**Other** **Technical** **Details**

1. **Origin** **of** **the** **Proposal:** *(Maximum* *1* *page)*

Heterogeneous computing clusters comprising multiple CPUs and GPUs have become the preferred platform of choice for execution of data parallel workloads with the introduction of parallel programming languages like OpenCL and CUDA. Workloads in this context typically comprise compute intensive applications in diverse domains ranging from linear algebra, image processing, to molecular physics and biomedical sciences. In the past few years, OpenCL has emerged as a parallel programming framework for developing data parallel programs suited for execution on both CPU and GPU devices. The OpenCL API supports program portability across different types of devices and provides the programmer with a vast array of options to write data-parallel programs efficiently for heterogeneous architectures.

However, programmers usually require a considerable amount of expertise for designing efficient parallel algorithms for data parallel computation on heterogeneous architectures. Apart from designing parallel algorithms, the programmer needs to focus on issues of synchronization, efficient transfer of data and understanding the target characteristics of devices in the system. The primary burdens faced by a programmer therefore from an implementation perspective are as follows.

1. Designing large complex programs which take into account the management of data and synchronization of computational segments during execution of one particular task on a system.
2. Designing efficient scheduling algorithms for executing multiple workloads in parallel on a heterogeneous architecture with a fixed number of processors.

This proposal aims to address the above two issues by developing a generic scheduling framework for a cluster comprising multiple heterogenous compute nodes enabled with both CPUs and GPUs. The framework shall facilitate rapid development and deployment of OpenCL applications, by analysing statically how tasks in a workload can be distributed for increased performance and determining optimum runtime scheduling decisions given a set of workloads and a target cluster architecture. The main objectives of this project are enumerated below.

1. Develop a static analysis module for OpenCL workloads that would generate relevant static program information which can be mined using Machine Learning classifiers in order to ascertain the CPU and GPU affinity of application tasks.
2. Use the static partitioning information derived above for code generation aiding automated runtime partitioning of OpenCL workloads across multiple CPU and multiple GPU devices.
3. Develop efficient partition-aware scheduling policies which exploit the static information generated in the previous steps. Such scheduling heuristics shall be employed by the system for task dispatch among heterogeneous devices while optimizing the throughput.
4. Providing a generic API for users of the framework so that new partition-aware job schedulers can be specified at a high level while the low level implementation is taken care of by the framework itself.

In short, the proposal aims to create a job scheduling and task dispatch system with automated on-the-fly task partitioning and code generation for CPU-GPU clusters with target workloads being complex scientific workloads written in OpenCL.

1. **Review** **of** **status** **of** **Research** **and** **Development** **in** **the** **subject**

* 1. **International** **Status:** *(Maximum* *2* *pages)*

There exists a fair amount of literature in the context of efficiently mapping data parallel workloads on heterogeneous architectures. The literature can be broadly divided in two distinct categories.

1) **Partitioning of workloads :** Mapping the computation of a single program across CPUs and GPUs. This entails distributing the data space to be processed in by application kernel across CPUs and GPUs.

2) **Scheduling of workloads :** Mapping computations of multiple programs across CPUs and GPUs. This entails devising static or runtime scheduling policies for a set of data parallel workloads on a heterogeneous architecture.

Partitioning can again be classified into two categories – Static and Dynamic. Dynamic strategies for partitioning programs take into account the current load of the system environment at runtime and perform load balancing of the computation of a single program across CPU and GPU devices. The work reported in [1] proposes a framework for dynamically distributing data parallel workloads in a system with a multicore CPU and single GPU. The framework is built using device specific parallel programming platforms - OpenMP for CPU and CUDA for GPU and has been devised to only consider Reduction Computation problems. Merge [2] is another dynamic framework which has been designed specifically for map reduce applications. The work reported in [3] discusses an online adaptive scheduling policy for partitioning work loads of OpenCL kernels in an integrated CPU-GPU system. In contrast, static partitioning methodologies for partitioning programs are actually compilation strategies which are designed to work without the knowledge of dynamic execution scenarios. Therefore these techniques are relatively lightweight in nature and easier to deploy. The work reported in [4] supports an adaptive mapping scheme for dispatching tasks to processors in a heterogeneous system by building an analytical performance model based on profiling programs with different problem sizes. The partitioning algorithm reported in [5] is also based on extensive offline profiling, assigning performance values to devices in the system and statically devising a distribution of the workload to the devices in the system. There exists another class of static methods [6 ,7 ,8] that use mechanisms like machine learning to predict near optimal program partitions. These methods rely on training machine learning based classifier models for taking the decision of partitioning a task across the cores of the heterogeneous system. The decision relies on the static program features of the program under consideration.

