IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 1 August 2022, accepted 19 August 2022, date of publication 25 August 2022, date of current version 2 September 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3201501

RESEARCH ARTICLE

SLePaaS: An Embedded Platform-as-a-Service Facilitating Research on Thermal Management of Embedded Platforms

RAKESH KUMAR, (Member, IEEE), AKASH SACHAN, AND BIBHAS GHOSHAL
Department of Information Technology, IIIT Allahabad, Prayagraj, Uttar Pradesh 211015, India

Corresponding author: Rakesh Kumar (pse2015002@iiita.ac.in)

This research work is supported by the Ministry of Human Resource and Development, Government of India, under the scheme "Imacting Research Innovation and Technology" (IMPRINT-INDIA), under Project 7482.

ABSTRACT In this paper, we present to the embedded research community an *Embedded Platform as a Service* facility named *SLePaaS* that allows researchers remote access to experimental hardware and a collection of online tools that would facilitate research on thermal management of embedded systems. *SLePaaS* serves a dual purpose - provides access to actual hardware for experiments on one hand and reduces experimental cost and time on the other. With this facility, researchers neither have to rely on simulation models nor purchase or customize any experimental platform for a specific problem. In addition, the platform provides tool support, specifically for researchers working in problems related to thermal management. Presently, three tools are provided by the SLePaaS platform. The first tool helps users obtain performance, power and thermal profile of applications running on different hardware platforms. The second is a thermal value predictor tool that aids task schedulers in allocating tasks to processors based on the thermal profile of applications. The third tool is a platform predictor that helps users to decide the best platform to run an OpenCL application given a target optimization objective. All the tools have been validated for their performance and accuracy against standard benchmark applications. A prototype implementation of *SLePaaS* along with the three tools is available for use.

INDEX TERMS Embedded system, thermal management, task scheduling, embedded platform, heterogeneous processor, OpenCL, RMI.

I. INTRODUCTION

Thermal management of embedded systems is a well researched topic and is still popular within the embedded research community. The thermal issues arising out of increased power density in modern day heterogeneous embedded systems have fuelled research on thermal management of these systems. Literature suggests that researchers have proposed a number of techniques for thermal management of embedded systems. The proposals primarily include resource management through load balancing of tasks on cores [9], [33], efficient thermal aware task and thread mapping [20] and thermal aware scheduling of tasks [36]. In addition to these static techniques, proposals for dynamic thermal management have been proposed as well where workload is decided based on the core temperature that is feedback [23], [43]. Present day embedded systems are heterogeneous devices integrating CPU and GPU cores. Thus, thermal management of such devices become far more challenging. Thermal effects on the system as a result of CPUs and GPUs manifest in different ways. A common technique employed for thermal management of such devices is effective GPU kernel management - choosing judiciously the tasks to be scheduled on GPUs and CPUs [17].

It has been observed that proposals for thermal management techniques of embedded devices have been validated either by implementing them on hardware platforms or through simulation runs. Though results obtained from hardware implementation are considered more accurate, obtaining these results become difficult. Researchers have to either

The associate editor coordinating the review of this manuscript and approving it for publication was Liang-Bi Chen.

look out for specialized experimental platforms that provide support for obtaining temperature readings or have to attach some add-on circuitry to the existing boards to collect thermal data. In case such experimental platforms are not available, researchers resort to simulation environments and modeling tools to estimate the power and thermal readings. Simulation models mimic the functionality of the real hardware and thus facilitate an alternate validation procedure of an idea.

Though simulation platforms serve as an alternative to real hardware, they have their own limitations. One such limitation is the accuracy of the models employed during simulation. Inaccurate or insufficient models lead to errors in findings. For example, Anastassia Butko *et al.* in [8] explored the widely used *McPAT* [32] modelling framework used for modelling power, area, and timing of many-core and multi-core processors and reported that the error caused by the *McPAT* model while computing power dissipation was 13% while average error reported during computation of performance was 20%.

Some simulation platforms employ a set of dependent models. In such cases, if any one of the model gets erroneous, the error propagates to all the dependent models causing inaccurate simulation results. For example, an error in the power model employed in the *Hotspot-6.0* tool propagates to its thermal model and leads to inaccurate temperature evaluation of the chip.

It has been observed that in order to perform a simulation based experiment, it requires integration of a number of software tools which are independent of one other, in both design and working. Thus, building a chain of such independent tools is a time consuming task. Moreover, a lot of compatibility and configuration issues may crop up which need to be handled in order to build the tool chain. These issues remain specific to the problem a researcher is currently working on and does not apply to tools required for other problems or by other researchers. Addressing these problems often becomes tedious.

A. MOTIVATION

The challenges mentioned in the previous subsection are usually faced by researchers while dealing with problems related to thermal management of embedded systems. Some of them reflect our experiences while working on problems mentioned in [25], [26], and [37] and others were shared by colleagues working on similar problems.

To summarize, the problems faced by the embedded systems research community while dealing with simulation models and hardware platforms can be listed as follows :

1) Simulation tools are model driven involving a number of dependent models. Thus, inaccuracy in one model can cause deviation in the result of some other model leading to overall degradation of the tool performance.
2) Building a simulation tool chain involves configuration of a number of independent tools which often becomes tedious and takes a lot of time. Moreover, one faces a lot of compatibility issues with the host platform while dealing with a set of independent tool set.
3) Each simulation tool has its set of constraints. These constraints severely limit the usage of a tool. For example, *Hotspot-6.0*, a very popular simulator for studying thermal effects in processors, requires a thermal floor plan of the hardware platform (processor) to be provided as input. However, thermal floor-plans for different embedded devices is not readily available and building the floor plan without block architecture and thermal parameter is not trivial. Thus, usage of *Hotspot-6.0* is only limited to processors whose floor-plan is available.
4) Hardware platforms used for embedded research have their own issues. Each platform is too specific for a particular problem and often customized to facilitate collection of power and temperature readings.
5) Prior knowledge of the thermal behaviour of an application can help task schedulers to choose the suitable processor for execution in order to maintain the temperature of the platform within an optimized range.
6) There can be unforeseen situations such as the one witnessed during the COVID-19 pandemic, when researchers would not have physical access to lab and its resources. Thus, it would be impossible to run the experiments which would stall the research progress.

The problems cited above necessitates development of a facility that would allow users remote access of embedded platforms through suitable and easy interfaces so that they can carry out their experiments on actual hardware platforms in real time with better accuracy. The platform should also provide necessary help to researchers through supporting tools and should cater to not only a certain group of researchers but the entire embedded research community in general.

B. CONTRIBUTION

The difficulties faced by us and our colleagues while working on different problems related to thermal management of embedded systems such as difficulty in capturing power and temperature readings, finding a suitable thermal model, performing thermal aware task allocation or deciding suitable platforms for task execution motivated us to propose **SLePaaS**, an *Embedded Platform-as-a-Service* facility for the embedded systems research community where researchers can be provided remote access to experimental hardware in order to carry out their research on thermal management of embedded systems. Such a facility would enable embedded researchers to carry out their experiments on actual embedded hardware platform without the need to purchase them, at the same time avoiding use of simulation models. This would not only provide accurate research findings in real-time but would also release the researchers from spending time on building tool chains for experiments. To the best of our knowledge, this is the first attempt to provide the embedded systems research community an online platform to remotely connect to the host and interact with different embedded platforms - upload codes, profile them to obtain thermal statistics in real-time, and perform a thermal aware compilation of the codes. The contributions can be summarized as follows :

- Accessibility to Hardware Platform via host : The proposed *Embedded Platform as a Service* facility facilitates end-users(clients) to remotely connect to a host and interact with the embedded platforms via APIs and utilities provided. Currently, users can make use of three hardware platforms : *Odroid XU4*, *Jetson TX1* and *Jetson Nano*. However, we plan to extend the support for other platforms such as family of DSP processors from Texas Instruments and processors from STMicroelectronics.
- Profiling Tool : An online profiling tool called *Nirikshak*[1] which provides temperature profile of codes as they get executed on any one of the three experimental platforms. In addition to the thermal profile, the tool also can be used to generate power and performance profile of applications.
- Thermal Aware ML Model: A tool called *Sahayak*[2] that helps users predict the average temperature of the application prior to the execution of the application. Knowledge about the average temperature of an application in turn helps task schedulers to perform a thermal aware task allocation on processor cores.
- Platform Predictor : A platform predictor called *Nirdharak*[3] which aids users in deciding the suitable platform of execution for OpenCL applications such that the thermal budget is maintained while the performance too is improved.

C. ORGANIZATION

The rest of the article is organized as follows. The next section provides a brief survey of literature on thermal management of embedded systems. It discusses the methods and models employed by researchers to measure temperature, followed by discussion on the state of the art tools in use for the purpose and their limitations. Section III provides the overall idea about *SLePaaS* the Embedded Platform-as-a-Service facility provided by *Systems Lab of IIIT Allahabad*, introducing the concept of the platform, the profiling tool, the platform decider and thermal predictor. Section IV describes the work flow of *SLePaaS* and role of each stakeholder. The subsequent sections detail the concept and working of each of the tools, first the concept and working of *Nirikshak* (the tool which provides temperature profile of applications) is presented in Section V followed by discussion of *Sahayak* (the thermal predictor) in Section VI and then finally the platform decider for OpenCL applications called *Nirdharak* is discussed in Section VII. A proof of concept of *SLePaaS* is presented in Section VIII to illustrate the working procedure of each tool and to evaluate the performance of each. Section IX provides the conclusions and our future plan with the platform.

[1]Hindi word meaning *one who monitors*.
[2]Hindi word meaning *one who assists*.
[3]Hindi word meaning *one who decides*.

II. RELATED WORK

As embedded systems tend to accommodate general purpose cores on the system-on-chip (examples are the *Samsung Exynos* or *Texas Instruments OMAP*), power density within these systems tend to rise causing elevation in operating temperature. Prolonged execution at elevated temperature causes several problems, such as introduction of faults, device wear out and higher energy consumption [50]. Thus, thermal management of embedded systems has gained importance over the years.

Thermal management techniques applicable for general purpose systems such as use of heat sink or fan can not be used for embedded systems due to space and energy constraints. Thus, software based techniques for thermal management are preferable. The most common techniques for software based thermal management have been thermal aware task partitioning and scheduling [14], Dynamic Voltage Frequency Scaling [29], thread migration [25], temperature aware task scheduling [10], [21] etc. An extensive survey on different thermal management techniques for microprocessors including embedded systems have been presented in [24].

Efficient and accurate monitoring of temperature is an important aspect in any proposal related to thermal management of a device (including embedded systems). Joonho Kong *et al.* in [24] list the various ways of monitoring temperature in microprocessors and classify them into three broad categories - thermal sensor based estimation, thermal model based estimation and software techniques. Thermal sensors either detect hot spots (digital sensors) or estimate die temperature(analog sensor). Model based estimation relies on measurements under process variation. Various models have been utilized such as Kalman filter based models [41], POMDP [46] power model and Performance counters [30]. Software techniques have been proposed with an aim to detect localized hot spots which could not be monitored using thermal sensors. They use sensor data, performance counter values, power and temperature models [30]. Baver *et al.* in [34] propose a processor temperature estimation model where they utilize built in temperature sensors of *Intel i7* and *ARM Cortex A-53* to determine system dependent model parameters and then go on to propose a generic modelling framework for dynamic temperature estimation.

Mohsen Ansari *et al.* in [4] proposed low energy/power aware scheduling techniques that meet both timing and reliability constraints. The proposed techniques optimise the energy consumption subject to thermal design power and core level thermal safe power constraints. To meet the timing constraints of the real time applications, the authors have scheduled tasks on suitable cores utilising different voltage/frequency variation of the embedded cores. Authors of [5] proposed a thermal-aware standby-sparing (TASS) technique that maximizes the quality of service of real time applications under given core level thermal safe power constraints. The proposed technique schedules the main task on the primary core while the backup tasks are schedule on spare core. The proposed fault tolerant techniques remove the overlapping of execution of main and backup task and drop the backup task on completion of main task. This ensures reduction in power consumption. Sepideh Safari *et al.* in [38] proposed thermal aware scheduling scheme that satisfies timing, temperature and reliability of high critical tasks while maintaining the QoS of low critical tasks. Authors have employed N-Modular Redundancy fault tolerant technique to ensure the reliability of multi-criticality system. Rajesh Devaraj and Arnab Sarkar in [12] proposed supervisory control synthesized framework that models real time applications as precedence-constrained task graphs and schedules the tasks such that timing constraints of each tasks get satisfied while maximizing fault tolerant capacity and minimizing peak power dissipation. Authors of [13] proposed an offline scheduling framework that guarantee the adherence of chip level power constraints while executing non-preemptive real time task.

