Computational performance of a Cloud GNSS receiver using multi-thread parallelization

Vicente Lucas-Sabola[1], Gonzalo Seco-Granados[1], José A. López-Salcedo [1]
José A. García-Molina[2,3], Massimo Crisci[2]

[1]Universitat Autònoma de Barcelona (UAB), Bellaterra, Spain
[2]European Space Agency (ESA), Noordwijk, The Netherlands
[3]HE Space, The Netherlands

Email: vicente.lucas@e-campus.uab.cat, {gonzalo.seco, jose.salcedo}@uab.es
{jose.antonio.garcia.molina, massimo.crisci}@esa.int

***Abstract*—The proliferation of GNSS-based (Global Navigation Satellite Systems) services and applications providing ubiquitous, seamless and secure/reliable positioning is driving the use of high-performance devices. This implies the requirement of a higher computational capability in miniaturized size and low power consumption devices such as smartphones or Smart-City sensors. In this context, a possible alternative is to carry out the computational tasks outside the device, making use of the scalable, secure and nearly unlimited resources of Cloud infrastructure. This work presents the implementation of a Cloud-based GNSS receiver by taking advantage of one of the available Cloud infrastructures, such as Amazon Web Services (AWS). We will provide a review of the most relevant features of AWS for GNSS signal processing, as a case of study for stimulating the use of Cloud infrastructures within the GNSS community, while paving the way for the development of next-generation GNSS receivers. Furthermore, parallel computing will be studied to improve the Cloud GNSS receiver performance.**

I. Introduction

The deployment of new GNSS-based applications and services providing seamless, ubiquitous and above all secure positioning, will force next-generation GNSS receivers to implement much more complex processing tasks than those currently implemented in existing receivers. However, the future trend in these computing devices will tend to be miniaturized and with low power consumption as in the case of Internet of Things (IoT), Smart City and Machine-to-Machine (M2M) applications. The negative aspect of this trend is that receivers will suffer from limited computational capabilities and stringent power consumption limitations. Hence, devices will experience serious troubles for fully implementing the required signal processing tasks that next-generation GNSS services will demand. On the other hand, in the coming years new GNSS will become fully operational apart from GPS, such as Galileo, Glonass and Beidou. Thus, users may have more than 40 visible GNSS at a time, that will help to solve many of the current matters such as urban environmental problems (e.g. NLOS, multipath) [1]. The consequences of having so many visible satellites at the same time is that the receiver will have to process an overwhelming amount of data.

To solve the arising GNSS receiver problems, this paper elaborates on the Cloud GNSS paradigm, see Fig 1 [2]. This concept facilitates the implementation of sophisticated GNSS signal processing techniques using miniaturized and low power consumption devices migrating the processing tasks to Cloud servers, thanks to the scalability, reduced cost and computing high-performance they offer. Doing so, GNSS receivers will only need to collect the RF samples, generate a GNSS raw samples file and send it to the Cloud infrastructure, where the Cloud GNSS receiver will carry out the computing tasks. Furthermore, any upgrades on the GNSS receiver would be easily and equivalently applied to all the user terminals at the same time since they only need to be applied at the Cloud side and not in the terminal itself. Lastly, the Cloud GNSS receiver opens up the door for potential and innovative applications that cannot be carried out by current GNSS receivers and make them suitable for the implementation using a cloud-based infrastructure (e.g. authenticated services, crowdsourcing or massive data analysis) [3].

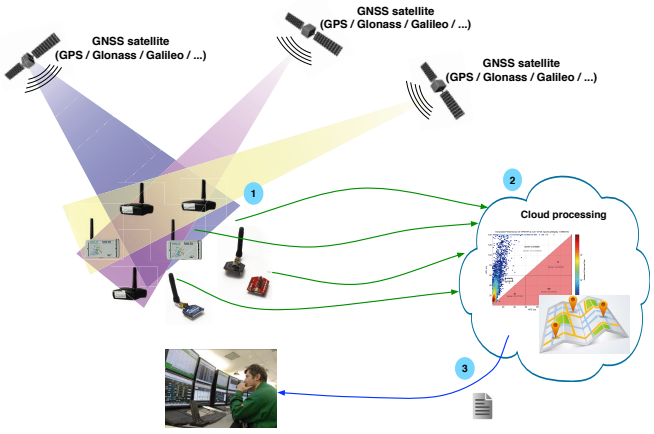

Fig. 1. Application of the Cloud GNSS receiver to GNSS sensors networks.

Nonetheless, the uplink communication between the RF front-end and the Cloud GNSS receiver can turn into a

978-1-5090-3885-5/16/$31.00 ©2016 IEEE

significant constraint due the quantity and size of packets, i.e. GNSS raw samples files, to be transferred. In this context, new 3GPP standards have specifically been developed for IoT technologies [4] with the mission of supplying the required downlink/uplink bandwidth to miniaturized wireless networks with low power consumption.

The aim of the present work is to illustrate how parallel computing can improve the performance of a software GNSS receiver such as the one implemented here within the Cloud GNSS receiver platform. An introduction to the Cloud computing concept and the use of AWS as Cloud computing platform will be made in Section II. It will focus on Elastic Compute Cloud (EC2) instances, which are suited for Cloud GNSS signal processing. Then, the memory requirements of the Cloud GNSS receiver will be explained in Section III. Next, a study of using parallel computing in GNSS signal processing, and in particular, in the Cloud GNSS receiver is performed in Section IV, with the mission of improving the performance, i.e. execution time, of the software receiver. It will also be studied how using parallel computing reduces the cost of using the EC2 service. Finally, conclusions will be drawn in Section VI.

