A Method for Energy-Efficient Sampling of Analog to Digital Converters

Neil Klingensmith, *Member, IEEE,* Suman Banerjee, *Member, IEEE,*

Abstract—We present PANDA, a data acquisition technique for energy-constrained sensor nodes that reduces the energy per operation required to sample and preprocess analog sensor data. PANDA takes advantage of the energy consumption patterns of commodity microcontrollers by sampling input signals in short bursts followed by long periods of inactivity. This approach reduces the overhead of repetitively transitioning the CPU and analog components in and out of low-power sleep states. This nonuniformly-spaced input data is then fed to a nonuniform FFT algorithm that computes the frequency spectrum. We show that the spectrum computed with the nonuniform FFT is very close to the spectrum that would be computed from uniformly sampled data preprocessed with a conventional FFT. The output of the nonuniform FFT can be filtered or postprocessed with conventional frequency domain analysis techniques, and a uniformly resampled output can be constructed with the conventional inverse FFT. We compare the energy consumption patterns of burst-mode sampling to those of conventional uniform sampling in several real sensor nodes. We demonstrate that for reasonably sized input datasets, burst mode sampling and postprocessing consumes more than 17% less energy than conventional uniform sampling, including the additional computations required to compute the nonuniform DFT.

1 Introduction

Energy consumption in mobile and IoT devices is a primary concern because power sources—batteries, energy harvesting devices, etc—are limited in capacity. In ultra-low power sensor systems such as SNUPI [1], MICA2 [2], Amulet [3], and others [4], [5], [6], [7], power consumed by analog-to-digital conversion circuitry is a primary contributor to battery drain. As microcontrollers become increasingly complex, adding features such as direct memory access, memory protection, etc., optimizing sleep state power consumption will become more important because wake state power draw will be higher. With PANDA, we aim to reduce the overall power consumption of battery-powered sensing devices by increasing the fraction of time the system spends in a low-power sleep state. PANDA targets applications that use mid-range microcontrollers which are sensitive to power consumption and also have sufficient resources to perform nontrivial computations. This mid-range segment, represented by 32-bit devices such as the ARM Cortex M line, is being invested in heavily by many microcontroller manufacturers, and it represents a large and growing portion of microcontroller sales.

To extend the battery life of sensor nodes, we seek to reduce the energy per operation of sampling and preprocessing our input sensor data. To do so, we put the CPU and analog components into sleep mode for as much time as our application will allow.

However, sleeping the analog components between sample acquisitions poses a problem for real time systems because the process of putting each component to sleep and waking it up consumes a significant amount of time and energy. We have found that for many microcontrollers, waking the ADC from sleep mode takes roughly the same amount of time and energy as acquiring a single sample. When sampling medium and high-frequency signals, the amount of time and energy consumed in powering down the analog subsystem between samples contributes significantly to the overall energy consumption of the analog to digital conversion process [8].

But sleep-wake overhead is not the only inefficiency in ADC sampling. In applications that need to sample multiple analog input channels at high frequency, the overhead of switching between channels is often a dominant contributor to the overall sampling period. This can significantly reduce the maximum sampling frequency of the ADC. Multichannel ADCs, which are common in mid-range microcontrollers, often use an analog multiplexer to direct signals from the input channels to the conversion circuitry (see Figure 1) rather than having an independent conversion circuitry for each channel. On many microcontrollers, the analog input mux is slow to switch input channels. Applications that need to sample many analog input channels could use PANDA to reduce the overhead of input channel switching by sampling each input channel in bursts before switching to the next input channel. This would reduce the overhead of switching between analog input channels, and it can significantly increase the maximum sampling frequency we can achieve. Table 1 gives an overview of ADC performance characteristics for a few popular microcontrollers. The rightmost column of the table shows the latency of switching ADC input channels between samples relative to sampling the same channel sequentially.

With PANDA, we seek to reduce energy and time wasted by repeatedly sleeping and waking a sensor node between samples. Figure 3 shows a comparison of (a) conventional sampling performed at uniform time intervals and (b) and

- *Neil Klingensmith is with the Department of Electrical and Computer Engineering, University of Wisconsin, Madison, WI, 53706. E-mail: see http://neilklingensmith.com*

Manuscript received January 9, 2019; revised April 8, 2019.

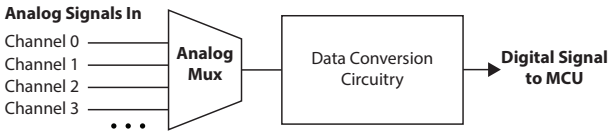

Fig. 1. Schematic of a typical multi-channel ADC circuit. A single data conversion circuit performs digitization of all analog input data. An analog multiplexor selects a single analog input for conversion. Changing the selected analog input is a long-latency operation that reduces the system performance.

(c) the nonuniform paradigm used by PANDA. Numbered circles represent samples collected by the microcontroller. Rather than waking the CPU and ADC for a single conversion at uniform time intervals, we propose collecting samples in bursts. After waking the microcontroller from sleep, several samples will be collected from the ADC in rapid succession before returning to the sleep state. Sampling in this way reduces the number of times we need to transition the CPU in and out of its low-power sleep state.

In Figure 3 (a), using uniform sampling, we transition the CPU in and out of sleep a total of six times, using overhead energy each time. In Figure 3 (b), we collect samples in bursts of two. Here, we collect the same number of samples in the same amount of time, but we only have to transition the CPU in and out of sleep three times. This reduces the amount of energy spent on overhead by half. In Figure 3 (c), we further reduce the energy overhead by collecting samples in bursts of three, transitioning in and out of sleep only twice and saving more energy still.

Unlike other burst-mode data acquisition techniques, PANDA provides meaningful frequency domain analysis tools including frequency spectrum analysis, linear filtering, and signal reconstruction. The availability of frequency domain signal processing techniques makes PANDA a practical tool for real-time signal processing applications like energy metering, biometric (ECG, EEG) monitoring, and so forth.

Intuitively, we think of burst-mode sampling as capturing information about a signal's value and its first few derivatives. In a short burst of four samples, for example, we can record the signal's value and its first three derivatives. Knowledge of the signal's first few derivatives at a moment in time gives us information about how the signal will evolve in the near future. This makes it possible for us to skip samples after the burst, which saves energy.

Unfortunately, direct Fourier analysis techniques—FFT, DFT, etc.— are not useful for interpolation of nonuniformly sampled signals because the derivatives of a signal are linearly dependent. Fourier techniques are successful at interpolating signals because the basis of sinusoids they use are orthogonal. To solve this problem, we use other methods of interpolating and resampling the signal before applying Fourier methods.

Under the burst sampling paradigm, we can amortize the overhead of bringing the microcontroller out of its sleep state over more than one ADC sample, allowing us to save time and energy. However, the data produced by PANDA is not sampled at uniform intervals, and therefore cannot be fed directly into conventional signal processing algorithms such as the FFT.

To deal with this problem, we piggyback off of work that has been done in the signal processing community in the area of nonuniform sampling and reconstruction. We adapt the existing work, which was initially developed for use in RADAR systems, to work on microcontrollers. We then evaluate the timing implications both for ADC sampling and data processing, showing that we can achieve **more than 17% reduction** in sampling energy compared to uniform sample spacing, even when accounting for the additional processing overhead required to compute the frequency spectrum. To our knowledge, PANDA is the first algorithm to adapt nonuniform DFT techniques for online use in a resource-constrained runtime environment.
The key contributions of this work are as follows:

- We adapt an existing nonuniform DFT algorithm to work on online in a resource-constrained runtime environment.
- We compare the energy used by PANDA to conventional uniformly spaced sampling, demonstrating that burst-mode sampling can **reduce the energy per sampling operation by more than 17%**, including additional computation overhead.
- We describe the limitations of frequency domain analysis on burst mode sampled data that we encountered in resource constrained real-time environments.

2 Background

The energy required to perform analog to digital conversions in commodity microcontrollers varies by architecture, but it can account for a significant portion of energy budget. For low-power applications, the MCU will enter a sleep state, powering down unnecessary components whenever possible, in order to conserve energy. Most microcontrollers support several different levels of sleep, where "deeper" sleep modes correspond to powering down more functions or peripherals of the microcontroller. When entering deep sleep states, the sensor node must typically reconfigure clock generation and distribution circuitry such as the microcontroller's phase-locked loop (PLL) and clock tree, a process which consumes time and energy. So each time the sensor node enters or returns from a sleep state, it must spend time and energy to do so.

