Preventive Detection of Mosquito Populations using Embedded Machine Learning on Low Power IoT Platforms

Prashant Ravi
prashant014@e.ntu.edu.sg
School of Computer Science and Engineering
Nanyang Technological University
Singapore

Uma Syam
uma005@e.ntu.edu.sg
School of Computer Science and Engineering
Nanyang Technological University
Singapore

Nachiket Kapre
nachiket@uwaterloo.ca
Electrical and Computer Engineering
University of Waterloo
Waterloo, Canada.

ABSTRACT

We can accurately detect mosquito species with 80% accuracy using frequency spectrum analysis of insect wing-beat patterns when mapped to low-power embedded/IoT hardware. We combine energy-efficient hardware acceleration optimizations with algorithmic tuning of signal processing and machine-learning routines to deliver a platform for insect classification. The use of low power accelerator blocks in cheap embedded boards such as the Raspberry Pi 3 and Intel Edison, along with performance tuning of the software implementations enable a competitive implementation of mosquito classification task on standard datasets. Our approach demonstrates a concrete application of embedding intelligence in edge devices for reducing system-level energy needs instead of simply uploading sensory data directly to the cloud for post-processing. For the mosquito classification task, we are able to deliver classification accuracies as high as 80% with Intel Edison processing times as low as 5 ms per set of 8K audio samples and an energy use of 5 mJ per sample (2 months of continuous non-stop use on an AA battery with 2000 mAh capacity or longer depending on insect activity). We envision a network of connected sensors and embedded/IoT platforms deployed in vulnerable areas such as construction sites, mines, areas of known mosquito activity, ponds, riverfronts, or other areas with standing water bodies. In our experiments, targeting a 20% packet loss rate, we observed the ad-hoc WiFi range for mesh networks using the Raspberry Pi 3 boards to be 14 m while the Photon board connecting to infrastructure WiFi router nodes can stretch this to 35 m.

1. INTRODUCTION

Vector-borne diseases such as malaria and dengue account for 17% of all infectious diseases, causing more than a million deaths annually [1]. It is estimated that almost 2.5 billion people in over 100 countries are at risk of contracting dengue alone. Malaria causes 600 000 deaths every year globally, most of them children under 5 years of age. To compound matters further, other diseases such as Chagas disease, leishmaniasis, and schistosomiasis affect hundreds of millions of people worldwide. In Figure 1, we show the geographical distribution of Zika and dengue incidents reported across the world over a three month period from February to May 2016. The data shows that the highest density of dengue and Zika incidents are concentrated in the Americas and South-East Asia making this a challenge in many developing countries. However, with high incidence rates in the US, Canada, and mainland Europe, this challenge is shared by the developed world as well. Disease control has a particularly severe socio-economic impact on the developing world. The impact of dengue fever in just eight countries alone: Brazil, El Salvador, Guatemala, Panama, Venezuela, Cambodia, Malaysia, and Thailand results in monetary losses of $1.8 billion every year [2]. In the Americas, dengue fever was estimated to cost, on average, about $2.1 billion annually (2009 estimates [3]). Despite these abysmal statistics, many of these diseases are preventable through informed protective measures [1]. Considering these high costs of treatment, it is considerably cheaper to invest in preventive measures.

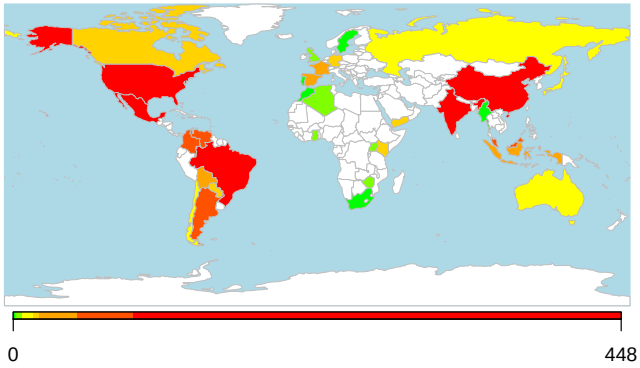

Figure 1: Global Distribution of Dengue, Zika and Chikungunya incidents from April to June 2016. Data from Healthmap.org [4]. Scale from green (low) to red (high).

There are a number of existing methods that different countries use for early detection of vector borne diseases. For example,

- In **India**, the method of *Sentinel Surveillance* [5] is used in order to obtain data regarding dengue and chikungunya

ACM DEV '16, November 17 - 22, 2016, Nairobi, Kenya

ISBN 978-1-4503-4649-8/16/11...$15.00

DOI: http://dx.doi.org/10.1145/3001913.3001917

outbreaks under the National Vector Borne Disease Control Programme (NVBDCP). Under this approach, a set of specific *sentinel* hospitals are monitored for cases of vector-borne diseases [6]. Once alarms are raise in these sentinel hospitals, corrective measures are activated as appropriate. The cost of diagnostic facilities and kits limits the deployment of this programme to specific sentinel hospitals. As a result of these cost-constrained measures, the actual number of dengue cases were under-reported by a factor of 282× [7] in 2006-2012.

- The National Environment Agency (NEA) in **Singapore** creates awareness regarding dengue outbreaks by marking geographical clusters [8] that are prone to outbreaks based on historical data. A cluster is declared when two or more cases have emerged within 14 days and are located within 150 m of each other. This method relies on waiting for two infections to be detected before kick-starting control measures. Unlike India, the economic pressures on Singapore are lower, and hence the analysis of data is based on hospitals island-wide across the tiny country. The NEA has offices and well-equipped labs in all hospitals across the country to track these metrics.
- **Mexico** uses *Syndromic Surveillance* [9] to trigger implementation of control measures. This is based on collecting data on patient symptoms during doctor visits. While this approach still waits for symptoms to manifest, control measures are applied before a disease is confirmed, thereby saving valuable time.

Thus, despite some variations in effectiveness and coverage, most of the early detection mechanisms discussed above wait for the disease to actually manifest through symptoms of confirmed outbreaks in human patients before any control action is taken to prevent a wider outbreak. Countries and national agencies need not wait for symptoms or confirmation of diseases prior to activating control measures. In this paper, we focus on a technological solution to these real-world challenges through the design of early-warning electronic systems. In particular, we are interested in low-cost implementation of electronic components of a sensory infrastructure that processes wing-beat frequencies to identify and classify species of mosquitoes. We combined machine learning techniques with hardware-software co-optimization of the classification algorithm on a low-power embedded hardware platform to deliver our solution. The classified data could be transmitted over any telematics medium to the disease control authorities in order to initiate preventive measures. We envision a mesh of connected low-cost, low-energy classifier hardware units operating in high vulnerability zones such as swamps, construction sites, flood-prone roads, hospitals and other areas. An early-warning system built on analysis of this data can help deploy limited preventive resources in a developing environment where it is needed the most. While we focus on mosquito analytics in this paper, there are many other vectors that can transmit infectious diseases between humans or from animals to humans. Beyond mosquitoes, other vectors include ticks, flies, sand-flies, fleas, triatomine bugs and some freshwater aquatic snails but their impact on spreading disease is relatively low. Our technological platform approach can be easily reconfigured to support detection of specific features of other vectors as appropriate.