II. Cloud computing platform

The *Cloud Computing* concept refers to the on-demand delivery of IT resources and applications via the Internet with pay-as-you-go pricing. This means that users can use the services provided by the Cloud without worrying about the resources they are using because resources are virtually unlimited. Cloud computing is characterized by the large scalability, security, liability, monitoring and cost. From a high-level perspective, the Cloud infrastructure can be divided into two parts: front-end and back-end. The front-end is the side that is visible to the user and includes the user terminal (e.g. computer, smartphone), whereby the user can interact with the Cloud platform. On the other hand, the back-end is the side that is not visible to the user and it comprises all the resources required to run and deliver the service to the user. The back-end is commonly known as the Cloud part of the Cloud computing architecture.

Due to the increasing number of emerging applications and services requiring heavy processing tasks, several commercial Cloud platforms have appeared in the past years such as AWS, Google Cloud Platform, Microsoft Azure, Rackspace, Oracle Cloud, or RedHat Openshift, providing services to software-developers and standard users as well. Because of the multiple services specifications offered by each of these platform, it can turn into an overwhelming amount of information for those facing for the first time the Cloud ecosystem.

In this context, the goal of this paper is to shed some light on the operation of a specific Cloud platform, to review the main services and functionalities being offered, and to discuss some of the configuration options that are available. In particular, by having in mind the specific application to GNSS sensor networks, as depicted in Fig. 1. The decision of selecting one Cloud platform in front of another is out of the scope of this work, since it may depend on some other non-technical issues not being addressed herein (e.g. the existence of some previous in-house experience with some of the already existing Cloud providers).

A. AWS services for Cloud-based GNSS applications

Amazon Web Services offers a wide range of services for the implementation of Cloud infrastructures. Some of the services of interest [5] for Cloud-based positioning applications and in particular for implementing a Cloud-based GNSS receiver are reviewed as follows:

- *Amazon Elastic Compute Cloud (EC2)*: virtual computing environment which allows the user to launch instances providing re-sizable and scalable computing capacity. Instances can be launched with different operating systems and loading them with custom application environment (e.g. snapshots of ohter instances volumes). The concept of instance is defined as a Virtual Machine (VM) which acts as a physical computing machine. In II-B a review of some of the EC2 instances type is provided.
- *Amazon Simple Storage Service (S3)*: secure, durable, and high-scalable Cloud storage. Data can be stored as resources called buckets with a maximum size of 5 terabytes, which can be stored, written, read and deleted.
- *Amazon Simple Queue Service (SQS)*: reliable, simple, scalable, secure, fast and inexpensive message queuing service. SQS makes simple and cost-effective to decouple the components of a Cloud application.
- *Amazon Elastic Block Storage (EBS)*: provides persistent block level storage volumes for use with Amazon EC2 instances in the AWS Cloud. That is to say, storage volumes can be created (e.g. snapshot) and then attached to instances. Snapshots can be used to instantiate multiple new volumes, expand the size of a volume, or move volumes across Availability Zones.

A deeper review of using of these services in GNSS signal processing can be found in [2]. Furthermore, Amazon services are hosted in multiple locations world-wide composed of regions and Availability Zones (AZ). Regions, which has multiple isolated location, i.e. Availability Zones, are designed to be completely isolated from others, achieving better fault tolerance and stability. AWS provides the ability to place resources in multiple locations. Thus, in case of a service failure in a specific Availability Zone, another AZ can handle the corresponding request.

B. AWS EC2 instances for Cloud-based GNSS applications

Amazon Elastic Compute Cloud (EC2) provides a large selection of instance types comprising different combinations of CPU, memory, storage and network capacity. This feature is quite convenient for GNSS signal processing purposes due the possibility of launching a specific instance type depending on the type of signal processing is going to be carried out (e.g. intensive computation, heavy memory storage).

Amazon EC2 instances are divided in two main types, i.e. Fixed Performance Instances (e.g. C3, M3, R3) and

