Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2010, Article ID 180296, 4 pages
doi:10.1155/2010/180296

Research Article

Distributed Arithmetic for Efficient Base-Band Processing in Real-Time GNSS Software Receivers

Grégoire Waelchli,[1] Marcel Baracchi-Frei,[2] Cyril Botteron,[1] and Pierre-André Farine[1]

[1] *Ecole Polytechnique Fédérale de Lausanne, EPFL IMT ESPLAB, Rue A.-L. Breguet 2, 2000 Neuchâtel, Switzerland*
[2] *University of Neuchatel, Switzerland*

Correspondence should be addressed to Grégoire Waelchli, gregoire.waelchli@epfl.ch

Received 26 August 2009; Accepted 1 November 2009

Academic Editor: Periasamy K. Rajan

The growing market of GNSS capable mobile devices is driving the interest of GNSS software solutions, as they can share many system resources (processor, memory), reducing both the size and the cost of their integration. Indeed, with the increasing performance of modern processors, it becomes now feasible to implement in software a multichannel GNSS receiver operating in real time. However, a major issue with this approach is the large computing resources required for the base-band processing, in particular for the correlation operations. Therefore, new algorithms need to be developed in order to reduce the overall complexity of the receiver architecture. Towards that aim, this paper first introduces the challenges of the software implementation of a GPS receiver, with a main focus given to the base-band processing and correlation operations. It then describes the already existing solutions and, from this, introduces a new algorithm based on distributed arithmetic.

1. Introduction

With the increasing performance of modern processors, it becomes now feasible to implement a real-time multichannel GNSS receiver in software (i.e., where all the basic base-band operations such as the correlation are performed on a general purpose microprocessor). However, a major issue with the software approach is the large computing resources required for the base-band processing. To illustrate this issue, let us consider a conventional base-band GPS architecture as shown in Figure 1.

In this architecture, the incoming satellite signal $S(n)$ is sequentially processed at the system sampling rate F_S for (1) residual carrier $C(n)$ removal, (2) PRN code $T(n)$ removal, and (3) integration and dumping.

In addition to unavoidable load, and store operations, Table 1 provides a rough estimate of the amount of integer additions and multiplications per second necessary to process N_{Sat} satellites with the architecture of Figure 1 (without considering carrier and code generation).

From Table 1, a 12-channel receiver operating at $F_S = 4\,\text{MHz}$ requires approximately $3 \cdot 10^8$ additions and $4 \cdot 10^8$ multiplications to be executed each second. Consequently, as several former studies have concluded (see, e.g., [1]), a straightforward transposition of standard hardware-based architectures into software leads to an amount of real-time operations that can difficultly be managed by even today's fastest computers. In that sense, new algorithms or architectures have to be developed in order to minimize the computational load for the base-band processing, in particular for the correlation operations. To overcome this, two main strategies have been proposed in the literature. The first one relies on the use of Single Instruction Multiple Data (SIMD) operations while the second consists in exploiting the bitwise representation of the incoming signal. Both approaches are discussed hereafter.

2. Single Instruction Multiple Data (SIMD) Operations

In 1995, Intel introduced the first instance of SIMD under the name of MMX. The SIMD are mathematical instructions that operate on vectors of data and perform integer arithmetic on eight 8-bit, four 16-bit, or two 32-bit integers packed into an MMX register. Unlike standard *x*86 instructions, also sometimes referred as Single Instruction

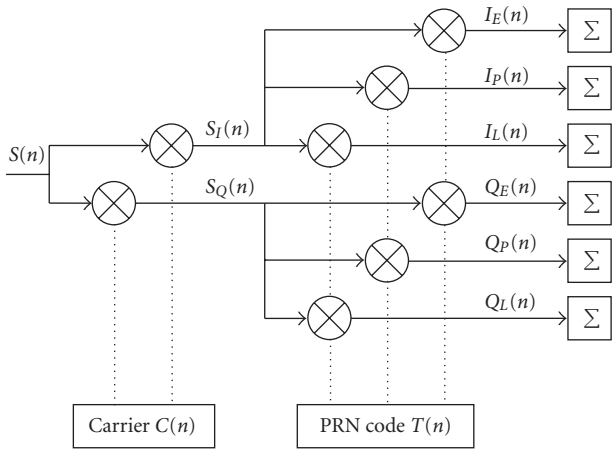

Figure 1: Standard base-band GPS architecture.

