Automated Precision Tuning in Activity Classification Systems: A Case Study

Nicola Fossati, Daniele Cattaneo, Michele Chiari, Stefano Cherubin, Giovanni Agosta
Politecnico di Milano, DEIB
nicola.fossati@mail.polimi.it,{daniele.cattaneo,michele.chiari,stefano.cherubin}@polimi.it,agosta@acm.org

ABSTRACT

The greater availability and reduction in production cost make wearable IoT platforms perfect candidates to continuously monitor people at risk, like elderly people. In particular these platforms, along with the use of artifical intelligence algorithms, can be exploited to detect and monitor people's activities, in particular potentially harmful situations, such as falling. However, wearable devices have limited computational power and battery life.

We optimize a situation-recognition application via the well-known *precision tuning* practice using a dedicated state-of-the-art toolchain. After the optimization we evaluate how the reduced-precision version better fits the use case of limited-resources platforms, such as wearable devices. In particular, we achieve over 500% of speedup in execution time, and consume about 6 times less energy to carry out the classification.

CCS CONCEPTS

• **Hardware** → *Power estimation and optimization*; • **Software and its engineering** → *Compilers*; • **Applied computing** → Consumer health.

KEYWORDS

Artificial Intelligence, Wearable Devices, Precision Tuning

ACM Reference Format:
Nicola Fossati, Daniele Cattaneo, Michele Chiari, Stefano Cherubin, Giovanni Agosta. 2020. Automated Precision Tuning in Activity Classification Systems: A Case Study. In *11th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures / 9th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM'20), January 21, 2020, Bologna, Italy*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3381427.3381432

1 INTRODUCTION

The ageing of the European population is a well-known trend since the last century, and it is projected to intensify, leading to over 25% of the population begin aged 65 or more by 2050. By the same date, it is expected that 10% of the population will be aged 80 or more [1]. One of the main health risks for frail elderly people and people with neurological disorders is related to falls. Technology, and particularly the Internet of Things, can play an important role for an active and healthy population ageing by supporting monitoring and rehabilitation of elderly people [2]. Monitoring of end users during daily life allows to analyse (both quantitatively and qualitatively) motor performances and to detect fall events. In rehabilitation scenarios, IoT devices can enable tele-rehabilitation and remote assessment of motor performance by enabling clinicians to remotely collect information on motor performance tests such as *Sit-to-Stand* [3] or *Timed Up and Go* [4], performed by the patient at home.

In the last decade, fall and activity detection solutions based on a variety of different technologies were proposed, leveraging sensors such as range-finders, cameras, and inertial sensors [5]. Among these, inertial sensors (accelerometers and gyroscopes) are suitable for wearable solutions, which enable monitoring in diverse daily life environments. Furthermore, inertial sensors are among the cheapest solutions, and can therefore reach a wide adoption. Several fall detection and activity classification algorithms have been developed [6; 7]. The main critical aspect for such wearable devices is the need to trade-off device bulk and cost against the battery life. The more energy is used to perform the tasks of sensing, computation and communication, the less the battery is going to last before needing to be recharged [8]. Otherwise, a larger battery will be required to reach the same battery life. In wearable devices, device size and bulk is a major factor of discomfort, which should be avoided if the device is to be worn continuously for the entire day. Thus, particularly for the monitoring scenarios, it is critical to achieve the maximum battery life.

In the design of wireless networked embedded systems, duty cycling is used to achieve longer battery life, by performing the sensing, computation and communication steps in a timed loop, and powering down the device between one iteration of the loop and the next [9–11]. Thus, the primary driver for energy efficiency is the duration of the loop iteration. To minimize communication, which is generally the most costly phase, the designer will minimize the amount of transmitted data, favoring the transmission of features rather than raw measurements. Thus, the problem arises of performing feature extraction or classification tasks on board, and minimizing the energy cost of this task.