For nodes like temperature sensors that take data samples at extremely low frequency, the energy consumed by sleeping and waking the CPU and peripherals is insignificant. In higher frequency applications, on the other hand, which require sampling real-time signals at rates of more than few hundred Hertz, the overhead of repetitively sleeping and waking the microcontroller represents a large fraction of the node's overall energy consumption. For extremely low-power applications, this overhead can be comparable to the amount of energy used by the communications interface [8].

Figure 2(a) shows the power consumed by a 32-bit ColdFire microcontroller's ADC and CPU during a single ADC sample. After waking from its sleep state, the MCU spends a relatively short amount of time—roughly 14 μs—sampling its ADC. Once the ADC has been sampled, the CPU and ADC return to a sleep state to conserve power, a process that takes more than 100 μs. For a single sample,

MCU	Number of Channels	Same Channel	Alternate Channels	Slowdown
Atmel AVR XMEGA	16	0.25μs	3.5μs	14x
Freescale Kenetis K50-72MHz	2	1.25μs	10μs	8x
Freescale ColdFire	24	5μs	15.95μs	3.19x
TI ARM Cortex M4	24	1μs	3μs	3x

TABLE 1

A comparison of ADC sampling time among several popular commodity microcontrollers. Same channel is the amount of time it takes to sample the same ADC channel sequentially. Alternate channel is the amount of time it takes to sample different channels sequentially. Slowdown is the relative performance penalty of alternating channels compared to sampling the same channel sequentially.

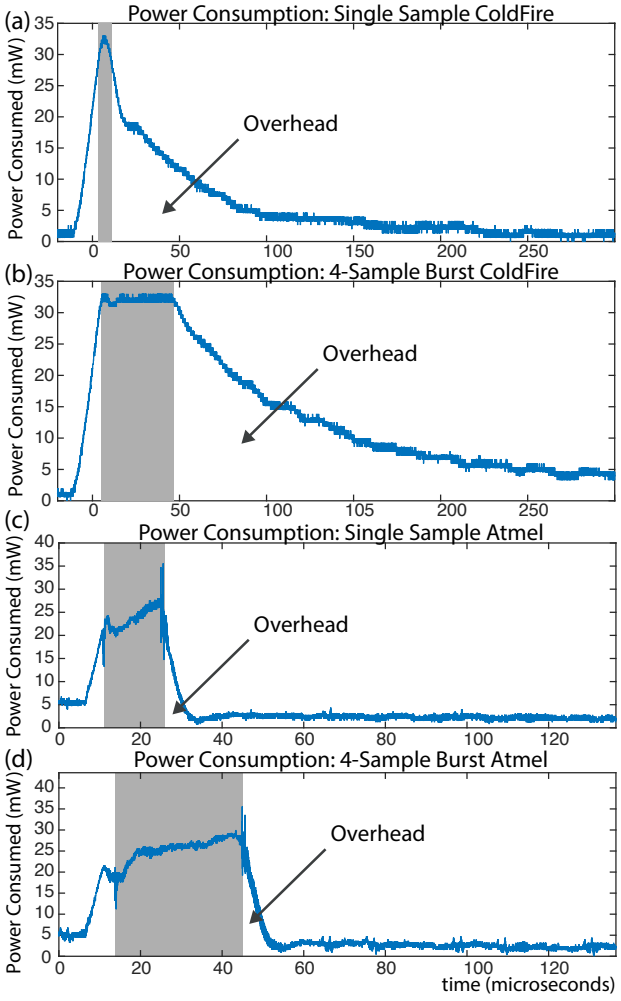

Fig. 2. Power consumed by the CPU and ADC to acquire (a) a single ADC sample and (b) a burst of four samples in succession. The area highlighted in gray shows the time during which the microcontroller is doing useful work. For the rest of the timei represent overhead during which time the MCU is consuming power but not performing any useful work. By amortizing this overhead over bursts of multiple ADC samples, PANDA reduces the overall energy consumption of ADC sampling.

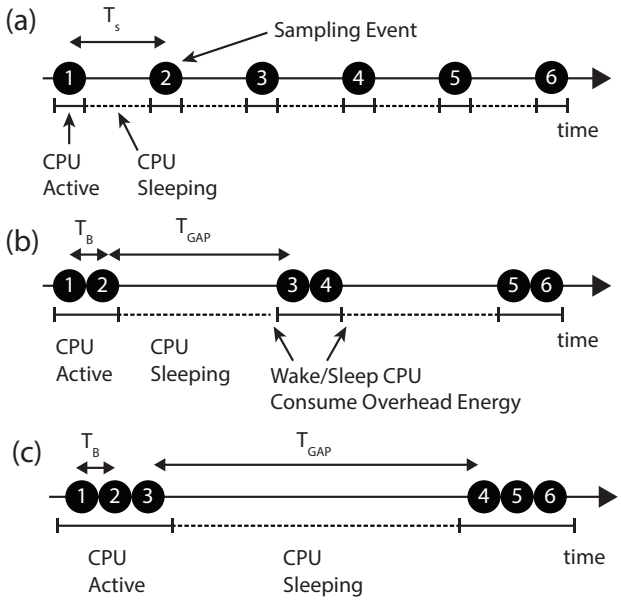

Fig. 3. A comparison of (a) conventional sampling performed at uniform time intervals to (b) and (c) nonuniform burst sampling used by PANDA. In this example, we amortize the overhead of sleeping and waking the microcontroller over three samples by collecting data in bursts.

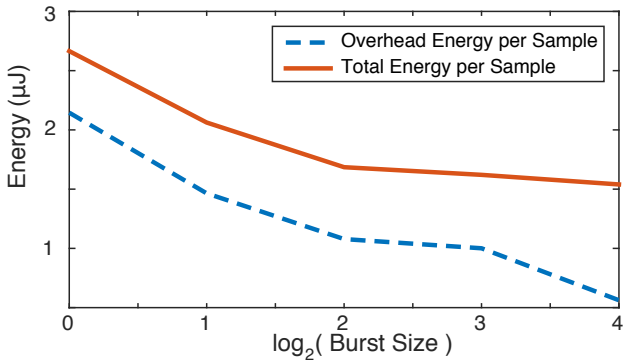

Fig. 4. Total energy and energy overhead per sample for different lengths of sample burst. As bursts get longer, we can amortize the overhead of waking and sleeping the microcontroller over more samples. In this application, we see diminishing returns above 4 samples per burst.

most of the energy is consumed during the power down phase, while components in the MCU return to their sleep state.

In contrast, Figure 2(b) shows the power consumed by the ColdFire during a burst of four sample acquisitions. In burst mode, the overhead of powering down the CPU and ADC can be amortized over several sample acquisitions. This is the key observation that allows PANDA to conserve power.

Figures 2 (c) and (d) show the power consumed by an 8-bit Atmel microcontroller in 1- and 4-sample bursts. Although it is clearly far more efficient in terms of its sleep overhead power consumption, the Atmel microcontroller exhibits similar tendencies.

By sampling our signal of interest in in high-frequency bursts followed by extended periods of deep sleep, we can effectively gather the same information about our target signal while conserving energy by reducing the overhead required to repetitively wake and sleep the MCU. Figure 4 shows how the overhead energy per sample decreases for increasing sample burst lengths. This approach has been used in many systems [9], but until now, only rudimentary

signal processing techniques could be applied[1].

Once the data is collected, we want to be able to analyze it using frequency-domain techniques. For example, we may wish to apply a linear filter to the data and then resample it uniformly in time to be analyzed further. To do so, we apply a matrix transform operator, much the same as the matrix formulation of the conventional Discrete Fourier Transform.

2.1 Predicting the Future

It may seem that we are suggesting a method that, in an extreme case, would allow us to predict signal values far into the future by clustering samples in the present. Unfortunately this is not possible.

One important property of Fourier analysis is that signals that are discrete in frequency are periodic in time with the period $T = N$, the sampling window length [10]. When we sample a continuous time domain signal, we get a periodic discrete time domain signal. Our representation of that signal in memory—as an ordered set of numbers in a buffer—is just one period of that signal. Since the mathematical representation of the sampled signal is periodic in time, it is not possible to interpolate values of the signal outside of the sample window.

