GPU processors for space surveillance and airborne radars. Towards highly efficient ARM-GPU architecture for embedded radar systems.

J-F. DEGURSE[1,2], B. DUGROSPREZ[1] L. SAVY[1], S. MARCOS[2], J-Ph. MOLINIÉ[1]

[1]ONERA, Electromagnetism and Radar Department
Chemin de la Hunière, BP 80100, 91123 Palaiseau Cedex, France

[2]Laboratoire des Signaux et des Systèmes (Supélec-CNRS-Univ.Paris-Sud)
3, Rue Joliot-Curie, 91192 Gif sur Yvette, France

jean-francois.degurse@onera.fr,laurent.savy@onera.fr
sylvie.marcos@lss.supelec.fr, jean-philippe.molinie@onera.f

Abstract – The increasing complexity of radar systems and associated signal processing involves a significant increase of the computational workload. The challenge for ground-based radars is to reduce the cost of computing infrastructure whereas for airborne radars, the challenge is to implement intensive signal processing, such as STAP and SAR processing, while reducing hardware size and electrical consumption. In a concrete example of a space surveillance radar, we analyze all the signal processing modules running on CPU and on GPU, then we compare to the current computing system. For embedded applications, we present the advantages of a STAP algorithm running on GPU compared to a traditional general-purpose processor. We also show that using an original ARM processor-GPU configuration leads to an even higher efficiency than the classical x86-GPU architecture.

1 Introduction

The development of radar and signal processing techniques systems has led to an increasing number of operations to be performed by computers. In particular, the development of multi-sensor radars pave the way for new processing that bring important performance improvement. Microprocessors also evolved and are now able to run complex processing while reducing the cost, the consumption and the size of computing systems. However, algorithms have to be parallelized to enjoy high performance on these processors.

Indeed, parallel systems are becoming ubiquitous in world of computing as evidenced by the development of general purpose x86 multi-core processors (CPU), specialized heterogeneous processors like the POWER architecture, and more recently, massively parallel processors like the Graphics Processing Unit (GPU). Each of these parallel microprocessors requires a different level of programming parallelism to take advantage of the computing power they offer. CPU processors whilst evolving more and more towards a parallel architecture, by increasing the number of processing cores, are inherently designed to handle sequential operations. Having as main purpose the display of 3D scenes, either for videos-games or Computer-Aided Design applications, graphics processing units (GPUs) are massively parallel processors based on thousands of elementary units. In recent years, they have also been used to perform scientific computing. The increase of computational power at each new generation with the increase of the number of elementary processing units is much faster than general purpose processors. GPU processors offer a raw computing power from 10 to 20 times higher than CPU processors for an equivalent cost.

We show, in the first part, the interest of the GPU for a practical application of a space surveillance radar. The goal is to reduce the cost and maintenance of the signal processing system. In the second part, we evaluate the contribution of a GPU for STAP in airborne radar. The goal here is to improve the radar performance by increasing computing power available, while keeping a small-sized and a low power consumption system. We pursue this goal with an analysis of the performance of STAP processing on a an original ARM-GPU platform.

2 GPU for Space Surveillance Radar

2.1 Signal Processing

The GRAVES radar, developed by ONERA in 2005, is the French Space Surveillance System operated by the Air Force[**?**]. Its mission is to detect, track and maintain a database of the orbital settings of all satellites passing over the country at an altitude between 400km and 1000km. The radar consist of a continuous wave bi-static system operating in VHF frequency. The reception site is made of 100 receivers. The acquisition time for each signal part is about 1 second and every signal part contains $N = 16{,}384$ gross temporal samples. The signal is divided into three stages :

- Stage ***A*** is made of an Hilbert transform aiming to transform the analogical received signal into a complex signal. This is done by applying a FIR filter by convolution on the received signal : $y(t) = x(t) \odot h(t)$.
  Then, a direct path cancellation is performed in order to suppress the continuous component of the signal which is at the emission frequency of the radar. The filter is performed by subtraction of the mean value of the temporal signal :
  $y'(t_n) = y(t_n) - \frac{1}{N}\sum y(t_n)$. These two operations are made for all the 100 antenna channels.
- Stage ***B*** is dedicated to digital beamforming. For each temporal sample, several beamforming are made in K directions, K being greater than 1000. This digital beamforming is done by multiplying the data matrix $\mathbf{Y'}$ by a beamforming matrix $\mathbf{S}$:

$$\mathbf{Z} = \mathbf{S}\mathbf{Y'} = \begin{pmatrix} s_1^*(\theta_1,\phi_1) & \dots & s_{100}^*(\theta_1,\phi_1) \\ \vdots & \ddots & \vdots \\ s_1^*(\theta_K,\phi_K) & \dots & s_{100}^*(\theta_K,\phi_K) \end{pmatrix} \begin{pmatrix} y'_1(t_1) & \dots & y'_1(t_n) \\ y'_2(t_1) & \dots & y'_2(t_n) \\ \vdots & \ddots & \vdots \\ y'_{100}(t_1) & \dots & y'_{100}(t_n) \end{pmatrix} = \begin{pmatrix} z_1(t_1) & \dots & z_1(t_n) \\ \vdots & \dots & \vdots \\ \vdots & \ddots & \vdots \\ z_K(t_1) & \dots & z_K(t_n) \end{pmatrix} \quad (1)$$

  The result is a matrix $\mathbf{Z}$ containing the K beams of N temporal samples.
- Stage ***C*** has the bigger computing load. First, a compensation of the Doppler spreading due to the radial acceleration of the satellites during the acquisition time is performed. The matrix $\mathbf{Z}$ est multiplied by p Doppler hypothesis, with p being usually between 10 and 100 : $\mathbf{Z_1} = \mathbf{Z} \odot \mathbf{H_1}$,...,$\mathbf{Z_p} = \mathbf{Z} \odot \mathbf{Z_p}$ where $\odot$ is the Haddamard product (point to point) and we obtain p matrices $\mathbf{Z_i}$. The second part consist of transforming the temporal signals into frequency signals, i.e performing a Fourier Transform (FFT) on all the temporal samples for each beam of each matrix $\mathbf{Z_i} : \mathbf{Z_i}(\mathbf{t_n}) \Longrightarrow \mathbf{Z_i}(\mathbf{f_n})$.
  Finally, the last part of the processing is devoted to target detection. Spectral power is first estimated and thresholded. Then, a Minimax algorithm looks for the beams corresponding to a group of detection.

2.2 Performance on GPU

When the system was built, a powerful and expensive calculator made of more than 100 PowerPC (PPC) G4 processors has been assigned to the signal processing. The processors have been distributed as follows : 1 PPC to Stage ***A***, 28 PPC to Stage ***B*** and 91 PPC to Stage ***C***. Each Stage has to be processed in less than 1 second to meet the real time requirement of the radar. These algorithms being massively parallel, we can expect very good performance on GPU processors[**?**]. Indeed, for optimal performance, a GPU requires the operations to be partitioned into a very large number of elementary tasks, called threads. In contrast, a CPU processor or a multiprocessor CPU system will be work efficiently with much heavier threads, each computational core or processor being efficient on sequential operations.
We have implemented the signal processing into a C language code with CUDA extensions. We compare the performance of a GPU with those obtained on a multicore CPU general-purpose processor of the same generation and similar cost. To do this, we have also implemented these algorithms in pure C language using the high performance mathematical libraries *MKL* and *FFTw*.
The tested CPU is an AMD Opteron 6140 (2,6 GHz, 8 cores), and the test GPU is an NVIDIA Tesla C2075 (448 CUDA cores, 1,15 GHz). Computations are made in single precision (32bit). The results of execution time for all Stages ***A***, ***B*** and ***C*** are here compared to the performance of the current system made of PowerPC G4 (400 MHz) processors, respectively 1, 28 and 91 PPC.

Stage	CPU PowerPC (1/28/91PPC)	CPU x86 AMD Opteron 6140	GPU NVIDIA Tesla C2075
A	698 ms	170 ms	34 ms
B	153 ms	128 ms	21 ms
C	671 ms	2428 ms	250 ms

Table 1 : Execution time of all stages on three computing architectures

The execution times in Tab.1 point out an important increase of performance when using a GPU instead of a CPU. Speedup ratios are between 6x to 10x in favor of GPU. Moreover, a single GPU card runs faster than the entire current PPC-based system and is able to perform all stages in less than one second.

