JOURNAL OF BASIC SCIENCE AND ENGINEERING

A STUDY OF GPGPU COMPUTATIONAL DEVELOPMENT FOR EMBEDDED DEVICES

Narayana Murty N[1]
[1]Research scholar, Department of Electronics and Electrical Engineering, Lovely Professional University.
nnmlinux@gmail.com.
Dr. Harjit Singh[2]
[2] Associate Professor, Department of Computer Science and Engineering, Lovely Professional University.
harjit.14952@lpu.co.in

Abstract: Embedded devices are increasingly demanding high computational power for AI and ML applications. This paper explores the architectural, operating system, and software development challenges associated with integrating General-Purpose computing on Graphics Processing Units (GPGPU) into embedded systems. The paper highlights the diverse approaches adopted by vendors to bring GPUs closer to CPUs. It discusses the challenges of real-time computing deadlines due to the lack of hardware preemption in GPGPUs and explores potential software solutions. The limitations of current operating system support for GPGPUs are addressed, emphasizing the lack of control over execution context compared to CPUs. The role of frameworks like CUDA and OpenCL in facilitating GPGPU programming and integrating with other computing devices (FPGAs, TPUs, DSPs) is explored. The paper advocates for OpenCL as a more widely accepted platform for GPGPU computations, contrasting it with CUDA's hardware specificity. The challenges faced by programmers in choosing frameworks and designing applications for diverse hardware and software environments are discussed. The paper concludes by outlining a focus on three areas for further exploration: architectural development of devices, operating system adaptations, and the evolution of frameworks for GPGPU programming in embedded systems.
Keywords: GPGPU, GPU scheduling, open CL

INTRODUCTION:

The Rise of AI and ML in Embedded Devices: Challenges and Solutions

The growing use of artificial intelligence (AI) and machine learning (ML) in embedded devices has created a demand for more powerful processing. This has led to significant changes in the architecture of embedded Systems-on-Chip (SoCs) and their software. This paper explores these architectural changes, operating system challenges, and application adaptations in detail.

The GPU Integration Challenge

Initially, different vendors took various approaches to integrate GPUs closer to host CPUs. These efforts involved fusion architectures and innovative memory access methods. However, the software

and operating system interfaces for these computing devices differ greatly.

Real-Time Constraints and Preemption

Most embedded systems operate with strict deadlines for real-time computations. Traditional General-Purpose computing on GPUs (GPGPUs) lack hardware support for preemption, making context switching a major challenge for real-time tasks. Software solutions are being explored to address this issue.

Limited OS Control over GPUs

Currently, GPGPU fusion devices are not fully utilized within the operating system context. The existing GPU interface resembles other devices, offering limited control over execution. Unlike CPU tasks that benefit from schedulers, I/O management, and memory management, GPU tasks run under a device context without guaranteed execution time. This prevents the operating system from using the GPU for tasks like disk drive encryption.

The Rise of Programming Frameworks

Frameworks like CUDA and OpenCL are instrumental in bringing GPU computing power to general applications. These frameworks not only provide an interface for GPU-CPU integration but also support other computing devices like FPGAs, TPUs, and DSPs. OpenCL is gaining wider acceptance as a unified platform for GPGPU computing, although CUDA remains popular for NVIDIA devices. However, CUDA is not designed for operating system integration or performance optimization – its focus is purely on application portability. This can lead to challenges for programmers who must choose the right framework based on their hardware and application needs.

Focus of the Paper

The following sections delve deeper into GPGPU programming and development considerations. Our study is divided into three main areas: architectural development of devices, operating system context, and the role of different frameworks.

A. GPU Architecture for Enhanced Heterogeneous SoC Performance:

The semiconductor industry is actively tackling challenges associated with heterogeneous SoCs, which integrate different processing cores like CPUs and GPUs on a single chip. This paper explores how a RISC-V based general-purpose GPU with OpenCL support is being implemented to address these issues.The proposed GPU utilizes a Single Instruction, Multiple Threads (SIMT) architecture with minimal extensions to the RISC-V instruction set. The authors, Fares Elsabbagh et al., delve into the implementation details and runtime considerations of this design. Notably, it leverages POCL, an open-source OpenCL implementation that offers flexible compilation using the LLVM compiler.