2.2 Related Work

There has been a lot of study on reconstruction of signals from nonuniform samples. Analysis of nonuniformly sampled signals was first investigated by Yen [11] in 1956. Early work in the area was concerned with analyzing RADAR return signals. Since that time, techniques have been developed to analyze nonuniformly sampled signals in astronomy [12], medicine [13], image processing [14], [15], [16], and others. Laguna and Moody studied the power spectral density of unevenly sampled data using least square analysis and modeled the uneven sampling as uniform sampling plus a stationary random deviation. They applied this technique to analyze the Heart Rate Signals [13]. Theoretical contributions have been made to improve the quality of frequency spectrum estimates and reconstruction techniques [17]. Jenq talks about perfect reconstruction of digital spectrum from the nonuniformly sampled signals where the timing offsets of each sampling instance are known and have a periodic structure [18].

Butzer and Hinsen also offer some interesting insight reconstruction of bounded signals from pseudo-periodic, irregularly spaced samples [19]. Moinch and Boche's work on the non-equidistant sampling for bounded signals considers sampling patterns that are made of zeros sine-type functions and analyze the local and global convergence behavior of the sampling series. They also discuss the influence of oversampling on the global approximation behavior and the convergence speed of the sampling series [20]. However, these techniques have generally been applied only when no other course of action is practical. That is, people do not generally sample signals nonuniformly when other options are available because less is known about the mathematical properties of spectral analysis and reconstruction of nonuniformly sampled signals. Nontraditional data acquisition techniques such as compressive sensing [21] have also been applied in sensor networks to reduce power consumption. [22] used this technique for scheduling the measurement for the estimation of soil moisture. The Sparse Fast Fourier Transform [23], [24] is another technique for reducing the computational overhead of computing the most prominent harmonics of a signal.

Energy profiling of embedded systems has been widely studied topic. Zhai studied nonuniform sampling of sensors on smartphones [25], using similar techniques to decrease energy consumption by clustering sensor sampling. Tiwari et al. [26] talks about power modeling of embedded processor to characterize power budget of different software running on a processor. Cycle accurate energy estimation technique has been presented in [27]. Characterizing energy split up in the overhead is out of the scope of this work and can be done as a future work using the techniques mentioned above. Our work focuses on finding the overall energy overhead of each sample and thus we went ahead measured energy used by the entire board. Since this is a straight forward method, we have not come up with any estimation model.

1. For example, burst mode sampling is commonly used to capture the value of a DC signal in additive white Gaussian noise by computing the mean value of many successive samples before returning the node to sleep

Algorithm	Computation Complexity	Stability
Marvasti [14]	$O(N^2)$	Low
Dutt [28]	$O(NlogN + Nlog(1/\epsilon))$	High
Andersen [29]	$O(N^2)$	Moderate
Greengard [30]	$O(NlogN + Nlog(1/\epsilon))$	High
Press [31], [32]	$O(N\log(N))$	High

TABLE 2
A comparison of nonuniform DFT algorithms. Here, N is the size of the input dataset.

3 Algorithm Overview

Broadly speaking, there are two classes of algorithms for computing the nonuniform DFT:

1) **Direct computation methods** use a matrix-vector multiplication, similar to the matrix form of the standard DFT:

$$f = As$$

where s is the signal of interest (a vector) sampled at nonuniform time intervals, A is a matrix whose rows are complex sinusoids sampled at the same times as s, and f is the computed frequency spectrum, also a vector. The advantage of this method is that it is a convenient and intuitive representation of the frequency spectrum. Its main disadvantage is that it tends to amplify noise in the input signal. Since the rows (and columns) of A are not orthogonal, frequency components from different harmonics of the input signal s can be confused. This effect becomes more pronounced as the sampling nonuniformity increases. For that reason, it is not suitable for burst mode sampling, which relies on highly nonuniform sampling patterns to conserve energy in the ADC.

2) **Interpolation and resampling methods** use some form of interpolation—for example, polynomial or

spline interpolation—of the input data to construct a continuous-time model of the signal of interest. This continuous time model is then resampled at uniform intervals, and the uniformly sampled signal is fed in to a standard FFT. The interpolation and resampling approach has the advantage of being more stable for highly nonuniform sample times. A disadvantage of this approach is that it is less computationally efficient because it must solve two matrix-vector problems, compared to one in the direct method.

The fastest and most stable NUFFT algorithm we are aware of in the literature is one published by Greengard and Lee [30] for processing RADAR signals. Their algorithm, designed to run on PCs, uses a technique they call *fast Gaussian gridding* to interpolate between nonuniformly spaced samples. The idea is to replace each sample—conventionally represented as a Dirac delta function—with a Gaussian bump. The continuous time signal is then represented as an expansion Gaussians:

$$f(t) = \sum_{n=0}^{N-1} a_n e^{-(t-t_n)^2/\sigma_n^2} \quad (1)$$

In the Gaussian gridding approach, Greengard and Lee find the a_n and σ_n that best match the samples and then interpolate uniformly spaced samples in the Gaussian basis. Normally, computation of those parameters would require solving an $N \times N$ system of equation, using $O(N^3)$ operations. Fast Gaussian gridding reduces that time by making the simplifying assumption that each Gaussian bump only affects signal values in its immediate neighborhood. This optimization produces a banded matrix which is approximately zero everywhere except on the diagonal a few subdiagonals. Fast Gaussian gridding achieves higher performance by only considering a small subset of the basis functions for each point in the interpolation, requiring $O(w^2N)$, where w is the number of subdiagonals.

We have modified the Greengard and Lee algorithm for use in real-time applications in resource-constrained environments:

- We have separated the initialization phase of the algorithm into its own function, called once at startup.
- We made the data representation selectable among fixed point, single-precision floating point, and double-precision floating point.

These optimizations reduce the runtime of fast Gaussian gridding by roughly 40%. At runtime, we only need to solve the banded matrix-vector system of equations to interpolate between sample bursts followed by a conventional FFT. A demonstration of the technique with plots can be found in [33].

4 Evaluation

4.1 Experiment Setup

In this section, we study how burst-mode sampling affects energy consumption in sensor nodes. Compared to conventional uniform sampling, burst-mode sampling in PANDA changes the way we collect and process sensor data. We consider energy used by data acquisition and preprocessing, roughly dividing the tasks into:

1) **Sampling energy** required to operate the sensors and analog to digital conversion hardware.
2) **Computation energy** required to compute the frequency spectrum of the sampled data.

For both operations, we compare the amount of energy used in conventional sampling to that used by PANDA. We then discuss how PANDA can be applied in several example applications. For each application, we will compare the energy spent sampling the waveform with PANDA to the energy spent with conventional sampling. We will also compare the amount of CPU time it takes to preprocess the data with PANDA to the amount of time spent with the FFT.

The applications we study in this work are:

Electric Power Meter

In this application, our goal is to measure the electric power consumed by some household appliance in real time. We use a current transformer to measure AC electric current and a resistor to measure AC voltage. An 8-bit Atmel ATmega328 microcontroller samples the analog output of the sensors with its analog to digital converter and computes the power by multiplying the 60 Hz components of the voltage and current waveforms together. A detailed description of this application is available in [5].

Water Flow Sensor

The water flow sensor tracks the volume of water passing through a piece of industrial water treatment equipment. The water sensor hardware outputs a square voltage waveform with a frequency proportional to the water flow rate. A 32-bit ARM microcontroller records the frequency generated by the flow sensor and converts it to flow rate.

To compress the data, our microcontroller transforms the computed flow rate into the frequency domain and discards low-amplitude harmonics.

Figure 5 shows the average amount of energy used per sample in different burst lengths, with bursts of length 1 corresponding to uniform sampling. Clearly, burst mode sampling reduces the energy required to sample the signal of interest in both applications. We will study the energy requirements of computing the frequency spectrum from burst mode samples in later sections.

Figure 6 shows a breakdown of the energy used in sampling for the power meter application using various sample burst lengths. We have divided the energy consumed into (1) *sampling energy* and (2) *overhead energy*. Sampling energy is the actual energy required to run the ADC, CPU, and analog circuitry to acquire the samples. This is the energy blocked out in gray in Figure 2. Overhead energy is the energy required to power up and power down the microcontroller before and after gathering samples.

This data was gathered by setting one of the microcontroller's GPIO lines to a 1 during the actual sampling process and setting it to a 0 just before sleep. We then

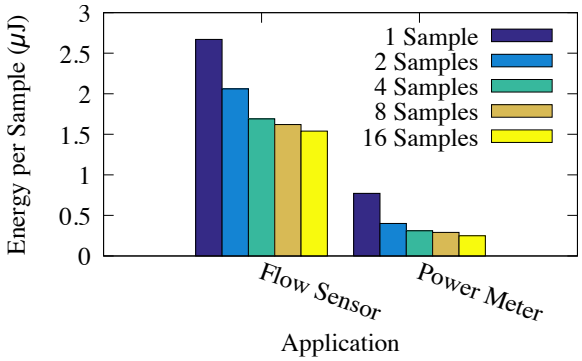

Fig. 5. Comparison of energy per sample for two different application using several different burst lengths.

