Received: 9 March 2022 | Revised: 14 September 2022 | Accepted: 24 November 2022

IET Computers & Digital Techniques

DOI: 10.1049/cdt2.12050

The Institution of Engineering and Technology WILEY

ORIGINAL RESEARCH

Machine learning guided thermal management of Open Computing Language applications on CPU-GPU based embedded platforms

Rakesh Kumar | **Bibhas Ghoshal**

Department of Information Technology, Indian Institute of Information Technology-Allahabad, Prayagraj, Uttar Pradesh, India

Correspondence

Rakesh Kumar and Bibhas Ghoshal, Department of Information Technology, Indian Institute of Information Technology-Allahabad, Prayagraj, Uttar Pradesh 211015, India.
Email: pse2015002@iiita.ac.in and bibhas.ghoshal@iiita.ac.in

Funding information

Ministry of Human Resource Development, Grant/Award Number: IMPRINT India Initiative 1 Project Number 7482 (h')

Abstract

As embedded devices start supporting heterogeneous processing cores (Central Processing Unit [CPU]–Graphical Processing Unit [GPU] based cores), performance aware task allocation becomes a major issue. Use of Open Computing Language (OpenCL) applications on both CPU and GPU cores improves performance and resolves the problem. However, it has an adverse effect on the overall power consumption and the operating temperature of the system. Operating both kind of cores within a small form factor at high frequency causes rise in power consumption which in turn leads to increase in processor temperature. The elevated temperature brings about major thermal issues. In this paper, we present our investigation on the role of CPU during execution of GPU specific application and argue against running it at the high frequency. In addition, a machine learning guided mechanism to predict the optimal operating frequency of CPU cores during execution of OpenCL GPU kernels is presented in this study. Our experiments with OpenCL applications on the state of the art *ODROID XU4* embedded platform show that the CPU cores of the experimental board if operated at a frequency proposed by our Machine Learning-based predictive method brings about 12.5°C reduction in processor temperature at 1.06% degradation in performance compared to the baseline frequency (default *performance* frequency governor of the embedded platform).

KEYWORDS

embedded systems, energy conservation, multiprocessing systems, temperature control

1 | INTRODUCTION

In order to process the large amount of data obtained from different sensors at sufficiently high performance, modern day embedded platforms include Graphical Processing Units (GPUs) in addition to the Central Processing Unit (CPU) cores. For applications that require data parallelism, the GPUs with their multiple computation threads tend to be more effective compared to the CPUs, while applications that need to be executed sequentially are executed on the CPU cores. Thus, through effective work load distribution among CPUs and GPUs, the overall system performance is improved. However, handling work load distribution at the application level becomes difficult for programmers.

Khronos Group introduced the *Open Computing Language (OpenCL)* programming framework [1] that enables programmers to write programs for heterogeneous systems which hosts both CPUs and GPUs. The performance of such heterogeneous systems is improved by partitioning the tasks (*kernel*) into several sub-tasks (known as *work-items*) and executing each sub-task parallel to each other on separate processing cores. Literature suggests that a considerable amount of research has been done on finding different techniques for task allocation on different cores. For example, Chi-Keung Luk et al. proposed an adaptive mapping technique *Qilin* that judicially allocates tasks on both CPU and GPU cores considering arbitrary input size [2]. The authors have proposed a linear regression model, based on offline profiling

to predict the execution time of the application. Dominik Grewe et al. proposed machine learning-based compiler model to partition and map the *OpenCL* applications on CPU-GPU processor [3]. The model is developed by training the data-set with static features of the code and corresponding optimal platform. Prasanna Pandit et al. proposed an *OpenCL* runtime framework *FluidiCL* where the kernel gets divided into *work-items* which are then distributed on both CPU-GPU devices [4]. The distribution is performed in a coordinated manner to minimise the data transfer overhead. The authors of Ref. [5] proposed static partitioning technique (using offline power performance characteristic) of tasks into sub-tasks in conjunction with dynamic voltage frequency scaling approach for loading kernels on CPUs and GPUs.

Similar approach of parallel computing on CPUs and GPUs have been proposed in Refs. [6–10]. In all the works referred above, the focus has been to improve the performance or optimise energy consumption. However, along with performance and energy, thermal effects should also be considered since packing CPUs and GPUs in a single chip and allowing them to work concurrently raises the power density of the system which eventually leads to increase in temperature. And as we know, running a system at an elevated temperature brings about a lot of thermal effects refs. [11–13].