The primary objective of heterogeneous CPU-GPU multicore scheduling algorithms is to report a schedule comprising the start times of tasks and the respective devices (CPU or GPU) to which the tasks have been mapped such that the makespan is minimized. The established scheduling algorithms first develop a functional performance model by profiling all the tasks on the architecturally different devices in the system. The execution time information for each task on each device is used to generate an initial task to device mapping. Based on the load of the system at a time instant, a runtime decision is taken to dispatch the task across a particular set of devices. In most scheduling methodologies, the input is a task set where the dependencies are captured by a directed acyclic graph (DAG). The goal of these methods are to efficiently map tasks to processors in a heterogeneous multicore so that the schedule makespan is minimized while respecting precedence constraints between the tasks. Typically heuristics are used to ascertain near optimal solutions for scheduling a DAG of tasks. List based scheduling algorithms are used extensively, the most popular among them being the Heterogeneous Earliest Finish Time (HEFT) algorithm [9]. The HEFT algorithm selects the task with the highest priority in the DAG and schedules the task on the processor which minimizes its earliest finish time. Another sophisticated heuristic known as Balanced Minimum Completion Time (BMCT) is reported in [10] where tasks are first mapped to processing cores based on minimum execution time. Each list of tasks mapped to a particular machine are again sorted in order of earliest start time on that machine. The works discussed so far are based on scheduling one single DAG of tasks. Strategies leveraging list based heuristics require execution profile information for each task on each device in the system before scheduling. Scheduling multiple DAGs poses a more complex problem as the aim now is not only to reduce the schedule makespan but also to ensure fairness [11]. Fairness measures are typically computed by how the execution time of a DAG is affected on the basis of resource usage by other DAGs. This work however considers that all DAGs arrive at the same time in the system. The work reported in [12] generalizes the problem by considering that each DAG may arrive at different points of time. For scheduling multiple DAGs, prior profiling information of each task needs to be available.

Several frameworks have emerged over the last few years that support scheduling on heterogeneous architectures. The most notable scheduling framework for heterogeneous platforms is StarPU [13] which provides the users an interface for designing and experimenting scheduling policies. The StarPU runtime system allows users to design scheduling priority functions for experimentation. An extension on StarPU [14] supports scheduling multiple tasks in parallel on a heterogeneous system. The work reported in [15] presents an unified OpenCL implementation called SOCL which directly extends upon StarPU for handling and managing execution of OpenCL workloads across multiple devices using different scheduling policies. The SOCL API typically supports scheduling algorithms that require profiling information for each task on each device in the system beforehand. The work reported in [16] extends upon the OpenCL API by incorporating extensions for automated utilization of devices in a heterogeneous multicore. This work also presents a unified OpenCL implementation by incorporating a task queuing extension layer.

* 1. **National** **Status:** *(Maximum* *1* *page)*

The work reported in [17] proposes a dynamic partitioning framework which performs fine grained workload partitioning of a single OpenCL kernel across a single CPU and single GPU. The work reported in [18] reports a machine learning based static partitioning scheme for OpenCL tasks which focuses on the impact of control flow behaviour on partitioning strategies.

* 1. **Importance** **of** **the** **proposed** **project** **in** **the** **context** **of** **current** **status** *(Maximum* *1* *page)*

(*Highlight* *what* *is* *the* *new* *area* *or* *gap* *which* *will* *be* *solved* *in* *the* *project* *in* *relating* *to* *what* *is* *already* *known.* *This* *is* *a* *very* *important* *section* *to* *project* *the* *novelty* *content* *of* *the* *proposal*)

Given the state of the art in the field, it is imperative that heterogeneous computing platforms are here to stay and become architecturally richer in future. For example, high-end computing platforms already come equipped with general purpose CPUs, accelerators (Xion-Phi), On-chip GPUs (HD-Graphics), and off-chip GPUs. Each component is suitable for a specific kind of computation. While off-chip GPUs offer more parallelism and computational power, they incur a data transfer overhead when compared with on-chip GPU components. In general, the OpenCL programming paradigm provides a generic language for programming all such devices uniformly and despatching compute kernels across devices in a partitioned manner. For saving computation bandwidth, it often makes sense to partition parallel tasks among multiple devices based on the task characteristics and compute intensiveness. The power offered by the OpenCL paradigm also comes with all these additional complexities related with online task partitioning, scheduling and dispatching which cannot be done efficiently using standard job schedulers who are simply incapable of such novelties like partitioning compute kernels among multiple devices. Hence,

1. it is essential to come up with such generic job schedulers which can effectively partition task nodes of complex application task graphs across multiple heterogenous devices in order to exploit the parallelism offered by the platform.
2. it makes sense to develop novel task partitioning algorithms which can split an OpenCL kernel (task node) among multiple CPU and GPU devices on-the-fly after making a decision whether such a split is advantageous for the overall end-to-end computation.
3. it is necessary to develop low-level API functions which can perform all the device level kernel partitioning, device-task mapping, scheduling and dispatch operations for partitioned tasks as dictated by some partition aware scheduling strategy.

* 1. **If** **the** **project** **is** **location** **specific,** **basis** **for** **selection** **of** **location** **be** **highlighted**: *(Maximum* *1/2* *page)*

NA

1. **Work** **Plan:**

* 1. **Methodology**: *(Maximum* *of* *5* *pages)*

The objective of this project is to present a scheduling framework built on top of the OpenCL API for execution of multiple data parallel workloads on heterogeneous architectures comprising multiple CPUs and multiple GPUs. An OpenCL workload can be represented as a graph G =<V,E> where V represents the set of individual data parallel tasks to be executed and E represents the set of dependencies between the tasks. Each member of E is typically a pair (vi,vj) which denotes that the task vimust finish execution before vj starts. The objective of scheduling algorithms in this context is to optimally schedule a set of k workloads G1,G2,…,Gk­ on a cluster of n CPUs and m GPUs so as to minimize the makespan of the schedule.

