PDAWL: Profile-based Iterative Dynamic Adaptive WorkLoad Balance on Heterogeneous Architectures

Tongsheng Geng[1], Marcos Amaris[2], Stéphane Zuckerman[3], Alfredo Goldman[2], Guang R. Gao[4], and Jean-Luc Gaudiot[1]

[1] University of California, Irvine, CA, USA
{tgeng, gaudiot} @uci.edu
[2] University of São Paulo, São Paulo, Brazil
{amaris, gold} @ime.usp.br
[3] Laboratoire ETIS, CY Paris Universités, ENSEA, CNRS, France
stephane.zuckerman@ensea.fr
[4] University of Delaware, Delaware,USA
ggao.capsl@gmail.com

Abstract. While High Performance Computing systems are increasingly based on heterogeneous cores, their effectiveness depends on how well the scheduler can allocate workloads onto appropriate computing devices and how communication and computation can be overlapped. With different types of resources integrated into one system, the complexity of the scheduler correspondingly increases. Moreover, for applications with varying problem sizes on different heterogeneous resources, the optimal scheduling approach may vary accordingly. We thus present PDAWL, an event-driven profile-based Iterative Dynamic Adaptive Work-Load balance scheduling approach to dynamically and adaptively adjust workload to efficiently utilize heterogeneous resources. It combines online scheduling (DAWL), which can adaptively adjust workload based on available real time heterogeneous resources, with an offline machine learning (profile-based estimation model) which can build a device-specific communication computation estimation model. Our scheduling approach is tested on control-regular applications, Stencil kernel (based on a Jacobi Algorithm) and Sparse Matrix-Vector Multiplication (SpMV) in an event-driven runtime system. Experimental results show that PDAWL is either on-par or far outperforms whichever yields the best results (CPU or GPU).

Keywords: Heterogeneous many-core computing · workload balance · Adaptive modeling · ML assisted scheduling

1 Introduction and Motivation

As the current TOP500 rankings show, most High-Performance Computing platforms feature heterogeneous hardware resources (CPUs, GPUs, FPGAs, *etc.*) [11]. In the future, the nodes of such platforms are expected to be even more heterogeneous and they will feature side-by-side, fast and slow computing

units mixed with accelerators, I/O nodes, *etc.* Heterogeneous platforms offer the promise of both better energy efficiency and performance. However, this comes at a cost in terms of code development and resource management.

Meanwhile, whole sectors of scientific computing continue to rely on iterative algorithms. In particular, Stencil-based computations are at the core of many essential scientific applications: Stencils are used in image processing algorithms, *e.g.,* convolutions; partial differential equation solvers, Laplacian transforms, or computational fluid dynamics, linear algebra, *etc.* More specifically, the Jacobi iteration method [19] has been proposed to solve sparse triangular systems arising from incomplete Cholesky preconditioning. A diverse set of realistic symmetric positive definite test problems have proved that Jacobi iterations are effective for a large range of problems [4], while block techniques can further help improve the performance. Other kernels are also used in iterative algorithms, such as sparse matrix-vector multiplications (SpMV). As opposed to Stencil (regular computing per row/column), the individual work-items of SpMV exhibit a different computational load profile since the numbers of non-zero elements per row may vary significantly. However, both Stencil and SpMV are control-regular, and the accelerator and host regularly synchronize until the computation is finished. Finding the right workload balance between accelerator and host for both Stencil and SpMV is the challenge.

Our research is based on the following observations: with a few exceptions (detailed in Section 5), most work dealing with accelerators—GPUs—has followed one of two paths: (1) fully offload the most compute-intensive parts of a given application to a GPU, or (2) statically partition the "hot" parts of an application between "CPU-friendly" and "GPU-friendly," *i.e.,* running solely on (respectively) the CPU or the GPU.

This paper presents a novel approach to dynamic scheduling of tasks on heterogeneous systems. It is based on a profile-based machine-learning approach and explores the concept of *co-running*, as defined by Zhang *et al.* [25]: a system has enabled co-running if it runs applications decomposed into tasks capable of running simultaneously on both CPUs and general-purpose accelerators. Our approach, PDAWL, offers the following characteristics:

1. PDAWL is a Profile-based Iterative Dynamic Adaptive WorkLoad balancing algorithm for heterogeneous systems. It can dynamically and adaptively adjust the workload based on the run time situation (dynamic) and hardware platform (static) information. An offline machine learning approach is employed to build the heterogeneous resources performance-workload (communication vs. computation) estimation model based on the analysis of the performance of pure CPUs and GPU. The online scheduler adaptively adjusts the workload allocation based on the run time situation. Combining online and offline information improves flexibility and accuracy.
2. The event-driven characteristics of PDAWL increase flexibility: Multiple levels of parallelism can be employed to improve the flexibility of scheduling.

3. The efficiency of PDAWL is evaluated with control-regular applications: Stencil-based kernels (Jacobi algorithm), featuring regular data process and SpMV-CSR kernels, featuring irregular data process.

The rest of the paper is organized as follows: Section 2 reviews the main concepts of this work; Section 3 describes our methodology; Section 4 focuses on our main experimental results; Section 5, reviews the state of the art Finally, Section 6 concludes this work and presents the planned future work.

2 Background

PDAWL leverages data-driven execution models and their implementation in large-scale heterogeneous machines at the runtime system level.

Runtime System: We extended DARTS [1,18], an implementation of the Codelet Model [26], to include GPU-aware scheduling capabilities. DARTS relies on dataflow-inspired event-driven parallelism. It can implement fine, coarse, or hybrid-grain parallelism as demanded by the scheduling algorithm.

Heterogeneous Computing: This work considers CPU-GPU heterogeneous systems where GPUs are connected to a host machine via a PCI Express (PCIe) bus. Both have different memory address spaces, and data must be explicitly copied back and forth between the two memory pools.

Concurrent Streams on GPUs: CUDA has been augmented with stream-based constructs starting with CUDA v7. This allows the accelerator to efficiently overlap computation and communication with the host.

