Int J Parallel Prog manuscript No.
(will be inserted by the editor)

Coarse-Grained Computation-Oriented Energy Modeling for Heterogeneous Parallel Embedded Systems

Adam Seewald · Ulrik Pagh Schultz · Emad Ebeid · Henrik Skov Midtiby ·

Received: date / Accepted: date

Abstract Limited energy availability is among the most challenging considerations developers face for heterogeneous systems and is critical for battery-powered devices. For complex systems composed of mechanical and computational units, such as drones and mobile robots, more than half of the power consumption can be due to the computational operations. Critically, these systems are often composed of many components, interacting concurrently to achieve specific functionality. As a result, power prediction and estimation can be a challenging task, especially if different computational units, such as CPU and GPU, should be modeled. In this paper, we focus on limited energy availability for mobile heterogeneous devices powered by a battery and present a coarse-grained computation-oriented energy modeling approach. Our approach predicts the energy consumption of a set of software components, in a specific configuration, executed according to a given scheduling policy. The model, determined numerically from several empirical power samples, describes the energy consumed by a software configuration and can be used for energy-aware planning and optimization from a computational point of view. It can potentially support a complex embedded system in maximizing the level of autonomy while minimizing power consumption and preserving the most appropriate amount of battery charge by finding the right rate of quality of service. Our approach is supported and validated by the design and implementation of a profiling tool. The tool abstracts computational energy behavior and describes the current battery drain as a function of all the admissible configurations.

Keywords Energy Profiling · Energy Modeling · Embedded Platforms · Heterogeneous Computing

This work is supported and partly funded by the European Union's Horizon2020 research and innovation program under grant agreement No. 779882 (TeamPlay).

A. Seewald, U. P. Schultz, E. Ebeid, H. S. Midtiby
SDU UAS Center, University of Southern Denmark
Odense, Denmark
E-mail: ads@mmmi.sdu.dk

1 Introduction

Numerous new challenges have been introduced in embedded systems through evolution from small and predictable to large and complex architectures featuring different computational units. Limited energy availability, security considerations, and time requirements are among the most common challenges and significantly impact the level of autonomy of modern heterogeneous systems. These systems use many-core architectures, where a CPU executes along a general-purpose GPU or GPGPU with energy-efficient small cores. A parallel CPU-GPU execution is a core aspect to achieve the best possible energy efficiency and speed-up [34].[1] In this paper, we focus on limited energy availability for mobile heterogeneous devices powered by a battery and present a coarse-grained computation-oriented energy modeling approach. The model *describes energy usage as a function of component configuration* and is based on physical measurements of power consumption of the on-board computing elements. The term *computing* is used here to denote the energy consumed by the software components of a complex embedded system. Estimates indicate that more than 50% of power consumption of complex devices featuring mechanical and computational units can occur due to computational operations, with the motion accounting for the remaining [20, 21]. Energy efficiency concerns are not limited to battery-powered devices, but also to the high-end systems [22], and thus it is becoming more important for computer architects as power consumption is growing with the introduction of new processors.

Complex software applications are often composed of many components interacting concurrently to achieve a specific functionality. Our approach aims to predict the energy consumption of a provided set of software components, in a specific configuration, executed according to a given scheduling policy. Statistical methods are used to derive a predictive model allowing prediction of the components' total power consumption as scheduling and configuration parameters are varied. High-level modeling of using a battery as energy-source is used to model the energy that would be drained from the system during computation. The energy model is segmented into two layers: the measurement layer, which describes energy consumption as a function of time, and the predictive layer, which describes energy consumption as a function of component configuration and scheduling. The model generation process starts from the developer, who specifies the configuration space within which software components can run on a given system. A profiling process collects power samples for every component configuration (measurement layer). A model that maps configuration parameters to an energy metric, such as overall energy or average power, is generated from the data, showing the energy evolution for every possible configuration (predictive layer).

This paper is structured as follows. First, the remaining of this section introduces a motivating case study and outlines our contribution. Then, Section 2 provides an overview of relevant background information, including related work and relations to our previous work. After this, Section 3 gives an overall picture of our approach towards energy modeling, Section 4 presents the measurement layer (power as a function of time), and Section 5 presents the predictive model (power as a function of configuration and scheduling). The experimental setup that we use is described in Section 6 including information on the hardware platforms under study. The experimental results are reported in Section 7, and finally, Section 8 concludes and presents future work.

[1] This is implied by Amdahl's law [2], and can be seen as its extension. For a program, the maximum achievable speed-up is limited by its sequential execution. The law, more recently reevaluated by Gustafson [11], is still relevant to assess the achievable speedup for heterogeneous parallel systems [18].

Case study: drones

We use software for drones as a case study for energy efficiency. Drones (known also as aerial robots, Unmanned Aerial Vehicles, or UAVs) are often composed of two computational units. A controller that manages real-time control of the physical aspects of the drone, and a companion computer that handles heavy computations. The latter is often a heterogeneous parallel system. A coarse-grained model of these systems can be used for energy-aware planning and optimization from a computational point of view. Moreover, energy constraints are of particular interest for drones, since their level of autonomy is directly proportional to the power-to-compute [8]. Unlike their grounded counterparts, drones must trade-off size, weight, and power, resulting in strict limits to their energy source that may not accommodate their computational needs. Operating from a limited energy source, typically a lithium-ion battery, can significantly reduce the amount of autonomy that the drone can carry on. One of the reasons why this happens is the high energy demand required by complex parallel algorithms used in several use cases, from computer vision to data processing. To this extent, many tasks in aerial robotics are still not fully autonomous. Advanced operations that require a significant level of autonomy, such as path planning, obstacle avoidance, environment mapping, and object detection, often involve human interaction. Thus there is still little difference between flying robots and their manned counterparts, except that the pilot operates from a ground station rather than onboard [32].

As a concrete example, we present the drone use case that recognize an object while performing some computationally heavy operations. The use case is composed of two components, object detection algorithm (`darknet-gpu`) that recognize a pattern using a neural network, and matrix exponentiation (`matrix-gpu`) that simulates heavy operations with varying scheduling patterns using sleep of different durations between consecutive operations. Both the components execute mostly on GPU. In a configuration file, the developer specifies these parameters per component: **a)** for `darknet-gpu` a frame-per-seconds rate $[5.8, 32]$ from the lowest acceptable, to the highest achievable, and **b)** for `matrix-gpu` two sets of rates, the matrix size $[256, 4096]$ from smallest to largest, and sleep interval $[0, 10]$ from no sleep to 10 seconds. A profiling process generates models that map time to power, one per combination in the configuration file. Based on this a high-level model is generated which combines component parameters to energy consumption. The energy consumption is estimated for any parameter combinations not profiled. This information is useful to define a proper trade-off between parameters and decide eventually, what configuration has to be used to optimize energy consumption. The information about the battery state, included in the analysis, enables the developer to select a specific runtime configuration (for instance, one that will allow completing the mission without recharging the battery by just adjusting the combinations of components and their parameters).