One of the primary bottlenecks in this context is memory sharing between the CPU and GPU. The paper explores various solutions, including the fine-grained Shared Virtual Memory (SVM) model introduced in OpenCL 2.0. Additionally, the fused architecture of CPU and GPU on the same SoC enables features like cache sharing, which can significantly improve performance. The work of Keitaro Oka on compressed caches for GPUs is also discussed, highlighting how this approach mitigates the negative effects of L1 cache misses and optimizes data utilization.

Furthermore, the paper examines techniques to enhance L1 data cache efficiency among multiple

streaming processors. Jianfei Wang et al. propose extending cache interfaces by incorporating general-purpose hardware caching into the on-chip memory hierarchy of the GPU. This includes the L1.5D cache mechanism, which improves efficiency by duplicating private L1D caches.
Finally, the paper references Yibin Tang and Ying Wang's work on the Mv-net neural network architecture, designed specifically for multicore SoCs. This architecture offers elasticity and contention-aware self-scheduling capabilities to enhance performance in mobile computing systems.In conclusion, this paper explores various approaches for addressing challenges in heterogeneous SoCs with GPUs, paving the way for more efficient and powerful embedded systems.

In recent developments of RISC-V instruction set, the Elsabbagh F, Tine B, Roshan P, et al [2] of Vortex, Propose a new GPU architecture with extended risk v instruction set. The proposed vertex Will allow software developers to run complete open CL Open based Software. In this paper the presented details of vertex GPU architecture and microarchitecture as well as complete open CL softwares stack.

Steepest descent local search (SDLS) algorithm's role in memetic algorithms to address low autocorrelation binary sequences (LABS) pitfalls with traditional mathematical models. the Russek P, Jamro E, Dąbrowska-Boruch A, and Wiatr K [5] try to examine four architectures that address these challenges LABS in loop pipelining, Loop reordering, Dynamic reconfiguration OpenCL platform is used for FPGA development.

The research paper introduces an innovative GPU-based parallel tabu search algorithm (GPTS) for hardware/software co-design, addressing the challenge of balancing solution quality and time in large-scale problems by Hou N, He F, Zhou Y, Chen Y. [6]. GPTS leverages a single GPU kernel for compacting neighborhood and a kernel fusion strategy to reduce GPU global memory accesses, enhancing efficiency. A specifically tailored optimized transfer strategy is proposed to minimize transfer overhead between CPU and GPU, for tabu evaluation on GPUs.

The Oka K, Kawakami S, Tanimoto T, Ono T, Inoue K [8] in this article tackles L1 cache conflicts in GPUs, a major performance hurdle due to limited cache capacity shared by numerous cores. Their solution involves a novel compressed cache architecture that exploits both within-line and between-line data similarity, specifically designed for shared L1 caches in GPUs. This sets it apart from prior work and achieves an impressive 11% performance improvement over existing GPU compression caches. Experiments confirm the effectiveness of this enhanced cache strategy. By reducing cache misses, it significantly boosts GPU performance, which is critical for GPU-reliant applications. Overall, this research offers valuable insights for optimizing GPU cache architectures and presents a promising approach to mitigate L1 cache conflict woes in multi-core processors.

The MV-Net architecture presented by Tang Y, Wang Y, Huawei LI, Xiaowei LI [15] work constitutes a significant advancement for deep neural networks, particularly in the context of mobile computing. The core innovation lies in the dynamic reconfiguration of network propagation paths. This approach allows MV-Net to achieve a commendable trade-off between computational efficiency

and prediction accuracy, paving the way for real-time deep learning on mobile devices. Notably, MV-Net prioritizes performance flexibility by adapting to variations in QoS constraints, output accuracy requirements, and resource contention within multi-tasking environments. This dynamic adaptation ensures consistent, guaranteed QoS, a critical feature for mobile applications with fluctuating performance demands

Wang J, Jiang L, Ke J, Liang X, and Jing N [16] presents a compelling approach to cache architecture with the introduction of the L1.5D cache, particularly for systems utilizing multiple streaming multiprocessors (SMs). The paper proposes replacing private L1D caches with a shared L1.5D cache to tackle data duplication and achieve a larger effective cache size per SM, aiming to enhance overall performance. This paper introduces a novel concept: shareable data aware cache management. This strategy prioritizes retaining valuable, shareable data within the cache by leveraging a lightweight PC-based history table for cache replacement decisions. This approach has the potential to significantly improve performance by mitigating early eviction of crucial data.