The key contributions of this paper include:

1. Optimization of the software stack for implementing analytics of wing-beat frequencies on an embedded platform through precision analysis, and parallelization.
2. Performance and Power-Usage Analysis and Tuning of the software stack on a set of hardware platforms with on-chip accelerators, such as the Raspberry Pi 3, and Intel Edison.
3. Radio power and throughput analysis for low-energy communication infrastructure.
4. Evaluation of the effectiveness of our classification approach on the Mosquito wing-beat dataset.

2. WING-BEAT CLASSIFICATION

While there are various solutions for analyzing and tracking insect populations, they are labor intensive and are useful for passive scientific analysis rather than active predictive purposes. Microsoft Project Premonition [10] aims to develop cloud-connected insect traps and drone-based insect collection devices. It relies on gene sequencing of pathogens to track their propagation in the environment. Instead of this expensive trap-based approach, our idea is to rely on inexpensive wing-beat sensors [11] that track insect wing-beat frequencies to classify and track species. This approach forms the basis of this paper.

2.1 Background

The effectiveness of the classification depends on the reliability of wing-beat frequency to classify insects as belonging to a particular species. The work in [11] shows that fundamental frequency as a classification parameter can deliver high accuracy detection of mosquito species. This idea of using wing beat frequency to classify insects dates back to the early 1900s [12]. During these early years, the scientists used bulky acoustic microphones for recording the wing-beat sound. But this technique possesses various limitations like sensitivity to wind and to ambient noise [13]. Batista et. al overcame this limitation by proposing a clever low-cost optoelectronic sensor arrangement [11].

The sensor arrangement consists of a laser beam and a photo-transistor array which is connected to an electronic board, and the laser is made to point at the array. When an insect flies across the laser beam, it causes small light fluctuations which are captured by the photo-transistor array as changes in current, and the signal is filtered and amplified by the custom designed electronic board which converts the changes in current into sound. The laser beam uses laser pointers as cheap as 99 cents and phototransistors similar to those employed in commodity TV remote controls. According to the authors, the entire sensor setup can be built in under $5.

Once the wing-beat sound is recorded, we must post-process the data to identify the insect species. The detection of certain kinds of insects or species can then be used by the health and environmental authorities to deploy corrective measures. The solution proposed in [11] requires logging the audio data in mp3 format followed by offline data analysis of the signals using analytics techniques implemented in Matlab on a PC. This approach is manual, time-consuming and delays corrective measures. A preliminary implementation that attempts to embed processing in the sensor was presented in an Arduino DUE [14]. However, the solution is power hungry and was not suited for high throughput classification in battery-operated environments. Our work provides a comprehensive exploration and solution for selection of a suitable, low power, embedded platform along with

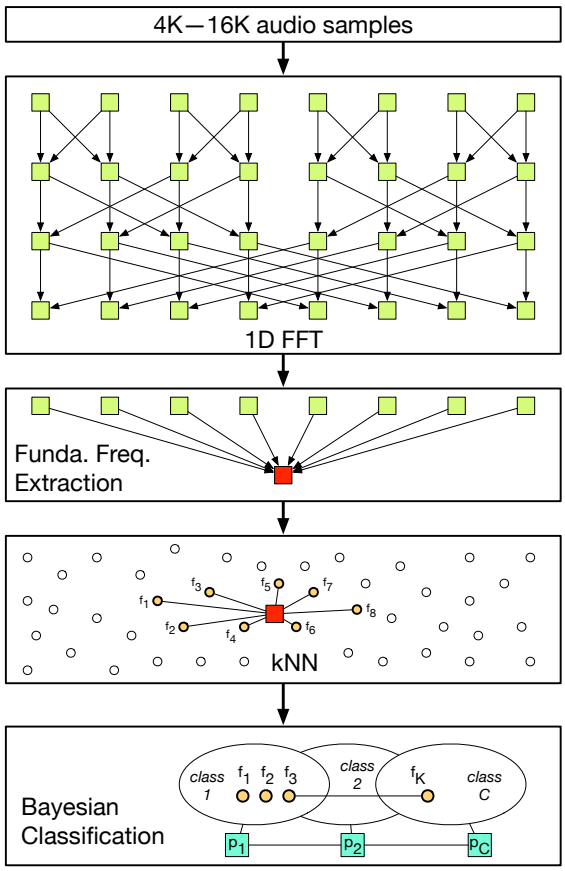

Figure 2: High-Level View of the Computational Flow of Insect Classification. (For our application, kNN is a 1-D problem)

associated software and communication optimizations.

2.2 Algorithmic Flow

We show a high-level view of the complete wing-beat classification flow in Figure 2. The key steps of the underlying algorithm are explained below:

- **FFT**: First, we compute the Fourier Transform of the unknown wing-beat audio samples to determine the fundamental frequency, which is the parameter used for the classification. The FFT can be mathematically represented as:

$$X(k) = \sum_{n=0}^{N-1} x(n).e^{-j(\frac{2\pi}{N})nk}; k = 0, 1, ..., N-1 \qquad (1)$$

  where, N is the number of samples in input signal $x(n)$
  The fundamental frequency is determined as the frequency corresponding to the maximum FFT magnitude in the wing-beat frequency range.
- **k-Nearest Neighbor** (kNN): The next step is to compute the distance of the unknown sample's fundamental frequency with every value in the training frequency set. We compute the Euclidean distance for the one-dimensional data set as the absolute value of the difference between the two frequencies, which is given as:

$$D = |F_i - F_o|; \qquad (2)$$

  F_i is the i^{th} sample frequency from the training set.
  F_o is the fundamental frequency of the input audio.
  The computed distances are inputs to the kNN algorithm and indicate the probability of belonging to a particular species. The kNN algorithm identifies the k nearest neighbors to the unknown mosquito's frequency. This is given by the k frequencies corresponding to minimum distance values. The value of k with the best accuracy works out to be eight for our dataset. To keep the system scalable, we do not continually update the clusters with newly sampled data.
- **Bayesian Classifier**: While there are multiple classification algorithms, we select the Bayes classifier due to its performance in minimizing the probability of misclassification [15]. The classifier is conservative in both CPU and memory requirements which makes it particularly well-suited for an embedded mapping. The ability to allow the incorporation of auxiliary features also supports our classification choices for this task, as it can gracefully incorporate evidence from multiple sources and in multiple formats. The probability values are The Bayesian classifier identifies the unknown insect sound based on probability computations. computed from the knowledge of eight nearest neighbors and the fraction of frequencies clustered around a particular insect class. The conditional probability of extracted frequency belonging to a class is computed as:

$$P(F_1/C_i) = k_{C_i}/k; i = 1, 2, 3 \qquad (3)$$

  $P(F_1/C_i)$ is the probability of the fundamental frequency of unknown insect F_1 belonging to a class C_i.
  k_{C_i} is the number of frequencies belonging to class C_i among the eight frequencies selected earlier.
  Finally, the insect sound is classified as the class which resulted in maximum probability.