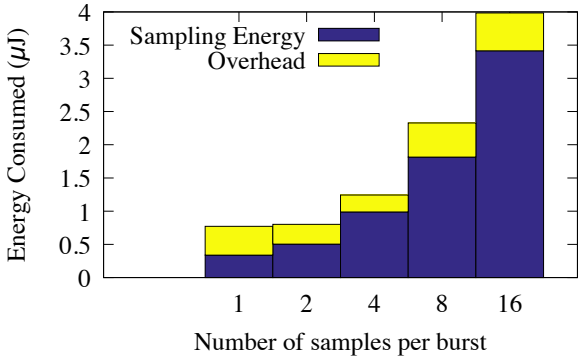

Fig. 6. Energy consumed by sampling in the power meter application (AVR microcontroller) using different burst modes.

gathered power consumption traces with an oscilloscope and integrated the traces numerically.

The overhead energy remains roughly constant for all burst lengths we studied because the energy required to sleep and wake the microcontroller is a constant, independent of the amount of work performed. The sampling energy grows roughly linearly with the number of sample acquired.

As we increase the number of samples gathered per burst in Figure 6, we can see that the overhead energy becomes a smaller fraction of the total. Using this fact, we can amortize the energy required to sleep and wake the CPU over more samples.

4.2 A Simple Demonstration

Part (a) shows $f_{AA}(t)$, the analog waveform to sample after it has been passed through an analog low-pass antialias filter. This is the signal that would actually be sampled by our ADC, with high frequency components removed. The spectrum of $f_{AA}(t)$ is shown in part (b).

Part (c) shows the samples of $f_{AA}(t)$ that would be taken by our ADC in burst mode. In this example, we use a burst length of two samples. Each burst of samples is followed by a period of inactivity equal in length to the burst. We then compute the nonuniform DFT of the burst-mode sampled signal, shown in part (d). This should be the same as (or very close to) the frequency spectrum we would have gotten had we sampled $f_{AA}(t)$ uniformly and computed the FFT with standard methods. Figure 7 (e) shows the error between the PANDA spectrum of $f_{AA}(t)$ and the spectrum computed with the FFT. Stable algorithms should keep this error as small as possible.

At this stage, the data gathered using PANDA is in the same format as if we had collected samples at uniform time intervals and transformed it using the conventional FFT. We could apply standard frequency domain filtering techniques to the data in Figure 7 (d). To reconstruct the time-domain samples at uniform intervals from the filtered data, we can apply the standard inverse FFT, shown in part (f).

We assume in this example that our objective is to apply some frequency domain analysis to the nonuniformly sampled signal. If, on the other hand, all we want is to resample a nonuniformly sampled signal at uniform time intervals, there are several direct techniques for doing so [19], [34], [35].

Requirements for Sampling Functions Nonuniformly

Marvasti et. al. provide an analysis of the requirements for nonuniformly sampled functions, which we will not repeat here [14]. They show that as long as the average sampling rate satisfies the Nyquist criterion, the continuous signal can be reconstructed without loss of information.

Time Complexity of PANDA

In this section, we compare the amount of time (and CPU cycles) required to compute the frequency spectrum of burst-mode sampled data gathered with PANDA. The purpose of this analysis is to be sure that the energy we save with burst-mode sampling is not wasted in extra computation time on the microcontroller. There are several efficient algorithms know for computing the frequency spectrum of nonuniformly sampled data (see Table 2, the fastest of which use $O(NlogN + Nlog(1/\epsilon))$ operations for N time-domain samples and precision of ϵ.

Figure 8 shows runtime measurements taken on an ARM CPU of a standard FFT algorithm (KISS FFT), an in-place DFT algorithm, and PANDA's nonuniform FFT algorithm.

4.3 Runtime Profiling

In this section, we compare the runtime PANDA with the standard FFT, analyzing real data collected from a large deployment of water flow and quality sensors. We analyze roughly 9000 data points collected at 5-minute intervals over a time span of several months. Using the raw samples as ground truth, we created burst-mode sampled data by intermittently saving and discarding data points from the original sample database. We then use the data collection platform (which includes an ARM CPU) to process the data with the PANDA NFFT algorithm and compare the results to uniformly spaced sampling mode.

Figure 8 compares the execution time of two library DFT algorithms with PANDA. These experiments were done on a ARM microcontroller, which is a commodity MCU used in our water flow sensor application.

1) **KISS FFT** [36] is a (relatively) simple implementation of the FFT designed to run on PCs with floating point support. KISS is an out-of-place implementation of the FFT, meaning that it requires two separate buffers (one input and one output) to compute the frequency spectrum.
2) **In-Place FFT** [37] is an in-place DFT implementation targeted at low-end embedded systems. Since it is

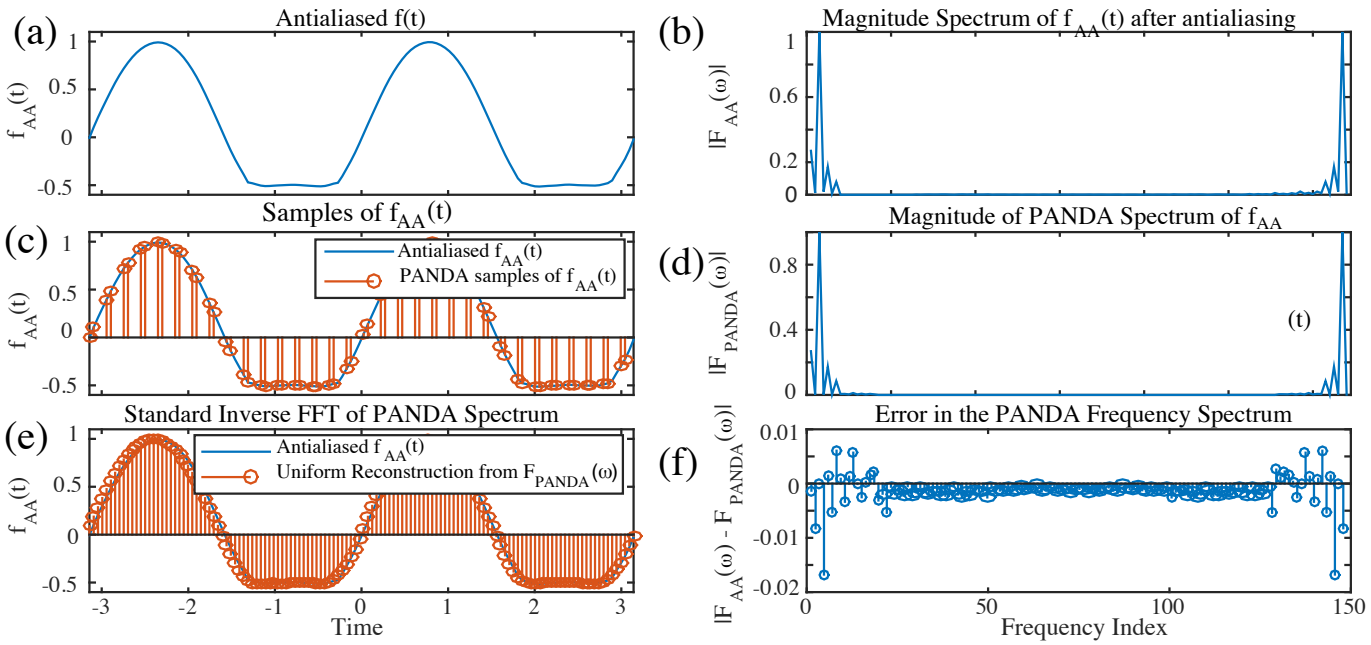

Fig. 7. A demonstration of sampling a time-domain signal at nonuniform time intervals, computing its frequency spectrum, and reconstructing the original waveform using a standard inverse FFT.

