Volume Title
*ASP Conference Series, Vol. **Volume Number***
Author
© ***Copyright Year** Astronomical Society of the Pacific*

Detection of binary pulsars with GPU-accelerated sinusoidal Hough transformations.

Christopher Laidler, Michelle M. Kuttel

Department of Computer Science, University of Cape Town, South Africa

Abstract. Analysis of relativistic binary pulsars is currently the best means by which to test theories of gravity in strong gravitational fields. Four-dimensional Hough transformations can detect sinusoids in noisy images. Hough transformations can by applied to Dynamic Power Spectra to detect the sinusoidal sift in observed spin frequency from binary pulsars in approximately circular orbits. We present four alternative GPU implementations of a Hough transformation algorithm, which we apply to synthesized Dynamic Power Spectra data to determine the GPU kernel that provides the best acceleration.

1. Introduction

Pulsars are highly magnetized rotating neutron stars that are observed from earth as periodic pulses across a range of frequencies. The ionized interstellar medium disperses the pulse signal over time: we observe a pulse at higher frequencies first. Searching survey data for pulsars with *a priori* unknown characteristics is known as blind searching. Standard blind search procedures scan the power spectrum (the Fourier transform of the de-dispersed survey data) and work well for periodic signals produced by solitary pulsars. However, if a pulsar is in a binary orbit, the orbital motion Doppler-shifts the observed pulse frequency, decreasing power and hindering detection.

Two methods are commonly used to search for binary pulsars. Acceleration searches assume that the orbital period is significantly longer than the observation period, allowing the simplifying assumption of a constant orbital acceleration during the observation. (Camilo et al. 2000; Ransom et al. 2002, 2001). Side-band searches are used when an observation covers several complete orbits. (Jouteux et al. 2002; Ransom et al. 2003). These two methods leave a sensitivity gap in detection of pulsars with orbital periods approximately equivalent to the duration of observation (a few hours), which is covered by the Dynamic Power Spectrum (DPS) search method. A DPS is generated by dividing the observation into short segments and calculating the power spectrum for each. These are stacked to produce a two-dimensional array in time and frequency. In a DPS, signals from pulsars in close to circular orbits appear as sinusoids. An automated search method that applies a sinusoidal Hough Transformation (HT) to a Dynamic Power Spectrum has been demonstrated to re-detect known binary pulsars in 47 Tucanae (Aulbert 2005, 2007). While this computationally intensive method is impractical for a truly blind search, there is potential for acceleration of the Hough transformation with commodity graphics processing units (GPUs). (Aulbert 2005). Here we use nVidia's Compute Unified Device Architecture (CUDA) to implement the sinusoidal

Hough Transformation on the GPU, with the hope that a significant acceleration will allow this transformation to be integrated into existing search pipelines.

2. The Hough Transformation

The Hough transformation is an image processing technique for estimation of the parameters of a parametric curve in a pixelated image. A standard Hough transform for detecting lines of the form $y = mx+c$ will convert an x, y point in image-space into a line in m, c-space. The intersection of two lines in parameter space represent the parameters of the line on which the two points in image space lie. With the Hough Transformation, this intersection is found by discretizing the parameter-space and counting the lines crossing a point in the discretized mc-space. This method is relativity insensitive to random noise and can detect features that are partially occluded or incomplete, making it a robust parameter detection method.

The polar form of the general equation of a horizontal sinusoid is given by:

$$y = a + p\cos(cx) - q\sin(cx) \tag{1}$$

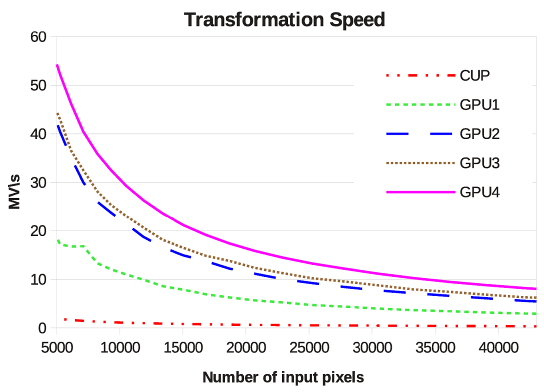
with $p = b\cos(d)$ and $q = b\sin(d)$. This is used as the master equation for the sinusoidal Hough transformation and has four adjustable parameters: a, c, p and q. The a parameter represents the "height" of the sinusoid or spin frequency of the pulsar (in Hz). The b parameter is the amplitude of the sinusoid and in the polar form is represented as the Euclidean distance to the $(0, 0)$ point in the $p\ q$ plain.

In general, a HT requires selecting input image pixels, calculating a hyperplane and incrementing all voxels that it intersects. Before delving into details, we make a number of observations and set conditions for an efficient implementation. For an input pixel with location (x, y) and for a given value of c, the transformation leads to a flat 3D plain in apq-space. If we stipulate that $p\ q$ and a must all have the same resolution, then for a given c, any combination of p and q will lead to a unique a. A complete 4D hyperplane can be generated by connecting the consecutive flat 3D planes. This is easily accomplished by widening in the a dimension.

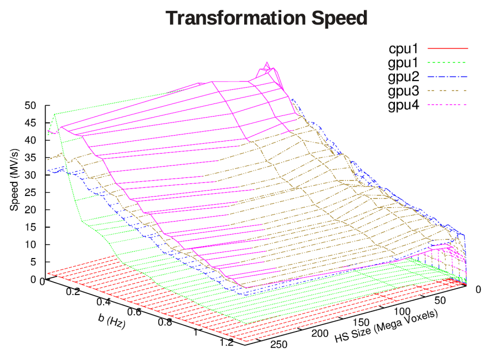
2.1. Implementations

We developed a serial and four parallel implementations to enable method comparisons. This Hough transformation requires many expensive trigonometric calculations, making it a complex problem to accelerate. The serial implementation uses intelligent decomposition and loop ordering and a large section of main memory to store previous calculations, to accelerate the transformation. This implementation first loops over x and then c, performing expensive trigonometric calculations as early as possible. We next loop over the polar coordinates and the subset of y which results in an a values within the bounds of the tile. To create a complete 4D manifold, we fill all points between a calculate a value and the a value calculated from the previous c, which is stored in main memory. Although not done here, this CPU version is easily parallelised for multiple cores with OpenMP, using an 'omp parallel for' directive on the initial loop.

