A Real-Time Compressed Sensing-Based Personal Electrocardiogram Monitoring System

Karim Kanoun, Hossein Mamaghanian, Nadia Khaled and David Atienza
School of Engineering (STI)
Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland; E-mail: {name.surname}@epfl.ch

***Abstract*—Wireless body sensor networks (WBSN) hold the promise to enable next-generation patient-centric mobile-cardiology systems. A WBSN-enabled electrocardiogram (ECG) monitor consists of wearable, miniaturized and wireless sensors able to measure and wirelessly report cardiac signals to a WBSN coordinator, which is responsible for reporting them to the tele-health provider. However, state-of-the-art WBSN-enabled ECG monitors still fall short of the required functionality, miniaturization and energy efficiency. Among others, energy efficiency can be significantly improved through embedded ECG compression, which reduces airtime over energy-hungry wireless links. In this paper, we propose a novel real-time energy-aware ECG monitoring system based on the emerging compressed sensing (CS) signal acquisition/compression paradigm for WBSN applications. For the first time, CS is demonstrated as an advantageous real-time and energy-efficient ECG compression technique, with a computationally light ECG encoder on the state-of-the-art Shimmer^{TM} wearable sensor node and a real-time decoder running on an iPhone (acting as a WBSN coordinator). Interestingly, our results show an average CPU usage of less than 5% on the node, and of less than 30% on the iPhone.**

I. Introduction

According to the World Health Organization, cardiovascular diseases are the number one cause of death worldwide, responsible for an estimated 17.1 million deaths in 2004 (i.e., 29% of all deaths worldwide) and economic fallout in billions [1]. These increasingly prevalent cardiac diseases are requiring escalating levels of supervision and medical management, which are contributing to skyrocketing health care costs and, more importantly, are unsustainable for traditional health care infrastructures. Wireless body sensor networks (WBSN) promise to allow inexpensive, continuous and remote health monitoring for next-generation of ambulatory personal tele-cardiology or e-cardiology systems. Outfitting patients with wearable, miniaturized and wireless sensors able to measure and wirelessly report cardiac signals to tele-health providers would enable the required personalized, real-time and long-term ambulatory monitoring of chronic patients, its seamless integration with the patient's medical record and its coordination with nursing/medical support.

While the resting electrocardiogram (ECG) monitoring is standard practice in hospitals, its ambulatory counterpart is still facing many technical challenges. For instance, the 3-lead ECG is still nowadays recorded on data-logging (Holter) devices during 1 to 5 days of normal daily activities of a patient. These systems, currently commercialized by GE health care, Sorin Group, Mortara and Philips health care, suffer from important limitations: limited autonomy, bulkiness and no or limited wireless connectivity. Recently, the realization of wireless-enabled ultra-low-power ECG monitors for ambulatory use has been receiving significant industrial and academic interest [2], [3], [4]. Yet, these state-of-the-art ECG monitors fall short in terms of either clinical relevance and/or autonomy figure, primarily because they transmit uncompressed ECG data over power-hungry wireless links. It is today widely acknowledged that the achievement of truly WBSN-enabled personal ECG monitoring systems requires more breakthroughs not only in terms of ultra-low-power read-out electronics and radios, but also and increasingly so, in terms of embedded ECG compression and feature extraction algorithms, and assorted ultra-low-power dedicated digital processors.

ECG compression relies on the *sparse* (and thus compressible) nature of the ECG, as it can be approximated by a compact representation in the wavelet domain [5]. Capitalizing on this sparsity, we propose to apply the emerging compressed sensing (CS) approach [6], [7] for a low-complexity, real-time and energy-aware ECG signal compression on WBSN motes. This is motivated by the observation that this new signal acquisition and compression paradigm is well suited for low-power implementations since it dramatically reduces the need for resource-intensive (both processing and storage) DSP operations on the encoder side.

CS is a new sensing and processing paradigm, which challenges the traditional analog-to-digital (ADC) conversion based on the Shannon sampling theorem. The latter theorem states that, given a signal of bandwidth Ω, it is sufficient to sample it at 2Ω samples per second (i.e., the Nyquist rate) to ensure faithful representation and reconstruction. For sparse signals such as the ECG, (above) Nyquist-rate sampling produces a large amount of redundant digital samples, which are costly to wirelessly transmit and require to be further compressed using non-linear digital techniques. If one sets course to design energy-aware embedded ECG sensors, it is desirable to reduce the number of acquired ECG samples by