3 Methodology

In this section, we present the two ways with which we spread the workload between the host and the accelerator. Dynamic adaptive work-load scheduling is discussed in Section 3.1; Section 3.2 presents our complementary profile-based approach. As will be discussed in Section 4.1, we will target two types of control-regular kernels: Stencil, and a sparse matrix-vector product.

3.1 Dynamic Adaptive Work-Load Scheduler

We aim at finding the right load balance that will maximize throughput when CPUs and GPUs execute. Different factors [9] should be taken into account: the accelerator's memory size, the throughput of PCIe, the structure of the memory hierarchy, the utilization of the cache, the respective computing capabilities of CPUs and GPUs *etc.*

Data-regular computations The workload can be decomposed into multiple instances of the same tasks and run on both CPUs and GPU.

$$GPU_{naive} = \text{memcpy}_{Host \to Device} + \frac{\text{Compute}_{Device}}{\text{NumThreads}_{Device}} + \text{memcpy}_{Device \to Host} \quad (1)$$

Eq. 1 models the total GPU execution time. However, it is overly simple.For example, when the various DMA engines are available on modern GPUs, as well as the `Stream` type in CUDA, it is possible to overlap communications and computations. This means that Eq. 1 is a pessimistic/worst-case view of a single GPU's performance. Conversely, it guarantees performance will be maximal if GPU_{naive} is "not too high."

$$CPU_{naive} = \frac{\text{Compute}_{Host}}{\text{NumThreads}_{Host}} \quad (2)$$

Eq. 2 models the CPUs execution time. This model is also rather naïve; While data transfers with the DRAM are not negligible, they take orders of magnitude less time than data transfers on a PCIe bus: they can be neglected. Further more, the performance does not always scale well over multiple cores and nodes. The memory/cache conflicts and synchronization issues incur quite a large overhead. Moreover, HPC processors tend to have a very efficient and aggressive way of prefetching data, which tends to fully hide the latency related to DRAM transfers—especially in the case of consecutive reads or writes. The overlapping data transfers (due to caches and prefetching operations) are included in the execution time.

$$r = \frac{\text{CPU}_{naive}}{\text{GPU}_{naive}} \quad (3)$$

In Eq. 3, r is the ratio between two quantities, GPU_{naive} and CPU_{naive}, computed in Eq. 1 and 2. If $r \gg 1$, then the workload will execute much faster if it is on an accelerator. Hence, most if not all of the computation will be carried on the GPU. On the contrary, if $r \ll 1$, then the amount of data transfers is saturating the PCIe bus when running it on a GPU, and in general, the overall computation is much faster using general-purpose processing elements. When $r \approx 1$, task scheduling must enable co-running, so that both the host and the accelerator are allocated their fair share of the work in order to complete the computation as fast as possible.

Data-irregular computations Irregular computations can lower GPU performance dramatically. To counter this effect, we can extract the irregular parts and assign them to CPUs. The remaining regular workload can then follow the same methodology as with data-regular computations.

The Dynamic Adaptive WorkLoad (DAWL) scheduler DAWL was created to decide what tasks should be scheduled and where to schedule workload to minimize the load imbalance between heterogeneous processing elements. It adjusts the workload distribution on different computing resources based on real-time information and the knowledge we have derived from Eq. 1, 2, and 3. It consists of seven main steps, outlined in Figure 1.

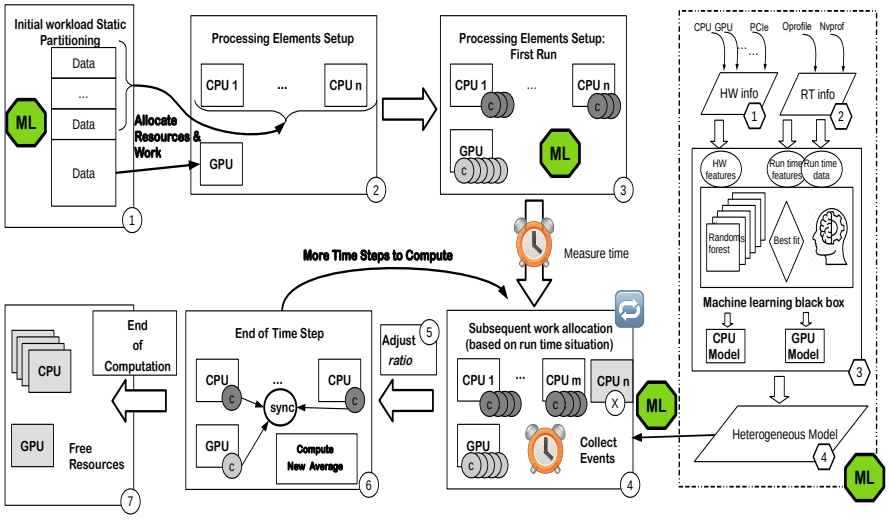

Fig. 1: PDAWL – The Dynamic Adaptive Work-Load scheduling algorithm coupled with Machine Learning. Machine learning occurs in steps 1, 3, and 4 (see the bold polygon ML).

1. Set up the initial workload on all the Processing Elements (PEs), namely CPUs and GPU.
2. Configure PEs according to the given workload. This includes how many CPUs will be put to work, whether the GPU will be also used, how much shared memory (for the host) and global memory (for the accelerator) must be allocated, the number of streams on the GPU, *etc.*
3. Simultaneously run tasks on both CPUs and GPUs, and time each execution for their specific workloads.
4. Check the status of the PEs, estimate the completion time of other devices based on the history timing measurements. Then allocate and run the next workload on available PEs. Repeat until the remaining workload is within 10% of the total workload.
5. Calculate the value of $ratio$, where $ratio = \text{CPU}_{cur}/(\text{CPU}_{cur} + \text{GPU}_{cur})$. CPU_{cur} and GPU_{cur} are the amount of all work finished on CPUs and GPU, respectively. The corresponding GPU ratio is obtained using the same method. The CPUs or the GPU only take $\lfloor ratio \times \text{remaining workload} \rfloor$ amount of work. The remaining workload is dynamically allocated to whichever (set of) PE(s) is available after completing early.
6. Evaluate the load-balance metrics collected during the time step execution, in particular the execution time. Adjust (coarsen) the task granularity based on available PEs and the metrics.
7. Free all resources: PEs and memory.

