On the Computation/Memory Trade-Off in Software Defined Radios

Vincenzo Pellegrini, Mario Di Dio, Luca Rose and Marco Luise

Abstract—Since J. Mitola’s seminal work in the 90’s, Software Defined Radios (SDRs) have been a hot topic in wireless communication research. Though many notable achievements were reported in the field, the scarcity of computational power on general purpose CPUs has always been a limiting factor. If conveniently applied within an SDR context, classical concepts known in computer science as space/time trade-offs can prove helpful when trying to mitigate this problem. This paper presents a novel criterion to design signal-processing software in an SDR terminal that we call Memory Acceleration (MA). The key feature of MA is making extensive use of memory resources of the SDR platform in order to accelerate the most critical signal processing functions. MA provides substantial acceleration factors when applied to conventional SDRs without reducing their peculiar flexibility and appears particularly suited to the implementation of a fully-SW SDR on a general-purpose processor (GPP). As a case study for the application of MA, results about the implementation of the ETSI DVB-T [1] Viterbi decoder [2] are presented. In such a case, MA provides an acceleration factor of 10.4x with respect to a standard, purely-computational implementation while having no impact on error correction performance of the decoder and by making no use of any other typical performance enhancement technique (e.g. low level programming or parallel computation).

Index Terms—Software Radio, Signal Processing, Space/Time Trade-off, Memory Acceleration

I. MOTIVATION AND OUTLINE

Focusing on the specific field of SDR, it is well known that fully-SW solutions based on GPPs and with no dedicated digital processing HW (such as ASICS or FPGAs) are very attractive for research, development and small-scale market deployment owing to short development times. On the other hand, the drawbacks of such solution are their limited computational power or, seen from another perspective, their low throughput per Watt when compared to equivalent HW-accelerated implementations.

Aim of this paper is to prove that, by making use of all the resources available on a general purpose computing system (i.e. not only CPU time, but also memory), it is possible to bridge the gap in terms of computational speed and power efficiency that now exists between GPP-based and HW-accelerated SDRs.

This efficiency boost is based on revisiting classical concepts already known in computer science under the collective denomination of space/time tradeoffs. In previous literature, space/time tradeoffs are intended either as increasing the degree of HW/SW parallelism of a given implementation (therefore consuming more space) in order to reduce the execution time, as in [3] and [4], or as pre-calculating data produced by some well-determined algorithm into some tabular form [5] (again sacrificing space in terms of size of the table to be stored, to gain execution time). We tried to adapt this vision to the framework of SDR design, in order to provide a convenient and rather general SW design approach that may enable fast (re-)implementation of any radio signal processing chain in a memory-aware fashion.

II. MEMORY ACCELERATION

A. System representation and useful quantities

We start our discussion by observing that any radio system and, more generally, any system performing signal processing functions, can be represented as the interconnection of a number of constituent functional blocks. Simple systems are arranged as a straightforward cascaded “chain” of elementary blocks, more complicated schemes (possibly with feedback connections) look more like a “web” of components and connections. Focusing for simplicity on a radio receiver, whatever the web of blocks and connection is, the end-to-end signal processing function of our system is equivalent to a mathematical function $f(\ldots)$ which maps a certain amount of soft-valued input symbols (for instance those collected in a data frame or in a channel code block) into the corresponding hard-valued demodulated information bits. We call the minimum amount of soft channel symbols that can be processed independently from the remainder of the stream the Minimum Independent Data Set (MIDS). For the ETSI DVB-T [1] standard this would be 4 Orthogonal Frequency-Division Multiplexing (OFDM) frames (i.e. what is called a superframe in [1]). We indicate the size (number of items) of the MIDS with symbol $l$ and with $A$ the cardinality of the alphabet each input datum of the MIDS belongs to. The domain of $f(\ldots)$ is then defined as the set of all possible messages which can be represented within the MIDS. We call input space the domain of $f(\ldots)$ and $C_i$ the cardinality of such space. Then we have:

$$C_i = A^l$$  \hspace{1cm} (1)

If we could find a convenient analytical expression for the function $f(\ldots)$, we could consider implementing our sample DVB-T demodulator by programming such analytical expression into a computing system via any high-level programming language like C/C++. Still, this would be a computation-only implementation of the system. Like any classical HW or SW implementation of a radio system - such implementation would only take advantage of computational resources being available on a general purpose computing system, with very little attention to the memory resources that are available on the computing platform.
After this remark on memory resources, it would be natural to think of replacing function $f(\ldots)$ with a *tabular* implementation of $f(\ldots)$: a table $t(\ldots)$ containing, for each of the $A^l$ items of the overall input space, the associated output value. This would be a *memory-only* implementation, and would not require any real-time computation of $f(\ldots)$. On the other hand, the size $C_i$ of the table would not be practical for any memory technology available today or in the foreseeable future. The table $t(\ldots)$ could be filled up by running once forever at instantiation time (i.e., at the time of initialization or configuration of the terminal) the standard, computation-only, implementation of function $f(\ldots)$ over the entire input space. Such considerations suggest that the path towards optimal SDR implementations lies somewhere in between, with a hybrid approach that could use both computational and memory resources.