The project involves a considerable amount of implementation for developing a robust interface which programmers can use for developing data parallel workloads. Additionally, algorithmic evaluation and validation of scheduling policies used in this context will also be addressed in detail. The entire project can be divided into the following tasks which is illustrated in Figure 1.

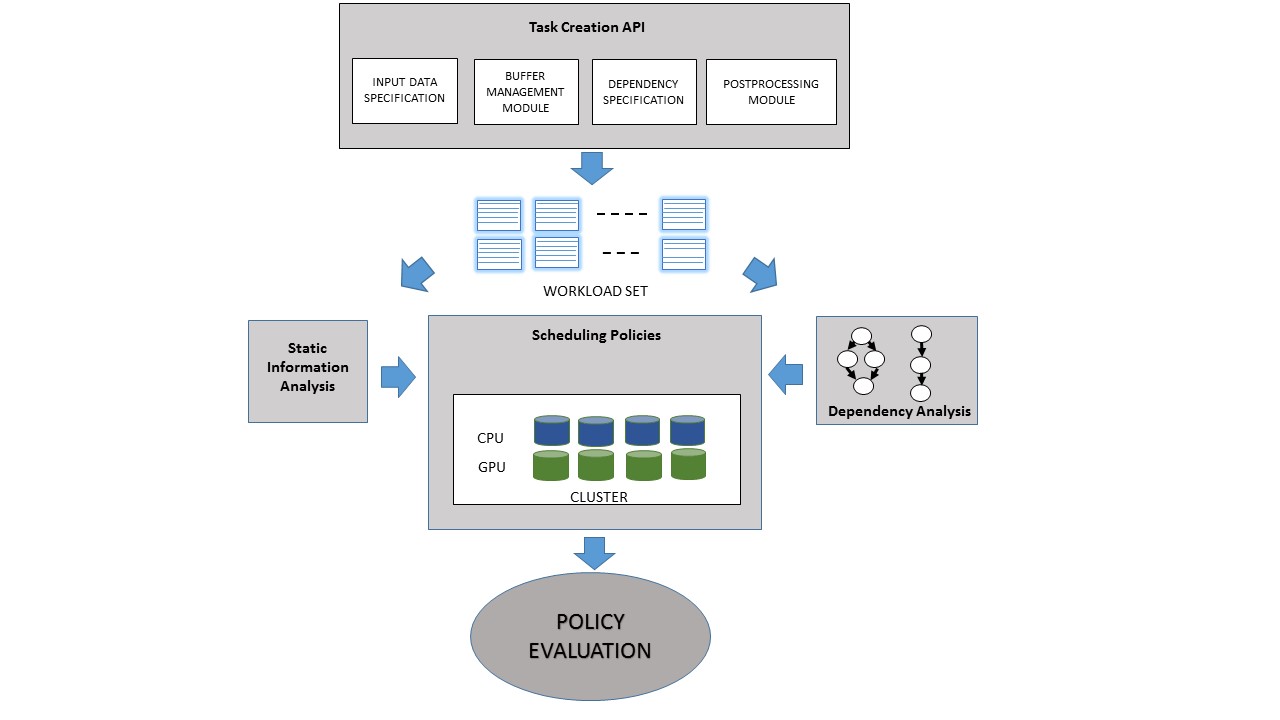
****

Figure : Overall Framework

**Task A: Development of Task Creation API**

Applications written using the OpenCL API typically entail designing data parallel tasks (kernels) which form the core computation of any data parallel workload. A data parallel workload comprises several kernels and a programmer is burdened with the task of writing program segments for ensuring the following conditions.

1. Efficient management of data between kernels,
2. Ensuring dependency constraints between kernels,
3. Designing synchronization mechanisms between kernels.

The primary reason for developing a Task Creation API is to relieve the programmer of the burden involved in ensuring the above conditions. The user would only be concerned with writing the code for the kernel and specifying the input specification and the dependencies between the kernels. The information to be specified can be provided in a specification file. The framework would process the specification file and execute the workload accordingly. In Figure 1, the Task Creation API expects four primary requirements from the user which are discussed as follows.