This research has been partially funded by the Nano-Tera.ch NTF Project BioCS-Node, which is financed by the Swiss Confederation. 978-3-9810801-7-9/DATE11/ ©2011 EDAA

taking advantage of the sparsity, or the reduced "information rate" of the ECG signal. CS is a methodology recently proposed to address this problem. The main idea behind CS is relatively simple. Suppose the considered signal has a (potentially large) bandwidth Ω but has a sparse approximation, i.e., it can be represented by a linear superposition of K elements of a dictionary, with $K << \Omega$. CS states that you only need to collect roughly K samples (<< Nyquist rate) using simple measurement waveforms, thus sensing/sampling and compressing at the same time. The price to pay for these advantages is a more complex decoder, which recovers the original signal by solving a convex optimization problem. Many algorithms were introduced to solve the CS reconstruction problem, e.g., interior-point algorithms [8], gradient projection [9], iterative thresholding [10], and greedy approaches such as orthogonal matching pursuit (OMP) [11]. In general, reconstruction algorithms for solving the inverse problem are computationally expensive and involve large and dense matrix operations, which prevents the real-time implementation on embedded platforms.

In this paper, we use the fast iterative shrinkage-thresholding algorithm (FISTA) [12], a popular method for CS recovery. The main contributions of this work are: (1) a novel CS approach that precludes large and dense matrix operations both at compression and recovery, and (2) several platform-dependent optimizations and parallelization techniques to achieve real-time CS recovery. To the best of our knowledge, this work is the first to demonstrate CS as an advantageous real-time and energy-efficient ECG compression technique, with a computationally light and energy-efficient ECG encoder on the state-of-the-art Shimmer™ wearable sensor node and a real-time decoder running on an iPhone (acting as a WBSN coordinator).

The rest of this paper is organized as follows. Section II introduces our proposed ECG compression and reconstruction scheme. Section III details the performance metrics and ECG database used for its evaluation. Section IV details our embedded implementations. Then, Section V presents a discussion of our results and main conclusions.

Notation: In all the following, normal letters designate scalar quantities, boldface lower-case letters indicate column vectors, and boldface capitals represent matrices. Moreover, m_i and $\mathbf{M}_{i,j}$ are the i^{th} entry of vector $\mathbf{m}$ and the $(i,j)^{th}$ entry of matrix $\mathbf{M}$, respectively. Finally, $(.)^H$ and $||.||_p$ denote the conjugate transpose, and the l_p-norm of a vector, respectively.

II. Methods

This work presents a complete real-time implementation of, on the one hand, a CS-based ECG compression/encoding on the embedded Shimmer™ wireless mote, and on the other hand, the corresponding ECG reconstruction on an iPhone. On the encoder side, the compression algorithm consists of the three processing stages depicted in Figure 1: a linear transformation is first applied to the original ECG signal, followed by an inter-packet "redundancy removal" stage, and an entropy encoding algorithm (i.e., *Huffman Coding*) is finally applied which outputs the compressed signal to be wirelessly transmitted. Correspondingly, the decoder consists of three stages as seen in Figure 1: the input codes are decoded using the same codebook used in the encoder side, followed by a packet reconstruction stage which re-inserts the inter-packet redundancy removed during the encoding, and finally the FISTA CS reconstruction algorithm is applied to find the best estimation of the original ECG.

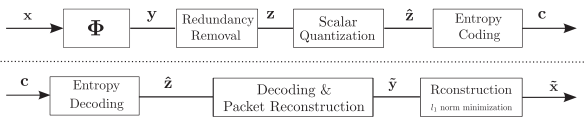


Fig. 1. *Block diagram of the ECG compression scheme on the Shimmer™ wireless mote (top) and the reconstruction scheme on the iPhone (bottom)*

A. *CS-Based ECG Compression Algorithm*

As aforementioned, the original ECG signal $\mathbf{x}$ has a sparse approximation, i.e., it can be represented by a linear superposition of S elements of an orthonormal wavelet basis $\mathbf{\Psi} = [\boldsymbol{\psi}_1|\boldsymbol{\psi}_2|\cdots|\boldsymbol{\psi}_N]$, as follows $\mathbf{x} = \mathbf{\Psi}\boldsymbol{\alpha}_S$, where $\boldsymbol{\alpha}_S$ represents the N-dimensional coefficient vector with only S non-zero entries ($S \ll N$). Accordingly, CS shows that it is sufficient to collect roughly S samples using simple *analog* measurement waveforms, thus sensing/sampling and compressing at the same time. Moreover, by merging the sampling and compression steps, CS removes a large part of the digital architecture. This so-called "analog CS", where the compression occurs in the analog sensor read-out electronics prior to ADC conversion is our ultimate goal. Its demonstration still requires extensive work on the analog sensor read-out electronics. Consequently, in the present work, we propose to approach it through "digital CS", where the linear CS compression is applied after the ADC.