3. IMPLEMENTATION

In this section, we describe the key implementation considerations when mapping the insect classification task to an embedded device. Our approach addresses several crucial engineering concerns such as (1) addressing the selection of a suitable embedded device(s) for the complete task from the vast set of possible IoT platforms, (2) the extent of associated algorithmic and software transformations that are possible for improving performance and lowering energy use , as well as (3) determining the ideal balance between radio communication and embedded computation in the device.

3.1 Choice of Embedded Hardware

In the extreme, we only really need a sensor connected to a radio-capable board with Bluetooth or WiFi support that allows easy uploading of sensory data directly to some central server node. While this seems convenient, the energy cost of radio communication of raw data is high. For instance, it would cost us 300–400 mJ of energy to communicate 8K samples of audio data using a highly efficient Particle Photon board with an efficient Broadcom BCM43362 WiFi chip. Under continuous use, this translates into a mere 20 hour lifetime with a 2000 mAh AA battery. Instead, we investigate the potential to perform lightweight computation on the embedded board to reduce the communication bandwidth needs and lower overall energy use. For instance, an optimized implementation of the algorithm, shown earlier

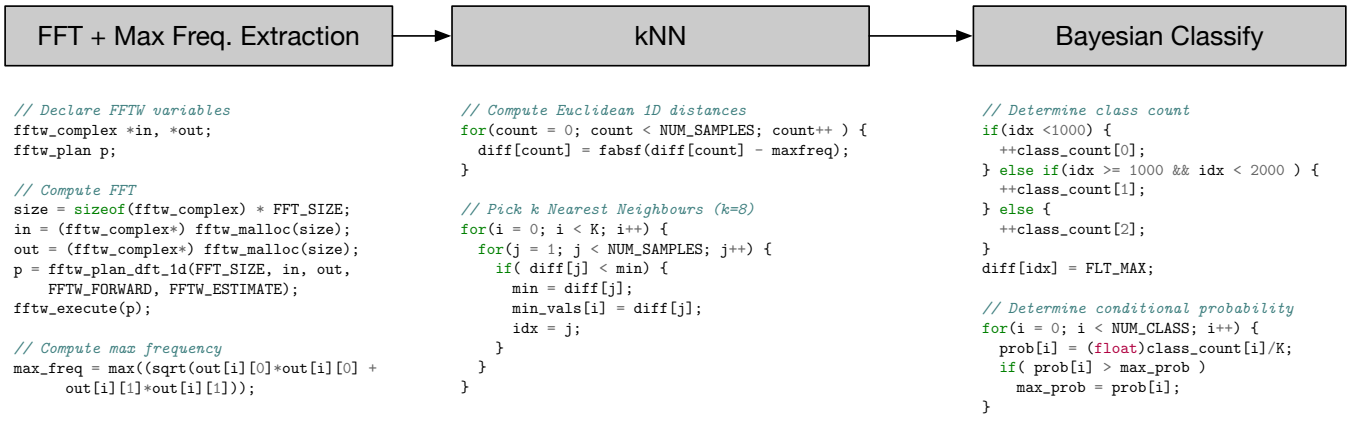

Figure 3: Pseudocode Sketches of the underlying algorithm and arithmetic.

in Figure 2, on an Intel Edison platform, we require 5 ms of compute time and under 5 mJ of energy. This represents an ≈80× reduction in best-case energy use stretching the battery life to around 2 months. This allows the device to be powered for long periods of time over a battery and be deployed even in remote, hard-to-service locations such as swamps, gutters, and large construction sites.

Specifically, we consider a set of boards optimized for compute (Arduino DUE, Raspberry Pi 3, and Intel Edison) along with those optimized for communication (Particle Photon 3). Our eventual solution combines two platforms instead of using a single one for both tasks resulting in sub-optimal overall behavior. In order to lower overall power consumption, it is essential that we rapidly perform our computations in short bursts as energy consumed is directly proportional to time. We can then put the systems in low power standby modes wherever supported to extend battery lifetime. The communication hardware is activated infrequently to send summarized digests once a day as appropriate.

- **Arduino**: They are trendy and easy to program devices that are available off-the-shelf. The low peak power profile of the Arduino, the vast community support, ease of peripheral integration, makes it particularly attractive. However, their compute capacity is somewhat limited, particularly when compared to other platforms such as the Raspberry Pi 3 and Intel Edison. Arduino DUE does not natively have radio/WiFi support but can be extended with a suitable shield extension card.
- **Particle Photon**: This is a relatively new development board with similar peripheral support like the Arduino. The Photon has a much more powerful processor and also has an in-built Broadcom WiFi module with its own cloud support. This makes the Photon particularly well-suited for our task requiring communication capabilities.
- **Raspberry Pi 3** and **Intel Edison**: In comparison with the Arduino and the Particle Photon, the Intel Edison and the Raspberry Pi 3 platforms are far more powerful. Their processors also provide in-built hardware acceleration blocks that are well-suited for energy-efficient acceleration of computations such as FFTs. In addition, some of these boards can be put into a low-power sleep mode when not in use in order to save energy even further.

3.2 Optimizations

At a high level, we have the opportunity to optimize the implementation of the classification code in various ways. We show the software code sketches of the three key computational blocks in Figure 3.

Hardware Accelerators: Preliminary mapping experiments on the different boards suggested that the FFT component takes up almost 80–90% of the overall runtime of the code. The FFT component can be accelerated through the use of hardware accelerator blocks such as SSE [16] (Intel Edison) or VideoCore GPUs (Raspberry Pi 3). These accelerators can help speedup this component by up to 2× (see Section 5 for more details). Furthermore, with the use of parallelism across multiple cores (Intel Edison), there is scope for further performance improvements through the explicit use of OpenMP [17] multi-threading extensions. The rest of the code blocks in Figure 3 are organized in the form of for-loops. These are also easy to parallelize and map to the SIMD accelerator units such as SSE (Intel Edison) and NEON [18] (Raspberry Pi 3). Thus, we can push the hardware capabilities to their fullest extent by careful use and exploitation of available accelerator resources. The easy-to-use Arduino environments typically do not expose these opportunities resulting in wasted capacity.

Precision Reduction: The overall effectiveness of the implementation is measured in terms of the final classification accuracy of the code. The baseline implementation uses expensive floating-point hardware that is slower and results in high storage costs for the 32b data types. We also consider the impact of reducing precision to 16b fixed-point data types. This allows us to use integer arithmetic units and greater parallelism in the SIMD blocks (SSE and NEON) along with reduced storage costs. This reduction in precision is supported through the use of KissFFT library [19] that allows configurable data-width support.