In our experiments, the resulting model shows an almost linear behavior for the `darknet-gpu` component, highlighting the energy behavior for any configuration in the interval $[5.8, 32]$. The `matrix-gpu` model shows the energy evolution with different scheduling options and addresses the impact of different scheduling to the battery depletion. Results for both components are presented and further discussed in Section 7.

Contribution

With this work we have: **a)** designed a computation-oriented energy modeling approach for heterogeneous platforms, **b)** evaluated the outcomes of the model on several benchmarks and devices, **c)** assessed the model's validity against external measurements and a fine-grained

approach, and **d)** developed a profiling tool for computation-oriented energy modeling. The tool, named `powprofiler`, is written in C++ and distributed under MIT license. [2] The tool was used to perform all of the experiments reported in this paper and is designed to be used, among others, in mobile robotics scenarios. It can profile a set of components being executed in a number of possible configurations and from this, build an energy model. The model allows the system under analysis to define an appropriate tradeoff between consumed energy and computations performed. This information can be used to increase energy efficiency, often linked to the level of autonomy in many modern scenarios, and meet the power requirements. For example, a drone can dynamically adjust autonomous detection capabilities to fit the current battery state of charge (referred to in this paper as SoC). Similarly, an autonomous vehicle can increase the amount of computation only when the collision detection system is required, and a mobile robot can automatically schedule tasks to compute in an energy-aware fashion. In these examples, awareness of precise computational energy cost can potentially lead to significant energy savings. Key to the approach is flexibility in terms of easy support for different heterogeneous platforms and extensibility to other classes of systems outside the mobile robotics domain.

2 Related Work

Marowka developed a power-metrics based analytical model, to exploit the possibility of considerably increasing energy efficiency by choosing an optimal chip configuration, which we extended to evaluate the impact of different architectural design choices on energy efficiency [19]. Marowka's theoretical contribution shows three processing schemes for heterogeneous computing with a comparison of their respective energy efficiency. Variations in chip configurations are done to investigate the impact on energy, power, and performance. Symmetric processor scheme consists of only a multicore CPU. Asymmetric CPU-GPU processor scheme consists of both CPU and GPU on the hardware side, and of a program running on CPU or GPU, but not on both in the same time interval, on the software side. CPU-GPU simultaneous processing scheme consists of a program running on CPU and GPU simultaneously. Our work extends and starts from Marowka's approach by building an experimental method to the CPU-GPU simultaneous processing scheme.

For evaluating the effects of a battery as an energy source, we used the work done by Rao et al. [25]. Their work summarizes state-of-the-art battery modeling into four classes of models that capture the battery state and its non-linearities. The lowest class contains the physical models that are accurate and model battery state evolution through a set of ordinary and partial differential equations. However, they suffer from a significant level of complexity that reflects on the time needed to produce predictions. The work proceeds by showing empirical models, that predict battery state from empirical trials. The third class consists of abstract models that we incorporated into our approach, in particular, by deriving the equation from the model developed by Hasan et al. [12] (they model battery state through an equivalent electrical circuit and its evolution in time). The fourth class consists of mixed models where experimental data are collected and subsequently refined with analytical expressions to determine the models' parameters.

System-level optimization techniques, such as dynamic voltage scaling, have been used for lowering the power consumption [13, 17, 6]. These techniques are available for hardware featuring dynamic voltage scaling and aim to achieve higher energy efficiency by including

[2] `https://bitbucket.org/adamseew/powprofiler`

information about configuration parameters into the scheduler. They however focus on homogeneous systems, unlike our work which is designed to work for heterogeneous systems. This approach to modeling has nevertheless been extended to include GPU features [14], to heterogeneous systems [3], to optimal software partitioning [10], and by Wu et al. to machine learning techniques [35]. The work by Wu et al. mostly relies on neural networks and has been introduced only recently in the field of power estimation and modeling for heterogeneous systems. In particular, for a collection of applications, Wu et al. train a neural network by measuring a number of performance counters for different configurations. Even if these techniques perform well for defining static optimization strategies, they are generally not suitable for heterogeneous parallel systems in aerial robotics. In these cases, systems suffer from a considerable level of uncertainty, for which reason a statically defined energy model often would not model the real energy behavior. An overview of energy estimation in the context of machine learning approaches has recently been presented by Garcia et al. [9]. They present a literature review motivated by a belief that the machine learning community is unfamiliar with energy models, but do not relate to GPU-featured devices nor in general heterogeneous devices. In contrast, in our work we aim at automating the generation of application-level power estimation models that can be adapted for machine learning algorithms. Our approach does not yet take detailed scheduling decisions into account, unlike the black-box approach for CPU-GPU energy-aware scheduling presented by Barik et al. [4]: they model the power by relating execution time to power consumption but otherwise do not focus on energy models.

Our approach shares the same principle of differentiating the microcontroller from the companion computer with [20, 21]. The controller acts on the actuators and reads the sensors, while the companion computer (can be found with different names in literature, such as secondary or embedded computer), performs computationally heavy operations. A similar approach for mobile robots is presented by Dressler et al. [7]. However, both contributions neither elaborate further on computational elements of a heterogeneous platform, such as GPU, nor focus on different robots except the one under analysis.

The majority of other contributions in the literature focus on optimizing motion planning to increase power efficiency. For instance, approaches to minimize UAV power consumption, such as the work by Kreciglowa et al. [16], aim to determine the best trajectory generation method for an aerial vehicle to travel from one configuration to another. Uragun suggests the use of power-efficient components [33]: an energy-efficient UAV system can either be built using conceptual product development with emerging technologies or using energy-efficient components. Kanellakis et al. affirm that integrating visual sensors in the UAV ecosystem still lacks solid experimental evaluation [15]. They suggest that to save energy, the available payload for sensing and computing has to be restricted. Our approach towards energy modeling shares a similar principle as the one presented by Sadrpour et al. [30, 29] for Unmanned Ground Vehicles or UGVs. They propose a linear regression-based technique in the absence of real measurements and a Bayesian networks-based one in their presence. We used a simplified approximation technique to limit the number of computations needed while focusing rather on an accurate battery prediction.

To validate our approach and quantify its outcomes, we used the models previously developed for fine-grained energy modeling by Nunez et al. [24], and Nikov et al. [23] respectively. In summary, fine-grained energy modeling uses hardware event registers to capture the CPU state under a representative workload. The energy-modeling consists of three stages. In data collection, the first stage, a benchmark runs on the platform and data are collected. The second and third stage, data processing and model generation, are performed offline on a dif-

ferent architecture. In these two stages, data are analyzed and a model that predicts possible future usage is generated. The comparison is further developed in Section 7.2.

Calore et al. develop an approach for measuring power efficiency for High-Performance Computing or HPC systems [5]. An external board is used to measure the power consumption, while the data are collected from NVIDIA Jetson TK1 board running one benchmark. Our initial analysis presented at HLPGPU 2019 was made using a similar technique [31]. A shunt resistor and digital multimeter integrated into the external board was used to evaluate the power efficiency. In this paper we extend our experiments to use internal power monitors and address a broader range of platforms. We now build a proper energy model that reflects the computational behavior of the device under study and shows the energy evolution. An early report on our work has been presented in the TeamPlay project's deliverable D4.3 [1] "Report on Energy, Timing and Security Modeling of Complex Architectures".

