International Journal of Advances in Electronics Engineering
Vol:1 Issue:1 ISSN 2278 - 215X

Analyzing and optimizing technique used to optimize RADAR application CFAR function

Thejas G.S,
M.tech 4th sem, CS&E,
SSIT, Tumkur,
Karnataka, India.
thejasssit@gmail.com

Prof.K.Karunakara,
Professor & HOD of IS&E,
SSIT, Tumkur,
Karnataka, India.
karunakara_k@rediffmail.com

Dr.M.Siddappa,
Professor & HOD of CS&E,
SSIT, Tumkur,
Karnataka, India.
siddappa.p@gmail.com

***Abstract*—The paper discuss about CFAR function is one of the signal processing characteristics which performs detection of false alarm rate in cells. CFAR is one of the applications in Radar signal processing system. Paper discuss about the CFAR application and its characteristics and about the algorithm. Analyzing real-time Radar signal processing application CFAR (Constant False Alarm Rate) function in order to get the statistics of CPU utilizations in terms of instructions per clock cycles (IPC), for further improvement in the performance of the application CFAR function. For that making use of profiling tool AMD CodeAnalyst tool which profiles the applications running on system and gives statistics according to our requirement that's IPC.**

To optimize the application CFAR function required to choose one of the most applicable optimizing technique to CFAR function is called "OpenMP (Open Multi-Processing).OpenMP is an Application Program Interface (API) that may be used to explicitly direct *multi-threaded, shared memory parallelism* which uses fork-join model for parallel execution. And in further section discuss about the time consumed by CFAR function before applying OpenMP and after applying OpenMP.

***Keywords*: radar, CFAR (constant false alarm rate), CLASP (Configurable Linux bAsed Signal Processor), IPC (instructions per clock cycles), CPI (cycles per instructions), retired instructions, OpenMP, Code Analyst.**

I. INTRODUCTION

The conceptual basic radar system makes a target present or absence decision in each of the range-angle-Doppler frequency resolution cells. In a thermal noise-only environment, either an operator or an automatic decision device makes a target present decision when the input waveform envelope exceeds a prespecified threshold. The threshold is computed and depends upon the thermal noise power and the desired false alarm probability. However, when the interference environment consists of the combination of thermal noise and noise jamming and clutter, the output interference power is greater than that due to thermal noise alone. This is true even if adaptive processing is used to "cancel" the jamming and/or clutter. By keeping some constant threshold values to compare against the above said condition CFAR function has been designed.

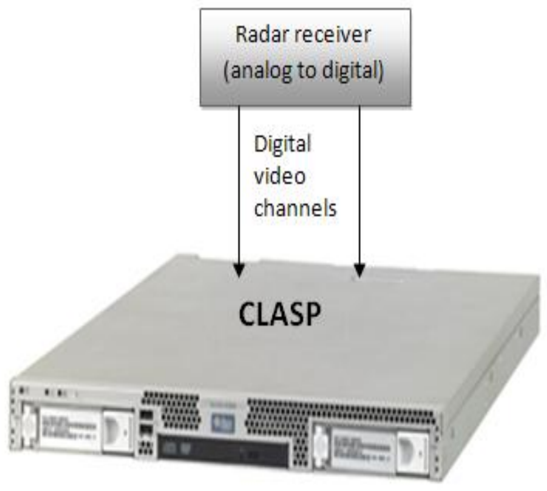

Fig 1: General block diagram of radar receiver and CLASP

Above figure depicts the way CLASP (Configurable Linux bAsed Signal Processor) gets digital video input from radar receiver. Where

CLASP is one of the part in radar system which consists of applications like Doppler filtering, pulse compression, CFAR function etc. CFAR application runs on CLASP. CFAR performs comparison of the cells based on number of filters for false alarm rate by keeping some constant threshold values.

II. CFAR CONCEPT

The basic concept of CFAR technique is that the voltage of a test cell is compared to the voltage of a set of reference cells. If the test cell voltage is “similar” to those of the reference cells, a target absent decision is made.

Figure below uses a range interval bracketing the test cell as the reference cells. Use of this set implies that the characteristic of the interference does not change over the range interval of the reference and test cells. This is true when the interference environment consists of the sum of thermal and constant power jamming noise. It is not true for environments that include ground clutter because the clutter cross section tends to change significantly in relatively small-range increments.