3 Airborne radar : detection of slow-moving targets

3.1 Space-Time Adaptive Processing (STAP)

Whereas conventional antenna processing exploit the space dimension of the received signals to filter received data, STAP (Space Time Adaptive Processing) processing jointly operate on the two dimension (space and time) of the received signals on a antenna array. This structure allows STAP processing to take advantage of specific space-time properties in the angle-frequency domain of the signal. Such a property can be used to filter ground echoes, also called ground clutter, received by an airborne radar. Indeed, the arrival angle and the Doppler frequency of the ground clutter are coupled, making it one-dimensional in the angle-frequency plan. STAP processing is used to detect low-speed targets that are not detectable by a simple Doppler analysis. The classical STAP detector, called Adaptive Matched Filter (AMF), is written as follows :

$$\mathbf{P}_{AMF} = \frac{|\mathbf{s_s^H \hat{R}^{-1} x_k}|^2}{\mathbf{s_s^H \hat{R}^{-1} s_s}} \underset{H_1}{\overset{H_0}{\lessgtr}} \eta \tag{2}$$

with $\mathbf{x_k}$ the data vector of range cell k, $\mathbf{\hat{R}_k}$ the covariance matrix of the range cell k, $\mathbf{s_s^H}$ the space-time steering vector and η the detection threshold.

A STAP processing can be split in 4 Stages :

- estimation of the covariance matrix and diagonal loading *diagonal loading*, for each range gate. Covariance $\mathbf{\hat{R}_k}$ is estimated on temporal samples : $\mathbf{\hat{R}_k} = \frac{1}{\mathbf{K_t}} \sum_{\mathbf{i=1}}^{\mathbf{K_t}} \mathbf{x_i x_i^H}$. The diagonal loading method *diagonal loading* prevent the degradation of filter performance due to a bad estimation of the covariance matrix : $\mathbf{\hat{R}_k} = \mathbf{\hat{R}_k} + \delta \mathbf{R_b}$ where $\mathbf{R_b}$ is the noise covariance matrix and δ is the loading factor, usually set to 1.
- covariance matrix inversion $\mathbf{\hat{R}_k}$ to get $\mathbf{\hat{R}_k^{-1}}$
- compuation of the filter $\mathbf{w_k} = \frac{\mathbf{\hat{R}_k^{-1} s_s}}{\mathbf{s_s^H \hat{R}_k^{-1} s_s}}$ for each range gate k
- filter application $\mathbf{y_k} = \mathbf{w_k^H x_k}$ and transformation to Doppler-distance image using FFT for target detection $\mathbf{y_k(t_n)} \Longrightarrow \mathbf{y_k(f_n)}$

This processing, or at least each one of these stages, has to be achieved in a lower time than the radar burst time. For a radar burst of M_p pulses and a pulse repetition frequency (PRF) f_r, this time is $T = \frac{M_p}{f_r} = M_p T_r$.

3.2 Algorithmic aspects and GPU performance

Unlike the previous case where the number of sensors and the number of temporal samples points were high, we here must deal with data from a dozen sensors and a hundred time samples. However, there are as many filters to estimate and to apply as distance cells ($>= 1000$) and this must be achieved in a much shorter time, typically between 10 to 100 milliseconds. The number of distance cells being important compared to the number of cores of a common CPU, it is very easy to parallelize this processing by assigning to each core a group of distance cells. CPU cores are as much efficient on heavy threads (sequential computation of N STAP filters STAP) as on lightweight threads (computation of one element of the STAP covariance matrix) unlike GPUs which perform well only on lightweight threads. The STAP processing had to be parallelized at an low level and and for each range cell. This was made by using specific algorithms for each Stages, for instance, the partial pivoting method for the inversion of the covariance matrix and by regrouping basic tasks of several range cells (or matrix) into blocks, which are sent to different GPU cores. We here simulate a nose radar antenna (AMSAR-like), made of 8 receiving channels. The number of range gate is $l = 1000$, the number of pulses snapshots is $M_p = 128$, the *PRF* is $f_r = 2kHz$ and the number of time taps chosen is $M = 8$. Computations are in double precision (64bit), the execution times for the different Stages are :

Stage	CPU x86 Intel Xeon X5570	GPU NVIDIA Tesla C2050
Covariance Matrix Estimation	131.23 ms	20.5 ms
Matrix Inversion	94 ms	36 ms
Filter computation	6.2 ms	4.1 ms
Filter application and FFT	5.3 ms	3.9 ms

Table 2 : Comparison between CPU and GPU on STAP operations

The GPU/CPU speedup is around 5-6x on the covariance matrix estimation and the matrix inversion which are the most demanding operations. On the two last stages, the speedup is between 1x and 2x, the compute load of these stages is too lightweight to fully exploit the GPU. In this particular application, the execution times on GPU are low enough to meet real-time requirements (total time must be less than 64ms).

