2024 Tenth Indian Control Conference (ICC)
9 - 11 December, 2024
IISER Bhopal, India

Temperature and Energy Optimization in GPUs: An Illustration

Rajarshi Das[1] and Sachit Rao[2]

Abstract— **Optimizing energy consumption of Graphics Processing Units (GPUs) has received considerable interest. Unlike Central Processing Units (CPUs), where energy consumption can be reduced by scaling the core clock frequency dynamically, for a class of workloads, denoted as compute-bound workloads, experimental data indicate that GPUs (the variants considered in this paper) consume the lowest energy at the highest core frequency, thus also leading to the lowest workload runtimes. Operating at the highest frequency, however, leads to a significant increase in the GPU temperature - thus, minimizing both temperature and energy become conflicting objectives. In this work, the simultaneous optimization of energy consumption and GPU temperature is posed as a Multi-Objective Optimization Problem (MOOP) and is solved using the *GICOV* kernel, borrowed from the medical imaging domain, as an illustration. Experimental data reveal that the speed of the on-board GPU fan and the clock frequency can be used as control variables to solve the MOOP. Pareto-optimal solutions are presented using classical and evolutionary approaches; both yield the same solutions. These solutions show that lower GPU temperatures can be obtained in comparison with default GPU settings. Two additional results are presented: the implementation of a dynamic frequency governor *i.* as a feedback controller to control temperature alone and *ii.* as the solution to the Brachistochrone problem to minimize runtime.**

I. Introduction

The use of GPUs to execute scientific computing tasks has proliferated as task execution time can be reduced even if the data being handled is large. The price of obtaining such a performance is increased energy consumption and increased temperature of the GPU. Owing to repeated execution of such tasks, for example in the machine learning domain, it becomes imperative to reduce both energy consumption and temperature rise. A GPU, such as the NVidia *Pascal* GPU, which is considered in this paper, allows for a task to be executed at a core frequency selected from a finite set of frequencies. The class of tasks or kernels considered in this paper may be termed as "compute-bound" kernels, as their operations are primarily driven by the GPU core frequency and have minimal CPU-GPU transfers and memory accesses. Such compute-bound kernels can be found in many critical healthcare applications, such as presented in [1]–[5].

In this paper, the *GICOV* and *convbench* kernels are considered; both these are compute-bound kernels. The first, in the medical imaging domain, is part of the *Leukocyte* workload in the *Rodinia* benchmark dataset, [6], and the second, in the machine learning domain, is part of the *DeepBench* benchmark dataset, [7]. Experimental data, presented in the later sections, acquired while executing these kernels indicate that the *Pascal* GPU (and the *Ampere* class of GPUs - not presented in this paper) consumes the lowest energy and executes them the fastest at the highest core frequency. Correspondingly, there is also a rise in the GPU steady-state temperature with increase in core frequency. These results clearly indicate that energy and temperature minimization are conflicting objectives when GPU core frequency is the lone control variable. On the other hand, data acquired by increasing the on-board GPU fan speed, while keeping the core frequency fixed, shows that the GPU temperature can be reduced by increasing the fan speed without any significant increase in energy consumption. Thus, it may be possible to solve the energy-temperature MOOP using core frequency and fan speed as control variables.

To make the trade-off between GPU temperature and energy consumption, a MOOP is defined with GPU core frequency and fan speed as decision variables and temperature and energy, being modeled using these input variables, as the objective functions; these functions are modeled for the *GICOV* kernel. To solve the MOOP, classical and evolutionary solutions are presented and shown to yield the same solution. The MOOP solution also yields the highest fan speed for all weights; this solution matches experimental data which indicates that fan speed variations do not lead to large variations in energy consumption. The MOOP solutions indicate alternatives to the default settings in GPUs, both for core frequency and fan speed, which lead to a reduction in GPU temperature - this is also the primary and novel contribution of this paper. In addition, based on an experimental temperature dynamic model, a closed-loop implementation of changing the core frequency as the output of a proportional controller with feed-forward component to control temperature at a desired value is also presented; the MOOP is solved using steady-state values. A time-varying frequency governor based on the Brachistochrone problem is also presented.