978-1-5090-3885-5/16/$31.00 ©2016 IEEE

TABLE I
INSTANCE TYPE SPECIFICATIONS

Model	vCPU	Memory (GB)	Storage (GB)	Physical processors	Clock speed (GHz)	Price ($/hour)
t2.nano	1	0.5	EBS Only	Intel Xeon family	up to 3.3	0.0075
t2.micro	1	1	EBS Only	Intel Xeon family	up to 3.3	0.015
t2.small	1	2	EBS Only	Intel Xeon family	up to 3.3	0.03
t2.medim	2	4	EBS Only	Intel Xeon family	up to 3.3	0.06
t2.large	2	8	EBS Only	Intel Xeon family	up to 3.3	0.12
m4.large	2	8	EBS Only	Intel Xeon E5-2676 v3	2.4	0.143
m4.xlarge	4	16	EBS Only	Intel Xeon E5-2676 v3	2.4	0.285
m4.2xlarge	8	32	EBS Only	Intel Xeon E5-2676 v3	2.4	0.57
m4.4xlarge	16	64	EBS Only	Intel Xeon E5-2676 v3	2.4	1.14
m4.10xlarge	40	160	EBS Only	Intel Xeon E5-2676 v3	2.4	2.85
m3.medium	1	3.75	1x4 SSD	Intel Xeon E5-2670 v2	2.5	0.079
m3.large	2	7.5	1x32 SSD	Intel Xeon E5-2670 v2	2.5	0.158
m3.xlarge	4	15	2x40 SSD	Intel Xeon E5-2670 v2	2.5	0.315
m3.2xlarge	8	30	2x80 SSD	Intel Xeon E5-2670 v2	2.5	0.632
c4.large	2	3.75	EBS Only	Intel Xeon E5-2666 v3	2.9	0.134
c4.xlarge	4	7.8	EBS Only	Intel Xeon E5-2666 v3	2.9	0.267
c4.2xlarge	8	15	EBS Only	Intel Xeon E5-2666 v3	2.9	0.534
c4.4xlarge	16	30	EBS Only	Intel Xeon E5-2666 v3	2.9	1.069
c4.8xlarge	36	60	EBS Only	Intel Xeon E5-2666 v3	2.9	2.138
c3.large	2	3.75	2x16 SSD	Intel Xeon E5-2680 v2	2.8	0.129
c3.xlarge	4	7.5	2x40 SSD	Intel Xeon E5-2680 v2	2.8	0.258
c3.2xlarge	8	15	2x80 SSD	Intel Xeon E5-2680 v2	2.8	0.516
c3.4xlarge	16	30	2x160 SSD	Intel Xeon E5-2680 v2	2.8	1.032
c3.8xlarge	32	60	2x320 SSD	Intel Xeon E5-2680 v2	2.8	2.064
r3.large	2	16	1x32 SSD	Intel Xeon E5-2670 v2	2.5	0.2
r3.xlarge	4	30.5	1x80 SSD	Intel Xeon E5-2670 v2	2.5	0.4
r3.2xlarge	8	61	1x160 SSD	Intel Xeon E5-2670 v2	2.5	0.8
r3.4xlarge	16	122	1x320 SSD	Intel Xeon E5-2670 v2	2.5	1.6
r3.8xlarge	32	224	2x320 SSD	Intel Xeon E5-2670 v2	2.5	3.201

Burstable Performance Instances (e.g. T2). The former are suited for applications that need a consistent and intensive CPU performance (e.g. GNSS applications or services). On the other hand, burstable insances provide a baseline level of CPU performance with the ability to burst above the baseline. T2 instances are for workloads that do not use the full CPU often or consistently, but occasionally need to burst. Thus, T2 are best suited for applications such as web servers (e.g. front-end), databases, etc.

A table with the specifications of instance types previously stated is presented in Table I. It is important to mention that a vCPU is an instruction stream within a CPU core, and several of these streams (also referred to as threads) can be executed at the same time in the same core. In that sense, a vCPU can be understood as a logical core. Current CPUs, as some of EC2 instance processors (e.g. c4, c3) can implement hyperthreading [6], meaning that each CPU core can work on up to two threads. Therefore, when using an instance with 8 vCPU, we are working with 8 threads or logical cores instead of 8 truly hardware cores. If we want to launch an instance with 8 hardware cores, we should better select an instance having 16 vCPU.

III. MEMORY REQUIREMENTS FOR HIGH-SENSITIVITY GNSS SIGNAL PROCESSING

A parameter of paramount importance in a snapshot-based software GNSS receiver is the required memory, i.e. RAM, to implement an execution. This is further aggravated when high-sensivity techniques are implemented, as it is the case considered herein, where long integration times must be implemented at the acquisition stage. Since the fine Doppler search depends on the inverse of the integration time, long integration times lead to a very fine frequency grid for each tentative primary code and secondary code alignment. The results of this search must all be stored in memory (i.e. in the so-called acquisition hypercube), before signal detection takes place. This certainly requires a very significant amount of memory, and must be accounted for in order to dimension the resources that will be needed by each execution of our GNSS software receiver.

In the case under study, the double-FFT (Fast Fourier Transform) algorithm [7] is used to compute the correlation hypercube. This algorithm optimally finds the frequency and code phase shift of the correlation peak to be acquired [8], including the alignment with the secondary code, at the expense of consuming most of the memory used by the GNSS software receiver. The hypercube is the time-frequency representation of the cross-correlation between the input signal and the local replica, whose dimensions are $N_{\mathrm{dFFT}} \times N_{scode} \times N_r \times N_{fbins}$ where: $N_{\mathrm{dFFT}} = 2 \cdot N_r$ is the number of points in the 2nd FFT of the double-FFT algorithm in which the 2 factor is used to obtain a finer frequency resolution through zero padding; $N_{scode} = T_{code} \cdot f_s$ is the number of samples per code; N_r is the coherent integration time, and N_{fbins} is the coarse frequency bins to be searched.

978-1-5090-3885-5/16/$31.00 ©2016 IEEE

The number of frequency bins to be searched is

$$N_{fbins} = \text{round}\left(\frac{F_{max}}{\Delta f}\right) \cdot 2 + 1 \tag{1}$$

with F_{max} as the maximum frequency error and Δf as the frequency resolution

$$\Delta f = \frac{f_s}{N_{\text{FFTx}}} \tag{2}$$

where N_{FFTx} is the FFT size for two code periods

$$N_{\text{FFTx}} = 2^{\lceil \log(2 \cdot N_{scode}) / \log(2) \rceil} \tag{3}$$

and N is a power of 2, thus allowing a very fast $O(N \log N)$ and efficient implementation of the FFT with "butterfly" computations [9].

