Plasmon-based Virus Detection on Heterogeneous Embedded Systems

Olaf Neugebauer, Pascal Libuschewski, Michael Engel,
Heinrich Müller, Peter Marwedel
TU Dortmund University
Dortmund, Germany
{firstname}.{lastname}@tu-dortmund.de

ABSTRACT

Embedded systems, e.g. in computer vision applications, are expected to provide significant amounts of computing power to process large data volumes. Many of these systems, such as used in medical diagnosis, are mobile devices and face significant challenges to provide sufficient performance while operating on a constrained energy budget.

Modern embedded MPSoC platforms use heterogeneous CPU and GPU cores providing a large number of optimization parameters. This allows to find useful trade-offs between energy consumption and performance for a given application. In this paper, we describe how the complex data processing required for PAMONO, a novel type of biosensor for the detection of biological viruses, can efficiently be implemented on a state-of-the-art heterogeneous MPSoC platform. An additional optimization dimension explored is the achieved quality of service. Reducing the virus detection accuracy enables additional optimizations not achievable by modifying hardware or software parameters alone.

Instead of relying on often inaccurate simulation models, our design space exploration employs a hardware-in-the-loop approach to evaluate the performance and energy consumption on the embedded target platform. Trade-offs between performance, energy and accuracy are controlled by a genetic algorithm running on a PC control system which deploys the evaluation tasks to a number of connected embedded boards. Using our optimization approach, we are able to achieve frame rates meeting the requirements without losing accuracy. Further, our approach is able to reduce the energy consumption by 93% with a still reasonable detection quality.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Evolutionary prototyping*; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*; J.3 [**Life and Medical Sciences**]: Biology and Genetics

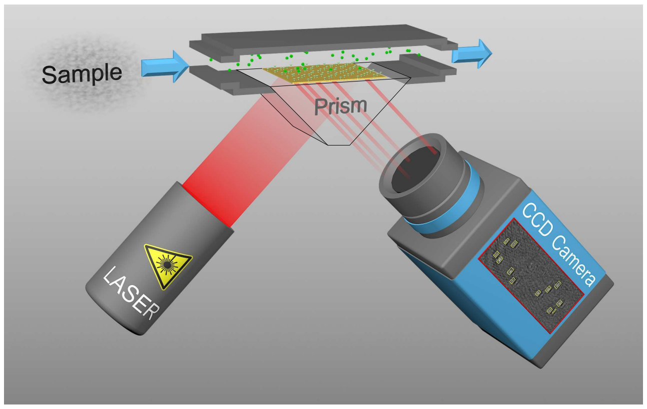


Figure 1: PAMONO biosensor. Samples are pumped through a flow cell. Viruses in the sample attach to antibodies on a gold layer, which is illuminated with a laser through a prism. The reflected light is recorded with a CCD camera. Each single virus, although only some tens of nanometers in scale, changes the amount of reflected light on the micrometer scale, which is automatically detected.

General Terms

Measurement, Algorithms, Design, Performance

Keywords

MPSoC, Parallelization, Heterogenity, Design Space Exporation, Embedded Systems, GPGPU

1. INTRODUCTION

In our globalized world, diseases can spread easily over the whole planet. To contain the spread of viruses, devices to enable fast and reliable virus detection are required. Most available systems based on computer vision approaches require complex computations and, consequentially, are large and heavy and thus bounded to specific locations like hospitals. Portable detection devices would drastically increase the versatility of such detection systems. High-performance, state-of-the-art embedded multi-core systems, combined with powerful detection algorithms, are a promising platform to enable battery-driven mobile usage of these systems. Applications from the computer vision domain, such as the virus detection discussed in this paper, are emerging more and more into the embedded systems domain. Additional examples include collision avoidance and autonomous driving systems in cars and other mobile robotics applications. All these examples expose new challenges to embedded systems in terms of computation power and energy consumption.

In recent years, the performance of embedded systems has improved significantly. Today, multiple processors, often providing different characteristics, are combined onto a single chip. These modern embedded multiprocessor systems on chip (MPSoC), such as ARM's big.LITTLE platform, are often bundled with dedicated graphic processing units (GPU) enabling the usage in the computer vision domain. MPSoCs are then still very energy efficient, e.g., they are also used in currently available mobile phones. Combining fast and powerful processors with slower and more energy efficient processors and GPU cores allows to achieve good trade-offs between performance and energy. However, the large design space offered by these heterogeneous multi-core systems poses a significant challenge to embedded systems developers in order to find good parameters to achieve the necessary performance for a given application.

In this paper we present a hardware-in-the-loop, multi-objective Design Space Exploration (DSE) approach for embedded systems, focusing on applications from the computer vision domain (cf. Section 3). Our approach, based on an existing DSE [9] method, takes execution time, energy consumption and detection quality into account. As a real world use case, the software processing pipeline of the PAMONO (Plasmon-Assisted Microscopy of Nano-Objects) sensor [18] is used. This biosensor can detect and count single viruses within less than three minutes. Due to the nature of computer vision applications, adjusting the detection quality is possible without loss of expressiveness. This enables a large optimization potential, particularly if the computation power provided by an embedded platform is constrained. In the embedded domain, optimization approaches commonly do not consider this opportunity, as discussed in Section 7. However, in the computer vision domain, varying the detection quality is quite common. Considering this trade-off enables an additional optimization dimension for our DSE approach.

The optimization objective for the PAMONO application is to find solutions with low energy consumption, sufficiently fast evaluation speed, and acceptable virus detection quality. Our extended DSE enables to analyze the optimization opportunities on a number different dimensions:

- How much energy or execution time can we save if we adapt hardware configuration parameters only?
- How much energy or execution time can we save if we optimize software parameters?
- What is the gained benefit of combining the optimization of hardware and software parameters?
- Can we save additional energy and execution time by decreasing the quality of service, i.e. detection accuracy?