In related work, Wang et al., [8], determine a pair of CPU and GPU frequencies in a CPU-GPU environment that minimizes runtimes of kernels; they do not consider GPU thermal constraints, but only of the CPU. Dev et al., [9], discuss how CPU-GPU scheduling can reduce temperature, power, and runtime for workloads when compared to execution on the CPU alone; they focus only on task scheduling. Prakash et al., [10], propose a control-theoretic based dynamic thermal management technique that cooperatively scales CPU and GPU frequencies to meet thermal constraints while achieving high performance for mobile gaming; in contrast, the GPU alone is the focus of this paper, where a fan speed-frequency

[1]Rajarshi Das is with Qualcomm India Pvt Ltd, Bangalore, India 560 066. rajarshi.das@iiitb.ac.in

[2]Sachit Rao is with the International Institute of Information Technology Bangalore, Bangalore, India 560 100. sachit@iitb.ac.in

pair is found as the solution to a MOOP. Bhat et al., [11], propose a thermal stability analysis technique to identify a steady state temperature based on power consumption. Based on user specified thermal and time limits, the most power-hungry process is moved to low power CPU cores; this method does not throttle frequency, and works for mobile devices which have both low power and high power CPU cores in addition to GPU cores. Kumar et al., [12], propose a machine learning driven method to identify an optimal CPU core frequency in order to reduce the CPU-GPU system temperature and power consumption for a task running in a CPU-GPU environment. Price et al., [13], identify a combination of voltage, core frequency, and temperature for higher energy efficiency (performance per watt) when compared with the default setting on their GPU device. The temperature is also controlled manually using a water cooling system. The authors hypothesize a temperature-aware voltage and frequency scaling system (T-DVFS), but do not provide any details about this extension. Dasgupta et al., [14], employ thermocouples to prepare a heat map of the various components of the GPU and identify the cores that should be activated in order to reduce the GPU temperature and power consumption; this approach requires a temperature model of all core combinations. Shafi et al., [15], propose a method to reduce the operating temperature of vision-based traffic intersection management system by varying the GPU frequency, such that a desired Frames Per Second rate is achieved, but not an actual temperature value.

The paper is organized as follows: the energy and temperature models are presented in Sec. II along with the definition of the MOOP. In the Results section, Sec. III, the classical and evolutionary solutions to the MOOP are presented and discussed; the time-varying frequency governor and the P-controller are also presented. Concluding remarks are made in Sec. IV.

II. Empirical Models

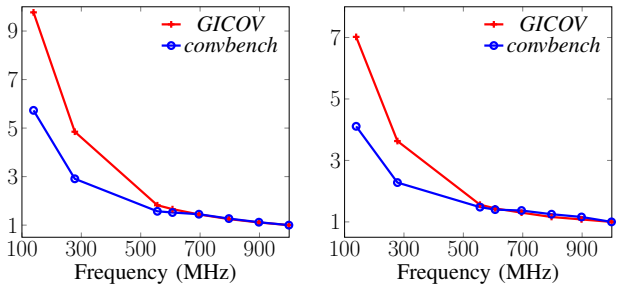
To model variation in energy consumption and temperature of the *Pascal* GPU as functions of core frequency and fan speed, as well as to execute the kernels at different frequencies, the NVIDIA System Management Interface (nvidia-smi), [16], and the NVidia CUDA API (v. 11.5) are used. The *GICOV* and *convbench* kernels are executed to highlight the energy characteristics, while the *GICOV* kernel is used for the MOOP. It is also highlighted that when these kernels are executed at two different CPU frequencies: 2.2 GHz and 3.4 GHz, while the GPU core frequency is kept fixed, the energy consumed, runtime, and GPU temperature are nearly the same. Thus, for this class of workloads, the role of the CPU is not considered further.