The energy consumption in heterogeneous CPU-GPU architectures, particularly within the framework of parallel evolutionary algorithms, is very high. Escobar JJ, Ortega J, Díaz AF, González J, and Damas M [18] proposed a promising approach to optimizing the algorithms. The Multi-Objective Workload Distribution aims to achieve a balance between energy savings and runtime performance, a crucial aspect for efficient parallel computing. The Targeted Application Area is particularly beneficial for bioinformatics, data mining, and other fields with similar parallel computing profiles. The paper delves into both Static Power Management (SPM) and Dynamic Power Management (DPM) techniques within clusters, demonstrating a thorough understanding of energy optimization strategies in heterogeneous systems. The inclusion of expressions for calculating CPU and GPU idle power based on workload allocation provides valuable insights.

Sadrosadati M, Ehsani SB, Falahati H, et al[26] makes significant contributions in the realm of power management for GPU execution units through the introduction of ITAP (Idle-Time-Aware Power Management). The ITAP tackles the shortcomings of prior proposals. which combines power-gating and multiple levels of voltage scaling to reduce static power consumption effectively. It proposes combining ITAP with idleness defragmentation techniques such as pattern-aware scheduling to further enhance static energy savings. The research explores various idle power management modes, implements prediction schemes, and provides accurate estimations of idle periods' lengths.

A new architecture for improving the efficiency of single-instruction multiple-data (SIMD) instructions in multicore processors called SIMD Stealing is introduced by Huang L, Lü Y, Ma S, Xiao N, Wang Z [27]. In single-threaded applications, only one core's SIMD engine is active, limiting performance gains even with multiple cores available. Modifying hardware to transfer SIMD workloads between cores and compiler extensions to identify opportunities for workload sharing improved the performance metric.

This paper Liu W, Ma S, Huang L, Wang Z. [37] proposes a novel approach for designing memory access scheduling on the Network-on-Chip (NoC) side of General-Purpose Graphics Processing Units (GPGPUs) to improve energy efficiency. The proposed scheme targets optimizing memory access patterns within the NoC to reduce energy consumption while maintaining performance. The effectiveness of the scheme is evaluated based on energy efficiency, performance, scalability, and OS support. The research demonstrates the feasibility of the approach and identifies areas for future work in real-world evaluation, adaptability, and heterogeneous architectures.

A. Operating system context (GPU resource management):

In this paper Yang SW, Qiu ZW, Chen YS [7] addresses the performance bottleneck in GPU applications due to data movement from limited physical memory. GPUs are commonly used for high-performance applications, but data swapping significantly impacts performance during memory oversubscription. The proposed solution includes memory contention-aware priority assignment and virtual memory management. The approach aims to reduce performance degradation caused by memory contention.

The Carvalho P, Cruz R, Drummond LMA, et al [10] presents a new approach to analyzing benchmark suites like SHOC, Parboil, and Rodinia. Moving beyond traditional methods is crucial in the rapidly evolving field of GPU computing bydelving into factors like computation types, memory interactions, and hardware utilization . Identifying opportunities for concurrent kernel execution is a significant step toward enhancing GPU efficiency. Simultaneous execution of kernels has the potential to unlock greater performance by better utilizing the hardware's processing resources. Understanding how different kernels interact when running concurrently can indeed pave the way for significant improvements in GPU performance and scalability. Optimized scheduling and resource allocation strategies informed by your research have the potential to drive advancements in GPU technology and its applications across various domains.

Kang JH, Lim JB, and Yu HC of the paper [11] address the challenge of GPU resource shortages in virtualized environments by introducing a novel partial migration technique for GPGPU tasks. This technique enables tasks to be migrated to servers with available GPU resources, thereby reducing the wait time for virtual machines (VMs) to access the GPU. Such a proactive strategy prevents resource shortages from occurring, ensuring uninterrupted operation. This work presents a compelling solution for managing GPU resources in virtualized environments. Through proactive task migration and minimal disruption, the proposed method effectively addresses GPU scarcity while delivering substantial performance gains

The Kiran Kumar M, Abdel-Majeed MR, Annavaram M [13] proposes a method for efficiently running single-GPU code on multiple GPUs. Current code often struggles due to data partitioning and remote memory access. The authors [13] propose a data-aware scheduler that assigns work to GPUs with the most-used data. They track data access patterns and optimize scheduling

based on the repetitive nature of workloads. Additionally, a mechanism migrates data between GPUs to minimize remote access.