Example: Let us determine the size of an hypercube for a coherent integration time of 20 ms and a frequency search range from -4000 Hz to 4000 Hz with a sampling frequency of 5 MHz. For the case of GPS-L1 C/A signals we have $N_r = 20$ and $T_{code} = 1$ ms. Therefore, the frequency resolution is $\Delta f = 305.18$ Hz and the number of frequency bins to be searched is $N_{fbins} = 27$. Then, the hypercube needs $40 \times 5000 \times 20 \times 27 = 108e{+}6$ positions of memory. In the Cloud GNSS receiver under consideration using 64 bits CPU, data is stored as double (8 bytes), meaning that the obtained hypercube would require 824 MB. The size of the hypercube can be diminished using assistance GNSS information, in which case $N_{fbin} = 1$, resulting on an hypercube of 30.52 MB. If, in addition, we only search 1 combination of the secondary code instead of 20 because we are reusing information from previous fixes, the hypercube size would be reduced to 78.13 KB. ■

The above equations define the size of the hypercube and thus the memory required by the GNSS software receiver. In order to simplify the understanding of those equations, the size of the hypercube can also be expressed in a canonical way as some function $f(\cdot)$ that depends on the parameters of the GNSS signal under analysis, the coherent integration time (i.e. N_r multiples of the code period) and the oversampling factor (i.e. N_{sc}). By doing so we have that,

$$\text{Memory} = f\left(T_{code}, R_c, N_{rSec}, N_r, N_{sc}\right). \tag{4}$$

Using the equations above, this function results in the following expression:

$$\text{Memory(GB)} = \frac{8}{1024^3}(2N_r)(T_{code}N_{sc}R_c)N_{rSec} \cdot \text{round}\left(\frac{F_{max}}{N_{sc}R_c}2^{\lceil \log(2T_{code}N_{sc}R_c)/\log(2) \rceil}\right)2 + 1 \tag{5}$$

where N_{rSec} is the length of the seconday code (e.g. in the previous example $N_{rSec} = N_r = 20$), T_{code} is the primary code period, R_c is the chip rate, F_{max} is the maximum frequency error and N_{sc} the number of samples per chip, also known as the oversampling factor. Therefore, the sampling frequency can be obtained as $F_s = N_{sc}R_c$.

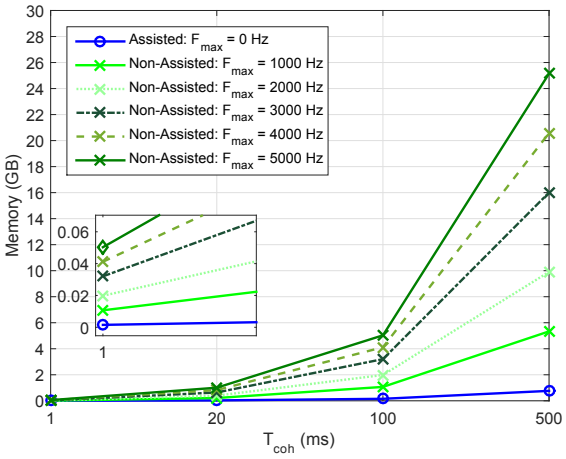

Fig. 2. GNSS software receiver memory requirements with didferent combinations of coherent integration times.

For the sake of clarity, in Fig. 2 are depicted the memory requirements of the GNSS software receiver under different conditions, i.e. coherent integration time and maximum frequency error, with $N_{sc} = 5$, $R_c = 1.023$ MHz, $T_{code} = 1$ ms and $N_{rSec} = 20$. We can observe that for a $T_{coh} = 20$ ms and $F_{max} = 0$ Hz or $F_{max} = 4000$ Hz we obtain the same hypercube size as in the previous example. Hence, with Fig. 2 we have an accurate picture of how the hypercube size can be reduced decreasing the number of frequency bins to be searched, the sampling frequency, the coherent integration time or the number of primary codes to be used. The reader should notice that an hypercube is generated for each of the satellites to be searched. Thus, if multiple satellites are searched using parallel computing, multiple hypercubes would be generated at the same time increasing the required RAM considerably.

IV. Parallel computing for GNSS signal processing

Users always want to experience low time-consumption when using any kind of software service. In this context, the aim of this section is to reflect the time-consuming improvement of the Cloud GNSS receiver [3] implementing parallel computing. As the name indicates, with parallel computing multiple processes can be carried out simultaneously thanks to the parallelization of the software. There are three types of parallelism: pipeline, data, and task parallelism [10]. As shown in [11], task parallelism (often implicit in *for* loops) is the better choice for maximizing the time-consuming improvement with software-defined GNSS receiver. In this manner, tests have been performed over two different cases: either satellite or snapshot parallelization. In the former case, the search of the satellites is done in parallel, allowing the acquisition of different satellites at the same time using different CPU threads. In the latter case, multiple snapshots are simultaneously processed using different CPU threads. The following tests have been performed solely in the acquisition stage without calculating the PVT.

978-1-5090-3885-5/16/$31.00 ©2016 IEEE

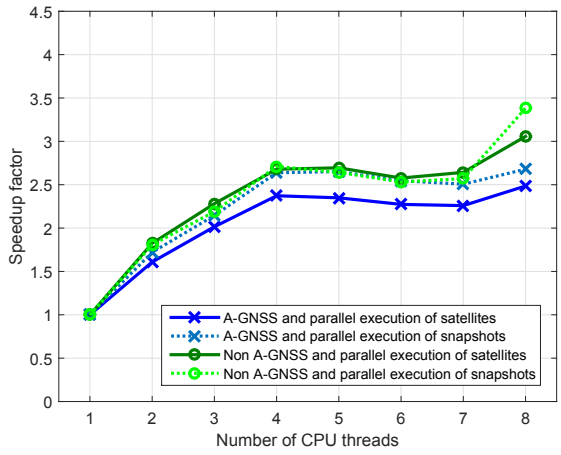

Fig. 3. Speedup factor under different number of threads in the hardware of Table II.