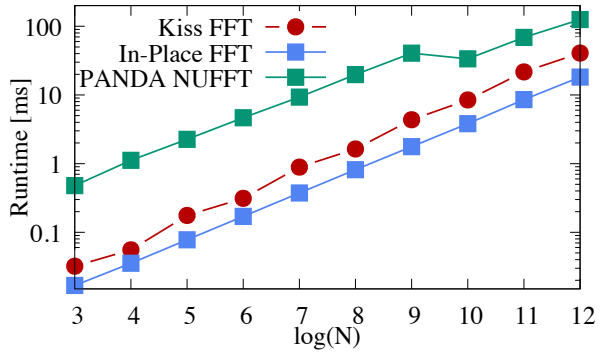

Fig. 8. Runtime comparison between in-place and out-of-place FFT on an ARM Microcontroller. The x-axis is the log of the dataset length, and they y-axis is the runtime of each algorithm. The jog in PANDA's runtime is caused by L1 data cache misses for smaller sample lengths.

an in-place implementation, it requires roughly half the data storage as KISS because the input and output buffers are the same.

3) **PANDA NUFFT**, to compute the frequency spectrum of a nonuniformly sampled input dataset.

Surprisingly, the PANDA NUFFT runs faster on this CPU for 2^{10} samples than for 2^9 because of the behavior of the microcontroller's data cache. We did not optimize our code for the microcontroller's cache configuration. The KISS FFT implementation takes advantages of optimizations which reduces the computational complexity to $O(n\log(n))$. The in-place DFT has a higher computational complexity than the FFT [38], because it cannot take advantage of FFT optimizations which require separate memory buffers to perform decimation in time. For the remainder of this paper, we will compare PANDA to the KISS FFT implementation because it can achieve a shorter runtime which will translate into less energy consumed by the CPU during the computation of the frequency spectrum.

4.4 An Alternative: Downsampling Without Bursts

Do we actually gain anything by sampling in bursts at lower frequencies? What would happen if, instead of sampling in bursts at lower frequencies, we just use uniform sampling at lower frequencies? We want to be sure that burst mode sampling actually gives improved resolution in the frequency domain.

Figure 9 shows how frequency resolution (on the y axis) depends on sampling intervals in our water quality dataset, with bigger numbers corresponding to worse resolution. The original dataset has samples taken at five minute intervals. To learn how frequency domain resolution depends on sampling frequency, the original data was downsampled, which was possible because the original sampling frequency was much higher than the Nyquist frequency of the underlying signal.

Downsampling was done in two modes: uniform in which the dataset was downsampled uniformly in time, and burst in which the dataset was downsampled in burst mode. Burst mode downsampling on the y axis is annotated as (number of points retained):(number of points discarded). For example, 2:4 burst downsampling means that two data points from the original dataset were retained, and the following four were discarded. Between groups, bars with the same colors correspond to samples taken at the same frequency in uniform sampling mode or burst sampling mode. For example, the blue bar corresponds to bursts of length two (keep two samples, throw out two samples) occurring at the same frequency as uniform downsampling by a factor of four (keep one sample, throw out three samples). Both blue bars represent sampling of the original signal at a period of 5*4 = 20 minutes. In burst mode downsampling, we take two samples every 20 minutes, and in uniform downsampling, we take one sample every 20 minutes. We want to confirm that by retaining more data in burst mode, we actually get better resolution in the frequency domain.

As we would expect, resolution in the frequency domain gets worse (coarser) as we downsample the data more.

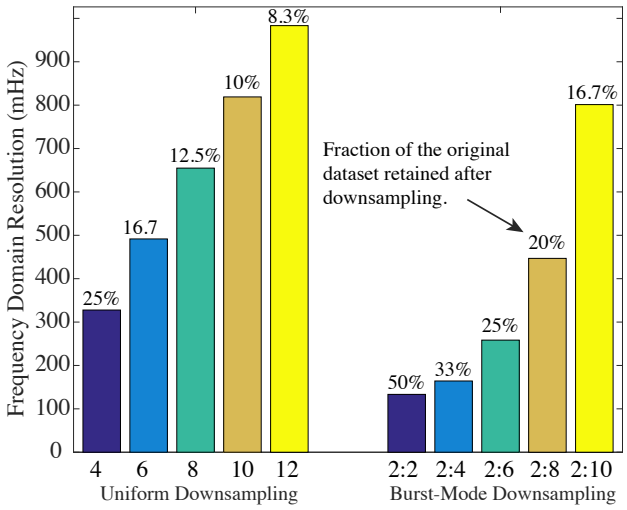

Fig. 9. Frequency domain resolution as a function of sampling frequency for uniform downsampling and burst mode downsampling. Small is good, big is bad.

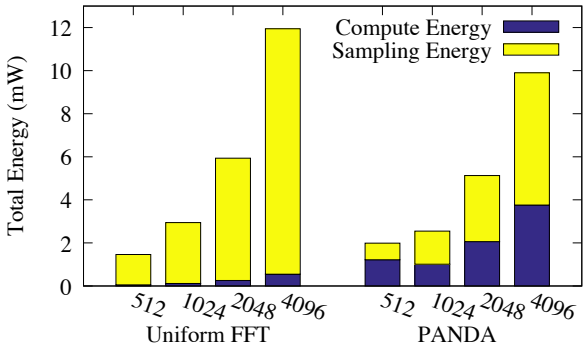

Fig. 10. Total energy used in sampling and frequency spectrum computation for the uniform KISS FFT and the PANDA NUFFT. The x-axis is the dataset size for each algorithm, and the y-axis is the total energy used to sample and compute the frequency spectrum. PANDA outperforms the uniform FFT in terms of energy consumption for data windows larger than 1024.

However, in burst mode, it is clear that resolution in the frequency domain degrades more slowly, despite the fact that the time interval between bursts is the same as the corresponding time interval between uniform samples in the other bar cluster. In terms of information content in the frequency domain, there is value in taking more than one sample in a bursts. If we take more than one sample in a burst, we can extract more information about the frequency domain content

Furthermore, for comparable amounts of retained data, Figure 9 shows that we get comparable resolution in the frequency domain. For example, 2:6 burst mode sampling retains 25% of the data from the original dataset (two samples kept, six discarded). Its resolution in the frequency domain is comparable to $4\times$ uniform downsampling, which also retains 25% of the original data.

4.5 Energy Profiling

4.5.1 Fixed Point Math

We profiled the energy consumption of PANDA on several ARM Cortex-M microcontrollers using both fixed-point and floating point math. Higher-end devices including the Cortex-M7 and Cortex-M4 have a floating point coprocessor

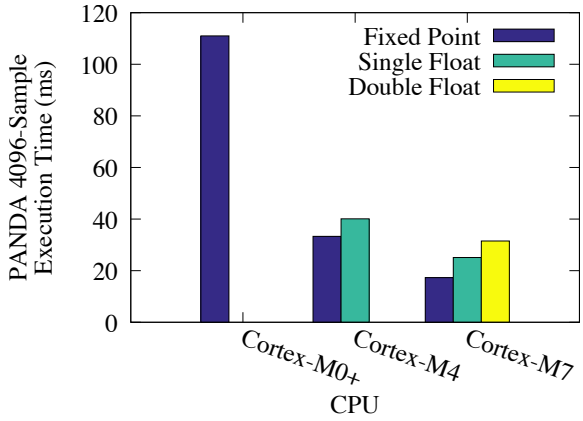

Fig. 11. Runtime comparison of PANDA frequency spectrum computation on three different ARM microcontroller cores.

that can do basic floating point operations in hardware. Unlike desktop and server CPUs, the floating point coprocessor on ARM microcontrollers does not include hardware support for higher-order functions like trigonometry, logarithm, etc. These functions must be performed in software.

Table 4 shows a comparison of the number of CPU cycles required for math operations used by fast Gaussian gridding in both floating point and fixed point. The `exp()` operation is the dominant slowdown in the floating point library implementation. We used the `libfixmath` [39] library for fixed point operations. These benchmarks were done on an ARM Cortex-M7 microcontroller.

4.5.2 CPU Performance

Figure 11 compares the runtime of PANDA on three different ARM CPU cores with 4096-sample buffers. The three CPU cores we chose to evaluate are the most popular mid-range microcontroller cores currently in use. The Cortex-M0+ core is a simple CPU designed for low-power and low-cost applications. The Cortex-M4 is a mid-range core, and the Cortex-M7 is a high-end superscalar CPU with a deep pipeline. Table 3 shows a comparison of features of these three cores. Most of our evaluation was done on the Cortex-M7, but Figure 11 shows how our techniques would port to lower-range devices.