Siqi Wang *et al.* proposed a framework *Optic* [47] that automatically predicts the partitioning point and operating frequency of CPU and GPU running concurrently under given thermal constraints. S.Sharifi *et al.* in [40] proposed *PROMETHEUS*, a framework for proactive temperature aware scheduling for embedded workload on heterogeneous processors. The authors proposed a thermal RC network based temperature predictor model *Tempo*. *PROMETHEUS* uses the temperature predictor to prepare thermal aware task assignment, migration and power aware scheduling mapping. Narendra Kumar Shukla *et al.* in [42] proposed context sensing based energy profiling for smartphones. Authors utilized sensor data and resource utilization to model power and energy. Yi-Fan Chung *et al.* in [11] proposed *ANEPROF*, an energy profiler that profiles energy consumption of the android applications at function level (distinguished power consumption among threads, java methods and java virtual machine services). Authors utilized real time measurements and correlated them with function level events to model energy profile. Youngmoon Lee *et al.* in [31] proposed *RT-TRM*, a real time thermal aware resource management framework for devices which have dynamic ambient temperature profile. Authors proposed task level dynamic power model, adaptive parameter assignment and online scheduling framework to perform thermal aware resource management.

Jude Angelo Ambrose in [3] proposed an interactive thermal management framework *HEATSMART* for multi-processor system on chip(MPSoC). The *HEATSMART* provides user interface to monitor the heat dissipation of each application and control the heat dissipation by giving the instruction such as stop the application, pause the instruction, migrate the application to other core or executes system level thermal management. Sumeet S.Kumar *et al.* in [27] proposed *Ctherm*, an integrated framework for cycle-accurate thermal and functional evaluation of systems-on-chip. *Ctherm* works in two stages. In first stage it generates the physical model based on generation of floor-plan and the estimation of latency, energy and dimension of the component. In second stage, *Ctherm* performs thermal-functional evaluation utilising physical model and SystemC top level file configured with system specification. Authors of [28] addressed the problem of high-power consumption due to display interface chipset used in smartphone. They have proposed thermal management for NANS (N-App N-Screen) services utilising DVFS and predictive thermal model. Onur Kayiran *et al.* in [22] proposed a framework (μC state) based on power and clock gating mechanism. They analysed the data path using queuing theory principle and found out the underutilised data path in big cores of the GPU. The μC state framework turns off the data path component which are underutilised or which do not become bottlenecks for the performance of the running application. Abhijeet Banerjee *et al.* in [6] proposed automatic test generation framework for android application to detect energy hotspot/bugs. The framework detects the energy hotspot and bugs based on evaluation of energy consuming events such as improper management of network resources, background services, resources leak and improper use of wake lock. Similarly authors of [16], [35], [48] used event driven techniques to develop energy profiler framework for android applications. Mohammad Javed Dousti *et al.* in [15] introduced *ThermaTap*, an online power and thermal profiler for android applications and devices. *ThermaTap* comprises *PowerTap* and *Therminator2*. *PowerTap* uses kernel level macro to record system activity and generates event driven power traces of system components while *Therminator2* uses physical characteristics and power traces to model temperature profile.

As software techniques for thermal management of embedded systems were proposed, their implementation and validation required proper tool support. A number of tools have been proposed over the years and have been in use. The most commonly used tool for temperature estimation of embedded processors is *Hotspot* [44]. It is used for thermal modelling based on thermal characteristics of a processor. However, *Hotspot* has few limitations. For example, it depends on other tools such as *Watch* [7] or *McPAT* [32] and *Gem5* [1] to provide the power estimates required for thermal modelling. Thus, integrating *Hotspot* with these tools becomes tedious. Moreover, any error in any one of these tools has an impact on *Hotspot* and may eventually lead to error in temperature estimation. *Hotspot* also needs a thermal floor plan of the processor for thermal modelling. However, thermal floor-plans for different embedded devices is not readily available and building the floor plan without block architecture and thermal parameter is not trivial. Thus, dependence on other tools and limited availability of floor plans restricts use of *Hotspot*. Additionally, some researchers have reported that *Hotspot* has limitation in accuracy and simulation time [19]. *Varsim* is another thermal management tool proposed by Hameedah Sultan *et al.* in [45] that can capture the temperature dependence conductivity and help in fast leakage and variability aware thermal estimation. However, the tool has only a

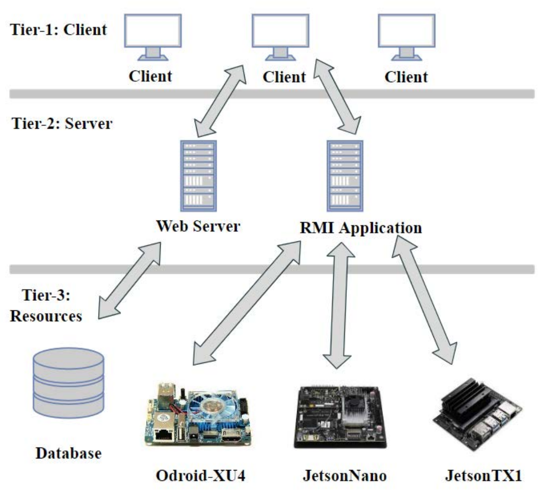

FIGURE 1. Architecture of SLePaaS.

limited usage. *Breazeflow* [18] is a tool that identifies wrong decisions taken by frequency governor and scheduler and thus helps kernel developers to efficiently perform thermal management.

The tools mentioned above have been in use and majority of embedded system researchers rely on them for temperature monitoring. However, all these tools have some limitations. They either depend on other tools, or lack proper library support or face compatibility issues with the host platform. Thus, we propose to release researchers from tool related issues and provide them with a platform so that they can carry out their experiments on actual hardware and report accurate results. The tool would additionally help them in taking more informed thermal aware decisions while executing their code.

III. ARCHITECTURE OF SLePaaS

Our proposed Embedded Platform as a Service facility *SLePaaS* is based on a Client-Server Architecture as presented in Figure 1. Tier-1 is the Service requester layer which represents clients (subscribers) who would like to avail the services (embedded platform for code execution). Tier-2 is the Service provider layer (Web application/Server) that provides services (access to the embedded platforms) to the clients. Tier-3 represents the resources provided to the clients (different hardware platforms and a database). The experimental setup of the proposed platform is presented in Figure 2.

A. THE SERVICE REQUEST AND SERVICE PROVIDER LAYERS

Clients who would like to avail the services *SLePaaS* need to subscribe through the following website: https://systemslab.iiita.ac.in/imprint/. Once subscribed, clients can request for an experimental session - request to run an application code on a specific target platform (*ODROID XU4 having Exynos SoC or Jetson Nano GPU board or Jetson TX1 GPU board*). Request from clients are forwarded to Tier 2 - the Service Provider layer, which depending on the request facilitates one of the services to the client. The following services are currently provided :

- **Code Analysis** : Facility to upload an user application to the *SLePaaS* platform and perform a static analysis (through control and data flow graphs generated out of the code) and dynamic analysis (through hardware events captured using performance counter values). The code analysis helps users to choose the most thermal aware execution platform - either CPU or GPU.
- **Profile Creation** : Facility to perform real time temperature profile generation for an user application, visualize and generate reports
- **ML based Thermal Predictor**: Predict the average temperature of the application prior to actual execution of the application.
- **Visualize and Download data of compiled benchmark applications** : Subscribers can visualize and download performance and thermal statistics of benchmarks such as *Polybench*, *ACC-Master*, *Parboil* and *Rodinia-3.1* when compiled on different target hardware platforms.

These services are provided in the Tier 2 layer through a *Web Server* and a RMI Interface. The Web server facilitates clients to upload their application codes to the computing nodes while the RMI interface allows them execute the uploaded code on a computing node (the target embedded platforms).

The Web Server : It provides an interface for the clients to interact with the computing nodes. Clients put a request to the web server which it accepts, processes and then sets up the environment to execute the application on experimental board. Once the execution is over, the web server makes arrangement for display of results related to performance, power, temperature variations etc. The web server expedites transfer of user application to the computing nodes (target embedded platforms) in two steps. First, it receives the client application as a file utilizing the Hyper Text Transfer Protocol(HTTP), and then makes use of connection-oriented socket programs to transfer the uploaded file received from the client to one of the computing nodes. During file transfer, the web-server acts as socket server while the computing node acts as socket client. The roles get reversed when results are transmitted back from the computing nodes to the web server. In addition, the web-server prepares separate experiment sessions for each client. The mechanism of session allotment is described in Section IV-B of this article.

RMI Interface : It allows clients to execute their codes on one of the computing nodes by invoking a remote method residing on a computing node. *RMI interface* is a remote java interface that declares a set of methods defined by the *RMI Server*. The *RMI Server class* defines the implementation of each method of *RMI interface* and registers them as *services* in the *RMI Registry*. In the proposed *SLePaaS* platform, three RMI interfaces (RMI Servers) have been implemented, one for each computing node. The *RMI* interfaces are identical

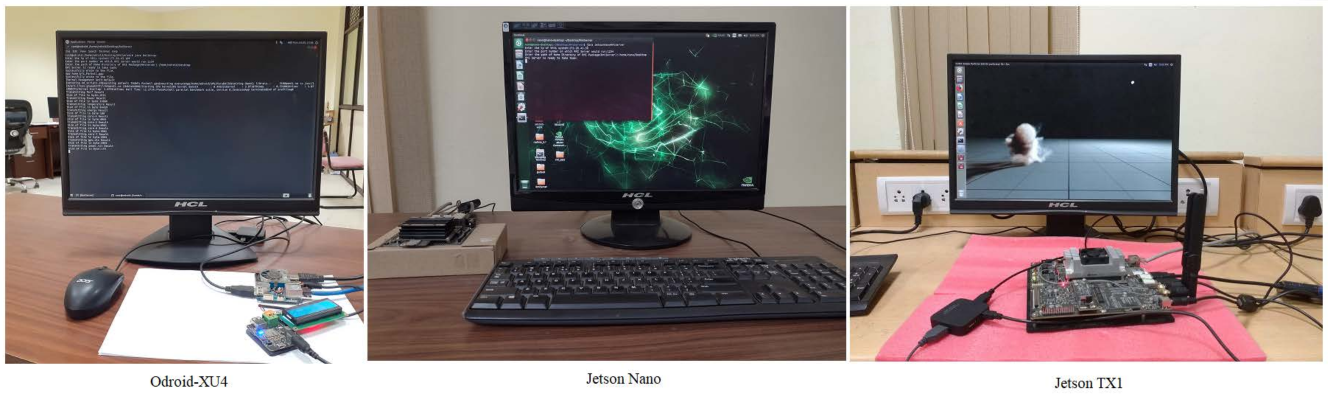

FIGURE 2. Experimental setup of SLePaaS.

at both client and server side. The RMI Client code resides at web-server machine and it runs on behalf of the user at the time an event gets triggered by the user. The client obtains the remote object of the server using *RMI registry* and thereafter it invokes the remote method.

B. THE RESOURCES LAYERS : TIER 3

The Tier-3 or the Resources layer of the *SLePaaS* consists of a database server and the computing nodes. The database server used is the *MySqlServer* and the computing nodes used are *Odroid-XU4*, *Jetson TX1* and *Jetson Nano*. In future we plan to add a few more computing nodes.

ODROID-XU4: The embedded platform *ODROID-XU4* is based on *ARM big.LITTLE* architecture having *Exynos 5422* processor with advanced *Mali T628 Graphics Processor Unit (GPU)*. The board is equipped with four high performance big cores and four low power LITTLE core. The big cores support 19 levels of clock frequency ranging from 200MHz to 2000 Mhz while the LITTLE cores support 13 levels of clock frequency within a range of 200-1400 MHz. The big cores have on-chip thermal sensors while the embedded GPU has six cores with OpenGL ES 3.1/2.0/1.1 and OpenCL 1.2 full profile support. The voltage and power values generated during application execution on the *ODROID-XU4* is captured using an add-on power measurement module called *SmartPower-2.0*(developed by HardKernel).

Jetson TX1: The NVIDIA Jetson TX1 GPU platform has 256 CUDA cores based on the NVIDIA Maxwell architecture delivering performance of the order of 1 TeraFLOPs. In addition it has quad core ARM Cortex A57 CPUs, 4K video encode and decode capabilities and a camera interface capable of 1400 MPix/s. The Jetson TX1 platform is suitable for running jobs related to deep learning, computer vision, graphics, and GPU computing. The power and thermal sensors on the board help in power and thermal management.

Jetson Nano: The NVIDIA Jetson Nano board is a GPU based embedded platform that allows parallel execution of multiple neural networks. This platform is specifically suited for applications such as image classification, object detection, segmentation, and speech processing. The board contains 128 GPU cores and quad core ARM Cortex A57 CPU. Moreover the board has power and thermal sensors to monitor the power and temperature of CPU and GPU.

IV. WORKING OF *SLePaaS*: THE EMBEDDED PLATFORM AS A SERVICE FACILITY

In this section we describe the role of the different components of the *Embedded Platform as a service* facility and explain its working in detail.