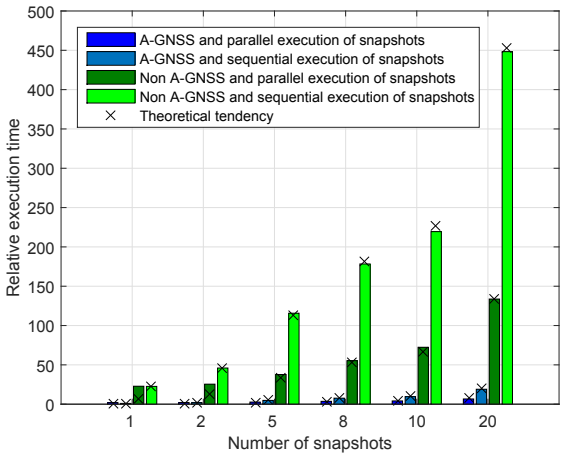

Fig. 4. Relative execution time with parallel and sequential execution of snapshots under different number of snapshots.

A. Speedup factor in parallel computing

In this section we will study how the time-saving changes as the number of threads increases. To do so, we use the hardware and raw GNSS samples file specifications of Table II. The execution is carried out searching 8 satellites in 8 snapshots. In order to compare the execution time between the sequential and parallel computing, we will use speedup factor as figure of merit, that can be calculated as

$$\text{Speedup} = \frac{\text{Sequential execution time}}{\text{Parallel execution time}} \tag{6}$$

In Fig. 3 the obtained results are depicted. We see as the number of working threads increases, also does the speedup factor. In particular, the parallelization of the snapshot execution offers a bigger speedup factor than the satellite parallelization. So, performing parallel computing and specifically, task parallelism, improves the performance of the Cloud GNSS receiver. Nonetheless, the speedup does not boost as expected: it is not proportional to the number of working threads. This is due the increase of communication time between threads as the number of working threads grows. Therefore, the execution time increases while the speedup factor decreases. Another significant behavior presented in Fig. 3 is that the speedup factor increases roughly linearly within the number of user threads until working with 4 threads, which is the number of CPU cores of the machine.

B. Improving performance with parallel computing

Next, we want to study how parallel computing improves the performance of the Cloud GNSS receiver. The specifications of the hardware and raw GNSS samples file used in this test are shown in Table II. As the parallelization is performed in *for* loops, i.e. task parallelism, the tested CPU allows the implementation of 8 *for* loops simultaneously (one for each thread). In addition, tests have been carried out with and without attaching assistance GNSS information. In the latter, the frequency search is performed from -4000 Hz to 4000 Hz (see Section III for a deeper explanation of the frequency search performed by the Cloud GNSS receiver). The theoretical tendency of the execution time is also depicted for each of the study cases, and can be expressed as

$$\text{Execution time} \approx T_{1snap-1thread}\,\frac{\#\text{Snapshots}}{\text{Speedup factor}} \tag{7}$$

where the execution time of 1 snapshots using 1 thread is divided by the speedup factor, which depends on the number of working threads (see Fig. 3), and multiplied by the number of snapshots to be processed. Therefore, (7) is used when implementing snapshot parallelism. It can be seen that as the speedup factor increases, the execution time increases more slowly than the number of snapshots to be executed, i.e. it is not proportional as in the sequential execution case. On the other hand, when performing satellite parallelism, the tendency is obtained as

$$\text{Execution time} \approx T_{1sat-1thread}\,\frac{\#\text{Satellites}}{\text{Speedup factor}} \tag{8}$$

where the execution time of 1 satellite using 1 thread is divided by the speedup factor and multiplied by the number of satellites to be searched. With satellite parallelization, the execution time increases more slowly than the number of satellites to be executed as the speedup factor increases.

In the first study case, the parallelization is carried out in the satellite execution, i.e. satellites are searched in parallel. Time measurements of parallel and sequential executions of 8 satellites under different number of snapshots have been made, see Fig. 5, whose results have been normalized at 4.14 seconds. There is a time-saving of roughly x6 to x24 as the number of snapshots increase if assistance GNSS information is used. On the other hand, the obtained time-saving when using task parallelism is about x3, depending on the use of assistance GNSS data.

978-1-5090-3885-5/16/$31.00 ©2016 IEEE

TABLE II
TESTED HARDWARE AND RAW GNSS SAMPLES FILE SPECIFICATIONS

Hardware specifications	
Processor	Intel Core i7-4702MQ
Number of cores	4
Number of threads	8
Base frequency	2.2 GHz
Cache	6 MB
Raw GNSS samples file specifications	
GNSS signal	GPS L1 C/A - Synthetic
Signal length	2 s
Sampling frequency	5 MHz
Quantization	16 bits
Visible satellites	8

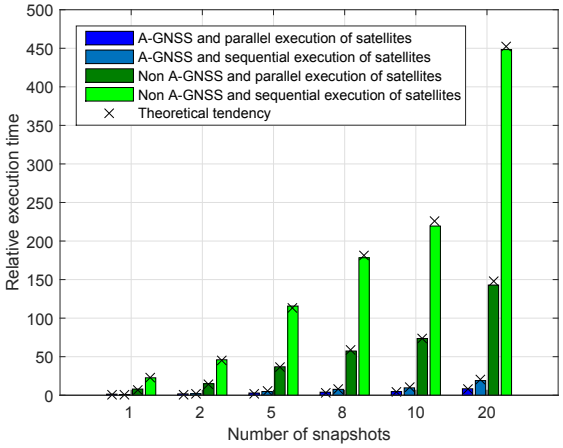

Fig. 5. Relative execution time of parallel and sequential execution of satellites under different number of snapshots.