In the case of activity classification and fall detection, the goal is to classify the motion of the subject in categories such as walking, standing, sitting, lying or falling. Fall detection generally aims at a binary classification (falling or not), whereas activity detection aims at a more precise classification. This class of applications often rely on machine learning algorithms, which are generally developed in high level languages by domain experts, and later

PARMA-DITAM'20, January 21, 2020, Bologna, Italy

ACM ISBN 978-1-4503-7545-0/20/01…$15.00
https://doi.org/10.1145/3381427.3381432

translated to C or C++ for implementation in embedded devices. These algorithms are therefore initially developed using large data types representing real numbers, usually through the IEEE 754 double precision representation ("double" in C and related programming languages) [12]. They are later manually re-written to use fixed point representations, since many ultra-low power embedded microcontrollers do not provided hardware implementations of floating point arithmetics. A large amount of domain knowledge is required in this case, since the limitations in range and precision of the small integers used in such systems force the developer to finely tune the representation, possibly readjusting it in different phases of the algorithm.

A viable alternative to manual re-writing is the use of precision tuning solutions, a type of approximate computing strategy [13] that has been gaining increasing traction both in High Performance Computing [14] and in hardware/software co-design [15].

In this paper, we demonstrate how it is possible to achieve major energy savings in the computation part of the duty cycle of an embedded system performing activity monitoring and classification, by automatically performing precision tuning on the computationally intensive kernel of a machine learning algorithm, k-nearest neighbors classification, that has been adopted for fall detection and activity classification [16]. To this end, we leverage a dedicated set of plugins developed for the LLVM compiler framework, which allow, starting from C/C++ code annotated with initial value ranges for the input variables, to automatically tune the data types and perform the necessary conversion to generate an equivalent fixed-point version of the code, while keeping error under control [17].

Through our approach, we are able to perform the classification task with a speedup over 500% with respect to the floating point implementation, when running on an STM32 board with an ARM Cortex M3 processor. The fixed-point version of the code has slightly more compute intensity, leading to a modest increase in average instant power consumption. However, this increase is small enough that the overall energy to solution also achieves a reduction of an order of magnitude. This performance and energy improvement is achieved with no loss of accuracy. As a result, we can achieve significant improvements in relation to both functional and non-functional characteristics of the activity detection system, like reactivity and battery life.

The rest of this paper is organized as follows. In Section 2 we briefly survey the main tools available for precision tuning during the code compilation stage, whereas in Section 3 we describe the hardware and software components of the fall detection system we aim to optimize, and we detail the precision tuning solution we employ. In Section 4 we provide an experimental evaluation of the system, while in Section 5 we draw some conclusions and highlight future research directions.

2 RELATED WORKS

Precision Tuning is traditionally a task related to hardware/software codesign [15]. More recently, this technique has also been explored in the high performance computing domain [18]. Several tools have been proposed in the literature to tune applications whose domain range from signal processing [19] to weather forecast [20].

GeCoS [21] is a compiler framework that specifically focus on custom hardware design. It allows building a compiler flow where the output is not only optimized for speed, but can depend on different parameters, like hardware size or power consumption. It also provides precision tuning capabilities. In particular, it allows to convert floating point operations into fixed point operations. Analyses and transformations from the *GeCoS* compiler have been adapted to the use case of fixed point exploitation in HPC [18].

FlexFloat [22] proposes an extension to the existing floating point types, that can be fully computed in hardware, to assess the effects of reduced precision data types for transprecision computing. In particular, they designed 8-bit and 16-bit floating point data types. Their goal is to demonstrate how a less complex FPU can carry out results with less circuitry and power consumption. Unfortunately these data types require a custom processor design – or a reconfigurable chip –, as such data types are currently implemented only in microprocessor prototypes.

Daisy [23] is a tool that combines two techniques, mixed precision tuning and code rewriting. The latter consists in reordering operations or exploiting arithmetic properties – like associativity or distributivity – in order to change the operations executed in the kernel without changing its semantic and precision requirements. Daisy can then output the modified code as Scala or C source code. Unfortunately, the input to the tool must be a description of the algorithm in a high level language. Therefore, this tool requires high implementation effort from the programmer.