However, some of the researchers in refs. [14, 15] have given their importance to satisfy the chip level Thermal Design Power (TDP) or Thermal Safe Power constraints in their proposed energy optimisation techniques. The TDP is the maximum allowed power for core to dissipate the heat without throttling the CPU performance. Addition to this, some of the researchers have proposed proactive thermal management techniques as mentioned in refs. [16–23] to maintain the operating temperature below given threshold temperature limit. Ganapati Bhat et al. [16] proposed predictive power and thermal model-based proactive thermal management technique utilising the on-demand frequency governors. In case of thermal violation, they proposed to throttle the processing cores considering minimal impact on performance. Alok Prakash et al. in ref. [17] proposed control-theoretic dynamic thermal management to lower the processing core temperature. The authors have shown that thermal coupling between heterogeneous processors can be mitigated by considering frequency throttling of CPU and GPU in coordinated manner. Samuel Isuwa et al. proposed a thermal-energy aware linear regression model to map the workload of *OpenCL* applications on heterogeneous processors [18]. The model takes user requirements such as average temperature and execution time as an input and predicts the optimal design point (possible combination of processing core and available operating frequency of the devices). Jorg Henkel et al. proposed smart thermal management techniques that exploit the heterogeneity at both chip level and application level [19]. At the chip level, authors considered processing element type, number of processing elements, voltage levels, frequency levels and last level cache while at application level authors have exploited thread level parallelism, instruction level parallelism and power consumption. Srijeeta Maity et al. introduce thermal load aware adapting scheduling for heterogeneous platform [20]. The authors have proposed three thermal management techniques such as task-shifting, frequency tuning and task migration. Task shifting is applied among cores of one tile (processor) when the number of tasks on the cores are low. Frequency tuning is applied when the number of tasks is high in same tile while task migration is applied for migrating the task (kernel) from high work loaded CPU core to idle GPU core. Siqi Wang et al. proposed a framework *Optic* [21] that automatically predicts the partitioning point and operating frequency of CPU and GPU running concurrently under given thermal constraints. Somdip Dey et al. in ref. [24] proposed hybrid resource mapping technique that reduce the design space exploration (all possible combination of processing element and operating frequency), minimise the profiling time and maximise the user defined rewards (given target such as performance of the application, energy and operating temperature of the processor). The authors of ref. [25] proposed *SoCodeCNN* a framework that convert the source code of the application into visual image where visual image is fed-back to computer vision related machine learning algorithm (e.g. Deep Convolution Neural Network) for training and classification purposes. The proposed framework classify the application into three classes such as *Compute intensive*, *Memory intensive* and *Mixed workload* and accordingly they applied dynamic power management technique to reduce the energy consumption.

1.1 | Motivation

Literature suggests that majority of works related to efficient utilisation of CPU-GPU heterogeneous systems have focussed on improving the performance of the system. As a result, researchers tried dividing tasks into sub-tasks and allocating them to CPUs and GPUs which run concurrently at a high frequency. Though such approaches improve performance, however, they lead to high power consumption which in turn causes an elevation in the temperature. Increase in temperature may cause malfunctioning of some components, device wear out or a complete system shutdown. Thus, it is essential to reduce the power consumption and the temperature of the system during application run.

One such approach would be reducing the frequency of the CPU as it does not have much role to play except for launching the GPU kernel. Thus, the CPU in a heterogeneous CPU-GPU embedded system does not require to be executed at higher frequency while executing a GPU bound application (kernel). We present a motivating example in the next section to validate this fact. Rather, the frequency of operation of the CPU should be determined depending on the processor workload. Based on this argument, we present in this paper a machine learning-based CPU frequency predictor that would aid the programmers in deciding the frequency of the CPU while launching the GPU kernel.

1.2 | Contribution

The novel contribution of this manuscript are summarised as below

- Investigated the effect of CPU frequency on performance and temperature of a GPU specific *OpenCL* application.
- Determined optimal frequency of the CPU cores to host GPU specific *OpenCL* application.
- Proposed a machine learning-based frequency predictor for CPU while it hosts the *OpenCL* application on the GPU.

1.3 | Organisation

The next section provides the details of our investigation on the effect of CPU frequency on temperature of a CPU-GPU heterogeneous embedded system during an *OpenCL* application run. Section 3 describes determination of optimal frequency for the CPU cores while they host a GPU bound *OpenCL* application. Section 4 illustrates the Machine Learning-based technique for predicting the CPU frequency during an execution of a GPU bound *OpenCL* application. The results are discussed in Section 5 while the conclusion summarising our contribution and findings are presented in Section 6.