The rest of this paper is structured as follows. Details of the PAMONO sensor are described in Section 2. Section 3 describes of our DSE framework, followed by an overview of the PAMONO application in Section 4. Section 5 describes the embedded heterogeneous MPSoC platform used to obtain the evaluation results presented in Section 6. Section 7 discusses related work and, finally, Section 8 concludes the paper and gives an outlook onto our future research ideas.

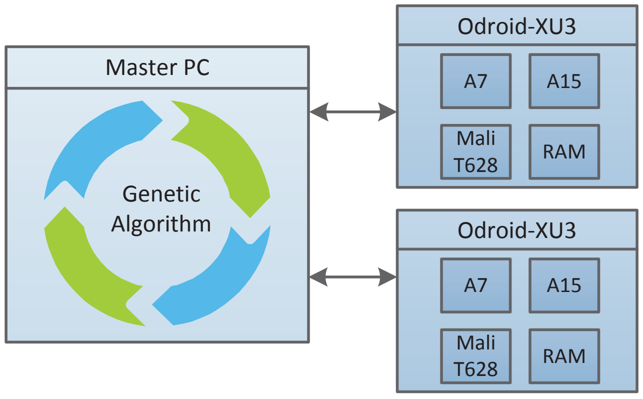


Figure 2: Distributed evaluation process. New individuals are generated on the master PC and transferred to the Odroid systems. The fitness evaluation uses the ARM Cortex-A15 cores, the Mali-T628 GPU and the RAM. The energy consumption of these components is measured by an energy measurement program, which runs on the Cortex-A7 cores.

2. PAMONO BIOSENSOR

Detecting physical structures on the nanometer scale, like viruses, is usually not possible using optical methods. Such structures have to be observed using an indirect approach. The PAMONO sensor technique allows the indirect observation of these small particles. The overall structure of the PAMONO biosensor is shown in Figure 1. The main part of the PAMONO sensor is a thin gold layer, which is coated with antibodies on top. The sample is continuously flowing through the flow cell, so that viruses in the sample can attach to the antibodies. Once a virus is attached to the antibodies it will stay on the gold layer. The gold layer is illuminated from the bottom side with laser light, which is reflected through a prism to the CCD camera. The laser light excites so called plasmon waves within the gold layer. As this excitement of the plasmon waves highly depends on the thickness of the layer, small changes in the thickness, like a virus attaching to the antibodies, strongly affect the plasmon waves and the amount of reflected light. As a result, the effects of an attaching virus can be seen on the micrometer scale as a small, faint spot appearing in the images. Only viruses that match the antibodies are attaching to the sensor, which is why only the desired virus stems are detected.

In contrast to rapid tests for virus infections like the flu or HIV, the PAMONO sensor detects and counts the viruses in the sample and does not test for antibodies in the sample. Antibodies can only be detected in the sample if the patient's immune system has already reacted to the infection. The PAMONO sensor, however, can detect the viruses as soon as they show up in the sample and therefore it closes the gap between the time the patient is becoming contagious and the patient's immune system is developing the first antibodies.

The PAMONO sensor has the abilities to be used as a fast and reliable virus sensor. But it still lacks a downsizing of

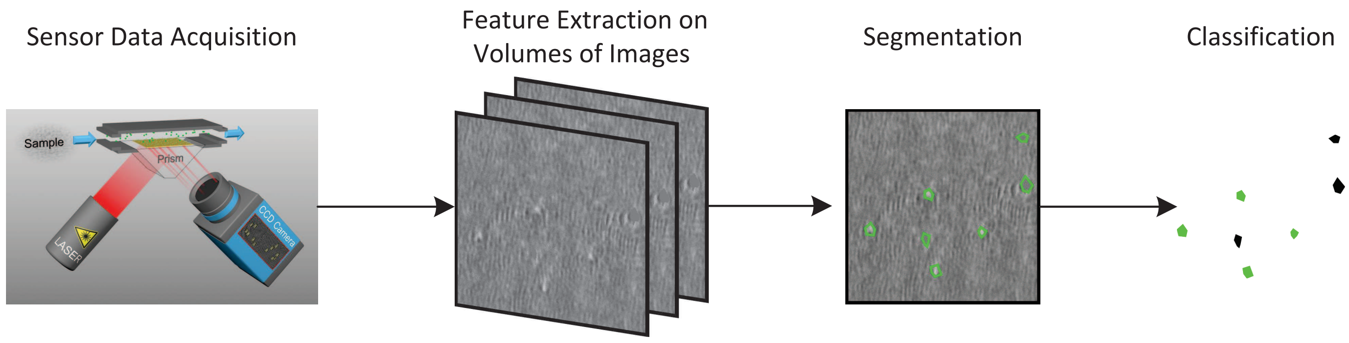


Figure 3: Simplified virus detection software workflow

the whole system to be portable. It has been shown [10, 8] that an automatic virus detection can be done on desktop and laptop systems using the graphics processing unit (cf. Section 4). If the automatic virus detection can be done on an embedded system, the whole system can be downsized to a portable device running on battery. Even the usage for area monitoring with multiple devices is then possible.

3. DESIGN SPACE EXPLORATION FRAMEWORK

Software running on an embedded system has to consider several restrictions, such as energy consumption and execution time. For embedded software developers, determining good software and hardware parametrization is often difficult, especially if the objectives are conflicting and additional constraints have to be fulfilled. In this section we describe our design space exploration (DSE) framework, which can be used to automatically identify hardware and software parametrization. Multiple and possibly conflicting objectives can be investigated. The presented framework is an extension of our previous GPU design space exploration framework [9]. It has already been shown that the framework can be used for different kinds of applications. We evaluated it for 20 different programs from three different benchmark suites, with a wide variety of industrial, physical and biological applications.

The framework is based on a heavily modified version of ECJ (Java-based Evolutionary Computation Research System) [11]. The design space exploration uses a genetic algorithm (GA) to explore the highly non-linear design space. The multi-objective evaluation is done with NSGA-II [6], which is part of ECJ.