Accordingly, as depicted in Figure 1 (top), we collect M samples using simple measurement vectors $\{\boldsymbol{\phi}_i\}_{1\leq i\leq M}$ as $y_i = \boldsymbol{\phi}_i^H\mathbf{x}$, $i = 1,\cdots,M$. Consequently, the CS linearly compressed data vector $\mathbf{y} \in \mathbb{R}^M$ is described by: $\mathbf{y} = \mathbf{\Phi}\mathbf{x}$, where $\mathbf{\Phi}$ denotes the $M \times N$ measurement or sensing matrix with the vectors $\boldsymbol{\phi}_1^H,\cdots,\boldsymbol{\phi}_M^H$ as rows. Notice that the sensing matrix $\mathbf{\Phi}$ does not depend on the signal. To guarantee robust and efficient recovery of the S-sparse signal $\boldsymbol{\alpha_S}$, the sensing matrix $\mathbf{\Phi}$ must obey the key *restricted isometry property* (RIP) [13], [14]:

$$(1-\delta_S)\,||\boldsymbol{\alpha}||_2 \leq ||\mathbf{\Phi}\mathbf{\Psi}\boldsymbol{\alpha}||_2 \leq (1+\delta_S)\,||\boldsymbol{\alpha}||_2\,, \qquad (1)$$

for all S-sparse vectors $\boldsymbol{\alpha}$. δ_S is the isometry constant of matrix $\mathbf{\Phi}$, which must be not too close to one. A universal good choice for the sensing matrix $\mathbf{\Phi}$ are random matrices, such as random matrices with independent identically distributed (i.i.d.) entries formed by sampling (1) a Gaussian distribution $\mathcal{N}(0,1/N)$; (2) a symmetric Bernoulli distribution $(P(\mathbf{\Phi}_{i,j} = \pm 1/\sqrt{N}) = 1/2)$. As we show later on, many efficient sensing matrices can be constructed with simple pseudo-random design that can be implemented using a surprisingly small amount of on-board memory and computation.

B. CS Reconstruction Algorithm

If the RIP holds, then accurate reconstruction can be accomplished in the presence of measurement noise by solving the following convex optimization problem:

$$\min_{\tilde{\boldsymbol{\alpha}} \in \mathbb{R}^N} ||\tilde{\boldsymbol{\alpha}}||_1 \qquad \text{subject to} \qquad ||\boldsymbol{\Phi\Psi}\tilde{\boldsymbol{\alpha}} - \mathbf{y}||_2 \leq \sigma, \tag{2}$$

where $||\tilde{\boldsymbol{\alpha}}||_1$ stands for the sum of the absolute values of the components of $\tilde{\boldsymbol{\alpha}}$, and σ bounds the amount of noise corrupting the data. The l_1 norm is used to induce the sparsity in the optimal solution. The problem in (2) can alternatively be cast as the following *non-smooth* convex optimization problem:

$$\min_{\tilde{\boldsymbol{\alpha}} \in \mathbb{R}^N} \{F(\tilde{\boldsymbol{\alpha}}) \equiv \underbrace{||\boldsymbol{\Phi\Psi}\tilde{\boldsymbol{\alpha}} - \mathbf{y}||_2^2}_{f(\tilde{\boldsymbol{\alpha}})} + \underbrace{\lambda ||\tilde{\boldsymbol{\alpha}}||_1}_{g(\tilde{\boldsymbol{\alpha}})}\} \tag{3}$$

where $f(.)$ and $g(.)$ are convex functions, and $g(.)$ is non-smooth. In most applications, this problem is large scale and involves dense matrices, which often precludes the use of the sophisticated interior point methods. This motivated the search for simpler gradient-based algorithms to solve (3), using relatively cheap matrix-vector multiplication. One of the most popular methods for solving the reconstruction problem (3) belongs to the class of *iterative shrinkage-thresholding* algorithms (ISTA), where each iteration involves a matrix-vector multiplication followed by a shrinkage/ soft-thresholding step [10]. ISTA is however notoriously slow. Therefore, different accelerations of ISTA have been proposed [15], [12]. Among these accelerations, we herein use the FISTA algorithm developed in [12]:

FISTA with constant step size
Input : $L = L(f)$ – a Lipschitz constant of ∇f
Step 0 : Take $\mathbf{y}_1 = \tilde{\boldsymbol{\alpha}}_0 \in \mathbb{R}^n, t_1 = 1.$
Step k : $(k \geq 1)$ Compute

$$\tilde{\boldsymbol{\alpha}}_k = \text{prox}_{t_k}(g)(\mathbf{y}_k - \frac{1}{L}\nabla f(\mathbf{y}_k)) \tag{4}$$

$$t_{k+1} = \frac{1 + \sqrt{1 + 4t_k^2}}{2} \tag{5}$$

$$\mathbf{y}_{k+1} = \tilde{\boldsymbol{\alpha}}_k + (\frac{t_k - 1}{t_{k+1}})(\tilde{\boldsymbol{\alpha}}_k - \tilde{\boldsymbol{\alpha}}_{k-1}) \tag{6}$$

The two general steps of FISTA are (4) and (6), and the simplicity of the algorithm depends on the ability to compute the ”prox” operation. Since $g(\mathbf{x}) = \lambda ||\mathbf{x}||_1$, the ”prox” operation is the same as a simple soft thresholding [15]. Therefore, while the convergence of the original ISTA to the solution is in the order of $\simeq O(1/k)$, k being the iteration number, FISTA converges faster at rate $O(1/k^2)$ [12].

III. ECG Database and Performance Metrics

To validate the performance of the considered compression schemes, we use the MIT-BIH Arrhythmia Database [16], [17] that is the most commonly used database for the comparative study of ECG compression algorithms. This database contains 48 half-hour excerpts of two-channel ambulatory ECG recordings, obtained from 47 subjects studied by the BIH Arrhythmia Laboratory. The recordings were digitized at 360 samples per second per channel with 11-bit resolution over a 10 mV range.

Moreover, to quantify the compression performance while assessing the diagnostic quality of the compressed ECG records, we employ the two most widely used performance metrics, namely *the compression ratio* (CR) and *percentage root-mean-square difference* (PRD). CR is defined as:

$$CR = \frac{b_{orig} - b_{comp}}{b_{orig}} \times 100, \tag{7}$$

where b_{orig} and b_{comp} represent the number of bits required for the original and compressed signals, respectively. The PRD, and associated signal-to-noise ratio (SNR), quantifies the percent error between the original signal vector $\mathbf{x}$ and the reconstructed $\tilde{\mathbf{x}}$:

$$PRD = \frac{||\mathbf{x} - \tilde{\mathbf{x}}||_2}{||\mathbf{x}||_2} \times 100; \quad SNR = -20\log_{10}(0.01PRD).$$

IV. Our Real-Time Personal ECG Monitor

In this section we describe the used embedded platforms, and report the platform-dependent optimizations we perform on the two algorithms of Section II to achieve real-time execution and optimized memory footprint.

A. Real-Time CS-Based ECG Compression on Shimmer™

1) The Shimmer™ embedded platform: Our target platform is the Shimmer wireless sensor mote [3]. From the hardware viewpoint, the Shimmer™ mainboard includes the low-power Texas Instrument 16-bit MSP430F1611 microcontroller and a Bluetooth module. The MSP430 microcontroller runs at 8 MHz, has 10 kB of RAM, 48 kB of Flash and includes a fast hardware multiplier, but does not include a floating-point unit. The mainboard also includes a Micro SD slot supporting up to 2 GB of Flash memory for data storage, and is powered by a rechargeable Li-polymer battery. Since our aim is to comparatively study the two compression algorithms on the standard MIT-BIH arrhythmia database, we used the Shimmer™ serial port to read in the database records re-sampled at 256 Hz, and to readout the encoded ECG data. From the software viewpoint, we used the open-source GCC 3.2.3 tool chain for the MSP430 [18] to generate the binaries.

2) Real-time CS-Based ECG compression: The implementation of Gaussian random sensing with matrix $\boldsymbol{\Phi} \in \mathbb{R}^{M \times N}$, requires the implementation of a Gaussian-distributed random number generator on the embedded platform and the computation of a large matrix multiplication. This is too complex, time consuming, and not real-time for both the encoder and decoder. Thus, we explored three different approaches to the implementation of the random sensing matrix $\boldsymbol{\Phi}$: (1) We implemented an 8-bit quantized version of a normal random number generator to form $\boldsymbol{\Phi}$; the normal random number generator on-board still made this approach not real-time. (2) We also circumvented the on-board generation of the normal random numbers by storing them on the platform; the large dense matrix multiplication still remained a main bottleneck. (3) We introduced an innovative approach to CS implementation using a sparse binary sensing matrix $\mathbf{\Phi}$, i.e., *sparse binary sensing*.

