Accelerating GNSS Software Receivers

ION GNSS+ 2016. Portland, OR, Sept. 14th.

Session A1: Advances in GNSS Software-defined Receivers

C. Fernández-Prades, J. Arribas, P. Closas

Centre Tecnològic de Telecomunicacions de Catalunya

10 years ago...

- Back in 2006, Gregory W. Heckler published an open source library implementing SIMD-based correlators for GNSS software receivers.

  G. W. Heckler and J. L. Garrison
  SIMD Correlator library for GNSS software receivers
  GPS Solutions, vol. 10, no. 4, pp. 269–276, Nov. 2006

- That library included arithmetic functions that operate on 16 bit integers, providing MMX and SSE2 versions of each function, as well as an MMX-enabled fixed point, radix-2, FFT.
- The code was then integrated into a software-defined GPS L1 C/A receiver (still available online[1]), achieving real-time operation in computers of that time.

[1]See `https://github.com/gps-sdr/gps-sdr`

Motivation

- *"The Rise of GNSS"*:
  - GNSS scenario with 100+ satellites, 10 different GNSS open signal waveforms for civilian usage, belonging to 4 different systems and broadcast at 4 different frequency bands.
  - New modulations require more bandwidth and more processing complexity on the receiver.
  - The natural target is a multi-constellation, multi-band GNSS receiver operating in real-time.
- Computing goes parallel:
  - A typical desktop in 2006: Pentium IV processor at 2.0 GHz, dual-core technology, with memory bandwidths $\sim$ 2 GB/s.
  - In 2016, Intel is planning to release their Broadwell-E processor series, with a clock speed up to 3.50 GHz. Can house up to 20 cores, with memory bandwidths $\sim$ 100 GB/s.

Computing Ecosystem

- In 2016, desktop computers are not the dominant form factor anymore.
  - Laptops, gaming consoles, mini PCs, tablets and smartphones has pushed into the market other sort of processors with low power consumption figures and specific features for multimedia content handling.
  - Cloud computing paradigm.

This is a software design challenge

- We need to address both *efficiency* and *portability* at the same time.

Overview

Parallelization Strategies for GNSS Signal Processing

Task parallelism

Shared-memory parallel computers can work on several tasks at once:

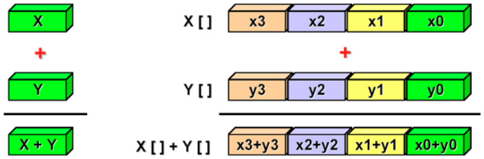
- by parceling them out to the different processors,
- by executing multiple instruction streams in an interleaved way in a single processor (multithreading), or
- by a combination of both strategies.

Multi-threading

To make this potential performance gain effective, the software running on the platform must be written in such a way that it can spread its workload across multiple execution cores.

Data parallelism

Instructions that can be applied to multiple data elements in parallel.
This computer architecture is known as *Single Instruction Multiple Data* (SIMD).



Most modern processors implement some SIMD technology (*e.g.*, SSEx, AVX, NEON, ...).

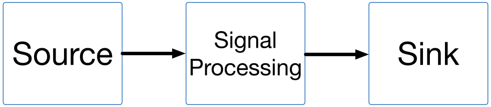
Other forms of parallelism

- Instruction-level parallelism
  - Computer processors are composed of a number of functional units that may be able to operate simultaneously.
  - A processor that supports this is said to have a *superscalar* architecture, and nowadays it is a common feature in general-purpose microprocessors.
  - Modern compilers put considerable effort into finding a suitable ordering of operations that keeps many functional units and paths to memory busy with useful work.
- GPU / FPGA Offloading
  - Some computational-intensive portions of the application can be executed in an external device.

Task Parallelization

A mathematical model for software radios

- A Kahn process describes a model of computation where processes are connected by communication channels to form a network.
- Processes produce data elements or tokens and send them along a communication channel where they are consumed by the waiting destination process.
- Communication channels are the only method processes may use to exchange information.