Compiler optimizations: Finally, it is possible to direct the C/C++ compilers to optimize your implemented code in specific ways. For instance, to compile the code with OpenMP pragmas we must use the compiler flag `-fopenmp`. In case of Intel Edison, we had to use the `-msse2` flags to explicitly make use of the SSE unit[20] [16]. This auto-vectorizes the code and run parts of the code in parallel in the SSE unit. In the Raspberry Pi 3, this auto-vectorization on the NEON unit is enabled by the flag `-Ofast`. Furthermore, we also use loop unrolling, reassociation and other arithmetic transformations by means of compiler flags to tradeoff precision for speed. This is acceptable as long as overall classification accuracy is unaffected.

Table 1: Comparing Embedded/IoT Platforms.

	Arduino DUE	Particle Photon	Raspberry PI 3	Intel Edison
Arch.	Cortex-M3	Cortex-M3	Cortex-A53	x86
Cores	1	1	4	2
Cache	32KB L1	32KB L1	16KB L1, 512KB L2	56KB L1, 1MB L2
DRAM	96KB	128KB	1GB	1GB
Freq. (MHz)	84	120	1200	500

4. METHODOLOGY

In this section we describe the key technical specifications of the various embedded/IoT platforms and compare their potential. It is important to note that these datasheet specifications may not directly translate into equivalent application runtimes due to various factors such as accelerators, software optimizations, and other platform-specific considerations. However, they give us an overview of the rich set of platforms that are available with varying capabilities and relative strengths.

In Table 1, we enumerate the important metrics of the various platforms in terms of CPU capacity, memory potential and speed. We also highlight the radio capabilities of these platforms as they are important for quantifying communication energy and time to compile the classification results. We list the compilation stacks, libraries, and operating systems used in Table 2.

The fundamental difference that separates these platforms is the choice of ARM-based processors or x86-based ones. The vast majority of embedded software tools and ecosystem is already built around the ARM platform with the x86 systems being very recent entrants into this space. For the IoT/connectivity-oriented applications, the distinction may not matter much if the energy profile meets application requirements. Even within the ARM family of products, there are micro-architectural differences between the low-cost, low-performance, low-energy Cortex-M3s and more capable, but higher energy Cortex A-53s. Hence, we must tune the classification algorithm carefully for each device so as to extract best results. Barring the Arduino which needs special shields, the chosen platforms support WiFi protocol stacks out of the box which is crucial for communication of classification results efficiently.

The Arduino DUE and Particle Photon are relatively low-end platforms, but they are cheap and can be programmed very quickly using well-established, easy-to-use tool flows with lightweight OSses or bare-metal modes. They have relatively low on-chip RAM capacity which makes it difficult to process long sample sequences of wing-beat recordings. In contrast, the Raspberry Pi 3 and Intel Edison platforms are significantly more powerful, support larger RAM capacities and can even support complete operating systems. This directly translates into a greater energy efficiency potential for the more capable boards as they are able to rapidly process incoming data with minimal expensive off-chip activity. Both the Raspberry Pi and Edison platforms provide accelerators that can help significantly speed up data-parallel computations. For instance, nested for loops can work in SIMD parallel fashion on NEON (Raspberry Pi) and SSE (Edison) hardware units. Furthermore, the Raspberry Pi has a Broadcom VideoCore IV GPU that can run graphics computations in parallel. With suitable extensions, we can use the GPU to perform other parallel tasks such as FFTs that are a key component of our application.

Table 2: Software Toolchains

Device	Raspberry Pi 3	Intel Edison	Particle Photon
OS	Debian Jessie Lite	Ubilinux	Free RTOS
Kernel	4.14	3.13	N/A
Compiler	gcc 4.8.4	gcc 4.8.1	arm-none-eabi-gcc 5.3.1
Acceleration Library	GPU_FFT release 3.0	OpenMP 4.0	None
Timing Lib.	PAPI 5.4.3	PAPI 5.4.3	Particle API

Table 3: Power Usage of Embedded/IoT Platforms.

Board	**Voltage** (V)	**Current** (mA) I_1	I_2	I_3	I_4
Raspberry Pi 3	5.1	0.2	0.2	0.24	0.26
Intel Edison	5.1	0.03	0.08	0.09	0.15
Particle Photon	5.1	0.02	0.04	0.06	0.06

I_1=Sleep Current, I_2=Stead-state no radio, I_3=Stead-state with radio, I_4=active current.

A final factor that matters to an energy efficient implementation is the cost of leakage current or effectiveness of standby modes. We expect our devices to be activated infrequently when a mosquito or insect flies past the sensor assembly. This means, we must be able to power down the board very effectively and wake up on demand without wasting energy. Particularly, in the case of a battery-operated unit out in the field, we want to maximize the lifetime of operation. In Table 3, we tabulate the standby currents of the different platforms. All currents are represented in mA. I_1 refers to sleep current, I_2 refers to steady state current without radio, I_3 refers to the steady state current with radio and I_4 represents the current drawn during computation. The Photon offers the lowest currents across the board while Edison is superior to the Raspberry Pi 3 in this comparison.

4.1 Wing-beat Datasets

We use audio samples from [21] for our analysis. The benchmark samples represent three species of mosquitoes being classified, namely *Culex stigmatosoma (female), Aedes aegypti (female), and Culex tarsalis (male)*. The dataset contains a thousand audio samples per species for a total of three thousand samples. The original audio files provided are used for training the classification models. A subset of these samples is used for testing and evaluating prediction accuracy. A Matlab-formatted *mat* file contains fundamental frequencies corresponding to three thousand audio files that are used for verification of correctness.

Each audio file contains the mosquito wing-beat sound centered and zero-padded elsewhere to create a one-second long audio sample. This results in each audio file containing 16384 samples. The raw data is processed in floating-point or fixed-point representations depending on the hardware platform. We also extract subsets of the audio stream to evaluate its effect on accuracy while significantly saving on FFT compute times.

5. RESULTS

In this section, we describe the results of various performance and energy metrics of the different software algorithms on the hardware platforms. We also quantify the accuracy of classification on mosquito detection datasets to help certify the correctness of our implementations.

5.1 Initial Benchmarking

We first microbenchmark the various IoT/embedded platforms for suitability of use for our problem. While we narrow down the vast set of potential platforms to those listed in Table 1 shown earlier, we must identify a few promising candidates. We run FFT computations cross various boards used in our study and plot the resulting trends, as a function of a number of input samples, in Figure 4. As expected, the runtime increases with sample count with large gaps opening up between the different platforms. The popular Arduino board is the slowest of them all taking as much as 2–3× more time than the nearest competitor. The Particle Photon board is popular among IoT enthusiasts due to its connectivity options, but it is only marginally faster than the Arduino. The two stand-out platforms in this initial analysis are the Intel Edison and the Raspberry Pi 3 boards. Their runtimes are as much as 100× faster than the Arduino implementations. This speed is important for intelligent edge devices. If we can quickly digest large volumes of audio data into small nuggets of information that can be periodically sent to a data collection server, we can lower the overall system cost and the energy requirements for radio transmissions at the edge. This is because energy use is proportional to time as most platforms roughly consume 2–3 W of power. For small sample counts, the Raspberry Pi 3 offers faster evaluation but the Intel Edison outperforms the Pi at sample counts beyond 2K. This crossover is important, as the larger sample counts generally deliver higher classification accuracy for detection.