We implemented four alternate CUDA kernels to perform the transformation, to evaluate different approaches. All kernels make use of CUDA atomic memory functions. Due to memory limitations, the GPU implementations cannot store large numbers of previously calculated values and thus duplicates some trigonometric calculations.



(a) Comparison of transformation speeds for the serial implementation (CPU) and four kernels (GPU1-4) on a 128x128x128x128 HS centered on $a = 250Hz$ $b = 0.5Hz$ $c = 0.000196$ $d = 1.445$



(b) The effects of tile size and amplitude parameter on transformation speed, for tiles centered on $a = 250Hz$ $c = 0.000196$ $d = 1.445$ and a rage of b values.

Figure 1.: Transform Speeds

This increased computation is compensated for by the large number of computational units.

The GPU1 kernel aims to minimize thread divergence by allocating a single input pixel to each CUDA thread. All the other kernels iterate only through the input pixels that may have an effect on the tile. This requires some divergence, but decreases the overall number of calculations required. The GPU2 kernel aims to reduce calculations by ignoring voxels that fall outside the tile. GPU2 dedicates a single CUDA thread to each p, q pair, each thread looping through x and c. The GPU3 kernel was implemented to test the use of SM to store a lookup buffer for y values. The final kernel GPU4 aims to use SM to remove duplication of trigonometric calculations, by allowing a thread to read the values calculated by the thread handling the adjacent c.

3. Results and Discussion

For testing, we generated synthetic DPS data, consisting of normally distributed noise with an added sinusoidal signal. While the transforms successfully detect the sinusoidal signal in the input images, we do not discuss the power of the transform here. We focus on the speed of the transformations, measured as the number of HS mega voxels filled per second (MV/s). All tests were performed on hypercube HS's tiles and run on an Intel Quad Core i7 930, 2.80 GHz, 16 GB of memory and a NVIDIA GeForce GTX 560 Ti with 2 GB of off-chip global memory.

Fig. 1a shows the transformation speeds for a sample tile, representing an average speedup of $9.73, 19.16, 20.66$ and 25.79 for GPU1, GPU2, GPU3, and GPU4 kernels respectively. Two key factors affecting transformation speed are the amplitude parameters covered by a tile and the tile size (Fig. 1b). The amplitude coved has two main effects firstly it influences the number of input pixels selected: lower b values lead to fewer points requiring transformation and thus faster filling of a tile of a given size.

The amplitude also affects the shape of the hyperplane: a large *b* will amplify changes in *c*, which can be thought of as increasing the separation of the 3D planes. Interestingly, each of the kernels has combinations of the two factors for which it outperforms the other kernels. For example, the GPU1 kernel, which allocates one thread per input pixel, is the fastest for values of *b* below 0.1 where there are few input pixels and planes are close to parallel, giving the low divergence of GPU1 an advantage. However, larger *b* values result in calculation of many *a* values that are are not in the HS and result in a rapid decrease in GPU1's performance.

For all transformation kernels except GPU2, larger tiles lead to higher speeds. GPU2 performs well when the tile dimensions (p and q) are close to the CUDA thread block dimensions, or low multiplies thereof. As threads are distributed across p, q pairs, large tiles result in a low number of thread blocks in which all threads are active. The use of SM in GPU3 to precalculate and store the lookup buffer does not significantly improve results, as the method used to find the first y value requires fewer cycles than the memory access required by the look-up method. For large tiles, GPU4 usually has the best performance, running 20 to 25 times faster than the serial CPU implementation. Here the SM is used to store the results of calculations involving complex trigonometric functions. With large tiles, there are enough active thread blocks to effectively hide shared memory access and synchronization. For all kernels, speeds flatten off once the hypercubes reach side lengths of ~64, at which point the GPU is fully utilized.

4. Conclusions

We have demonstrated that our parallel GPU implementations of the 4D sinusoidal Hough Transformation significantly reduce transformation time by 20 to 25 times. As a general rule, it is best to use as large a tile size as possible. With GPU implementations, best results are obtained with the use of SM to store complex calculations and intelligent loop ordering to allow selection of applicable input pixels. In future work, we will discuss the power of these transforms for automated blind searching of binary pulsars.

References

Aulbert, C. 2005, Ph.D. thesis, Max Planck Institute for Gravitational Physics, University of Potsdam, Germany

— 2007. `astro-ph/0701097`

Camilo, F., Lorimer, D. R., Freire, P., Lyne, A. G., & Manchester, R. N. 2000, Astrophys. J., 535, 975

Jouteux, S., Ramachandran, R., Stappers, B. W., Jonker, P. G., Van Der Klis, M., & van der Klis, M. 2002, Astron. Astrophys., 384, 532

Ransom, S. M., Eikenberry, S. S., & Middleditch, J. 2002, Astro J., 124, 1788

Ransom, S. M., Greenhill, L., Herrnstein, J., Manchester, R., Camilo, F., Eikenberry, S., & Lyne, A. 2001, Astrophys. J. Letters, 546, L25

Ransom, S. M. S., Cordes, J. M., & Eikenberry, S. S. 2003, Astrophys. J., 589, 911