3 Computation-Oriented Energy Modeling

In this paper, we aim to provide a tool for automatic generation of hardware-specific energy models of a given application. The energy model is generated through a profiling process which collects several power samples and builds an energy abstraction upon them. Computational energy is addressed by mapping a component configuration to an energy metric, such as average power or overall energy. Our approach includes several on-board computing elements, such as CPU and GPU, and analyzes their role in the tradeoff related power decisions.

Figure 1 shows an overview of our approach. For a specific application on the system under study, the developer describes how components behave in a configuration file. For each component, any parameter range or value can be specified. By varying the parameters and scheduling policy, the tool generates n combinations of components called configurations $\mathcal{D} = \{d_1, d_2, \cdots, d_n\}$. Once the set $\mathcal{D}$ is generated, the profiling process starts by collecting a set of samples and by generating a layer 1 model for every configuration d_k (with $1 \leq k \leq n$). The layer 1 model maps a time interval t to a power measure $f(t)$ and estimates the power-to-compute value for a given component. Later, the model is integrated with battery state evolution and quantified in the amount of SoC left in the battery. Finally, both $f(t)$ and SoC are used to generate a layer 2 model, that maps a set of configurations $\mathcal{D}$ to the energy metric (i.e., average power consumption or overall energy usage) and SoC $g(\mathcal{D})$. This information can be used to select the best configuration and define a better scheduling policy to optimize the average computational energy.

The modeling approach is tested across four different platforms, ODROID XU3, NVIDIA TK1, TX2, and Nano (see Section 6 for details), and a selection of benchmark components is used. Each component has a set of configuration parameters varied inside defined boundaries, that represent the acceptable rate of quality of service or QoS. We use the following benchmark components: a random matrix of a predefined size that is multiplied by another or exponentiated by a given exponent on CPU (`matrix-cpu`) or GPU (`matrix-gpu`). A computer vision component built upon the darknet [27, 26] implementation of the YOLO library [28], that uses a deep neural network to detect an object from a video stream on CPU (`darknet-cpu`) or GPU (`darknet-gpu`), and NVIDIA custom-made benchmarks, modified so they can be iterated several times over a time interval, that perform matrix multiplication (`nvidia-matrix`) and quicksort (`nvidia-quicks`) of a given size on GPU.

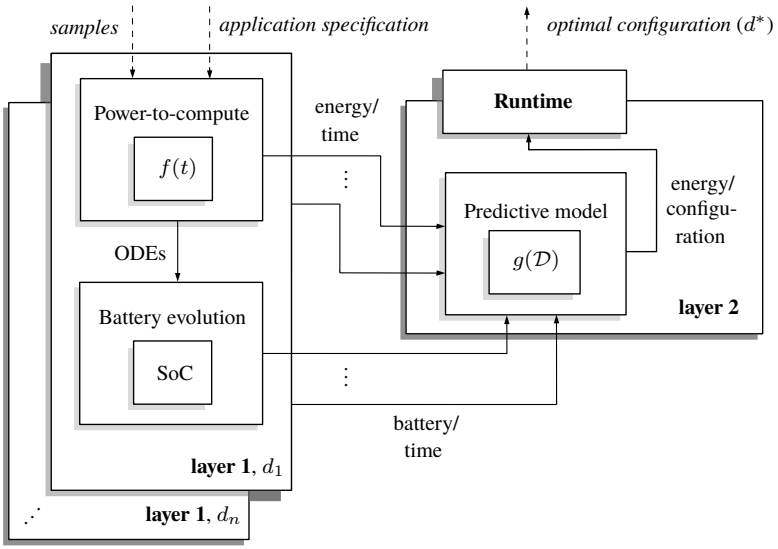


Fig. 1: Overview of coarse-grained energy modeling for parallel embedded systems.

4 Measurement Layer

The measurement layer describes an energy sample for a given configuration and scheduling of a component executed on specific hardware as a function of time. It provides average power or overall energy over a time interval for every computational unit that allows power measurement.

4.1 Overview

The developer provides for every application a number of components in the configuration file along with their parameters. An application in this layer is structured as several configured components that execute in parallel according to a specific scheduling policy. First, immediate power consumption is measured. This information is used to model the power into the power-to-compute and battery drain. An *application specification*, described later in this section, is used to define all the possible configurations of an application in the configuration file. The power is measured throughout a time interval set by the developer. Multiple metrics can be captured at this level and sampled at a fixed rate specified in the application specification, including CPU, GPU, and total power. Other metrics, as the system's load and sensors' evolution, can be added by extending the approach. Based on the information obtained at this layer, the energy that the computation would draw from a battery can be estimated using the *battery model*, also described later in this section. As a concrete example, Figure 2 shows the total energy usage as a function of time for CPU and GPU of the NVIDIA TX2 board executing the `darknet-gpu` component in four different QoS configurations.

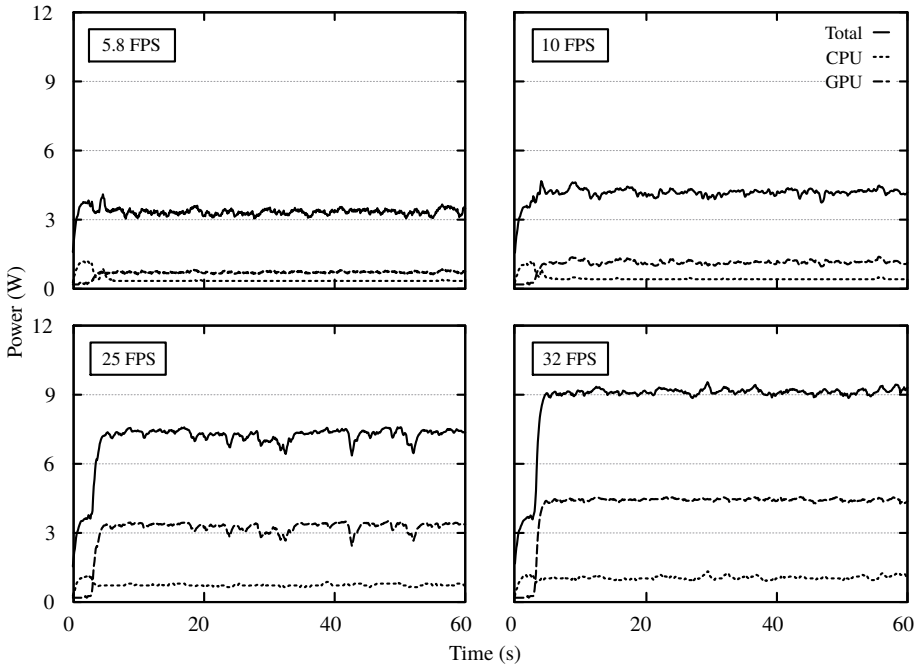


Fig. 2: Four different layer 1 models of the `darknet-gpu` component executing object detection on a video stream. The first model on top left shows the power consumption over time while the frames-per-second rate is constrained to 5.8. Rate is constrained then to 10 on right, to 25 on the bottom left and to 32 on the bottom right. With regards to the desired QoS, the frames-per-second rate can be adjusted and a specific energy target can thus be met.