A. Energy and Core Frequency

Using the CUDA API, the *Pascal* GPU is commanded to execute a kernel at a core frequency, in MHz, belonging to the finite set

$$f = \{139, 278, 556, 607, 696, 797, 898, 999\}. \quad (1)$$

Data on energy consumption and runtime, acquired by executing the *GICOV* and *convbench* kernels at these frequencies, are shown in Fig. 1; to acquire these data, both kernels were executed at their default workload sizes. As can be seen, the *Pascal* GPU consumes the lowest energy and executes the workloads the fastest at the highest core frequency for these kernels. Though, with increase in core frequency, the power consumption increases (data not shown), net energy consumption reduces; this is mainly because the kernel is executed faster as frequency increases. For the *GICOV* kernel, the average power increases by about 50% as the core frequency changes from 139 to 999 MHz, while the runtime *reduces* by about 90%. Since the rate of runtime reduction is higher than rate of power increase, the total energy consumption drops; note that energy is found as the sum $E = \sum_{k=1}^{N} P(k)\delta T$, where $P(k)$ is the power drawn by the GPU, δT is the sampling interval when power measurements are made, and the index k denotes the number of iterations required for the kernel to be executed.



(a) Ratio of runtime at different frequencies to max(runtime)

(b) Ratio of energy consumption at different frequencies to max(energy consumption)

Fig. 1: Runtime and energy characteristics of the *GICOV* and *convbench* kernels on the *Pascal* P2200 GPU

These results indicate that for such compute bound tasks, operating the GPU at the highest core frequency is both energy *and* time optimal. This behaviour is in stark contrast with workload execution on CPUs where, typically, energy and time optimality *cannot* be satisfied simultaneously. Further, the *Pascal* GPU results also hint that there is no need to change the core frequency during task execution; this is the essence of Dynamic Frequency Scaling approaches in CPUs. To investigate if operating at the highest core frequency has an impact on any other variable associated with the GPU, the variation of GPU temperature with varying core frequencies is explored; this choice is also motivated by the need to minimise system (CPU-GPU) temperature as has been discussed in the literature.

B. Temperature and Core Frequency

By executing the *GICOV* kernel at different core frequencies, the variation in steady-state temperature with change in frequency is shown in Fig. 2a. As can be seen, there is indeed a rise in GPU steady-state temperature with increase in core frequency. Thus, the highest core frequency also leads

to the highest GPU temperature, while on the other hand, it leads to the lowest energy consumption. Therefore, there is a need to perform a trade-off between energy and temperature minimization. Prior to setting up the MOOP to determine this trade-off, the influence of fan speed - the control variable in addition to core frequency - is discussed.

C. Influence of GPU fan speed

Experimental data indicate that there is a significant reduction in GPU temperature with an increase in fan speed (measured in percentage) at all core frequencies, as can be seen in Fig. 2b. Further, when compared with the default fan speed setting, data shows that the speed of 100% results in a $9°$C lower temperature, while the fan speed of 45% results in a $8°$C higher temperature. The energy consumption however does not change significantly at any frequency, as seen in Fig. 2c. It was also observed during experimentation, that at fan speeds higher than the default, a workload is executed faster, thus providing some marginal benefits in runtime and hence energy consumption; however, the significant change is seen in the GPU temperature.

D. MOOP Models

Based on the data presented in the preceding sections, models that relate energy consumption and temperature with core frequency and fan speed are now derived; standard regression methods are used to determine them. Noting that energy consumption may be non-linearly related to core frequency, but remains constant for fan speed, polynomial-type functions of core frequency are investigated for the energy model. The results of regression analysis, which fit the data the best, are presented in Table I.

TABLE I: Coefficients in the energy and temperature models

Coeff. in (2)	Value	Coeff. in (3)	Value
δ	$-5.92e+05$	α	0.0145
ζ	$1.34e+06$	β	-0.2523
η	$-1.002e+06$		
κ	$2.83e+05$	γ	64.66

From the regression analysis, which yielded the best fit as measured by the R^2 value (close to 1) and *p*-values less than a significance level of 5%, the energy model is selected as

$$E = \delta f'^3 + \zeta f'^2 + \eta f' + \kappa, \;\; f' = \frac{f}{1000}, \tag{2}$$

where f is the core frequency, whose values are listed in (1). The temperature model is found to be influenced both by the core frequency and the fan speed F_S; this model is chosen as

$$T = \alpha f + \beta F_S + \gamma. \tag{3}$$

The values predicted by these models and the actual experimental data are shown in Fig. 3; as can be seen, the models (2) and (3) fit the respective data satisfactorily.