In a sparse binary matrix $\mathbf{\Phi}$, each column has exactly d nonzero entries equal to $1/\sqrt{d}$, with $d \ll N$. The positions of the d non-zero elements are randomly chosen to keep the incoherence between the columns of the sensing matrix. Obviously, the choice of the number of non-zero elements depends on the sparsity of the signal. For this sensing matrix, the RIP property of (1) is not valid. However, such a sensing matrix satisfies a different form of this property, so-called RIP_p property [19], which was shown to be sufficient to guarantee a good sparse approximation recovery by a linear program. Since sparse sensing matrices are amenable to very fast and efficient implementation of the large matrix multiplication required by the CS, we herein explore the use of sparse sensing matrices to decrease execution time. Figure 2 shows the average output SNR vs. CR for sparse binary sensing with $d = 12$ on the MSP430 and the optimal Gaussian sensing on Matlab. The results obtained validate that there is no meaningful performance difference between the two approaches. $d = 12$ was identified as the minimum value that the optimal trade-off between execution time (a 2-second vector is now CS-sampled in 82 ms) and (signal) recovery/reconstruction error.

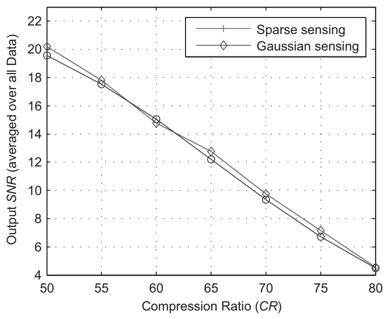


Fig. 2. *Performance benchmarking of sparse binary CS*

The use of a fixed binary sensing matrix, combined with the quasi-periodic nature of the ECG signal, yields to very similar consecutive measurement vectors $\mathbf{y}$. So, a large inter-packet redundancy exists, which must be removed prior to encoding and wirelessly transmit. Hence, the redundancy removal module computes the difference between consecutive vectors, and only this difference is further processed. At the end, a state-of-the-art entropy coding module is used for further compression. Since the range of the difference signal just before encoding is between $[-256 : 255]$, a complete Huffman codebook of size 512 is needed with a maximum codeword length of 16 bits, for a given compression ratio. The storage of the offline-generated codebook requires 1 kB for the codebook itself and 512 B for its corresponding codeword lengths. The complete CS implementation requires 6.5 kB of RAM and 7.5 kB of Flash, 1.5 kB of which are for Huffman codebook storage.

B. Real-Time ECG reconstruction optimization on the iPhone

Our target platform is the iPhone 3GS mobile phone, which includes an ARM Cortex$^{\text{TM}}$-A8 processor [20]. This architecture supports multi-thread execution and single instruction multiple data (SIMD) instructions, which we exploit to achieve real-time operation of the CS-based ECG reconstruction.

1) High-Level Application Partitioning: We have developed a multi-thread application using the producer-consumer paradigm to display and decode the ECG. The first thread manages the Bluetooth connection using BTStack [21], decodes the data and stores 2 sec. of ECG (i.e., 512 sampled values) into a shared buffer. The second thread reads data from the shared buffer and draws it on the screen. Then, to avoid the decoder to be paused, the second thread is called each 15 ms to draw 4 new pixels. Finally, the buffer needs to store 6 sec. of ECG: 2 sec. for reading, 2 sec. for writing and 2 additional sec. due to the delay on the iPhone drawing hardware.

2) Low-Level Optimization of the CS Reconstruction Algorithm: The techniques used to optimize the CS reconstruction algorithm are based on exploiting code vectorization and SIMD extensions, typical of a large set of embedded microprocessors nowadays. In the case of the ARM Cortex-A8 of the iPhone 3GS, the SIMD support is implemented using intrinsics in a general-purpose SIMD engine called NEON [20]. These intrinsics provide similar functionality to in-line assembly and appear as function calls in C that are replaced at compile-time by a sequence of low-level instructions specific to the Cortex-A8. Vectorizing the CS reconstruction algorithm involves operating on vectors of data lengths between 2 and L elements, according to the underlying SIMD hardware support, where L is the maximum allowed float elements per vector. Regarding our iPhone implementation, we vectorized the reconstruction algorithm to operate on vectors of 4 float data, as this is the largest value of L supported on the iPhone 3GS. These vectors are processed inside loops, which are defined by the matrices of the CS reconstruction process. Thus, the loops size of the CS reconstruction algorithm is the only hard limitation on the vectorization process. Hence, we perform two different optimizations related to array padding, loop unrolling and loop peeling [22] in the hierarchy of loops to achieve real-time operation, namely:

a) Single-loop optimization: When the number of iteration in a loop is a multiple of L, it can be then vectorized into vectors of L elements. Each iteration is then independent from all previous ones and all vectorization pointers can operate on L successive addresses. However, if the iteration is not processing L successive addresses, then the vectors need to be loaded lane by lane. The following example shows an implementation of a simple multiply-accumulate in the code of the CS reconstruction algorithm. In particular, in the iPhone the Vector Floating Point (VFP) implementation takes $18-21$ cycles for a single-precision multiply-accumulate, but by exploiting NEON intrinsics, we can execute two multiply-accumulate in 1 cycle, achieving large performance saving.