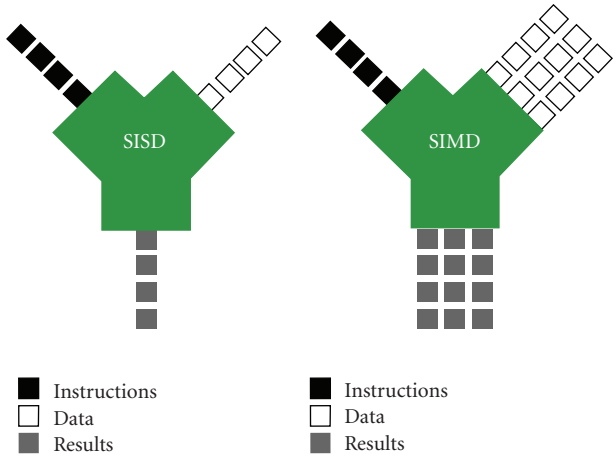

Figure 2: Comparison between SISD and SIMD [2].

Table 1: Amount of integer operations per second necessary to process N_{Sat} satellites.

	Number of additions	Number of multiplications
Carrier mixing	—	$2 \cdot N_{Sat} \cdot F_S$
Code mixing	—	$6 \cdot N_{Sat} \cdot F_S$
Accumulation	$6 \cdot N_{Sat} \cdot F_S$	—
Total	$6 \cdot N_{Sat} \cdot F_S$	$8 \cdot N_{Sat} \cdot F_S$

Single Data (SISD), the data are manipulated in blocks and a number of values can be loaded simultaneously, as illustrated in Figure 2.

On average, the SIMD operations require more clock cycles than the traditional *x*86 operations. However, since they operate on multiple integer values at the same time, SIMD operations can result in a significant gain in execution speed, especially for repetitive and parallel tasks like the base-band processing ones. Similarly, Digital Signal Processors (DSPs) can also offer great code optimization possibilities as some of them are capable of performing several multibit multiplications in parallel. However, both SIMD operations

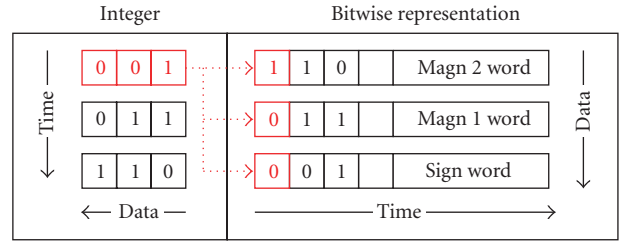

Figure 3: 3-bit integer versus bitwise data representation.

and DSP are tied to very specific hardware implementations which severely limit the portability of the code. In order to maximize the flexibility of the receiver, this kind of solutions is not further considered in this document.

3. Bitwise Processing

Contrary to the SIMD operations, the bitwise processing (sometimes also referred in the literature as vector processing) uses a universal CPU instructions set and exploits the native bit representation of the signal. The data bits are stored in separate vectors—generally one sign and one or several magnitude words—on which bitwise parallel operations can be performed independently. The objective is to take advantage of the high parallelism and speed of the bitwise operations for which a single integer operation is translated into a few simple parallel logical relations.

Consequently, while the integer arithmetic interprets the data horizontally as a single word, the bitwise processing manipulates the bits separately in a vertical way, as illustrated in Figure 3.

Many software receivers (see, e.g., [3, 4]) exploit the bitwise processing. Depending on the configuration, the code and carrier mixing can be carried out by a few basic logical relations in parallel on several samples, making the architecture particularly efficient. However, as the data bits are vertically spread over several sign and magnitude words, a reconversion into the integer representation is finally required to perform the accumulation and the data readout. This is done at the cost of numerous bitwise operations that needlessly increase the complexity. In conclusion, the inherent drawback of the bitwise processing is the lack of flexibility as the complexity becomes bit-depth dependant and may increase drastically with respect to the data quantization.

4. Distributed Arithmetic

The original concept of distributed arithmetic was developed for optimizing the implementation of digital filters into Field Programmable Gate Arrays (FPGAs). The main idea is to rearrange the multiplies and adds of a sum of products at the bit level to take advantage of small tables of precomputed sums (see, e.g., [5]). However this concept can also be adapted to a GNSS software receiver design in order to optimize the accumulations involved in the correlation process, as explained hereafter.

In Figure 1, the accumulation consists in summing up the consecutive samples $I_E(n)$, $I_P(n)$, and $I_L(n)$ and $Q_E(n)$, $Q_P(n)$, and $Q_L(n)$ over the integration period as:

$$I_{\Sigma,x} = \sum_{n=1}^{N} I_x(n), \quad x \in \{E, P, L\} \tag{1}$$

with N being the number of samples per integration.