III. Results

A. MOOP

A few fundamental concepts of MOOP are introduced in the context of this paper; the book [17] provides additional information on this topic. For the GPU, the MOOP is defined as

$$\begin{aligned} &\min && E\ (2), T\ (3) && \text{(4a)}\\ &\text{subject to} && 139 \leq f \leq 999,\ 45 \leq F_S \leq 100 && \text{(4b)} \end{aligned}$$

where the core frequency and fan speed, f, F_S, respectively, are the decision variables that define the objective functions (2) and (3) - their values $\forall\ f, F_S$ is termed as the objective space; note that a MOOP can also have inequality and equality constraints. The values of f, F_S that satisfy (4b) are termed as the decision space. A solution f^*, F_S^* in the decision space that satisfies all the constraints and bounds is called the *feasible solution*; all feasible solutions together make up the *feasible region* or *search space*.

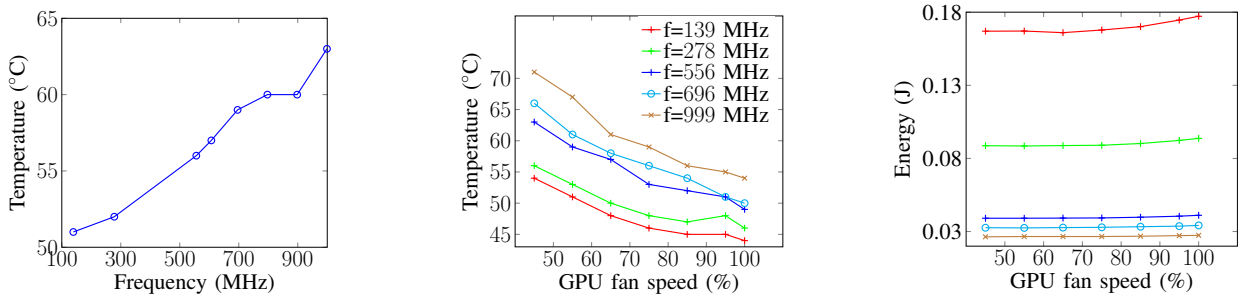
As has been shown, minimizing E and T are conflicting in nature - this is essential to a MOOP. As there could be multiple solutions for the MOOP, the "goodness" of a solution compared to the others is identified in terms of *dominance*; this idea can be interpreted as: A solution, x_1^*, dominates another, x_2^*, if x_1^* is better than x_2^* in one dimension of the objective space and no worse than x_2^* in the other dimensions of the objective space. There could also be "non-dominating" solutions to a MOOP and this set of solutions is called the *Pareto Optimal Set* and the boundary in the objective space defined by all the points in the Pareto Optimal Set is called the *Pareto Optimal Front* (POF). Thus, the main objective of solving the MOOP is to obtain solutions that are close to the POF, since these have near-optimality properties.

As shown in Sec. II, the lowest energy is consumed at $f = 999$ MHz and the lowest temperature is achieved at $F_S = 100\%$ and $f = 139$ MHz. Thus, to find the POF in the objective space for the GPU, after some manipulation of (2) and (3), T is found as a function of E for different values of core frequency f and at $F_S = 100\%$. This analytical solution of the POF (marked in black) matches the solution obtained using the NSGA-II algorithm. The entire objective space is shown in Fig. 4; the points in red are the actual operating points, which as can be seen, are close to the POF.

1) Classical Approach: The ϵ-constraint method, [17], is chosen to solve the MOOP (4). This method is selected as it can handle non-convex POFs; as can be seen in Fig. 4, the POF for the GPU is indeed non-convex. Following this approach, the temperature model, (3), is selected as the objective function and the energy model, (2), is expressed as a constraint; this formulation leads to the single-objective constrained optimization problem:

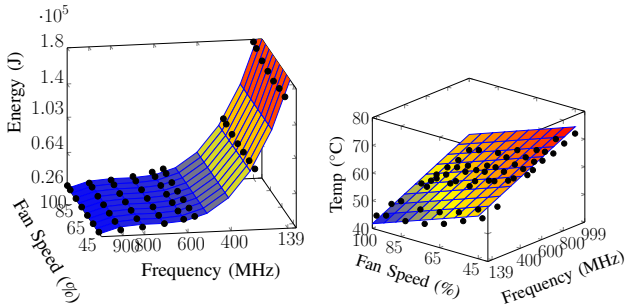
$$\begin{aligned} &\min\ T = \alpha f + \beta F_S + \gamma. && \text{(5a)}\\ &\text{subject to } E = \delta f'^3 + \zeta f'^2 + \eta f' + \kappa \leq \epsilon, && \text{(5b)} \end{aligned}$$

as well as the bounds (4b). Following the procedure described in [17], by introducing the Lagrange multiplier λ,



(a) Temperature and core frequency (b) Temperature and fan speed (c) Energy and fan speed

Fig. 2: Influence of core frequency and fan speed on energy consumption and temperature in executing the *GICOV* kernel at different core frequencies



(a) Energy (b) Temperature

Fig. 3: Variation in Energy and Temperature with frequency and fan speed; model (surface) and experimental data (points) for the *GICOV* kernel

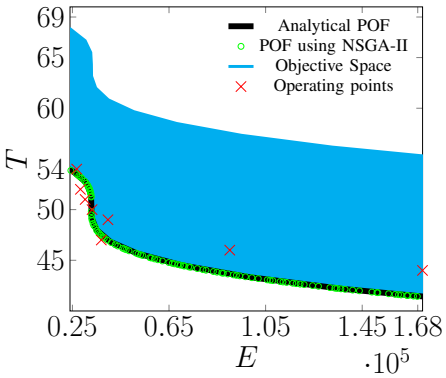


Fig. 4: POF and Objective Space for the *GICOV* kernel

the Kuhn-Tucker optimality conditions are stated; these are

$$\frac{\partial T}{\partial f} - \lambda \frac{\partial (E - \epsilon)}{\partial f} = 0, \tag{6a}$$

$$(E - \epsilon) \geq 0, \ \lambda \geq 0, \ \lambda (E - \epsilon) = 0 \tag{6b}$$

After simplification and applying the conditions in (6), which results in $\lambda > 0$ needing to be true, the optimal solution for the core frequency f'^*, is given by

$$\delta f'^{*3} + \zeta f'^{*2} + \eta f'^* + \kappa = \epsilon, \tag{7}$$

which clearly varies for different values of ϵ. Note that as E is not a function of fan speed, F_S, the resulting solutions yield an optimal value for energy E for any value of F_S.

For values of frequency f given by the set (1), the parameter ϵ is bounded according to $24368 \leq \epsilon \leq 167965$; this parameter partitions the solution space at $E = \epsilon$. As the POF is determined for $F_S = 100\%$, a choice of ϵ determines the solution of f^* to the MOOP (5). It can be observed from the POF, Fig. 4, that choosing a large value of ϵ leads to a lower temperature, as f^* is small, but the energy consumption increases; this solution prioritizes temperature over energy, and hence runtime. On the other hand, choosing a smaller value of ϵ prioritizes energy and runtime over temperature, by increasing f^*. This is the trade-off that is sought by solving the MOOP.

Remarks: In the vicinity of $f^* \approx 667$ MHz, the derivative $dE/df \approx 0$, while $dT/df \neq 0$; thus, this section of the POF does not constitute a solution; this is also confirmed by the solution found by the evolutionary approach, discussed next.

2) Evolutionary Approach: The genetic algorithm NSGA-II is selected to solve the original MOOP, (4). For a population size of 100, the resulting POF is shown in Fig. 4 (marked in green circles). As can be seen, it matches the analytical solution (along with the part of the POF where no solutions can be found) and yields $F_S^* = 100\%$ as the solution in the design space. Using the ASF routine[1], different weights can be assigned to energy consumption and temperature; a choice of weight yields a corresponding f^*, F_S^* solution in the decision space. The operating points closest to solutions found for a selected set of weights are presented in Table II; the closest operating point is found by determining the smallest Euclidean distance between the NSGA-II solution and all feasible f, F_S pairs. Note that during actual implementation, the feasible solutions lead to sub-optimal results. As can be seen in Table II, giving a higher weight to energy leads to an increase in temperature - thus, both the Classical and Evolutionary approaches yield the same solution. In comparison with operating the GPU at the default fan speed setting, for instance, operating at $f^* = 999$, $F_S = 100\%$, as given by the MOOP solution, leads to a 14% reduction in GPU temperature with only a 1% increase in energy consumption.