A. ROLE OF STAKEHOLDERS

As already described, *SLePaaS* uses a 3-Tier Client Server Architecture. Tier-1 represents clients who interact with the tool, Tier-2 represent the Server(Web Application) which provides services to the users while Tier-3 represent the resources (Database server and computing platform) of the tool. Thus, there are four stakeholders who interact with *embedded platform as a service* facility - the Client, An Admin, The Web Server and the Computing Node. Their role and interaction are illustrated using the Use-case diagram shown in Figure 3. Client and Admin are the human actors while Node and Web-Server are the non-human actors(system). The role of each actor is as follows :

Clients : Clients are online users who interact with the tool to perform experiments and obtain profile results. Clients need to register in order to avail the SLePaaS services. Once registered, clients can login, update profile and request for an experiment session. The experiment session defines the time slot (scheduled date and time) at which the computing node is available to the client. Once the session is scheduled, clients can view the scheduled date and time for the experiment session on a dashboard. On the scheduled date and time, client is allowed access to the computing node in order to run applications, perform different experiments on it and view the generated results. The sequence diagram to initiate session request is illustrated in Figure 4. The sequence diagram of other important usecases such as *AllotSession*, *ViewSession*, *StartExperiment*, *ExecuteProfiling*,

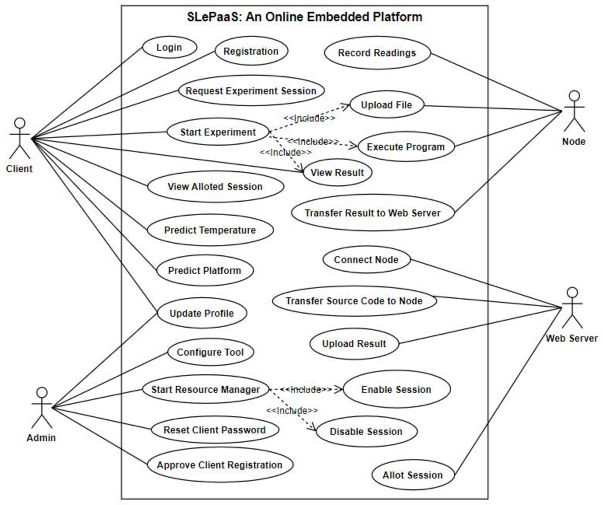

FIGURE 3. Usecase diagram of SLePaaS.

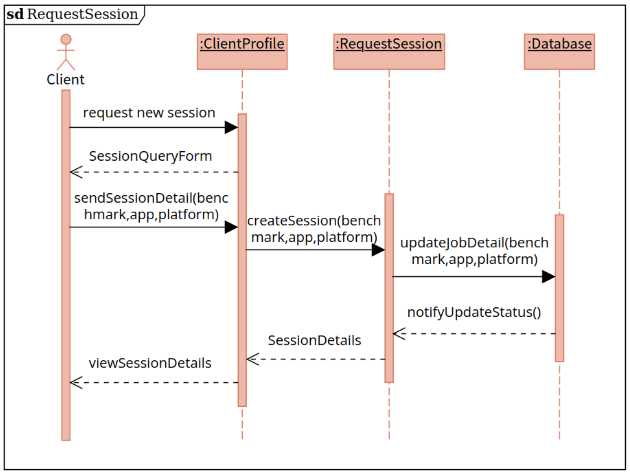

FIGURE 4. Request session sequence diagram.

PredictAvgTemperatute, *ViewResult* and *PredictPlatform* are presented in Appendix A. In addition, all sequence diagram are also provided at https://systemslab.iiita.ac.in/imprint/sequencediagram.html

Admin : The main role of admin is to configure the tool and start the resource manager application. The configuration of the tool includes configuration of *IP address* of each computing node, to set the *IP address* of the database server and initialize the database name, user name and password. This is an one-time process required at the initiation of the web-server. The resource manager is an application that runs three clocks namely *startClock-1,2 & 3*, which run continuously in the loop. The first clock (startClock-1) checks the scheduled date and time of the allotted application. If the scheduled date and time of the application matches with current date and time then the privilege of the respective application gets elevated and the application is put in a *wait state*. The other two clocks, check the current date and finish time of the application and lower the privilege of the application, if the finish time of the application crosses the current time. The application status is marked as *complete* if the finish time of the application crosses the current time. The other role of admin includes management of login - check and update passwords, approve the client registration and update admin profile.

Web-Server : The web-server is responsible for implementing the functionalities of the *SLePaaS*. It prepares an interface for client - computing node interaction. The role of the interface is to establish a connection between web-server and computing nodes(target hardware platforms), allow transfer of application code sent by the clients to the computing nodes, facilitate execution of the codes at the computing nodes and display results thereafter.

A socket based communication is used for data transfer between the web server and the computing nodes. When the web server needs to communicate with any one of the computing nodes, it acts as a *RMI Client* and calls the remote method corresponding to requirement of the client. For example, if a client wants to execute the uploaded application on the *Odroid XU4 embedded platform*, the web server calls the corresponding remote method of *Odroid RMI Server*. The web-server prepares the separate experiment session for each client. The mechanism to allot the experiment is described in Section IV-B. *RMI Servers* installed on each computing node launches a set of services for the clients, executes applications uploaded by the clients and sends the result back to the web server.

B. EXPERIMENT SESSION MANAGEMENT

Since the embedded platform as a service facility caters to multiple clients, managing experimental sessions created by multiple users becomes crucial. Multiple users accessing the same computing node (experimental board) at same time can influence the profiling results of the benchmark applications. To overcome this situation we create and maintain separate experiment sessions for each user. Each client subscribed to *SLePaaS* if wishes to execute an application code on some experimental board needs to request the web server for an experimental session through an online form mentioning about *benchmark name, application name, computing platform and user_id*.

Session Allotment : Experimental sessions are managed using two tables in the database, namely, the *wait_table* and the *run_table*. The *wait_table* stores the requests for the experiment sessions while *run_table* stores the scheduled session of each experiment. The *run_table* starts as empty and gradually fills up as the experiment sessions get scheduled. The technique for allotment of experiment sessions to various clients is explained through Algorithm 1.

The Algorithm1 takes two input sets *WQ*, *RQ*. The set *WQ*, *RQ* represent *wait_table* and *run_table* respectively. The attributes of the *WQ* table are user_id (u^i_{id}), job_id (j^i_{id}), expected execution time (eet^i), platform id (P^i_{id}) and status($stat^i$) of the job i. Similarly the attributes of RQ are schedule start date (d^i_s), start time (t^i_s), finish date (d^i_f), finish time (t^i_f), platform id (P^i_{id}), user id (u^i_{id}), job id (j^i_{id})

Algorithm 1 Allot Session

```
Input: WQ={$R_1, R_2, R_3 \ldots\ldots R_n$},
       RQ={$S_1, S_2, S_3 \ldots\ldots S_n$}
Output: $S_i$
Each $R_i$= $< u^i_{id}, j^i_{id}, eet^i, P^i_{id}, stat^i, priv^i >$
Each $S_i$= $< d^i_s, d^i_f, t^i_s, t^i_f, P^i_{id}, u^i_{id}, j^i_{id}, stat^i >$
1: $R_i \leftarrow getFirstRow\,(WQ)$
2: $P^i_{id} \leftarrow getPlatformId\,(R_i)$
3: $S_{i-1} \leftarrow getLastScheduleSession\left(RQ, P^i_{id}\right)$
4: $eet^i \leftarrow getExpectedExecutionTime\,(R_i)$
5: $d_c \leftarrow getCurrentDate\,(clock)$
6: $t_c \leftarrow getCurrentTime\,(clock)$
7: if ($S_{i-1} == NULL$) then
8:     $d_c \leftarrow getCurrentDate\,(clock)$
9:     $t_c \leftarrow getCurrentTime\,(clock)$
10: else
11:     $d_f^{i-1} \leftarrow getLastFinishDate\,(S_{i-1})$
12:     $t_f^{i-1} \leftarrow getLastFinishTime\,(S_{i-1})$
13:     if $\left(d_f^{i-1} > d_c\right)$ then
14:         $d_c = d_f^{i-1}$
15:         $t_c = t_f^{i-1}$
16:     else
17:         if $\left(d_f^{i-1} == d_c\right)$ then
18:             if $\left(t_f^{i-1} > t_c\right)$ then
19:                 $d_c = d_f^{i-1}$
20:                 $t_c = t_f^{i-1}$
21:             else
22:                 $d_c \leftarrow getCurrentDate\,(clock)$
23:                 $t_c \leftarrow getCurrentTime\,(clock)$
24:             end if
25:         else
26:             $d_c \leftarrow getCurrentDate\,(clock)$
27:             $t_c \leftarrow getCurrentTime\,(clock)$
28:         end if
29:     end if
30: end if
31: $t^i_s = t_c + \Delta T$
32: $d^i_s = d_c + \Delta T$
33: $t^i_f = t^{i+1}_s + eet^i$
34: $d^i_f = d^{i+1}_s + eet^i$
35: $S_i \leftarrow createNewSession(t^i_s, d^i_s, t^i_f, d^i_f, P^i_{id}, u^i_{id}, j^i_{id}, stat^i)$
36: Allot Session $S_i$ to the platform of id $P_{id}$
37: $RQ = RQ \cup S_i$
```

and status($stat^i$) of the job i. The algorithms begins with extraction of a request(R_i) from the wait_table WQ(line-1). The platform id of the extracted request is obtained by calling function *getPlatformId(R_i)* (line-2). The last schedule session on platform P^i_{id}, the same platform for which R_i made request, is obtained by calling function *getLastScheduleSession (RQ,P^i_{id})*(line-3). The expected execution time of the request R_i is extracted in line-4 while current date(d_c) & time(t_c) of the system is initialised in line-5 & 6 respectively. The set RQ either has an empty scheduled session or at least one scheduled session for the platform P^i_{id}. If set RQ has an empty schedule session or has the last scheduled session as NULL(line-7) on platform P^i_{id} then current date and time is initialized by obtaining the current date and time from the sql clock object(line-8,9). The current date and time is considered later as the start date and time of the next session. If the last scheduled session of set RQ is not NULL(non empty schedule) on platform P^i_{id} then start date and time for the next session is computed through line 11-29. If *run_queue* has at least one scheduled session then the last finish date and time of the last scheduled session is extracted in line-11 & 12 respectively. A comparison is performed between current date and last finish date(line-13) to find out if the last finish date is ahead of current date. In case it is true, then the current date and time is initialized to last finish date and time respectively(line-14, 15). The else part of line-13 further checks whether the cases of last finish date matches with current date and last finish date is behind the current date. If current date and last finish date matches then current date and time initialised through line 17-24. If last finish time is ahead of current time(line-18) then current date and time is initialised by last finish date and time respectively(line-19, 20) else current date and time is initialized by getting current date and time from the clock(line-22 & 23). If last schedule date is behind the current date then current date and time is initialized by getting the current date and time from the clock(line-26, 27). The start time of the session is computed by adding break time ΔT to the current time(line-31). The break time ΔT denotes the ready time for the client. The changes in current time can change the current date also. Therefore, start date of the experiment is computed by adding the break time ΔT to the current date(line-32). Similarly finish time and finish date is computed in line 33 & 34 by adding execution time of the application. A new session S_i is created with computed start date and time for the request R_i(line-35). The newly created session S_i is alloted to the platform of id P^i_{id}(line-36) and added to the set run_queue RQ(line-37).

Algorithm 1 involves mostly initialization(line 5-34) instructions that execute in constant time (O(1)) and function call such as $getFirstRow\,(WQ)$, $getPlatformId\,(R_i)$, $getLastScheduleSession\left(RQ, P^i_{id}\right)$ and $getExpectedEx-ecutionTime\,(R_i)$ (line 1-4) which also execute in O(1) time, since they are mostly read operation on database table. Therefore, overall time complexity of the algorithm is O(1). The space complexity of the algorithm is determined by the size of database table. The database table is of size $n \times k$ where n is the number of previously stored sessions(row) and k is fixed number of attributes(column). Therefore, space complexity of the algorithm is O(nk) which eventually comes down to O(n) as k is constant.

Grant of Privileges : The attribute privileged $priv^i$ (type : Boolean) in wait_queue represents the permission to execute

Algorithm 2 StartClock-1

Input: WQ=$\{R_1, R_2, R_3 \ldots\ldots R_n\}$,
RQ=$\{S_1, S_2, S_3 \ldots\ldots S_n\}$
Output: $priv^i$, $stat^i$
Each $R_i = < u^i_{id}, j^i_{id}, eet^i, P^i_{id}, stat^i, priv^i >$
Each $S_i = < d^i_s, d^i_f, t^i_s, t^i_f, P^i_{id}, u^i_{id}, j^i_{id}, stat^i >$

```
1:  while true do
2:      $S_i \leftarrow getScheduleSession\ (RQ)$
3:      if ($S_i == true$) then
4:          $d^i_s \leftarrow getStartDate\ (S_i)$
5:          $t^i_s \leftarrow getStartTime\ (S_i)$
6:      end if
7:      while ($S_i == true$) do
8:          $d_c \leftarrow getCurrentDate\ (clock)$
9:          $t_c \leftarrow getCurrentTime\ (clock)$
10:         if $(d^i_s == d_c\ \&\ t^i_s == t_c)$ then
11:             $priv^i == TRUE$
12:             $stat^i == WAITING$
13:             $updatePriviledge\ (priv^i, WQ)$
14:             $updateJobStatus\ (stat^i, RQ)$
15:             break
16:         end if
17:     end while
18: end while
```

the job *i* on the experimental board and is set to a default value *False*. The value gets updated to *True* dynamically when the current date and time matches with the schedule start date and time. The technique to set the privilege of a job is described in Algorithm 2.