The Efficient GPU Cloudification Architecture proposed by Gutiérrez-Aguado J, Claver JM, and Peña-Ortiz R [14] as a transparent and efficient approach to GPU integration in cloud environments. The focus on location transparency is particularly noteworthy, as it fosters flexibility in resource management while mitigating potential user over-utilization. Furthermore, the automation of configuration tasks, encompassing both VM setup and distributed component deployment, significantly improves operational efficiency. The dedicated GPU network is another strong element, guaranteeing optimal performance by isolating GPU data traffic from general VM and management traffic. This work appears to effectively address critical GPU cloudification challenges, presenting substantial advantages for both cloud providers and users alike.

The LLOS Warp Scheduling Policy presented in the paper Do CT, Choi HJ, Chung SW, Kim CH [20],(long-latency operation-based scheduler) addresses the inefficiencies of conventional warp schedulers in GPUs often encountered. The inefficiencies when managing long-latency operations such as global memory loads and stores. LLOS is designed to tackle this issue and enhance overall GPU performance. Performance Gains: The paper showcases, via experimental evidence, that LLOS delivers an average performance boost of 24.4% compared to existing schedulers. This improvement is achieved by LLOS Warp Partitioning divides warps into distinct pools based on their instruction characteristics. Warps unaffected by long-latency operations can proceed while others wait, effectively masking latencies by overlapping execution with other warps.

The Spanning-Tree based algorithm for extracting parallelism in stream-based computing. The Wang G, Wada K, and Yamagiwa S [21] leverage a spanning tree to represent the workflow and automatically determine the execution order for multiple programs running on a distributed system. Map processing units (kernels) to nodes and data streams (I/O) to edges in a spanning tree. Nodes at the same depth can run in parallel if their parent nodes are done (spatial parallelism). The algorithm can be slowed down by waiting for loops to complete. This is addressed through communication pattern optimization during tree generation. Unequal execution times across kernels can impact performance. Load balancing is incorporated to mitigate this issue.

Troodon is a machine learning-based approach for efficient task scheduling in heterogeneous CPU-GPU systems, leading to significant performance improvements introduced by Khalid YN, Aleem M, Ahmed U, Islam MA, Iqbal MA [24]. While processing CPU and GPUs in Traditional scheduling methods might not consider factors like suitability of a task, leading to performance issues. Troodon addresses these challenges by using a machine learning model to classify tasks and a speedup predictor to estimate performance improvement. Troodon leverages an existing scheduling mechanism (E-OSched) for balanced task allocation across CPU and GPU.

In essence, the Kohl N, Hötzer J, Schornbaum F, et al[25] introduces a highly efficient and scalable checkpointing solution for large-scale simulations, boosting their reliability through fault tolerance.

- **Large-Scale Simulations:** This novel scheme creates and recovers snapshots of massive simulations (billions of cells and floating-point values).

- **Efficiency and Scalability:** Checkpoint creation is remarkably fast (seconds) and scales well with system size (up to 260,000 processes). The scheme effectively handles simulations with billions of computational elements.

- **Fault Tolerance:** Recovery algorithms leverage ULFM MPI, enabling runtime recovery from failures and improving simulation reliability.

This method significantly improves the reliability of large-scale simulations by enabling recovery from failures without slowing them down much. (This is a reduction of about 50% from the original passage).

The Navarro A, Corbera F, Rodriguez A, Vilches A, and Asenjo R [29] contributions enhance the state-of-the-art in efficiently executing parallel loops on heterogeneous CPU-GPU chips, offering novel LogFit is a new strategy for dynamically partitioning (dividing) parallel processing tasks (loops) for heterogeneous systems. The LogFit provides Dynamic chunk size for GPU(amount of work assigned to the GPU at runtime), Adjusts the number of loop iterations assigned to CPU cores to avoid imbalances and a code structure for implementing these techniques in parallel loops reduces the programmers efforts. Comparative analysis with existing methodologies demonstrates superior performance and energy savings of up to 57% and 31%, respectively.