3.2 Profile-based Machine Learning Estimation Model

Eq. 1 and Eq. 2 are too naïve to model complex situations. The growing variety of hardware devices as well as their combinations, increases the difficulty of building accurate mathematical estimation models. Furthermore, *any* change in the hardware configuration may cause great performance variations and result in a need to rebuild the mathematical model. Moreover, the mathematical model cannot capture the run time situation which is another important factor that affects the accuracy of the performance estimation model.

Considering these factors, we designed a profile-based Machine Learning (ML) approach to reduce the complexity of establishing an estimation model while promoting its accuracy. We call the resulting algorithm PDAWL, short for Profile-based DAWL. It follows four phases, as shown in Figure 1's dotted box:

1. Collect hardware information. Table 1 lists some parameters. In addition to these, the host's cache related and more GPU parameters have also been collected.
2. Collect the application's profile information at run time as training data. The pure CPU and pure GPU performance model are used to predict the heterogeneous (co-running) performance model.
   - CPU: We collect cache and branch related events using `Oprofile` [10]
   - GPU: We used the `gpu-trace` and `api-trace` APIs to collect CUDA run time information and events.
3. Normalize the collected data to a common scale
4. Cluster features: a hierarchical agglomerative clustering algorithm (HAC) is utilized to group similarity features, collected from `Oprofile` and `Nvprof`, and finally obtain 4 to 12 features.
5. Build a pure CPU and pure GPU profile-based estimation model.
   - Run a set of ML algorithms such as linear regression, support vector machine (SVM) and random forest model. Specifically, the linear regression model can be shown in the form: $ln(F(X)) = \sum_{i=1}^{n} w_i \phi_i(x_i)$. Where $\phi_i(x)$ are functions from the set of $x, x^2, x^3, x^4, e^x, lnx, x \cdot lnx$; x_i are features from last cluster step. The logarithmic scale is used to fit the final data $F(X)$. It provides reasonable approximations with the target variable and reduce the non-linearity factors [2]. For SVM, we try the polynomial and Gaussian kernels.
   - Overfitting: we use 10-fold cross validation and L2 regularity to reduce the overfitting problems.
   - Evaluate models: To evaluate how well the model fits the data, a coefficient of determination, $R_{squared}$, is used. $R_{squared} = \frac{\text{Explained variation}}{\text{Total variation}}$, with $0\% \leq R_{squared} \leq 100\%$. 0% indicates the model explains none of the variability of the response data around its mean while 100% says that the model explains all the variability of the response data around its mean.
   - Build an estimation model with the best matched ML algorithm to predict an application's performance on this specific heterogeneous platform.

6. Build a heterogeneous prediction model based on the pure CPU and GPU model. Based on sections 2 and 3.1, and Eq. 1 and 3, the concurrent streams technique will be utilized when the workload is far larger than the GPU's global memory. Then, the huge workload will be split into relatively small concurrent workload tasks. In this case, we can use the small workloads performance information, obtained from the GPU model, to predict the large workload allocation and execution on GPU.

PDAWL results from the combination of the heterogeneous prediction model and DAWL. DAWL can dynamically adjust the workload allocation depending on the run time execution situation. It monitors the actual execution time and compares it with the ML-provided baseline. It then increases the confidence interval for the next tasks and can further compensate for the insufficient off-line ML method. The ML model remains suitable or provides some guidance when the software or the hardware changes. This approach is suitable for all iterative algorithms, as they often require some form of global synchronization. The reason why we combine offline ML with online scheduling methods together is to expect the test applications can satisfy the real-time requirements. If there is no real-time requirement, we can use the online ML (such as stochastic gradient algorithm) to replace offline ML to build prediction model.

4 Algorithm Implementation and Experiment Results

4.1 Experimental Testbed

Table 1: Hardware Platforms

Param. / Machines	CPU Parameters					GPU Parameters				PCIe
	Cores	**Clock**	**Socket**	**L3 Size**	**Mem**	**SM**	**Clock**	**L2 Size**	**Mem**	
Machine1 (K20)	32	2.6 GHz	2	20 MB	64 GB	13	0.71 GHz	1.25 MB	4.8 GB	6.1 GB/s
Machine2 (K20)	40	3 GHz	2	25 MB	256 GB	13	0.71 GHz	1.25 MB	4.8 GB	6.1 GB/s
Machine3 (k40)	8	3.4 GHz	1	8 MB	16 GB	15	0.75 GHz	1.5 MB	12 GB	10.3 GB/s
Machine4 (Titan)	12	3.4 GHz	1	12 MB	31 GB	14	0.88 GHz	1.5 MB	6 GB	11.5 GB/s

We ran the experiments on four heterogeneous systems, as shown in Table 1 and 2. Stencil-based computations and Compressed Row SpMV (SpMV-CSR) were selected to evaluate our DAWL and PDAWL.

Target Applications: Stencil computation To emphasize a worst-case scenario, we used the Stencil kernels described in [8], without ghost cells, which enhances the need for synchronization. Specifically, we focused on kernel: a 5-point 2D Stencil, double precision. We fixed the number of time steps to 30, removing the convergence test at the end of each time step for simplification and to make it more deterministic. Note that the CPU tasks and GPU tasks within one time step were totally independent and that a global barrier was inserted at the end of each iteration. Each experiment was repeated 20 times.