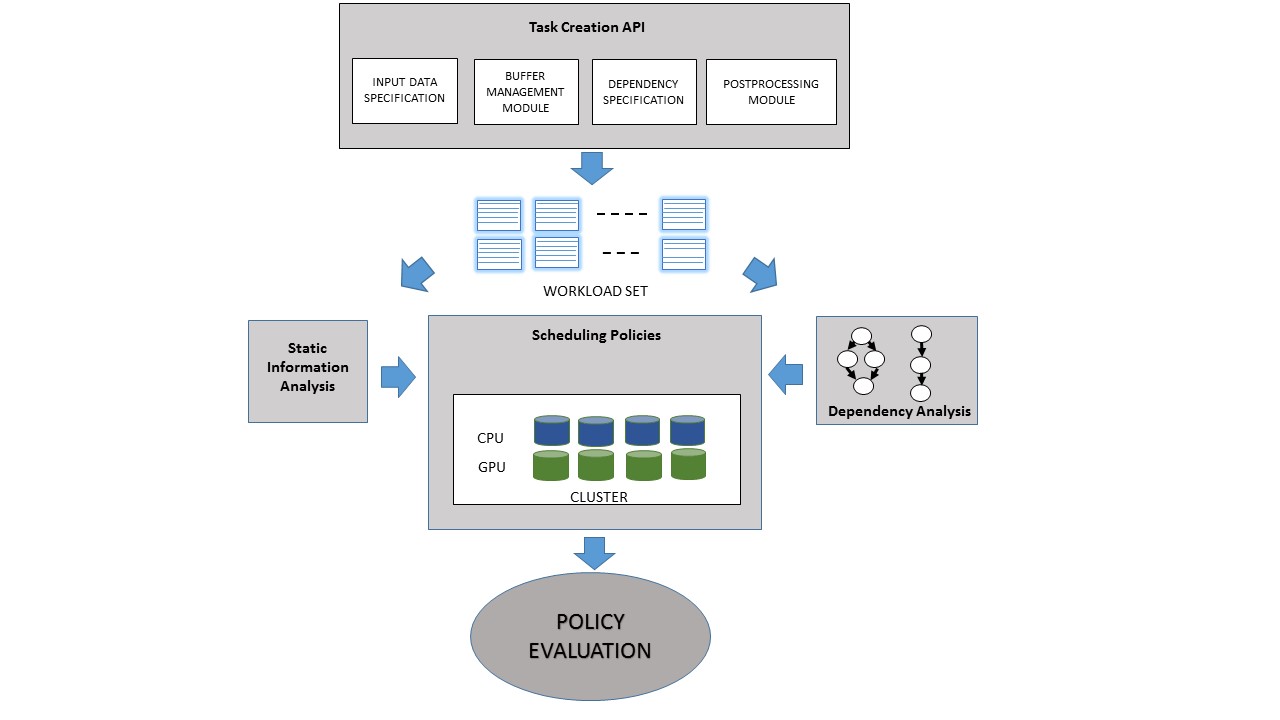
1. **Input Specification:** This refers to the data type, size and number of buffers involved.

2. **Kernel Dependency:** This refers to the specification of precedence constraints between tasks (kernels).

3. **Buffer Specification:** This refers to the read and write dependencies between buffers of different kernels.

4. **Post Processing Module:** This refers to additional functional modules other than the data parallel kernels which can be designed and incorporated by the user.

The above specification shall be used by the low level system to automatically set up suitable OpenCL buffers and data transfer directives which satisfy the conditions as identified earlier.

**Task B: Static Information Analysis**

The primary objective of this module is to ascertain how static information of programs can be used for characterizing CPU / GPU affinity. Static information entail obtaining program features representing properties by virtue of which we characterize whether the concerned program is more suited for a GPU or a CPU or for both in a partitioned fashion. Different features capture different computing aspects of the program and can be utilized for making decisions at the time of scheduling. Our objective would be to test with features that characterize total computation, control flow behaviour, memory requirements and total data size in the workload. Such static program features are listed in Table 1 and discussed in [6] and [18].

A thorough evaluation of the features listed in Table 1 above must be made to identify which features would prove most useful and for eliminating redundant features. The static analysis module would support automated extraction of program features from OpenCL kernels. A thorough evaluation of machine learning algorithms will be done to train classifier models using these program features for ascertaining what would be the optimum distribution of computation across the CPU and the GPU for a single kernel task.

**Table 1:** List of Static Program Features

|  |  |
| --- | --- |
| **Feature Category** | **Feature Description** |
| Compute | Number of integer operations |
| Number of vector integer operations |
| Number of float operations |
| Number of vector float operations |
| Number of builtin OpenCL functions |
| Total compute operations |
| Total compute operations per memory access |
| Synchronization | Number of barrier function calls |
| Memory | Total memory accesses |
| % of shared memory accesses |
| % of coalesced memory accesses |
| Control  Flow | % of branch divergence |
| Total number of divergent branches |
| Problem  Size | Total number of work items launched |
| Total data size transferred between host and device |

## TASK C: Dependency Analysis

## 

The primary objective of this module is to first analyse dependencies inherent in each workload and static information obtained in the previous step to infer an initial task to device mapping on a heterogeneous architecture. A dependency analysis typically analyses the precedence constraints between the tasks in the DAG and understand which device is best suited for executing a particular task in each workload. A composite graph is created from the set of workloads given as input. The corresponding graph for 3 workloads G1, G2 and G3 is shown in Fig 2. The edges of the graph represent precedence constraints between the tasks and the amount of data transferred between them.