Algorithm 2 takes two input sets the wait table *WQ* and *RQ*, the run table. The algorithm begins with extraction of a scheduled session from the run_queue *RQ* (line-2) followed by extraction of scheduled start date and time of the extracted session, if existence of session S_i is *True* (line 3-5) i.e. the session exists. The current date and time gets initialized at line-8 & 9 respectively. The inner loop between line 7-17 continuously compares the current date and time of a job to the scheduled start date and time(line-10). If *True*, the attribute $priv^i$ gets initialized with value *True* (line-11) while attribute $stat^i$ gets initialized with value *WAITING* (line-12). The attribute $stat^i$ of run_queue *RQ* defines the status of the job and holds three type of information such as *WAITING*, *RUNNING* & *COMPLETED*. The *WAITING* state denotes that a job has the permission to execute but the clients have not started it as yet. The *RUNNING* state denotes that the job is currently running and hasn't completed while the state named *COMPLETED* denotes that the job has completed its execution. The privilege $priv^i$ & and state of the job $stat^i$ get updated through line-13 & 14 in *WQ* and *RQ* respectively. Once the privilege and status is updated, the inner loop breaks(line-15) and allow outer loop to reiterate to enable the permission of next scheduled job. The outer loop iterates forever to enable all future scheduled sessions (line 1).

The status of the job gets updated from *WAITING* to *RUNNING* automatically whenever a client starts the experiment. Once a task get completed, its privilege is lowered(the value of $priv^i$ turned back to *False*) and the value of $stat^i$ gets updated to *COMPLETED* from *RUNNING*. This is ensured by the algorithm *StartClock-2*. We do not provide its details since it is similar to *StartClock-1* except lines 11 and 12. A third algorithm named *StartClock-3* is also similar to the *StartClock-1* except line 11 and 12 where $stat^i$ gets updated to *COMPLETED* from *WAITING* and the privilege gets lowered from *True* to *False*. The algorithm *StartClock-3* is required to prevent session overlapping which may occur if a client executes the experiment at a later date and time which may be scheduled for some other client.

Algorithm 2 runs for infinite time in order to enable execution permission and change job status of future allotted sessions. Therefore, time complexity of the algorithm is not expressible. The space complexity of the algorithm mainly depends upon function call *getScheduleSession* (*RQ*) (line-2). The algorithm extracts one session (S_i) in each iteration from a view of large session (virtual table) of size O(nk) where k is fixed number of attributes and n is the number of previously scheduled sessions, therefore space complexity of the function *getScheduleSession* (*RQ*) is O(nk) that eventually comes down to O(n) as k is constant.

C. VISUALIZING THE UTILIZATION OF THE PLATFORM

As different applications get executed on different computing nodes, it is necessary to monitor the utilization of each platform so that new client requests can be scheduled effectively. With this in view, we provide two dashboards, one for the client and the other for the admin. These dashboards help in visualizing the status of the requested session and utilization of the platform. The client can access the dashboard by the link given in the profile page. The dashboard of the client provides the schedule status of the requested session. The client can view the schedule start date, start time and finish time of the experiment. Similarly the admin can enquire about how many session requests have arrived, how many got scheduled and which platform is used for the experiment and list of the idle platforms.

D. TOOL SUPPORT

Our proposed *SLePaaS* platform provides three tools for three different purposes. The clients subscribed to the platform can make use of these tools for their own purpose. The tools are :

1) *Nirikshak*[4]- *The Profiling Tool* : This tool allows generation of thermal profile of applications during execution on a certain computing node (target hardware platform). The generated thermal profile not only provides an estimate of the range of different temperatures during execution but also serves as input to the other two tools. In addition to generating thermal profile

[4]Hindi word meaning *one who monitors.*

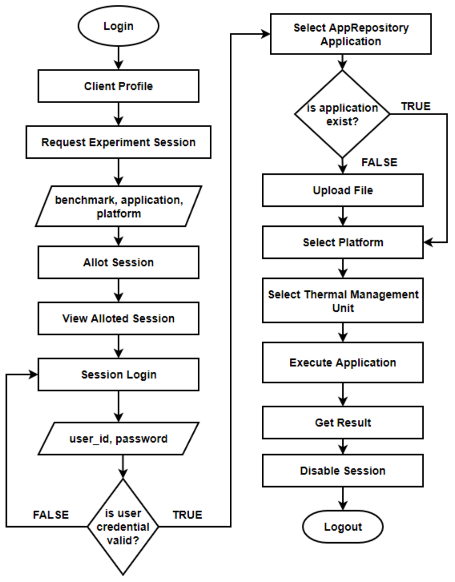

FIGURE 5. Operational flow of Nirikshak.

of applications, *Nirikshak* can be used for generating performance and power profile of applications as well.

2) *Sahayak*[5] - *The Thermal Predictor* : This tool helps users to predict the average temperature of the application prior to execution of the application. The prior knowledge of average temperature helps the task schedulers to select the suitable processor for the application depending on the thermal profile.

3) *Nirdharak*[6]- *The Platform Predictor* : This tool aids users in deciding the suitable platform of execution for OpenCL applications such that the expected thermal characteristics are met.

In the next three sections we discuss in details each of the three tools.

V. Nirikshak : THE PROFILING TOOL

This tool facilitates subscribed users of the *SLePaaS* to execute their application code on three different computing nodes (Hardware Platforms) and obtain the performance, power and thermal profile of their applications. The operational flow of the *Nirikshak* tool has been illustrated in Figure 5.

Assuming a client has already subscribed to the *SLePaaS*, he/she logs in and moves to the *Request for Experiment Session* stage. In this step, the client requests for a separate experiment session by providing the **name of the benchmark**, **application** and **platform** he/she wishes to run. The next step is *Allot Session* where an **experiment session id** is allotted against the client request, a query is performed on the session database and new session details are provided to the client. The details include **session start date, start time and finish time**. The client can then view the allotted experiment session details in the *View Allotted Session* step.

[5]Hindi word meaning *one who assists*.

[6]Hindi word meaning *one who decides*.

The start experiment feature (in client profile) of the tool is disabled by default for all users and is enabled by the resource manager. This ensures that only one user of each platform is allowed to execute an application on a computing node. The resource manager enables the experiment session exactly on the same start date and time as scheduled by the allotted session. The next step is *Session-login* where the user has to login on the scheduled start date and time. If the session start date and time matches with current date and time then such user can access the computing nodes (hardware platforms i.e. the experimental boards) and run their application on them. The user session remains valid for three hours after scheduled start time.

In addition, an App Repository is maintained that provides users features of pre-compiled applications of the repository on request. The applications maintained within the App repository includes applications from different benchmark suites such as *Polybench-ACC-Master*, *Parboil* & *Rodinia-3.1*. The tool also provides facility to add applications to the existing repository.

Once the user uploads the application, next he/she needs to select the *computing node (experimental platform)* and one of the thermal management units. The next step is the *Execute Application* which execute the application on the selected platform. At the end of the execution, results are obtained and displayed on request. After the execution is over, the user session is disabled and the user profile is logged out.

A. THERMAL MANAGEMENT UNIT SUPPORTED BY NIRIKSHAK

The prediction of peak temperature of the application is non trivial in real time and inappropriate thermal management techniques can lead to several thermal issues. Therefore, efficient dynamic thermal management techniques(DTM) are required to prevent peak thermal violation. In order to avoid peak thermal violation we propose two DTM technique based on dynamic voltage frequency scaling(DVFS) and task migration(TM) in addition to the baseline approach.

Baseline Approach: The baseline approach executes applications on default system configuration. The baseline approach does not include a thermal management at user level. The default scheduler used in baseline approach is *Complete Fair Scheduler(CFS)* that shares the CPU time equally among all processes. CFS scheduler is provided by most of the latest linux operating systems.

Customized Frequency Governor(CFG): The CFG is our proposed thermal management unit(TMU) developed at user level with the help of system level utilities. It uses *userspace* frequency governor to scale the operating frequency dynamically. The mechanism to regulate the core temperature is illustrated in Algorithm 3.

Algorithm 3 takes two threshold temperature limit(T^{θ}_{high} & T^{θ}_{low}), max and min frequency(f^{ci}_{max} & f^{ci}_{min}) of the operating core and application name as input. The high threshold temperature limit(T^{θ}_{high}) is used to guard the critical temperature of the processor, similar to the techniques used

Algorithm 3 Customized Frequency Governor

```
Input: $App^i$, $T^\theta_{high}$, $T^\theta_{low}$, $f^{c_i}_{max}$, $f^{c_i}_{min}$
1: $P^i_{id} \leftarrow getProcessId(App^i)$
2: $c_i \leftarrow getMappedCore(P^i_{id})$
3: while ($P^i_{id} > 0$) do
4:     $f^{c_i}_{cur} \leftarrow getClockFreq(c_i)$
5:     $T^{c_i}_{cur} \leftarrow getCoreTemperature(c_i)$
6:     if ($T^{c_i}_{cur} > T^\theta_{high} \& f^{c_i}_{cur} > f^{c_i}_{min}$) then
7:         $f^{c_i}_{cur} = f^{c_i}_{cur} - \Delta f$
8:         $f^{c_i}_{op} = f^{c_i}_{cur}$ // Initializing max operating frequency
9:     else if ($T^{c_i}_{cur} < T^\theta_{low} \& f^{c_i}_{cur} < f^{c_i}_{max}$) then
10:        $f^{c_i}_{cur} = f^{c_i}_{cur} + \Delta f$
11:        $f^{c_i}_{op} = f^{c_i}_{cur}$ // Initializing max operating frequency
12:    end if
13:    $P^i_{id} \leftarrow getProcessId(App^i)$
14: end while
```

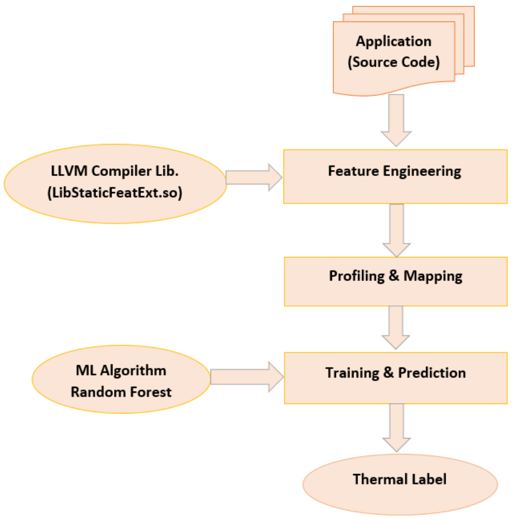
Application (Source Code)
LLVM Compiler Lib. (LibStaticFeatExt.so)
Feature Engineering
Profiling & Mapping
ML Algorithm Random Forest
Training & Prediction
Thermal Label

FIGURE 6. Work flow of *Sahayak*.

in [4] to guard the thermal design power(TDP). The low threshold temperature limit(T^θ_{low}) is used to scale up the core frequency and to enhance the performance when temperature of the processing core become normal. The *max* and *min* frequency defines the operating frequency range of the processing core. The algorithm starts with initialization of process identifier (P^i_{id}) and initial core mapped of the application(line 1-2). The while loop continues until process terminates(line 3-14). In each iteration current frequency($f^{c_i}_{cur}$) and current temperature($T^{c_i}_{cur}$) of the processing core is obtained through system level utilities(line 4-5). The current temperature($T^{c_i}_{cur}$) is compared with high threshold limit of the processing core(T^θ_{high})(line-6). If current temperature($T^{c_i}_{cur}$) is greater than high threshold temperature limit(T^θ_{high}) and current frequency($f^{c_i}_{cur}$) is greater than min frequency of the core($f^{c_i}_{min}$) then operating frequency is scaled down by Δf where Δf is the incremental frequency difference between two frequency label(line 7-8). Similarly, if current temperature($T^{c_i}_{cur}$) is lower than low threshold temperature limit(T^θ_{low}) and current frequency($f^{c_i}_{cur}$) is lower than max frequency of the core($f^{c_i}_{max}$) then operating frequency is scaled up by Δf to increase the performance of the system(line 10-11). At the end of iteration, process identifier is again initialized(line 13) and this iteration continue until process terminate(line 3).

The time complexity of Algorithm 3 mainly depends on iterative loop. The while loop continues until process id P^i_{id} of i^{th} application exists. The process id P^i_{id} remains valid until process terminates. Therefore, the termination condition of while loop depends upon the execution time of application. Let us assume an application has m seconds of execution time and polling interval of P^i_{id} of a process i is k seconds. Therefore, the time complexity of the loop can be expressed as O(m/k). Since k is constant, therefore time complexity of while loop comes down to O(n). The other operation involves initialization of variable that can be executed in O(1) time. Therefore, overall time complexity of the algorithm is O(n). Since the algorithm uses a fixed number of variables to hold temporary values, therefore space complexity of the algorithm is O(1).

