Aoki Laboratory of Tohoku University and TohtoCTech decided to port Multi-View Stereo Revisited application to GPGPU with HMPP Workbench. We present the results of this experience…

Aoki Lab, Graduate School of Information Sciences of Tohoku University, focuses on the research and education of Computer Vision and their application to image/video signal processing. Aoki Lab has developed a novel image matching technique using Phase-Only Correlation (POC) - a technique for high-accuracy registration of 1D, 2D and 3D signals using phase information of discrete Fourier transform and applied the POC technique to a wide range of applications such as smart image sensors, super-resolution video signal processing, 3D machine vision, automotive image processing, biometrics authentication, medical image analysis, etc. In November 2009, Aoki Lab and TohtoCTech started a collaboration founded on the integration and the commercialization of research results of the latest image processing technology.

Introduction to Multi-View Stereo Reconstruction

The principle of multi-view stereo is to reconstruct a complete 3D object model from a large number of images taken from different camera viewpoints. This computer vision application can be used to reconstruct an organ in medical imaging for example.

Images are captured from different angles. 3D reconstruction operates with multi-baseline stereo matcher. Multi-view stereo method consists to find a cross point by gradually changing a depth map between the model and neighboring views. First, partial reconstructions are computed by taking neighbors images. The final 3D object model is obtained by merging all partial 3D object models.

This process is executable in parallel, based on each pixel unit of the modeled view. The application is adequate for GPGPU computation.

A first step was to realize an OpenMP version (8 threads) which achieved a speedup of 5 compared to the x86 sequential version on a core i7.

TohotCTech then concentrated its efforts on the main kernel which calculates an intermediate 3D model. And the calculations ran with a set of normalized images without the capture of those images. The given performances only take into account this kernel.

From this implementation, TohotCTech developed a GPU proof of concept of Multi-View Stereo Reconstruction using CUDA in order to port the application to GPGPU. The CUDA version achieved a speedup of 36 on Fermi architecture compared to the OpenMP version which proved that GPU are useful.
After this first experience, TohotCTech decided to port the same application with HMPP Workbench, a solution based on a set of directives that preserve legacy codes.

The first steps were to realize a diagnosis using profiling tools in order to find hotspots and to exhibit the application parallelism. Then application hotspots were pushed on GPU by using simple HMPP directives. In fact, only 4 directives were inserted in the sequential code:

- A codelet directive to identify the function to offload on GPU
- A callsite directive, to explicit a call to this specialized function in the application
- 2 parallel directives to indicate the loop nest

```c
// definition of the codelet
#pragma hmpp stereo codelet, target=CUDA, args[im_dep,im_corr,im_conf].io=inout
void Stereo(int rec_p1_x, int rec_p1_y, int rec_p2_x, int rec_p2_y, float inv_A[3][3],
            float d_step, int num, float th,
            unsigned char flRefImg[IMAGE_SIZE], unsigned char flCImg[10][IMAGE_SIZE], float d_max, float d_min,
            float dP_c[10][3][4], float im_dep[IMAGE_SIZE], float im_corr[IMAGE_SIZE], float im_conf[IMAGE_SIZE])
{
    ...
}
```

In this case, the amount of data is low compared to the amount of calculation. So, transfers are not critical.

This version achieved a speedup of 7.5 on Fermi architecture compared to the OpenMP version.

This initial phase was interesting and confirmed that this application can be ported to GPGPU with almost no extra coding in particular with HMPP. Even if the performance was not optimal with (the CUDA handmade version was about 4 to 5 times faster than the HMPP version), the result with HMPP was obtained in less than 1 week whereas CUDA version was developed in about 2 months.

From this basic but promising version, TohotCTech relied on CAPS experience and competencies to work on a fine tuned HMPP version to obtain, if possible, equal or better performance than CUDA handmade version.

**Algorithm Rethinking and Fine Tuned HMPP Version**

CAPS experts, while working on the GPU tuning phase, found that the algorithm could be changed in order to significantly improve the performance. Which performance can be improved gradually by adding more complex directives that are called HMPPCG for HMPP Codelet Generator. HMPPCG consists in a set of directives to tune the generated code and can be specific to a device.

After modification, the algorithm is memory bound (i.e. the limiting factor for a computation is the memory bandwidth of the card). The major part of the optimization consists in working on memory access by optimizing computations.

The first steps were to apply standards optimizations for CUDA Devices:

- Put parameters in constant memory;
- Suppress local arrays that generate access to Global Memory;
- Preload some arrays in shared memory;
- Extract the initializations of local variables;

Then, the next optimization consists in moving some computations outside of the loops where they were computed since they didn’t depend on the loop parameters.

Some optimizations were more particularly linked to the Fermi architecture. They consisted in fitting perfectly to the hit ratio of the cache without spilling. Finally, the resulting code obtained a hit ratio of about 95% in the cache memory.

**Results**

Environment is Windows 7, 64 bits, with HMPP 2.4 and CUDA 3.1. Tests were executed on a CPU corei7-930 and a GPU GeForce GTX 480 Fermi. The final version achieved a speedup of 120 on Fermi architecture compared to the OpenMP version.

The application is particularly adapted to GPU since its graphical aspect is a native feature of a GPU. Others characteristics (no critical transfers, float calculation, massively parallel code) permits to obtain impressive performances.

Today, calculations run with a set of images already normalized. A complete 3D reconstruction should consist in reconstruct an object in real time from images taken by a camera. Those images require an important normalized step – acquisition, (recalage), reorientation, brightness normalization – which becomes the compute intensive problem. Next step will be to port those steps on GPU.