[1] https://pymoo.org/misc/decomposition.html

TABLE II: Results of NSGA-II and ASF routines

Weightage for optimizing Energy	Closest f^*, F_S^*	E (kJ,), T (°C)
0.5	278, 100	90.1, 46
0.8	556, 100	39.8, 49
0.9	696, 100	33.1, 50
0.99	999, 100	26.8, 54

B. Frequency governors

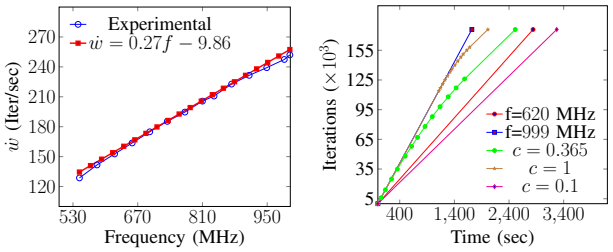
In this section, we briefly describe the implementation of two frequency governors to solve a Single Objective Optimization Problem: minimize energy consumption or temperature.

1) Brachistochrone governor to minimize E: The Brachistochrone curve determines the path of *fastest* descent of an object between two points in the gravitational plane; the speed is defined by $v = \sqrt{2gy}$, where y is the height difference. For the GPU, by posing the size of the workload *remaining to be executed*, denoted by w, as the height, and the work rate, $\dot{w}$, as v, a frequency governor is designed to examine if a faster runtime can be achieved. From experimental data, Fig. 5a, it can be seen that $\dot{w} \propto f$; hence, theoretically, a lower runtime should be possible to achieve. Thus, f is selected as $f_B = \sqrt{cw}$, where the user-defined parameter c is analogous to $2g$.

With f_B, the kernel execution is defined by the dynamics $\dot{w} = -k\sqrt{cw}$, where k is a proportionality constant and $w = w_{\max}$ at $t = 0$, since the task is still to be executed. By integration, it can be shown that the kernel runtime, t_{FB}, which is when $w = 0$, is given by $t_{FB} = 2\sqrt{\frac{w_{\max}}{c}}$. Suppose the kernel is executed at a constant frequency, say f_c; in this case, the runtime is the simple ratio $t_{Fc} = \frac{w_{\max}}{kf_c}$. Now, a faster runtime from the Brachistochrone governor is achieved only if the ratio $\frac{t_{FB}}{t_{Fc}} = 2\frac{f_c}{\sqrt{cw_{\max}}} < 1$, that is, if $\sqrt{cw_{\max}} > 2f_c$. Noting that $\max(f_B) = \sqrt{cw_{\max}}$, this time-varying governor can achieve a faster runtime only if $\max(f_B) > 2f_c$. While in theory this appears feasible, since only the frequencies in (1) can be implemented, it is evident that only for some values of f_c can f_B yield a lower runtime. Note that in the original Brachistochrone problem, *no bounds* on the speed of the object are imposed, whereas in the GPU, such bounds exist.

The results of executing the *GICOV* kernel using f_B and at different constant frequencies are shown in Fig. 5b. Given the size of this kernel and that $\max(f) = 999$ MHz, $c = 0.365$ results as a choice for this parameter. As can be seen, only for $f_c = 620$ MHz and lower (and $c > 0.1$) does the Brachistochrone governor yield any reduction in runtime and hence energy consumption. It is again highlighted that since only discrete clock frequencies can be implemented, the governor f_B cannot strictly follow the proposed design. Further, since it also commands very small values of f (close to 0 even) when the workload is close to completion, in the implementation, a lower-bound of $f = 544$ MHz is set.