Precimonius [24] is a tool based on the LLVM compiler toolchain which focuses on finding the best precision for each variable in a program. The precision mix found is subject to two constraints: a given error bound on the output, and the presence of a speedup in the execution time of the optimized kernel. While being a very versatile tool – as it is language agnostic – it does not support the usage of fixed point data representations.

CRAFT [25] is a framework for analyzing and instrumenting applications exploiting floating point representations. CRAFT allows the user to analyze a program – provided as a compiled binary – and determine the smallest data type required to perform a correct computation w.r.t. the precision constraints.

TAFFO [17] is a framework based on the LLVM toolchain which performs precision tuning of C and C++ programs. TAFFO focuses on the exploitation of the fixed point numeric representation, and operates on the LLVM intermediate representation. Additionally, TAFFO includes a component named Feedback Estimator which provides an estimate for the errors introduced by the conversion, and a prediction of whether a speedup will be achieved [26].

3 APPLICATION SCENARIO

The task of activity classification can be performed by systems of varying complexity and size, depending on the specific use case the system is aimed at. Complementarily, there are more than one algorithm being used for such a task, even though the majority of them is based on machine-learning principles.

We target very small computing systems, such as microcontrollers. Such systems are employed in real world scenarios for

activity detection in wearable devices and internet-of-things applications. Given this premise, in this section we analyze the scenario in which we envision our solution to be utilized, and the characteristics of the activity detection system that is the object of the improvements we propose.

3.1 Activity classification system

We consider a typical activity detection system featuring accelerometer, gyroscope, and magnetometer sensors, alongside the storage, processing and user-interface elements. The operation of such a system acquires data from sensors continuously in real time. Subsequently to the acquisition, a data aggregation step is performed, in order to extract specific features of the data. In the case of activity classification, the following features are calculated:

- Total acceleration, computed as the norm of the acceleration vector
- Sum of the absolute values of the acceleration components
- Average of the total acceleration over a moving window
- Variance of the total acceleration over a moving window
- Minimum and maximum values over z-acceleration component over a moving window

These data features are later processed by a classification algorithm to detect the current activity of the subject. In this work, we consider the *k-nearest neighbour* (KNN) [27] algorithm.

The data aggregation process can be performed either on the CPU of the wearable device, or by exploiting custom hardware logic. However, the most complex task is certainly the classification. Since the offloading of this task to remote or cloud devices may lead to large delays and/or to unacceptable service downtime, the classification typically runs on the CPU of the wearable device. In addition to the real-time-collected data, the classification algorithm also requires a pre-defined *model*. In the case of the KNN algorithm, the model consists of a collection of a data entries representing known user situations. This model data-set has to be in the device memory before running the classification.

Whenever the classification system detects a dangerous situation, the processing element initiates the emergency routine related to that situation. This routine typically involves the activation of a user interface and/or other outward-facing components – such as phone calls to an emergency service. The effectiveness of the system as a whole mainly depends on its accuracy and on its reactivity.

3.2 Improving the classification algorithm

The accuracy of the KNN classification depends on the quality of the model that the vendor installed on the device, and on the time granularity of the aggregated data to test. We aim at improving the latter factor. Indeed, a shorter classification time allows our system to fetch and to aggregate data more frequently from the sensors. Moreover, the classification time also impacts the reactivity of the system. The ability of the system to detect and to react to a threat – such as a fall event – before the user is actually harmed enables the possibility to trigger countermeasures to reduce or to nullify the effect on the user. This is the case of air-bag systems inflated before the fallen person hits the floor[28].

ALGORITHM 1: Pseudocode of the software system.

```
load dataset;
model ← scale(dataset, 0.0, 1.0);
while true do
    data_in ← read sensors;
    test ← aggregate(data_in);
    prediction ← classify(model, test);
    if prediction = danger then
        emergency();
    end
end
```

Algorithm 1 shows the typical software structure of a continuous activity detection system. Our goal is to improve the execution time of the control loop. This metric is tightly coupled with hardware performance figures, such as CPU clock rate, and memory latency. Since the amount of data to be processed after the aggregation step is fixed, the impact of memory accesses on the execution time depends on the size of the model. The CPU processing time depends on the size of the model and on the layout of the data to be processed.