Next, the parallelism is implemented in the snapshot execution (Fig. 4), i.e. snapshots are executed in parallel. The plot is normalized at 4.14 seconds. Similar results as the obtained in the first study case are achieved: roughly x6 to x24 if assistance GNSS information is used and approximately a x3 time-saving between sequential and parallel execution.

Then, a comparison of both study cases is shown in Fig. 6 (normalized at 5.84 seconds). It can be seen that as the number of snapshots to be executed increases, the snapshot parallelism offers a better time-saving. On the other hand, for low snapshot executions, satellite parallelism is faster.

In the third and forth case, the task parallelism is implemented for satellite and snapshot, Fig. 7 and Fig. 8 respectively, for the execution of 8 snapshots under different number of satellites. In both cases, there is a time-saving of approximately x5 to x23 if assistance GNSS data is attached, and a time-saving of x1.5 to x3 when using parallelization as the number of satellites to be executed increases.

Finally, in Fig. 9 is depicted the comparison between the satellite and snapshot parallelization under different number of satellites. The reader can notice that as the number of satellites to be searched remains low, the snapshot parallelism offers a higher time-saving. Nevertheless, as the number of satellites to be searched increases, also does the satellite parallelism speed.

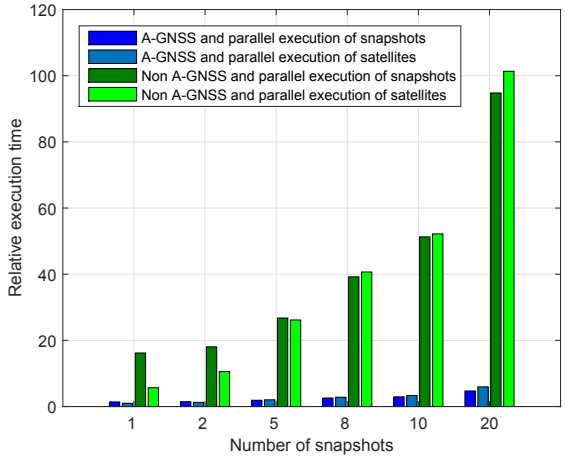

Fig. 6. Relative execution time with parallel execution of snapshots and satellites under different number of snapshots.

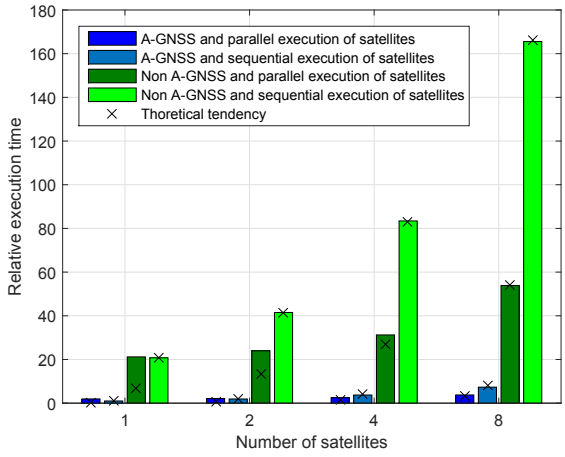

Fig. 7. Relative execution time with parallel and sequential execution of satellites under different number of satellites.

During this section we have studied how using parallel computing decreases the execution time when acquiring GNSS signals under different cases. The parallelization has been implemented both in satellite and snapshot execution, obtaining faster results in the latter case. Thus, the resource manager should be able to implement satellite or snapshot parallelism depending on the execution specifications. We also have seen how the obtained results of the above figures agrees with the theoretical tendency calculated with (7) and (8). Main differences are found when executing low number of snapshots or satellites, because in those cases not all threads are used (e.g. with 4 satellites, only 4 threads are used), and the speedup factor used to calculate the theoretical tendency is calculated when working with 8 threads.

C. Speedup factor in EC2 instances

In previous sections we have studied how the use of parallel computing improves the time-consuming factor of executions.

TABLE III

EC2 INSTANCE COST IMPROVEMENT - CONTINUOUSLY PROCESSING SNAPSHOTS OF GPS L1 C/A WITH $T_{coh} = 20$ ms, 16 BITS OF QUANTIZATION AND SAMPLING FREQUENCY OF 5MHz OR SPORADICALLY PROCESSING SNAPSHOTS UNDER A 1 GB/MONTH PLAN (LAST COLUMN)

Model	vCPU	Memory (GB)	Price per hour (EU-Frankfurt)	Number of threads	Number of snapshots	Parallelization	Max. number of snapshots per hour	Cost per snapshot	Monthly cost per sensor (1GB/month)
c4.2xlarge	8	15	$0.534	8	8	Execution per thread	5,339.27	$0.00010	$0.22
c4.2xlarge	8	15	$0.534	8	1	Satellite	4,311.38	$0.00012	$0.27
c4.2xlarge	8	15	$0.534	8	1	Snapshot	4,682.93	$0.00011	$0.25
c4.large	2	3,75	$0.135	1	1	No	1,071.43	$0.00013	$0.28
c3.2xlarge	8	15	$0.516	8	8	Execution per thread	3,559.95	$0.00014	$0.31
c3.2xlarge	8	15	$0.516	8	1	Satellite	3,945.21	$0.00013	$0.28
c3.2xlarge	8	15	$0.516	8	1	Snapshot	4,222.87	$0.00012	$0.26
c3.large	2	3,75	$0.129	1	1	No	831.41	$0.00016	$0.35