A very simple flow graph.

Kahn's model of process networks

Systems that obey Kahn's mathematical model are determinist: the history of tokens produced on the communication channels does not depend on the execution order.

With a proper scheduling policy, it is possible to implement software defined radio process networks holding two key properties:

- **Non-termination**: understood as an infinite running flow graph process without deadlocks situations, and
- **Strictly bounded**: the number of data elements buffered on the communication channels remains bounded for all possible execution orders.

Baseband processing

At this level of abstraction, parallelization of operations to the incoming signal is performed at a targeted satellite signal basis.

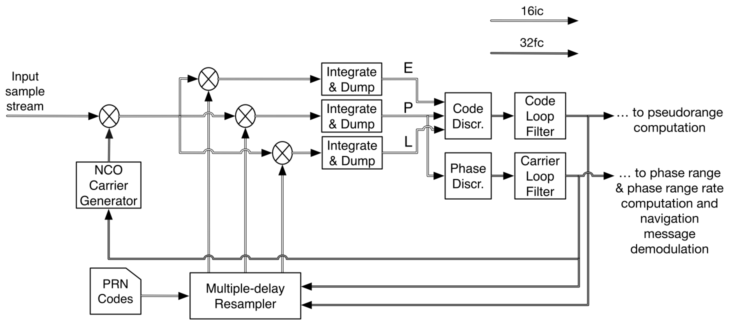
Input signal:

$$x_q[n]=\sum_{i=0}^{N_s-1} \tilde{\alpha}_i(t_n)\tilde{s}_{i,T}(t_n-\tau_{i_n})e^{-j2\pi f_{d_{i_n}} t_n} e^{j2\pi f_{\mathrm{IF}} t_n} + \tilde{w}(t_n)$$

At the matched filter output:

$$y_{i_k} = \frac{|a_{i_k}|}{2} K \frac{\sin(\pi \Delta f_{d_{i_k}} T_{int})}{\pi \Delta f_{d_{i_k}} T_{int}} \cdot d_i\left([k]_{\frac{T_b}{T_{int}}}\right) \cdot R_{\tilde{p}q}(\Delta\tau_{i_k}) \cdot e^{-j(\pi \Delta f_{d_{i_k}} T_{int} + \Delta\phi_{i_k})} + \tilde{w}_{i_k}$$

Typical receiver architecture



$$P_{i_k} = y_{i_k}\left(\Delta\hat{\tau}_{i_{k-1}}, \Delta\hat{f}_{d_{i_{k-1}}}, \Delta\hat{\phi}_{i_{k-1}}\right)$$

$$E_{i_k} = y_{i_k}\left(\Delta\hat{\tau}_{i_{k-1}}+\epsilon, \Delta\hat{f}_{d_{i_{k-1}}}, \Delta\hat{\phi}_{i_{k-1}}\right)$$

$$L_{i_k} = y_{i_k}\left(\Delta\hat{\tau}_{i_{k-1}}-\epsilon, \Delta\hat{f}_{d_{i_{k-1}}}, \Delta\hat{\phi}_{i_{k-1}}\right)$$

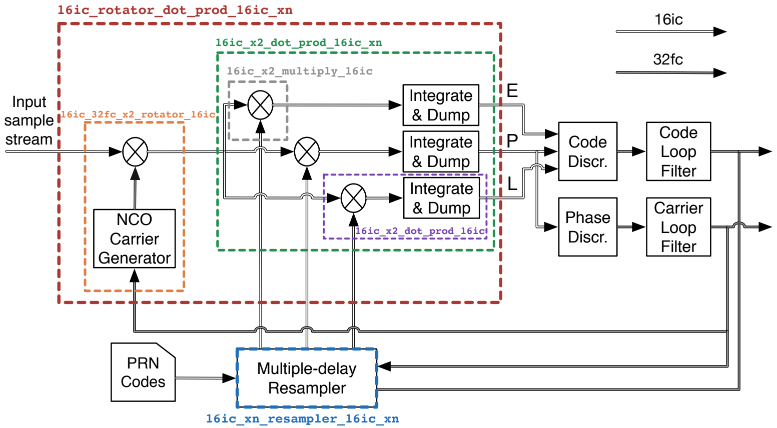
Data Parallelization

Single Instruction - Multiple Data (SIMD)