2) Temperature governor: By executing the *GICOV* kernel at various core frequencies and from the resulting data,



(a) Rate of execution vs GPU core frequency (b) Runtime change with f_B using different values of c

Fig. 5: Design and performance of a Brachistochrone frequency governor for the *GICOV* kernel

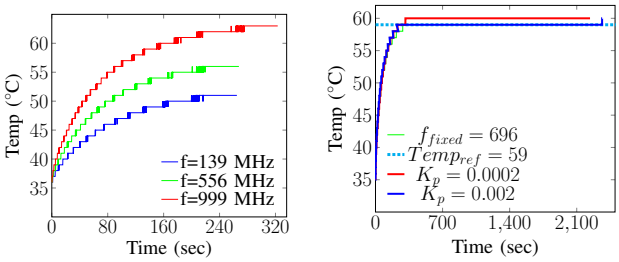
shown in Fig. 6a, an open-loop dynamic model relating temperature T and frequency f can be derived. The dynamics is approximated by the first-order model

$$\tau \frac{dT}{dt} + T = b_1 f + b_0, \tag{8}$$

where the time-constant $\tau = 72.96$ sec and the constants $b_1 = 0.014\ \frac{^\circ\text{C}}{\text{MHz}}$ and $b_0 = 48.65^\circ$C. The temperature governor is selected as a simple Proportional controller with a feed-forward component

$$f = \frac{1}{b_1}\left(T_{\text{ref}} - b_0\right) - K_p\left(T - T_{\text{ref}}\right)\frac{\tau}{b_1} \tag{9}$$

to meet the control objective $T \to T_{\text{ref}}$, where T_{ref} is the *target* temperature and K_p is the proportional gain. In the closed-loop, the dynamics of the error $T_{\text{err}} = (T - T_{\text{ref}})$ follows $dT_{\text{err}}/dt = -\left(K_p + \frac{1}{\tau}\right)T_{\text{err}}$, which implies that $T_{\text{err}} \to 0$ at a rate decided by $\left(K_p + \frac{1}{\tau}\right)$. The transient component of the response can also be modified by selecting K_p accordingly. Note that the feedback component can also be selected as a Linear Quadratic Regulator or other control techniques. The results of implementing this governor with different values of K_p are shown in Fig. 6b.



(a) Open-loop dynamics (b) Controlling the temperature at $T_{\text{ref}} = 59^\circ$C

Fig. 6: Modeling and control of GPU temperature using core frequency as control; the *GICOV* kernel is executed at different frequencies.

As can be seen in Fig. 6b, the governor is able to control the temperature close to the desired value of $T_{\text{ref}} = 59^\circ$C; an error of 1°C is achieved. Further, the frequency reaches a steady-state value 696 MHz, which is also when the GPU temperature is close to the chosen reference value.

IV. Conclusion

From experimental data, execution of compute-bound kernels on the *Pascal* GPU consumes the least energy at the highest core frequency; however there is a corresponding increase in temperature. By posing a MOOP, it is shown that it is possible to perform the temperature-energy trade-off, by introducing an additional degree of control as the GPU fan speed. Experimental results indicated that increasing the GPU fan speed at a constant core frequency decreases the final temperature significantly, with minimal change in energy consumption and vice versa. Two methods, both of which yielded the same solution, were demonstrated. Two single objective control problems were also addressed: control of runtime and temperature, both using core frequency as a control variable. As this control is bounded, as shown in the Brachistochrone governor, it is possible to reduce the runtime and save energy only when compared to a run at a fixed frequency lesser than the maximum allowed frequency. While the results were demonstrated for a single kernel, the results of this paper can be extended to other kernels. Further, the occurrence of changes in the empirical models with changes in software (say, with updates) may also be considered.

Acknowledgment

The authors would like to thank IIIT-B for financial support provided to Rajarshi Das towards earning his M.S. by Research degree and the Mphasis Cognitive Computing CoE for access to the GPU.

Appendix

Genetic algorithms (GAs) are a class of evolutionary approaches to solving the MOOP. In most GAs, a random population or set of solutions is selected and the corresponding objective value and constraint violations are identified, and combined together to generate a fitness value. A termination condition (e.g. the number of iterations) is checked. If it is not true, a sequence of three operators (namely selection, crossover, and mutation) is applied and a modified population is prepared. The generation counter is incremented by 1, and the entire cycle starting from evaluation repeats. The selection operator (for example, tournament-based) is used to make duplicates of the existing population and eliminate the bad solutions in the extended population while keeping the population size constant. The crossover operator picks two strings at random from the mating pool, and some portion of the strings are exchanged in order to create new strings; this is equivalent to creating new solutions. The mutation operator is used to maintain diversity in the population. The NSGA-II algorithm is of the *elitist* type of GAs, where the best $\epsilon\%$ of the current population is directly copied over to the next generation, and the remaining $(100-\epsilon)\%$ of the new population is created after applying the three operators on the entire current population including the best $\epsilon\%$ solutions.