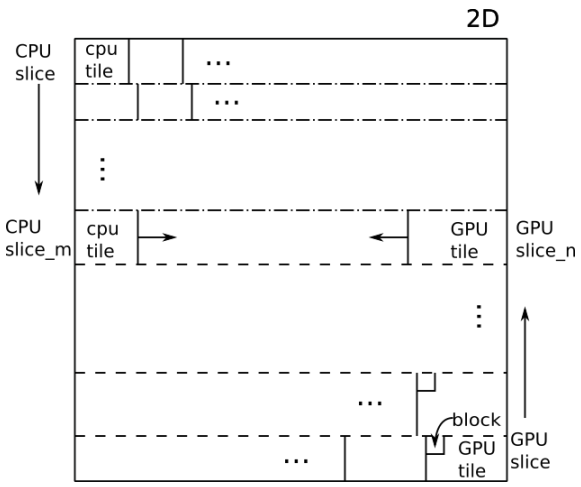

Fig. 2: GPU/CPU hybrid: 2D Stencil slicing and tiling

There are no confidence intervals as the standard deviations were small, the larger one being 5% and the average smaller than 1%.

The partitioning approach we employed for CPU/GPU tasks entails two steps named "Slicing" and "Tiling," respectively, as shown in Figure 2. A static Blocks-Tile size was selected for DAWL. As mentioned in Section 3, different systems architecture can yield different parameters for our ML model. Hence, it tries to find the right match between a given Blocks-Tile size and the number of concurrent streams to issue. This results in near-optimal compute-communication overlap.

Target Applications: SpMV computation We reuse the SHOC benchmark suite's implementation of SpMV-CSR [5], for both the CUDA and CPP sequential versions. We convert the sequential code to parallel code where every CPU core can calculate one or multiple rows. Considering that the number of non-zero elements per row in a sparse matrix may make a significant difference, we call the denser rows (with many more non-zero elements) "irregular rows," whereas the others are deemed "regular rows." Once the irregular rows can be processed separately, the majority regular rows that are left over can be considered at regular computing and can run with our DAWL and PDAWL. To split regular and irregular rows, we build up a co-running model on SHOC SPMV-CSR. More specific steps will be shown in the following:

1. Analyze and evaluate statistic information (see Table 3) to estimate the sparsity degree of the matrix. NNZ is the number of total non-zero elements; μ is the average number of non-zero elements per row; σ is the variance of the number of non-zero elements per row; CV is the coefficient of variation per row; MAX is the maximum number of non-zero elements per row.

2. Build priority groups based on the information. The highest priority level contains the maximum non-zero number per row(s). The majority regular rows construct of the lowest level. In the same level, group members have similar non-zero numbers so they can run parallel. To simplify the model, we statically set the ratio (30%) as the threshold. Top 30% maximum non-zero number per row(s) will be extracted from the matrix and added to CPUs priority groups. The ratio can be learned using ML model, but it will increase training cases and time.
3. Run irregular and regular computations on CPUs and GPU, in parallel. CPUs will proceed from the higher to the lower level and GPU will proceed from the lowest level. Concurrent streams are leveraged here.
4. Synchronize all the computations at the end.

Parameter Space of Our Experiments We used `numactl` to allocate memory in a round-robin fashion and avoid NUMA-related issues. We configure the GPU memory to 2 GB as an example to explain our methodology. We will not show experiments with other GPU memory configuration since the overall trend is the same for all of them.

Table 2: Software Environment

Machines	GCC	CUDA
`Machine1`	v6.2/v8.1	v8.0
`Machine2`	v4.85/v6.2	v8.0
`Machine3`	v5.4	v9.0
`Machine4`	v4.92	v9.1

Table 3: Matrices for SpMV

Name	Dimension	NNZ	μ	σ	cv	MAX
`circuit5M`	5.56 M	59.52 M	10.71	1356.62	126.68	1290501
`eu-2005`	0.86 M	19.24 M	22.30	29.33	1.32	6985
`in-2004`	1.38 M	16.92 M	12.23	37.23	3.04	7753
`FullChip`	2.99 M	26.62 M	8.91	1806.80	202.73	2312481
`kmer_U1a`	67.7 M	138.8 M	2.05	0.37	0.18	35

Matrices Used for our Experiments. We use 50 sparse matrices from the University of Florida Sparse Matrix Collection (UFSMC) [6] to train and 5 matrices 3 to evaluate our DAWL/PDAWL.

4.2 DAWL: Performance Analysis

To comprehensively characterize DAWL, we performed a series of workload performance analysis. We compared the `DARTS-DAWL` performance with `GPU-Only`, `CPU-Seq`, `DARTS-CPU`, and `DARTS-GPU` (see Table 4 for details). `DARTS-DAWL` is the implementation of DAWL on DARTS. Based on the parameters mentioned in section 3.1, `DARTS-DAWL` may run on multiple CPUs or GPU, or be co-running on both CPUs and GPU.

Figure 3 shows the speedup of different variants for the Stencil. `DARTS-GPU` use concurrent streams all time. while, `GPU-Only` use a one-stream method when the problem size is smaller than the GPU memory capacity. This is to avoid superfluous synchronizations between the host and device. Concurrent streams are utilized when the problem size is larger than the GPU's memory capacity to overlap communication and computation.

Table 4: Stencil kernel implementation

Implementation	Illustration
`CPU-Seq`	Sequential c++ code
`GPU-Only`	CUDA code
`DARTS-CPU`	Multi-threads c++ code
`DARTS-GPU`	CUDA code on DARTS (concurrent streams)
`DARTS-DAWL`	DAWL hybrid code on DARTS

Figure 3 demonstrates the validity of our model. With 30 iterations constraints on Stencil kernels, when the workload is less than the available device memory, r from Eq. 3 is far larger than 1, and the application allocates all the workload to the device so as to yield maximum performance. When the workload is larger than the available device memory, it is allocated to both the host and the device. Adding the communication & synchronization costs between two resources types ultimately causes the total performance to drop. The speedup ratios are quite different on different systems, which is due to the differences in hardware. *e.g.,* the GPU of machine 3 is a Tesla-K40, which has a higher clock and memory frequency than Tesla-K20.

`DARTS-DAWL` on machine 3 should run in pure GPU mode based on the analytic model. Here, `DARTS-DAWL` is hard coded to co-running to show our ML approach will improve performance even in the worst case which chooses the wrong target device as shown in Figure 5.