2 | MOTIVATIONAL CASE STUDY: EFFECT OF CPU FREQUENCY ON TEMPERATURE AND PERFORMANCE OF CPU-GPU SYSTEM

In this section, we present our findings on the effect of CPU frequency while executing GPU specific application on a heterogeneous CPU-GPU system.

2.1 | Hardware platform and benchmark applications

The hardware used for the study was the *ODROID-XU4* embedded platform [26]. The *ODROID-XU4* experimental board houses the *Exynos Systen-on-Chip (SoC)* (used in *Samsung Galaxy* series phones) which is equipped with four high performance *big* cores, four low power *LITTLE* cores and six *Mali-T628 GPU* cores. The *big* cores and the GPU cores have on-chip thermal sensors which were utilised for recording the on-chip temperature for our study. The applications to be run were chosen from standard benchmark suites such as *Rodnia-3.1*, *Polybench-ACC-Master* and *Parboil*. We chose the *OpenCL* version of the applications for the study.

The *OpenCL* version of all applications consist of two parts namely the *host* code and *kernel* code.

host code—code executed by the CPU which helps the GPU execute the *kernel*. The basic functionalities of this code involves initialising the platform, selecting *OpenCL* devices, creating a command queue and transferring code from host to the target device. CPU has to execute this code because the main memory of the *Exynos Systen-on-Chip (SoC)* chip is only accessible to it while GPU has to rely on the CPU to load the *kernel* code from the main memory to its local memory. In order to perform the intended study of CPU performance, we executed the *host* code on *big* cores of *Exynos Systen-on-Chip (SoC)* chip.

kernel code—the code which performs actual computation and runs on the target device (in our case the *Mali* GPU cores).

2.2 | The experiment

Our intention was to study the effect of CPU frequency during execution of a GPU bound *OpenCL* application on an embedded platform (*ODROID XU4* experimental board). Thus, we executed the *host* code on the high performing *big* cores (*ARM A15* cores) of the *ODROID XU4* platform, varied the frequency and recorded the execution time of the application and the average temperature readings of the entire system during the application run. The low energy *LITTLE* cores and the *Mali GPU* cores were allowed to run at their default frequency. While varying the frequency of the *big* cores, we considered operating them at the provided *frequency governors* as well. The *ODROID XU4* platform is provided with two *frequency governors*, namely *Performance* and *On-demand*. The former if selected runs the cores at the highest frequency available while the later runs them depending on the load of the processing core cluster. The *kernel* code which perform actual computation runs on GPU core. In this experiment, *kernel* code is not divided and the entire *kernel* code runs on GPU core. During the experiments each application was executed separately three times in three runs at particular frequency label and average readings of all three runs were considered as resultant average temperature rise and execution time of the application.

2.3 | Motivational results

In order to provide an idea of the frequency of operation and their effect on performance, we provide a performance profile of the *fdtd-2d* application of *Polybench-ACC-Master* benchmark through Figure 1. During profiling of *fdtd-2d* application, the *kernel* code was divided on the CPU and GPU in the ratio of 1:100. The CPU frequencies considered to run the *host* code also included the frequencies corresponding to the *Performance* and the *On-demand* frequency governors available to the *ODROID-XU4* board, as shown in Figure 1. An interesting point to note from Figure 1 is that for all frequencies selected for the CPU cores, the maximum performance (measured in terms of the execution time) of an *OpenCL* application remains more or less same. Thus, we conclude that the performance of a GPU specific *OpenCL* application during its run depends more on the GPU frequency than the frequency of the CPU cores. It is supported by the fact that the CPU cores do not have much role to play except running the *host* code.

Figure 2 on the other hand emphasises that though CPU cores do not affect the performance of the system, they can be the cause of increased system temperature, if run at high frequency. Running the cores at either of the two *governors* causes high power consumption and in turn causes increase in temperature of the cores. This has been highlighted by the thermal profile of the *fdtd-2d* application displayed in Figure 2. Thus, it is essential to judiciously decide on the frequency of operation of the CPU cores during an *OpenCL* application run in order to keep the system temperature within reasonable limits.