- In SIMD technologies, the same set of instructions is executed in parallel to different sets of data.
- This reduces the amount of hardware control logic needed by N times for the same amount of calculations, where N is the width of the SIMD unit.
    - 64-bit registers: MMX (1997).
    - 128-bit registers: SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, NEON (for ARM processors).
    - 256-bit registers: AVX, AVX2 (2013).
    - 512-bit registers: AVX-512 (2016).
- Designed for the acceleration of multimedia processing, can be used for GNSS signal processing.

SIMD in software-defined GNSS receivers

Typical functions in a software-defined GNSS receiver:



Data Types (I/II)

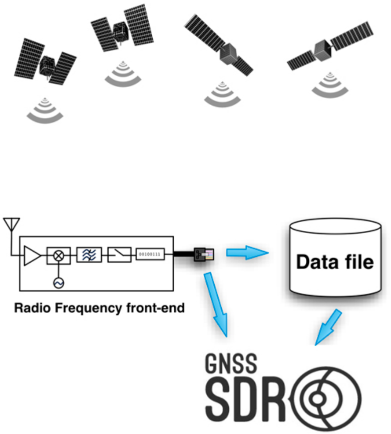
Type name in VOLK	Definition	Sample stream
"8i"	Signed integer, 8-bit two's complement number ranging from -128 to 127. C type name: `int8_t`	$[S_0], [S_1], [S_2], \ldots$
"8u"	Unsigned integer, 8 bits ranging from 0 to 255. C type name: `unsigned char`	$[S_0], [S_1], [S_2], \ldots$
"8ic"	Complex samples, with real and imaginary parts of type `int8_t` C type name: `lv_8sc_t` (*)	$[S_0^I + jS_0^Q], [S_1^I + jS_1^Q], \ldots$
"16i"	Signed integer, 16-bit two's complement number ranging from -32768 to 32767 C type name: `int16_t`	$[S_0], [S_1], [S_2], \ldots$
"16u"	Unsigned integer, 16 bits ranging from 0 to 65535. C++ type name: `uint16_t`	$[S_0], [S_1], [S_2], \ldots$
"16ic"	Complex samples, with real and imaginary parts of type `int16_t` C type name: `lv_16sc_t` (*)	$[S_0^I + jS_0^Q], [S_1^I + jS_1^Q], \ldots$

Data Types (I/II)

Type name in VOLK	Definition	Sample stream
“32u”	Unsigned integer, 32 bits ranging from 0 to 4294967295. C type name: `uint32_t`	$[S_0], [S_1], [S_2], \ldots$
“32f”	Signed numbers with fractional parts, can represent values ranging from $\approx 3.4\times10^{-38}$ to 3.4×10^{38} with a precision of 7 digits (32 bits). C type name: `float`	$[S_0], [S_1], [S_2], \ldots$
“32fc”	Complex samples, with real and imaginary parts of type `float` C++ type name: `lv_32fc_t` (*)	$[S_0^I + jS_0^Q], [S_1^I + jS_1^Q], \ldots$
“64u”	Unsigned integer, 64 bits ranging from 0 to $2^{64}-1$. C type name: `uint64_t`	$[S_0], [S_1], [S_2], \ldots$
“64f”	Signed numbers with fractional parts, can represent values ranging from $\approx 1.7\times10^{-308}$ to 1.7×10^{308} with a precision of 15 digits (64 bits). C type name: `double`	$[S_0], [S_1], [S_2], \ldots$