References

[1] B. Peng, X. Huang, S. Wang, and J. Jiang, "A Real-Time medical ultrasound simulator based on a Generative Adversarial Network Model," in *Proceedings of the International Conference on Image Processing, 2019*, pp. 4629–4633. [Online]. Available: https://doi.org/10.1109/icip.2019.8803570

[2] R. Jacques, R. Taylor, J. Wong, and T. McNutt, "Towards real-time radiation therapy: GPU accelerated superposition/convolution," in *Computer Methods and Programs in BioMedicine*, vol. 98, 2010, pp. 285–292.

[3] K. Vanommeslaeghe, O. Guvench, and A. D. J. MacKerell, "Molecular mechanics. current pharmaceutical design," *Journal of Computational Chemistry*, vol. 20, no. 20, pp. 3281–3292, 2014.

[4] S. Jung, "Parallelized pairwise sequence alignment using CUDA on multiple GPUs," in *Proceedings of the UT-ORNL-KBRIN Bioinformatics Summit, 2009*, vol. 10, 2009.

[5] D. Lee, I. Dinov, B. Dong, B. Gutman, I. Yanovsky, and A. W. Toga, "CUDA optimization strategies for compute- and memory-bound neuroimaging algorithms," in *Computer methods and programs in biomedicine, 2019*, vol. 106, pp. 175–187.

[6] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of the IEEE International Symposium on Workload Characterization, October 2009, doi: 10.1109/IISWC.2009.5306797*, 2009.

[7] Baidu, "DeepBench benchmarking suite," https://github.com/baidu-research/DeepBench, 2016, accessed: 2023-12-12.

[8] S. Wang, G. Ananthanarayanan, and T. Mitra, "OPTiC: Optimizing Collaborative CPU-GPU Computing on Mobile Devices with Thermal Constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 3, pp. 393–406, 2019.

[9] K. Dev, I. Paul, W. Huang, Y. Eckert, W. Burleson, and S. Reda, "Implications of Integrated CPU-GPU Processors on Thermal and Power Management Techniques," 2018.

[10] A. Prakash, H. Amrouch, M. Shafique, T. Mitra, and J. Henkel, "Improving mobile gaming performance through cooperative CPU-GPU thermal management," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.

[11] G. Bhat, S. Gumussoy, and U. Y. Ogras, "Power and thermal analysis of commercial mobile platforms: Experiments and case studies," in *2019 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2019, pp. 144–149.

[12] R. Kumar and B. Ghosal, "Machine learning guided thermal management of Open Computing Language applications on CPU-GPU based embedded platforms," *IET Computers and Digital Techniques*, vol. 17, no. 1, pp. 20–28, Dec 2022.

[13] D. C. Price, M. A. Clark, B. R. Barsdell, R. Babich, and L. J. Greenhill, "Optimizing performance-per-watt on GPUs in high performance computing," *Computer Science - Research and Development*, vol. 31, no. 4, pp. 185–193, Sep 2015.

[14] H. K. Aniruddha Dasgupta, Sunpyo Hong and J. Park, "A New Temperature Distribution Measurement Method on GPU Architectures Using Thermocouples," 2012.

[15] G. A. Omais Shafi, Sachin Kumar Chauhan and R. Sen, "DynCNN: Application Dynamism and Ambient Temperature Aware Neural Network Scheduler in Edge Devices for Traffic Control," in *Proceedings of ACM SIGCAS/SIGCHI Conference on Computing and Sustainable Societies (COMPASS), 2022*, 2022, pp. 513–528.

[16] NVidia, "NVidia System Management Interface program," http://developer.download.nvidia.com/compute/DCGM/docs/nvidia-smi-367.38.pdf, 2016, accessed: 2023-12-12.

[17] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. USA: John Wiley & Sons, Inc., 2001.