3 | DETERMINATION OF OPTIMAL FREQUENCY OF THE CPU CORES

From Figures 1 and 2, we infer that the CPU cores need not be operated at high frequency during an *OpenCL* application run. However, an optimal frequency of operation should be decided. Optimal frequency in this work is defined as the operating frequency of CPU core to run the *host* code of GPU specific application such that overall system temperature remains below tolerable limit and performance does not degrade either. In order to obtain the optimal frequency, we performed the following. First, thermal and performance profile of different *OpenCL* applications were obtained by varying frequency of the host embedded platform (*ODROID-XU4* board in our case). Next, for each frequency (f_i), the following parameters were calculated:

1. Temperature Drop (ΔT_i)—difference between the temperature recorded at the highest frequency of *performance* governor ($T_{\max\_freq}$) in baseline approach and the temperature recorded at that particular frequency (T_{f_i}) of *userspace* governor in proposed approach.
2. Performance Drop (ΔP_i)—difference between the execution time recorded at that particular frequency (T_{f_i}) of *userspace* governor in proposed approach and the execution time recorded at the highest frequency of *performance* governor ($T_{\max\_freq}$) in baseline approach.
3. Temperature drop per unit performance drop- ratio of temperature drop to the % of performance drop ($R_i = \frac{\Delta T_i}{\Delta P_i}$).

Once (ΔT_i) and (ΔP_i) for each frequency was recorded, the one with maximum temperature drop per unit performance drop gave the optimal frequency of operation for that application.

For example, Table 1 illustrates determination of optimal frequency of the *fdtd-2d* application. The performance and temperature drop at various frequencies are recorded and accordingly their ratios (R_i) are computed. The ratio (R_i) for each frequency is computed as seen in Table 1. The maximum value of R_i is obtained at 1600 MHz thus it is considered as the optimal CPU frequency for the *fdtd-2d* application (represented as bold entries in Table 1). Optimal frequency of the other applications are also determined in the similar way.

Table 2 lists the optimal frequency of the CPU cores obtained for 15 *OpenCL* benchmark applications executed on the *ODROID-XU4* board. Since each application has a different optimal frequency of execution, determining such an optimal frequency for any application other than the ones listed in the table would be time consuming since all such applications have to be experimented for the entire range of frequencies. Instead it would be better if a prediction can be performed that would predict the most optimal CPU frequency for an *OpenCL*

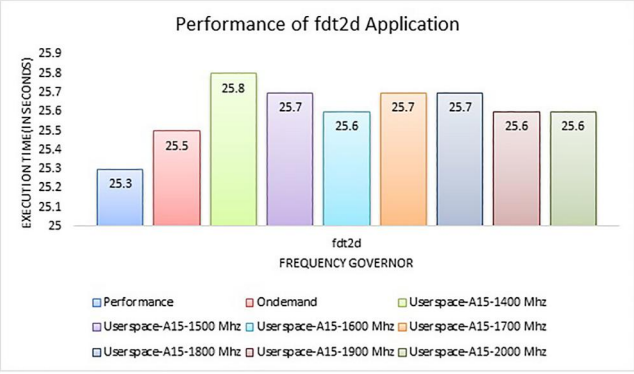

FIGURE 1 Performance profile of fdtd-2d.

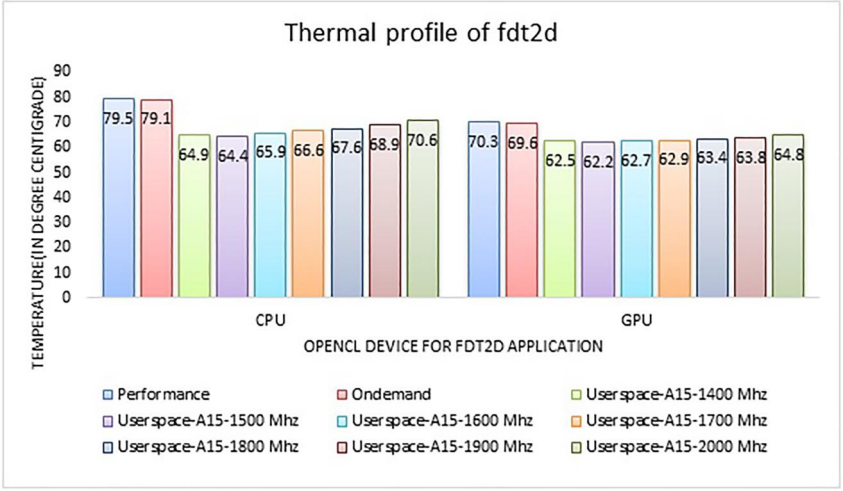

FIGURE 2 Thermal profile of fdtd-2d.

application. Such a predictor based on supervised machine learning (ML) approach is proposed and detailed in the next section.