4.3 Profile-based Estimation Model and Result

Section 3.2 shows how we use the performance of pure CPU or GPU to predict that of concurrent CPU-GPU. Our training/validation/test set is split into two, CPU and GPU. The "CPU set" is to build a CPU performance-resource estimation model which can provide the "best" scheduler using minimum computing resources to obtain the maximum performance (shortest execution time) for a specific workload. combining `spread` and `compact` mapping policies, we run experiments with different active CPU threads number (*e.g.* 2, 4, 8, 16...) to obtain the necessary run time information by using `Oprofile`. PDAWL utilizes this information to provide an accurate prediction model even when (for example) some PEs are suddenly turned off because of power issues.

The "GPU set" is used to build a GPU communication-computation overlap model, to estimate data transfer and execution time. In particular, the right Block-Tile size can perfectly overlap communication and computation on a system; and yet, the overlap ratio may be very low on other systems since the available SM, PCIe throughput, *etc.* are different. Specifically, the estimation model consists of: API launching, data transfer between host and device and device computation parts. We run the two version of GPU code, with or without concurrent streams, combining with different Block-Tile size.

The information collected by the runtime helps gather more than two hundreds features for each type of device. HAC was employed to group features. Figure 4 shows one dendrogram aiming at grouping the features in five clusters. One feature with the maximum variance is selected from each cluster. We

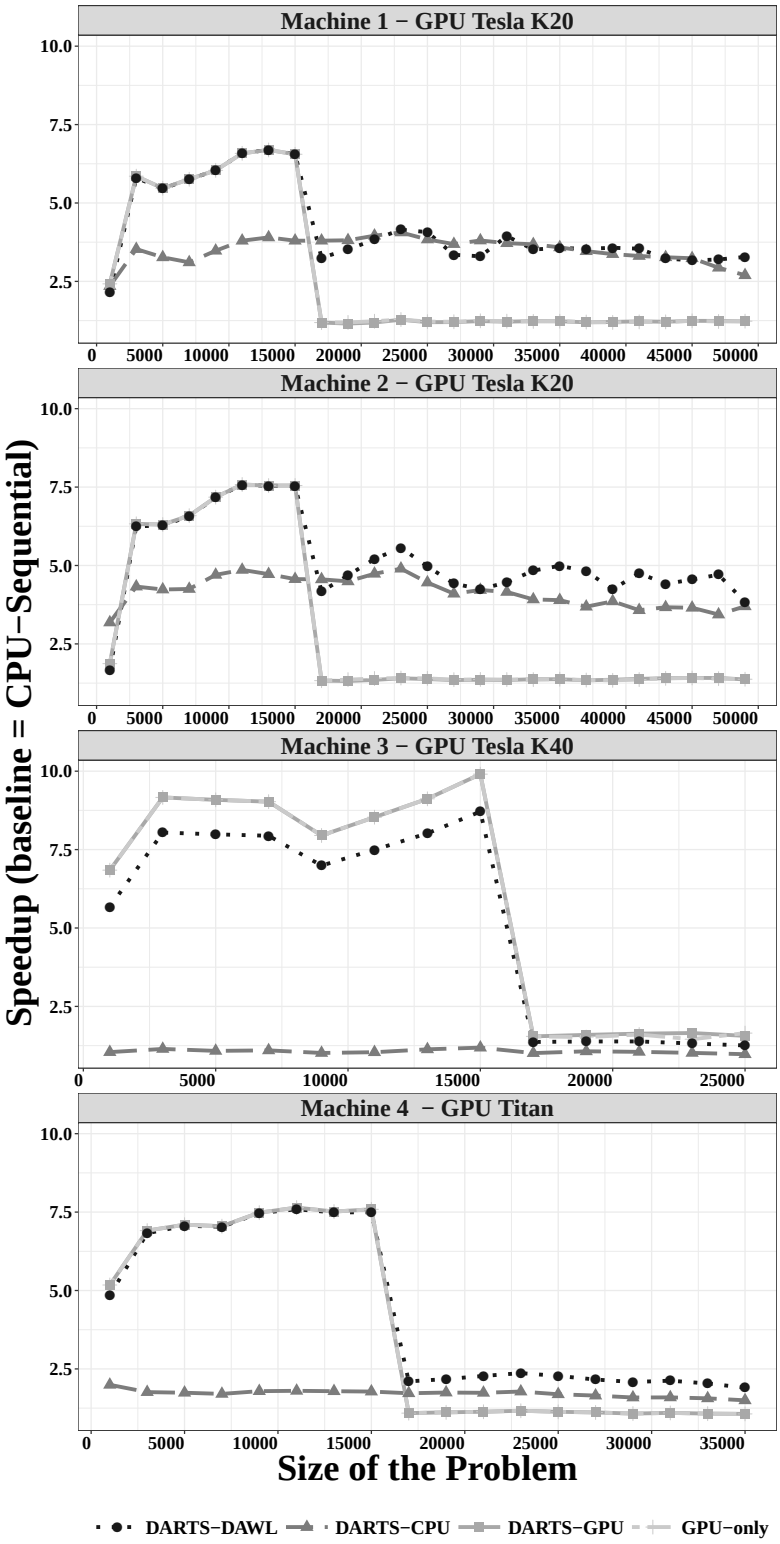

Fig. 3: Stencil: Speedup of the different versions

then selected different group of features running the dendrogram algorithm with different numbers of cluster groups (*e.g.,* 4, 5, 6, $\cdots$, 12).

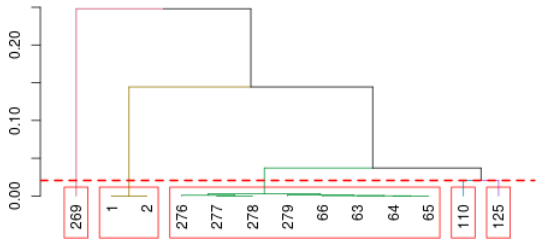

Fig. 4: Stencil: Dendrogram with 5 clusters from features with correlation between the execution times higher than |0.75|

Table 5: Stencil : Mean Absolute Percentage Error

Machines	#1	#2	#3	#4
MAPE	6.43%	7.41%	3.45%	1.68%

After running various ML algorithms, as described in Section 3.2, it turns out that the model that finds the majority of the best matches both for Stencil and SpMV computation is *linear regression:* $0.93 \leq R_{squared} \leq 0.94$. The chosen model may change with different types of applications and hardware configurations. To measure the progress of the learning algorithm the Mean Absolute Percentage Error (MAPE) was used. Table 5 shows the MAPE of the linear model for each machine in the Stencil experiments. The important factors vary with the hardware configuration and cluster group numbers.