We extended the framework to a more general approach. The previous framework is strongly coupled to a GPU simulator and a specialized fitness evaluation method. These limitations were removed. For example, the previous design required that the fitness calculation is done in a single fitness evaluation method. If new fitness values should be calculated the method had to be adjusted. The new framework can integrate fitness values from arbitrary sources. Therefore, new objectives can be examined easily. Also, the previous framework was designed to use a PC cluster for the parallel simulation of the GPUs, here arbitrary target platforms can be easily integrated.

To measure real hardware, instead of simulated GPUs, we integrated our own measurement tool, which is described in Section 6.1. We also modified the fitness evaluation in such a way that it does not influence the energy or execution time measurements on the target platform. The details of this modification are presented in Section 6.1. The used target platform, see Section 5, also made it possible to integrate the energy consumption of the GPU, CPU and RAM in our measurements.

Another peculiarity of the presented DSE framework is that it can handle a lot of different types of parameters and parameter dependencies. For example, it is common that software or hardware parameter values must lie in a particular range, like powers of two values for GPU work group sizes or frequencies that can only be changed in 100 kHz steps. These restrictions can be directly modeled within the input specification of the DSE. Complex parameter dependencies can also be modeled. One example is, if two parameters should be optimized and a third parameter needs to adapt to the first two parameters. The third parameter can therefore be set as a dependency and derived from the other parameters. A second example is, if parameters have logical dependencies, such as one parameter needs to be in a certain range only if another parameter is set. This, too, can be easily specified. All these features keep the search space small and prevent that invalid solutions are generated, resulting in a faster evaluation time.

The procedure to set up the DSE is as follows: A master PC is used to distribute and coordinate the DSE. All target platforms are attached to the master PC, e.g. via network. The sources that should be used for the fitness calculation are set as dependencies. All required data files and the program that should be evaluated can also be specified as a dependency. The command line calls and the order of the execution need to be specified. Finally, it must be declared how the different fitness values can be obtained after the programs were executed.

The evaluation works as follows: All dependent files are automatically distributed to the target platforms. This ensures that the same data and code base is used on all the target platforms. The parameters that need to be optimized and their dependencies are resolved. New parameter files are generated within the GA and are automatically distributed to the target platforms. The measurement can be automatically repeated several times to decrease inaccuracies. The (averaged) fitness results are gathered and transferred back to the master PC. Then a new evaluation is scheduled. If a generation is finished, the new generation is created on the master PC, a checkpoint and all partial results are stored and the evaluation for the next generation is executed.

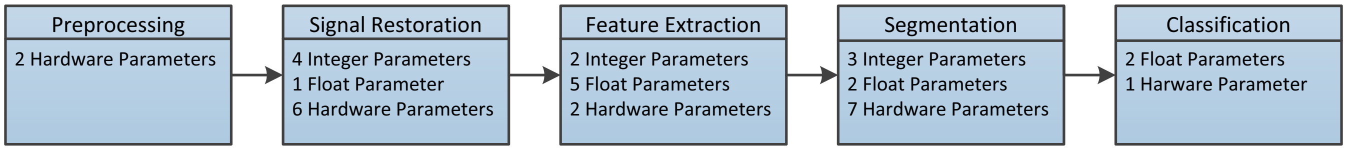


Figure 4: Pipeline for the automatic virus detection. For each pipeline step, the number of parameters to be optimized are given.

Figure 2 illustrates the distributed evaluation process used in this paper. A master PC controls the evaluation process and generates new individuals. The individuals are transferred to the Odroid target platforms, which are connected via network. The multi-objective fitness is then evaluated on the target platform. The evaluation includes the run of the program, the execution time and energy measurement and the calculation of the detection quality. For details on the Odroid target platform see Section 5.

To summarize, we extended our existing GPU design space exploration [9] in the following parts:

- The presented design space exploration measures real hardware, whereas the previous approach uses a GPU simulator.
- It is extended to optimize software parameters in conjunction with hardware parameters, whereas the previous is a hardware design space exploration only.
- An embedded system is used as target platform, whereas in the previous approach (simulated) NVIDIA desktop or mobile GPUs are used as target platforms.
- The GPU, CPU and RAM energy consumptions are measured, whereas the previous approach only measures the energy consumption of the (simulated) GPU.
- The DSE was extended to a more general approach, where arbitrary sources can easily be integrated to cooperatively calculate the desired fitness values.
- A reduction of the quality of service is used to explore solutions with a reduced detection quality but better energy consumption and/or execution time.

4. USE CASE: AUTOMATIC VIRUS DETECTION SOFTWARE

As a real world use case the automatic virus detection software for the PAMONO sensor (cf. Section 2) was chosen. This is a typical future application for embedded systems. Being a computer vision application, it has huge performance demands. In this case, several hundred images must be processed to obtain a reliable result in a short period of time. The use case has soft real time requirements, as it is desired, but not required, to process the sensor images with a frame rate of at least 25 frames per second, which is the frame rate of the used camera. By meeting this soft real time requirement, buffers between the camera and the detection software can be removed. This enables a theoretically infinite execution without the risk of buffer overflows. It has been shown (cf. Section 6) that this application is complex enough to enable the DSE to explore and solve some of the problems which are exposed through embedded systems.

The program is written in C for the CPU code and OpenCL for the GPU code. It consists of $14,000$ lines of C code and $4,000$ lines of OpenCL code (source lines of code).

A brief overview of the work flow of the virus detection is shown in Figure 3. The images taken by the PAMONO sensor are analyzed by the software. A feature extraction process, e.g. noise reduction, works on a volume of images. In the resulting images, a segmentation process identifies structures by applying polygon structures around prominent areas. Finally, a classification process determines if the polygons correspond to viruses or not.

Figure 4 shows the detailed pipeline structure for the automatic virus detection [10]. In the pre-processing step, 16-bit gray-scale images are uploaded to the GPU and converted to floating point arrays. In the signal restoration step, the physical signal model of the PAMONO sensor is used to restore the signal of the attaching virus particles and to remove the constant background signal and noise. Within the feature extraction step, different per-pixel and per-polygon features are calculated. The per-pixel features can be understood as the degree of membership of the pixel to the class of pixels corresponding to a virus adhesion, while the per-polygon features represent a degree of membership of the whole polygon to the class of a virus adhesion. In the best case, for each virus adhesion in the images, a corresponding polygon is identified. In addition, the polygon size should match the size of the virus adhesion.