Now that we have narrowed down the promising platforms to two, we try to understand the runtime breakdown across the different constituent components of the classifier. As shown in Figure 5 for 8K samples, we see that the FFT dominates overall runtime on both platforms. This is not unexpected as the FFT computation scales as $N \times log(N)$ where N is the number of samples. The data read from the sensor consumes <10% of the time while the final K-means and Bayes classification also take up ≈10% or less. As we can see, the total runtime gap between the two boards

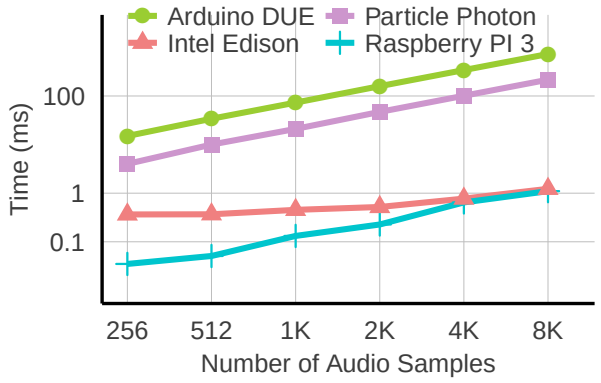

Figure 4: Initial Benchmarking of the different Embedded/IoT Hardware Platforms.

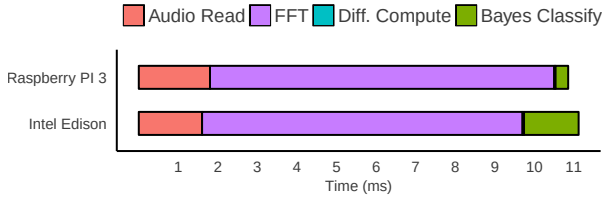

Figure 5: Performance runtime breakdown of different compute stages of the Classifier.

is not large, but when logging a large number of detection events, this translates into non-trivial energy reduction. The plot also suggests that we need to first focus our optimization efforts on the FFT component of the flow as it is the most dominant. As we will show subsequently, we can do this either through software optimizations such as precision modification or sample pruning, or through acceleration.

5.2 Hardware Optimization

We now investigate the impact of using hardware accelerators and optimization flags that help produce high-performance code on the two embedded platforms. In particular, we look at the low-hanging fruits such as compiler flags, and the use of optimized libraries. The first case we consider labeled as `opt1` simply chooses the `-Ofast` flag in GCC when compiling the classifier code. This automatically enables vectorizing optimizations such as the use of SSE instructions on the Intel Edison, and NEON instructions on the Raspberry Pi's ARM CPU. The case labeled as `opt2` takes into account further optimizations that are possible through the use of additional coding to use parallelism in the form of multiple threads on the dual-core Intel Edison and GPU offload that is possible on the exotic VideoCore GPUs of the Raspberry Pi 3.

First, in Figure 6, we show the shows the impact of acceleration on the overall execution time of the classifier. The trends show a significant 2–3× reduction in the overall computation time for both devices when various stages of optimizations are applied. The computations are done for 8K samples of data, (1) in floating point precision on Raspberry Pi 3, and (2) fixed-point precision on Intel Edison so as to compare the fastest possible implementations on the respective platforms. This choice has an impact on accuracy as we show later in Section 5.3. Without any optimizations, both platforms are roughly equivalent in their capabilities. As we start to apply optimizations cumulatively, a performance gap opens up between the two platforms. The use of

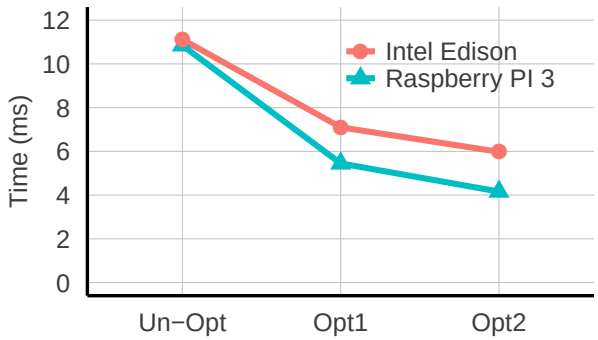

Figure 6: Impact of Hardware and Software optimizations on performance.

simple SIMD optimizations (`-Ofast`) delivers the first 1.5–2× reduction in runtime. The next stage of optimization involves GPU offload (Raspberry Pi 3) and OpenMP (for Intel Edison). When this optimization combines with the earlier optimization, we achieve a cumulative 2–3× reduction over the baseline unoptimized mapping. Thus, simply through the use of SIMD parallelism, and multi-threading (OpenMP) or GPU offload, we are able to accelerate our code by a large margin. Thus, we observe an overall reduction in wall-clock time from 10–11 ms down to 4–6 ms for the classifier operating on 8K samples of input.

5.3 Software Optimization

Now that we are able to squeeze 2–3× in performance through the clever use of embedded hardware, we turn our attention to understanding the accuracy impact of precision and sample count pruning on overall accuracy. Ultimately, we want to classify insects with highest possible accuracy at a reasonable cost in runtime or energy per classification event. Thus, we consider two FFT libraries `fftw` and `kiss-fft` that implement the FFTs in floating-point and fixed-point precision respectively. These are compiled and optimized for embedded platforms. For reference, we also consider a version of FFT that has been hand-coded for the VideoCore GPU in assembly – we term this `gpu-fft`.

In Figure 7, and Figure 8, we plot the accuracy trends for the various libraries when considering the Intel Edison and Raspberry Pi 3 platforms respectively. The original insect classification paper [13] used 16K samples for the classification processes, but we evaluate shorter lengths here as well. In both plots, we see an accuracy peak at 8K samples, and only a slight degradation at 4K samples (mostly equivalent to the 16K accuracy). This trend could be associated with the fact that the 16K samples are mostly zeroes used for padding. This suggests that there is an opportunity to significantly reduce classification time and energy by simply using a smaller burst of data. This may correspond to the periodicity of the wing-beat patterns of various flying insects and would likely need to be tuned per insect profile. We also see that fixed-point `kiss-fft` generally delivers equivalent accuracy as compared to floating-point precision thereby helping reduce runtime and energy further. However, when using the Raspberry Pi GPU, the use of sample lengths smaller than 16K results in a monotonic decrease in classification accuracy. This is likely due the non-IEEE compliant nature of the floating-point arithmetic units on the GPU that are optimized for visual accuracy rather than pure arithmetic accuracy. Finally, the 33% classification floor is due to the fact that our classifier generates one of three mosquito classification outputs and is likely to be correct by sheer luck one out of three times even when predicting a fixed outcome. For practical purposes that is equivalent to a null result.