Figures 5 show the results for PDAWL. Compared to `DARTS-CPU`, the number of PEs changes with runtime. Our scheduler can reach up to 6× speedups compared to sequential runs, 1.6× speedup compared to the multiple core version, and 4.8× speedup compared to the pure GPU version in the Stencil. Figures 5 shows that one cannot always obtain significant speedups using profiling. This is especially true around *drop points* (drop points are unstable points and are affected by multiple co-running hardware/software conflicts parameters, which our machine learning estimation model did not take into consideration).

Figure 6 compares the SpMV of the five Matrices listed in table 3 on Machine 1. `DARTS-PDAWL` executes 30.5× faster than the GPU version and 1.37× better than the multi CPU version. Our ML algorithm can be further improved by combining online learning Algorithms and neural-networks with our learning estimation model.

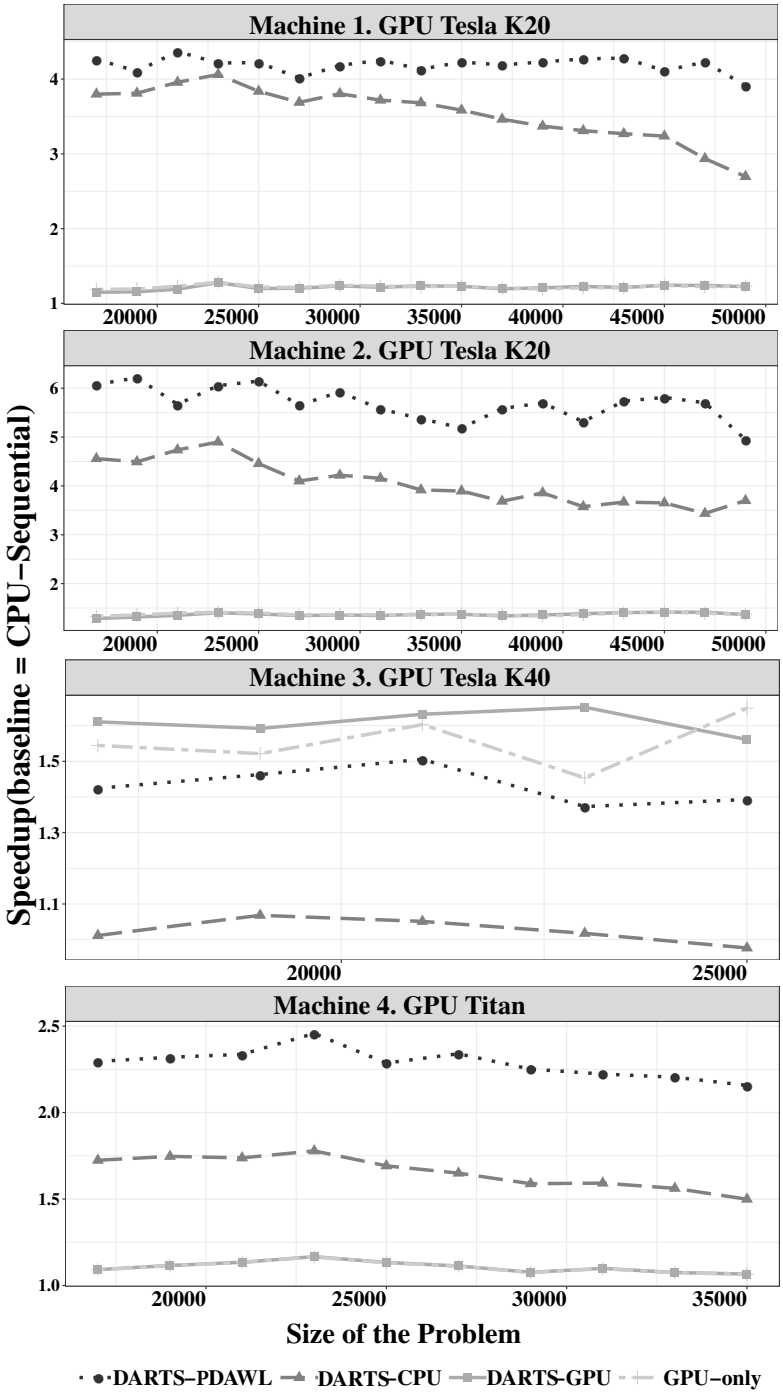

Fig. 5: Stencil: Speedup when matrices are larger than 17K (PDAWL)

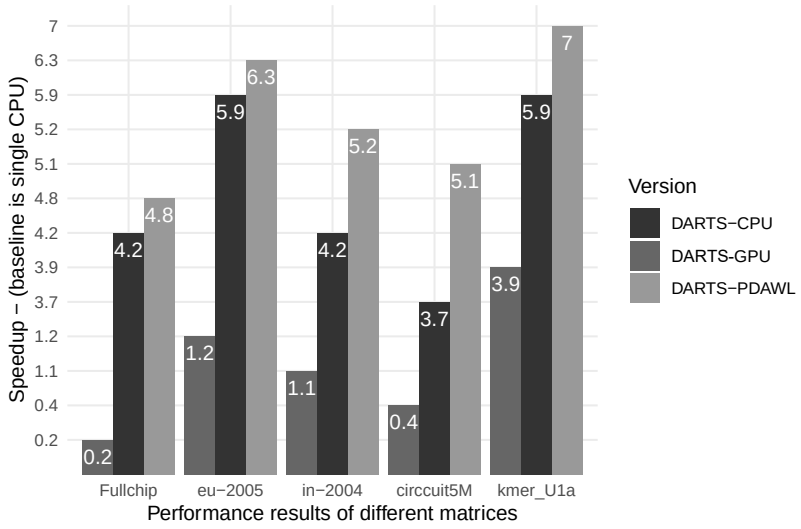

Fig. 6: SpMV Performance(SpeedUP)

5 Related Work

The main challenge of the load-balancing mechanism is to precisely divide workload on processing units. A simple heuristics division approach may actually result in worse performance than a simple uniform division. Machine-learning-based prediction mechanism or/and online profiling-based scheduling algorithms have been deployed to determine the workload partitioning decision on many-core homogeneous/heterogeneous systems.