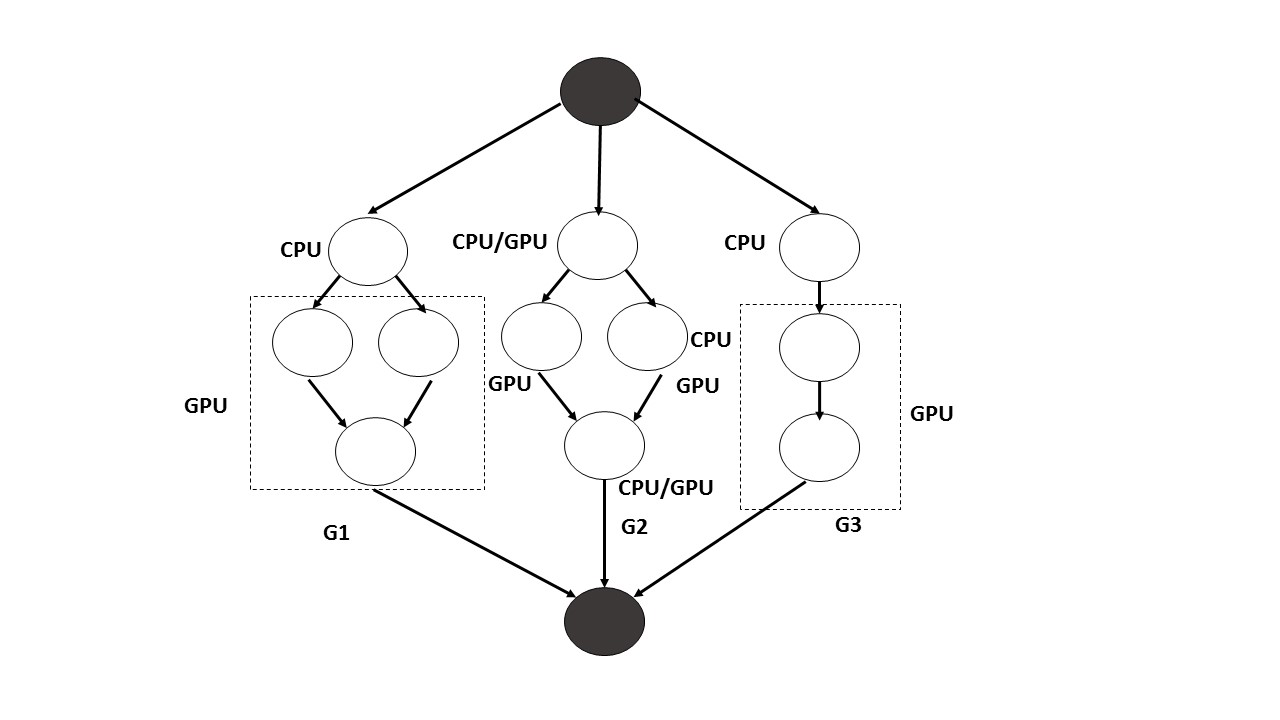


Figure : Application Graph with Kernel level CPU/GPU affinity

The primary objective of this module is to design and analyse graph partitioning algorithms on the composite graph which would ascertain the initial task to device mapping while respecting the precedence constraints in each workload. This entails associating either a task or a subset of tasks with a preferred mode of execution. The modes of execution typically are CPU ,GPU and CPU/GPU (partitioned). This phase uses the static analysis results and CPU/GPU affinity of tasks as calculated in the earlier phase.

The primary motive of the dependency analysis is to ascertain whether it is worthwhile to partition individual tasks across CPU and GPU devices when there is an overhead in terms of data transfer. Eg: In G3, the last two tasks are associated with one GPU. This is because the last task in G3 would process a large amount of data processed by G2 and therefore it makes sense if the data is kept in the GPU in order to avoid the overhead of transferring the data back to the CPU. Once, an initial task to device mapping is ascertained, scheduling policies which take into consideration availability of resources during runtime can be designed.

**Task D: Scheduling policies and Evaluation**

Scheduling policies entail using the information obtained from Task B and Task C and designing runtime algorithms which take into consideration the availability of resources.

