds of structure and gravitational waves
- Gravitational waves
- Big Bang plus 300,000 Years
- Big Bang plus 15 Billion Years
- Light
- Now

---

DEBRECZENI, Gergely -- GPUs in gravitational wave research
Why interesting?

- Completely new face of astrophysical objects, of Universe
- One can observe so far hidden phenomenae
- Would be the new „sense-organ” of Humanity
- Testing theories
Interferometers

- Incident GW changes the arm length
- Light intensity changes in the interference point
- The outcome of the measurement is the difference of arm lengths
- The world's most precise relative measurement with a precision of $10^{-18}$.
- Huge collection of environmental noise have to be cancelled
- Isolated detectors are needed for coincidence detection of same event (see LIGO).
The Virgo experiment

- A Virgo detektor az EGO (European Gravitational Observatory) telephelyén az Arno folyó síkságán Pisa melletti Cascina-ban található

- Az építkezés 2003-ban fejeződött be

- A Wigner FK Rácz István vezetésével 2008-ban csatlakozott a kísérlethez

- A francia-olasz együttműködésként induló kollaborációknak ma már magyar és lengyel tagjai is vannak.
The GWTools project

GWTools - the C++/OpenCL [1] based Gravitational Wave data analysis Toolkit - is an algorithm library aimed to bring the immense computing power of emerging many-core architectures - such as GPUs, APUs and many-core CPUs - to the service of gravitational wave research. GWTools is a general algorithm library intended to provide modular building blocks for various application targeting the computationally challenging components of GW data analysis pipelines.
The Compute Wrapper - features

- Forever ongoing discussion about cards and programming languages
- We created a Compute Wrapper (CW) and using multiple backend (CUDA / OpenCL / CPU / etc.)
- Development and testing happens in OpenCL, than the code ported to the appropriate backend.
- OpenCL \rightarrow CUDA kernel translation is trivial (almost semi-automatic)
- CUDA \rightarrow OpenCL can be problematic in some cases and requires special attention
- Built-in GL interop
- The Compute Wrapper also provides a
  - thread pool
  - a host side scheduler with
  - relational workflow handling
  - for CPU side optimisation for optimal load balancing.
The Compute Wrapper – code example

```c
#include <ComputeWrapper/include/ProgramManager.h>
#include <ComputeWrapper/include/Factory.h>
#include <ComputeWrapper/include/IBuffer.h>
#include <System/include/ModulParameterFactory.h>
#include <Utils/include/ImageTypes.h>

int main() {
    // Host side objects and problem size
    math::Size m_size(1024, 2048);
    float * h_buffer = new float[m_size.getCount()];

    // Create device objects
    cw::IBuffer * n_buffer = cw::Factory::createImage2D(m_size, IMAGE_UCHAR_4, MEM_READ_WRITE);
    cw::IImage * n_image = cw::Factory::createBuffer(m_size.getCount() * sizeof(float), MEM_READ_ONLY | MEM_COPY_HOST_PTR, h_buffer);

    // Create a program
    m_program_id = FIND_PROGRAM_ID("hough_transform");

    // Kernel arguments
    cw::cWInt arg1(10);
    cw::cWInt arg2(20);

    // Call the kernel
    PROGRAM_BY_ID(m_program_id)("integral_map", cw::Range(global_size_x, global_size_y), cw::Range(32, 23), 4, m_buffer, m_image, &arg1, &arg2);

    // Finish every kernel execution
    cw::ContextManager::FinishQueue();

    // Read back the results
    m_buffer->readBuffer(h_buffer);

    // Exiting
    return 0;
}
```
About gravitational wave data

- GW data analysis is compute intensive not data intensive
- By quantity it is much less than LHC data: $O(100-300 \text{ TB} / \text{detector} / \text{year})$
- However, it's arithmetic density is much higher
- Available computing power directly translates to detector sensitivity!
  - For CW searches sensitivity goes as $\sim 1.0 / \sqrt{T}$, while necessary computing power is $\sim \exp(T)$
  - For CBC $\sim$ number of background estimation and lowering clustering thresholds $\sim N$
- In contrast to HEP experiments, events cannot be regenerated
- Quick, low latency analysis are necessary for externally triggered searches
- Order of CPU cores $\sim 20K$
- Order of GPUs $\sim 300$
Analysis example I:

Compact Binary Coalescence
Compact Binary Coalescence

\[(A, B) = \int \frac{A^*(f)B(f)}{S_n(f)} df,\]

\[\text{SNR} = \frac{z}{\sqrt{\langle (\delta z)^2 \rangle}} = \frac{\langle \tilde{Q}, \tilde{s} \rangle}{\sqrt{\langle \tilde{Q}, \tilde{Q} \rangle}},\]
Compact Binary Coalescence

Algorithmic steps of the analysis:

- Spectral density calculation (OK)
- Theoretical template generation (OK)
- Template filtering
  - vector multiplication (OK)
  - Inverse FFT (OK)
  - Maximum finding (OK)
  - Clustering (can be challenging depending on thresholds)
- Post-processing (on CPU)

- Golden application of GW searches
- Runs on AMD and Nvidia
- CUDA FFT is superior
- Typical data segment length $10^{20} - 10^{24}$
- Runs mainly on Teslas and cheap GTX cards

Almost optimal in all sense (occupancy, throughput, CPU/GPU balance, etc)!
Analysis example II:

Continuous Waves: Pulsars
CW: The Hough search I

- Searching for **neutron stars with unknown frequency** evolution on the sky
- Nor the frequency neither its time derivatives (spin down parameters) are known -> has to be scanned

Basic steps of the method:

- Segment the data \( (T_{\text{total}}) \) into coherent segments \( (T_{\text{coh}}) \), where the frequency can be approximated to be constant (order of 30 min, but depend on the actual frequency bin).
- Calculate the power spectra of the entire data and normalize the Fourier Transformed segments with it. The normalized FFT domain segments are then thresholded -> converted to 0s and 1s -> the peakmap is obtained in the time observed - frequency plane
- For each position in the sky the peakmap is doppler de-modulated to obtain the (time - real frequency) peakmap. The happens on-the-fly, no need to store them.
CW: The Hough search II

- Taking into account only the first spindown parameter $d$, (i.e. assuming linear relationship) one can write up the following equation:

$$f_{\text{instantaneous}} = f_0 - d \times (t - t_0)$$

that is a straight line in the $f_0 - d$ plane.

- Since we have a finite frequency resolution ($df$) this equation becomes

$$-f_0 / (t - t_0) + (f - df / 2) / (t - t_0) < d < -f_0 / (t - t_0) + (f + df / 2) / (t - t_0)$$

i.e. two straight lines on the plane

- For every point in the peak map (every $t$, $f_{\text{instantaneous}}$ value) draw a line in the $f_0$, $d$ plane and increase the value of each bin by 1 which is intersected by the line, i.e. a 2D histogram.

- There are tricks
  - to increase the resolution one uses oversampled frequency bins
  - and to decrease computational cost one uses differential Hough maps when drawing lines with 1s and -1s and integrating only at the very end. This save significant computational cost, since it not necessary to shade / fill large areas in memory.
CW: The Hough algorithm: The peakmap

- A peakmap example with strong signal

... and the same but weaker signal
CW: The Hough algorithm: The Hough - map

- The Hough map of the previous peakmap with the weak signal

... and the same without signal
A possible implementation

The original peakmap

The Hough map

- Each core reads a (some) peak
- Draws the corresponding line on the Hough map
- Problems
  - looping through on global memory is slow
  - intersection of lines requires atomic operations

- input data
- result data
- memory
- gpu core
A better implementation

The original peakmap (constant memory)

The Hough map (global memory)

- Hough map is partitioned
- Partitions are filled in local memory and copied back to global and the end
- All the line which intersects the given partition are drawn
  - much faster
  - no atomic operations involved
Algorithmic steps of the analysis:

- FFT coherent segments (OK)
- Thresholding (OK)
- Differential Hough-map (OK)
- Integral Hough-map (OK)
- Maximum finding, clustering (can be tricky)

By now also very much optimized
Future (?) and testing: Tegra K1

- Good experiences with Tegra K1
  - Enables for unified memory access
    - (framework handles it)
- 192 compute core
- 4 ARM cores with NEON
- Ubuntu Linux on devboard
- Low power consumption
- CUDA compiles quite OK, profiling, debugging sometime difficult
- Considering for future use
Conclusions

- GPUs are optimal for GW analysis
- With proper framework multi-language, multi-platform development does not require additional manpower
- GW analysis is kind of prepared for the challenges of forthcoming years