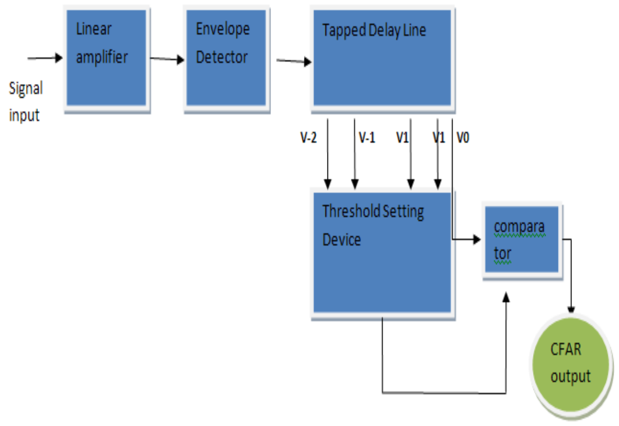

Fig 2: CFAR processor using range window for reference cells

A technique that is appropriate for ground clutter obtains the reference cells from previous scans of the resolution cell. This technique is called clutter map. The data from prior scans is used to estimate the clutter-plus-thermal noise power of every resolution cell. When a target, such as an aircraft, enters the cell, the increase in reflected power allows target detection in the presence of the clutter. often the clutter map is used with Doppler filter bank and a different map is determined for each filter of the bank.

III. CELL-AVERAGING CFAR

The threshold value T, the thermal noise power $P_{n,}$ and the design false alarm probability P_f when using square law envelop detector. The equation is also correct when the noise power is the sum of the thermal and jamming noise or the residue power after cancellation. Denote the power as P_r for either case. Taking the natural logarithm of both sides of this equation derives that the threshold equals the total interference noise power multiplied by constant, k1, which depends only up on the design false alarm probability. Denote the square law envelope detector output for the cell as q_0. Then a target present decision is made when

$$q_0 > T = k_1 P_r \qquad (1.1)$$

And the probability of detection for a specified false alarm probability is given by

$$P_D = P_f^{[1/(1+\gamma a)]} \qquad (1.1a)$$

Where γ_a is the average SNR for the swerling fluctuating target. Equation 1.1a can be solved for γ_a as

$$\gamma_a = \ln (P_f / P_D) / \ln (P_D) \qquad (1.1b)$$

When the total interference power is unknown, it seems reasonable to estimate the interference power as P_r and make target present decision when

$$q_0 > T_a = k_2 P_r \qquad (1.2)$$

Where T_a is an adaptive threshold and k_2 is a new multiplier constant that is usually not equal to k_1.

A common method to estimate the interference power is by averaging the output of the reference cells. For M reference cells, the estimate is

$$P_r = (\sum^{M}_{m=1} q_m) / M \qquad (1.3)$$

Where q_m denotes the square law detector output of the mth reference cell.

The reference cells are sometimes denoted as normalized cells because a variations of implementation for CFAR. Thus, (1.2) and (1.3) can be combined and arranged to equivalent target present/absent decision of

$$q_0 / (\sum^{M}_{m=1} q_m) / M > k_2 \qquad (1.4)$$

and in this form it can be seen that the test cell voltage is normalized by the average of the reference cell voltage and compared to the threshold multiplier constant.

All the above things have to be repeated for number of Doppler filter banks in each of the clutter map.

IV. ANALYZING CFAR FUNCTION USING PROFILER AND ANALYSIS

As the application runs on quad core AMD (Advanced Micro Devices) Opteron processor that's the CLASP hardware is composed of quad core AMD Opteron processor and Linux operating system. So to analyze the performance of CFAR function in terms of IPC can make use AMD CodeAnalyst performance analyzer which is developed by Advanced Micro Devices, Inc.

AMD CodeAnalyst tool is a type of profiler that can be used to determining the speed of a particular operation is often known as profiling. The term "profiling" can also be used when other information about an operation's profile is queried -- such as the number of calls to a function.

Profiling is used to determine which parts of a program to optimize for speed or memory usage. A general rule of thumb is that 90% of a program's time is spent in just 10% of the code. Profiling enables you to determine which 10% of the code.