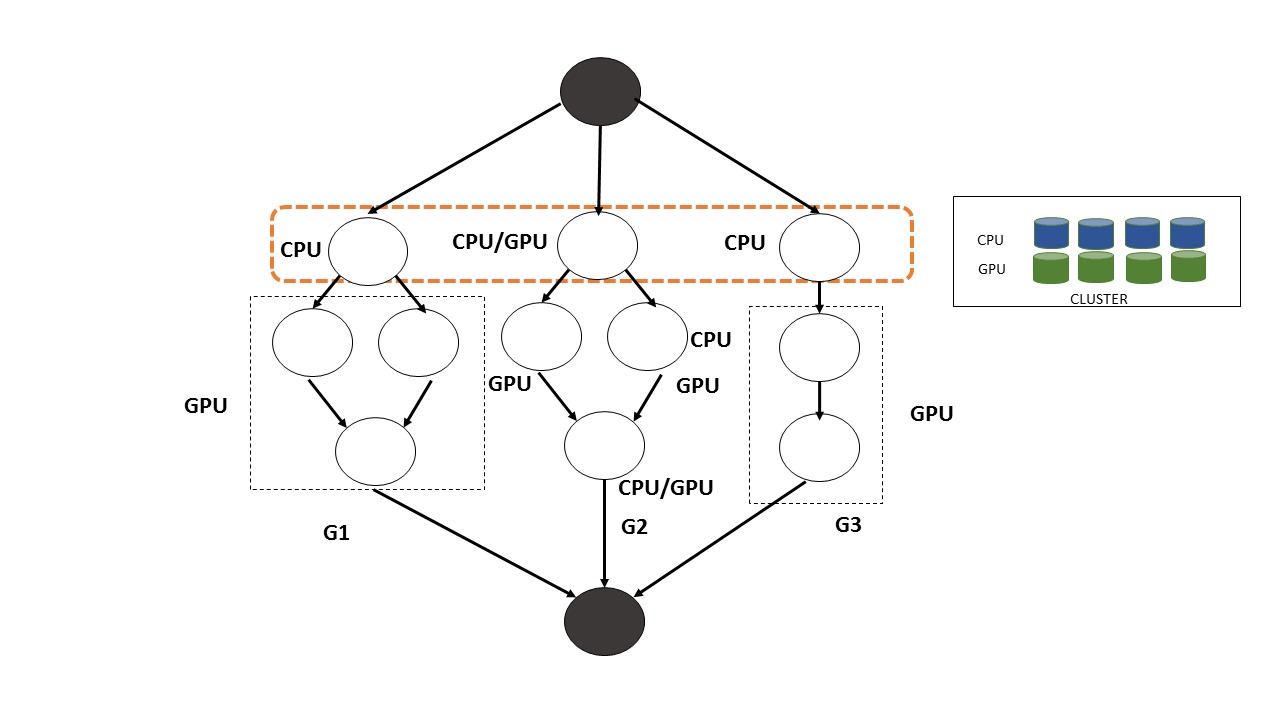


Figure : Hierarchical heterogeneous scheduling

As illustrated in Fig 3. scheduling policies will consider frontiers of tasks in the composite graph and associate devices based on the preferred mode of execution for each task. A frontier typically represents a set of independent tasks either from the same workload or from different workloads. Scheduling policies designed in this context must consider the availability of resources in the system and accordingly make decisions of whether the preferred mode of execution for each task should be respected or not. As for example, for tasks with modes of execution CPU/GPU which are highly biased towards one particular device, it makes sense to change its preferred mode to execution to CPU or GPU so that a device is relinquished when there is a scarcity of resources in the system. Again when there is an excess of resources the scheduling framework should consider scheduling individual tasks to multiple CPU and GPU devices. The main objective of Task D is to design various scheduling heuristics and frontier based scheduling policies and evaluate which one would be best suited for a heterogeneous cluster. A key point in designing heuristics would be to understand which features obtained from the static analysis can be used for making scheduling decisions at runtime. Eg: the total number of floating point operations may serve as an estimate for execution time of a program, the total number of memory operations may serve as an estimate for total memory requirements during scheduling at runtime. A thorough empirical and algorithmic analysis will be made over a variety of OpenCL workloads for evaluating which policy would serve best for a heterogeneous cluster and for what kinds of computation intensive application graph does such a policy provide best results. This shall finally lead to a meta-heuristic solution as an API for which user specified parameters shall automatically initiate different partition-aware scheduling heuristics and choose the best one while keeping in mind the earlier experience for a *similar* application task graph.

* 1. **Time** **Schedule** **of** **activities** **giving** **milestones** **through** **BAR** **diagram.** (Maximum
     + 1. page)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Activity | 1 -­‐ 6  months | 7 -­‐ 12  months | 13 – 18  months | 19 – 24  months | 25 – 30  months | 31 – 36  months |
| Literature Survey | X |  |  |  |  |  |
| Task A | X | X |  |  |  |  |
| Task B |  | X | X |  |  |  |
| Task C |  | X | X |  |  |  |
| Task D |  |  | X | X | X |  |
| Dissemination and Publication |  |  |  |  | X | X |

* 1. **Suggested** **Plan** **of** **action** **for** **utilization** **of** **research** **outcome** **expected** **from** **the** **project.** *(Maximum* *½* *page)*
* Setting up a heterogeneous cluster system with the OpenCL task management API and runtime system created as part of this project. This infrastructure shall be used as part of the institute level central computing infrastructure.
* Proper campus wide training shall be imparted to the students/research scholars who usually work with largescale scientific computing workloads encouraging them to use and benefit from the infrastructure.
* Explore possiblities of using the knowledge and expertise gained through the project for the upcoming national initiative on high performance computing.
  1. **Environmental** **impact** **assessment** **and** **risk** **analysis.** *(Maximum* *½* *page)*

NA

1. **Expertise:**
   1. **Expertise** **available** **with** **the** **investigators** **in** **executing** **the** **project:** *(* *Maximum* *1* *page)*

**(***Professional* *expertise* *existing* *with* *each* *of* *the* *investigators* *in* *terms* *of* *publications,* *Patents* *and* *preliminary* *results,* *to* *execute* *every* *component* *of* *the*

*proposal* *should* *be* *highlighted)*

The work requires expertise in program analysis, code generation and scheduling techniques for high performance architectures. The investigator has steadily published in the area of program analysis and verification. In the last one year he has also published a conference paper on analysis of OpenCL workloads and he has a work on Scheduling of OpenCL workloads under revision in the well known journal of “Parallel Processing Letters”. For the last two years, the investigator has been actively involved with a renowned CPU/GPU provider MNC for development of Speech applications using CPU-GPU partition of OpenCL programs on their architetures.