4.2 Application Specification

The application specification is used to define all the computations to run for a specific energy model. It depends on the schedule that is used when performing measurements, in the sense that scheduling policies will be specific to the kind of scheduling supported by this scheduler. Currently, only a basic application specification is supported that corresponds to the capabilities of the `powprofiler` tool described later in Section 6.3. Support for more advanced models is considered future work. In particular, a dynamic measurement-oriented scheduler should be adopted in the future to define the optimal scheduling policy during computation.

The basic application specification is defined as follows. The choice between component implementations (if needed) is implicitly supported by requiring the developer to select the specific component implementation to use for the measurement. Components are configured by providing concrete, fixed values for all of their configuration parameters. A scheduling policy is implicitly supported by requiring each component to provide parameters that explicitly control fixed scheduling patterns in time (e.g., what rate to execute, when to execute). Components are assumed to implicitly control scheduling in space, for example by having different implementations for CPU and GPU (and hence being explicitly selected as part of the application configuration).

4.3 Battery Model

The battery model is a mathematical abstraction that models draining energy from a battery used to power the computation being measured. It was derived from the state-space problem representation of the empirical battery model through an equivalent electrical circuit, presented by Hasan et al. [12]. The battery evolution model can be represented in this way using an ordinary differential equation that is a function of the power drained from the system, and represents the SoC of the battery at a given instant (Equations 1 and 2):

$$\frac{d}{dt}\mathrm{SoC}(t) = -\frac{I_{\mathrm{int}}(t)}{Q_c}, \tag{1}$$

$$I_{\mathrm{int}}(t) = \frac{U_{\mathrm{int}} - \sqrt{U_{\mathrm{int}}^2 - 4 \cdot R_{\mathrm{int}} \cdot U_{\mathrm{sta}} \cdot I_{\mathrm{load}}(t)}}{2 \cdot R_{\mathrm{int}}}, \tag{2}$$

where Q_c is the constant nominal capacity, U_{int} is the internal battery voltage, I_{int} is the current load that depends on the power requirements, R_{int} is the internal resistance of the battery, U_{ext} is the external battery voltage, U_{sta} is the stabilized voltage (a fixed value determined by the load of the system), and I_{load} is the current required by the load. The constants are chosen to describe a small drone battery. The external battery voltage U_{ext} can be also expressed as follows:

$$U_{\mathrm{ext}}(t) = U_{\mathrm{int}} - R_{\mathrm{int}} \cdot I_{\mathrm{int}}(t) \tag{3}$$

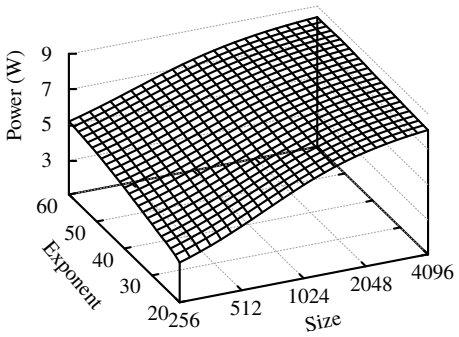
The approach allows modeling the computational system not only in terms of the overall power consumption but also in terms of the actual amount of power drained from the battery. This is especially important with a mobile robot dependent on a limited and time-dependent energy supply, running computationally heavy parallel algorithms. Critically, although this mathematical model is a simple abstraction, it models how a stable power consumption over time can induce less drain from the battery compared to using the same amount of energy distributed in a number of spikes. The battery model allows this information to be determined for specific usage scenarios and thus provides a useful way to determine what configuration corresponds to the best power usage combination for a mobile use case.

5 Predictive Model

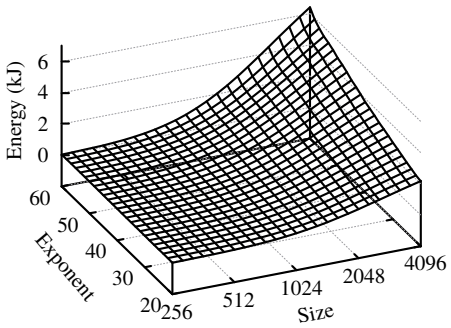
The predictive model is expressed in terms of the measurement layer and describes “average power over time frame” and other similar coarse-grained metrics as a function of component configuration parameters and scheduling policies.

5.1 Overview

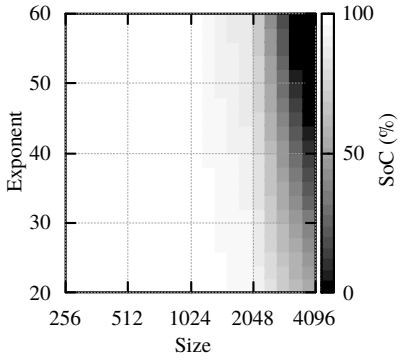
The predictive model concerns the average power over a time frame as a function of varying application models (i.e., configuration parameters, scheduling policy, . . .). The average power can be any of the outputs of the measuring layer, such as average power used by the embedded board, or average power drained from a battery. First-layer measurements are used to generate functions that map application models to energy metrics. Any aspect of the application model can be varied, as defined by the *sampling strategy* described later in this section. Varying selected aspects of the application model corresponds to changes



(a) Average power



(b) Overall energy



(c) Remaining battery capacity

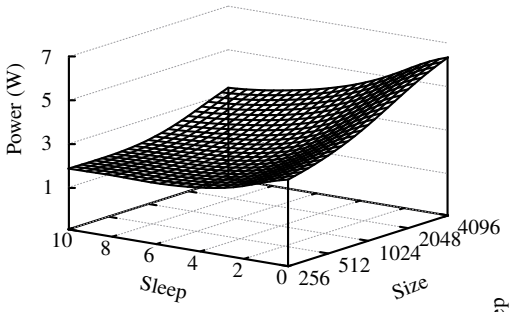
Fig. 3: Execution of GPU exponentiation, average total power 3a, overall energy 3b consumption, and battery depletion 3c as a function of size and exponent parameters. The plot shows how the choice between specific configurations impacts the energy performance of the system under analysis.

in component configuration parameters and scheduling policies, thus making it possible to determine the average power as a function of specific variations of selected component configuration parameters and aspects of the scheduling policy. Due to the potentially large configuration space, it is critical that a model covering all relevant parameter/scheduling variations can be generated from a subset of samples. This is handled by the *approximation method*, also described later in this section.

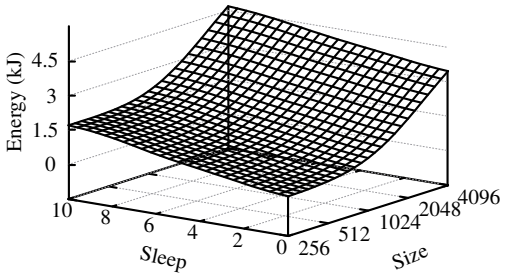
As a concrete example, Figure 3 shows the total average energy usage of the TX2 board executing the `matrix-gpu` component as a function of the size and exponent parameters. Results are discussed in further detail in the context of experimental results, see Section 7.

