Benchmarking CPUs and GPUs on Embedded Platforms for Software Receiver Usage

T. Pany, J. Dampf, W. Bär, J. Winkel, C. Stöber
IFEN GmbH, Alte Gruber Straße 6,
85586 Poing, Germany

K. Fürlinger
Ludwig-Maximilians-University (LMU) Munich, Computer Science Department, MNM Team, Oettingenstr. 67, 80538 Munich, Germany

P. Closas
Centre Tecnològic de Telecomunicacions de Catalunya, Communcation Systems, Parc Mediterrani de la Tecnologia (PMT) - Building B4
Av. Carl Friedrich Gauss 7, 08860 – Castelldefels, Spain

J. A. Garcia-Molina
ESA/ESTEC, Keplerlaan 1, 2201 AZ Noordwijk, The Netherlands

BIOGRAPHIES

Dr. Thomas Pany works for IFEN GmbH as a senior research engineer in the Receiver Technologies department. He also works as a lecturer (Priv.-Doz.) at the University FAF Munich and for the University of Applied Sciences in Graz. His research interests include GNSS receivers, GNSS/INS integration, signal processing and GNSS science.

Dr. Wolfgang Bär is head of Mobile Solutions department at IFEN GmbH since 2014. He joined IFEN in 2006 as a system engineer. Before, he has been research associate at the Institute for Geoinformatics and Remote Sensing of the University of Osnabrück receiving his PhD in 2007.

Jürgen Dampf received his M.Sc. in Aviation Engineering at the University of Applied Sciences in Graz in 2013. Since 2012 he is working for IFEN GmbH as a system engineer emphasizing on GNSS reflectometry, particle filters and embedded software receivers.

Dr. Jón Ó. Winkel is head of Receiver Technologies at IFEN GmbH since 2001. He studied physics at the universities in Hamburg and Regensburg. He received a PhD (Dr.-Ing.) from the University of the FAF in Munich in 2003 on GNSS modeling and simulations.

Carsten Stöber works for IFEN GmbH as a system engineer. He received his diploma in Geodesy at the Technical University in Berlin in 2005. For seven years he has been research associate at the Institute of Space Technology & Space Application at the University of the Federal Armed Forces in Munich.

Dr. Karl Fürlinger is a lecturer and senior researcher at the Ludwig-Maximilians-University (LMU) Munich, working in the area of parallel and high performance computing. His research focuses on performance tools and all aspects parallel programming, algorithms, and systems. Before joining LMU Munich he was a postdoctoral researcher at the University of California at Berkeley, and at the NERSC supercomputing center and prior to that he was a senior research associate at the University of Tennessee at Knoxville (UTK).

Dr. Pau Closas received the M.Sc. and Ph.D. in Electrical Engineering from the Universitat Politècnica de Catalunya (UPC) in 2003 and 2009, respectively. He also holds a M.Sc. degree in Advanced Mathematics and Mathematical Engineering from UPC since 2014. Currently he is Senior Researcher and Head of Department at the Centre Tecnològic de Telecomunicacions de Catalunya (CTTC) in Barcelona. His expertise is on statistical signal processing, with applications to GNSS receiver design, indoor positioning and wireless communications. He is the recipient of the EURASIP Best PhD Thesis Award 2014 and the ninth Duran Farell Award for Technology Research, both for his contributions in the areas of signal processing and GNSS.

J.A. Garcia-Molina is Radio Navigation engineer at ESA/ESTEC in Noordwijk, The Netherlands. His primary areas of interest include signal processing, estimation theory, GNSS receivers and signals and navigation applications.

ABSTRACT

Smartphones containing multi-core central processing units (CPUs) and powerful many-core graphics processing units (GPUs) bring supercomputing technology into your pocket (or into our embedded devices). This can be exploited to produce power-efficient, customized receivers with

flexible correlation schemes and more advanced positioning techniques. For example, promising techniques such as the Direct Position Estimation paradigm or usage of tracking solutions based on particle filtering, seem to be very appealing in challenging environments but are likewise computationally quite demanding. This article sheds some light onto recent embedded processor developments, benchmarks Fast Fourier Transform (FFT) and correlation algorithms on representative embedded platforms and relates the results to the use in GNSS software radios. The use of embedded CPUs for signal tracking seems to be straight forward, but more research is required to fully achieve the nominal peak performance of an embedded GPU for FFT computation. Also the electrical power consumption is measured in certain load levels.

INTRODUCTION

When building a GNSS receiver we have to accomplish three major tasks: detection of GNSS signals, tracking them, and using the obtained ranging information to compute the user position. In contrast to hardware GNSS receivers, a software GNSS receiver allows engineers to easily adapt the used algorithms and design principles to each application domain. For the above-mentioned three core tasks there are various characteristic numerical operations that must be performed. For GNSS signal acquisition, the use of the FFT is virtually inevitable. For signal tracking, the correlation of the received signal with internally generated replica signals has to be performed. This is realized either as a dot-product operation with multiply-and-add commands or as an exclusive-OR operation if 1-bit sampling is used. Positioning filters typically make use of various floating point operations to compute, for instance, satellite positions or update the navigation filter.

In mass-market receivers the first two tasks are accomplished by application specific integrated circuits (ASICs) and the user position is computed either on the ASIC or by a general purpose processor. Current ASIC technology is highly efficient and an ASIC-based GNSS receiver can solve the above-mentioned tasks consuming only milliwatts of power allowing to track the user position in the background, facilitating applications like geofencing, user motion detection in wearables etc [1]. However, the accuracy of those receivers is still in the order of several (dozens of) meters or worse, especially if operated in a mobile phone under degraded signal conditions [2]. Needless to say, mass market receivers have limited capabilities to be tailored for specific applications as the core algorithms are built into the silicon chip and positioning algorithms are intentionally kept simple to maintain the low power consumption.