The parameters for the signal restoration are mainly used for noise reduction. The feature extraction parameters are detection thresholds and parameters for switching between different feature extraction algorithms. The segmentation parameters influence how the polygons are created and how the extracted features per pixel are combined to features per polygon. Finally, the classification parameters are used to sort out false detections.

The virus detection quality is measured with the F_1 score (also F-measure), which is defined as the harmonic mean of precision p and recall r: $F_1 := 2\frac{p \cdot r}{p+r}$. The precision p is defined as $p := \frac{TP}{TP+FP}$ and the recall r is defined as $r := \frac{TP}{TP+FN}$. TP are the true positives (correctly detected viruses), FP the false positives (falsely detected viruses) and FN the false negatives (missed viruses).

The detection quality indicates how accurately the detection program counts the number of viruses in the sensor images. It should be made clear that the detection using the virus detection program is more precise than a simple virus test with just a positive or negative result. If a F_1 score of 100% is reached, this means that every individual virus appearing in the images was detected and no falsely detected particle is in the result.

In cases where a F_1 score of 100% is not necessary, new optimization opportunities arise. For example, the detection

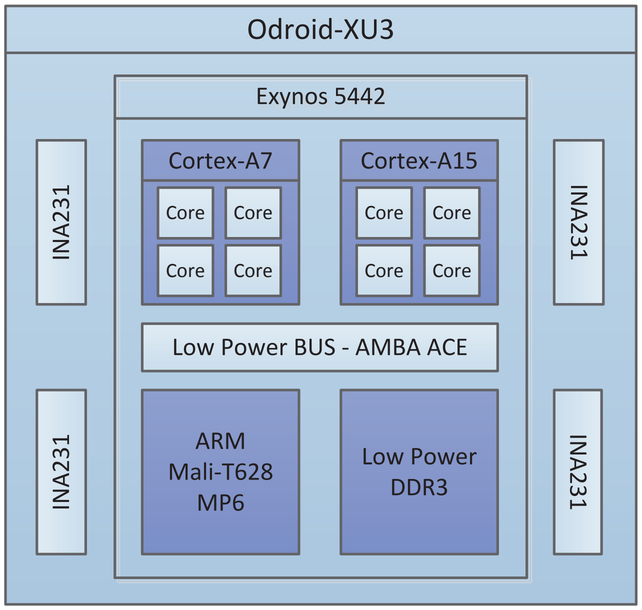


Figure 5: Key features of the Odroid-XU3. The Exynos 5442 chip consists of a quad-core Cortex-A15 CPU, a quad-core Cortex-A7 CPU, a Mali-T628 GPU and a low power RAM. The INA231 sensors are used for measuring the energy consumption of the two CPUs, the GPU and the RAM.

quality can also be used as a measurement for the Quality of Service (QoS). The experiments in Section 6.2 will use variations in the F_1 score to identify trade-offs between energy consumption and execution time.

5. TARGET PLATFORM

Modern multiprocessor systems on chip (MPSoC) using the ARM big.LITTLE [13] architecture are entering today's market. Those heterogeneous embedded systems contain processors running at different clock frequencies, featuring differing pipeline structures, cache sizes, bus systems, or memory hierarchies. By utilizing the different capabilities efficiently, energy and performance trade-offs can be found to satisfy the requirements of mobile embedded devices and their applications. Thus, today's MPSoCs with integrated dedicated powerful graphics processing units (GPU) are good candidates as target platforms for our DSE approach. The first available systems were limited in terms of task scheduling, preventing real heterogeneous multi-processing (HMP). However, current systems provide true HMP; one of the first available MPSoCs was the Exynos 5442 application processor used, e.g., on the Odroid-XU3 board [7] and Samsung's S5 mobile phone. The Odroid-XU3 board is used as the target platform in this paper.

A brief overview of the Odroid-XU3 platform is given in Figure 5. The system is composed of an Exynos 5442 application processor with four ARM Cortex-A15 and four Cortex-A7 processors. Furthermore, the Mali-T628 MP6 GPU with full OpenCL 1.1 profile support is integrated. In addition, a shared 2GB low power DDR3 memory is available. All components are connected through an energy-efficient bus. To measure the energy consumption of these key components, four INA231 [17] sensors are used to measure current and power separately for "big" cores, "little" cores, the GPU, and the DDR3 memory. The sensors are connected via an I^2C interface to the MPSoC. This system, with its powerful GPU, enables the exploration of the utilization of such heterogeneous embedded systems for mobile virus detection as described earlier.

6. EVALUATION

The mobile usage of the virus detection sensor requires a powerful yet energy-efficient computing platform. With the presented approach, we try to find good trade-offs to achieve the necessary precision with respect to energy consumption and execution time. To evaluate the abilities of our DSE approach, we analyzed three different evaluation scenarios: A design space exploration of the software parameters only, a design space exploration of the hardware configuration only and a combined hardware/software design space exploration.

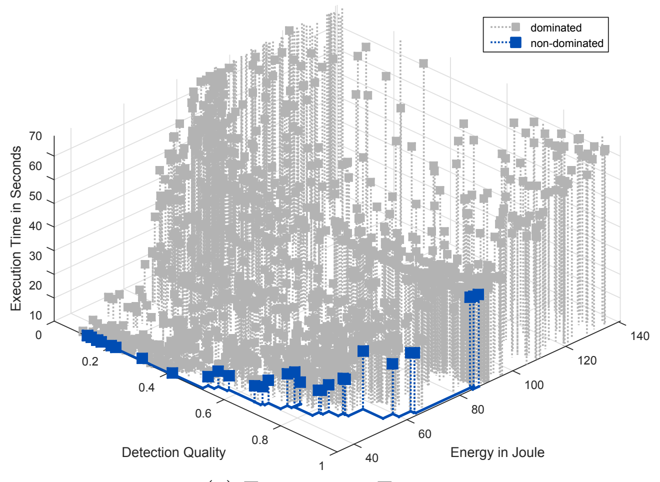
For each evaluation during the optimization, three objectives were measured, namely: virus detection quality (QoS), energy consumption and execution time. All energy and execution time measurements are repeated three times and are averaged to reduce noise in the results, which has been shown in our previous measurements to be a good compromise between precision of the measurement and evaluation time. Further, the measurements were done in a temperature controlled room and with the system's cooling fans running at full speed to reduce the influences of temperature on the energy consumption. By keeping a constant temperature during the whole evaluation, the energy consumptions of all solutions are comparable.