Thermal Aware Scheduler(TAS): The second DTM technique proposed at user level is TAS which uses a similar algorithms as the Customized Frequency Governor except for the DVFS technique (line 7-8 & line 10-11) where it uses task migration instead of frequency scaling. It migrates a task from high processing core to low power core when current temperature of the core is greater than high threshold temperature limit while it migrates a task from low power core to high performing core when current core temperature is lower than low threshold temperature limit of the core.

VI. Sahayak : THE THERMAL PREDICTOR

Literature suggests that over the years a number of software based techniques for thermal management of embedded devices have been proposed [21], [39], [49]. Majority of them involve a task scheduler that performs thermal aware task allocation to cores. A prior knowledge about the temperature profile of an application can help the task schedulers to take more informed decision while allocating tasks to cores. Our proposed tool *Sahayak* is able to predict the average temperature of the application prior to the execution of the application. The prior information about average temperature of the application will help the task scheduler to allocate tasks to cores judiciously at the same time will foster research on other thermal management techniques.

A. WORKING PRINCIPLE OF SAHAYAK

The *Sahayak* tool works in three phases as illustrated in Figure 6.

The first phase is the **Feature Engineering phase**, where the static and dynamic features of the application are extracted. Static features of an application are the attributes that represent the execution flow of the application. They include information about *number of basic blocks, number of edges in control flow graph(CFG) of the application, critical edges in the CFG, number of direct calls in a method, number of conditional branches in CFG, number of local, static and external variables in a method* etc. The complete list of all static features of an application has been illustrated in Table-1. These features uniquely represent an application. We extract the static features of the application using LLVM front-end *Clang* making use of LLVM compiler library (*LibStaticFeatExt.so*) to analyse the source code and collect the features. The list of all static features generally associated with programs has been illustrated in Table-1.

Dynamic Features of an application are the attributes which state about the actual resources used during execution of the application such as *cpu-cycle*, *bus-cycle*, *cache-references*, *branch-instruction* etc. The complete list of dynamic feature consider in this experiment are illustrated in Table 2. The dynamic features of the application are extracted by the hardware event monitoring tool *perf* [2] during the execution of the application.

The next phase is the **Profile and Mapping phase** where the applications are executed on the hardware platforms and the corresponding thermal variations are recorded. The temperature values are recorded using the on-chip thermal sensors available on the target hardware platforms. Once an application completes its execution on a target platform, the average range of temperature is obtained from its temperature profile. This average temperature is the thermal label that is associated with the application. Table 3 highlights the thermal labels obtained for the different set of applications.

Once thermal labels have been determined, the thermal label are mapped to the static and dynamic features of the applications extracted during the *Feature Engineering phase*. Thus, the output of the **Profiling and Mapping** Phase yields a vector having features appended with the thermal label. It looks something as :

Static Feature 1, Static Feature 2, ...,Dynamic Feature 1, Dynamic Feature 2, ...: label where label is the average temperature range obtained while executing the application on CPU and GPU.

The final phase involves the **Training and Prediction phase** where the labelled data set (feature vector with the temperature label) is fed to a machine learning model. We tried with a number of ML models such as Naive Bayesian, K-Nearest Neighbour(KNN), Support Vector Machine(SVM) and Random Forest (RF) on the prepared data-set. Among them, Random Forest algorithm provided the highest accuracy of 66.6 % in predicting the thermal label of an application when only static feature of the application were considered. The accuracy improved on considering dynamic feature along with static features. The accuracy of the ML model in Random Forest (RF) algorithm increased to 75% when both static and dynamic features of the application were considered. The predictive model used in *Sahayak* was validated by testing it on a varied applications chosen from different benchmark suits such as *Parboil*, *Rodinia-3.1* and *Polybench-ACC-Master*. 75% of the data set generated from

TABLE 1. Static feature considered in this experiment.

sft1	Number of basic blocks in the method
sft2	Number of basic blocks with a single successor
sft3	Number of basic blocks with two successors
sft4	Number of basic blocks with more then two successors
sft5	Number of basic blocks with a single predecessor
sft6	Number of basic blocks with two predecessors
sft7	Number of basic blocks with more then two predecessors
sft8	Number of basic blocks with a single predecessor and a single successor
sft9	Number of basic blocks with a single predecessor and two successors
sft10	Number of basic blocks with a two predecessors and one successor
sft11	Number of basic blocks with two successors and two predecessors
sft12	Number of basic blocks with more then two successors and more then two predecessors
sft13	Number of basic blocks with number of instructions less then 15
sft14	Number of basic blocks with number of instructions in the interval [15, 500]
sft15	Number of basic blocks with number of instructions greater then 500
sft16	Number of edges in the control flow graph
sft17	Number of critical edges in the control flow graph
sft18	Number of abnormal edges in the control flow graph
sft19	Number of direct calls in the method
sft20	Number of conditional branches in the method
sft21	Number of assignment instructions in the method
sft22	Number of binary integer operations in the method
sft23	Number of binary floating point operations in the method
sft24	Number of instructions in the method
sft25	Average of number of instructions in basic blocks
sft26	Average of number of phi-nodes at the beginning of a basic block
sft27	Average of arguments for a phi-node
sft28	Number of basic blocks with no phi nodes
sft29	Number of basic blocks with phi nodes in the interval [0, 3]
sft30	Number of basic blocks with more then 3 phi nodes
sft31	Number of basic block where total number of arguments for all phi-nodes is in greater then 5
sft32	Number of basic block where total number of arguments for all phi-nodes is in the interval [1, 5]
sft33	Number of switch instructions in the method
sft34	Number of unary operations in the method
sft35	Number of instruction that do pointer arithmetic in the method
sft36	Number of indirect references via pointers ("*" in C)
sft37	Number of times the address of a variables is taken ("&" in C)
sft38	Number of times the address of a function is taken ("&" in C)
sft39	Number of indirect calls (i.e. done via pointers) in the method
sft40	Number of assignment instructions with the left operand an integer constant in the method
sft41	Number of binary operations with one of the operands an integer constant in the method
sft42	Number of calls with pointers as arguments
sft43	Number of calls with the number of arguments is greater then 4
sft44	Number of calls that return a pointer
sft45	Number of calls that return an integer
sft46	Number of occurrences of integer constant zero
sft47	Number of occurrences of 32-bit integer constants
sft48	Number of occurrences of integer constant one
sft49	Number of occurrences of 64-bit integer constants
sft50	Number of references of a local variables in the method
sft51	Number of references (def/use) of static/extern variables in the method
sft52	Number of local variables referred in the method
sft53	Number of static/extern variables referred in the method
sft54	Number of local variables that are pointers in the method
sft55	Number of static/extern variables that are pointers in the method

TABLE 2. Dynamic feature considered in this experiment.

dyft1	cpu-cycles:u
dyft2	instructions:u
dyft3	bus-cycles:u
dyft4	cache-references:u
dyft5	cache-misses:u
dyft6	branch-instructions:u
dyft7	branch-misses:u
dyft8	L1-dcache-loads:u
dyft9	L1-dcache-load-misses:u
dyft10	L1-dcache-stores:u
dyft11	L1-dcache-store-misses:u
dyft12	L1-icache-loads:u
dyft13	L1-icache-load-misses:u
dyft14	dTLB-load-misses:u
dyft15	iTLB-load-misses:u
dyft16	LLC-loads:u
dyft17	LLC-load-misses:u
dyft18	LLC-stores:u
dyft19	LLC-store-misses:u

TABLE 3. Thermal label for training dataset.

Thermal Label	Application
A : $T_{app} < 70$	atax, bicg, correlation, covariance, gesummv, gramschmidt, heartwall, nn
B : $70 <= T_{app} < 75$	gemm, gemver, jacobi-1D, syrk, nw, particle filter
C : $75 <= T_{app} < 80$	bfs, cutcp, histo, spmv, backprop, hotspot3D, 2mm, 2D_convolution, 3D_convolution, doitgen, gaussian, lud, myocyte

TABLE 4. Application for validation.

Application	Short Description
adi	Alternating Direction Implicit solver
lu	LU decomposition
mvt	Matrix Vector Product and Transpose
syr2k	Symmetric rank-2k operations
sgemm	Dense Matrix Operation
stencil	An iterative jacobi stencil operation on a regular 3D grid
pathfinder	Dynamic programming algorithm to find the shortest path of a 2D grid
3mm	3 Matrix Multiplications
fdt2d	2-D Finite Different Time Domain Kernel

these benchmarks was used for training while 25% was used for testing. The results predicted by the ML model were compared with the average temperature range recorded during execution of the same applications on the actual target hardware platform. The prediction and execution results were compared and have been presented in Table 5.

The prior information about thermal characteristic of an application such as average temperature (thermal label) helps to classify applications as a hot apps and cold apps (when compared with a threshold limit). Such task classification allows the task scheduler to allocate tasks in an intelligent way so that the core temperature remains within an optimal range. For example, for the *ODROID-XU4* platform that contains both high performing cores and low power cores, the task scheduler can allocate hot tasks to low power cores (*LITTLE* cores) while the cold tasks to the high performing cores (*big* cores) [26].

TABLE 5. Validation of predictive ML model.

Application	Predicted Thermal Label	Actual Average Temperature
adi	$A : T_{app} < 70$	68.4
lu	$A : T_{app} < 70$	69.8
mvt	$B : 70 <= T_{app} < 75$	72.9
syr2k	$A : T_{app} < 70$	68.6
sgemm	$C : 75 <= T_{app} < 80$	79.7
stencil	$C : 75 <= T_{app} < 80$	78.2
pathfinder	$C : 75 <= T_{app} < 80$	79.2
3mm	$C : 75 <= T_{app} < 80$	78.9
fdt2d	$C : 75 <= T_{app} < 80$	77.9

TABLE 6. OpenCL BenchMarks application supported by Nirdharak tool.

Benchmark	Aplication
Parboil	bfs, cutcp, histo, sgemm, spmv, stencil
Rodinia-3.1	backprop, hotspot3D
Polybench-ACC-Master	2mm, 3mm, 2DConvolution, 3DConvolution, adi, atax, bicg, correlation, covariance, doitgen, fdt2d, gemm, gemver, gesummv, gramschmidt, jacobi1D, lu, mvt, syr2k, syrk

VII. Nirdharak : PLATFORM PREDICTOR FOR OpenCL APPLICATIONS

OpenCL is a programming framework that allows users to develop applications for multiple platforms. These applications can be developed for Central Processing Units(CPUs), Graphical Processing Units(GPUs), Digital Signal Processors(DSPs) or for Field Programmable Gate Arrays(FPGAs). Presently, the *Nirdharak* tool considers OpenCL app development only for CPUs and GPUs. The GPU bound OpenCL applications contain two parts: the host code and the kernel code. The host code runs on the CPU and sets up the environment for the application run. The host code performs functions such as initializing the platform id, device type, OpenCL context and command queue; creates memory buffer and program object, load and executes the kernel. The kernel code on the other hand gets executed on the GPU and performs the actual computation on the data.

Nirdharak supports a set of OpenCL applications developed from different benchmark suits such as *Parboil*, *Rodinia-3.1* and *Polybench-ACC-Master*. Each application has two variants - a CPU based and the other GPU based. The CPU based variant is targeted to run on a CPU while the GPU based variant is targeted to run on a GPU platform. The two variants of the same application running on different hardware platforms yield different performance profile. Thus, it becomes important for the OpenCL application user to decide whether to execute the CPU variant or the GPU variant in order to achieve better performance. The *Nirdharak* tool helps the user to make such a choice. If an user wishes to execute an OpenCL application on the *ODROID XU4* board provided through the *SLePaaS*, *Nirdharak* is able to predict whether the application should be run on the *Exynos-5422* System-on-Chip(SoC) (the ARM based CPU platform) or on the *Mali-T628* (GPU platform) for better performance.

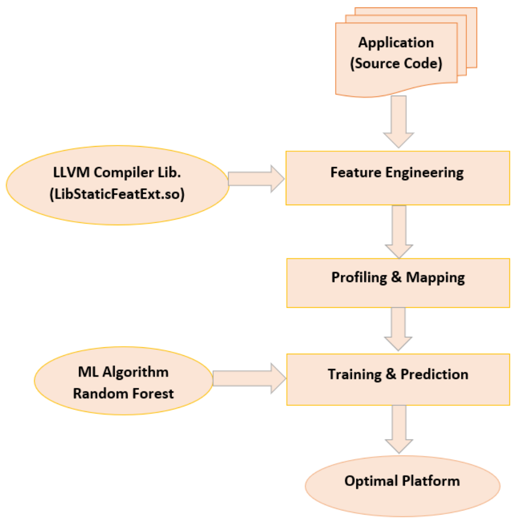
Application (Source Code)
LLVM Compiler Lib. (LibStaticFeatExt.so)
Feature Engineering
Profiling & Mapping
ML Algorithm Random Forest
Training & Prediction
Optimal Platform

FIGURE 7. Operational flow of Nirdharak.

Table 6 provides the complete list of applications supported by the *Nirdharak* tool.