Let us now come back to the web representation of the signal processing functions of our SDR. We call this representation the 0-step of a general approach that we labeled *algorithm segmentation*. Such 0-step may be the direct translation of the signal processing functions described in a standard or in a reference implementation. The underlying assumption of this decomposition is that each of the functional blocks $f_n(\ldots)$ in the web is *atomic*, i.e., impossible to break-up in a further web of constituent algorithmic functional blocks. On the contrary, the aim of our *algorithm segmentation* approach is just coming to a decomposition of a functional block $f_n(\ldots)$, formerly assumed to be atomic, into a chain (or web) of constituent sub-blocks $f_{n,p}(\ldots)$ whose end-to-end behavior is equivalent to the original function $f_n(\ldots)$. One advantage of this is that the input spaces of sub blocks $C_{i_{n,p}}$ will be different from and significantly smaller than $C_{i_n}$, provided that algorithm segmentation is performed correctly. Algorithm segmentation cannot be considered as a form of algorithm re-design: as a consequence, algorithm segmentation *does not change the overall computational cost of the segmented algorithm*. An expedient visual representation of the SDR signal processing web is obtained as follows: we call $W_n$ the computational cost of the $n$-th functional block and we use a graphical representation of the SDR in which the size of the functional block is directly proportional to such cost, see figure 1. This gives at a glance an indication of the relative computational weight of each block (function) within the whole terminal. Let us also introduce the symbol $W_m$ as the total computational cost of *memory management* for table $t_m(\ldots)$, something that has nothing to do with algorithm complexity, but that represents the cost of memory address calculation and the memory access latency (if significant). The latter parameter can be made equivalent to a computational cost by reducing it to CPU time or equivalent flops/ops/clock cycles.

**B. Acceleration Design**

We come now to the core of the MA technique. Broadly speaking, the aim of MA is aiding the GPP in processing the informative signal through the proper (extensive) usage of memory resources. Such result is obtained by convenient replacement of the functional blocks $f_n(\ldots)$ that followed a purely-computational approach (with marginal usage of memory), with pre-computed tables $t_m(\ldots)$ as introduced in II-A. The replacement is done after one or more steps of algorithm segmentation are carried out, according to a Recursive Table Aggregation Rule (RTAR) which will be described in the following subsection. Before going on, let us also remark that the *input space cardinality* $C_{i_n}$ of each $f_n(\ldots)$ is assumed to be *independent of its computational cost* $W_n$ or, in the worst case, weakly correlated. Think for example of three typical signal processing blocks being present in any modern radio system: a block-based Forward Error Correction (FEC) decoder, its associated interleaver and a scrambler preforming energy dispersal on a set of data whose size is the same as for the FEC block. Such three blocks share the very same $C_i$, but yield enormously different computational costs (i.e. at least one order of magnitude), with the FEC decoder being dramatically heavier than the other two. As long as the amount of memory resources $[M]$ is finite, it is necessary that those resources are used to accelerate the computationally-heaviest blocks of the chain. Performing memory-acceleration of a low $W_n$ block would only waste resources, as long as the memory used to replace computation yielded by such block could be better used in order to replace an heavier block. The MA process reaches optimum configuration whenever the available memory space is exhausted, and the maximum possible number of operations (or the maximum possible amount of CPU time) has been replaced by memory look-ups providing the same output.

<table>
<thead>
<tr>
<th>Table 1: MA Symbols and Taxonomy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symbol</strong></td>
</tr>
<tr>
<td>$f_n(\ldots)$</td>
</tr>
<tr>
<td>$f_n, m(\ldots)$</td>
</tr>
<tr>
<td>$t_m(\ldots)$</td>
</tr>
<tr>
<td>$I$</td>
</tr>
<tr>
<td>$A$</td>
</tr>
<tr>
<td>$C_{i_n}$</td>
</tr>
<tr>
<td>$W_n$</td>
</tr>
<tr>
<td>$W_m$</td>
</tr>
<tr>
<td>$W_{m(m)}$</td>
</tr>
<tr>
<td>$M_m$</td>
</tr>
<tr>
<td>$S_m$</td>
</tr>
<tr>
<td>$\alpha$</td>
</tr>
<tr>
<td>$\eta$</td>
</tr>
</tbody>
</table>