For notation simplicity in this paper, we only provide equations for the in-phase (I) components (the same operations apply for the quadrature (Q) components). We also omit the index $x \in \{E, P, L\}$ of $I_{\Sigma,x}$ and $I_x(n)$.

Let us express the signal $I(n)$ as a linear combination of its M quantization bits. For the sake of simplicity, we consider here the two's complement notation which decomposes the signal as follows:

$$I(n) = -2^{M-1} \cdot I_{M-1}(n) + \sum_{m=0}^{M-2} 2^m \cdot I_m(n) \tag{2}$$

with I_m(n) being the mth bit of the signal $I(n)$ at the sampling instant n.

We define the partial sum P_m associated with the mth data bit of the signal $I(n)$ as:

$$P_m = \sum_{n=1}^{N} I_m(n), \quad m \in [0; M-1]. \tag{3}$$

By combining and rearranging the terms of the above two equations, we obtain:

$$I_\Sigma = -2^{M-1} \cdot P_{M-1} + \sum_{m=0}^{M-2} 2^m \cdot P_m. \tag{4}$$

The accumulation I_Σ is now expressed as a linear combination of M partial sums P_m. The challenge now consists in efficiently computing (3) for the M bits of $I(n)$. Since P_m is the arithmetic sum of all the bits contained in the word $I_m(n)$, it can be estimated by simply counting the number of bits equal to the logical value 1. Although some modern processors now propose an embedded instruction to perform this operation (which also limits the portability of the code), the most straightforward solution is to implement a Look-Up Table (LUT) that is directly addressed by the word itself and that outputs the corresponding partial sum P_m.

Thanks to the above distributed arithmetic implementation, the conversion from the bitwise representation into the integer one is performed in parallel to the accumulation. Furthermore the architecture can easily accommodate various signal configurations as the complexity stays almost proportional to the bit-depth of the incoming signal $I(n)$.

5. A GPS Implementation Example

Unlike the traditional bitwise approach, the distributed arithmetic requires the signal $I(n)$ to accumulate to be expressed as a linear combination of its M data bits (cf. (2)). This introduces an additional constraint on the former bitwise processing for carrier and code removal. To illustrate this aspect, let us consider the example of an incoming GPS satellite signal $S(n)$ digitized with 2 bits per sample $\{S_1(n), S_0(n)\}$ that are associated to the integer values ± 1 and ± 3.

Table 2: Possible integer output values of the carrier mixer.

Mixer output		Local carrier $C(n)$			
$S_I(n)$		1	2	−1	−2
Signal S(n)	1	1	2	−1	−2
	3	3	6	−3	−6
	−1	−1	−2	1	2
	−3	−3	−6	3	6

Table 3: Truth table of the carrier mixer.

Mixer output		Local carrier $\{C_1(n), C_0(n)\}$			
$\{S_{I,3}(n), S_{I,2}(n), S_{I,1}(n), S_{I,0}(n)\}$		00	01	10	11
Signal $\{S_1(n), S_0(n)\}$	00	0001	0010	1110	1101
	01	0011	0111	1100	1000
	10	1110	1101	0001	0010
	11	1100	1000	0011	0111

In the receiver of Figure 1, $S(n)$ is first mixed with a complex carrier $C(n)$ quantized with 2 bits $\{C_1(n), C_0(n)\}$ and associated to the integer values ± 1 and ± 2. The mixing of $S(n)$ and $C(n)$ results in the signals $S_I(n)$ that can take one of the integer values shown in Table 2.

An appropriate binary representations for $S_I(n)$ must be selected in order to minimize the number of bitwise operations necessary to perform the carrier and code mixing operations. In our example, $M = 4$ bits are needed to represent $S_I(n)$ in Table 2 if we use the following (heuristic) bits decomposition:

$$\begin{aligned} S_I(n) = {} & -6 \cdot S_{I,3}(n) + 3 \cdot S_{I,2}(n) \\ & + 2 \cdot S_{I,1}(n) + S_{I,0}(n). \end{aligned} \tag{5}$$

Using (5) and the sign and magnitude representation of $S(n)$ and $C(n)$, Table 2 can be encoded as shown in Table 3.

We can now translate the truth table of Table 3 into the following logical equations that can be carried out with 7 operations or even 6 by storing one intermediate result:

$$\begin{aligned} S_{I,3}(n) &= S_1(n) \oplus C_1(n), \\ S_{I,2}(n) &= S_{I,3}(n) \oplus (S_0(n) \cdot C_0(n)), \\ S_{I,1}(n) &= S_{I,2}(n) \oplus S_0(n) \oplus C_0(n), \\ S_{I,0}(n) &= \overline{S_{I,2}(n) \oplus C_0}. \end{aligned} \tag{6}$$

Following the carrier mixing operations (cf. Figure 1), the code mixing simply consists in a sign inversion, respectively, noninversion, which can be translated into an exclusive OR between the code $T(n)$ and all the respective signal bits $S_{I,m}(n)$:

$$I_m(n) = T(n) \oplus S_{I,m}(n), \quad m \in [0, 3]. \tag{7}$$

Table 4: Amount of operations per second necessary to process N_{Sat} satellites with the distributed arithmetic.