Tuning the size of the model is beyond the scope of this work. We aim at reducing the execution time by acting on the layout of the data to be processed. In particular, we explore precision tuning to exploit fixed point data types instead of floating point ones.

3.3 Precision tuning

Precision tuning aims to substitute a computational kernel with an equivalent one that features a different set of data types. Reducing the data size typically allows a reduction of the time-to-solution, as it is well-known that the amount of time required for performing arithmetic computations is heavily dependent on the number of bits that they process. Some precision tuning approaches focus on replacing floating point representations with other smaller floating point representations, – e.g. double precision with single precision.

A different approach consists in the use of fixed point representations, instead of floating point. Fixed point notation, although not appropriate for data with high dynamic ranges, allows a more fine grained tuning. Additionally, it exploits the same functional units as integer representations. This last peculiarity is most relevant in embedded systems, as Floating Point Units are often missing or very inefficient on such platforms. Thus, the usage of fixed point representations allows to carry out the computation faster and using less energy. Since the activity-detection system we are considering is a kind of embedded system, such an optimization is indeed relevant for the use case at hand.

The chief side-effect of precision tuning is that the output of the tuned software will be necessarily different with respect to the original software. However, machine-learning algorithms such as KNN are inherently error-tolerant, and thus constitute prime candidates for exploiting precision tuning.

Adopting precision tuning often requires laborious and error-prone rewriting of the software. In addition to this inherent workload, additional effort is required to determine the most appropriate

level of accuracy-to-performance trade-off. For this reason, we rely on an automatic tool from the state of the art – TAFFO [17] – to guide our process.

More specifically, the process of precision tuning involves several stages. At first, we should apply an analysis of the values involved in the computation. Since we know in advance from Algorithm 1 that the classification task immediately follows the scaling, we know that test values at runtime will range within known upper and lower bounds. Such information must be given to the compiler as source code annotations. An example of annotation is shown in Listing 1. The information in such annotations is converted to LLVM-IR metadata by the CLANG compiler frontend. Later stages of the precision tuning process will be applied on the LLVM-IR representation of the KNN algorithm.

```
1 double minSMV
2 __attribute((annotate("target('minSMV') scalar(range(-25,25))")));
```

Listing 1: Example of annotation, as required by TAFFO. This annotation specifies that the variable *minSMV* shall be converted into fixed point representation, and that the numerical range it will assume at run-time will be $[-25, 25]$.

It follows an analysis stage that propagates value range information to all the intermediate values. This stage can be automatically performed by an analysis pass of the LLVM compiler infrastructure. We rely on the TAFFO toolchain – which features one of such passes – to implement this step.

From this information on the dynamic range of each value of the kernel, TAFFO is able to automatically allocate a fixed point data type and bit partitioning to each of them. The size of the fixed point data types is constrained to 32 bit. The point position is automatically determined individually for each value, such that the integer part is large enough to represent the dynamic ranges computed by the previous pass. Then, TAFFO performs the code transformation proper, by replacing floating point data types and instructions within the LLVM-IR with fixed point equivalent ones.

The output of aforementioned process is a LLVM-IR fixed point version of the kernel. It can be compiled to machine code using the same CLANG compiler used to compile the rest of the program. TAFFO also features an error estimation component that provides an upper bound estimation of the quantization error introduced by the floating point to fixed point conversion. However, this error estimation can only provide information on the distance between the values as using original data type and such as using fixed point version. Hence, it is impossible to estimate the misprediction rate a priori. In our experimental campaign we evaluate this metric experimentally.