4 | ML-BASED FREQUENCY PREDICTOR FOR CPU DURING LAUNCH OF GPU KERNEL

In this section, we present a machine learning-based frequency predictor model for CPU cores while launching the GPU kernel (while running the host code). It is a supervised machine learning model that takes the static and dynamic features of the application as an input and predicts the most optimal frequency of operation for the CPU cores during the application run. The frequency predictor model works in three phases as illustrated in Figure 3.

TABLE 1 Determination of optimal CPU frequency of fdtd-2d application

Frequency	% Performance drop (ΔP)	Temperature drop (ΔT)	Ratio of $\frac{\Delta T}{\Delta P}$
1400	1.97	14.6	7.41
1500	1.58	15.1	9.55
1600	**1.18**	**13.6**	**11.5**
1700	1.58	12.9	8.16
1800	1.58	11.9	7.53
1900	1.18	10.6	8.98
2000	1.18	8.9	7.54

Note: The bold value entries represent the optimal CPU frequency of fdtd-2d application.

TABLE 2 Optimal CPU frequency for graphical processing unit (GPU) specific open computing language (OpenCL) application

Application	Optimal frequency (MHz)	Performance drop (ΔP)	Temperature drop (ΔT)
2mm	1400	0.2	9.5
gramschmidt	1400	0.5	15.1
covariance	1400	−9.4	17.1
correlation	1400	−10.6	16.5
myocyte	1400	−12.6	13.9
pathfinder	1500	0.42	16
sgemm	1500	0.4	13.9
lu	1500	−0.2	12.4
backprop	1600	−0.82	9.7
cutcp	1600	0.8	11.3
fdtd-2d	1600	0.3	13.6
bfs-parboil	1600	−1.1	2.9
stencil	1600	0.5	4.9
jacobi-1d	1600	0.4	12.7
convolution-2d	1600	1.2	10

4.1 | Feature extraction phase

The first phase of the ML model is the *Feature Extraction Phase* where static and dynamic features of applications are extracted.

Static features of an application refer to the attributes that represent execution flow of the application such as *number of basic blocks, number of edges in control flow graph (CFG), 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 static features are illustrated in Table 3. In our case, we used the front-end of *Low Level Virtual Machine (LLVM) compiler* namely *Clang* to extract the static features of the 15 *OpenCL* applications. *Clang* utilises the *LLVM compiler* library named *LibStaticFeatExt.so* to analyse the source code of the applications and extract the static features.

Dynamic features of an application represent the attributes which are obtained during execution of the application on a certain hardware platform (in our case the *ODROID XU4* board). These features represent the actual need of resources to execute the application which includes attributes such as *number of clock-cycles, number of instructions executed per second, cache-hit and cache-miss* etc. The complete list of dynamic features considered in our case are illustrated in Table 4. These features were extracted using *Perf* [27],

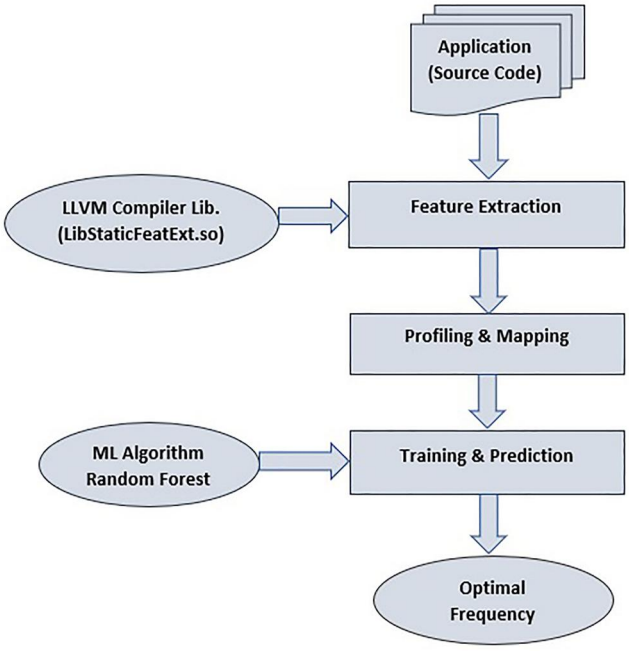

FIGURE 3 Flow chart of ML-model. ML, machine learning.

TABLE 3 Static feature considered in our work

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 than 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 than 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 than two successors and more than two predecessors
sft13	Number of basic blocks with number of instructions less than 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 than 3 phi nodes
sft31	Number of basic block where total number of arguments for all phi-nodes is in greater than 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 than 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 4 Dynamic feature considered in this experiment

dyft1	cpu-cycles
dyft2	Instructions
dyft3	Bus-cycles
dyft4	Cache-references
dyft5	Cache-misses
dyft6	Branch-instructions
dyft7	Branch-misses
dyft8	L1-dcache-loads
dyft9	L1-dcache-load-misses
dyft10	L1-dcache-stores

the event monitoring tool available for *Linux* platforms. *Perf* logs different hardware events during application run on a target hardware platform. These hardware events are obtained through the different run time values of hardware performance counters available in present day microprocessor and microcontroller cores [26].