Implementation

GNSS-SDR: An Open Source Software-Defined Receiver

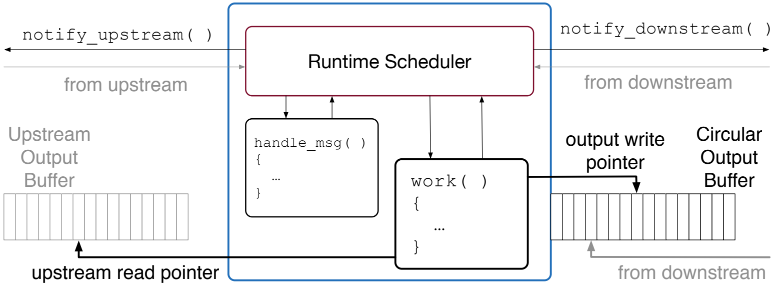


- Implements the whole processing chain from the output of a RF front-end up to computation of observables and PVT solutions.
- Flexible and scalable design.
- Free and open source software.

Source code available at `https://github.com/gnss-sdr/gnss-sdr`

GNU Radio signal processing block

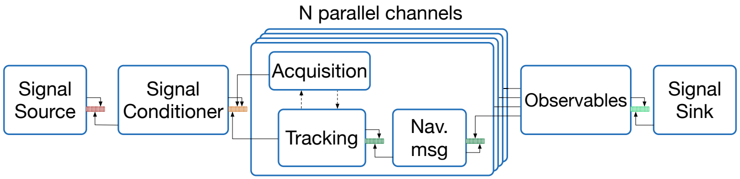
Kahn's mathematical model is implemented by GNU Radio, an open source framework for software-defined radios.



J. Corgan, "*GNU Radio runtime operation*," in Proc. GNU Radio Conference, Washington, DC, Aug. 24–28 2015.

GNSS Receiver flow graph

GNSS-SDR features a thread-per-block implementation, which scales well with the number of processors.



The underlying scheduler avoids processing bottlenecks and optimizes memory usage.

A new module of the Vector Optimized Library of Kernels

- Typical functions in a GNSS receiver have been implemented in an open source library: VOLK_GNSSSDR.
- VOLK_GNSSSDR provides several implementations for each function:
  - A *generic*, plain C implementation,
  - other implementations making use of different SIMD technologies (SSE3, SSE4.1, AVX, NEON, ...).
- A program runs all the implementations that can be executed in your computer, annotating which is the fastest and then selecting it at runtime when the function is called.

This strategy allows addressing efficiency and portability at the same time!

Results

Processing platforms (I/IV)

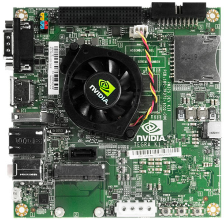
- **Platform #1 - Server**: a Dell's PowerEdge R730 server housing a CPU with two Intel Xeon E5-2630 v3 at 2.4 GHz (8 cores, 16 threads each) and an NVIDIA Tesla K10 GPU with 2 x 1536 CUDA cores clocked at 745 MHz. The operating system during tests was GNU/Linux Ubuntu 14.04, 64 bits, using GCC 4.9.2.

Processing platforms (II/IV)

- **Platform #2 - Laptop**: Apple's MacBook Pro Late 2013, with an Intel Mobile Core i7-4558U (quad-core) CPU at 2.4 GHz (active cores can be speeded up to 3.8 GHz), and Hyper Threading technology allows the system to recognize eight total "cores" or "threads" (four real and four virtual), plus an NVIDIA GeForce GT 750M GPU with 384 CUDA cores clocked at 967 MHz. The operating system during tests was Mac OS X 10.11, using Apple LLVM / Clang version 7.0.2.

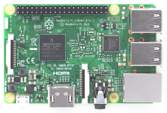
Processing platforms (III/IV)

- **Platform #3 - Embedded development kit**: NVIDIA's Jetson TK1 developer kit, equipped with a quad-core ARM Cortex-A15 CPU at 2.32 GHz and an NVIDIA Kepler GPU with 192 CUDA cores clocked at 950 MHz. The operating system during tests was GNU/Linux Ubuntu 14.04, 32 bits, using GCC 4.8.4.