A. WORKING PRINCIPLE OF NIRDHARAK

The *Nirdharak* tool also has three phase working procedure as illustrated in Figure 7. The first phase is the **Feature Engineering phase**, where the static features of the application are extracted following the technique mentioned in Section VI. The next phase is the **PCM (Profiling, Classification and Mapping)** Phase. In this phase, each application is executed twice, the CPU variant of the application is executed on a CPU platform and then the GPU variant of the same application on the GPU platform. We considered the *Exynos-5422* System-on-Chip(SoC) as the experimental platform. The SoC has both the CPU as well as the GPU cores. We make use of both to run the CPU and GPU variants of an application. We executed the CPU variant of an application on the ARM cores (supporting the big.LITTLE architecture) and then the GPU variant was executed on the *Mali-T628* embedded GPU. During execution of the applications, the temperature values were recorded using the thermal sensors available on the CPU and GPU cores.

Once each application has been profiled for temperature on CPU and GPU, the values are compared to decide on the platform that operates at minimum temperature for an application. This classifies an application as a *CPU bound* application - one that operates at minimum temperature when executed on a CPU or *GPU bound* - one that operates at minimum temperature when executed on a GPU. In addition to thermal profile, performance of an application on a CPU and GPU platform can also be used to classify applications as CPU bound or GPU bound. The performance and thermal profile of different benchmark applications from the *Polybench-ACC-Master*, *Parboil* & *Rodinia-3.1* suites are presented in Tables 7 and 8 respectively. In both tables the applications performing better on the CPU platform have

TABLE 7. Performance of OpenCL application on CPU & GPU platform.

Application	Execution Time on CPU(in Seconds)	Execution Time on GPU(in Seconds)	Predicted Platform
bfs	**24.4**	**30.5**	**CPU**
cutcp	726.9	40.1	GPU
histo	**205.3**	**648.8**	**CPU**
sgemm	146.0	14.5	GPU
spmv	16.1	8.9	GPU
stencil	555.3	41.7	GPU
backprop	**27.4**	**32.9**	**CPU**
hotspot3d	**7.1**	**20.6**	**CPU**
2mm	204	4	GPU
3mm	10	1	GPU
2DConvolution	6	4	GPU
3DConvolution	3	2	GPU
2mm	204	4	GPU
adi	1	0.1	GPU
atax	1	0.1	GPU
bicg	1	0.1	GPU
correlation	606	239	GPU
covariance	594	232	GPU
doitgen	6	0.1	GPU
fdt2d	143	24	GPU
gemm	3	0.1	GPU
gemver	3	1	GPU
gesummv	1	1	GPU
gramschmidt	1371	36	GPU
jacobi1D	8	4	GPU
lu	52	11	GPU
mvt	2	1	GPU
syr2k	39	14	GPU
syrk	18	3	GPU

been highlighted by bold font. Both these tables classify the applications into two classes CPU bound and GPU bound. A CPU bound application performs better on the CPU platform whereas a GPU bound application performs better on a GPU platform. After *Classification* comes the *Mapping* phase where each application is mapped to its class depending on the target objective - better performance or lower temperature. In the mapping phase, a label (better performance or minimum temperature platform) is attached to the static features of the application extracted during the *Feature Engineering phase*. Thus, the output of the **PCM (Profiling, Classification and Mapping)** Phase yields a vector having features appended with the output class. It looks something as :

Feature 1, Feature 2, ...: label

The label is either CPU or GPU.

The third and the last phase is the *Training and Decision Phase* where a supervised learning approach is utilized to predict the most suitable platform for an OpenCl application. The output of the **PCM (Profiling, Classification and Mapping)** Phase is used as input data set to the *Random Forest* machine learning algorithm which once trained is able to predict the suitable platform for any OpenCL application. The *Nirdharak* tool when experimented with the benchmark set mentioned in Table 7 was found to predict the suitable platform for an application with 80% accuracy. In order to validate the prediction of the tool, we chose to execute all applications used in the test set of the ML model on both platforms and then compared with the results predicted by *Nirdharak*.

TABLE 8. Thermal profile of OpenCL application on CPU & GPU platform.

Application	Compiled for CPU		Compiled for GPU	
	CPU Avg.Temp.	GPU Avg.Temp.	CPU Avg.Temp	GPU Avg.Temp
bfs	**79.3**	**65.5**	**79.4**	**65.5**
cutcp	**79.4**	**65.7**	**79.9**	**72**
histo	**79.6**	**65.9**	**79.8**	**67.8**
sgemm	79.7	66.2	78.3	64.6
spmv	79.4	65.5	77.8	64
stencil	**78.2**	**66.6**	**79.5**	**67.9**
backprop	78.8	65.3	78.6	64.7
hotspot3d	**78.2**	**64**	**78.5**	**64.7**
2mm	78.8	66	69.8	64
3mm	78.9	65.5	66	60.5
2DConvolution	75.1	62.7	72.1	60.6
3DConvolution	75.3	62	68.5	57.6
adi	68.4	57.6	65	55.6
atax	69.1	58.8	67.3	58.6
bicg	69.8	59.1	66.8	58
correlation	67.7	58.9	64.4	61.4
covariance	66.8	58.4	62.4	60
doitgen	77.9	64.3	64.4	57.9
fdt2d	77.9	68.8	62.9	62
gemm	73.3	60.2	59.6	54
gemver	73.1	60.7	65.8	57.5
gesummv	68.7	58	68.4	58.8
gramschmidt	63.9	57.5	61.7	59.8
jacobi1D	73.1	60.6	71.8	59.6
lu	69.8	60.7	60.3	59.1
mvt	72.9	60.5	66.7	57.5
syr2k	68.6	59.8	63.8	62.5
syrk	71.5	62.3	62.5	61

VIII. EVALUATION OF SLePaaS

The *SLePaaS* prototype is available for use. One may visit https://systemslab.iiita.ac.in/imprint/ to know about the details. In order to use the platform and the tools, each user has to register by filling up an on-line form.

Proof-of-Concept Implementation: As a proof-of-concept of the prototype, we present the execution of the *pathfinder* application from the *Rodinia* benchmark suite on the *Odroid-XU4* board making use of the *SLePaaS* platform. The *Odroid-XU4* is an **ARM based embedded platform** having *Exynos 5422 System-on-Chip(SoCs)* with advanced *Mali T628 Graphics Processing Unit(GPU)*. The SoCs contains four high performance *big* cores and four low power *LITTLE* cores. However, only the *big* cores are equipped with temperature sensors. Therefore, to record the thermal profile of the application all the host code of the application has to be executed on the *big* cores. The legends *Core 4,5,6,7* used in Figure 8, 9 and 10 represent the *big* cores of the ODROID XU4 board. The GPU is also equipped with temperature sensors that are utilized to capture the thermal readings during execution of the the *GPU kernel code* of the application. In the current prototype version, the tools *Sahayak* and *Nirdharak* have been developed for the *Odroid-XU4* embedded platform. However, we plan to extend the architectural support in course of time. The experiments were performed on default frequency governor *Performance*(Baseline TMU) with maximum clock frequency of 2000MHz for the *big* cores and 1400MHz for the *LITTLE* cores. The ambient temperature was recorded at 21^oC.

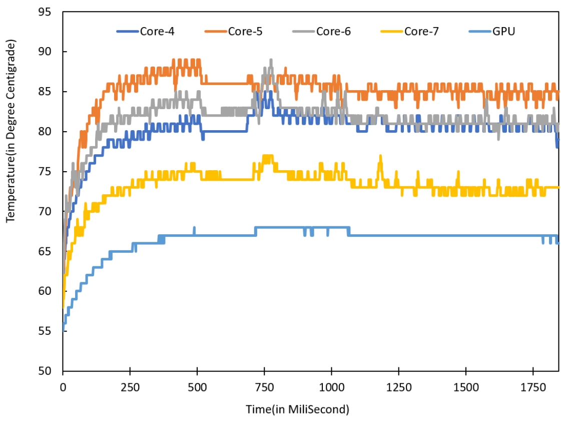

FIGURE 8. Thermal profile of pathfinder application.

The process begins with a request for an experiment session which gets scheduled on the allotted date and time. This is ensured by the *StartClock-1()* module of resource manager utility. Once a session is allotted, user uploads the source code or selects the application from *AppRepository*. Then he/she selects the platform, thermal management unit and executes it using the interface provided through HTML page.

Next we invoke the *Sahayak* tool to predict the thermal label of the *pathfinder* application deciding on *Odroid-XU4* as the computing node running with maximum frequency. *Sahayak* utilizes feature vectors of the application and ML models to find the average temperature range (thermal label) of the application prior to its execution on the computing node. The predicted average temperature of the *pathfinder* application as computed by *Training and Decision* Phase of *Sahayak* lies in the temperature range of 75^oC-80^oC(Thermal Label-C). The predicted thermal label of the application is then validated by comparing it with the experimental readings obtained after execution of the application on the actual hardware platform. Figure 8 displays the thermal gradient of the application when executed on *Odroid-XU4*. The average temperature of the application is observed as 79.2^oC. The experimental readings lie in the temperature range as predicted by *Training and Decision* Phase of *Sahayak*, ensuring that the tool *Sahayak* is working properly. The prior knowledge of average temperature of the application obtained through *Sahayak* helps the user to select suitable processor or clock frequency for an embedded application.

The *Nirdharak* tool utilizes the static feature vector set and machine learning models to predict the optimal platform for executing an OpenCL application given a target optimization objective. For the *pathfinder* application considered for validation, if the objective is to optimize performance, *Nirdharak* predicted that executing the CPU variant of the *pathfinder* application on the ARM *big* or *LITTLE* cores would be better off in terms of performance rather than executing the GPU variant of the same on the Mali GPU of the *ODROID*

TABLE 9. Performance counter stat of CPU variant of pathfinder application.

Hardware Performance Counter	Value
execution time(in seconds)	36.603208730
cpu-cycles:u	64,907,240,897
instructions:u	34,425,054,063
bus-cycles:u	12,986,206,304
cache-references:u	14,285,591,355
cache-misses:u	7,953,073
branch-instructions:u	6,596,128,588
branch-misses:u	10,052,507
L1-dcache-loads:u	9,063,913,713
L1-dcache-load-misses:u	4,180,160
L1-dcache-stores:u	5,263,504,498
L1-dcache-store-misses:u	3,696,657
L1-icache-loads:u	8,674,158,917
L1-icache-load-misses:u	139,143,866
dTLB-load-misses:u	797,495
iTLB-load-misses:u	626,662
LLC-loads:u	164,756,971
LLC-load-misses:u	3,426,564
LLC-stores:u	7,753,260
LLC-store-misses:u	2,377,055

TABLE 10. Performance counter stat of GPU variant of pathfinder application.

Hardware Performance Counter	Value
execution time(in seconds)	53.349068530
cpu-cycles:u	66,224,915,185
instructions:u	35,059,555,369
bus-cycles:u	13,257,788,983
cache-references:u	14,624,440,855
cache-misses:u	12,054,466
branch-instructions:u	6,730,890,569
branch-misses:u	11,932,275
L1-dcache-loads:u	9,278,686,863
L1-dcache-load-misses:u	7,579,883
L1-dcache-stores:u	5,411,828,999
L1-dcache-store-misses:u	4,239,571
L1-icache-loads:u	8,919,531,555
L1-icache-load-misses:u	140,598,025
dTLB-load-misses:u	1,334,009
iTLB-load-misses:u	1,251,611
LLC-loads:u	176,127,580
LLC-load-misses:u	7,073,703
LLC-stores:u	14,306,385
LLC-store-misses:u	3,304,299

XU4 board. Later the prediction was validated by executing both the CPU and GPU variant of the application in separate sessions on the CPU and GPU cores respectively. The performance profile of CPU and GPU variants are presented in Table 9 & 10 respectively. The execution time recorded for CPU variant of *pathfinder* is 36.6 seconds while the execution time of GPU variant is recorded as 53.3 seconds. The results clearly show that execution of the CPU variant of the *pathfinder* application on the CPU cores is a better choice than running its GPU variant on the GPU cores. However, the results change if the objective is to optimize temperature.

As predicted by the *Nirdharak* tool, GPU variant of the *pathfinder* application when executed on the GPU cores has a lower peak temperature than CPU variant of *pathfinder* when

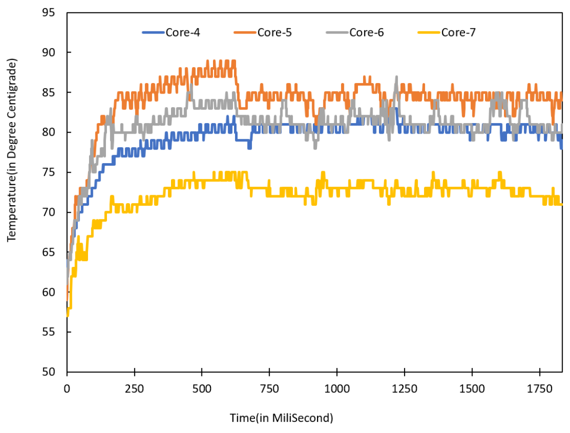

FIGURE 9. Thermal profile of CPU variants of pathfinder application.