6.1 Measurement Setup

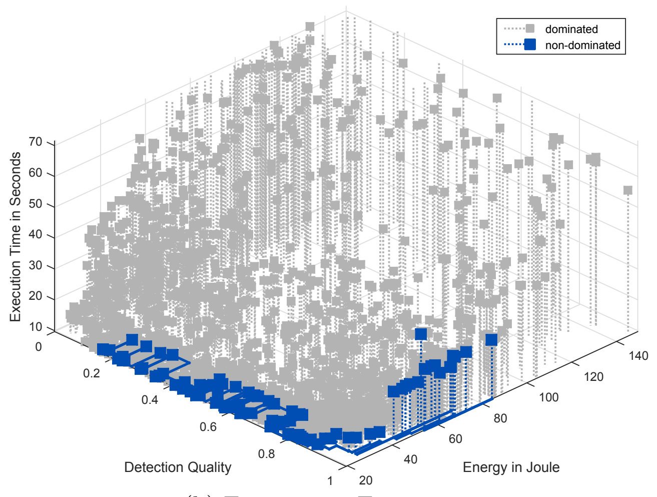
This section describes the experimental setup for the evaluation environment. The measured key characteristics are detection quality, energy consumption and execution time. An overview of the general setup is given in Figure 2. Here we will focus on the technical aspects, for a general overview see Section 3.

An Ubuntu 14.04 LTS[1] OS with a modified Linux kernel is running on the Odroid-XU3 platforms. We enabled the use of all available governors which control the frequencies of the processors. With the *performance*/*powersave* governor, the systems run at the highest/lowest possible frequency. With *userspace*, the frequency is controlled by the user respectively the algorithm. Using *interactive*, *ondemand* or *conservative*, the CPU frequency is set depending on the current workload, where *interactive* is the most aggressive strategy, *conservative* changes the frequencies gracefully and *ondemand* is in between the two other strategies. The settings of the INA231 sensor are depicted in Table 1. We chose to measure shunt and bus voltage in continuous mode, thus measurement is done continuously and not triggered by an event. The sensor samples 16 values and builds an average value for both shunt and bus. Each sample measurement takes 4.156 ms. This leads to an update period of 132.992 ms. A more precise measurement is not possible due to limitations of the I^2C interface which connects the INA231 sensors with the Exynos 5442. Thus, new values can be observed with a frequency of about 8Hz. To verify the correct INA231 configuration, we extended the provided Linux

[1]Based on kernel version 3.10.63 provided by Hardkernel



(a) Experiment Exp2$_{sw}$



(b) Experiment Exp3$_{hw\&sw}$

Figure 6: Pareto fronts and dominated points for two of the three experiments. The non-dominated points are plotted on top of the dominated points for better visualization. For execution time and energy consumption lower values are better and for detection quality higher values are better. An excerpt of the points of the Pareto fronts can also be found in Table 2.

Table 1: INA231 [17] Configuration

Number Of Averages	16
Bus Voltage Conversion Time	4.156 ms
Shunt Voltage Conversion Time	4.156 ms
Operation Mode	continuous
Update Period	132.992 ms

driver to make the configuration register of the sensor accessible through the driver.

In the following, we describe the developed energy measurement tool, which we made publicly available [12]. The tool is running on the Odroid-XU3 and directly interfaces with the drivers of the INA231 sensors (cf. Figure 5) to extract the energy values with the correct timing. The measurement can be controlled with signals (via named pipes). As the measured program takes some time to initialize, e.g. to build the OpenCL kernels and fill all the queues, the signals were used to measure exactly the main part of the program, skipping all of the pre- and post-processing. Thus, only the real execution times of the individuals are measured. Finally, the tool generates a summary with all key values which is then transferred to the master PC.

In order to not influence the measurement through its own execution or through the fitness calculation, these parts run on the Cortex-A7 cores and the energy consumption of the Cortex-A7 cores is excluded from the results. The program that should be measured runs on the Cortex-A15 cores and the Mali-T628 GPU. With this configuration, a precise energy measurement is possible.

6.2 Experiments

Three experiments were conducted, named Exp1$_{hw}$, Exp2$_{sw}$ and Exp3$_{hw\&sw}$. For the first experiment we chose to only optimize the hardware configuration and to not modify the parameters of the virus detection. For the second experiment we chose to only optimize the software parameters for the virus detection. Finally, for the third experiment, both software parameters and hardware configurations were modified in a combined hardware/software design approach.

The hardware parameters are:

- Used governor on the Odroid: *performance*, *powersave*, *userspace*, *interactive*, *ondemand* or *conservative*,
- Frequency of the Cortex-A15 core, if the governor is set to *userspace*,
- Work group sizes of the Mali-T628 GPU, and
- Memory allocation size for some buffers on the Mali-T628 GPU.

The work group sizes on the Mali-T628 GPU affect the number of threads concurrently running on the GPU. For example, if a work group size of 16 × 16 is used for noise filtering in the second pipeline step in Figure 4, the sensor images are partitioned into 16 × 16 blocks. Then, 16 × 16 threads on the GPU are used to calculate the filter output. Partial results within the work group are shared by synchronizing using the shared memory on the streaming multiprocessor on the Mali-T628 GPU.