As indicated earlier, the scaling trends of an FFT computation are $N \times log(N)$ which suggests a potential for substantial reduction in runtime as we reduce N. In Figure 9, we show the effect of the number of samples of audio data used for classification on the accuracy of classification and runtime of the computation. It can be seen that irrespective of the type of FFT we use, when the number of samples used for computation increases, the computation time increases as expected. Furthermore, correlating from Figure 7 and Figure 8, we also observe that accuracy is higher for

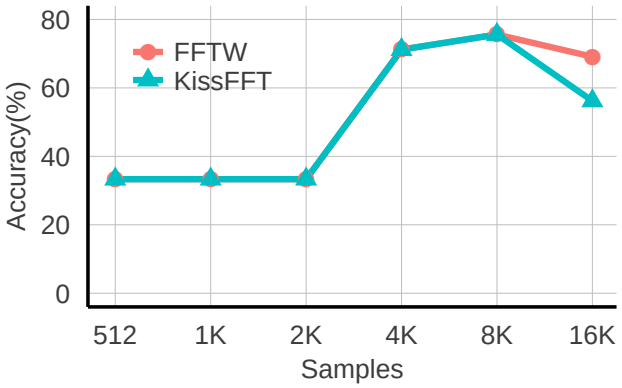

Figure 7: Effect of FFT size, and FFT precision on accuracy (on Intel Edison)

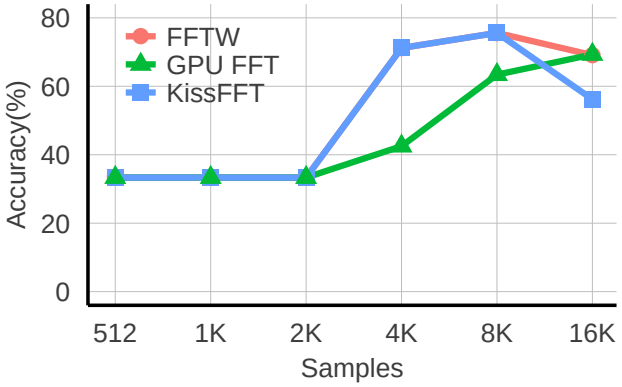

Figure 8: Effect of FFT size, FFT precision, and GPU accelerator on accuracy (on Raspberry Pi 3)

8K samples than 16K. More importantly, we see a 1.2–1.4× reduction in runtime for a 5–20% increase in accuracy when moving from 16K samples to 8K samples. The use of 4K samples results in only a ≈4% reduction in accuracy (to 71%) but a 1.1–1.4× reduction in time.

5.4 Energy Analysis

Finally, we quantify the energy requirements of the different optimized implementations and contrast those with communication energy. This data should help clarify the need for embedding intelligence in the edge devices rather that merely uploading all audio samples directly to a cloud-based platform. Embedded classification on the board directly helps us avoid transmitting 16K samples of data for each event, and potentially pruning and sending a summarized digest at infrequent intervals instead. The energy data is for the Intel Edison platform when using WiFi to communicate events.

To understand the extent of improvement from our approach, we show a comparison of data transfer energy costs for the different platforms in Figure 10. Here, we clearly see the Particle Photon outperforming all other platforms by 2–10× when considering the energy efficiency of transfer. This is expected, as the Photon board is geared towards better connectivity. As we see later in Section 6.1, this makes an integrated platform that includes the Photon board a key piece of the final solution.

In Figure 11, we compare the cost of transferring the di-

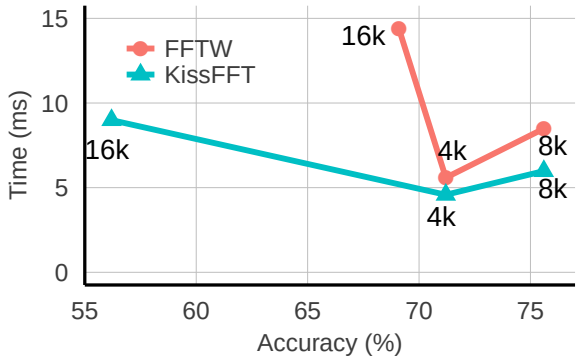

Figure 9: Exploring the tradeoffs between accuracy and runtime (on Intel Edison).

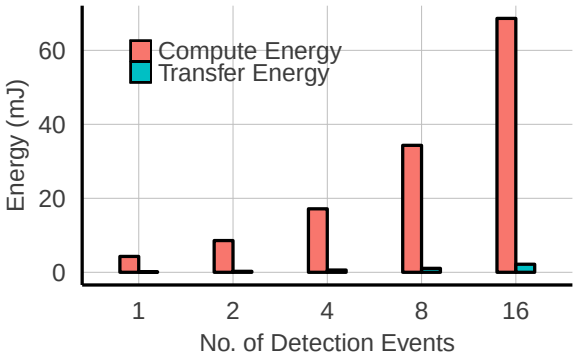

Figure 11: Exploring the tradeoffs between computation energy and communication energy (radio/WiFi on Intel Edison).

gested results as a function of number of detection events. If we transmit a detection result after every few classifications, we see that the cost of compute overwhelms the cost of data transfer in all cases. For our scenario, we can periodically send a digested `count` of the number and type of insects or mosquitoes detected rather than activate the radios after every event. Despite this apparent imbalance in energy costs, we still cannot send all input data to the cloud for post processing; the energy cost of transferring the 8K audio samples is 1000 mJ which is 20× larger than the 5 mJ compute cost.

Finally, in Figure 12, we show the energy-time tradeoffs on the two IoT boards and find a linear relationship between these quantities. This means that it is possible to accelerate the computation at the expense of choosing the more capable board and consuming more energy. A specific board exhibits a particular linear trend (slope) as we increase the sample counts. The Intel Edison has a gentler slope and sacrifices runtime for lower energy use. Specifically, in the extreme ends of the curve, a doubling in compute time only results in a 20% increase in energy. In contrast, the Raspberry Pi 3 has a steeper energy-time slope resulting in an approximate doubling of energy cost for a corresponding doubling in compute time.

6. DISCUSSION

A closely parallel effort to ours in the insect classification space is Microsoft Research's Project Premonition [10]. It seeks to detect pathogens in flying insects early before they get a chance to infect human populations. It does this by treating a mosquito as a device that can find animals and sample their blood. The project uses drones and robotic mosquito traps to capture mosquitoes from the environment, and then analyzes their body contents for pathogens. Pathogens are detected by gene sequencing the blood collected from mosquitoes and computationally searching for known and unknown pathogens in the sequenced genetic material. This approach not only helps find unknown pathogens but also study the effect of their behavior well in advance before they affect humans. Our approach described in this paper is a little less clairvoyant in nature. Our system aims at giving quick and accurate results and is mainly targeted as a predictive tool for deployment in various geographic regions to combat vector-borne diseases. It is easily possible to augment the Microsoft Premonition drone hardware with our platform to gather a richer set of wing-beat data prior to pathogen analysis in a lab.