4.2 | Profiling and mapping phase

The second phase of the ML model is *Profiling and Mapping* in which the applications are mapped to their target class,

obtained through their profile. We executed the host code of the 15 OpenCL benchmark applications on the high frequency A15 cores (*big* cores) of the *ODROID XU4* platform and obtained their temperature and execution profiles. Frequency was varied between 1400 and 2000 MHz. After the temperature and performance profiles of each application were obtained, next the optimal frequency was determined using the procedure detailed in Section 3.

The 15 OpenCL applications are grouped in three classes based on their optimal frequency as presented in Table 5.

Feature Vector and Mapping: Once the frequency class of each application is obtained, next a feature vector is created using the static features and dynamic features of the application obtained using static code analysis and performance counter values respectively. The feature vector is then mapped to the corresponding frequency class of the application. The feature vector along with its mapped class looks something like:

<Static Feature-1, Static Feature-2..., Dynamic Feature-1, Dynamic Feature-2...: frequency label>. This forms the data set for training. In our case, the data set was created using variations of 55 static features and 10 dynamic features each of 15 applications while executing them within a frequency range of 1400–2000 MHz.

Training and Prediction Phase: The last phase of the ML model is *Training and Prediction Phase* where the feature vector set along with its mapped class is fed to the machine learning model for training. We tried different machine learning algorithms such as Naive Bayesian, K-Nearest Neighbour, Support Vector Machine and Random Forest (RF) on the prepared feature vector during training. Among all the above ML algorithms *Random Forest* algorithm yielded the highest accuracy of 75% during prediction. Thus, we considered it for predicting the frequency of the CPU.

4.3 | Validation of CPU frequency predictor

We considered validating our proposed frequency predictor through applications independent of the training data set. The application chosen for cross validation are presented in Table 6. The frequency label predicted by proposed CPU frequency predictor and optimal frequency obtained through the procedure mentioned in Section 3 are presented in Table 6. The table shows that the frequency predicted by proposed CPU frequency predictor matches with the optimal frequency obtained using the brute force method of executing all seven applications for a range of frequencies and thereafter determining the optimal frequency for each.

TABLE 5 Training dataset

Frequency label	Application
A: 1400 MHz	gramschmidt, covariance, correlation, myocyte, 2 mm
B: 1500 MHz	pathfinder, sgemm, lu
C: 1600 MHz	backprop, cutcp, fdtd-2d, bfs, stencil, convolution-2d, jacobi-1d

5 | RESULT

We validated our proposed approach by comparing experimental result with baseline approach and state-of-art work [17]. The baseline approach utilises default Linux scheduler and default frequency governor (*performance*). The default scheduler migrates the task from one CPU core to another CPU core to dissipate the heat generated uniformly while default frequency governor runs applications with maximum frequency to achieve higher performance. The proposal of ref. [17] tends to lower the overall system temperature by scaling down the operating frequency of CPU and GPU in a coordinated manner. The state-of-art approach uses proactive feedback control approach to perform thermal management. The threshold temperature for the multiprocessor systems on chip in the experiments was fixed at 74°C as mentioned in ref. [17]. In all three experiments, each application was executed separately three times in three runs and average readings of all three runs were considered as resultant average temperature rise and execution time of the application. The thermal and performance profile of the experiments are illustrated in Tables 7 and 8 respectively.

Table 7 represent the thermal profile of baseline, proposed and that of the state-of-art work [17]. The average temperature drop in the proposed approach with respect to baseline approach was recorded as 12.5°C while average temperature drop in state-of-art approach was recorded as 9.8°C. This show that proposed approach lowers the processor temperature more effectively than proposal of ref. [17]. Table 8 represents performance profile of all three experiments. The average performance drop in proposed approach with respect to baseline approach was recorded as 1.06% while average performance drop in state-of-art approach with respect to baseline approach was recorded as 16.04%. The proposed approach does not scale down the GPU frequency and as investigated in Section 2 the scaling down CPU frequency does not degrades the performance of GPU specific application too much. Therefore, the overall performance degradation in proposed approach is minimal as compared to state-of-art