Realizing all three receiver tasks in software, generally allows implementation of more flexible algorithms by a larger community of engineers and researchers. The time to build or adapt a so-called software receiver is significantly lower since, apart from the RF front-end, no dedicated hardware development is involved. Indeed nowadays software GNSS receivers are the gold standard in research and development (R&D) especially when developing new algorithms, testing new navigation signals or fusing of GNSS with other sensors. R&D software receivers typically run on a conventional desktop personal computer (PC) or a laptop, which not only allows real-time processing of hundreds of channels, but also re-processing the same signal many times to test various algorithms and parameters.

This R&D market is definitely not very large, but the technology fully profits from the ongoing developments in the PC sector including more and more powerful CPUs and graphics processors.. Around eighteen years after Dennis Akos presented a first post-processing software GPS receiver in his PhD thesis [3], all-in-view tracking of the complete GNSS constellation is possible on a PC costing the same as it did eighteen years ago.

At the latest when Samsung introduced the quad-core mobile phone S4 with its embedded powerful graphics processor, it became clear that we have entered the era of mobile super-computing. Within one of the latest phones (Samsung Note 4) the graphics chip Adreno 420 is able to perform more than one hundred billions floating point operations per second (GFlop/sec), enabling hyper-realistic 3D gaming. Naturally, the power consumption and the form factor of the chips and boards in question is now much more intriguing for using mobile supercomputing devices to design and build a new generation of software receivers able to leave the R&D niche behind and extend into a wider application market, which may have significant benefits for the end-user. For example, precise positioning in degraded or indoor environments is still a challenge not achieved with conventional technology. In this context, sophisticated methods of GNSS/INS integration and/or nonlinear estimation filters could make much better use of the available information from the GNSS signals than existing technology does. In this article we will for example discuss how Direct Position Estimation (DPE) fuses GNSS signal tracking and position estimation and how more flexible software receivers can foster its adoption.

Also niche markets like GNSS space receivers are possibly more efficiently realized as software receivers, as they can better adapt to the challenging environment and specific requirements. For example launch receivers not only face a bad satellite signal geometry (and thus rather low received signal power) due to the mounting of the antenna but they have to

cope together with high signal dynamics. The European Space Agency (ESA) is investigating this technology for its application within a space borne receiver. The use of GNSS for space applications is manifold, and includes launch monitoring or precise injection of satellites into geostationary orbits; applications which exhibit a high user dynamics. Standard GNSS space applications are positioning in low Earth orbits or on geostationary orbits and partly centimeter accuracy is achieved. Also formation flying or rendezvous and docking with the help of GNSS is demanding in terms of the required accuracy. Navigation in geostationary orbits, on highly elliptical orbits or navigation to and from the Moon requires a very high sensitivity of the GNSS receiver. All those applications could potentially benefit from a software receiver based approach as it can be fully optimized for the specific application.

In the remainder of this article, we first summarize the current state of mobile supercomputing and then match those developments to the GNSS software receiver core activities. Benchmarks will be presented for power consumption, FFT and signal tracking, all evaluated on embedded platforms purchased at the end of the year 2014. The results will be mapped into the effective number of correlators available for signal acquisition and the number of receiver channels. The obtained numbers are matched to the nominal peak performance of the platform and an explanation is given, if it is not achieved. The article is completed with a discussion of the importance of these computing advances in the implementation feasibility of advanced tracking and positioning techniques, which would benefit from the supercomputing capabilities of current processor technology.

RECENT COMPUTING DEVELOPMENTS

It is common wisdom that computers get ever faster, smaller, and more efficient. In the world of supercomputers there is actual empirical data to back up and quantify this impression. The Top500 project (www.top500.org) has been tracking the performance of the world's 500 most powerful computing systems for over 20 years. The No. 1 system on the first issue of the list in June 1993 achieved a whopping 59.7 GFlop/sec when solving a linear system of equations using an LU decomposition approach. This benchmark is still used today to determine performance in the form on the Linpack application.

When looking at the historic development of the Top500 list provided in Figure 1, we can identify a remarkable performance growth since 1993. The average performance of the 500 top systems (green points) grew by a factor of 84% per year since 1993 and the performance of the number one system (yellow points) improved even faster (90 % per year). This improvement is fundamentally fueled by advances in semiconductor technology governed by Moore's law, which states that the number of transistors doubles roughly every 18 months (or, in other words, improves by about 60% per year). Computer architecture enhancements (in the past) and increased parallelism (a more recent trend) account for the performance growth beyond what Moore's law predicts.

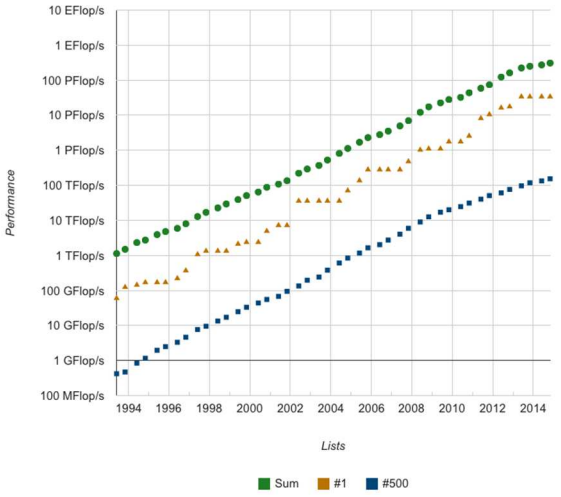

Figure 1: The Top500 list of supercomputers

The trend towards parallelism is however not limited to supercomputers. While the big systems in the Top 500 list are today composed of hundreds of thousands of cores, multi- and manycores are dominating desktop and server systems and have more recently also appeared in mobile and embedded platforms. One speaks of a multi core system, if it utilizes a few (below or around a dozen) computational units, and a many core system may have hundred or more units. Multiple computing cores are however not the only form of parallelism employed by modern architectures. A particularly efficient form of parallelism is known as SIMD (single instruction multiple data), which is employed in virtually all computers today. Here, the same operation is performed on multiple data items instead of just one. For some application areas, such as dense linear algebra, this directly translates to increased performance. SIMD (with the number of data items that can be processed in parallel) is constantly increasing. For desktop CPUs the SIMD registers are today 256 bits wide and they can thus operate on 4 double precision floating point (FP) numbers or 8 single precision FP numbers at the same time. Upcoming processor generations will double the SIMD width to 512 bits.