5.2 Sampling Strategy

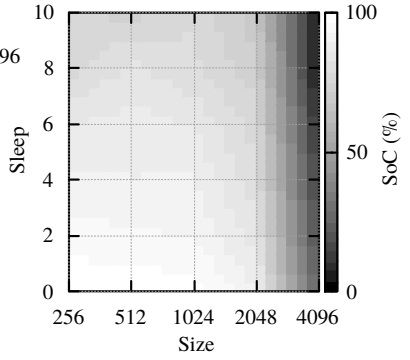
The sampling strategy controls how selected component configuration parameters are varied between measurements performed using the measurement layer. Different sampling models can be supported, for example, linear variation or random exploration. Currently only linear



(a) Average power



(b) Overall energy



(c) Remaining battery capacity

Fig. 4: Execution of GPU matrix exponentiation, average total power 4a, overall energy consumption 4b and battery depletion 4c as a function of the matrix size and simulated scheduling in the form of sleep between iterations.

and exponential variation are supported, allowing parameters to vary linearly (and exponentially) for any number of components, exhaustively sampling combinations of parameter values. The programmer specifies a range and step. Since per-component scheduling policies currently are expressed as parameters (see Section 4.2), basic scheduling policies can also be varied using this approach.

Parameter ranges are chosen such that they include the complete range within which they should be able to vary at runtime, and the step size is simply chosen to make the measurement process run in a reasonable amount of time on the chosen hardware. For the concrete example of Figure 3 (the TX2 executing the `matrix-gpu` component as a function of the size and exponent parameters), the physical measurements are performed on the parameters' values shown on the two axes, and the total running time is roughly one hour. In the drone example described in the introduction, the drone would be able to adjust parameters dynamically as the battery charge decreases, shifting from the area with major energy consumption to the minor (from the black area in Figure 3c to the gray/white area).

5.3 Approximation Method

The approximation method controls how approximations of energy usage in the prediction model are computed from a limited number of samples. This allows predictions of energy

usage for any combination of component parameters within the sampled range. Different approximation methods can be supported. However, since we have no a priori knowledge of the actual energy model, specific models such as linear or polynomial models cannot in general be assumed. For this reason, we simply interpolate the energy usage at a given point based on averaging nearby concrete samples.

Concretely, approximations can be computed using a weighted average of nearby measurements. Assuming all component parameter ranges are normalized, we approximate a vector $\overrightarrow{s}$ of predicted values (i.e., energy consumed, battery drained) for component parameters $\overrightarrow{x}$ as follows:

$$\overrightarrow{s}(\overrightarrow{x}) = \frac{\sum_i \overrightarrow{s_i} \cdot f(|\overrightarrow{x} - \overrightarrow{x_i}|)}{\sum_i f(|\overrightarrow{x} - \overrightarrow{x_i}|)} \tag{4}$$

where $\overrightarrow{s_i}$ is the measured value at point $\overrightarrow{x_i}$, and f is a weighting function defined as follows:

$$f(\gamma) = e^{\frac{-\gamma^2}{\sigma^2}} \tag{5}$$

The value σ determines the bandwidth, higher values increase the distance at which measured points are included for measurement.

6 Experimental Setup

In the following section, we present hardware platforms under study, the two measuring technique to obtain power metrics, and a summary of the profiling tool. Finally, Experimental Methodology outlines our experimental approach before introducing results in the next Section.

6.1 Hardware Platforms and Benchmarks

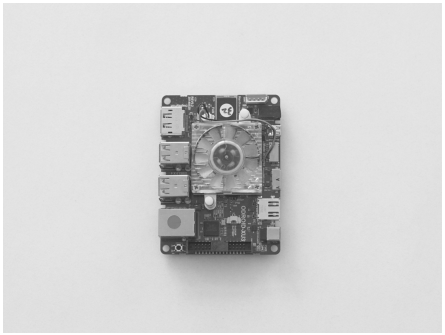
We studied four different hardware platforms, ODROID XU3, NVIDIA Jetson TK1, TX2, and Nano that are shown in Figure 5:

ODROID XU3 Provides an ARM Cortex-A15 and -A7 CPU, 2 GBytes of LPDDR3 RAM, a MALI GPU, and storage via microSD. It includes built-in sensors that enable accurate power measurements of the two CPUs, RAM, and GPU.

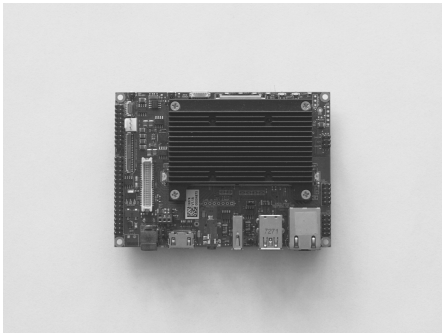
NVIDIA TK1 Provides an ARM Cortex-A15 CPU, 2 GBytes of DDR3L RAM, 16 GB of non-volatile storage, and an NVIDIA Kepler GPU with 192 CUDA cores. The TK1 does not provide any built-in sensor for power measurement, so an external sensor is used to measure the total power consumed.

NVIDIA TX2 Provides an ARM Cortex-A57 CPU, 8 GBytes of LPDDR4 RAM, 32 GB of non-volatile storage, and an NVIDIA Pascal GPU with 256 CUDA cores. The TX2 comes with on-board power measurement functionality that can measure the power of different components, and was used in our model to gather independently the instantaneous power usage of the CPU, GPU, and the whole board.

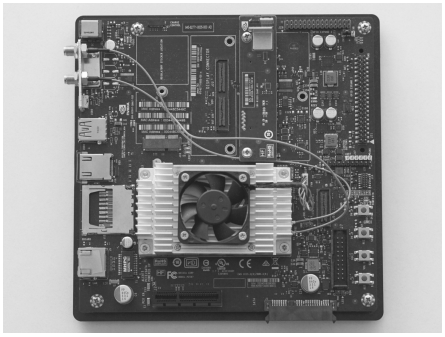
NVIDIA Nano Provides an ARM Cortex-A57 CPU, 4 GBytes of LPDDR4 RAM, an NVIDIA Maxwell GPU with 128 CUDA cores, and storage via microSD. Similarly to the TX2, the Nano comes with on-board power measurement functionality that is able to measure the power of different components and can be used within our model to gather independently the instantaneous power usage of the CPU, GPU, and the whole board.



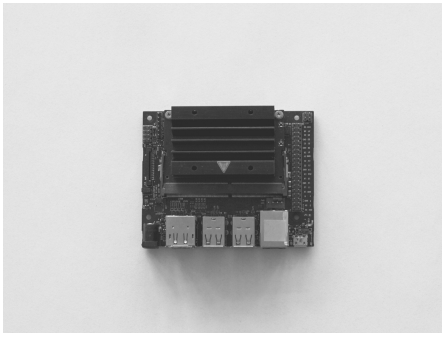
(a) ODROID XU3



(b) NVIDIA Jetson TK1



(c) NVIDIA Jetson TX2



(d) NVIDIA Jetson Nano

Fig. 5: Hardware platforms used in the analysis. ODROID XU3 5a is a 94x70 mm integrated heterogeneous board. NVIDIA Jetson TK1 5b uses a Toradex Ixora 125x95 mm carrier board, TX2 5c and Nano 5d a 170x170 mm and a 100x80 mm Jetson Developer Kit board respectively. All the images have the same scale.