So by AMD CodeAnalyst can able to get statistics in terms of IPC by using event based profiling where can select events CPU clock cycles and Retired instruction means successfully executed instruction. Can measure the efficiency of code by keeping CPU as subsystem.

Event select	**Event abbreviation**	**Event**
0x76	CPU_clocks	CPU clocks not halted
0xC0	Rest_instructions	Retired instructions

Table 3: Event selection

After getting the statistics formula used to calculate IPC & CPI are:

IPC = Ret_instructions / CPU_clocks

CPI = CPU_clocks / Re_instructions

The CPU run state affects (1) the CPU Clocks Not Halted event. Operating systems handle idling in one of two ways: (a) By temporarily halting the CPU until there is work to do, or (b) By executing an idle loop. (2) CPU clock cycles can also be consumed by repeated loops where each loops consists of several number of instructions that's reflected by number of retired instructions.

In the first case, (a) the CPU clock halts. CPU clock events are not counted and the resulting CPI and IPC figures directly reflect the behavior of the workload. (b) the CPU clock continues to run as the idle loop executes. CPI and IPC figures then include the effects of the idle loop. The idle loop usually has good CPI and IPC, which produces optimistic system-level values. The effects of the idle loop must be isolated and mitigated to state the correct CPI and IPC for the workload.

In second case, the workload is nothing but number of loops that has to execute number of instructions. Lets concentrate on this case as in CFAR algorithm the decision of present/absence of the target is compared and computer for each cells and it is repeated for number of Doppler filter banks in each of the clutter map. This is the area where optimization can be done as there is no dependency from one filter to another filter just checking of each cells has been done. Thus here the real workload is number of filters.

V. OPTIMIZING CFAR FUNCTION

To optimize CFAR, the number of instructions executed in each of the iterations depends up on number of filters which runs in serial order on Quad Core AMD Opteron processor. Hence number of instructions per clock cycles can be reduced by applying OpenMP technique. By applying on the iterations of filters which makes to run on processor instead in serial it enables to run parallel iterations.

Below figure shows the serial execution by single thread for example 12 filters.

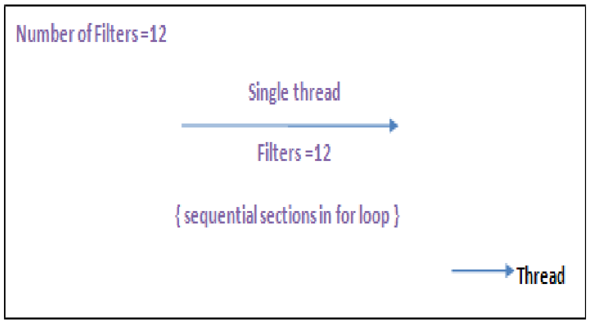

Fig 3: Serial execution by single thread

OpenMP is an Application Program Interface (API), jointly defined by a group of major computer hardware and software vendors. OpenMP provides a portable, scalable model for developers of shared memory parallel applications. The API supports C/C++ and Fortran on multiple architectures, including UNIX & Windows NT. OpenMP is an explicit (not automatic) programming model, offering the programmer full control over parallelization. OpenMP uses the fork-join model of parallel execution.

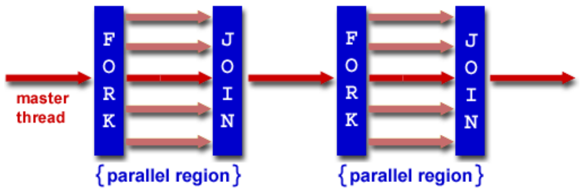

Fig 4: Fork & join model of OpenMP

OpenMP programs begin as a single process: the master thread. The master thread executes sequentially until the first parallel region construct is encountered.

FORK: the master thread then creates a team of parallel threads

The statements in the program that are enclosed by the parallel region construct are then executed in parallel among the various team threads

JOIN: When the team threads complete the statements in the parallel region construct, they synchronize and terminate, leaving only the master thread.

OpenMP is a technique which can be applied to for loops as the number of iterations is compared in for loops for different filter banks for each clutter maps. By applying parallel construct of OpenMP to number of filters bank it divides the serial execution by single thread into parallel execution by several threads. Thus reduces the time consumption of each serial iteration's and increases the IPC and CPI.