For mobile and embedded systems there is an ever increasing need to provide strong computing capability to support a variety of demanding applications (video encoding, augmented reality, gaming). This has driven the performance of these platforms where they can actually be compared to supercomputers of the past. A modern mobile device

can achieve a Linpack performance of 2-4 GFlop/sec, which means that it would indeed have been placed on the Top500 list in the early 1990s.

At least as important as the improvement in raw performance is the energy efficiency of computing devices. Analysis of historic devices going back to the 1940s has uncovered what is sometimes referred to as Koomey's law: The amount of computations that can be performed per unit energy improves by about 58% per year. The energy efficiency metric has the nice property that it can be used to compare across the whole spectrum of computing devices. Figure 2 shows such a comparison between the achieved energy efficiency of systems in the Top 500 list (blue dots), the prediction made by Koomey's law as red line, and the Apple iPad2 as an example mobile system. Highlighted are systems that have been specifically designed for energy efficiency (the BlueGene line of systems by IBM) and more recent accelerator platforms based on Xeon Phi and GPU hardware plus the iPad2.

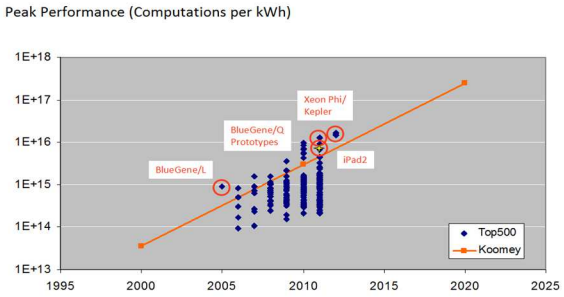

Figure 2: Computational power per kilowatt

TEST HARDWARE

For benchmarking the software receiver algorithms we use several development platforms. All are shown in Figure 3.

The first one is the ~200 $ ArndaleOcta board with a Samsung Exynos 5420 CPU consisting of four 1.8 GHz Cortex-A15 cores plus an ARM Mali-T628 GPU. It is used in some versions of the Samsung Galaxy S5 mobile phone. The GPU has a remarkable floating point performance of 102 GFlop/sec if all six GPU cores are used. Our benchmark application utilises only four GPU cores due to minimisation of the implementation effort in the Linux Kernel yielding a theoretical peak performance of 68 GFlop/sec for the GPU.

The second embedded board is a HardKernel ODROID-XU3 Lite featuring a Samsung Exynos 5422 CPU (Cortex-A15 1.8GHz quad core and Cortex-A7 quad-core). We used this board only for tracking benchmarks and not for FFT tests. The tracking benchmarks were carried out on one of the A15 cores running at 1.8 GHz.

The third platform is the Firefly single board computer using a Rockchip RK3288 CPU with a Cortex-A17 1.8 GHz quad-core. The tracking benchmarks were run on one of the Cortex-A17 cores with 1.8 GHz.

The fourth test platform is a One Plus One mobile phone based on a Qualcomm Snapdragon 801 processor. Again it was used only for tracking benchmarks on one of the Krait 400 cores. In contrast to the first three Linux boards, it runs Android 5.0 in the CyanogenMod version CM12S.

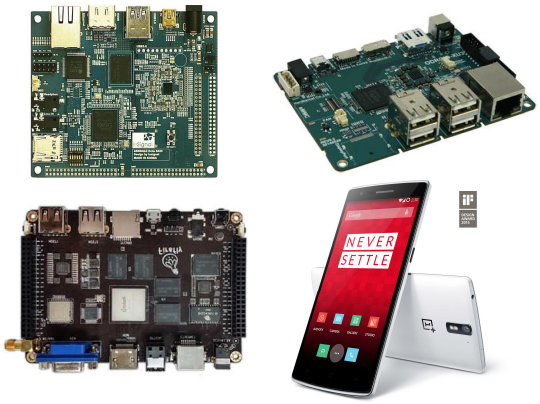

Figure 3: Embedded test platforms: Upper left: Arndale Octa, upper right: Odroid, lower left: Firefly, lower right: One Plus One

SOFTWARE RECEIVER CORE

IFEN is currently developing a new embedded software receiver which combines the algorithmic performance of the PC-based software receiver SX3 with the efficiency of the firmware from the field-programmable-gate-array (FPGA) based receiver NavX-NTR. This new embedded software receiver runs under the Linux Operating System but can in principle also be compiled for Android or without any operating system at all. Additionally Windows is supported but for Windows, the IFEN's SX3 software receiver delivers more features.

The embedded software receiver receives IF samples via USB, TCP/IP or from file input. It is designed as a multi-GNSS and multi-frequency receiver plus inertial measurement unit (IMU) support. Currently it only supports multiple correlators for tracking in DLL, FLL or PLL mode, but will be upgraded to vector tracking and deep GNSS/IMU integration soon. It operates most efficiently with 1-bit samples and signal tracking is performed on the CPU. Signal acquisition uses either the CPU or the GPU and a FFT based algorithm with resampling is employed [4].

Current applications are to proof the concept of a dual frequency GPS/Galileo space software receiver in low Earth orbit, and to be used for precise positioning with an integrated RTK or PPP module within UAVs also supporting GPS and Galileo. A

screenshot of the minimal user interface is shown in Figure 4 with the receiver in low Earth orbit mode.