6.1 Deployment Configurations

We tabulate the approximate bill of materials for assembling one sensor and compute block in Table 4. The sensor unit comprising an optoelectronic sensor (laser pointer + custom photo-transistor board) costs under 5$ when produced in bulk. Surprisingly, the embedded/IoT compute boards require 7–10× higher budget than the low-cost sensor. The communication-centric Photon board is 4× the cost of the sensor. This cost imbalance suggests various ways to organize a network of interconnected sensors and compute boards which we discuss here.

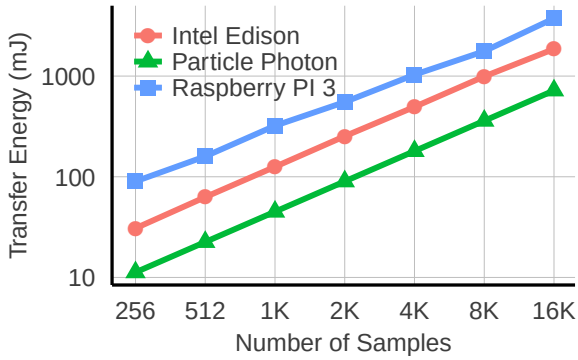

Figure 10: Quantifying the effect of data transfer size on energy of radio (WiFi transfer).

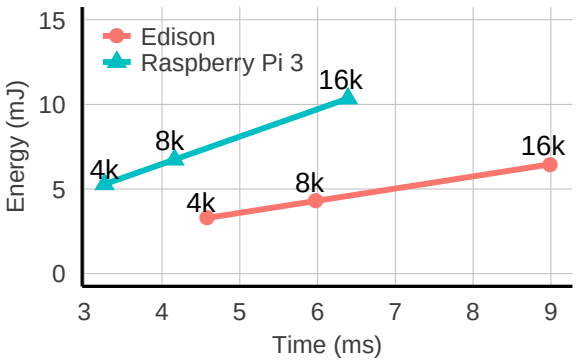

Figure 12: Overall Energy-Time tradeoffs for various audio sample counts for the two boards.

Table 4: Bill Of Materials

Item	USD
Sensor Setup	5
Raspberry Pi 3	35
Intel Edison	53
Particle Photon	19

An effective use of our embedded platform is in a networked environment with multiple sensors distributed geographically over a large area (swamp, riverside, pond, construction site). This form of multi-sensor installation would help determine the concentration and spatial distribution of mosquitoes over a vast area. Depending on the engineering constraints, we may choose between two primary system-level configurations for deploying this insect classification platform. If performance and energy and the prime concerns, an Intel Edison platform with a Particle Photon coupled for radio transfers provides a somewhat expensive but ideal solution. The Edison with SSE acceleration can first be used to compute the signal processing and machine learning computations rapidly. Then we can use the in-built standby support for powering down the board and lowering standby energy. The Photon will be used for efficient radio transfers when required and powered down when not needed. If cost is the primary concern, a single-board solution based on the Raspberry Pi 3 (with inbuilt WiFi radio + antenna) may be ideal. However, we must still consider WiFi coverage areas to determine overall system-level costs.

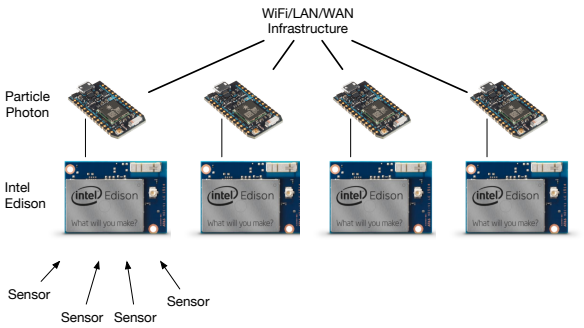

Figure 13: Energy-efficient tree network for well-connected environments.

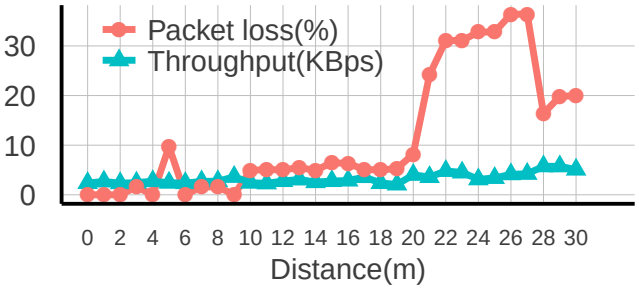

Figure 14: Photon WiFi strength from router.

Well-connected environment with energy efficiency target. In this scenario, we consider the case of well-connected environments such as construction sites or urban areas. Here, we can integrate multiple optoelectronic sensors interfaced to one Intel Edison for signal processing and machine learning tasks. We then provide radio connectivity using a dedicated Particle Photon chip for low energy transfers. While the Edison does have inbuilt WiFi antenna and radio chip, it has a significantly higher energy cost. We can then arrange this system in a tree-like network as shown in Figure 13. This would be easier to deploy, maintain, and update as it relies on infrastructure WiFi or LAN/WAN network infrastructure. For instance, Singapore plans to have island-wide connectivity for such low throughput urban sensor deployments. In this case, the high cost of the hybrid solution is worth the longer term reductions in energy requirements. Additionally, when considering Figure 14, we see that it is possible to place Photon nodes at a distance of up to 35 m with 20% packet loss. With an external antenna, it may be possible to push this to ≈100 m. As we see later in Figure 16, the in-built antenna range for ad-hoc mesh on the Raspberry Pi 3 yields a third of this range (≈14 m). Thus, from a system-level perspective, the high cost of one hybrid node amortizes out over a larger physical space.

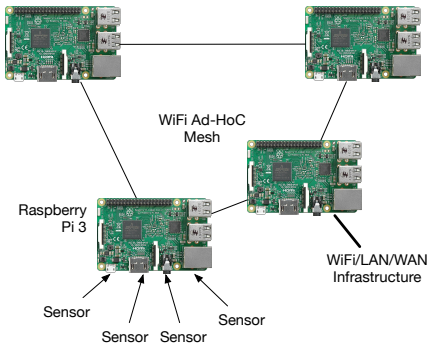

Figure 15: Low-cost ad-hoc mesh in remote areas with limited connectivity.

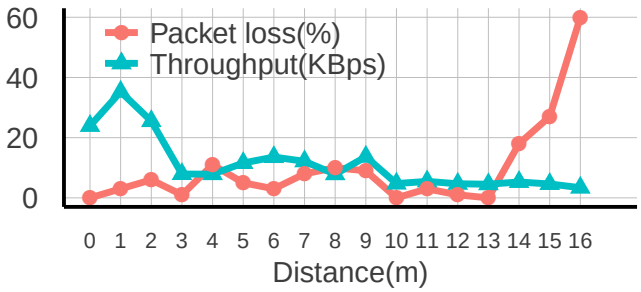

Figure 16: Raspberry Pi3 WiFi Mesh Network Signal Strength.