The Cortex-M4 and M7 share the same instruction set, but the M0+ only implements a subset of of the other devices' ISA. Importantly, the M0+ does not support $32 \times 32 \rightarrow 64$ bit multiplication, which must be emulated in software. This feature disproportionately slows down execution on the M0+ core.

Figure 12 compares PANDA's compute energy on the same three CPU cores. Surprisingly the M0+ core requires the least amount of energy to compute the frequency spectrum in fixed-point math even though its compute time is nearly three times as long.

We observed a significant difference in the floating point performance between the Cortex-M4 and Cortex-M7 microcontrollers. This seems to be caused by more sophisticated pipeline in the M7, which has two execution lanes for load/store, integer ALU, and floating point. If the compiler can generate interleaved instructions, it is possible for 4 instructions (2 int + 2 float) to execute in parallel. The two-lane load/store buffer can also hide some latency in the CM7

CPU	Core Frequency	ISA	Pipeline Depth	Issue Width	Cache	Floating Point
Cortex-M0+	96 MHz	Thumb Subset	2	1	No	No
Cortex-M4	120 MHz	Full Thumb	3	1	4k I$/4k D$	Single Precision
Cortex-M7	150 MHz	Full Thumb	7	4	16k I$/16k D$	Single + Double Precision

TABLE 3
A comparison of the ARM CPU cores we used to evaluate PANDA.

Operation	Floating Point Cycles	Fixed Point Cycles	Num Operations Per Loop
Addition/Subtraction	1	42	5-10[2]
Multiplication	1	52	10-15[3]
Division	12	430	5
Square Root[4]	14	284	0
Exponentiation	328	42	3

TABLE 4
Number of cycles used by the main operations in fast Gaussian gridding for floating point [40] and fixed point [39] operations on the ARM Cortex-M7 core.

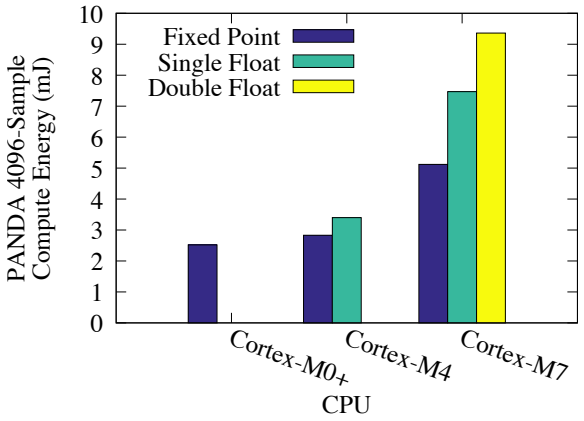

Fig. 12. Comparison of compute energy for PANDA to generate frequency spectrum on three different ARM microcontroller cores.

pipeline. By contrast, the M4 only has a single execution lane that is shared by integer and load/store instructions. The difference in pipeline structure between the two devices could explain the improved performance, but our experiments show improvements near the theoretical maximum of 4-5x speedup. This is likely the result of unintended interleaving between integer, load/store, and floating point operations in the source C code. Our compiler (`gcc`) does not seem to be intentionally interleaving integer and floating point instructions

We have demonstrated that burst-mode data acquisition, when paired with the PANDA NUFFT algorithm, can achieve similar resolution in the frequency domain as uniform data acquisition. The process of sampling in burst mode clearly uses less energy than the uniform sampling alternative, but the data postprocessing takes more CPU time, as outlined in Figure 8. When we put all steps of the process together, how does the overall energy compare between the two techniques?

Using data gathered from our deployment of water quality sensors, we compared data acquisition and postprocessing with the PANDA algorithm and standard uniform sampling using various dataset sizes. Runtime profiling and energy measurements were done on the water quality sensors using historical data that had been gathered over the course of several months. Results of total energy consumption for sampling and frequency spectrum computation are shown in Figure 10.

In the burst-mode sampling used by PANDA, computation energy is a larger component of total energy consumption than in uniform sampling mode. In our experiments with the water sensing platform, PANDA's burst mode sampling outperform uniform sampling for datasets larger than 1024 samples. We get a 17% reduction in overall energy per sample for data windows of 4096.

The reason that the relative compute energy goes down as buffer size increases is that the fast Gaussian gridding step uses a relatively smaller amount of energy for large window sizes. Fast Gaussian gridding uses a banded matrix to resample the burst-mode signal uniformly. The number of bands in the matrix is proportional to the burst size, not the buffer size. For example, if we use a burst size of four samples, we may have a banded diagonal matrix with ten bands, regardless of the length of our data buffer (512, 1024, 2048, or 4096 samples).

The time complexity for factoring an $n \times n$ banded matrix is $O(w^2 n)$, where w is the number of subdiagonals. In fast Gaussian gridding, w depends on the burst length and n depends on the buffer length (total number of samples). So the computational complexity only scales linearly with the buffer length.

4.6 Choosing A Burst Length

PANDA is a useful tool to reduce energy consumption in the digitization of analog sensor signals. As we will demonstrate in this section, sampling signals in burst mode also reduces the accuracy of the acquired signal. There is a tradeoff between accuracy and energy consumption.

There are two parameters that generally affect the accuracy of a burst-mode sampled signal. The first is the number of samples per burst, N_B. The second is the ratio T_B/T_{GAP}, which is the ratio of the time between samples within a burst to the time between bursts (see Figure 3). Generally, we will have direct control over the number of samples per burst and T_{GAP}.

T_B, the interval between successive samples in a burst, is a property of the ADC that we are using—its minimum value is the ADC's conversion time, which varies among devices. To maximize energy savings, we would want to keep T_B at the minimum value that can be achieved by the ADC. In some cases, there may be a reason to increase T_B above that minimum value, but that would require the microcontroller and ADC to be in a higher power mode for a longer period of time, which increases the overall energy

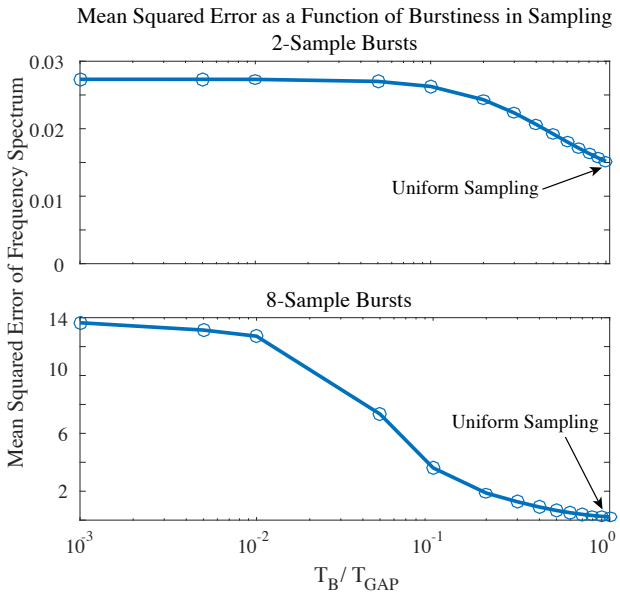

Fig. 13. Mean squared error in the frequency spectrum as a function of burstiness in sampling. On the x-axis, sampling becomes more uniform toward the right side of the graph.

consumption of the sampling process. Modern microcontrollers are typically capable of acquiring 10^6 samples per second (1 megasample per second), making $T_B = 1\mu s$.

The main parameters we will consider tuning are N_B and T_{GAP}. In this experiment, we calculated the frequency spectrum of a bandlimited sawtooth wave that was sampled both uniformly and in burst mode with varying burst parameters. We used the same number of samples grouped into different burst patterns for all trials.

4.6.1 Choosing T_{GAP}

Figure 13 shows how mean squared error (MSE) in the computation of a signal's frequency spectrum varies as a function of T_{GAP}. Note that when $T_B = T_{GAP}$, we have uniform sampling (as labeled in the figure). As we would expect, the errors are generally bigger for more clustered data (smaller T_B) because the mutual information between two closely spaced samples is lower than for two distant samples. Intuitively, we learn less about the signal when we take samples with a small ratio of T_B/T_{GAP} (tight bursts followed by long gaps). But we also spend less energy to acquire the signal when T_B/T_{GAP} is small.

For an ADC that is capable of acquiring 1 megasample per second, we might set $T_{GAP} = 10$ to $100\mu s$ depending on the bandwidth of the signal we are trying to acquire. This would result in the ratio $T_B/T_{GAP} = 10^{-1}$ or 10^{-2}.