BENCHMARKS

The following sections cover the power consumption, acquisition (FFT) and tracking benchmarks.

POWER CONSUMPTION

Mobile applications often have some power constraints, especially in the aviation or space sector, where the power-to-weight ratio plays a dominant role. Thus the power consumption of such an embedded software receiver is an interesting attribute which we have measured for typical load levels. One has to be careful when performing a power consumption measurement for a specific application due to the fact that also (for the specific application unnecessary) peripherals, such as the video connection, can contribute to the overall power balance significantly. This video connection is part of our Arndale development board and can not be deactivated.

Three measurement scenarios were set up to determine the power consumption of the Arndale Octa development board running a software receiver acquiring and tracking common GPS L1 signals. All power measurements are listed in Table 1.

The test setup consists of the ArndaleOcta connetected to a power adapter delivering 5V up to 3 Ampere in form of Direct Current (DC). A multimeter was used to measure the actually consumed current by serial placement within the power supply circuit. The consumed power P is calculated with P=V*I, while V refers to the voltage of 5V and I to the measured current.

The first measurement acts as a baseline and determines the power consumption of the board with the default peripheral settings and only the Linux operating system running. The second scenario determines the increase of power consumption during nominal tracking (after acquiring the signals). This is achieved by setting 12 tracking channels and switching off the acquisition unit. The third scenario is similar to the second, but with the acquisition unit permanently active, which causes an additional increase in CPU load and is heavily loading the GPU. The overall power consumption with acquisition activated increases to 4.35 W with the largest part attributed to the background power.

	System state	CPU load [%]	No. tracked channel	Power consumption [W]
1	OS Linux started SW Receiver Off	0.20	-	2.75
2	SW Receiver On Tracking only	52.0	12	4.08
3	SW Receiver On Tracking + Acquisition	70.6	12 + acq.	4.35

Table 1: Power consumption of the embedded software receiver on the Arndale board

All measurements were performed during the early development phase of the embedded software receiver. Especially for the CPU load and therefore in the power to number of tracking channels ratio, a lot of optimizations have not yet been implemented. The increase in CPU load during acquisition is based on a first implementation of the OpenCL acquisition which pre-computes all Doppler bins for acquisition search and moves the whole data to the OpenCL based FFT engine consuming a lot of CPU power.

```
<SwCorr> PERFORMANCE: PrmTime = 972.652587890625; RealTime = 0.97265261411666870117
<BEGIN> AUTO,PVT
<PVT> Position: #Obs=24 x=-1504210.97 y=-4021318.75 z=-5248476.35 [m] vx=7156.63 vy=-1421.43 vz=-963.53 |v|=7359.77 [m/s] Sol=SPS|3D
<PVT> ClkBias=-0.009064842[s] ClkDrift=-5.899449863[m/s] = -31.011[Hz]
<PVT> lat=-50.892538 lon=-110.508769 h=415565.36 h(mean-sea)=415565.36
<PVT> HDOP=0.69 VDOP=0.94 TDOP=0.78
<END>
<BEGIN> AUTO,CHANSTATE
<CH> Clock: 22436182439270400 sec=1095516720.667500019
<CH> Time: 2014-09-23 05:12:50.879598633 [1811/191570.879598633]
<CH> Chan   Doppler       Code   C/N0     SNR     Track   Stat Config        Mng   Residuum CdDoppler   CoCa
<CH>    0   1669.48   353.0682   44.8   29.87      98.7  TPbfN GPS-L1-I-15           -7.04   1157.90   3.7
<CH>    1  28614.09   209.1043   42.9   19.54      98.4  TPbfN GPS-L1-I-20           -6.41  28779.47  -4.8
<CH>    2  -1944.76   469.8911   44.8   30.14      98.1  TPbfN GPS-L1-I-23           -7.00  -2082.60  -2.3
<CH>    3  28559.05   646.0318   42.9   19.62      97.9  TPbfN GPS-L1-I-24           -3.34  28838.45   4.3
<CH>    4 -14898.84   695.0613   44.3   26.76      97.6  TPbfN GPS-L1-I-19           -5.88 -14692.39   6.2
<CH>    5  -6133.24   233.0564   44.7   29.57      97.3  TPbfN GPS-L1-I-6            -5.35  -6319.47   7.1
<CH>    6  27006.88  2141.1974   36.9   19.63      96.7  TPbfN GAL-E1-B-17           -7.86  26950.89   0.3
<CH>    7  34166.50  1105.4088   35.9   15.51      93.0  TPbfN GAL-E1-B-23          -10.13  34728.20  -0.8
<CH>    8 -19824.19  3591.4284   37.7   23.82      54.6  TPbfN GAL-E1-B-8            -7.62 -20891.18   1.1
<CH>    9   -408.45  1036.8461   38.6   29.21      83.2  TPbfN GAL-E1-B-16           -7.68   -104.37   0.3
<CH>   10 -15057.58  2720.6784   38.1   25.65      85.6  TPbfN GAL-E1-B-1            -7.58 -15678.12   0.0
<CH>   11 -24186.01  4037.4734   37.3   21.50      91.2  TPbfN GAL-E1-B-9            -9.81 -23935.05  -0.7
<CH>   12   1248.15  3530.2777   45.9   38.83      91.6  TPbfN GPS-L5-I-15     *      4.79   1251.69  -0.6
<CH>   13  21367.70  2090.6106   45.5   35.32      89.6  TPbfN GPS-L5-I-20     *      6.27  21374.81   0.2
<CH>   14  -1446.06  4698.4291   46.0   40.11      89.6  TPbfN GPS-L5-I-23     *      7.13  -1460.62  -0.1
<CH>   15  21327.19  6460.0105   45.5   35.32      91.6  TPbfN GPS-L5-I-24     *      5.68  21343.90  -0.4
<CH>   16 -11124.67  6950.2184   46.0   39.74      85.6  TPbfN GPS-L5-I-19     *      5.69 -11139.30  -1.3
<CH>   17  -4577.38  2330.1946   45.9   38.98      91.6  TPbfN GPS-L5-I-6      *      5.46  -4632.18  -0.3
<CH>   18  20165.75   951.4244   39.8    9.58      86.8  TPbfN GAL-E5a-I-17    *      8.23  20567.35  -1.2
<CH>   19 -11245.94  6746.2340   40.3   10.72      82.8  TPbfN GAL-E5a-I-1     *      8.54 -11369.88   3.4
<CH>   20  25511.26   823.4625   39.7    9.35      79.7  TPbfN GAL-E5a-I-23    *      8.19  25380.79  -2.1
<CH>   21   -299.96   137.9135   40.2   10.54      80.8  TPbfN GAL-E5a-I-16    *      8.35   -407.48  -1.4
<CH>   22 -18063.18  9684.0782   40.1   10.32      81.8  TPbfN GAL-E5a-I-9     *      9.41 -18217.65   1.2
<CH>   23 -14810.52  5223.7526   40.2   10.48      51.8  TPbfN GAL-E5a-I-8     *      7.96 -14770.95   3.0
<END>
```