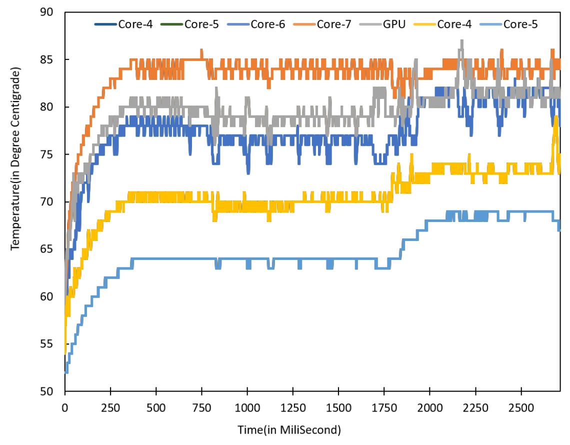

FIGURE 10. Thermal profile of GPU variants of pathfinder application.

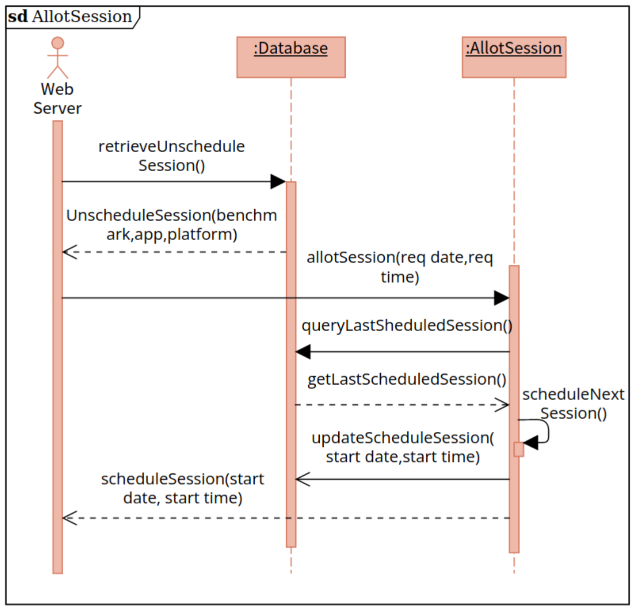
sd AllotSession
Web Server
:Database
:AllotSession
retrieveUnschedule Session()
UnscheduleSession(benchm ark,app,platform)
allotSession(req date,req time)
queryLastSheduledSession()
getLastScheduledSession()
scheduleNext Session()
updateScheduleSession(start date,start time)
scheduleSession(start date, start time)

FIGURE 11. Allot session sequence diagram.

executed on the CPU cores. This is validated by the temperature profiles recorded during execution of the CPU and GPU

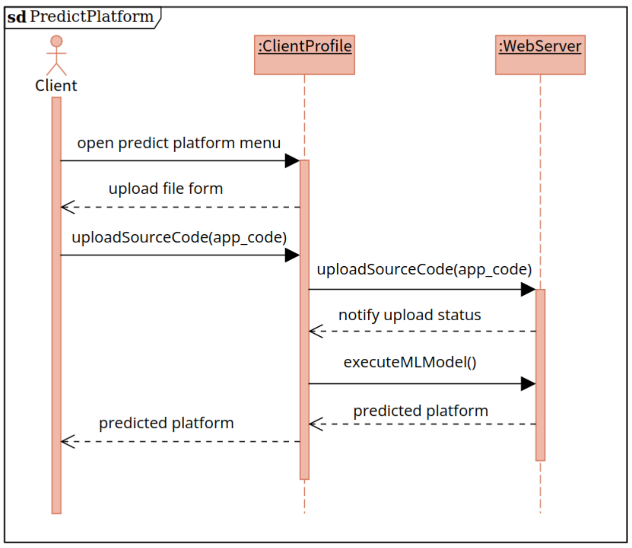

FIGURE 12. Predict platform sequence diagram.

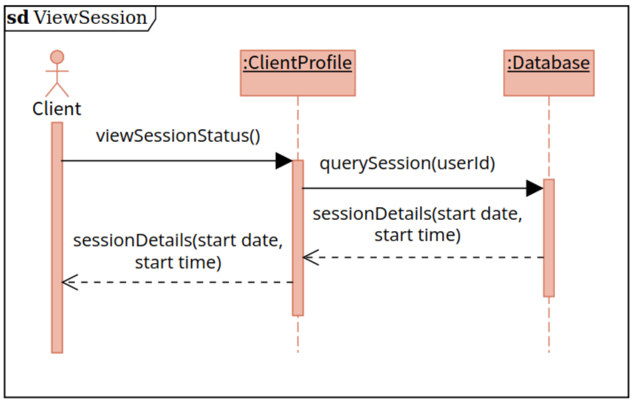

FIGURE 13. View session sequence diagram.

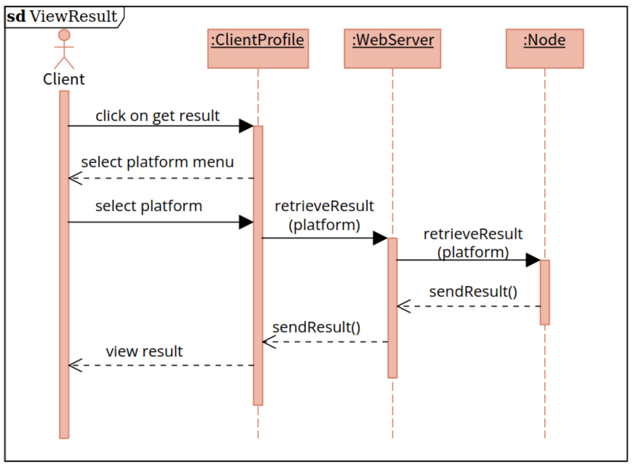

FIGURE 14. View result sequence diagram.

variants of the code on CPU and GPU cores respectively. Figure 9 & Figure 10 present the thermal profile of CPU and GPU variants of *pathfinder* application when executed

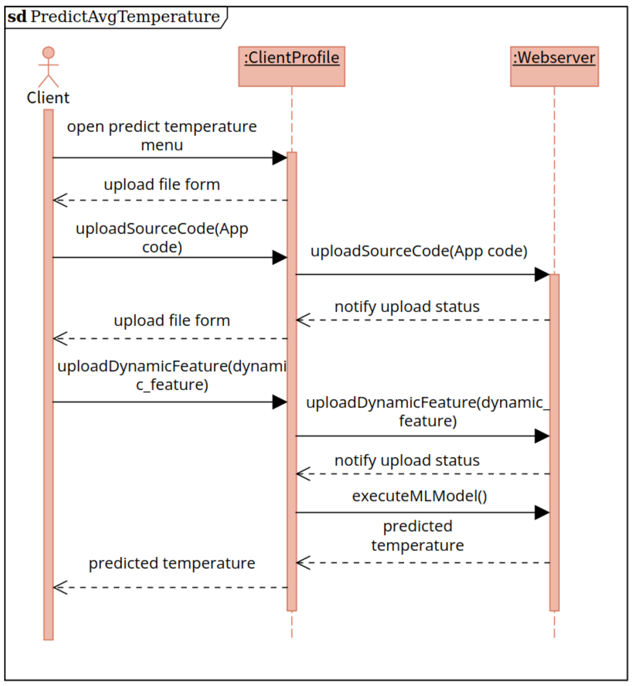

FIGURE 15. Predict average temperature sequence diagram.

on their respective cores. The average temperature of CPU variant is recorded as 79.2^oC while average temperature of GPU variant is recorded as 79^oC. The results clearly validate the claim by *Nirdharak* tool that if the operating platform has to be decided based on operating temperature, executing the GPU variant of the *pathfinder* application on the GPU core is a better option than running the CPU variant on the CPU cores.

IX. CONCLUSION AND FUTURE WORK

In this paper, we present to the embedded research community an embedded platform as a service facility *SLePaaS* which would facilitate research on thermal management of embedded systems. We present the basic architecture and working of this embedded platform as a service facility and discuss in details each of its components. In addition to providing users the remote access of hardware platforms, *SLePaaS* also provides three tools which help users in obtaining performance, power and thermal profile of applications, help users to choose the suitable processor based on predicted average temperature and to decide the platform (CPU or GPU) that would be suitable for execution of applications considering a certain target objective. The tools in the developed prototype currently support three hardware platforms namely *ODROID XU4*, *Jetson TX1* and *Jetson Nano*. However, in future, we plan to add support to more architectures. In the paper we explain the working of each tool and validate their working by comparing their results with the results obtained when same applications are executed on the actual hardware.

A prototype implementation of the platform-as-a service facility and the three tools is available at https://systemslab.

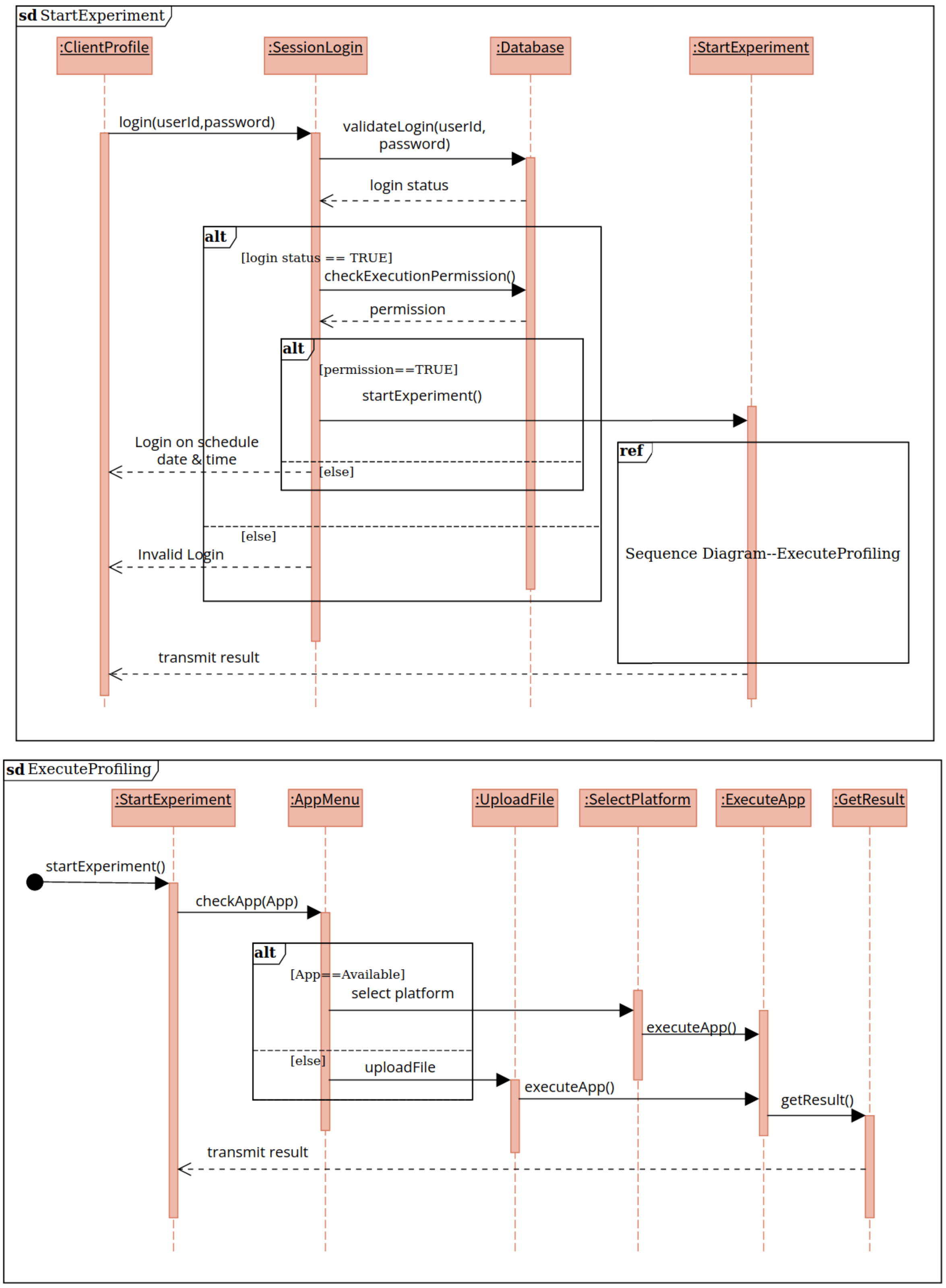

FIGURE 16. Sequence diagram of start experiment and execute profiling.

iiita.ac.in/imprint/. A proof-of-concept implementation of the prototype is presented by making use of the *SLePaaS* facility and validating the working of the *pathfinder* application on each of the tools.

APPENDIX A: SEQUENCE DIAGRAM

See Figures 11–16.

ACKNOWLEDGMENT

Project URL: https://imprint-india.org/knowledge-portal-project-pages-list, SLePaaS: https://systemslab.iiita.ac.in/imprint/.