[12] proposes an empirical adaptive mapping, a fully automatic technique to map computations to processing elements on heterogeneous multiprocessors. [21] utilizes an ML approach to decide whether to parallelize a loop and how to schedule candidates on multi-core platforms. [16,17] proposed two profile-based scheduling algorithms for data-parallel applications in heterogeneous CPU-GPU clusters. The ML approach is utilized to predict the best distribution of data block size among different processing units. [25] performs a series of workload characterization analysis to understand the co-running behaviors on integrated CPU/GPU architecture. The main factors affecting the co-running performance: the architectural differences between CPUs and GPUs and the limited shared memory bandwidth. Based on this information, an ML model can be built to predict coarse-grain workload partitioning on a co-running device before porting the program. [24] proposes a fine-grain workload reshaping approach which combines performance prediction, from an ML model, and partitioning threshold, from an online-turning model, to partition the workload between CPU and GPU on integrated architectures. When the workload is lower than the threshold, it is executed on GPUs. Otherwise, CPUs are employed. [14] and [15] focus on the accelerator sharing control for multiple kernels and propose to use ML to

determine whether to run OpenCL code on GPU or OpenMP code on multi-core CPUs. [22] uses ML to decide whether to merge or separate multi-user OpenCL tasks running on the most suitable devices in a CPU-GPU systems.

Except for architectural differences, communication between CPUs, GPUs, and the memory has a pivotal role. [3, 7, 20, 23, 25] propose an analytical performance model that includes PCIe transfers and overlapping computation and communication. [13] proposes PARTANS, an autotuning framework for CPUs and GPUs to execute Stencil computations over two nodes with multiple GPUs. Data transfer on the PCIe bus play a crucial role to determine the number of GPUs to be utilized. To handle the communication-synchronization problem between CPUs and GPUs,

Most of these are aimed at coarse-grain workload partitioning and loosely synchronized parallel workloads where specific tasks are often run only a specific type of processing element (*e.g.*, CPU or GPU). [24] works for fine-grain partitioning, but static workload partition is inherently rigid. Furthermore, the precision of the ML model determines the efficiency of workload partitioning approach. The hardware change during runtime may have a catastrophic effect on the performance. At the same time, hardware changes during runtime may happen frequently, and as much as half of the CPU cores may be turned off because of power issues.

Our work focuses on synchronization between CPUs and GPUs. Further, the communication between CPUs and GPUs plays a central role in our dynamic scheduling approach. Finally, our approach is neither purely static nor dynamic. We combine the two models: an offline ML model provides us with workload allocation, while DAWL dynamically balances the workload to compensate offline-ML inaccuracies.

6 Conclusions and Future work

We have presented PDAWL, an iterative event-driven scheduling algorithm designed to better load balance tasks in a heterogeneous system. It leverages a profile-based approach based on offline machine learning and an online scheduling approach. The Machine-Learning estimation model can help build an estimation model in a heterogeneous resource context. It consists of a CPU model and a GPU model. We used ML to find the best workload-resource match to improve the CPUs' utilization rate as well as the optimal estimation model to improve GPU performance since building an accurate mathematical general-purpose GPU performance model is nigh-impossible, as the search space is too large. Online event-driven scheduling can make up for the inflexibility of offline machine learning and increase accuracy of scheduling.

Two applications, Stencil and SpMV, have been chosen to evaluate our approach. Experiments with Stencil and SpMV show that PDAWL yields speedups up to 1.6× and 1.37× for a multi-core baseline, 4.8× and 30.5× for pure GPU execution.

Future work includes augmenting our model with power consumption parameters to enrich PDAWL and determining good trade-offs between performance

and power on heterogeneous architectures. We plan on adding Deep Learning algorithms to PDAWL. We will also employ meta learning to reduce training time when run our PDAWL on other configuration Hardware environment.

References

1. Arteaga, J., Zuckerman, S., Gao, G.R.: Generating fine-grain multithreaded applications using a multigrain approach. ACM Trans. Archit. Code Optim. **14**(4), 47:1–47:26 (Dec 2017). https://doi.org/10.1145/3155288, http://doi.acm.org/10.1145/3155288
2. Barnes, B.J., Rountree, B., Lowenthal, D.K., Reeves, J., de Supinski, B., Schulz, M.: A regression-based approach to scalability prediction. In: Proceedings of the 22Nd Annual International Conference on Supercomputing. pp. 368–377. ICS '08, ACM, New York, NY, USA (2008). https://doi.org/10.1145/1375527.1375580
3. Chen, Q., Guo, M.: Contention and locality-aware work-stealing for iterative applications in multi-socket computers. IEEE Transactions on Computers **67**(6), 784–798 (June 2018). https://doi.org/10.1109/TC.2017.2783932
4. Chow, E., Anzt, H., Scott, J., Dongarra, J.: Using jacobi iterations and blocking for solving sparse triangular systems in incomplete factorization preconditioning. Journal of Parallel and Distributed Computing **119**, 219–230 (2018)
5. Danalis, A., Marin, G., McCurdy, C., Meredith, J.S., Roth, P.C., Spafford, K., Tipparaju, V., Vetter, J.S.: The scalable heterogeneous computing (shoc) benchmark suite. In: Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units. pp. 63–74. GPGPU-3, ACM, New York, NY, USA (2010). https://doi.org/10.1145/1735688.1735702, http://doi.acm.org/10.1145/1735688.1735702
6. Davis, T.A., Hu, Y.: The university of florida sparse matrix collection. ACM Trans. Math. Softw. **38**(1), 1:1–1:25 (Dec 2011). https://doi.org/10.1145/2049662.2049663, http://doi.acm.org/10.1145/2049662.2049663
7. García, V., Gomez-Luna, J., Grass, T., Rico, A., Ayguade, E., Pena, A.J.: Evaluating the effect of last-level cache sharing on integrated gpu-cpu systems with heterogeneous applications. In: 2016 IEEE International Symposium on Workload Characterization (IISWC). pp. 1–10 (Sep 2016). https://doi.org/10.1109/IISWC.2016.7581277
8. Geng, T., Zuckerman, S., Monsalve, J., Goldman, A., Habib, S., Gaudiot, J.L., Gao, G.R.: The importance of efficient fine-grain synchronization for many-core systems. In: International Workshop on Languages and Compilers for Parallel Computing. pp. 203–217. Springer (2016)
9. Lee, V.W., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A.D., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., Singhal, R., Dubey, P.: Debunking the 100x gpu vs. cpu myth: An evaluation of throughput computing on cpu and gpu. In: Proceedings of the 37th Annual International Symposium on Computer Architecture. pp. 451–460. ISCA '10, ACM, New York, NY, USA (2010). https://doi.org/10.1145/1815961.1816021, http://doi.acm.org/10.1145/1815961.1816021
10. Levon, J., Elie, P.: Oprofile: A system profiler for linux (2004)
11. List, T.S.: http://www.top500.org (visited on Nov. 2017)