To aid in selecting T_B and T_{GAP}, Figure 14 shows the MSE as a function of independent choices of T_B and T_{GAP} for a burst length of eight samples. For longer burst lengths, the plot has a more pronounced sigmoid shape. We have not plotted the MSE for $T_B < T_{GAP}$. The diagonal dashed line in the $x-y$ plane represents the error for uniform sampling.

4.6.2 Choosing N_B

Figure 15 shows MSE as a function of N_B on the x-axis for a fixed $T_B = 100\mu s$. Perhaps nonintuitively, the MSE increases as the burst length increases. As the burst length increases relative to the gap length (moving to the right on

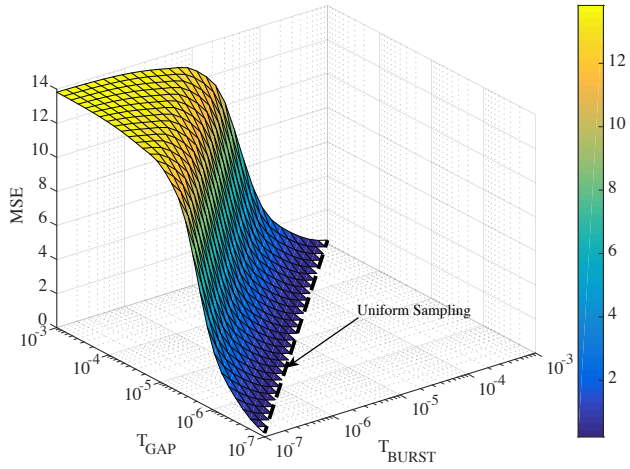

Fig. 14. MSE as a function of T_{BURST} and T_{GAP}.

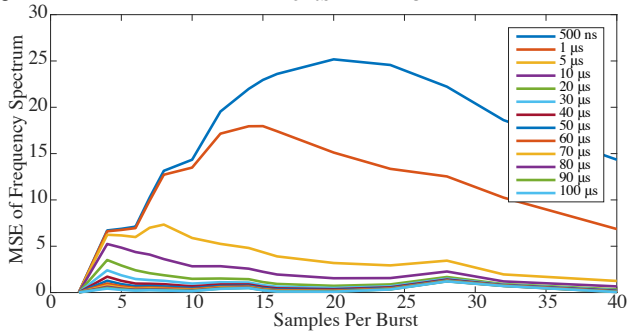

Fig. 15. MSE in the PANDA's computation of the frequency spectrum as a function of burst length. In this experiment, we fixed spacing between bursts (T_{GAP}) at 100 μs and varied the spacing between samples within a burst (T_B) and the burst length (N_B).

the x-axis), we get better resolution in the frequency domain. This was demonstrated in Figure 9. However, the error in the frequency domain computation also increases.

The MSE initially increases as the burst length N_B increases because sampling intervals become less uniform and fast Gaussian gridding constructs a less accurately uniformly resampled signal. But as burst lengths become very long (greater than 16 samples), the length of the burst becomes comparable to the length of the gap between bursts. This has the effect of making fast Gaussian gridding interpolation more accurate because it needs to interpolate points in a relatively smaller gap.

4.7 Memory Requirements

One of the challenges of using PANDA is that its memory requirements are relatively high compared to the conventional FFT. In particular, it needs about 2.5 times as much data memory (RAM) to compute the frequency spectrum. This is because the fast Gaussian gridding operation uses two separate memory buffers (for input and output), whereas the FFT uses only one buffer.

We compiled each of the three algorithms for an ARM microcontroller using the Thumb-2 instruction set and the `-O2` compiler flag to optimize for speed. The code space used by each algorithm is shown in Table 5. None of the algorithms uses a significant amount of data memory—the only data memory used is for storing the input and output buffers. PANDA uses roughly an order of magnitude more code space than the uniform FFT implementations. This

	In Place	KISS FFT	PANDA
Prog Mem	0.89 kbytes	1.98 kbytes	9.2 kbytes
Data Mem	8 bytes/sample	8 bytes/sample	20 bytes/sample

TABLE 5
Code and data memory requirements for each of the three algorithms we evaluated.

could be a drawback for low-end microcontrollers with constraints on code memory. However, with rapid advancements in integration on new microcontrollers, devices with code memory sizes in the range of a few megabytes are now commonly available, making PANDA's NUFFT implementation feasible.

4.8 Implementation Challenges

Floating point constants are automatically promoted to double precision in C. This causes all computations involving constants to be handled as double-precision floating point by default. On the Cortex-M4, which does not have hardware support for double-precision floating point math, all double-precision math would be handled by software, significantly slowing down the computation. To force the compiler to use single-precision floating point math, we must either cast the constant as a `(float)` or use `gcc`'s `-fsingle-precision-constant` compiler switch to demote all constants to single-precision floating point.

Different C library functions must be used for double-precision and single-precision math. Most of the standard C library math routines (like `sqrt()` and `exp()`) use double precision floating point by default for input and output. If we call these functions when compiling for a CPU with no hardware support for double precision floating point math, the computations will be done in software, greatly increasing execution time. Even if the input data type is single-precision floating point, the compiler will promote it to a double and call the double-precision software emulated routine. For single precision data, we must call library routines with the *f* suffix (like `sqrt`**`f`**`()` and `exp`**`f`**`()`) to ensure that the floating point math is done in hardware at single precision.

4.9 Limitations of PANDA

In the right execution environment—one with enough memory and CPU throughput to support fast Gaussian gridding—PANDA can reduce the average energy required to sample a signal and compute its frequency spectrum. On low-end microcontrollers, such as the Atmel device used in our power meter application, we were not able to compile the PANDA code because of code size constraints. We found that on other 8-bit platforms that the fast Gaussian gridding algorithm performs poorly because of lack of instruction set support. In particular, PANDA requires native support for integer division and barrel shift, which are not usually available on low-end 8-bit platforms.

In PANDA, we assume that the microcontroller's CPU must be awake in order for the ADC to take a sample. Some more sophisticated microcontrollers have the ability to trigger ADC conversions without intervention from the CPU and transfer the recorded value directly to memory. This feature is only available on highly-integrated devices, which are unlikely to be used in low-power applications. PANDA is targeted at mid-range devices that are commonly used in low-power applications.

4. Used only once regardless of input size.

5 Conclusion

In the previous section, we demonstrated the utility of PANDA by sampling a signal in burst mode, computing its frequency spectrum, and computing the inverse transform to produce a uniformly resampled signal in the time domain. Using our method, we can save energy by sampling nonuniformly in the time domain while still applying standard frequency domain analysis and filtering techniques.

In our evaluation, we found that a large portion of the energy used by a sensor is consumed in the process of sampling. For this reason, we conclude that there is significant opportunity to reduce the energy consumed in the data acquisition process. However, in order to gather samples in burst mode, we need to use an alternative method of preprocessing the samples to compute the frequency spectrum. PANDA's approach of burst-mode sampling, therefore, only makes sense if the energy used in preprocessing the burst-mode data does not overwhelm the savings in the data acquisition phase. We found that PANDA can have benefits for common applications with average sized data sets.

We analyzed the energy consumed by PANDA's nonuniform DFT algorithm running on a real sensor application using real sensor data. We found that its runtime (and therefore its energy consumption) is higher than that of the conventional FFT for datasets of less than 1024 samples. However, for larger data sets, we demonstrated that PANDA can reduce the energy required to sample a signal and compute its frequency spectrum by as much as 17%.

References

[1] G. Cohn, E. P. Stuntebeck, J. N. Pandey, B. P. Otis, G. D. Abowd, and S. N. Patel, "Snupi: sensor nodes utilizing powerline infrastructure." in *UbiComp*, ser. ACM International Conference Proceeding Series, J. E. Bardram, M. Langheinrich, K. N. Truong, and P. Nixon, Eds. ACM, 2010, pp. 159–168. [Online]. Available: http://dblp.uni-trier.de/db/conf/huc/ubicomp2010.html#CohnSPOAP10

[2] CrossBow, "Mica2 wireless measurement system," http://www.eol.ucar.edu/isf/facilities/isa/internal/CrossBow/DataSheets/m

[3] J. Hester, T. Peters, T. Yun, R. Peterson, J. Skinner, B. Golla, K. Storer, S. Hearndon, K. Freeman, S. Lord, R. Halter, D. Kotz, and J. Sorber, "Amulet: An energy-efficient, multi-application wearable platform," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, ser. SenSys '16. New York, NY, USA: ACM, 2016, pp. 216–229. [Online]. Available: http://doi.acm.org/10.1145/2994551.2994554

[4] S. T. Artem Dementyev, Steve Hodges and J. Smith, "Power consumption analysis of bluetooth low energy, zigbee and ant sensor nodes in a cyclic sleep scenario," in *International Wireless Symposium*. IEEE, 2013.