Fig. 1. Computational cost weighted functional block representation. Blocks 1 and 4 are peripheral.
C. Recursive Table Aggregation Rule

Assuming that we have an atomic web decomposition of our end-to-end algorithm, what is the optimum level of break-up to replace computational-intensive blocks with tables? We tried to give an answer to this fundamental question through the RTAR. Considering that each implemented table yields, at least, a look-up act in order to provide its pre-stored output, we conclude that minimizing \( W_m \) necessarily requires to aggregate as many functional blocks \( f_n(\ldots) \) as possible into a single table \( t_m(\ldots) \). On the other hand, we have to face the limitation of the memory resources that prevents us from letting the tables grow beyond a reasonable size. Our criterion will also preserve memory-contiguity of information that is used contiguously in time, which determines cache-friendliness of generated tables.

We call table boundary (TB) the closed line delimiting the subsystem we intend to memory-accelerate at a certain step of RTAR. Interfaces from and towards the remainder of the system are given by the system chain connection arrows that do cross the TB. A block is called peripheral if all of its input or all of its output connections cross the table boundary. RTAR is applied as follows:

1) Define whole radio as sub-system to be memory-accelerated. This is equivalent to enclosing the entire radio within the table boundary. Calculate \( SC_i \). If the obtained table fits in memory, then go to step 3), otherwise consider the 0-step of algorithm segmentation. The table boundary now encloses all blocks of the SDR broken down to the block level.

2) Identify the computationally-lightest block contained within the table boundary and release it by moving it outside the table boundary (figure 2). If the released block is not peripheral, then release all blocks depending on its output, see figure 4. Calculate \( SC_i \) again, if table fits then go to step 3), otherwise iterate 2) until either i) the table fits, or ii) the \( f_n(\ldots) \) atomicity limit is reached (figure 3). If the latter is true, perform a further algorithm segmentation step over \( f_n(\ldots) \) and restart iterating step 2). Note that, in the case atomicity limit is reached, the block which will undergo algorithm segmentation is the heaviest block of the whole radio, then the sub-blocks obtained from segmentation collectively yield the majority of the computational cost of the SDR, subsequent iterations will thus be performed leaving the table boundary around such sub-blocks without re-initializing. Whenever one of the sub-blocks obtained from algorithm segmentation is released, check whether the table boundary encloses a computational cost \( W_{TB} \) which is greater than the cost of any functional block outside the TB. In case this condition becomes false, re-initialize the TB to enclose the entire system and iterate step 2).

3) Implement the subsystem being enclosed in the current table boundary by substituting its computation-only functional blocks with a suitable table \( t_m(\ldots) \). Table \( t_m(\ldots) \) will be an input/output map completely equivalent to the replaced subsystem. If there are still some blocks not yet implemented in memory, and memory resources are not exhausted, initialize the table boundary for next iteration by enclosing all the remaining blocks of the radio system, then go to step 2).

The proposed RTAR rule is admittedly sub-optimal - an exhaustive approach to algorithm segmentation to find the absolute optimal configuration appears unfeasible. Nonetheless, we believe that our rule captures the majority of the achievable MA gain with a manageable approach. In some test cases (conducted upon rather heterogeneous signal processing algorithms), it was shown to provide substantial speedup factors (roughly one order of magnitude) with an acceptable MA design effort.

Cache-friendliness provided by RTAR also constitutes the basis for MA compatibility with parallel programming. Memory access contentions that could indeed happen when loading the required memory table (or table portion) from the external Random Access Memory (RAM) into the core-dedicated caches of a multicore computing system are made extremely rare by performing most of the look-ups within the cache, therefore minimizing the number of fetches being necessary from the RAM.
In practice, it turns out that functional blocks that do perform several different sub-functions will have a large \( l \) and will require segmentation in order to be (even partially) implemented in memory. Nothing is being assumed at this stage concerning the computational cost of the functional blocks, and we are not stating the presence of any kind of correlation between the input space cardinality \( C_i \) and the computational cost of a block. As radio signal processing algorithms do differ very much from one another, they do offer very different opportunities for algorithm segmentation. It is therefore difficult to give optimality bounds for algorithm segmentation into an MA context - still, we can say that the best algorithm segmentation is the one providing the finest possible granularity of input spaces of the obtained sub blocks. This is true because the smaller the granularity is, the closer the RTAR will manage to bring the total memory occupancy of the MA-ed SDR, \( \sum_{m=0}^{N_t-1} M_m \), to the memory capacity of the system \( M \). Broadly speaking, the more sub-blocks \( N_{sb} \) algorithm segmentation obtains from the given block, the better algorithm segmentation was performed.

