GPU version of the Polgraw all-sky F-statistic search code

M. Bejger (Copernicus Center),
collaboration with Aleksander Garus (Jagielonian University)

CW teleconf, 27.11.13

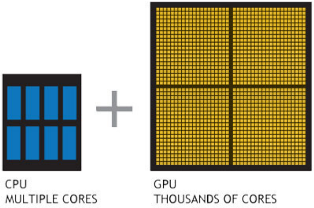
Outline

- CPU vs GPU concept,
- description of the all-sky F-stat candidates' search,
- Results for the GPU version & prospects.

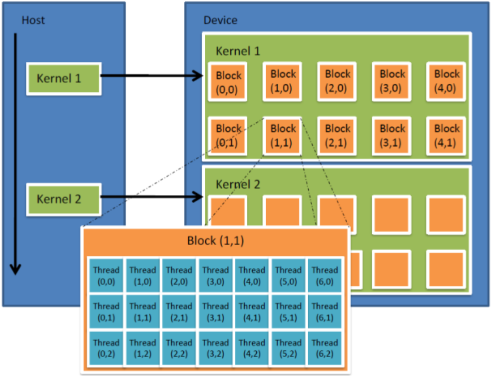
Central Processing Units vs Graphics Processing Units

GPU: **thousands** of smaller ($\Rrightarrow$ more efficient) cores designed for handling **multiple tasks simultaneously**

CPU: a **few** cores optimized for **sequential serial processing**



⋆ Host (CPU) – Device (GPU) interaction, executing many kernels (device functions) in parallel



Platform & programming model for this project: CUDA (Compute Unified Device Architecture) of NVIDIA

C vs CUDA: Hello world! example

```c
#include <stdio.h>
#define N 7

int main() {

    char a[N] = "Hello ";
    int b[N] = {15, 10, 6, 0, -11, 1,0};

    printf("%s", a);

    // addition
    int i;
    for (i = 0; i< N;i++)
        a[i] += b[i];

    printf("%s\n", a);

    return 0;
    // example from
    // http://web.eecs.utk.edu/~lindsey
}
```

```c
#include <stdio.h>
#define N 7

__global__ void add_arrays(char *a, int *b)  {
    a[threadIdx.x] += b[threadIdx.x];
}

int main() {

    char a[N] = "Hello ";
    int b[N] = {15, 10, 6, 0, -11, 1,0};

    char *ad; int *bd;
    const int csize = N*sizeof(char);
    const int isize = N*sizeof(int);

    printf("%s", a);

    cudaMalloc((void**)&ad, csize);
    cudaMalloc((void**)&bd, isize);

    cudaMemcpy(ad, a, csize, cudaMemcpyHostToDevice);
    cudaMemcpy(bd, b, isize, cudaMemcpyHostToDevice);

    dim3 dimBlock(N); dim3 dimGrid (1);
    // addition
    add_arrays<<<dimGrid, dimBlock>>>(ad, bd);

    cudaMemcpy(a, ad, csize, cudaMemcpyDeviceToHost);
    cudaFree(ad);

    printf("%s\n", a);
    return EXIT_SUCCESS;
}
```

Calculation of the F-statistic

$$\mathcal{F} = \frac{2}{S_0 T_0}\left(\frac{|F_a|^2}{\langle a^2\rangle} + \frac{|F_b|^2}{\langle b^2\rangle}\right))$$

where S_0 is the spectral density, T_0 is the observation time, and

$$F_a = \int_0^{T_0} x(t)a(t)\exp(-i\phi(t))dt, F_b = \dots$$

related to the model of the signal

$$h(t) = \sum_{i=1}^{4} A_i\, h_i(t),$$

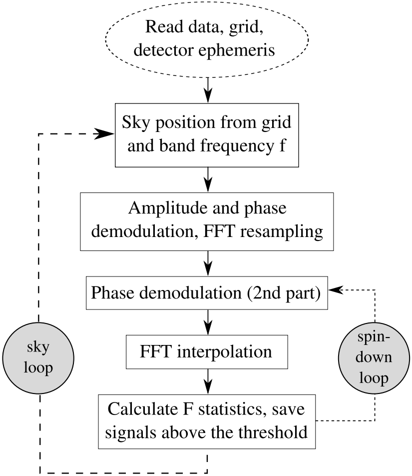
where h_i ($i = 1, \dots, 4$) are

$$h_1(t) = a(t)\cos\phi(t), \quad h_2(t) = b(t)\cos\phi(t),$$

$$h_3(t) = a(t)\sin\phi(t), \quad h_4(t) = b(t)\sin\phi(t),$$

$a(t)$ and $b(t)$ are amplitude modulation functions (depend on the detector location and sky position of the source).