Several benchmarks (see Section 3) have been performed on the four hardware platforms under study. (Not all benchmarks components are compatible with every platform, e.g., GPU-based benchmarks are implemented in CUDA which only runs on NVIDIA platforms, limiting the set of experiments that can be performed.)

6.2 Power Measurement

We use two different power measurements techniques for our experiments: the current shunt and current probe. With the current shunt method, an internal circuit on the embedded board composed of a shunt resistor is used, along with a digital acquisition unit. It is the preferred approach as it can precisely sample the power consumption of specific computing elements as CPU and GPU. Such circuits are present on the ODROID XU3 and NVIDIA TX2 or Nano boards and can be sampled directly by the `powprofiler` tool described in Section 6.3.

As an alternative to the current shunt method, we have also implemented the current probe method, where an external circuit measures the power on the connection to the main embedded board (i.e., the carrier board). We adopted the current probe method to measure

the overall power consumption of the NVIDIA TK1. This measurement setup consists of three hardware units, henceforth referred to as nodes. The first node is the board under analysis, the second node a multimeter that performs the sampling of the power consumption at a specific sampling frequency, and the third node is an external computer that collects the data for subsequent processing by `powprofiler`. The whole setup is described in detail in our prior work [31]. Data and metrics are stored for use with `powprofiler`, essentially allowing an arbitrary power measurement technique for the system under analysis. This means that a layer 2 model can be generated from the data with no distinction of what type of measuring device was used to obtain the layer 1 model.

All experiments reported in this section are performed using the standard performance governor present in each of the systems under observation. In particular, dynamic effects such as DVFS have not been disabled. Our experiments have not revealed any effects on performance due to, e.g., ambient temperature or the temperature of the embedded systems increasing.

6.3 The `powprofiler` tool

The `powprofiler` tool implements all the features described in this paper and is designed to be useable for heterogeneous parallel embedded systems. The tool supports all the platforms described in Section 6.1. It is designed for extensibility and can, in principle, support any Linux-based embedded device that provides power measurement metrics. Concretely, to support a new embedded device, a subclass defining the device-specific features can be derived from a virtual class, and only requires the implementation of a function that returns the current measurements in the form of a weighted vector where every element can represent a different metric. In a first iteration, the tool profiles a set of configurations of a component and builds a layer 1 model as described in Section 4. Once `powprofiler` obtains power profiles and other metrics that are considered useful for the purpose of energy modeling, it builds a predictive model as described in Section 5. The tool relies on `gnuplot` utility to visualize energy evolution if the configuration search space allows it. Automatically generated output includes CSV files containing measured data as well as the 2D- and 3D-plots shown in this paper.

6.4 Experimental Methodology

The power modeling experiments described in this paper are, unless otherwise mentioned, done using `powprofiler` tool. The tool uses two temporal schemes to profile data. A component can be profiled for a specific amount of time with the first, or until it terminates its computation with the second. The second scheme is used for all the components that end in a specific amount of time, i.e., matrix multiplication. The first for components that run continuously, i.e., an image detection algorithm operating on an image stream. For both, the longer the profiling, the better the accuracy.

Ideally, any component combination can be profiled by specifying possible sets of parameters in the configuration file. Each entry corresponds to a component and defines how its execution is varied as described in application specification described in 4.2. In summary, a layer 1 model is evaluated and stored for every possible combination. The layer 1 model is then integrated with the battery model. Data containing energy consumption, battery drain, and other information as system load, are evaluated into the layer 2 model. This model builds

an n-dimensional function and provides energy consumption data for every possible combination. The missing data are integrated using estimation with the approximation technique describe in 5.3. The n-dimensional space is derived by the use of the battery model to map each value with a weight describing the SoC.

As an example of energy modeling, we implemented a combination of three different parameters and generated a surface that shows approximately where the minimum power consumption is expected. Figures 3–4 shows how the output of the predictive model includes, but is not limited to, overall energy 3b–4b and average power consumption 3a–4a for all the combinations.

7 Experimental Results

This Section shows and assesses experimental results for the benchmarks, and validates the presented approach.

7.1 Benchmarks

We now describe the experimental results of the benchmarks previously introduced. A summary of these results is outlined in Table 1.

Component	ODROID	NVIDIA		
	XU3	TK1	TX2	Nano
`matrix-cpu`	5284 J	4067 J	2413 J	2736 J
`matrix-gpu`	-	81 J	45 J	39 J
`darknet-cpu`	(-)	(-)	2400 J	(-)
`darknet-gpu`	-	-	5255 J	(-)
`nvidia-matrix`	-	(-)	4054 J	(-)
`nvidia-quicks`	-	(-)	1995 J	(-)

Table 1: The overall energy consumption for each benchmark. Unsupported platforms are indicated by '-' and '(-)' indicates supported but not included in this paper.

Matrix Exponentiation

Execution of GPU matrix exponentiation, while varying size and exponent parameters, was previously shown in Figure 3. The figure shows the average power as well as overall energy consumption along with the battery depletion as a function of size and exponent parameters. Average power consumption is reported independently of the running time of the component and thus does not reflect the total power consumption. For small problem sizes, the computation terminates before reaching the maximal power level. This effect is visible in Figure 2 (also previously shown), where power consumption is low at the beginning and then reaches the maximum, for which reason the average power consumption is low for small problem sizes. Battery depletion is reported in terms of the total amount of energy consumed by the computation for the duration of the execution. The effect of introducing "scheduling" in the form of sleep of various durations in between iterations of the matrix computations can be seen in Figure 4. Here, the duration of the sleep affects the total power

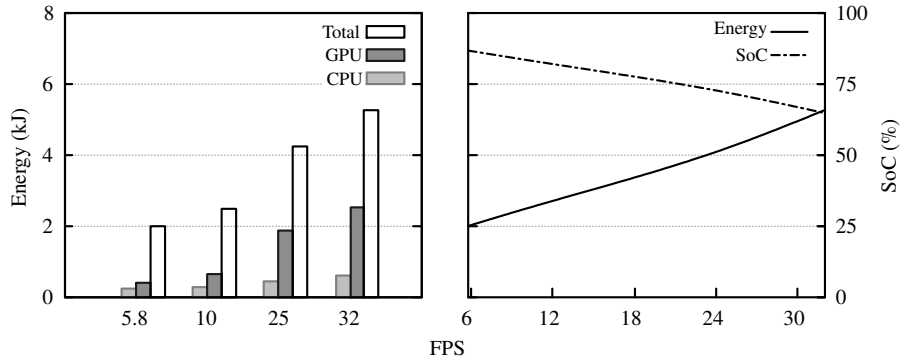


Fig. 6: Layer 2 model of `darknet-gpu` component running under four configurations, respectively 5.8, 10, 25 and 32 frames-per-second. The figure shows the per-minute energy consumption in terms of CPU, GPU, and overall. On the right side, energy consumption for any possible frame per second rate is shown along with SoC (the y2-axis, the y-axis is shared with the left side).

consumption: the higher the sleep value in between the iterations, the greater the battery depletion.