Figure 4: Screen shot of IFENs embedded software receiver tracking the international space station on L1/E1/L5/E5a

Naturally, this should happen directly in the OpenCL kernel running on the GPU. Thus, the numbers listed in table 1 are upper limits on the power consumption, with potential for improvement.

SIGNAL ACQUISITION

In the FFT acquisition context, the OpenCL standard shall be mentioned providing a computing language compatible with many CPUs and GPUs. Indeed also our Arndale embedded test system was able to support OpenCL on the GPU after some Linux kernel patches. Without OpenCL it would have been virtually impossible to program the GPU on the embedded system at all. Furthermore, the clFFT library [5] exists as an open source project facilitating an efficient implementation of FFTs based on OpenCL. This library was initially implemented and optimized for AMD GPUs for desktop PCs, but also runs on other OpenCL compatible GPUs including our embedded test system to perform benchmarks.

The results of the FFT benchmarks for the Arndale Octa board using the ARM Mali-T628 are shown in Table 2, where an FFT length of 8192 points was used. This length matches acceptable well with the Galileo E1 codes having 4092 chips and a PRN code duration of 4 ms. The Galileo E1 correlation function is computed with two samples per chip.

Within the currently used FFT-based acquisition algorithm, one FFT can be used to search all code phases within one Doppler bin yielding a coarse estimate of the possible number of correlators shown in Table 2. As outlined in [4], the resampling FFT acquisition requires two forward FFTs, and one inverse FFT for each Doppler bin. Only the inverse FFTs are considered in the following as typically a large number of Doppler bins is considered. The effective number of correlators corresponds to the number of (hardware) correlators running in parallel in real-time to achieve the same result. Please note that the correlators discussed here are complex correlators including I and Q components. The computation of the GFlop/sec value is based on the usual $5N\log_2 N$ formula estimating the number of floating point operations (multiplication or addition) for a N-point FFT.

In our benchmark implementation we use clFFT to solely compute the inverse FFTs and to compute all Doppler bins in parallel on the GPU. The forward FFTs are computed on the CPU. All FFTs use single precision float data types. The stated GPU execution time refers to the time needed to perform all inverse FFTs, where each inverse FFT corresponds to a Doppler bin. The measured time neither considers the forward FFTs nor considers time needed for memory shifts to and from the GPU.

No. of parallel FFTs	GPU execution time [ms]	GFlop/sec	Effective no. of correlators for Galileo E1 [Thousands]
1	1.073	0.5	30
2	1.633	0.65	40
4	2.572	0.83	50.6
16	8.575	1.0	60.7
50	24.720	1.1	65.8

Table 2: FFT performance on the embedded system

We found that the GNSS signal acquisition generally benefits from executing the FFT operation on a GPU. Performing the acquisition on the GPU has the benefit of de-stressing the CPU and enabling more resources for tracking. In the embedded world, we realize that FFT libraries and GPU hardware exist, that can be more or less brought together to compute FFTs. Although the number of achievable correlators is still impressive (more than 50 thousands), we noticed that a performance of only around 1 GFlop/sec was achieved, with a theoretical limit of 68 GFlop/sec supported by the hardware.

However, a more detailed analysis reveals that this theoretical peak capability of the GPU can only be achieved if the dot product, scalar, and vector units are all used at the same time [6]. Any real world code is unlikely to be able to utilize all these units at the same time and the maximal possible performance will therefore be lower than 68 GFlop/sec. If, for example, a kernel can only make use of the scalar units, the maximal performance reduces more than eightfold to 8 GFlop/sec. Another point of consideration is that besides the compute aspect, application kernels can be limited by memory access. This situation is best visualized employing the roofline model [7, 8], where FFT is known to often be among the memory bound kernels. This is consistent with our finding that executing several FFTs in parallel did not show a big performance gain if at least two FFTs were executed.

In [9] a similar GPU (T-604) is benchmarked and only a rather low main memory bandwidth of 8.39 GByte/s has been obtained. A similar value of 9.2 GByte/s has been obtained on the One Plus One mobile phone for the device-to-device memory bandwidth of the Adreno 330 GPU using the OpenCL-Z app. It is not trivial to measure the required amount of memory transfers for a FFT implementation but assuming that each of the $5N\log_2 N$ multiplications or additions operates on the main memory an upper bound can be established. Each operation has two complex numbers as input and one as output. If working with 32-bit float

numbers, maximally $3*2*4*5*N\log_2 N$ bytes are transferred. This evaluates for 8192 FFT points to 12.8 Mbyte taking 1.5 ms of time. The 8192 FFT points represent a time span of 4 ms and thus the memory transfer consumes a large portion of real time.