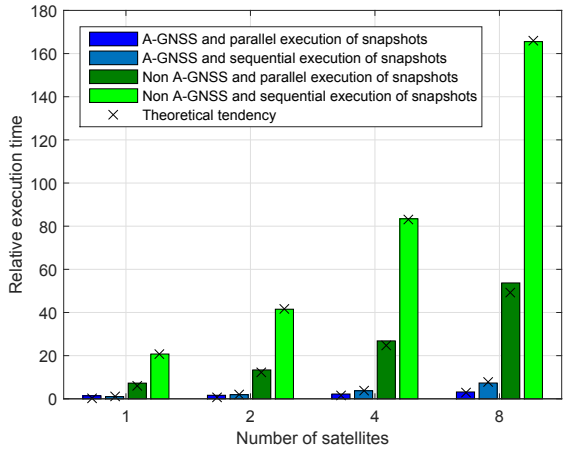

Fig. 8. Relative execution time with parallel and sequential execution of snapshots under different number of satellites.

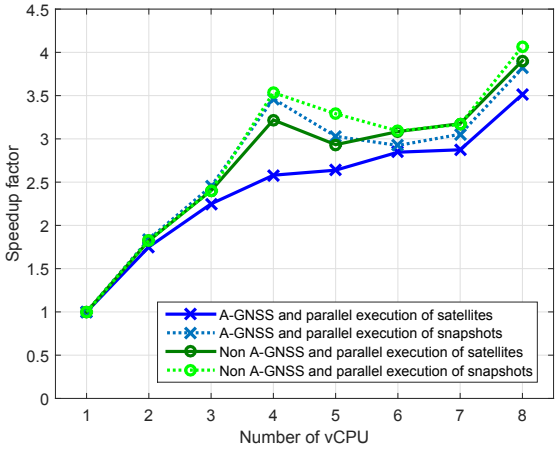

Fig. 10. Speedup factor under different number of threads in a c4.2xlarge instance.

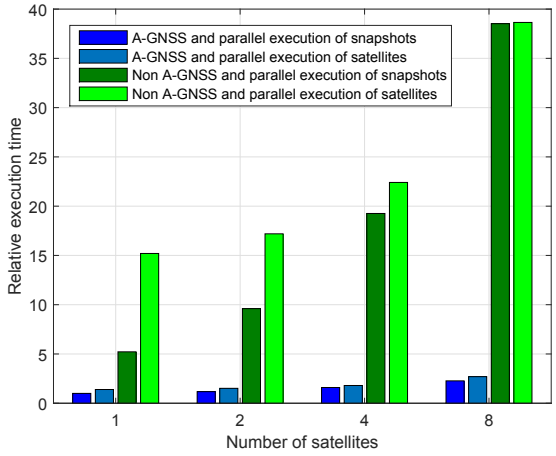

Fig. 9. Relative execution time with parallel execution of snapshots and satellites under different number of satellites.

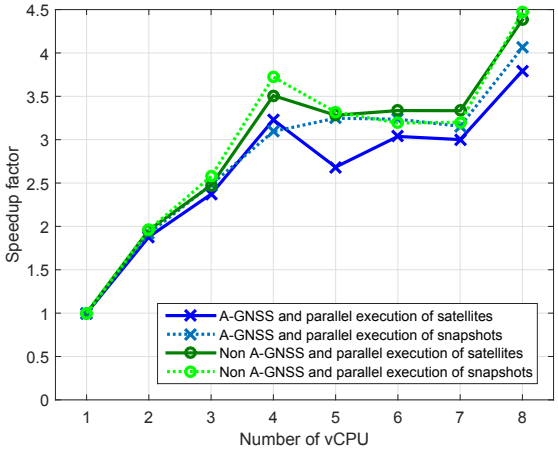

Fig. 11. Speedup factor under different number of threads in a c3.2xlarge instance.

In this section, it is presented the cost reduction of performing parallel computing in EC2 instances, that forms the backend bulk of the Cloud GNSS receiver. In particular, the speedup factor of the *c4.2xlarge* and *c3.2xlarge* instances (see Table I for specifications) is obtained and depicted in Fig. 10 and Fig. 11 respectively.

A higher speedup factor is achieved with *c3.2xlarge* instances due the use of SSD storage instead of EBS Only even

978-1-5090-3885-5/16/$31.00 ©2016 IEEE

with a physical processor with lower clock speed. Therefore, the speedup factor depends on all the EC2 instance resources, i.e. RAM, storage, processor. Nevertheless, the execution time is lower when using *c4.2xlarge* instances because of the higher CPU clock speed with regarding to *c3.2xlarge*.