TABLE 6 Validation of CPU frequency predictor

Application	Predicted frequency label	Optimal frequency
spmv	C	1600
syr2k	A	1400
bfs-rodinia	C	1600
hotspot3D	A	1400
histo	A	1400
Computational Fluid Dynamics (CFD)	C	1600
syrk	B	1500

TABLE 7 Thermal profile of baseline, proposed and state-of-art experiments

Application	Baseline approach (in °C)	Proposed approach (in °C)	State-of-art approach (in °C)
2mm	71.8	62.3	69.4
gramschmidt	79.1	64	68.4
covariance	79.7	62.6	67.2
correlation	79.8	63.3	67.2
myocyte	79.3	65.4	66.3
pathfinder	79.3	63.3	67.4
sgemm	78.9	65	68.4
lu	77.2	64.8	69.4
backprop	78.6	68.9	68.3
cutcp	80	68.7	69.3
fdtd-2d	79.5	65.9	69.6
bfs-parboil	79.1	67.8	65.8
stencil	78.4	73.5	67.7
jacobi-1d	77	64.3	68.5
convoultion-2d	72.1	62.1	68.7

TABLE 8 Performance profile of baseline, proposed and state-of-art experiments

Application	Baseline approach (in second)	Proposed approach (in second)	State-of-art approach (in second)
2mm	4.4	4.6	4.6
gramschmidt	38.9	39.4	39.7
covariance	242.7	233.3	245.4
correlation	249.3	238.7	247.8
myocyte	149.5	136.9	175.4
pathfinder	56.3	56.8	66.4
sgemm	16.5	16.9	23.9
lu	12.7	12.5	13.4
backprop	30.6	29.8	35.3
cutcp	38.5	39.3	47.8
fdtd-2d	25.3	25.6	26
bfs-parboil	29.8	28.7	37.3
stencil	26.7	27.2	42.5
jacobi-1d	8.9	9.3	10.7
convoultion-2d	5.5	6.7	5.5

approach. The proposal of ref. [17] lowers the operating frequency of both CPU and GPU which degrades the overall performance of the application, while our approach reduces processor temperature more effectively with minimal degradation in performance.

6 | CONCLUSION

In this work, we investigated the effect of CPU frequency on performance and temperature of the processor while executing OpenCL applications on CPU-GPU-based embedded platforms. It was observed that the CPU only indulges in launching the GPU kernel and thus does not need to run at high frequency. The frequency of operation of CPU thus had to be determined. Considering the role of the CPU in running an OpenCL application, we proposed a ML-based frequency predictor to find optimal frequency for *host* code of GPU-specific application. The frequency predicted by our proposed frequency predictor model when evaluated on the state of the art *ODROID XU4* embedded platform reduced the temperature of the overall system by 12.5°C on average and maximum of 17.1°C with average performance degradation of 1.06%.

AUTHOR CONTRIBUTIONS

Rakesh Kumar: Investigation; Methodology; Supervision; Writing – review & editing.

ACKNOWLEDGEMENTS

This research work is supported by **Ministry of Human Resource and Development, Government of India** under the scheme, *Impacting Research Innovation and Technology (IMPRINT-INDIA), Project No. 7482*, *Project URL*: https://imprint-india.org/knowledge-portal-project-pages-list.

CONFLICT OF INTEREST

We declare that we have no competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

ORCID

Bibhas Ghoshal https://orcid.org/0000-0002-8228-6481

REFERENCES

1. OpenCL: The Open Standard for Parallel Programming of Heterogeneous. https://www.khronos.org/opencl/
2. Luk, C.-K., Hong, S., Kim, H.: Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In: 2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 45–55 (2009)
3. Grewe, D., O'Boyle, M.F.P.: A static task partitioning approach for heterogeneous systems using OpenCL. In: Knoop, J. (ed.) Compiler Construction, pp. 286–305. Springer Berlin Heidelberg, Berlin (2011)
4. Pandit, P., Govindarajan, R.: Fluidic kernels: cooperative execution of OpenCL programs on multiple heterogeneous devices. In: Proceedings of Annual IEEE/ACM International Symposium on Code Generation