A more detailed performance analysis for the FFT implementation on our embedded platforms is still under investigation, however we believe the performance gap can be significantly reduced, with a potential speed-up of at least an order of magnitude provided a detailed understanding of the MALI GPU and its memory cache hierarchy can be obtained.

SIGNAL TRACKING

For the signal tracking benchmark we set up a dedicated program, simulating a receiver tracking channel including code and carrier NCOs plus correlators. Then, we measured the number of samples the platform can process within one second. This number is called million-correlations-per-second (MCOPS) and is measured on a single core of the CPU. The channel was per default configured to work with five correlators (e.g. very early, early, prompt, late and very late) but also other configurations with 2, 3 or 21 correlators have been used. The implemented correlation scheme operates with 1-bit samples and actually realizes the correlation as an exclusive-OR operation plus bit counting. It has been verified that the code correlation is the bottleneck as the same carrier NCO is used for all correlators.

Figure 5 shows this implementation symbolically with a 10.24 or 20.48 MHz one bit I/Q input sample stream and three correlators (Early, Prompt and Late) as well as a typical carrier and code steering loop. As mentioned above, the most time consuming part belongs to the correlation operation. One correlation operation relates to correlating one sample of the input stream with the reference signals, thus the resulting number of tracking channels depends on the working sample rate. The benchmark is performed on a single CPU core, thus the maximum number of tracking channel is a multiple of the CPU cores and is calculated by #CH = MCOPS / SampleRate * CPUcores.

The above described benchmark application was cross-compiled with highest compiler optimization flag -O2 in order to achieve highest performance on each development board. To get rid of bottlenecks caused by reading samples from any interface such as network, USB or internal storage, samples are preloaded and read from the main memory.

As depicted in Table 3, increasing the number of correlators per channel results in a reduction of the MCOPS value.

No. of corr.	MCOPS per core Arndale Octa	MCOPS per core Odroid	MCOPS per core Firefly	MCOPS per core One plus one
2	428	900	825	882
3	326	715	650	702
5	231	490	460	470
21	66	Not tested	Not tested	Not tested

Table 3: Tracking performance on the embedded boards

In the above table the ArndaleOcta board shows quite bad results compared to the other boards of almost similar CPUs. However, at the start of our work this was the only board available with a working OpenCL implementation under Linux. This was based on an unofficial board support image by Linaro which had OpenCL support in place but to the expense of only basic support for other board features. One of the main problems is the lack of support for CPU frequency handling and therefore it is unclear at which CPU speed the ArndaleOcta board was actually running in the benchmarks. For the other (newer) boards it could be ensured that the CPUs were running at full speed at 1.8 GHz. This can also be seen by running the openssl benchmark which shows less than half the performance for ArndaleOcta compared to the ODROID board.

For a given MCOPS value, the number of possible tracking channels is determined by the sample rate and by the number of used CPU cores. Exemplary relationships are shown in Table 4.

MCOPS per core	Sample rate	No. of used CPU cores	Channel no. upper limit
900	10.24 MHz	4	351
715	20.48 MHz	4	139
490	20.48 MHz	4	95
66	10.24 MHz	4	25

Table 4: Relating MCOPS and sample rate to number of tracking channels

For example all the embedded system easily allows implementation of a single frequency GPS, Galileo, GLONASS plus BeiDou Receiver. Alternatively, wideband tracking (20 MHz bandwidth and 40 MHz sample rate) of dual frequency GPS is possible. Another conclusion that these numbers suggest is that multi-correlation (that is, more than 5 per channel) schemes are possible. The latter opens the door for implementing sophisticated synchronization and positioning solutions that

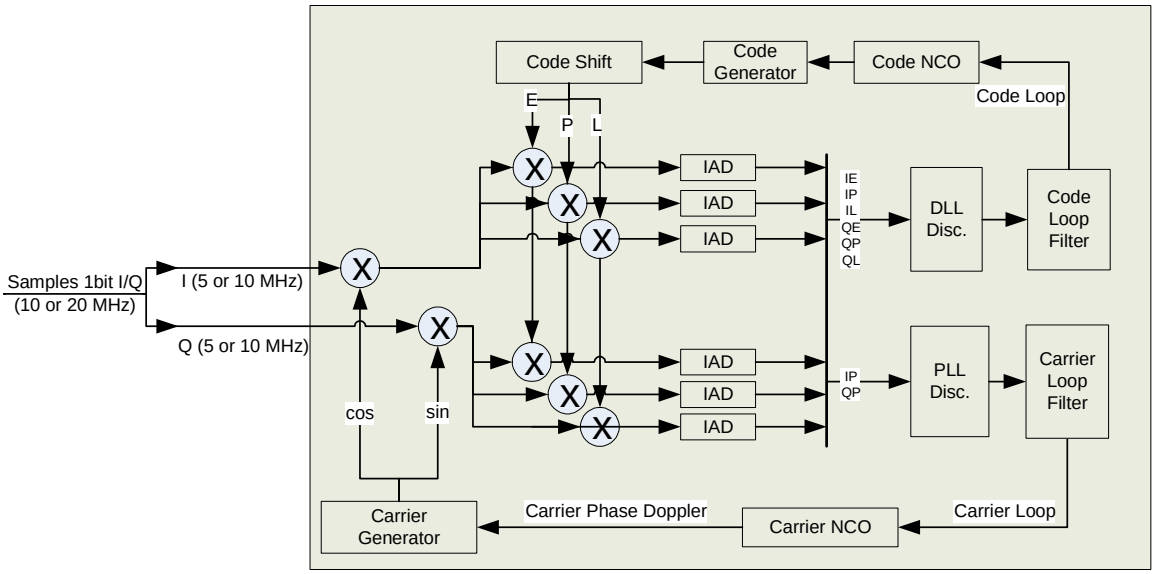

Figure 5: Tracking channel block diagram

typically use large number of correlators per integration time.

Similar as for the FFT, a large amount of memory is transferred. But as the memory is accessed linearly, the CPU caches are much better exploited.