V. Application to a GNSS sensors network

The presented EC2 service cost reduction has been performed in the context of a GNSS sensors network, as depicted in Fig. 1. In this framework, it is assumed that each sensor has an internet flat rate of 1GB/month, a usual rate in many cellular plans, and sends 3 packets of roughly 488 Kbyte per hour. Such packets include the raw GNSS samples file that will be executed and processed in the Cloud GNSS receiver and the Cloud GNSS receiver will make a search of 8 satellites in 1 snapshot for each incoming packet. Therefore, in this study case, each packet is a snapshot that will be processed and the PVT can be computed for each independent snapshot execution. So, taking advantage of parallel computing in EC2 instances produces a monthly cost-effective solution, as seen in Table III. A cost reduction up to 10.5% is achieved when using parallelization, i.e. working with 8 threads, instead of working with sequential processing, i.e. working with 1 thread, combined with a lower execution time with *c4.2xlarge* instances. In addition, a cost reduction of 27% with *c3.2xlarge* instances is obtained when using parallel computing. Thus, a cost and execution time reduction is reached with the use of the multiple vCPUs thanks to the Cloud GNSS software receiver parallelization.

Nevertheless, if the execution time is not a parameter of paramount importance, the user may prefer to exploit the EC2 instance resources implementing multiple executions at the same time, e.g. 8 different executions (which in this case an execution is equivalent to processing one snapshot) in parallel, one for each thread. In such case, a cost reduction of the 26% and 7% for *c4.2xlarge* and *c3.2xlarge* instances respectively is accomplished. In any case, the price per month of the EC2 service without performing parallelization is remarkably low: \$0.28 and \$0.35 for *c4.2xlarge* and *c3.2xlarge* instances respectively per sensor. Thus, the user must choose between a faster execution time or slower execution time but cheaper price per execution (or in this case, snapshot), as in the case of *c4.2xlarge* instances. The obtained differences of cost reduction between both instances is due the speedup factor they offer when implementing parallel computing. The obtained results are highly scalable varying the number of packets (or snapshots in this study case) per hour: $\text{Monthly cost per sensor} = \text{Cost per snapshot} \cdot N_{packets} \cdot 24 \cdot 30$. For a deeper understanding of how are packets generated and calculate their size, see [2].

VI. Conclusion

This paper has presented the use of AWS for the implementing a Cloud GNSS receiver. First of all, a study of the memory requirements of a high-sensitivity GNSS software receiver has been presented. With the obtained results, the use of assistance GNSS data becomes mandatory if long correlations need to be implemented. Next, we have observed that with the use of parallel computing, we can achieve a remarkable improvement in the execution time of high-sensitivity GNSS signal processing, thus compensating the increase in computational time due to the implementation of long correlations. Following this approach we have implemented two different task parallelisms: snapshot and satellite parallel execution. A larger reduction of the execution time has been achieved with snapshot rather than with satellite parallelization. In addition, we also have seen that the use of assistance GNSS information reduces both the required memory of the hypercube and the execution time. Finally, an example has been provided for the monthly cost of the EC2 services that are required by a Cloud GNSS receiver periodically processing snapshots of GPS L1 C/A signals using 20 ms integration time. The application was circumscribed to the context of GNSS sensors networks, and the resulting cost was found to be below \$1 using a fixed data plan with 1 GB per month. Therefore, the results clearly confirm the feasibility of implementing Cloud GNSS signal processing in real-life applications.

Acknowledgment

The views presented in this paper represent solely the opinion of the authors and not necessarily the view of ESA. This work was partly supported by the European Space Agency (ESA) under contract No. 4000113891/15/NL/HK and by the Spanish Government under grant TEC2014-53656-R.

References

[1] N. G. Ferrara, J. Nurmi, and E. S. Lohan, "Multi-GNSS analysis based on full contellations simulated data," in *Proc. International Conference on Localization and GNSS (ICL-GNSS)*, June 2016.

[2] V. Lucas-Sabola, G. Seco-Granados, J. A. López-Salcedo, J. García-Molina, and M. Crisci, "Demonstration of Cloud GNSS Signal Processing," in *Proc. ION GNSS)*, September 2016.

[3] ——, "Cloud GNSS receivers: new advanced applications made possible," in *Proc. International Conference on Localization and GNSS (ICL-GNSS)*, June 2016.

[4] D. Flore, "3GPP Standards for the Internet-of-Things," in *GSMA MIoT*, Feb 2016.

[5] "Amazon Web Services Cloud Products," Amazon Web Services, https://aws.amazon.com.

[6] D. Koufaty and D. T. Marr, "Hyperthreading technology in the netburst microarchitecture," *IEEE Micro*, vol. 23, no. 2, pp. 56–65, March 2003.

[7] D. Jiménez-Baños, N. Blanco-Delgado, G. López-Risueño, G. S. Granados, and A. Garcia-Rodriguez, "Innovative techniques for GPS indoor positioning using a snapshot receiver," in *Proc. ION International Technical Meeting of the Satellite Division, (ION GNSS)*, 2006.

[8] G. Seco-Granados, J. López-Salcedo, D. Jiménez-Baños, and G. López-Risueño, "Challenges in Indoor Global Navigation Satellite Systems: Unveiling its core features in signal processing," *IEEE Signal Processing Magazine*, vol. 29, no. 2, pp. 108–131, March 2012.

[9] V. O. Alan, W. S. Ronald, and R. John, "Discrete-time signal processing," *New Jersey, Prentice Hall Inc*, 1989.

[10] W. Thies, "Language and compiler support for stream programs," Ph.D. dissertation, Massachusetts Institute of Technology, 2009.

[11] T. E. Humphreys, J. Bhatti, T. Pany, B. Ledvina, and B. OHanlon, "Exploiting multicore technology in software-defined GNSS receivers," in *Proc. ION GNSS*, 2009, pp. 326–338.