* 1. **Summary** **of** **roles/responsibilities** **for** **all** **Investigators:**

*(If* *the* *proposal* *contains* *more* *than* *one* *Investigator,* *it* *is* *important* *to* *clearly* *mention* *the* *role* *of* *each* *Investigator* *in* *implementing* *the* *objectives* *of* *the* *proposal.* *The* *Board* *does* *not* *encourage* *Investigators* *who* *do* *not* *have* *specific* *scientific* *role* *in* *the* *proposal)*

|  |  |
| --- | --- |
| Name of the Investigators | Roles/Responsibilities |
| Soumyajit Dey | PI |
|  |  |
|  |  |

* 1. **Key** **publications** **published** **by** **the** **Investigators** **pertaining** **to** **the** **theme** **of** **the** **proposal** **during** **the** **last** **5** **years**
  2. Anirban Ghose, Soumyajit Dey, Pabitra Mitra and Mainak Chaudhuri, "Divergence Aware Automated Partitioning of OpenCL Workloads", India Software Engineering Conference (ISEC) - 2016
  3. Debasmita Lohar, Anudeep Dunaboyina, Dibyendu Das, and Soumyajit Dey, "Failure Estimation of Behavioral Specifications", SETTA 2016
  4. Majid Zamani, Soumyajit Dey, Sajid Mohamed, Pallab Dasgupta and Manuel Mazo Jr., "Scheduling of Controllers' Update-rates for Residual Bandwidth Utilization", FORMATS 2016
  5. Saurav Kumar Ghosh, Akash Mondal, Souradeep Dutta, Aritra Hazra and Soumyajit Dey, "Synthesis of Scheduler Automata Guaranteeing Stability and Reliability of Embedded Control Systems", VDAT 2016
  6. Rajib Lochan Jana, Shashank Kuchibhotla, Soumyajit Dey, Pallab Dasgupta, "Planning Based Guided Reconstruction of Corner Cases in Architectural Validation ", VDAT 2016
  7. M. Becker, S. Mohamed, K. Albers, P.P. Chakrabarti, S. Chakraborty, P. Dasgupta, S. Dey, R. Metta, "Timing Analysis of Safety-Critical Automotive Software: The AUTOSAFE Tool Flow" Asia-Pacific Software Engineering Conference (APSEC) – 2015
  8. Debasmita Lohar, Soumyajit Dey "Integrating Formal Methods with Testing for Reliability Estimation of Component Based Systems", International Symposium on Software Reliability Engineering (ISSRE, Industry Track) – 2015
  9. Rajorshee Raha, Soumyajit Dey, Pallab Dasgupta, "Adaptive Sharing of Sampling Rates among Software Based Controllers", IEEE Multi - Conference on System and Control (MSC) – 2015
  10. Saurav Kumar Ghosh, Aritra Hazra and Soumyajit Dey "RELSPEC: A Framework for Early Reliability Refinement of Embedded Applications", IEEE International Conference on VLSI Design 2015.
  11. Rajorshee Raha, Aritra Hazra, Akash Mondal, Soumyajit Dey, Partha Pratim Chakrabarti and Pallab Dasgupta "Synthesis of Sampling Modes for Adaptive Control", IEEE International Conference on Control System, Computing and Engineering 2014.
  12. Rajorshee Raha, Soumyajit Dey, Pratha Pratim Chakrabarti, Pallab Dasgupta, "Multi-mode Sampling Period Selection for Embedded Real-Time Control" - [POSTER] in Work-in-progress track of 51st Design Automation Conference (DAC) – 2014
  13. Plaban K. Bhowmick, Soumyajit Dey, Abinash Samantaray, Debnath Mukherjee, Prateep Misra "A temporal constraint based planning approach for city tour and travel plan generation", IHCI 2012.
  14. Soumyajit Dey, Dipankar Sarkar, Anupam Basu "A Kleene Algebra of Tagged System Actors for Reasoning about Heterogeneous Embedded Systems", IEEE Transaction Computers , vol 62(10), pp 1917-1931, 2013
  15. Soumyajit Dey, Dipankar Sarkar, Anupam Basu "A Kleene Algebra of Tagged System Actors" - IEEE Embedded Systems Letters, vol 3(1), pp 28-31, 2011
  16. **Bibliography**