Cruz RAQ, Bentes C, Breder B, et al. [34], proposed an Optimisation approach to dynamic reorder the kernel invocation focusing on maximizing resource utilization because the scheduling decisions are taken at the runtime by the hardware itself. so the earring method improved turnaround time and maximum resource utilisation. They use a dynamic programming approach to model the kernel assignments to hardware resources. They evaluated their approach on modern GPUs only.

This paper by Zhao Y, Chen L, Xie G, Zhao J, Ding J. [35] proposes a novel approach to scheduling dependent tasks in physical system simulations. They implement a cellular genetic algorithm (CGA) on a GPU to address scalability limitations of traditional GAs. The GPU-based CGA leverages parallel processing for faster execution and maintains solution quality.The high intensity computational tasks of mechatronics system simulation and modelling using direct acyclic graph (DAG). For scheduling this DAGs cellular genetic gives better results for physical system simulation programs the simulation of genetic algorithms extremely time consuming,.This research validates the feasibility of GPU-based algorithms for scheduling in simulations, demonstrating significant performance improvements and paving the way for further exploration in this area.

Li K. [36] In this paperproposes a scheduling algorithm for parallel tasks with energy and time constraints on multiple manycore processors in a cloud environment. The algorithm aims to

minimize energy consumption while meeting deadlines by considering both energy consumption and time constraints. It utilizes various techniques including post-determination, pre-determination, energy-aware scheduling, and deadline-constrained scheduling. The effectiveness of the algorithm is evaluated based on energy consumption, performance, scalability, and OS support. The research demonstrates the feasibility of the approach and identifies areas for future work in dynamic energy management, fault tolerance, and multi-objective optimization.

Reuther A, Byun C, Arcand W, et al.[39] In this article they discuss an analysis model for job schedulers which run on supercomputers and big data systems. Based on the scheduler latency the defined performance characteristics and id3 article model for designing and measuring a benchmarking system.The proposed scheduling system focuses on three main areas: task scheduling, resource management, and workflow orchestration. Task scheduling involves allocating computing resources and prioritizing job execution based on various factors like dependencies, resource needs, and deadlines. The system utilizes algorithms, optimization techniques, or even machine learning to make informed scheduling decisions. Resource management ensures efficient utilization and fair allocation of resources (CPU cores, memory, storage, network bandwidth) among competing tasks. This may involve mechanisms for provisioning resources, dynamically scaling them based on workload, and monitoring performance to adapt to changing conditions. Finally, workflow orchestration, relevant for Big Data applications, coordinates the execution of multi-stage data processing pipelines. The system provides tools for defining, scheduling, and monitoring complex workflows with interconnected tasks and data dependencies.

Kim H, Patel P, Wang S, (Raj) Rajkumar R. [40] The proposed server-based approach to manage GPU. The GPU server handles the request from other tasks time bound service to the requested tasks and also it has a suspension mechanism to save GPU and CPU cycles. This model also addresses the real time synchronisation-based GPU management. This model is implemented and tested on a real embedded platform.

C. Beyond OpenCL & CUDA: Novel Programming Models for CPU-GPU Co-design in Embedded Systems(framework):

The traditional component-based embedded systems development Struggles with integrating GPU into embedded systems due to Platform specific requirements of that particular component. This problem was addressed by Campeanu G, Carlson J, and Sentilles S. [1] Introducing flexible components GPUs for processing. Flexible component is a Code block written in open CL To generate platform dependent code for CPU and GPUs. this allows the system developers to choose System level requirements and reduces The efforts needed to handle memory allocation and Data transfer between CPU and GPU. It improves the design flexibility and Automatic communication between CPU and GPU.