F-stat all-sky search description



Coherent search for CW signal:

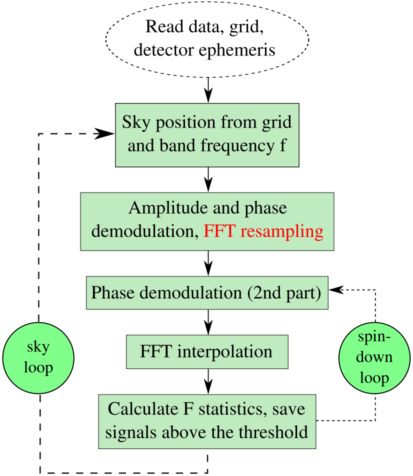
- ⋆ bandwidth 1Hz
- ⋆ sampling time 0.5 s
- ⋆ data length N = 344656 (two sideral days)

⋆ 4D grid: α, δ, f, $\dot{f}$ - sky positions, frequency and spindown

⋆ Uses the F-statistic defined in Jaranowski, Królak & Schutz (1998), serial code described and tested in Astone et al. (2010)

⋆ No. of F-statistic evaluations $\simeq f^3$

F-stat all-sky search description



Basically the whole loop over sky ($\alpha, \ \delta$) can be computed in parallel since the sky positions are independent of each other

The majority of computing is spent on

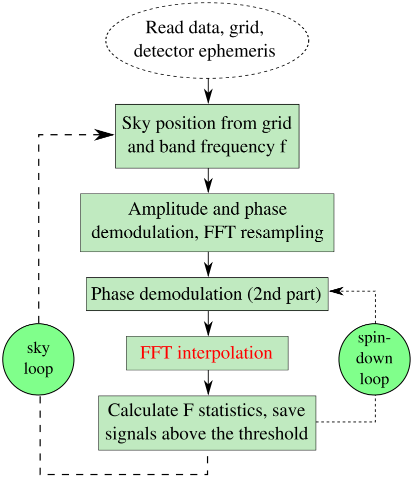
- ⋆ calculating the phase (trigonometric functions, $\gtrapprox$ 20%)
- ⋆ FFT ($\gtrapprox$ 70%)

Efficient FFT requires 2^N data points ($N_{data} = 344656 < 2^{19}$), padding with zeros to $N = 2^{19}$

Resampling

- ⋆ Resampling to barycentric time - FFT and inverse:
  - ⋆ nearest-neighbour ($\simeq$ 5% error),
  - ⋆ splines ($\simeq$ 0.1% error)

F-stat all-sky search description



The majority of computing is spent on

- ⋆ calculating the phase (trigonometric functions, $\gtrapprox$ 20%)
- ⋆ FFT ($\gtrapprox$ 70%)

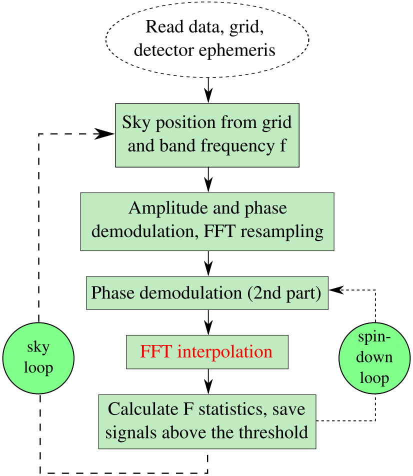
Efficient FFT requires 2^N data points ($N_{data} = 344656 < 2^{19}$), padding with zeros to $N = 2^{19}$

Interpolation

Grid coincides with Fourier frequencies - possible loss of signal (max. 36.3% when f is half way between the Fourier frequencies)

- ⋆ FFT (length N) & interbinning (max. $\simeq$ 13% error): DFT component in the middle of two Fourier frequencies approximated by $X((k + 1/2) \simeq (X(k+1) - X(k))/\sqrt{2}$
- ⋆ FFT zero-padding (length $2N$, max. $\simeq$ 10% error)

F-stat: parallelization strategy



How to do FFT with GPU:

- ⋆ use CUDA cuFFT library:
  - ⌣ well-optimized (Cooley-Tukey, Bluestein), 1D/2D/3D double precision complex/real transforms, multiple transforms, in- and out-of-place transforms,
  - ⌢ cannot launch many instances at the same time.
- ⋆ write custom kernel for FFT, launch concurrently.

⋆ cuSPARSE (sparse matrix routines)

Results

- Currently a sequence of kernels launched in a loop from CPU,
- Input data loaded to device once,
- Output transfered to host once per hemisphere.

Preliminary results: we obtain $\simeq \times 30$ speedup **with respect to the serial code**

The test case 42/271 ($f = 362$ Hz) results in 50 min w.r.t 24+ hours

- Intel(R) Core(TM) i5, 2.8GHz
- GPUs:
  - GeForce GTX 560 Ti
  - GeForce GTX 480

Things to improve

- `AtomicAdd` is slow (locking the memory until the operation is complete)
- Avoid algorithms with loops of the type

  ```
  for i in range(1,N):
      F(i) = F(i) + F(i-1)
  ```

  ("parallel splines"?)
- Tests using fast math, e.g., __sin() (when possible loss of precision is not important)
- Off-loading the results to host storage asynchronously
- Experiment with "production" cards (Tesla, Fermi, Kepler).

References

- P. Astone, K. M. Borkowski, P. Jaranowski, M. Piętka and A. Królak, PRD, **82**, 022005 (2010)
- `https://developer.nvidia.com/cuFFT`
- P. Jaranowski, A. Królak, and B. F. Schutz, PRD **58**, 063001 (1998).