REFERENCES

[1] *Gem5 Architectural Simulator*. Accessed: Aug. 1, 2022. [Online]. Available: https://www.gem5.org/

[2] *Perf Tool*. Accessed: Aug. 1, 2022. [Online]. Available: https://perf.wiki.kernel.org/index.php/Tutorial

[3] J. A. Ambrose, ''HEATSMART: An interactive application aware thermal management framework for MPSoC embedded systems,'' in *Proc. 6th Int. Conf. Ind. Inf. Syst.*, Aug. 2011, pp. 54–59.

[4] M. Ansari, M. Pasandideh, J. Saber-Latibari, and A. Ejlali, ''Meeting thermal safe power in fault-tolerant heterogeneous embedded systems,'' *IEEE Embedded Syst. Lett.*, vol. 12, no. 1, pp. 29–32, Mar. 2020.

[5] M. Ansari, S. Safari, S. Yari-Karin, P. Gohari-Nazari, H. Khdr, M. Shafique, J. Henkel, and A. Ejlali, ''Thermal-aware standby-sparing technique on heterogeneous real-time embedded systems,'' *IEEE Trans. Emerg. Topics Comput.*, early access, Oct. 20, 2021, doi: 10.1109/TETC.2021.3120084.

[6] A. Banerjee, L. K. Chong, S. Chattopadhyay, and A. Roychoudhury, ''Detecting energy bugs and hotspots in mobile apps,'' in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE)*. New York, NY, USA: Association for Computing Machinery, Nov. 2014, p. 588, doi: 10.1145/2635868.2635871.

[7] D. Brooks, V. Tiwari, and M. Martonosi, ''Wattch: A framework for architectural-level power analysis and optimizations,'' *ACM SIGARCH Comput. Archit. News*, vol. 28, no. 2, pp. 83–94, 2000.

[8] A. Butko, F. Bruguier, A. Gamatié, G. Sassatelli, D. Novo, L. Torres, and M. Robert, ''Full-system simulation of big.LITTLE multicore architecture for performance and energy exploration,'' in *Proc. 10th IEEE Int. Symp. Embedded Multicore/Many-Core Syst. Chip, (MCSOC)* Feb. 2016, pp. 201–208.

[9] D. Castro, P. Romano, A. Ilic, and A. M. Khan, ''HeTM: Transactional memory for heterogeneous systems,'' in *Proc. 28th Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, Sep. 2019, pp. 232–244.

[10] T.-H. Chien and R.-G. Chang, ''A thermal-aware scheduling for multicore architectures,'' *J. Syst. Archit.*, vol. 62, pp. 54–62, Jan. 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762115001538

[11] Y.-F. Chung, C.-Y. Lin, and C.-T. King, ''ANEPROF: Energy profiling for Android Java virtual machine and applications,'' in *Proc. IEEE 17th Int. Conf. Parallel Distrib. Syst.*, Dec. 2011, pp. 372–379.

[12] R. Devaraj and A. Sarkar, ''Resource-optimal fault-tolerant scheduler design for task graphs using supervisory control,'' *IEEE Trans. Ind. Informat.*, vol. 17, no. 11, pp. 7325–7337, Nov. 2021.

[13] R. Devaraj, A. Sarkar, and S. Biswas, ''Supervisory control approach and its symbolic computation for power-aware RT scheduling,'' *IEEE Trans. Ind. Informat.*, vol. 15, no. 2, pp. 787–799, Feb. 2019.

[14] S. Dey, E. Z. Guajardo, K. R. Basireddy, X. Wang, A. K. Singh, and K. McDonald-Maier, ''EdgeCoolingMode: An agent based thermal management mechanism for DVFS enabled heterogeneous MPSoCs,'' in *Proc. 32nd Int. Conf. VLSI Design 18th Int. Conf. Embedded Syst. (VLSID)*, Jan. 2019, pp. 19–24.

[15] M. J. Dousti, M. Ghasemi-Gol, M. Nazemi, and M. Pedram, ''ThermTap: An online power analyzer and thermal simulator for Android devices,'' in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, Jul. 2015, pp. 341–346.

[16] X. Gao, D. Liu, D. Liu, H. Wang, and A. Stavrou, ''E-Android: A new energy profiling tool for smartphones,'' in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 492–502.

[17] T. H. Hetherington, M. Lubeznov, D. Shah, and T. M. Aamodt, ''EDGE: Event-driven GPU execution,'' in *Proc. 28th Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, Sep. 2019, pp. 337–353.

[18] A. Hoffman, A. Pathania, P. H. Kindt, S. Chakraborty, and T. Mitra, ''BrezeFlow: Unified debugger for Android CPU power governors and schedulers on edge devices,'' in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.

[19] P.-S. Huang, Q.-C. Chen, C.-W. Huang, and S.-L. Tsao, ''An efficient thermal estimation scheme for microprocessors,'' in *Proc. IEEE 20th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2014, pp. 1–10.

[20] S. Isuwa, S. Dey, A. K. Singh, and K. McDonald-Maier, ''TEEM: Online thermal- and energy-efficiency management on CPU-GPU MPSoCs,'' in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 438–443.

[21] R. Jayaseelan and T. Mitra, ''Temperature aware scheduling for embedded processors,'' in *Proc. 22nd Int. Conf. VLSI Design*, Jan. 2009, pp. 541–546.

[22] O. Kayiran, A. Jog, A. Pattnaik, R. Ausavarungnirun, X. Tang, M. T. Kandemir, G. H. Loh, O. Mutlu, and C. R. Das, ''μc-states: Fine-grained GPU datapath power management,'' in *Proc. Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, 2016, pp. 17–30.

[23] Y. G. Kim, M. Kim, J. M. Kim, and S. W. Chung, ''M-DTM: Migration-based dynamic thermal management for heterogeneous mobile multi-core processors,'' in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2015, pp. 1533–1538.

[24] J. Kong, S. W. Chung, and K. Skadron, ''Recent thermal management techniques for microprocessors,'' *ACM Comput. Surv.*, vol. 44, no. 3, pp. 1–42, Jun. 2012, doi: 10.1145/2187671.2187675.

[25] R. Kumar and B. Ghoshal, ''Classification and workload balancing of multi-threaded application on embedded platforms,'' in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2020, pp. 568–573.

[26] R. Kumar, A. Sachan, A. Gogoi, and B. Ghoshal, ''Application phase behavior-guided thermal management of embedded platforms,'' *IEEE Embedded Syst. Lett.*, vol. 12, no. 4, pp. 121–124, Dec. 2020.

[27] S. S. Kumar, A. Zjajo, and R. Van Leuken, ''Ctherm: An integrated framework for thermal-functional co-simulation of systems-on-chip,'' in *Proc. 23rd Euromicro Int. Conf. Parallel, Distrib., Netw.-Based Process.*, Mar. 2015, pp. 674–681.

[28] O. Kwon, W. Jang, G. Kim, and C.-G. Lee, ''Accurate thermal prediction for NANS (N-app N-screen) services on a smart phone,'' in *Proc. IEEE 13th Int. Symp. Ind. Embedded Syst. (SIES)*, Jun. 2018, pp. 1–10.

[29] E. Le Sueur and G. Heiser, ''Dynamic voltage and frequency scaling: The laws of diminishing returns,'' in *Proc. Int. Conf. Power Aware Comput. Syst.* Berkeley, CA, USA: USENIX Association, 2010, pp. 1–8.

[30] K.-J. Lee and K. Skadron, ''Using performance counters for runtime temperature sensing in high-performance processors,'' in *Proc. 19th IEEE Int. Parallel Distrib. Process. Symp.*, Apr. 2005, pp. 1–8.

[31] Y. Lee, H. S. Chwa, K. G. Shin, and S. Wang, ''Thermal-aware resource management for embedded real-time systems,'' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2857–2868, Nov. 2018.

[32] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, ''McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures,'' in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2009, pp. 469–480.

[33] S. Maity, A. Ghose, S. Dey, and S. Biswas, ''Thermal load-aware adaptive scheduling for heterogeneous platforms,'' in *Proc. 33rd Int. Conf. VLSI Design 19th Int. Conf. Embedded Syst. (VLSID)*, Jan. 2020, pp. 125–130.

[34] B. Ozceylan, B. R. Haverkort, M. De Graaf, and M. E. T. Gerards, ''A generic processor temperature estimation method,'' in *Proc. 25th Int. Workshop Thermal Invest. ICs Syst. (THERMINIC)*, Sep. 2019, pp. 1–6.

[35] A. Pathak, Y. C. Hu, and M. Zhang, ''Where is the energy spent inside my app? Fine grained energy accounting on smartphones with eprof,'' in *Proc. 7th ACM Eur. Conf. Comput. Syst. (EuroSys)*. New York, NY, USA: Association for Computing Machinery, 2012, pp. 29–42, doi: 10.1145/2168836.2168841.

[36] A. Pattnaik, X. Tang, A. Jog, O. Kayiran, A. K. Mishra, M. T. Kandemir, O. Mutlu, and C. R. Das, ''Scheduling techniques for GPU architectures with processing-in-memory capabilities,'' in *Proc. Int. Conf. Parallel Archit. Compilation*, Sep. 2016, pp. 31–44.

[37] A. Sachan and B. Ghoshal, ''Learning based compilation of embedded applications targeting minimal energy consumption,'' *J. Syst. Archit.*, vol. 116, Jun. 2021, Art. no. 102116. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762121000886

[38] S. Safari, H. Khdr, P. Gohari-Nazari, M. Ansari, S. Hessabi, and J. Henkel, ''TherMa-MiCs: Thermal-aware scheduling for fault-tolerant mixed-criticality systems,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 7, pp. 1678–1694, Jul. 2022.

[39] S. Shaik and S. Baskiyar, ''Proactive thermal aware scheduling,'' in *Proc. 8th Int. Green Sustain. Comput. Conf. (IGSC)*, Oct. 2017, pp. 1–6.

[40] S. Sharifi, D. Krishnaswamy, and T. Š. Rosing, "PROMETHEUS: A proactive method for thermal management of heterogeneous MPSoCs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 7, pp. 1110–1123, Jul. 2013.

[41] S. Sharifi, C. Liu, and T. S. Rosing, "Accurate temperature estimation for efficient thermal management," in *Proc. 9th Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2008, pp. 137–142.

[42] N. K. Shukla, R. Pila, and S. Rawat, "Utilization-based power consumption profiling in smartphones," in *Proc. 2nd Int. Conf. Contemp. Comput. Informat. (IC)*, Dec. 2016, pp. 881–886.

[43] F. Sironi, M. Maggio, R. Cattaneo, G. F. Del Nero, D. Sciuto, and M. D. Santambrogio, "ThermOS: System support for dynamic thermal management of chip multi-processors," in *Proc. 22nd Int. Conf. Parallel Archit. Compilation Techn.*, Sep. 2013, pp. 41–50.

[44] M. Stan, R. Zhang, and K. Skadron, "Hotspot 6.0: Validation, acceleration and extension," Univ. Virginia, Charlottesville, VA, USA, Tech. Rep. CS-2015-04, 2015.

[45] H. Sultan and S. R. Sarangi, "VarSim: A fast and accurate variability and leakage aware thermal simulator," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.

[46] Y. Tan and Q. Qiu, "A framework of stochastic power management using hidden Markov model," in *Proc. Conf. Design, Automat. Test Eur.*, Mar. 2008, pp. 92–97.

[47] S. Wang, G. Ananthanarayanan, and T. Mitra, "OPTiC: Optimizing collaborative CPU–GPU computing on mobile devices with thermal constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 3, pp. 393–406, Mar. 2019.

[48] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "AppScope: Application energy metering framework for Android smartphones using kernel activity monitoring," in *Proc. USENIX Annu. Tech. Conf. (USENIX)*, 2012, pp. 387–400.

[49] B. Yun, K. G. Shin, and S. Wang, "Predicting thermal behavior for temperature management in time-critical multicore systems," in *Proc. IEEE 19th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2013, pp. 185–194.

[50] S. Zhang and K. S. Chatha, "Approximation algorithm for the temperature-aware scheduling problem," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 2007, pp. 281–288.

RAKESH KUMAR (Member, IEEE) received the B.Tech. degree from the National Institute of Technology Patna, India, and the M.Tech. degree in software engineering from the Indian Institute of Information Technology Allahabad, India, where he is currently pursuing the Ph.D. degree with the Department of Information Technology. His current research interest includes software-based thermal management for embedded platforms.

AKASH SACHAN received the M.Tech. degree in IT from the Indian Institute of Information Technology Allahabad, Uttar Pradesh, India, where he is currently pursuing the Ph.D. degree with the Department of Information Technology. His research interests include low power embedded systems and compiler for embedded processors.

BIBHAS GHOSHAL received the Ph.D. degree in computer science and engineering from the Indian Institute of Technology Kharagpur, in 2015. He is currently working as an Assistant Professor with the Department of Information Technology, Indian Institute of Information Technology Allahabad, Uttar Pradesh, India. His current research interests include embedded and real time systems, VLSI design and test, the Internet of Things, computer architecture, low power systems, and distributed systems. He is also a PI of the prestigious "IMPRINT INDIA" Project sponsored by the Ministry of Human Resource and Development, Government of India. He is also a PI of the prestigious "Research Constrained AI" Project sponsored by the Ministry of Electronics and Information Technology (MeitY), Government of India.