[5] N. Klingensmith, D. Willis, and S. Banerjee, "A distributed energy monitoring and analytics platform and its use cases," in *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*, ser. BuildSys'13. New York, NY, USA: ACM, 2013, pp. 36:1–36:2. [Online]. Available: http://doi.acm.org/10.1145/2528282.2534156

[6] N. Klingensmith, A. Sridhar, Z. LaVallee, and S. Banerjee, "Water or slime? a platform for automating water treatment systems: Poster abstract," in *Proceedings of the 2nd ACM Conference on Embedded Systems for Energy-Efficient Buildings*, ser. BuildSys '15. New York, NY, USA: ACM, 2015.

[7] N. Klingensmith, J. Bomber, and S. Banerjee, "Hot, cold and in between: Enabling fine-grained environmental control in homes for efficiency and comfort," in *Proceedings of the 5th International Conference on Future Energy Systems*, ser. e-Energy '14. New York, NY, USA: ACM, 2014, pp. 123–132. [Online]. Available: http://doi.acm.org/10.1145/2602044.2602049

[8] S. Microelectronics, "Ta0342: Accurate power consumption estimation for stm32l1 series of ultra-low-power microcontrollers," 2013, http://www.st.com/.

[9] S. DeBruin, B. Campbell, and P. Dutta, "Monjolo: An energy-harvesting energy meter architecture," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '13. New York, NY, USA: ACM, 2013, pp. 18:1–18:14. [Online]. Available: http://doi.acm.org/10.1145/2517351.2517363

[10] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-time Signal Processing (2Nd Ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1999.

[11] J. Yen, "On nonuniform sampling of bandwidth-limited signals," *Circuit Theory, IRE Transactions on*, vol. 3, no. 4, pp. 251–257, 1956.

[12] N. R. Lomb, "Least-squares frequency analysis of unequally spaced data," *Astrophysics and space science*, vol. 39, no. 2, pp. 447–462, 1976.

[13] P. Laguna, G. B. Moody, and R. G. Mark, "Power spectral density of unevenly sampled data by least-square analysis: performance and application to heart rate signals," *Biomedical Engineering, IEEE Transactions on*, vol. 45, no. 6, pp. 698–715, 1998.

[14] F. Marvasti, *Nonuniform sampling: theory and practice*. Springer, 2001, vol. 1.

[15] D. Chen and J. Allebach, "Analysis of error in reconstruction of two-dimensional signals from irregularly spaced samples," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 35, no. 2, pp. 173–180, Feb 1987.

[16] J. I. Yellott, "The photoreceptor mosaic as an image sampling device," in *Advances in photoreception: Proceedings of a symposium on frontiers of visual science*, 1990, pp. 117–134.

[17] R. J. Marks, Ed., *Advanced Topics in Shannon Sampling and Interpolation Theory*. New York, NY: Springer, 1993.

[18] Y.-C. Jenq, "Perfect reconstruction of digital spectrum from nonuniformly sampled signals," *Instrumentation and Measurement, IEEE Transactions on*, vol. 46, no. 3, pp. 649–652, 1997.

[19] P. L. Butzer and G. Hinsen, "Reconstruction of bounded signals from pseudo-periodic, irregularly spaced samples," *Signal processing*, vol. 17, no. 1, pp. 1–17, 1989.

[20] U. J. Mönich and H. Boche, "Non-equidistant sampling for bounded bandlimited signals," *Signal processing*, vol. 90, no. 7, pp. 2212–2218, 2010.

[21] D. Donoho, "Compressed sensing," *Information Theory, IEEE Transactions on*, vol. 52, no. 4, pp. 1289–1306, April 2006.

[22] X. Wu and M. Liu, "In-situ soil moisture sensing: measurement scheduling and estimation using compressive sensing," in *Proceedings of the 11th international conference on Information Processing in Sensor Networks*. ACM, 2012, pp. 1–12.

[23] H. Hassanieh, P. Indyk, D. Katabi, and E. Price, "Nearly optimal sparse fourier transform," in *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '12. New York, NY, USA: ACM, 2012, pp. 563–578. [Online]. Available: http://doi.acm.org/10.1145/2213977.2214029

[24] ——, "Simple and practical algorithm for sparse fourier transform," in *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '12. SIAM, 2012, pp. 1183–1194. [Online]. Available: http://dl.acm.org/citation.cfm?id=2095116.2095209

[25] S. Zhai, L. Guo, X. Li, and F. X. Lin, "Decelerating suspend and resume in operating systems," in *Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '17. New York, NY, USA: ACM, 2017, pp. 31–36. [Online]. Available: http://doi.acm.org/10.1145/3032970.3032975

[26] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 2, no. 4, pp. 437–445, 1994.

[27] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye, *Energy-driven integrated hardware-software optimizations using SimplePower*. ACM, 2000, vol. 28, no. 2.

[28] A. Dutt and V. Rokhlin, "Fast fourier transforms for nonequispaced data, {II}," *Applied and Computational Harmonic Analysis*, vol. 2, no. 1, pp. 85 – 100, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S106352038571007X

[29] C. Anderson and M. D. Dahleh, "Rapid computation of the discrete fourier transform," *SIAM J. Sci. Comput.*, vol. 17, no. 4, pp. 913–919, Jul. 1996. [Online]. Available: http://dx.doi.org/10.1137/0917059

[30] L. Greengard and J.-Y. Lee, "Accelerating the nonuniform fast fourier transform," *SIAM review*, vol. 46, no. 3, pp. 443–454, 2004.

[31] W. H. Press and G. B. Rybicki, "Fast algorithm for spectral analysis of unevenly sampled data," *The Astrophysical Journal*, vol. 338, pp. 277–280, 1989.

[32] B. P. Flannery, W. H. Press, S. A. Teukolsky, and W. Vetterling, "Numerical recipes in c," *Press Syndicate of the University of Cambridge, New York*, vol. 24, 1992.

[33] N. Klingensmith and S. Banerjee, "PANDA: Performance acceleration through nonuniform data acquisition," in *Proceedings of the Ninth International Conference on Future Energy Systems*, ser. e-Energy '18. New York, NY, USA: ACM, 2018, pp. 200–210. [Online]. Available: http://doi.acm.org/10.1145/3208903.3208904

[34] E. Margolis and Y. C. Eldar, "Nonuniform sampling of periodic bandlimited signals," *Signal Processing, IEEE Transactions on*, vol. 56, no. 7, pp. 2728–2745, 2008.

[35] F. Marvasti, M. Analoui, and M. Gamshadzahi, "Recovery of signals from nonuniform samples using iterative methods," *Signal Processing, IEEE Transactions on*, vol. 39, no. 4, pp. 872–878, 1991.

[36] M. Borgerding, "KISS FFT," http://kissfft.sourceforge.net.

[37] T. Roberts, "Fixed-point fast fourier transform," 1989, https://gist.github.com/Tomwi/3842231.

[38] W. Cochran, J. W. Cooley, D. Favin, H. Helms, R. Kaenel, W. Lang, J. Maling, G.C., D. Nelson, C. Rader, and P. D. Welch, "What is the fast fourier transform?" *Proceedings of the IEEE*, vol. 55, no. 10, pp. 1664–1674, Oct 1967.

[39] P. Aimonen, "libfixmath," 2017, https://github.com/PetteriAimonen/libfixmatrix.

[40] "Floating point unit demonstration on stm32 microcontrollers," 2016.

Neil Klingensmith is a graduate student in Computer Engineering at the University of Wisconsin. He is also co-founder of Emonix, a startup that makes IoT devices for commercial and industrial building management. His work focuses on software and hardware for resource constrained embedded, mobile and IoT applications.

Suman Banerjee received his PhD in Computer Science from University of Maryland in 2003 and joined the faculty of University of Wisconsin-Madison. He is the winner of the ACM SIGMOBILE Rockstar Award in 2013 (awarded for early career achievements in the field of mobile computing and wireless networking). Suman leads the WIsconsin WIreless and NetworkinG Systems (WiNGS) Laboratory at Wisconsin, which conducts research in the areas of networking and distributed systems with a primary focus on wireless and mobile networking.