Through the application of automatic precision tuning via TAFFO, we are able to dramatically reduce the time-to-solution and energy consumption of an embedded activity-detection system, at a much lower cost in terms of effort spent in adapting the classification algorithm, and without significantly impacting other metrics such as storage requirements and detection accuracy. Thus, our methodology allows to widen the spectrum of ultra-low-power platforms where it is advantageous to exploit machine-learning algorithms, and to improve non-functional characteristics – such as battery life – of similar existing applications.

Tool	Option set	Size [KiB]	T [ms]	$I_{\mu C}$ [mA]	$V_{\mu C}$ [V]	P [mW]	E [mJ]
TAFFO	-O3	39.67	366.5	36.25	3.23	117.1	42.9
	-Ofast	39.79	<u>366.0</u>	36.25	3.23	117.1	42.9
	-Os	39.68	366.3	36.11	3.23	116.6	<u>42.7</u>
	-Oz	33.26	384.7	35.88	3.23	115.9	44.6
LLVM	-O3	40.35	2281.3	35.54	3.23	114.8	261.9
	-Ofast	41.39	2281.2	35.83	3.23	115.7	264.0
	-Os	31.53	2330.0	35.87	3.23	115.9	270.0
	-Oz	31.13	2356.6	35.67	3.23	115.2	271.5

Table 1: Execution data relative to the temporal, spatial and energetic performance of the proposed solution on the STM32F207IG microcontroller, compared to the standard LLVM toolchain. The minimum values for the time-to-solution T and the energy-to-solution E are marked with an underline.

4 EXPERIMENTAL EVALUATION

In order to evaluate the performance and energy consumption improvements we achieve through our solution in a concrete scenario, we perform a series of experiments aimed at quantifying such measures. While the approach we employ operates by modifying the software development methodology, the results we seek are specific to a particular hardware context.

4.1 Hardware Setup

As hardware platform, we select a development board that simulates the actual physical device. More specifically, we employ the STM3220G-EVAL board[1]. This board features the STM32F207IGH6 microcontroller based on the ARM Cortex M3 CPU core, alongside 16 Mbit of SRAM and several input-output devices such as a screen, an infra-red receiver, and a MEMS device. Such a selection of peripherals is typical for a device aimed at activity detection, thus we can consider this board representative of a typical hardware configuration employed in our use case.

In order to evaluate the energy consumption relative to our solution, we employ a IDM-8351 digital multimeter to measure the power supply voltage and the current draw of the STM32F207IGH6 microcontroller. As our solution does not involve modifications to the input-output and storage elements of the device, we do not measure the power consumption of such components.

4.2 Software Setup

In order to construct a realistic classification model, and for evaluating the classification performance, we simulate the sensor data

[1] https://www.st.com/en/evaluation-tools/stm3220g-eval.html

acquisition component with an open source data-set[2], which contains about 4.5 hours of annotated activity data, and has been used in other activity detection works[16]. The raw movement data was collected from 16 male and female subjects between 23 and 50 years of age, and each data sample has been manually labeled. The labels given to the activities include Sitting, Standing, Walking, Running, Jumping, Falling, and Lying. We pre-process the aforementioned data-set, to serialize it in a format suitable for storage into persistent memory. This task is performed offline, by means of a dedicated script written in Python.

To assess the classification algorithm performance, we rely on an implementation of the KNN classifier written in C. We annotate the source code of the classifier as required by the TAFFO tool. Annotations include the KNN kernel and the scaling function. Since we assume that values in the data-set file are obtained by data aggregation, as in Algorithm 1, we know a priori the range of values that such data can assume at runtime. In our experimental campaign, we derive the value ranges of the inputs to the algorithm by automatic inspection of the values in the data-set file.

As a last step, we embed the classifier in a test program, consisting in a loop which continuously samples data from the data-set in persistent memory, and calls the classification subroutine on that data. The role of the test program is to emulate the behavior of a typical activity detection system, from the perspective of the classification algorithm. To reduce the development overhead, we relied on the Miosix[3] real-time embedded operating system. This test program is outlined in Algorithm 2.