12. Luk, C.K., Hong, S., Kim, H.: Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In: Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture. pp. 45–55. MICRO 42, ACM, New York, NY, USA (2009). https://doi.org/10.1145/1669112.1669121, http://doi.acm.org/10.1145/1669112.1669121
13. Lutz, T., Fensch, C., Cole, M.: Partans: An autotuning framework for stencil computation on multi-gpu systems. ACM Transactions on Architecture and Code Optimization (TACO) **9**(4), 59 (2013)
14. Margiolas, C., O'Boyle, M.F.P.: Portable and transparent software managed scheduling on accelerators for fair resource sharing. In: 2016 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). pp. 82–93 (March 2016)
15. O'Boyle, M.F.P., Wang, Z., Grewe, D.: Portable mapping of data parallel programs to opencl for heterogeneous systems. In: Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). pp. 1–10. CGO '13, IEEE Computer Society, Washington, DC, USA (2013). https://doi.org/10.1109/CGO.2013.6494993, http://dx.doi.org/10.1109/CGO.2013.6494993
16. Sant'Ana, L., Cordeiro, D., Camargo, R.: PLB-HeC: A profile-based load-balancing algorithm for heterogeneous CPU-GPU clusters. In: 2015 IEEE International Conference on Cluster Computing. pp. 96–105 (Sept 2015). https://doi.org/10.1109/CLUSTER.2015.24
17. Sant'Ana, L., Cordeiro, D., de Camargo, R.Y.: Plb-hac: Dynamic load-balancing for heterogeneous accelerator clusters. In: European Conference on Parallel Processing. pp. 197–209. Springer (2019)
18. Suettlerlein, J., Zuckerman, S., Gao, G.R.: An implementation of the codelet model. In: Proceedings of the 19th International Conference on Parallel Processing. pp. 633–644. Euro-Par'13, Springer-Verlag, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40047-6_63, http://dx.doi.org/10.1007/978-3-642-40047-6_63
19. Tribbey, W.: Modern database systems. In: Kim, W. (ed.) Modern Database Systems, chap. Numerical Recipes: The Art of Scientific Computing (3rd Edition) is Written by William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, and Published by Cambridge University Press, ©2007, Hardback, ISBN 978-0-521-88068-8, 1235 Pp., pp. 30–31. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (1995). https://doi.org/10.1145/1874391.187410, http://dx.doi.org/10.1145/1874391.187410
20. Van Craeynest, K., Jaleel, A., Eeckhout, L., Narvaez, P., Emer, J.: Scheduling heterogeneous multi-cores through performance impact estimation (pie). SIGARCH Comput. Archit. News **40**(3), 213–224 (Jun 2012). https://doi.org/10.1145/2366231.2337184, http://doi.acm.org/10.1145/2366231.2337184
21. Wang, Z., Tournavitis, G., Franke, B., O'boyle, M.F.P.: Integrating profile-driven parallelism detection and machine-learning-based mapping. ACM Trans. Archit. Code Optim. **11**(1), 1–26 (Feb 2014). https://doi.org/10.1145/2579561, http://doi.acm.org/10.1145/2579561
22. Wen, Y., O'Boyle, M.F.: Merge or separate?: Multi-job scheduling for opencl kernels on cpu/gpu platforms. In: Proceedings of the General Purpose GPUs. pp. 22–31. GPGPU-10, ACM, New York, NY, USA

(2017). https://doi.org/10.1145/3038228.3038235, http://doi.acm.org/10.1145/3038228.3038235

23. Yang, C., Wang, F., Du, Y., Chen, J., Liu, J., Yi, H., Lu, K.: Adaptive optimization for petascale heterogeneous cpu/gpu computing. In: IEEE International Conference on Cluster Computing. pp. 19–28 (Sept 2010). https://doi.org/10.1109/CLUSTER.2010.12
24. Zhang, F., Wu, B., Zhai, J., He, B., Chen, W.: Finepar: Irregularity-aware fine-grained workload partitioning on integrated architectures. In: 2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). pp. 27–38 (Feb 2017). https://doi.org/10.1109/CGO.2017.7863726
25. Zhang, F., Zhai, J., He, B., Zhang, S., Chen, W.: Understanding co-running behaviors on integrated cpu/gpu architectures. IEEE TPDS **28**(3), 905–918 (March 2017). https://doi.org/10.1109/TPDS.2016.2586074
26. Zuckerman, S., Suetterlein, J., Knauerhase, R., Gao, G.R.: Using a "codelet" program execution model for exascale machines: Position paper. In: Proceedings of the 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era. EXADAPT '11, ACM, New York, NY, USA (2011)