The memory allocation size for some of the buffers on the Mali-T628 GPU can affect the detection quality. Within the different pipeline steps, ring buffers on the GPU store some of the previously processed images. Depending on the ring buffer sizes, the number of available images varies, e.g. for noise reduction or the feature extraction, which increases/decreases the quality of the results.

The software parameters for the different pipeline steps, shown in Figure 4, are:

- 6 parameters for the signal restoration, mainly parameters for the noise reduction,

- 7 parameters for the feature extraction, like detection thresholds and for switching different detection algorithms,
- 5 parameters for the segmentation, mainly thresholds, and
- 2 parameters for the classification, which affect which polygons get sorted out.

As data sets for the virus detection program we used a training and a testing data set, each consisting of 1,000 16-bit gray scale sensor images with size 706 × 167 pixels. The positions and corresponding polygons of the appearing viruses within the images are labeled. Within each evaluation, the virus detection program generates a file which contains the positions and corresponding polygons of the viruses found. These polygons can be matched to the polygons of the labeled data and the F_1 score can be calculated. The training data set is used within the optimization, while the previously unseen testing data set is used afterward for measuring the quality of the found solutions. The data sets are publicly available [16] under the Open Database License.

For the evolutionary algorithm, the parameters that should be optimized are coded as integer vector genes. All floating point parameters, e.g. the detection thresholds for the virus detection, are converted to fixed point numbers and also coded as an integer gene. For each floating point parameter, the accuracy of the fixed point number is chosen to meet but not exceed the accuracy requirements. This saves some amount of evaluation time.

The mutation rate for the NSGA-II algorithm was set to 0.1 with a likelihood of 1.0. The crossover was set to a tournament selection with a likelihood of 0.9. For each of the three evaluations, 4,000 individuals were created and measured. Each measurement was repeated three times to average out noise, resulting in 36,000 single measurements. The population size for the evolutionary algorithm was 100 and the number of generations was 40. The evaluation time for each experiment was approximately four days, with two Odroid systems. The evaluation time can easily be reduced by adding more Odroid boards. Also the number of evaluated individuals can be reduced, as 500 evaluations show already good results.

6.3 Results

In Figure 6, results for experiment Exp2_{sw} and $\text{Exp3}_{\text{hw\&sw}}$ are shown. Good solutions are indicated by either a high detection quality, a low energy consumption, or short execution times. Thus, the Pareto front is oriented to the bottom right of the diagrams. Table 2 shows excerpts of the Pareto fronts for all experiments.

For the baseline experiment $\text{Exp0}_{\text{baseline}}$ (cf. Table 2), the unmodified virus detection software was measured on the Odroid systems. The measured energy consumption was 370 Joule and the execution time 119.8 seconds. The F_1 score (detection quality) of 100% for the training data set has been previously optimized on a desktop system and was not further optimized on the Odroid. The detection quality on the testing data set attained 99.5%.

Within the first experiment Exp1_{hw} (cf. Table 2), only the hardware configuration of the Odroid system was optimized. As no parameters of the virus detection program were optimized, the detection quality is fixed to the same values as in the baseline experiment. In result the energy consumption varies from 233.5 Joule to 344.6 Joule. The execution time is 116.2 seconds to 118.9 seconds, which corresponds to a frame rate of 7.7 frames per second (fps) in the best case. The frame rate is calculated for 900 images of the data set, as 100 of the 1,000 images are used for the initialization phase, which is excluded from the measurements.

For the second experiment Exp2_{sw} (cf. Table 2 and Figure 6(a)), only the software configuration of the virus detection software is optimized. The energy consumption is 87.2 Joule for the best detection quality of 100% and 34.4 Joule for the lowest detection quality of 41.3%. The execution time varies from 29.7 seconds, which corresponds to a frame rate of 30.3 fps for the best detection quality, to 10.4 seconds with a frame rate of 86.5 fps for the lowest detection quality. As expected, the detection qualities on the testing data set are slightly lower than on the training data set, but still show good results.

The third experiment $\text{Exp3}_{\text{hw\&sw}}$ (cf. Table 2 and Figure 6(b)), where software and hardware parameters were optimized, shows similar results for the detection quality on the training and testing data set, but further improvements for the energy consumption and execution time compared to the other experiments. The energy consumption for the best detection quality could be reduced to 57.5 Joule, which saves 84% energy compared to the baseline experiment. With a reduced, but reasonable detection quality of 96.9%, the energy consumption could be reduced by 93% compared to the baseline and by more than 50% compared to the solution from Exp2_{sw} with 95.3% detection quality.

6.4 Discussion

As the results show, the genetic algorithm is able to achieve a large diversity in the results. Thus, the resulting Pareto fronts are composed of several solutions with different performance characteristics. As a consequence, the genetic algorithm is able to explore the solution space exposed by the application and target platform.

The evaluation of $\text{Exp3}_{\text{hw\&sw}}$ shows that the optimization of software and hardware parameters simultaneously achieves the best results. Solutions achieving a F_1 score of 100% on the training data and nearly 100% on the testing data at almost 31 fps were found. Only these results show that the soft real time requirements can be met to enable a live/on the fly virus detection. However, for these solutions the energy consumption is quite high. The results show that by allowing a Quality of Service (F_1) decrease of only 3.1% the energy consumption could be reduced drastically by about 52%. In addition, the amount of frames processed per second could by almost doubled. For Exp2_{sw}, similar observations can be made.

Further, the results show that to achieve additional reductions of the energy consumption the Quality of Service decrease must be quite large. If such a decrease is useful in clinical practice highly depends on the task. For example a F_1 score of 50% can still be feasible for a detection task, where only a positive or negative result is the output of the test and the actual count of individual viruses in the sample is of no interest.

The results indicate that the energy consumption is not always tightly coupled to the execution time. Exp1_{hw} shows that for this application, changing only the hardware parameters can influence the energy consumption in the opposite

Table 2: Results for the three experiments. Excerpt of the three Pareto fronts for the objectives virus detection quality (F_1 score), energy consumption and execution time. In addition the detection quality (F_1 score testing) for the unseen testing data set is shown. As baseline/comparative measurement an unoptimized run is given in the first row ($\text{Exp0}_{\text{baseline}}$), which was measured with an unmodified system and program.