In addition, the loops (or the number of the iterations of the loop) of the CS reconstruction algorithm use arrays not multiple of L elements for the filtering functions (cf. Section II-B). Therefore, the remaining elements of the loop after the vectorization are processed separately, which implies three different cases, as shown in Figure 3.

```
Original code
  for $i = 1$ to $n$ do
    $d[i] \leftarrow a[i] + b[i] * c[i]$
  end for
Vectorized code
  for $i = 1$ to $n/4$ do
    $d[4*i] \leftarrow a[4*i] + b[4*i] * c[4*i]$
    $d[4*i+1] \leftarrow a[4*i+1] + b[4*i+1] * c[4*i+1]$
    $d[4*i+2] \leftarrow a[4*i+2] + b[4*i+2] * c[4*i+2]$
    $d[4*i+3] \leftarrow a[4*i+3] + b[4*i+3] * c[4*i+3]$
  end for
Vectorized code with NEON Intrinsics
  for $i = 1$ to $n/4$ do
    initializing $pointers$ and $registers$
    $r4 \leftarrow vmlaq\_f32(x4, y4, z4)$
    storing results and incrementing pointers
  end for
```

This figure illustrates the vectorization of a loop operating on an array of $L * Iter + A$ elements with $A < L$, assuming $Iter = 2$, $L = 4$ and $A = 3$. Thus, the first $Iter$ iterations are vectorized without left element, but during $Iter + 1$ iteration, the vector only has A elements and one of the following three vectorization methods needs to be applied:

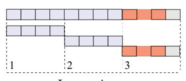
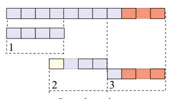
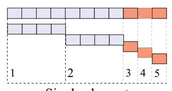
Combining array padding [22] techniques with loop unrolling by Allocating larger array. More memory will be then consumed. The new padding elements should be well initiated in order to not affect the result of the calculation.	 Larger Array
Combining loop peeling [22] techniques with loop unrolling by dividing the loop into two overlapping loops. (*L - A*) elements of the array will be computed twice. Overlapping can be used only when Iter > *0* and the operation applied to the input data does not vary with the number of times the operation is applied.	 Overlapping
The third method also combines loop peeling [22] techniques with loop unrolling, but the two loops are processed without overlapping. The remaining *A* elements will be computed one by one. First loop for vectors and a second loop for single remaining elements. This method will double the size of the code.	 Single elements

Fig. 3. *Handling leftover elements (fastest approach first)*

Finally, we transform the loops of the CS reconstruction containing any if statement to be correctly vectorized. Consider the optimization of the following code from the CS matrix projection:

```
for $i = 1$ to n do
  $y[i] \leftarrow fabs(u[i]) - T$
  $y[i] \leftarrow y[i] * (y[i] > 0.0f)$
  if $u[i] > 0$ then
    $y[i] \leftarrow y[i]$
  else if $u[i] < 0$ then
    $y[i] \leftarrow -y[i]$
  else
    $y[i] \leftarrow 0$
  end if
end for
```

Indeed vectorizing part of this loop and keeping the if statement written in its original code implies large performance penalties, as the processor keeps loading and storing elements from the NEON pipeline to the ARM pipeline. Hence, this loop can be parallelized (i.e., where $Y[i]$ is multiplied by the sign of $U[i]$) by considering two vectors of L elements, as shown in Figure 4 with $L = 4$. This implementation was inspired from a specific if-conversion technique [23], which uses the results of the comparison operator as values.

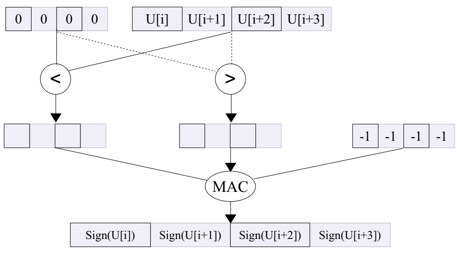


Fig. 4. *Using vectors to get the sign of 4 successive elements*

b) Multi-level loop optimization: In the filtering functions of the reconstruction algorithm, two levels of loop nests occur frequently and we can vectorize the inner or outer loop. For example, in the multi-band pass filter code excerpt shown next, the inner loop computes 2 elements: X, output of the low pass filter, is calculated from T_input and $H0$; Y, output of the high pass filter, is computed from T_input and $H1$. Then, $H0$ and $H1$ are constant arrays that contain the filters coefficients:

```
for $i = 0$ to I do
  for $j = 0$ to m do
    $X+ \leftarrow T\_input[i+j] * H0[j]$
    $Y+ \leftarrow T\_input[i+j] * H1[i]$
  end for
  $out\_l[i] \leftarrow X$
  $out\_h[i] \leftarrow Y$
end for
```