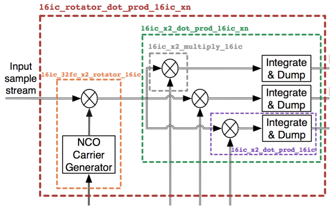


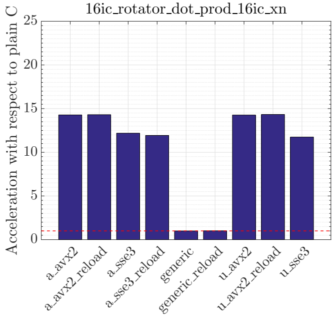
Processing platforms (IV/IV)

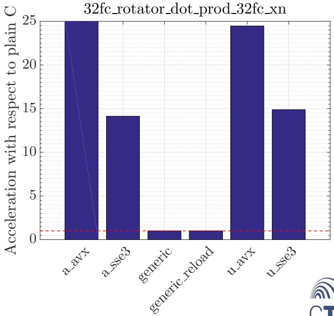
- **Platform #4 - Mini-computer**: Raspberry Pi 3 Model B, equipped with a Broadcom BCM2837 CPU (64 bit, ARMv8 quad-core ARM Cortex A53) clocked at 1.2 GHz. The operating system used during tests was Raspbian GNU/Linux 8 (jessie), 32 bits, using GCC 4.9.2.



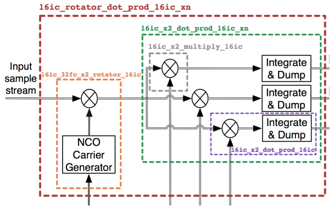
Processing results for Platform #1 - Server

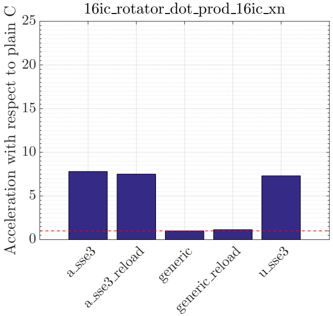


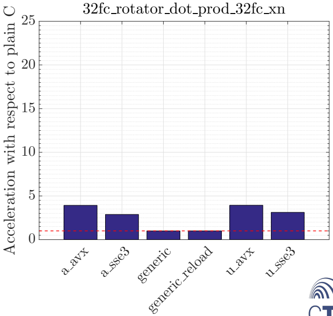




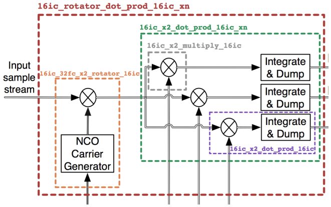
Processing results for Platform #2 - Laptop

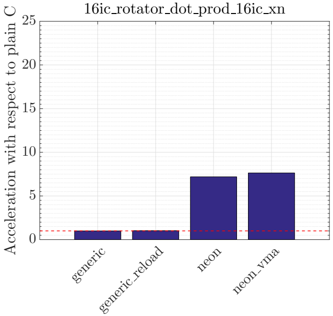


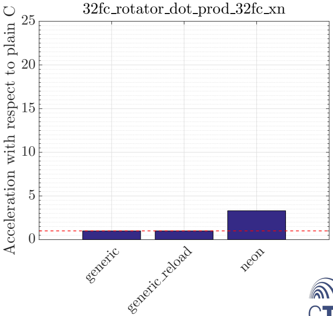




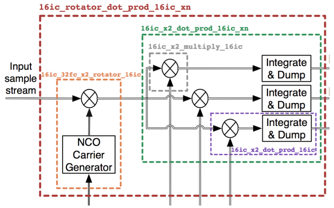
Processing results for Platform #3 - ARM EDK