Experiment	F_1 Score Training	F_1 Score Testing	Energy Cons.	Energy Sav.	Exec. Time	Speedup	Frame Rate
$\text{Exp0}_{\text{baseline}}$	100% (fixed)	99.5% (fixed)	370.0 Joule	-	119.8 s	-	7.5 fps
Exp1_{hw}	100% (fixed)	99.5% (fixed)	233.5 Joule	37%	118.9 s	1	7.6 fps
	100% (fixed)	99.5% (fixed)	239.8 Joule	35%	117.1 s	1	7.7 fps
	100% (fixed)	99.5% (fixed)	246.4 Joule	33%	116.7 s	1	7.7 fps
	100% (fixed)	99.5% (fixed)	257.7 Joule	30%	116.6 s	1	7.7 fps
	100% (fixed)	99.5% (fixed)	344.6 Joule	7%	116.2 s	1	7.7 fps
Exp2_{sw}	100%	99.5%	87.2 Joule	76%	29.7 s	4.0	30.3 fps
	98.5%	93.1%	64.6 Joule	83%	22.7 s	5.3	39.6 fps
	95.3%	88.3%	59.7 Joule	84%	20.6 s	5.8	43.7 fps
	87.0%	84.4%	50.5 Joule	86%	17.4 s	6.9	51.7 fps
	83.0%	73.4%	48.6 Joule	87%	15.8 s	7.6	57.0 fps
	75.3%	72.3%	43.9 Joule	88%	17.4 s	6.9	51.7 fps
	75.0%	69.3%	46.3 Joule	87%	14.7 s	8.1	61.2 fps
	68.4%	61.2%	39.2 Joule	89%	13.8 s	8.7	65.2 fps
	51.9%	41.3%	36.4 Joule	90%	12.0 s	10.0	75.0 fps
	41.3%	40.0%	34.4 Joule	91%	10.4 s	11.5	86.5 fps
$\text{Exp3}_{\text{hw\&sw}}$	100%	99.5%	57.5 Joule	84%	29.3 s	4.1	30.7 fps
	100%	99.5%	84.5 Joule	77%	28.9 s	4.1	31.1 fps
	98.5%	97.4%	47.9 Joule	87%	25.5 s	4.7	35.3 fps
	97.4%	99.5%	69.3 Joule	81%	23.9 s	5.0	37.7 fps
	96.9%	87.8%	27.7 Joule	93%	14.8 s	8.1	60.8 fps
	87.9%	76.6%	22.3 Joule	94%	10.8 s	11.1	83.3 fps
	84.2%	60.5%	20.7 Joule	94%	11.4 s	10.5	78.9 fps
	74.2%	63.9%	23.5 Joule	94%	10.7 s	11.2	84.1 fps
	74.2%	64.7%	33.6 Joule	91%	10.4 s	11.5	86.5 fps
	51.9%	55.8%	33.0 Joule	91%	10.0 s	12.0	90.0 fps

direction to the execution time. Another example can be seen in $\text{Exp3}_{\text{hw\&sw}}$, where a slightly faster solution (86.5fps, 33.6J) consumes 42% more energy that the slightly slower (84.1fps, 23.5J) one. However, in most cases, reducing the execution time also reduces the energy consumption.

In $\text{Exp3}_{\text{hw\&sw}}$ one can observe several peak shaped solutions on the Pareto front in the Figure 6(b), where points with similar detection quality and execution time differ in the energy consumption. Further investigations show that this behavior is caused by different governor settings. The slightly faster results use *conservative* or *performance* governors, whereas the more energy efficient solutions use the *userspace* governors with a fixed frequency of 1.1 GHz for the Cortex-A15.

Overall, the results show that the design space exploration algorithm was able to decrease the execution time of the virus detection algorithm drastically. By keeping the detection quality of 100%, a speedup of 4.1 with energy savings of 77% − 84% could be achieved. For a slightly reduced detection quality of 96.9%, a speedup of 8.1 with energy savings of 93% could be achieved. By a further reduction of the detection quality, speedups up to a factor of 12 and energy savings of up to 94% are possible. This strongly indicates that a mobile virus detection using embedded systems in combination with the PAMONO sensor, meeting the soft real time requirements, is feasible.

7. RELATED WORK

For mobile battery driven solutions, energy consumption is a key objective. As presented in this paper, modern embedded applications are emerging from the computer vision domain. Thus, GPUs play an important role. The energy efficiency of GPUs has been analyzed by Cebrian et al. [5]. This work emphasizes that the energy consumption of desktop GPUs heavy depends on the actual applications. Thus, energy-aware computing is mandatory to achieve necessary efficiency for mobile usage. A survey of energy-aware computing in general was done by Ahmad and Ranka [2].

Exploring all possible parameters is quite complex and often infeasible. Thus, (semi-) automatic Design Space Exploration (DSE) methods have been presented in the last decades. A DSE approach combining offline and online exploration was presented by Pham et al. [14]. Their approach is able to generate a mapping to heterogeneous MPSoCs considering energy and throughput. A mapping generated offline is refined during run time to increase performance. This approach only considers mapping and no hardware/software parameters as well as QoS. Agosta et al. [1] presented a hardware software co-exploration which is able to explore architectural parameters and source-level transformations concurrently. Beside hardware parameters like memory hierarchy levels this approach also considers function inlining and loop unrolling to find trade-offs between energy and delay. This approach focuses on embedded low-power applications and does not consider QoS. In addition, only two software transformations are used and thus opportunities exposed by the application e.g. algorithm parameters not considered. The MADNESS framework [4] is able to compose MPSoCs by using Evolutionary Algorithms (EAs) considering multiple objects simultaneously. Therefore, a library of predefined hardware blocks and software implementations is used. Applications are specified as Kahn Process Networks (KPN). This approach is able to explore different application scenarios. A scenario captures a specific provided implementation of the same application to take e.g. QoS into account.