In this case, if I is a multiple of L, vectorizing the outer loop implies having two vectors X and Y with L elements each. Thus, the total number of executed multiply accumulate (MAC) instruction is $2 * (I/L) * m$. Since $L < m$ and m is a multiple of L in the loops of the reconstruction algorithm, vectorizing the inner loop is also possible, but it would require the execution of $2 * I * (L - 1)$ additional add instructions. Figure 5 shows the difference between vectorizing either the inner loop or the outer loop ($I = 4$, $m = 8$ and $L = 4$). As a result, in the multi-level loops of the reconstruction

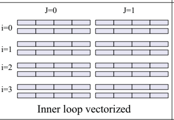
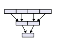
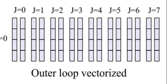
Vectorization	Additional Instr.
 Inner loop vectorized	For each iteration of the outer loop, the computed elements of vector of the inner loop need to be accumulated 
 Outer loop vectorized	The output vector contains already the final results

Fig. 5. *Difference between inner loop and outer loop vectorization*

algorithm where I is a multiple of L, we vectorize the outer loop. However, in the implementation of l_1 algorithm, I is not constant and smaller than L in several loops. Nonetheless, we can vectorize the outer loop by considering only one vector containing both elements X and Y together. Hence, the total number of MAC instruction is reduced to $I * m$.

V. Real-Time Performance Results

We have implemented the proposed optimizations (cf. Section IV) for the 16-bit CS encoder and 32-bit reconstruction

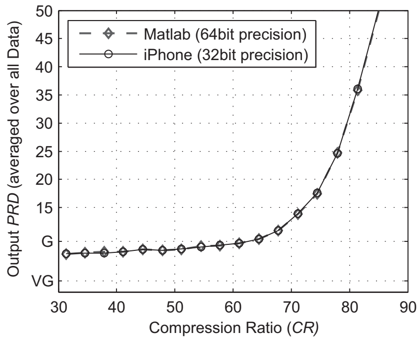


Fig. 6. *Performance comparison of ECG reconstruction*

algorithm. Figure 6 shows that the real-time implementation of the CS-based ECG system for WBSNs provides the same accuracy as the original 64-bit Matlab design. Moreover, after applying the low-level optimizations of the CS reconstruction algorithm (cf. Section IV-B), the algorithm runs 2.43 times faster for a compression ratio of 50%, which guarantees an accurate ECG reconstruction. Without these low-level optimizations, the maximum number of iterations of the CS reconstruction process to respect the real-time operation of the decoding part (i.e., 1 sec. of total time spent in ECG reconstruction every 2 sec.) reaches 800 iterations, while the optimized code reaches up to 2000 iterations.

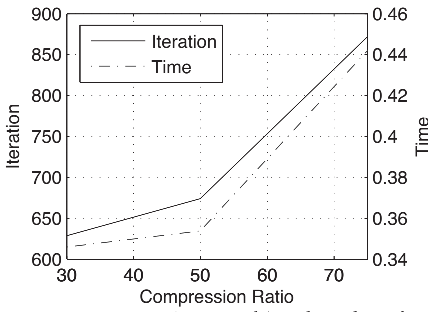


Fig. 7. *Average execution time (in seconds) and number of iterations on the iPhone to reconstruct 2-second ECG packet*

Figure 7 shows the average number of iterations and the average execution time per packet while testing the MIT-BIH Arrhythmia database [17] on the iPhone for different ratios. As Figure 8 illustrates, the optimized system accurately receives and reconstructs in real-time the ECG signal on the iPhone 3GS (as a WBSN coordinator), only taking a 17.7% of total CPU usage on average for a compression ratio of 50%. At the same time, the Shimmer™ node is able to sense, compress and transmit the ECG signal to the WBSN coordinator in real-time, while having an average CPU usage of less than 5%. These results translate into a 12.9% extension in the node lifetime, with respect to streaming uncompressed data, which suggests the energy efficiency of CS. Moreover, the processing energy can be further reduced using a more power-efficient microcontroller than the TI MSP430 of the Shimmer™.