ADVANCED TRACKING AND POSITIONING TECHNIQUES

Legacy GNSS receivers use phase lock loop (PLL) and delay lock loop (DLL) variations to perform signal tracking. There is a deep literature comparing the various approaches, mainly based on the selection of different discriminator functions that provide enhanced performance to the receiver, for instance, in terms of multipath rejection capabilities. Typically, these PLL/DLL schemes require 3 or 5 correlation points to operate, depending on the specific implementation. Arguably, these approaches provided a good balance between tracking performance and computational complexity. Particularly, these savings were important when the computational resources were scarce. As motivated in this article, this limitation does no longer exist in modern computer platforms, or it is going to vanish in the near future, at least if not absolute minimum power consumption is sought, e.g. for background operation of a GNSS receiver.

Imagine going beyond the classical PLL/DLL architectures. The possibility of implementing multi-correlation architectures turns this wish into a reality. For the last decades we have seen a number of proposals for more sophisticated tracking and positioning techniques, most of them requiring multiple (i.e. more than 5) correlation outputs to be of use [10-14]. Their benefits have been quantified in the literature in terms of performance enhancements with respect to legacy solutions. However, these proposals could not compete against legacy solutions in terms of complexity and implementation feasibility in state-of-the-art processing technologies. This is not the case nowadays and the gap will reduce quickly.

Figure 6 shows the baseband processing diagram of a generic GNSS receiver with K channels. There, we used the complex notation for the sake of clarity, thus avoiding I&Q duplicated arms in the diagram. The focus is on the tracking and positioning operations, and thus acquisition is assumed as a previously accomplished stage. In the digital domain, a closed-loop receiver has to perform carrier wipe-off and code correlation, which is fed to a Digital Signal Processing (DSP) block implementing the chosen tracking/positioning algorithm. Notice that we have L correlation outputs, although this number could vary among channels. The diagram in Figure 6 is sufficiently generic to represent legacy and advanced tracking techniques. For instance, whether the outputs from the K channels are processed independently or jointly is handled at the DSP module. The former corresponds to the usual PLL/DLL schemes, where each channel tracks one satellite (with $L \leq 5$). Then, an additional navigation filter is in charge of delivering position, velocity and time (PVT) estimates after computing the necessary observables.

Other more sophisticated techniques fall in this category, such as the Multipath Estimating DLL (MEDLL) [14] or the Multipath Estimating Particle Filter (MEPF) [11]. Both techniques share the idea of jointly estimating the parameters of the line-of-sight signal and those from the multipath echoes, with the goal of mitigating the effects of the latter in the former. Their main difference is in the statistical principle used to derive the estimators: whereas MEDLL is based on the Maximum Likelihood (ML) principle, the MEPF resort to Bayesian filtering theory. Their implementation is consequently

different, but they share the fact that L should be larger than the values considered in PLL/DLL schemes. Roughly, several tenths of correlators per channel should be considered when implementing these techniques for full exploitation of their capabilities and enhanced performances [13]. Consequently, the computational resources required are moderately large, particularly when a large number of particles are used in MEPF. However, the intrinsic parallelization properties of particle filtering should benefit from the computational advances in this field, with the ability to share the load among various processing resources.

On the other hand, the receiver depicted in Figure 6 can accommodate as well combined tracking structures where the outputs from the K channels are jointly processed. The output of this processing is the control signal to drive the tracking loops and the PVT solution. In this category we find Vector Tracking Loops (VTL), a convenient modification of the usual PLL/DLL schemes that allows for exploiting the synergies among channels. Therefore, the number of correlators in VTL schemes is driven by the underlying PLL/DLL techniques ($L \leq 5$) and thus its bottleneck is not in this part of the baseband processing chain.

An alternative, more general approach has been proposed under the name of Direct Position Estimation (DPE) [12]. DPE has been initially derived under the ML principle, but an implementation based on Bayesian filtering methods was presented as well. Therefore, open and closed loop architectures are possible for DPE, in contrast to VTL. In both cases, DPE was seen to enhance the performance of legacy receivers in terms of multipath mitigation, operation in weak signal conditions, or other challenging situations. As a payoff, DPE needs a receiver able to compute a larger number of correlator outputs L. On the order of tenths of them, as in the channel-per-channel advanced techniques mentioned earlier. With regards to important operations to be performed, in a fully open loop DPE scheme, there is a multivariate optimization to be solved which involves systematical evaluation of an operation with complexity asymptotically proportional to K^2. This systematic evaluation could be relaxed in closed-loop DPE schemes (for instance using particle filtering) [10], in which case this computation could be parallelized.

CONCLUSIONS

The increase of computational power can be used to build small and power efficient GNSS software receivers and our experimental benchmarks show how well the currently available embedded technology can be exploited for these purposes.

The tracking and correlation code on the ARM CPU can and has been coded without making use of dedicated libraries. Correlation with 1-bit samples is directly supported by SIMD instructions of the ARM architecture. The maximum number of channels is several hundreds, if standard tracking is used. Remarkably, the low power embedded platforms would support up to 25-50 channels with e.g. 21 correlators covering a rather large ranging uncertainty of ±585 meters for high bandwidth signals to be used for e.g. Direct Position Estimation or other advanced tracking or positioning technique.