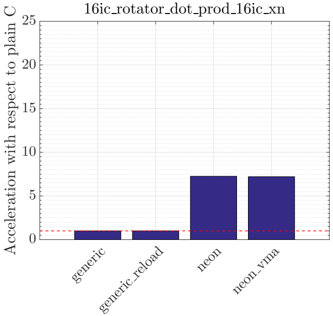


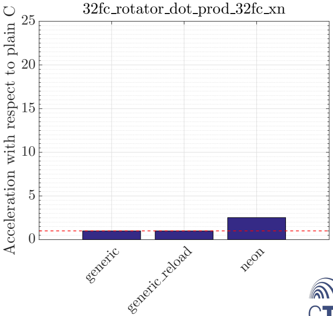




Processing results for Platform #4 - Raspberry Pi 3

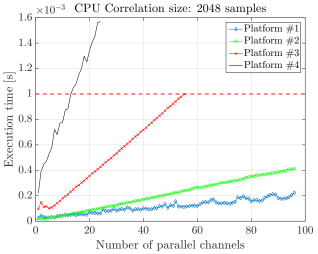


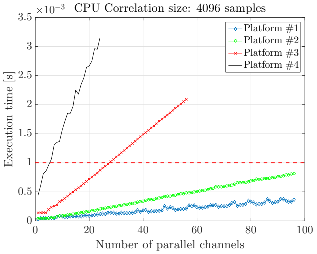


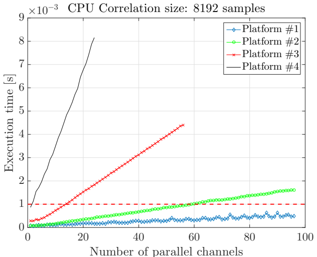


Processing results with CPUs

3-correlator GPS L1 C/A channels with different sampling rates.







Processing results of GPU offloading

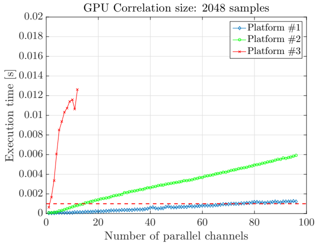
3-correlator GPS L1 C/A channels with different sampling rates.

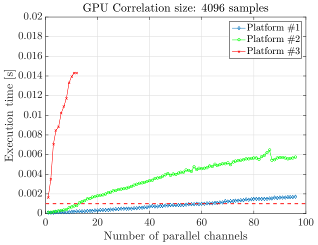
Table: Maximum number of real-time parallel channels for each platform using GPU accelerators.

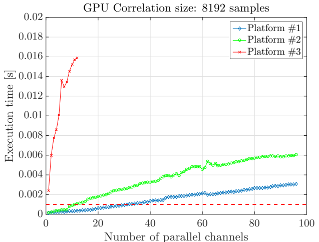
Correlator length	2048	4096	8192
Platform #1	65	45	25
Platform #2	12	11	10
Platform #3	1	0	0

Processing results with GPUs

3-correlator GPS L1 C/A channels with different sampling rates.







Conclusions

Conclusions (I/II)

- We described several parallelization techniques addressing computational efficiency at different abstraction layers.
- All those concepts were applied into a practical implementation available online under a free and open source software license.
- Building upon well-established open source frameworks and libraries, we showed that it is possible to achieve real-time operation in different computing environments.

Conclusions (II/II)

- Portability was demonstrated by building and executing the same source code in a wide range of computing platforms, from high-end servers to tiny and affordable computers, using different operating systems and compilers, and showing notable acceleration factors of key operations in all of them.
- As a practical outcome of the presented work, this paper introduced, to the best of authors' knowledge, the first free and open source software-defined GNSS receiver able to sustain real-time processing and to provide position fixes in ARM-based devices.

More info available online

Thank you for your attention!

Find out more at:

- Source code: `https://github.com/gnss-sdr/gnss-sdr`
- Webpage: `http://gnss-sdr.org`

Acknowledgements

AUDITOR - Advanced Multi-Constellation EGNSS Augmentation and Monitoring Network and its Application in Precision Agriculture.

This work has received funding from the European GNSS Agency under the European Union's Horizon 2020 research and innovation programme under grant agreement no. 687367.