To sum up, the joint action of algorithm segmentation and RTAR is to i) decompose the given SDR system down to the finest possible level of computational granularity; ii) generate a re-implementation which uses the available memory resources in order to perform as much computation as possible by means of memory look-ups and iii) do it with the smallest possible computational cost of memory management. This is the gist of the MA concept.

\section*{E. Some MA Analytics}

We define \( \eta \) as the acceleration efficiency for functional block \( f(...) \):

\[
\eta = \frac{\sum_{n=0}^{N_{sb}-1} W_n - \sum_{m=0}^{N_t-1} W m_m}{\sum_{m=0}^{N_t-1} M_m} \quad (2)
\]

where \( N_{sb} \) is the number of the obtained sub-blocks \( f_n(...) \) that will be implemented in memory through the use of suitable tables, and \( N_t \) is the number of tables that will be used to produce such an implementation. In other words, the acceleration efficiency \( [\eta] \) is the ratio between the computational effort being saved by means of the resulting memory-based implementations, reduced by the amount of computational work needed for table management, and the total memory footprint being required. A negative value for \( \eta \) indicates that the chosen MA design will reduce system performance. Once the acceleration process is completed, it is possible to calculate the obtained acceleration factor \( a \) as

\[
a = \frac{W_T + \sum_{n=0}^{N_{sb}-1} W_n}{W_T + \sum_{m=0}^{N_t-1} W m_m} \quad (3)
\]

where \( W_T \) accounts for the computational cost of the remaining blocks which were not implemented in memory.

As previously stated, different algorithms offer different opportunities for segmentation. The consequence of this statement is that it is very difficult to give an upper bound for
a. Still, an estimate depending on the number of tables \( N_t \) that the RTAR obtains from the segments \( N_{ab} \) the given radio system \( f(... \) was broken into by algorithm segmentation can be found as follows. Assuming that the whole SDR \( f(...) \) fits into the available memory, we have:

\[
  a_{\text{max}} = \frac{\sum_{n=0}^{N-1} W_n}{\sum_{m=0}^{N_t-1} L_m + (i_m - 1)(x + \sigma)} \tag{4}
\]

where \( L_m \) is the access latency for each table (that depends on the table size and on the chosen implementation platform), \( i_m \) is the number of inputs to each table, \( x \) is the computational cost for one multiplication by a constant, and \( \sigma \) is the computational cost of one sum with a variable. All such quantities, including \( L \), can be expressed in terms of number of elementary operations as well as of required CPU time. The denominator general term \( L_m + (i_m - 1)(x + \sigma) \) is indeed our estimate for \( W_{mn} \).

F. Performance results

We give now examples for application of MA over a GPP platform. On a standard Intel Q9400, 2.66GHz CPU, the computation-only C/C++ implementation of our ETSI DVB-T [1] Viterbi decoder [2] takes about 7.71 times the realtime. After undergoing memory acceleration, the same implementation takes 0.74 times the realtime, yielding an acceleration factor \( a = 10.4 \). The total memory occupancy of the memory-accelerated implementation is 50.0 MiB. On the same computational platform, the OFDM time/frequency offset estimator described by van de Beek, Sandell and Borjesson in [6] was accelerated through MA by a factor \( a = 12 \), with a memory occupancy of 256 MiB. These two memory-accelerated modules enabled us to obtain a completely real-time, fully software ETSI DVB-T receiver on a low budget general purpose CPU. To the knowledge of the authors, such implementation result was not achieved before and was described in [7]. The total memory occupancy of the SDR is not critical within the context of modern GPP platforms.

Our performance test results were obtained by compiling source code with g++ compiler, version 4.3.2 – 7. Due to space constraints, it is not possible to detail here the MA-driven modifications to the classical implementations which led to such acceleration factors. Accurate description of the two MA implementations will be material for a future Journal paper.

III. Conclusions and perspectives

Based on the results obtained by applying the MA approach to the DVB-T demodulator [7] (and in particular to two very different radio signal processing algorithms), we believe the MA technique is effective in providing substantial performance boost to GPP-based SDR systems. The point in our approach that deserves further research is in our opinion a more systematic technique to identify the optimal segmentation of the single radio signal processing block. The performance boosts of at least one order of magnitude that we could obtain can make an SDR terminal more energy-efficient without any impact on its inherent flexibility/reconfigurability features.

The acceleration factors presented within this work are obtained by using both computing architectures and compilers (GNU g++) that are totally unaware of the MA approach and therefore neglect memory access optimization in favor of pure computation. It is thus expected that applying MA on computational back-ends that take into account memory management and access optimization would result in even bigger acceleration factors.

Though developed for general purpose CPUs, Memory Acceleration can be also easily applied to other computational architectures, including for example multicore Digital Signal Processors (DSPs).

REFERENCES