Poorly-connected environments with moderate budget. In geographical areas with limited network connectivity and cost concerns, it may be useful to design a simple single-platform solution based on the Raspberry Pi 3. As the performance is on-par with the Intel Edison, we expect the lower per-unit cost of the Raspberry Pi to be a key deciding factor. We can then build an ad-hoc mesh between the Pi units as shown in Figure 15 to allow collecting the resulting messages in a distributed fashion. This mesh can be interfaced with a central server or network at just one node reducing the need to rely on LAN/WAN coverage.

We implement the mesh network using `batman-adv` [22] software package, which is an implementation of the B.A.T.M.A.N. routing protocol [22]. `batman-adv` operates

entirely on ISO/OSI Layer 2, transporting the routing information using raw data frames and handling the data traffic in kernel space. Processing packets in userland is very expensive especially on low-end devices. `batman-adv` is implemented as a kernel driver which introduces only a negligible packet processing overhead even under a high load. The protocol encapsulates and forwards all traffic until it reaches the destination, hence emulating a virtual network switch of all nodes participating. Therefore all nodes appear to be link local and are unaware of the network's topology as well as unaffected by any network changes or IP to participate.

We deploy an ad-hoc WiFi mesh between Raspberry Pi 3 nodes and characterize signal strength in Figure 16. We record under 20% packet loss for distances as far apart as 14 m. Longer ranges ≈30 m are possible with USB WiFi dongles that add cost and energy beyond nominal.

7. CONCLUSIONS

We develop an embedded mosquito species detector using wing-beat frequency analysis implemented on low-power, low-cost IoT platforms such as the Intel Edison, Raspberry Pi 3 and Particle Photon platforms. We are able to deliver 80% classification accuracy while requiring 5 ms per detection even for 8 K samples of wing-beat audio data in under 5 mJ of energy (2 months of AA battery life at 2000 mAh). This allows us to develop a network of intelligent sensors that can be deployed in vulnerable areas such as construction sites, swamps, ponds and lakes, and infestation-prone urban zones. A network based on the in-built WiFi capability of the Raspberry Pi 3 IoT board can provide a communication range of around 14 m between devices while one built using Photon board provides a coverage of 35 m. The network can collectively help detect mosquito species populations trends which can be useful to authorities for deployment of preventive resources particularly in developing countries.

8. REFERENCES

[1] WHO. (2016, Feb) Vector-borne diseases. [Online]. Available: http://www.who.int/mediacentre/factsheets/fs387/en/

[2] D. S. Shepard, L. Coudeville, Y. A. Halasa, B. Zambrano, and G. H. Dayan, "Economic impact of dengue illness in the americas," *The American journal of tropical medicine and hygiene*, vol. 84, no. 2, pp. 200–207, 2011.

[3] J. A. Suaya, D. S. Shepard, J. B. Siqueira, C. T. Martelli, L. C. Lum, L. H. Tan, S. Kongsin, S. Jiamton, F. Garrido, R. Montoya *et al.*, "Cost of dengue cases in eight countries in the americas and asia: a prospective study," *The American Journal of Tropical Medicine and Hygiene*, vol. 80, no. 5, pp. 846–855, 2009.

[4] healthmap.org. (2016, Feb) Health map. [Online]. Available: http://www.healthmap.org/en/

[5] W. H. Organization. (2016, Feb) Sentinel surveillance. [Online]. Available: http://www.who.int/immunization/monitoring_surveillance/burden/vpd/surveillance_type/sentinel/en/

[6] I. National Vector Borne Disease Control Programme. (2009, Feb) Govt. of India initiatives for Dengue and Chikungunya. [Online]. Available: http://nvbdcp.gov.in/dengue-goi-activities.html

[7] K. S. Jayaraman. (2014, Oct) Dengue cases under-reported in India: study. [Online]. Available: http://www.natureasia.com/en/nindia/article/10.1038/nindia.2014.135

[8] N. E. Agency. (2016, Feb) Dengue clusters. [Online]. Available: http://www.dengue.gov.sg/subject.asp?id=74

[9] S. M. Lemon, P. F. Sparling, M. A. Hamburg, D. A. Relman, E. R. Choffnes, A. Mack *et al.*, *Vector-borne diseases: understanding the environmental, human health, and ecological connections, workshop summary (forum on microbial threats)*. National Academies Press, 2008.

[10] M. Research. (2015, Jun) Microsoft Project Premonition. [Online]. Available: http://research.microsoft.com/en-us/um/redmond/projects/projectpremonition/default.aspx

[11] G. E. Batista, Y. Hao, E. Keogh, and A. Mafra-Neto, "Towards automatic classification on flying insects using inexpensive sensors," in *Machine Learning and Applications and Workshops (ICMLA), 2011 10th International Conference on*, vol. 1. IEEE, 2011, pp. 364–369.

[12] M. C. Kahn, W. Celestin, W. Offenhauser *et al.*, "Recording of sounds produced by certain disease-carrying mosquitoes." *American Association for the Advancement of Science. Science*, pp. 335–6, 1945.

[13] Y. Chen, A. Why, G. Batista, A. Mafra-Neto, and E. Keogh, "Flying insect classification with inexpensive sensors," *Journal of insect behavior*, vol. 27, no. 5, pp. 657–677, 2014.

[14] I. Potamitis, K. Fysarakis, D. Longueville, and S. Ntalampiras, "Hardware implementation of a system classifying the optoacoustic signature of insects wing-flap."

[15] L. Devroye, L. Györfi, and G. Lugosi, *A probabilistic theory of pattern recognition*. Springer Science & Business Media, 2013, vol. 31.

[16] S. Siewert, "Using Intel streaming SIMD extensions and Intel integrated performance primitives to accelerate algorithms," *White Paper*, 2009.

[17] R. Chandra, *Parallel programming in OpenMP*. Morgan Kaufmann, 2001.

[18] V. G. Reddy, "NEON technology introduction," *ARM Corporation*, 2008.

[19] M. Borgerding, "Kiss FFT," 2009. [Online]. Available: https://sourceforge.net/projects/kissfft/

[20] C. Lomont, "Introduction to Intel advanced vector extensions," *Intel White Paper*, 2011.

[21] Y. Chen. (2015, nov) Flying insect classification with inexpensive sensors. [Online]. Available: https://sites.google.com/site/insectclassification/

[22] D. Johnson, N. Ntlatlapa, and C. Aichele, "A simple pragmatic approach to mesh routing using BATMAN," in *In 2nd IFIP International Symposium on Wireless Communications and Information Technology in Developing Countries, Pretoria, South Africa*, 2008.