CPU vs GPU comparison shows, expectedly, that `matrix-gpu` is very performant compared to `matrix-cpu`. While the `matrix-cpu` requires 2 413 J, `matrix-gpu` requires only 45 J for the same operation on the TX2 board. Therefore, running the benchmark on GPU results in 16% more SoC against the same trial on CPU. Such a high speed-up is observed due to the highly parallelizable nature of the matrix exponentiation and hence cannot be used as a general rule. From our experiments, we additionally observe the power-related effect of running components sequentially versus in parallel on different computational units. Although the CPU and GPU are different computational units, the energy consumed by running components independently (i.e., sequentially in some order) on CPU and GPU is 20% larger compared to running them in parallel, even when subtracting the base power consumed by the board. Thus energy can be conserved by running computations in parallel on the CPU and GPU, compared to scheduling them sequentially.

Darknet

We modified the `darknet` component to simulate different scheduling options and evaluated the outcome on a video stream: `darknet` now accepts an argument that indicates the amount of sleep between two invocations of the image recognition algorithm. In this way, we were able to simulate different frames-per-second options and assess the power evolution using the model.

Our experimental data depicted in Figure 6 shows that an increment in frames-per-second corresponds to an increased power consumption along with a higher battery depletion. The resulting model can be used to define an appropriate trade-off that represents an acceptable rate of QoS, by i.e., correlating the FPS rate to the battery depletion and hence highlighting the dynamic behavior of a mobile scenario. Moreover, we observe that `darknet-cpu` consumes less energy per minute compared to the `darknet-gpu` component, but is considerably slower: while running `darknet-cpu` for one minute on TX2 board requires 2 400 J, `darknet-gpu` requires 5 255 J. When considering the energy cost per frame the computation is however

considerably more efficient on GPU, where it requires just 1.3 J per frame, against the 175 J on GPU.

NVIDIA

The `nvidia-matrix` benchmark differs from `matrix-gpu`, even if both perform matrix multiplication on GPU, since `nvidia-matrix` includes a significant CPU computation after GPU matrix multiplication to check whether the two match. The `matrix-gpu` benchmark was similarly tested during development, but does not perform this test at runtime. We observe that the `nvidia-matrix` benchmark has a similar energy behavior to `matrix-gpu`, with the problem size affecting the overall energy and henceforth battery depletion. Average power consumption on GPU is however higher for `matrix-gpu` while it is approximately the same on CPU for both benchmarks (presumably none of the two benchmarks focus on energy efficiency on CPU). For the quicksort benchmark, as expected energy consumption increases with the problem size, as more operations are performed. Nevertheless quicksort differs from matrix multiplication: there is more noise due to a higher dependence on the random data that is being sorted.

7.2 Validation

We validate our approach by: **a)** demonstrating that `powprofiler` can be used on a number of heterogeneous platforms, **b)** comparing model generated with internal metrics to external physical measurements on the TX2, and **c)** comparing the model to a fine-grained one.

A cross-platform comparison shows energy-related behavior of running the same benchmark on different boards. We observed, for instance, that the most energy-efficient board for `matrix-cpu` benchmark is TX2, which consumes 2 413 J to perform the operation, followed by Nano with 2 736 J, TK1 with 4 067 J, and ODROID with 5 284 J.

A measurement analysis is used to compare internal against external power metrics on the TX2 board. We observe that both externally and internally measured power models are close one to each other, with an error of less than 3% measured over one minute. The externally measured power exceeds the internally measured power — this is natural as the externally measured power also includes the carrier board, which requires additional energy to operate. Moreover, the measurements performed using `powprofiler` include the power consumed by `powprofiler` itself, while the multimeter setup excludes `powprofiler`'s energy from the evaluation. We therefore assume that the tool has a marginal effect on power consumption and that the model is showing realistic behavior.

In a last validation step, we compared our approach to fine-grained energy model of Nunez et al. [24] and Nikov et al. [23] for the ODROID-XU3 embedded board. To relate the two approaches for energy-modeling, we used the `matrix-cpu` benchmark component as follows. First, we evaluated the matrix exponentiation benchmark by raising a 512 by 512 matrix to the 30th power (512^{30}) to train the fine-grained model. We obtained a value of expected energy per operation that we compared to the measured one and measured a relative error of 3.42%. Second, we applied an equivalent approach to our model. We sampled some configurations that were distant from the expected one, concretely we ran configurations 512^x with $x \in \{5, 15, 25, 35, \ldots\}$, and we used the approximation method described in Section 5.3 to evaluate the energy of the configuration 512^{30}. This value was then compared to the measured one and we obtained a relative error of 2.25%. We can conclude that both

models produce similar results on this benchmark, and have a similar relative error even if they adopt a different approach towards energy-modeling.

7.3 Assessment

We performed a number of experiments on different boards. Most of the data in this paper were obtained from NVIDIA TX2 due to the similarity with TK1 and Nano and the easy accessibility of the internal power measurement units. These data allowed us to validate our model, and to show that different scheduling options can correspond to different energy models. The model can describe energy consumption and instantaneous power, together with the battery depletion.

Coarse-grained whole-system energy modeling can take into account some behaviors that cannot otherwise easily be observed. As an example, consider the effect of simple scheduling previously shown in Figure 4. Here, we expected that introducing a higher sleep period between executions would result in an energy cost. We used the model to investigate this assumption, and we found that it is not always happening. For instance, a period of 2.5s between two consecutive schedules of 512^{12} matrix exponentiation iterated for 12 times is more energy-efficient than executing the whole 512^{144} operation. The model can relate these observations, provide information about the battery depletion, and predict the total time the system can operate on a given energy source. It can also highlight battery-specific behaviors since different scheduling options drain battery differently. In particular, Using the `powprofiler` tool on the NVIDIA TX2, we experimentally observe that:

- Running a set of components separately, and simply adding their energy consumption (while excluding base energy), leads to a different model versus running them in parallel (Section 7.1). This behavior was observed for matrix exponentiation components running on separate computational units (CPU, GPU).
- Scheduling of computations directly impacts the battery drain: processing a computation in its entirety with a steady power load drains the battery less than scheduling that same computation into smaller steps with resulting spikes in the power load (Section 7.3). This behavior was observed for a matrix exponentiation component running on CPU.

8 Conclusion and Future Work

Coarse-grained whole-system energy modeling allows measurement and prediction of the energy consumption of a complete set of components executing on a given physical system. This approach is supported by our `powprofiler` tool, and is experimentally demonstrated to work across the ODROID XU3, NVIDIA TK1, TX2 and Nano platforms. It can be used to measure, analyze, and subsequently predict the energy consumption for single components performing specific computations, for multiple components performing computations in parallel, and for components executing on-demand in a dataflow. In particular, energy consumption can be predicted in terms of component configuration parameters, allowing the energy consumption of a given QoS level to be predicted.