This can be achieved by dividing the filters into threads for this care to be taken while assigning number of threads which may lead to deadlock or race condition. So prior analysis have to be done to know suitable number of threads that are applicable. As OpenMP is specific to hardware that's processor the number of threads depends up on processor cores.

This can be analyzed by using OpenMP environmental variable OMP_NUM_THREADS = number of threads. So that can check the exceeding condition and preceding condition in terms of performance metric chosen .i.e. IPC, CPI, total time taken to run the application. This type of scheduling threads is called "static scheduling".

As processor consist of 4 cores so that it can serve at a time for 4 number of threads if number of threads a more than 4 means remaining threads need to wait for CPU which again degrades performance. And also proper care has to be taken while scheduling threads if number threads are more than 4 threads it may lead to a deadlock condition or race condition. Below figure shows parallel execution by 4 threads for example 12 filters.

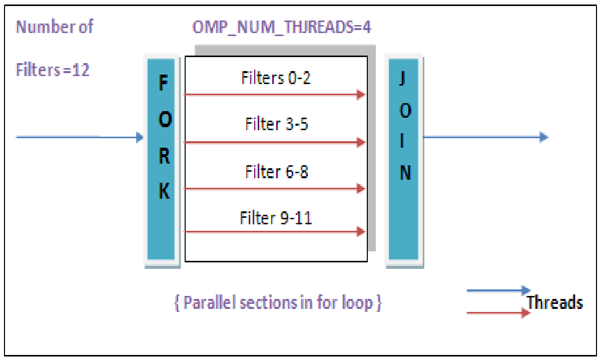

Fig 5: Parallel execution by several threads

That's in each parallel section for 12 filters it divides 3 filter per each threads which are executed in 4 parallel section thus the time taken to run CFAR function on Quad Core AMD Opteron processor can be reduced and also the number of retired instruction will be more per clock cycles thus we can see increase in IPC and decrease in CPI. This can be analyzed by AMD CodeAnalyst tool by profiling application CFAR function for subsystem CPU.

If proper scheduling and assignment of applicable threads not done means it leads to performance degrade that's in terms of time taken to run the application by processor.

VI. CONCLUSION

CLASP is one of the parts of radar signal processing system which consist of several functionalities. CFAR function is one of the characteristic in CLASP which performs detecting false alarm rate.

CLASP composed of Quad Core AMD Opteron processor and Linux operating system. CFAR function can be fitted optimally on processor by increasing its performance specific to processor.

It's possible to analyze and to get statistic of CFAR function by profiler AMD CodeAnalyst performance analyzer which analyzes performance in terms of IPC and CPI by keeping CPU as subsystem.

As the workload is to reduce number of instructions executed per clock cycle and the total time consumption of CFAR function which depends upon number of filter banks can be scheduled in parallel execution by applying OpenMP construct to the number of filters.

Proper scheduling and assigning applicable number of threads needed for parallel executions of filter iterations that results into increased performance in terms of higher IPC and lower CPI and time taken.

VII. ACKNOWLEDGMENT

I am very thank full to my HOD Dr. M. Siddappa, SSIT, and my internal guide Prof.K.Karunakara, SSIT and also my external guide Jayabrata Chakrabarty, Scientist 'D', 'E' Radar, DRDO/LRDE, Bengaluru, for their guidance in my project work and also their guidance for preparing this paper.

REFERENCES

[1] Basic Performance Measurements for AMD Athlon™ 64, AMD Opteron™ and AMD Phenom™ Processors, Paul J. Drongowski. AMD CodeAnalyst™ Performance Analyzer Development Team Advanced Micro Devices, Inc. Boston Design Center, 25 September 2008.

[2]M.I. Skolnik, Introduction to radar systems, Boston: McGraw Hill, 2001.

[3] OpenMP Specification: http://www.openmp.org*

[4] James Reinders, Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism. O'Reilly Media, Inc. Sebastopol, CA, 2007.

[5] AMD CodeAnalyst Performance analyzer http://developer.amd.com/cpu/codeanalyst/Pages/default.aspx

[6] "Introduction to OpenMP", Kyle T. Mandli, Department of Applied Mathematics, University of Washington, HPSC Seminar 2009.