ALGORITHM 2: Pseudocode of our system emulation.

```
for i = 1 to 100 do
    load dataset;
    <train,test> ← random(dataset);
    <train,test> ← scale(<train,test>, 0.0, 1.0);
    performance ← classify(train, test);
end
```

Additionally, the *scale* and *classify* operations have been instrumented in order to measure their execution time, and report the classification performance. We compile the test program with the LLVM 8.0.1 toolchain — more specifically the CLANG compiler — with TAFFO conversions enabled, and without employing TAFFO. Furthermore, we also exploit different optimization levels provided by the compiler. The optimization levels we use are the following:

-O3 Highest optimization level available in LLVM for general use. Performs a tradeoff between code size and speed.
-Ofast Like -O3, but prioritizing speed over code size.
-Os Like -O3, but prioritizing code size over speed.
-Oz Like -Os, prioritizing code size even further.

4.3 Analysis of the Results

As shown in Algorithm 2, through the test program we collect 100 samples related to the execution time and the classification error rate. Simultaneously, we collect energy consumption data, as described in Section 4.1. The data relative to the performance of the application is shown in Table 1. For each version, we show:

- The size of the application binary in KiB,
- The average execution time of the classification task T over 100 samples,
- The average current draw $I_{\mu C}$ and supply voltage $V_{\mu C}$ of the microcontroller during the classification task,
- The average power consumption P of the microcontroller during the classification task,
- The average energy E required to execute the entire classification task.

We observe that the application versions optimized through TAFFO present the same error rate as the un-optimized versions. The prediction error we observe is 30.49%, both for the kernel optimized by TAFFO and the floating point kernel. In particular, the misclassification events happens on the same entries for both versions. We can conclude that the kernel after the conversion to fixed-point is functionally equivalent with respect to the kernel implementation exploiting floating-point representations.

Regarding the temporal efficiency of the classification task, our approach based on automatic precision tuning presents substantial improvements. Let us define the *speedup* metric as follows:

$$\text{Speedup}[\%] = 100\left(\frac{T_{\text{un-optimized}}}{T_{\text{optimized}}} - 1\right)$$

Independently of the optimization option being exploited, the versions optimized with TAFFO consistently reach speedups of approximately 500% with respect to the corresponding LLVM version. This data is shown in Figure 1.

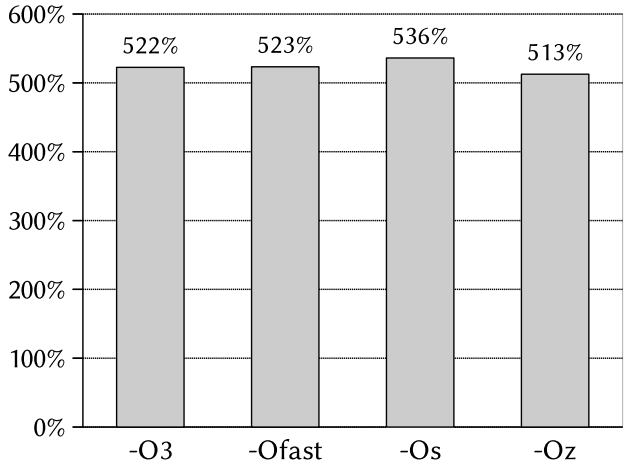

Figure 1: Speedup of the version optimized with TAFFO w.r.t. the version compiled with LLVM alone.

The reduction in the time-to-solution that we obtain is primarily determined by the fact that the original classification program heavily exploits floating point representations. In fact, the Cortex M3 CPU core we employ for our experiments does not have a

[2]https://www.dlr.de/kn/en/desktopdefault.aspx/tabid-12705/22182_read-50785/
[3]https://miosix.org

floating point unit. Thus, all floating point operations are performed by a software emulation. Therefore, in such a scenario the usage of TAFFO in the application development phase is particularly effective, as it allows to achieve the considerable speedups that derive from avoiding floating point emulation transparently.