VI. Conclusion

WBSN can offer cost-effective solutions to enable next-generation patient-centric tele-cardiology or e-cardiology solutions. In this regard, we have presented a novel real-time energy-aware ECG monitoring system based on the emerging compressed sensing (CS) signal acquisition/compression paradigm for WBSN applications. Our system demonstrated the feasibility of using CS in real-time for ECG compression, by implementing a light ECG encoder on the Shimmer™ wearable sensor node and a real-time decoder running on an iPhone 3GS, which acts as a WBSN coordinator.

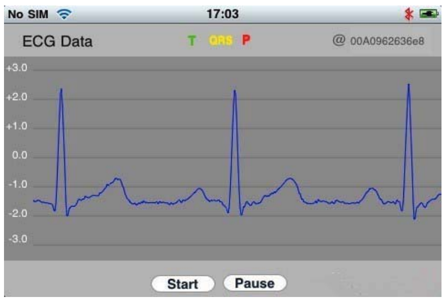


Fig. 8. *ECG on the iPhone*

References

[1] World Health Organization, "Cardiovascular diseases," 2009. [Online]. Available: http://www.who.int/topics/cardiovascular_diseases/

[2] Toumaz Technology, 2009. [Online]. Available: http://www.toumaz.com/public/news.php?id=92

[3] Shimmer Research. [Online]. Available: http://shimmer-research.com

[4] R. F. Yazicioglu and *et al.*, "Ultra-low-power wearable biopotential sensor nodes," in *Proc. of the IEEE EMBC'09*, Sep. 2009.

[5] L. Sörnmo and *et al.*, *Bioelectrical Signal Processing in Cardiac and Neurological Applications*. Elsevier Academic Press, 2005.

[6] E. Candès and *et al.*, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *IEEE Trans. on Inform. Theory*, vol. 52, no. 2, pp. 489–509, Feb. 2006.

[7] D. L. Donoho, "Compressed sensing," *IEEE Trans. on Inform. Theory*, vol. 52, no. 4, pp. 1289–1306, Apr. 2006.

[8] S. S. Chen and *et al.*, "Atomic decomposition by basis pursuit," *SIAM J. Sci. Computing*, vol. 20, no. 1, pp. 33–61, 1999.

[9] M. Figueiredo and *et al.*, "Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems," *IEEE Jrl. of Selec. topics in Signal Proc.*, vol. 1, no. 4, pp. 586–597, 2007.

[10] I. Daubechies and *et al.*, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Comm. Pure Appl. Math.*, vol. 57, pp. 1413–1457, 2004.

[11] J. Tropp, "Greed is good: Algorithmic results for sparse approximation," *IEEE Trans. on Inform. Theory*, vol. 50, no. 10, pp. 2231–2242, 2004.

[12] A. Beck and *et al.*, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM J. Img. Sci.*, vol. 2, no. 1, 2009.

[13] E. J. Candès and *et al.*, "Decoding by linear programming," *IEEE Trans. on Inform. Theory*, vol. 51, no. 12, pp. 4203–4215, Dec. 2005.

[14] E. Candes and *et al.* "Stable signal recovery from incomplete and inaccurate measurements," *Comm. on Pure and Applied Mathematics*, vol. 59, pp. 1207–1223, 2006.

[15] J. Bioucas-Dias and *et al.*, "A new twist: Two-step iterative shrinkage/thresholding algorithms for image restoration," *IEEE Trans. Img Proc.*, vol. 16, no. 12, pp. 2992 –3004, 2007.

[16] G. B. Moody and *et al.*, "The impact of the MIT-BIH arrhythmia database," *IEEE Eng. in Med. and Bio.*, vol. 20, no. 3, pp. 45–50, 2001.

[17] "MIT-BIH arrhythmia database." [Online]. Available: http://www.physionet.org/physiobank/database/mitdb/

[18] "GCC toolchain for the TI MSP430 MCUs." [Online]. Available: http://mspgcc.sourceforge.net/

[19] R. Berinde and *et al.*, "Combining geometry and combinatorics: A unified approach to sparse signal recovery," in *Proc. of Allerton*, 2008.

[20] "Cortex-a8 series." [Online]. Available: http://infocenter.arm.com

[21] "Btstack." [Online]. Available: http://code.google.com/p/btstack/

[22] D. F. Bacon and *et al.* "Compiler transformations for high-performance computing," *ACMComput. Surv.*, vol. 26, pp. 345–420, 1994.

[23] M. Suzuki and *et al.* "SIMD optimization in COINS compiler infrastructure," in *Proc. of IFGHPCS*, pp. 10pp, 2005.