1. V. T. Ravi, W. Ma, D. Chiu, and G. Agrawal, “Compiler and Runtime Support for enabling Generalized Reduction Computations on Heterogeneous Parallel Configurations,” in *International Conference on Supercomputing*, 2010, pp. 137–146.
2. M. D. Linderman, J. D. Collins, H. Wang, and T. H. Meng, “Merge: A Programming Model for Heterogeneous Multi-core Systems,” in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2008, pp. 287–296.
3. R. Kaleem, R. Barik, T. Shpeisman, B. T. Lewis, C. Hu, and K. Pingali, “Adaptive Heterogeneous Scheduling for Integrated GPUs,” in *International Conference on Parallel Architectures and Compilation Techniques*, 2014, pp. 151–162.
4. C.-K. Luk, S. Hong, and H. Kim, “Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping,” in *Micro*. IEEE, 2009, pp. 45–55.
5. J. Lee, M. Samadi, Y. Park, and S. Mahlke, “Transparent CPU-GPU Collaboration for Data-parallel Kernels on Heterogeneous Systems,” in *International Conference on Parallel Architectures and Compilation Techniques*, 2013, pp. 245–256.
6. D. Grewe and M. F. O’Boyle, “A Static Task Partitioning Approach for Heterogeneous Systems using OpenCL,” in *International Conference on Compiler Construction*, 2011, pp. 286–305.
7. D. Grewe, Z. Wang, and M. F. O’Boyle, “OpenCL Task Partitioning in the Presence of GPU Contention,” in *Language and Compilers for Parallel Computing*, 2011, pp. 87–101.
8. K. Kofler, I. Grasso, B. Cosenza, and T. Fahringer, “An Automatic InputSensitive Approach for Heterogeneous Task Partitioning,” in *International Conference on Supercomputing*, 2013, pp. 149–160.
9. H. Topcuoglu, S. Hariri, and M. Wu, “Performance-effective and Lowcomplexity Task Scheduling for Heterogeneous Computing,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. 13, no. 3, pp. 260–274, 2002.
10. R. Sakellariou and H. Zhao, “A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems,” in *International Parallel and Distributed Processing Symposium*. IEEE, 2004, p. 111.
11. H. Zhao and R. Sakellariou, “Scheduling Multiple DAGs onto Heterogeneous Systems,” in *International Parallel and Distributed Processing Symposium*. IEEE, 2006, p. 14.
12. J. G. Barbosa and B. Moreira, “Dynamic Scheduling of a Batch of Parallel Task Jobs on Heterogeneous Clusters,” *Parallel Computing*, vol. 37, no. 8, pp. 428–438, 2011.
13. C. Augonnet, S. Thibault, R. Namyst, and P. Wacrenier, “StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures,” *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 187–198, 2011.
14. A. Hugo, A. Guermouche, P. Wacrenier, and R. Namyst, “Composing Multiple StarPU Applications over Heterogeneous Machines: A Supervised Approach,” *International Journal of High Performance Computing Applications*, vol. 28, no. 3, pp. 285–300, 2014.
15. S. Henry, A. Denis, D. Barthou, M. Counilh, and R. Namyst, “Toward OpenCL Automatic Multi-device Support,” in *Euro-Par 2014 Parallel Processing*. Springer, 2014, pp. 776–787.
16. E. Sun, D. Schaa, R. Bagley, N. Rubin, and D. Kaeli, “Enabling Task-level Scheduling on Heterogeneous Platforms,” in *Workshop on General Purpose Processing with Graphics Processing Units*. ACM, 2012, pp. 84–93
17. P. Pandit and R. Govindarajan, “Fluidic Kernels: Cooperative Execution of OpenCL Programs on Multiple Heterogeneous Devices,” in *International Symposium of Code Generation and Optimization*, 2014, p. 273.
18. A. Ghose, S. Dey, P. Mitra, and M. Chaudhuri, “Divergence Aware Automated Partitioning of OpenCL Workloads,” in *India Software Engineering Conference*, pp. 131–135.

1. **List** **of** **facilities** **being** **extended** **by** **parent** **institution(s)** **for** **the** **project** **implementation.**

* 1. **Infrastructural** **Facilities**

|  |  |  |
| --- | --- | --- |
| **Sr.**  **No.** | **Infrastructural** **Facility** | **Yes/No/** **Not** **required** **Full** **or** **sharing** **basis** |
| 1. | Workshop Facility |  |
| 2. | Water & Electricity |  |
| 3. | Laboratory Space/ Furniture | Yes (sharing basis) |
| 4. | Power Generator |  |
| 5. | AC Room or AC |  |
| 6. | Telecommunication including e-mail & fax |  |
| 7. | Transportation |  |
| 8. | Administrative/ Secretarial support |  |
| 9. | Information facilities like Internet/Library |  |
| 10. | Computational facilities |  |
| 11. | Animal/Glass House |  |
| 12. | Any other special facility being provided |  |

* 1. **Equipment** **available** **with** **the** **Institute/** **Group/** **Department/Other** **Institutes** **for** **the** **project:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Equip ment** **availab le** **with** | **Generic** **Name** **of** **Equipment** | **Model,** **Make** **&** **year** **of** **purchase** | **Remarks** **including** **accessories** **available** **and** **current** **usage** **of** **equipment** |
| **PI** **&** **his** |  |  |  |
| **groPI'su** **p** | 2 Workstations | Dell PC, 2014 | NA |
| **DOtherepar** **tment**  **Institute(s)** |  |  |  |

1. **Name** **and** **address** **of** **experts/** **institution** **interested** **in** **the** **subject** **/** **outcome** **of** **the** **project.**

1. Prof. Arobinda Gupta,

Dept of Computer Science and Engineering,

IIT Kharagpur

West Bengal, India, – 721302

1. Prof. Bhargab bhattacharya,

Advanced Computing and Microelectronics Unit

ISI Kolkata

203, B.T. Road - 700108

1. Prof. Krithi Ramamritham

Department of Computer Science  
Indian Institute of Technology Bombay  
Mumbai 400076 India