3.3 STAP processing on efficient ARM-GPU architecture

When building a ground-based radar, the main motivation about the computing system is hardware cost. On embedded systems, we are looking to reduce power consumption and hardware size. In this context, the ARM-GPU (NVIDIA CARMA) architecture is very promising. CARMA is an architectural prototype for high performance, energy efficient hybrid computing. Today, power-efficient computing is driven by phones and tables. The ARM processors have an architectural and experience advantage, most power optimization work is being done for ARM. Due to their architecture, GPUs have an architectural efficiency advantage. A GPU is optimized for throughput and power efficiency whereas CPU is optimized for latency caches. For instance, a NVIDIA Fermi GPU has a thermal-to-compute power of 225 pJ/flop, an Intel Westmere CPU has 1700 pJ/flop. A GPU however is not able to launch an operating system kernel on his own, and has to be driven by a CPU. The goal of the CARMA prototype is to explore the efficiency and performance trade-offs for existing ARM+GPU hardware. The traditional x68 CPU found in all GPU systems is replaced by an ultra low power consumption host CPU Tegra T30 "Kal-El" having four ARM A9 cores with NEON and VFPv3 extensions, as found on smart-phones and tablets. The GPU is a NVIDIA Quadro 1000M, as found in many laptops. The Quadro 1000M has a theoretical compute power of 268 GFLOPS, while Tesla C2050 peaks at 1030 GFLOPS. The OS of the board has a Linux 3.1.10 kernel with enhancements to support Tegra features. The program however have to be compiled on an host x86 system as the current GPU compiler does not support ARM processors. As the Quadro 1000M module is limited in double precision, the tests have been made in single precision. We test here the two first stages of the STAP processing. Together, these two stages account for 90% of the workload.

Stage	CPU x86 Intel Xeon X5570	GPU NVIDIA Tesla C2050	NVIDIA CARMA
Covariance Matrix Estimation	72.67 ms	11.46 ms	62.41 ms
Matrix Inversion	54.40 ms	14.1 ms	84.1 ms

Table 3 : Execution time on CPU, GPU on x86 system and GPU on ARM CARMA board for STAP operations.

The results of Tab. 3 show than the CPU and the CARMA board have similar performance on this two stages. The Tesla C2050 GPU is 6-7x faster. We have also measured the electric consumption of the CARMA board and of the x86 CPU-GPU system.
Idle electric consumption of the x86 system is measured at $P = 146.5W$, and $P = 11.5W$ for CARMA. When the X5570 CPU is used at full load running a STAP processing, the consumption increases to $P = 223W$, when computation is made on the GPU $P = 348W$. On the CARMA board, when running STAP processing at full load, $P = 37W$. We then calculated the performance per Watt for the two heavyweight stages of the STAP processing. For Covariance Matrix Estimation, the total workload is $8lMNK_tMN$ whereas for matrix inversion workload can be approximated to $8l(MN)^3$. Then, the workload is divided by the execution time and the electrical consumption.

Stage	x86 system (CPU only)	x86 system (GPU)	NVIDIA CARMA
Covariance Matrix Estimation	0.24 GFLOPS/W	0.99 GFLOPS/W	1.71 GFLOPS/W
Matrix Inversion	0.17 GFLOPS/W	0.25 GFLOPS/W	0.68 GFLOPS/W

Table 4 : Performance per Watt of CPU, GPU on x86 system and GPU on ARM CARMA board for STAP operations.

Although not being the fastest solution, the CARMA platform is almost twice more efficient than the high performance GPU running on a x86 system. The GPU system has itself a higher performance per Watt compared to the same system without GPU. Note that the gain of the GPU architecture is higher on the Covariance Matrix Estimation stage than on Matrix Inversion stage. Indeed, this algorithm is highly parallel, hence it exploit efficiently the massively parallel architecture. The Matrix Inversion stage has some iterative parts and more cache memory access which lowers its performances on GPU. As predicted, this does not really affect the performance of the CPU.
Military commercial product such as GE-IP's GRA111 and GRA112 or Mercury's GSC6200 deliver high performance for signal processing applications, yet they are limited by the use of x86 architecture which places them at the same level of efficiency as our x86-GPU system.

Conclusion

With the fast-growing smart-phone and tablets market, ARM processors are to become more and more efficient. Coupled with a GPU, ARM-GPU platforms offer a high efficiency alternative to traditional x86-based systems. The next generation of GPU released in early 2013 offer a 3x increase of the performance per Watt. As the GPU module is a standard MXM module, it can be easily replaced by a GPU from this next generation. Works are ongoing to test the ARM processor for lightweight iterative operations such as post-STAP detection to validate the use of this architecture for practical embedded signal processing.