	Number of logical operations and integer additions	Number of LUT accesses
Carrier mixing	$12 \cdot N_{\text{Sat}} \cdot F_S / R$	
Code mixing	$24 \cdot N_{\text{Sat}} \cdot F_S / R$	
Accumulation	$24 \cdot N_{\text{Sat}} \cdot F_S / R$	$24 \cdot N_{\text{Sat}} \cdot F_S / R$
Total	$60 \cdot N_{\text{Sat}} \cdot F_S / R$	$24 \cdot N_{\text{Sat}} \cdot F_S / R$

With the above signal representation, the whole carrier and code mixing can be realized with 10 basic instructions that operate in parallel on 8, 16, 32, 64, or 128 bits depending on the CPU registers size R.

From each of the M words $I_m(n)$ obtained with (7), the partial sum P_m is calculated and summed up with the previous ones in order to form the final accumulation I_Σ expressed as (using (4)):

$$I_\Sigma = -6 \cdot \sum P_3 + 3 \cdot \sum P_2 + 2 \cdot \sum P_1 + \sum P_0. \quad (8)$$

As explained in the previous section, in order to save some operations, the partial sums are computed by the means of a LUT. The table must fit into the microprocessor cache to allow fast execution but must be also large enough to minimize the memory accesses.

6. Performances Comparison

The efficiency of the proposed distributed arithmetic depends on the register size R of the host computer (i.e., R bits can be processed in parallel). For the previous 2-bit data example, the total amount of operations becomes as shown in Table 4.

With respect to the integer base-band processing of Figure 1, the best improvement lies in the quasiabsence of integer multiplications, advantageously substituted by parallel logical operations. This way and assuming a 16-bit CPU and a 2-bit data quantization, the amount of integer additions to perform the correlation is reduced by almost 40%.

In comparison to the conventional bitwise processing, (2) may require more logical operations for the carrier and code mixing. On the other hand, no additional stage is needed to convert from bitwise into integer representation. Thus, in the case of a 2-bit data configuration as proposed in [6], the distributed arithmetic lowers the complexity by almost a factor two. It becomes even more efficient for higher signal bit-depths setup as the complexity grows almost proportionally with the data bit quantization (while it increases exponentially in the standard implementation such as in [6]).

7. Conclusion

As the software implementation of standard base-band architectures is not really suitable for real-time operation, new approaches must be developed. While the bitwise processing represents a very interesting and popular alternative, taking advantage of the parallelism and universality of the basic logical CPU instructions, it still suffers from a lack of flexibility and scalability with respect to the integer operations. On the other hand, the distributed arithmetic constitutes the perfect bridge between bitwise and integer representations, combining the efficiency of the first with the flexibility of the latter. The concept is simple and elegant. This paper has demonstrated that for a standard configuration (2-bit signal and carrier quantization), the amount of arithmetic operations is divided by nearly a factor of two with respect to both integer arithmetic and bitwise processing implementations. Finally, while our example only considers GNSS, it is also applicable to other types of receivers such as DS-CDMA ones.

References

[1] G. W. Heckler and J.-L. Garrison, *Architecture of Reconfigurable Software Receiver*, Purdue University, West Lafayette, Ind, USA, 2004.

[2] S. Charkhandeh, et al., "Implementaztion and testing of a real-time software-based GPS receiver for x86 processors," in *Proceedings of the ION National Technical Meeting (NTM '06)*, Monterey, Calif, USA, January 2006.

[3] B. M. Ledvina, et al., "Real-time software receiver," US patent application no. US0227856 A1, October 2006.

[4] A. Fridman and S. Semenov, *Architectures of Software GPS Receivers*, Springer, Berlin, Germany, 2000.

[5] R. Andraka and A. Berkun, "FPGAs make a radar signal processor on a chip a reality," in *Proceedings of the 33rd Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 559–563, Pacific Grove, Calif, USA, October 1999.

[6] B. M. Ledvina, M. L. Psiaki, S. P. Powell, and P. M. Kintner, "Bit-wise parallel algorithms for efficient software correlation applied to a GPS software receiver," *IEEE Transactions on Wireless Communications*, vol. 3, no. 5, pp. 1469–1473, 2004.