In the domain of approximate computation the precision of calculation can be adjusted to achieve certain objectives. The GREEN system [3] uses controlled approximation to reduce energy consumption by meeting a certain QoS. The user provides multiple implementation of an algorithm with different precision and e.g. energy consumption. Then, the GREEN system builds a QoS model to determine the impact of the provided implementations on the QoS. Afterwards, a solution is selected which meets the user-specified QoS requirements. Thus, a trade-off between energy consumption/performance and QoS can be made. Nevertheless, the user has to provide several inputs like implementation or QoS which is complex and not always possible. The SAGE approach [15] combines automatic code generation, to generate various levels of approximation, with a runtime system to achieve speedups under user-defined output quality requirements. This approach concentrates only on GPUs as target platform and trade-offs between speedup and output quality.

8. CONCLUSION

In this paper we presented a design space exploration approach in combination with a complex biological virus detection application. In combination with the PAMONO sensor, the application is able to identify viruses in images. Usually, due to its demands on calculation power, this application is executed on PCs or laptops. To enable a mobile battery-driven virus detection, embedded systems must be used. The availability of powerful embedded multiprocessor systems on a chip allows a usage in this computer vision domain. However, if a software should run on such system, one is confronted with several difficulties, e.g., the limited computing power or tight energy consumption restrictions.

We could show that we were able to extend an existing multi-objective aware DSE approach to target embedded platforms taking the restrictions exposed by these systems into account. With three experiments we were able to show the influences of a hardware parameter optimization, a software parameter optimization and a combined hardware and software parameter optimization on the detection software.

A surprising result is that the frame rate could be increased from 7.5 fps to 30.7 fps (speedup of 4.1) without losing accuracy in the detection quality. This enables the embedded system to process the images with the best possible detection quality live from camera. At the same time, the energy consumption could also decreased by 84%, from 370 Joule down to 57.5 Joule. By taking a reduction of the Quality of Service into account, the energy consumption could be reduced by 93% with a reasonable detection quality of 96.9% and a frame rate of 60.8 fps (speedup of 8.1). These positive results have exceeded our expectations and lay the foundation for a mobile usage of this virus detection.

As future work the PAMONO sensor will be extended to detect more than one virus stem at a time. For this approach a larger sensor area will be used and the sensor will be divided into different areas, each covered with different antibodies. The new challenge with this approach is that the image sizes must adapt to the larger sensor area, which is more demanding for the target platform and the design space exploration. One opportunity for the detection software and the design space exploration is an automatic adaptation of the detection quality, energy consumption and execution time, independently to each single area of the sensor. If viruses are appearing only in one part of the sensor images, this area can be inspected with an increased detection quality. While in the other areas a low quality of service might be enough, to not miss the moment where particles start to appear and the detection quality needs to be adapted.

Acknowledgment

Part of the work on this paper has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 “Providing Information by Resource-Constrained Analysis”, project B2. URL: `http://sfb876.tu-dortmund.de`

9. REFERENCES

[1] G. Agosta, G. Palermo, and C. Silvano. Multi-objective co-exploration of source code transformations and design space architectures for low-power embedded systems. In *Proc. of SAC*, 2004.

[2] I. Ahmad and S. Ranka. *Handbook of Energy-Aware and Green Computing*. Chapman & Hall/CRC, 2012.

[3] W. Baek and T. M. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. In *Proc. of PLDI*, 2010.

[4] E. Cannella, L. Di Gregorio, L. Fiorin, et al. Towards an ESL design framework for adaptive and fault-tolerant MPSoCs: MADNESS or not? In *Proc. of ESTIMedia*, 2011.

[5] J. Cebrian, G. Guerrero, and J. Garcia. Energy efficiency analysis of GPUs. In *Proc. of IPDPSW*, 2012.

[6] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Proc. of PPSN VI*, Berlin, 2000.

[7] Hardkernel. Odroid-XU3. `http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127`, Dec. 2014.

[8] P. Libuschewski, D. Kaulbars, D. Siedhoff, F. Weichert, H. Müller, C. Wietfeld, and P. Marwedel. Multi-objective computation offloading for mobile biosensors via the LTE network. In *Proc. of Mobihealth*, Nov 2014.

[9] P. Libuschewski, P. Marwedel, D. Siedhoff, and H. Müller. Multi-objective energy-aware GPGPU design space exploration for medical or industrial applications. In *Proc. of CITIMA*, 2014.

[10] P. Libuschewski, D. Siedhoff, C. Timm, A. Gelenberg, and F. Weichert. Fuzzy-enhanced, real-time capable detection of biological viruses using a portable biosensor. In *Proc. of BIOSIGNALS*, 2013.

[11] S. Luke. *The ECJ Owner's Manual*, 2013.

[12] O. Neugebauer and P. Libuschewski. Odroid energy measurement software, 2015. `http://sfb876.tu-dortmund.de/auto?self=Software`.

[13] Peter Greenhalgh, ARM. Big.LITTLE processing with ARM Cortex-A15 & Cortex-A7. `http://www.arm.com/files/downloads/big_LITTLE_Final_Final.pdf`, 2013.

[14] N. K. Pham, A. K. Singh, A. Kumar, and K. M. M. Aung. Incorporating energy and throughput awareness in design space exploration and run-time mapping for heterogeneous mpsocs. In *Proc. of DSD*, pages 513–521, Sept 2013.

[15] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke. Scaling performance via self-tuning approximation for graphics engines. *TOCS*, 32(3), 2014.

[16] D. Siedhoff, A. Zybin, V. Shpacovitch, and P. Libuschewski. PAMONO sensor data, 2014. doi:10.15467/e9ofylrdvk.

[17] Texas Instruments Incorporated. *High- or Low-Side Measurement, Bidirectional CURRENT/POWER MONITOR with 1.8-V I^2C Interface*, Feb. 2013.

[18] A. Zybin and et al. Real-time detection of single immobilized nanoparticles by surface plasmon resonance imaging. *Plasmonics*, 5:31–35, 2010.