and Optimization, ser. CGO '14, pp. 273–283. Association for Computing Machinery, New York (2014)
5. Prakash, A., et al.: Energy-efficient execution of data-parallel applications on heterogeneous mobile platforms. In: 2015 33rd IEEE International Conference on Computer Design (ICCD), pp. 208–215 (2015)
6. Grewe, D., Wang, Z., O'Boyle, M.: OpenCL task partitioning in the presence of GPU contention. In: International Workshop on Languages and Compilers for Parallel Computing, vol. 8664, pp. 87–101 (2014)
7. Wen, Y., Wang, Z., O'Boyle, M.F.P.: Smart multi-task scheduling for OpenCL programs on CPU/GPU heterogeneous platforms. In: 2014 21st International Conference on High Performance Computing (HiPC), pp. 1–10 (2014)
8. Seo, S., et al.: Automatic OpenCL work-group size selection for multicore cpus. In: Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques, pp. 387–397 (2013)
9. Wang, H., et al.: Workload and power budget partitioning for single-chip heterogeneous processors. In: 2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 401–410 (2012)
10. Wang, H., et al.: Memory scheduling towards high-throughput cooperative heterogeneous computing. In: 2014 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT), pp. 331–341 (2014)
11. Yeh, L., Chu, R.: Thermal Management of Microelectronic Equipment: Heat Transfer Theory, Analysis Methods and Design Practices. Electronic Packaging Thermal Management of Microelectronic Equipment. ASME Press, New York (2002)
12. Sabry, M.M., Ayala, J.L., Atienza, D.: Thermal-aware compilation for system-on-chip processing architectures. In: Proceedings of the 20th Symposium on Great Lakes Symposium on VLSI, ser. GLSVLSI '10, pp. 221–226. ACM, New York (2010)
13. Zhang, Y., et al.: Hotleakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for Architects. Tech. Rep. (2003)
14. Ansari, M., et al.: Meeting thermal safe power in fault-tolerant heterogeneous embedded systems. IEEE Embed. Syst. Lett. 12(1), 29–32 (2020). https://doi.org/10.1109/LES.2019.2931882
15. Ansari, M., et al.: Thermal-aware standby-sparing technique on heterogeneous real-time embedded systems. IEEE Trans. Emerg. Top. Comput. 10(4), 1883–1897 (2022). https://doi.org/10.1109/TETC.2021.3120084
16. Bhat, G., et al.: Algorithmic optimization of thermal and power management for heterogeneous mobile platforms. IEEE Trans. Very Large Scale Integr. Syst. 26(3), 544–557 (2018). https://doi.org/10.1109/tvlsi.2017.2770163
17. Prakash, A., et al.: Improving mobile gaming performance through cooperative CPU-GPU thermal management. In: 2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6 (2016)
18. Isuwa, S., et al.: TEEM: online thermal- and energy-efficiency management on CPU-GPU MPSoCs. In: 2019 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 438–443 (2019)
19. Henkel, J., Khdr, H., Rapp, M.: Smart thermal management for heterogeneous multicores. In: 2019 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 132–137 (2019)
20. Maity, S., et al.: Thermal load-aware adaptive scheduling for heterogeneous platforms. In: 2020 33rd International Conference on VLSI Design and 2020 19th International Conference on Embedded Systems (VLSID), pp. 125–130 (2020)
21. Wang, S., Ananthanarayanan, G., Mitra, T.: OPTiC: optimizing collaborative CPU–GPU computing on mobile devices with thermal constraints. IEEE Trans. Comput. Aided Des. Integrated Circ. Syst. 38(3), 393–406 (2019). https://doi.org/10.1109/tcad.2018.2873210
22. Ohrling, J., Truscan, D., Lafond, S.: Enabling fast exploration and validation of thermal dissipation requirements for heterogeneous SoCs. In: 2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 114–123 (2021)
23. Ansari, M., et al.: Meeting thermal safe power in fault-tolerant heterogeneous embedded systems. IEEE Embed. Syst. Lett. 12(1), 29–32 (2020). https://doi.org/10.1109/les.2019.2931882
24. Dey, S., et al.: RewardProfiler: a reward based design space profiler on DVFS enabled MPSoCs. In: 2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), pp. 210–220 (2019). https://doi.org/10.1109/CSCloud/EdgeCom.2019.00028
25. Dey, S., et al.: SoCodeCNN: program source code for visual CNN classification using computer vision methodology. IEEE Access 7, 157158–157172 (2019). https://doi.org/10.1109/ACCESS.2019.2949483
26. Odroid-XU4 specification. https://www.hardkernel.com/shop/odroid-xu4-special-price/
27. Perf tool. https://perf.wiki.kernel.org/index.php/Tutorial

How to cite this article: Kumar, R., Ghoshal, B.: Machine learning guided thermal management of Open Computing Language applications on CPU-GPU based embedded platforms. IET Comput. Digit. Tech. 17(1), 20–28 (2023). https://doi.org/10.1049/cdt2.12050