These observations support the idea of coarse-grained modeling of the energy consumption of a whole system, defined as a network of components in a specific configuration being scheduled according to a specific policy. Using the numerical model generated by

`powprofiler`, energy consumption can be predicted for such scenarios. We showed that effective energy modeling can reduce overall energy consumption and increase autonomous capabilities for many modern applications by finding the right rate of QoS.

In terms of future work, we will define additional application models to support more advanced scheduling policies. Concretely, advanced robotics software is often developed using the ROS component-based middleware. A scheduler could be implemented such that it schedules components by publishing data to components that should be executed. This would allow arbitrary static or dynamic schedules to be executed by a single, central scheduling component implemented in ROS. A corresponding application specification could then be defined to allow power measurement of ROS-based systems.

References

1. (2019) Public deliverables of the TeamPlay horizon2020 project. `https://teamplay-h2020.eu/index.php?page=deliverables`, accessed: 2019-08-25
2. Amdahl GM (1967) Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the April 18-20, 1967, spring joint computer conference, ACM, pp 483–485
3. Bailey PE, Lowenthal DK, Ravi V, Rountree B, Schulz M, De Supinski BR (2014) Adaptive configuration selection for power-constrained heterogeneous systems. In: 2014 43rd International Conference on Parallel Processing, IEEE, pp 371–380
4. Barik R, Farooqui N, Lewis BT, Hu C, Shpeisman T (2016) A black-box approach to energy-aware scheduling on integrated cpu-gpu systems. In: Proceedings of the 2016 International Symposium on Code Generation and Optimization, ACM, pp 70–81
5. Calore E, Schifano SF, Tripiccione R (2015) Energy-performance tradeoffs for hpc applications on low power processors. In: European Conference on Parallel Processing, Springer, pp 737–748
6. Chowdhury P, Chakrabarti C (2005) Static task-scheduling algorithms for battery-powered dvs systems. IEEE transactions on very large scale integration (VLSI) systems 13(2):226–237
7. Dressler F, Fuchs G (2005) Energy-aware operation and task allocation of autonomous robots. In: Proceedings of the Fifth International Workshop on Robot Motion and Control, 2005. RoMoCo'05., IEEE, pp 163–168
8. Fabiani P, Fuertes V, Piquereau A, Mampey R, Teichteil-Königsbuch F (2007) Autonomous flight and navigation of vtol uavs: from autonomy demonstrations to out-of-sight flights. Aerospace Science and Technology 11(2-3):183–193
9. García-Martín E, Rodrigues CF, Riley G, Grahn H (2019) Estimation of energy consumption in machine learning. Journal of Parallel and Distributed Computing 134:75–88
10. Goraczko M, Liu J, Lymberopoulos D, Matic S, Priyantha B, Zhao F (2008) Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems. In: 2008 45th ACM/IEEE Design Automation Conference, IEEE, pp 191–196
11. Gustafson JL (1988) Reevaluating amdahl's law. Communications of the ACM 31(5):532–533
12. Hasan A, Skriver M, Johansen TA (2018) Exogenous kalman filter for state-of-charge estimation in lithium-ion batteries. In: 2018 IEEE Conference on Control Technology and Applications (CCTA), IEEE, pp 1403–1408

13. Hong I, Kirovski D, Qu G, Potkonjak M, B SM (1999) Power optimization of variable-voltage core-based systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 18(12):1702–1714
14. Hong S, Kim H (2010) An integrated gpu power and performance model. In: ACM SIGARCH Computer Architecture News, ACM, vol 38, pp 280–289
15. Kanellakis C, Nikolakopoulos G (2017) Survey on computer vision for uavs: Current developments and trends. Journal of Intelligent & Robotic Systems 87(1):141–168
16. Kreciglowa N, Karydis K, Kumar V (2017) Energy efficiency of trajectory generation methods for stop-and-go aerial robot navigation. In: 2017 International Conference on Unmanned Aircraft Systems (ICUAS), IEEE, pp 656–662
17. Luo J, Jha NK (2001) Battery-aware static scheduling for distributed real-time embedded systems. In: Proceedings of the 38th annual Design Automation Conference, ACM, pp 444–449
18. Marowka A (2012) Extending amdahl's law for heterogeneous computing. In: 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications, IEEE, pp 309–316
19. Marowka A (2017) Energy-aware modeling of scaled heterogeneous systems. International Journal of Parallel Programming 45(5):1026–1045
20. Mei Y, Lu YH, Hu YC, Lee CG (2004) Energy-efficient motion planning for mobile robots. In: IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004, IEEE, vol 5, pp 4344–4349
21. Mei Y, Lu YH, Hu YC, Lee CG (2005) A case study of mobile robot's energy consumption and conservation techniques. In: ICAR'05. Proceedings., 12th International Conference on Advanced Robotics, 2005., IEEE, pp 492–497
22. Mudge T (2001) Power: A first-class architectural design constraint. Computer 34(4):52–58
23. Nikov K, Nunez-Yanez JL, Horsnell M (2015) Evaluation of hybrid run-time power models for the arm big. little architecture. In: 2015 IEEE 13th International Conference on Embedded and Ubiquitous Computing, IEEE, pp 205–210
24. Nunez-Yanez J, Lore G (2013) Enabling accurate modeling of power and energy consumption in an arm-based system-on-chip. Microprocessors and Microsystems 37(3):319–332
25. Rao R, Vrudhula S, Rakhmatov DN (2003) Battery modeling for energy aware system design. Computer 36(12):77–87
26. Redmon J (2013–2016) Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`
27. Redmon J, Farhadi A (2017) YOLO9000: better, faster, stronger. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, USA, pp 6517–6525
28. Redmon J, Divvala S, Girshick R, Farhadi A (2016) You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 779–788
29. Sadrpour A, Jin J, Ulsoy AG (2013) Experimental validation of mission energy prediction model for unmanned ground vehicles. In: 2013 American Control Conference, IEEE, pp 5960–5965
30. Sadrpour A, Jin J, Ulsoy AG (2013) Mission energy prediction for unmanned ground vehicles using real-time measurements and prior knowledge. Journal of Field Robotics 30(3):399–414
31. Seewald A, Ebeid E, Schultz UP (2019) Dynamic energy modelling for soc boards: Initial experiments. In: HLPGPU 2019: High-Level Programming for Heterogeneous

and Hierarchical Parallel Systems, p 4
32. Siciliano B, Khatib O (2016) Springer handbook of robotics. Springer
33. Uragun B (2011) Energy efficiency for unmanned aerial vehicles. In: 2011 10th International Conference on Machine Learning and Applications and Workshops, IEEE, vol 2, pp 316–320
34. Woo DH, Lee HHS (2008) Extending amdahl's law for energy-efficient computing in the many-core era. Computer 41(12):24–31
35. Wu G, Greathouse JL, Lyashevsky A, Jayasena N, Chiou D (2015) Gpgpu performance and power estimation using machine learning. In: 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), IEEE, pp 564–576