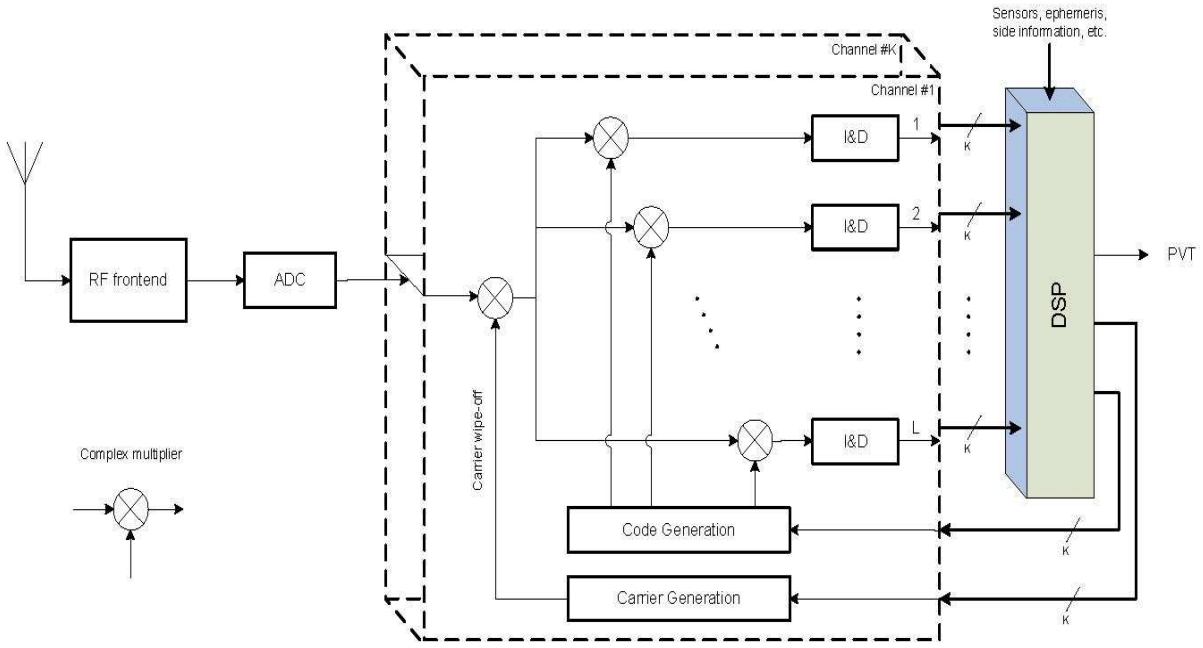

Figure 6: Block diagram for multi-correlation tracking in a GNSS receiver

The signal acquisition on the embedded GPU supporting OpenCL provides more than 50000 correlators and we think that the memory bandwidth is limiting this number. More efficiently coded FFT libraries may possibly give a much larger number of correlators. As a positive side effect the currently used FFT implementation on the GPU uses only 0.3 W of electrical power.

We also pointed out that modern techniques coping with well-known impairments have appeared in the last years, promising important improvements at the cost of increase computational burdens. Here we discussed a brief sample of them, which served to motivate the idea that: a) multicorrelation strategies are the desired choice in most advanced techniques, and b) these techniques often require complex matrix operations to be computed, which sometimes can be parallelized. Both points are doable either with the current technology or that to come in the near future.

ACKNOWLEDGEMENTS

The research leading to the part of those results has received funding from the European Space Agency (ESA) under the ESA Contract no. 4000111113/14/NL/CBi/fk (eSTAR).

REFERENCES

[1] Press release: Broadcom Announces Industry's First GNSS Location Hub Chip for Smartphones to Support Galileo Satellite System, http://www.broadcom.com/press/release.php?id=s885589

[2] P. Dabove and M. Petovello, “What are the actual performances of GNSS positioning using smartphone technology?”, InsideGNSS, November/ December 2014.

[3] D. M. Akos, “A Software Radio Approach to Global Navigation Satellite System Receiver Design,” PhD Dissertation. Ohio University. Columbus. Ohio. USA. August 1997.

[4] T. Pany, “Navigation Signal Processing for GNSS Software Receivers”, Artech House 2010, ISBN: 978-1-60807-027-5

[5] Advanced Micro Devices, Inc. “clFFT”. *clFFT*. GitHub, n.d. Web. 13 Feb. 2015. < https://github.com/clMathLibraries/clFFT>.

[6] ARM White Paper "OpenCL on MALI FAQs" <http://malideveloper.arm.com/downloads/OpenCL_FAQ.pdf>

[7] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. Commun. ACM 52, 4 (April 2009), 65-76.

[8] Georg Ofenbeck, Ruedi Steinmann, Victoria Caparros, Daniele G. Spampinato and Markus Püschel "Applying the Roofline Model"; Proc. International Symposium on Performance Analysis of Systems and Software (ISPASS), 2014

[9] Jee Choi, Marat Dukhan, Xing Liu, Richard Vuduc, “Algorithmic time, energy, and power on candidate HPC compute building blocks”, Proc. IPDPS 2014.

[10] P. Closas, C. Fernández-Prades, Bayesian Nonlinear Filters for Direct Position Estimation , in Proceedings of IEEE Aerospace Conference, 6-13 March 2010, Big Sky, MT (USA).

[11] P. Closas, C. Fernández-Prades, J.A. Fernández-Rubio, “A Bayesian Approach to Multipath Mitigation in GNSS Receivers,” IEEE Journal of Selected Topics in Signal Processing, special issue on Advanced Signal Processing for GNSS and Robust Navigation. Vol. 3, No. 4, pp. 695-706, August 2009.

[12] P. Closas, “Bayesian Signal Processing Techniques for GNSS Receivers: from multipath mitigation to positioning,” PhD Dissertation. Universitat Politècnica de Catalunya (UPC). Barcelona. Spain. June 2009.

[13] A. Fernández, M. Wiz, P. Closas, C. Fernández-Prades, F. Zanier, R. Prieto-Cerdeira, M. Crisci, “ARTEMISA: Preliminary Results of Advanced Receiver Techniques for Multipath Mitigation,” in Proc. of the ION GNSS 2012 meeting, September 2012, Nashville, TN.

[14] R. D. J. Van Nee, J. Siereveld, P. C. Fenton, and B. R. Townsend, “The multipath estimating delay lock loop: approaching theoretical accuracy limits,” Position Location and Navigation Symposium, pp. 246–251, April 1994.