In contrast to the reduced execution time, the average power dissipation P relative to the program optimized with TAFFO is higher than the unmodified application. However, this effect is more than offset by the reduction in execution time. As a result, the total energy-to-solution E is greatly reduced by exploiting TAFFO. We believe that the higher energy consumption of the TAFFO-optimized versions is due to the fact that they achieve higher utilization of the CPU core of the microcontroller.

5 CONCLUSIONS

We optimized a machine learning kernel by applying *precision tuning*. The analyzed use case closely simulates the behaviour of a human activities classification system. We compared different levels of code optimization (execution time, code size) by using both the standard LLVM toolchain and the TAFFO toolchain. We demonstrated that this technique improves both the execution time and the energy consumption on a wearable-like platform without increasing the misprediction rate.

We believe that this technique can achieve similar results on several other wearable applications based on machine-learning techniques. Although the fixed point KNN classifier for fall detection does not worsen the misprediction rate, applications based on machine-learning typically tolerate further approximations. Thus, we can gain better battery life on existing devices. This achievement implies improvements on the ability to run the activity detection algorithm longer and more frequently. Other possibilities push towards devices with smaller batteries, which are easier to be worn during the whole day by elderly people and can even be embedded in *smart clothes*.

Future directions involve the exploitation of custom hardware that implements energy-efficient small floating point units – i.e. units with less than 32 bits of size. These architectures are proven to be effective on microbenchmarks, yet they lack industry-wide adoption because of production cost and lack of compiler tools that can properly exploit such features.

ACKNOWLEDGMENTS

Work supported by the FET-HPC project *RECIPE*, G.A. n. 801137.

REFERENCES

[1] Emily M Grundy and Michael Murphy. Population ageing in europe. *Oxford Textbook of Geriatric Medicine*, page 11, 2017.
[2] Mirza Mansoor Baig, Shereen Afifi, Hamid GholamHosseini, and Farhaan Mirza. A systematic review of wearable sensors and IoT-based monitoring applications for older adults–a focus on ageing population and independent living. *Journal of medical systems*, 43(8):233, 2019.
[3] Richard W Bohannon. Sit-to-stand test for measuring performance of lower extremity muscles. *Perceptual and motor skills*, 80(1):163–166, 1995.
[4] Anne Shumway-Cook, Sandy Brauer, and Marjorie Woollacott. Predicting the probability for falls in community-dwelling older adults using the Timed Up & Go test. *Physical therapy*, 80(9):896–903, 2000.
[5] Maria Cornacchia, Koray Ozcan, Yu Zheng, and Senem Velipasalar. A survey on activity detection and classification using wearable sensors. *IEEE Sensors Journal*, 17(2):386–403, 2016.
[6] Yueng Santiago Delahoz and Miguel Angel Labrador. Survey on fall detection and fall prevention using wearable and external sensors. *Sensors*, 14(10):19806–19842, 2014.
[7] Stefano Abbate, Marco Avvenuti, Paolo Corsini, Janet Light, and Alessio Vecchio. Monitoring of human movements for fall detection and activities recognition in elderly care using wireless sensor network: a survey. In *Wireless Sensor Networks*, chapter 9. IntechOpen, 2010.
[8] G. Agosta, W. Fornaciari, D. Atienza, R. Canal, A. Cilardo, J. Flich, C. Hernandez Luz, M. Kulczewski, G. Massari, R. Tornero Gavila, and M. Zapater Sancho. Challenges in deeply heterogeneous high performance systems. In *2019 22nd Euromicro Conference on Digital System Design (DSD)*, pages 428–435, Aug 2019.
[9] Prabal Dutta, Mike Grimmer, Anish Arora, Steven Bibyk, and David Culler. Design of a wireless sensor network platform for detecting rare, random, and ephemeral events. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, page 70. IEEE Press, 2005.
[10] Vijay Raghunathan, Saurabh Ganeriwal, and Mani Srivastava. Emerging techniques for long lived wireless sensor networks. *IEEE Communications Magazine*, 44(4):108–114, 2006.
[11] D. Zoni, L. Cremona, and W. Fornaciari. All-digital energy-constrained controller for general-purpose accelerators and cpus. *IEEE Embedded Systems Letters*, 2019.
[12] William Kahan. IEEE standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE*, 754(94720-1776):11, 1996.
[13] Sparsh Mittal. A survey of techniques for approximate computing. *ACM Computing Surveys*, 48(4):62:1–62:33, March 2016.
[14] Michael O Lam and Jeffrey K Hollingsworth. Fine-grained floating-point precision analysis. *The International Journal of High Performance Computing Applications*, 32(2):231–245, 2016.
[15] Markus Willems, Volker Bürsgens, Thorsten Grötker, and Heinrich Meyr. FRIDGE: an interactive code generation environment for HW/SW codesign. In *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 287–290 vol.1, April 1997.
[16] Z. Liu, Y. Cao, L. Cui, J. Song, and G. Zhao. A benchmark database and baseline evaluation for fall detection based on wearable sensors for the internet of medical things platform. *IEEE Access*, 6:51286–51296, 2018.
[17] Stefano Cherubin, Daniele Cattaneo, Michele Chiari, Antonio Di Bello, and Giovanni Agosta. TAFFO: Tuning assistant for floating to fixed point optimization. *IEEE Embedded Systems Letters*, pages 1–1, 2019.
[18] Stefano Cherubin, Giovanni Agosta, Imane Lasri, Erven Rohou, and Olivier Sentieys. Implications of Reduced-Precision Computations in HPC: Performance, Energy and Error. In *Parallel Computing is Everywhere*, volume 32: Advances in Parallel Computing, pages 297 – 306, Mar 2018. International Conference on Parallel Computing (ParCo), Sep 2017.
[19] Daniel Menard, Daniel Chillet, François Charot, and Olivier Sentieys. Automatic floating-point to fixed-point conversion for dsp code generation. In *Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '02, pages 270–276, 2002.
[20] Matthew Chantry, Tobias Thornes, Tim Palmer, and Peter Düben. Scale-selective precision for weather and climate forecasting. *Monthly Weather Review*, 147(2):645–655, 2019.
[21] A. Floc'h, T. Yuki, A. El-Moussawi, A. Morvan, K. Martin, M. Naullet, M. Alle, L. L'Hours, N. Simon, S. Derrien, F. Charot, C. Wolinski, and O. Sentieys. GeCoS: A framework for prototyping custom hardware design flows. In *International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 100–105, 2013.
[22] Giuseppe Tagliavini, Stefan Mach, Davide Rossi, Andrea Marongiu, and Luca Benini. A transprecision floating-point platform for ultra-low power computing. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1051–1056, March 2018.
[23] Eva Darulova, Einar Horn, and Saksham Sharma. Sound mixed-precision optimization with rewriting. In *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, ICCPS '18, pages 208–219, 2018.
[24] Cindy Rubio-González, Cuong Nguyen, Hong Diep Nguyen, James Demmel, William Kahan, Koushik Sen, David H. Bailey, Costin Iancu, and David Hough. Precimonious: Tuning assistant for floating-point precision. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 27:1–27:12, Nov 2013.
[25] Michael O. Lam. CRAFT: Configurable Runtime Analysis for Floating-point Tuning. https://github.com/crafthpc/craft, 2018. Accessed: 2018-08-07.
[26] Daniele Cattaneo, Michele Chiari, Stefano Cherubin, and Giovanni Agosta. Feedback-driven performance and precision tuning for automatic fixed point exploitation. In *International Conference on Parallel Computing*, ParCo, Sep 2019.
[27] Thair Phyu. Survey of classification techniques in data mining. *Lecture Notes in Engineering and Computer Science*, 2174, 03 2009.
[28] Robert F Buckman, Jay A Lenker, and Donald J Kolehmainen. Air bag inflation device, March 28 2006. US Patent 7,017,195.