The embedded system requires CPU GPU and some specialized accelerators; the existing programming models Cuda andOpen CL are difficult to manage the hardware components and

scheduling. To overcome the shortfall Raca V, Mehofer E. [3] introduced ClusterCl Framework. Cluster CL employes varying capabilities requires strategies for partitioning data and handling communication to distribute the load across the various devices such as 1.Automatic Work Partitioning to Allow programmers to choose a strategy, sot that framework handles partitioning and adaptation.. 2. Device-Aware Dispatching Minimizes communication overhead by sending tasks to specific devices. 3. Performance and Energy-aware Distribution 4. Cluster Node Execution Coordination Handles device failures and errors. for Steering for Single/Multi-Kernel Applications:

The existing Sparse Matrix-Vector Multiplication (SpMV) on computing platforms that combine CPUs and GPUs are limited to single CPU and GPU. Benatia A, Ji W, Wang Y, Shi F. [4] solution for workload partitioning and mapping on heterogeneous CPU-GPU systems with potentially more complex configurations (multiple CPUs and/or GPUs). It leverages machine learning models to predict SpMV performance for different submatrices under various sparse formats on each processing unit such as Structure-based, Graph-based and Cost-based partitioning.

In this paper, Nozal R, Bosque JL, and Beivide. R[9] introduces EngineCL, a novel OpenCL-based runtime system designed to streamline the co-execution of large, data-parallel kernels across all devices within a heterogeneous computing system. EngineCL takes care of the heavy lifting, including device management, memory allocation on disparate memory spaces, and workload scheduling. This layered API offers a user-friendly experience for developers while still allowing for performance optimization. The capabilities of EngineCL have been validated on two compute nodes, each containing a mix of six devices with various architectures. This successful validation demonstrates EngineCL's effectiveness in handling the complexities of heterogeneous systems.

This research by Beheshti Roui M, Shekofteh SK, Noori H, and Harati A.[12]proposes a revolutionary framework for GPU stream scheduling. It predicts the best stream configuration for two data streams, eliminating exhaustive testing and boosting efficiency. The framework defines a performance model and scheduler: the model estimates execution time for concurrent program sections, and the scheduler uses these estimates to find the optimal stream set up for peak performance. Remarkably, even with a 33% error in the model, the scheduler achieves 100% precision in predicting the best stream sets. This not only ensures reliability but also saves significant time by avoiding testing all possibilities, making stream scheduling practical for real-world applications.

A novel machine-learning approach for dynamic policy selection for GPU warp scheduling was introduced by Chiou LY, Yang TH, Syu JT, Chang CP, and Chang YJ [17]. Unlike prior studies that relied on static analysis, this approach injects adaptability and intelligence into policy selection, effectively addressing the challenge of diverse application requirements on GPUs.The proposed approach's ability to maintain performance comparable to the best static policy across various applications. This finding highlights the potential of machine learning for intelligent policy selection in GPU warp scheduling, ensuring near-optimal performance regardless of the specific application characteristics.

In this paper, Liao SW, Kuang SY, Kao CL, Tu CH. [22] propose a new framework for

Halide, a programming model for mixed CPU-GPU systems. The framework tackles challenges like data movement and workload balancing, allowing programmers to better leverage both processors. It achieves significant performance gains by optimizing CPU code and minimizing data copies. The paper analyzes two image processing programs using the framework, demonstrating its effectiveness in utilizing both CPU and GPU for faster execution. Overall, this research offers a valuable tool for optimizing programs on heterogeneous computing systems.

An extension to OmpSs, a parallel programming framework, to simplify development for multi-unit systems (CPUs and GPUs) was proposed by Pérez B, Stafford E, Bosque JL, et al [23]. OmpSs allows parallel programming but struggles with running a single program on multiple devices at once. The extension tackles this by enabling concurrent execution of a single program across devices in a single node. It solves data distribution and load balancing challenges for these heterogeneous systems. An Auto-Tune feature dynamically optimizes performance based on hardware and applications. This extension simplifies heterogeneous system programming while maximizing performance and resource utilization.

This Hartmann C, Margull U [28] introduces GPUart, a software framework enabling real-time scheduling for GPUs in embedded systems. Traditional GPUs lack real-time capabilities like task preemption, hindering their use in critical areas like autonomous vehicles that require guaranteed timely processing. GPUart archives it by

- Software-only approach: Achieves preemption without hardware or driver modifications.
- Fixed preemption points: Allows task interruption at specific points within thread blocks.
- Portable resource management: Enables scheduling tasks on GPUs using gang scheduling algorithms.
- Scheduling policies: Supports Gang-EDF and Gang-FTP for task scheduling.

This added the 0.28% minimal memory overhead, reduced the worst-case response time by 221 times and achieved. guaranteed task deadlines are met.

The presented KOCL **Framework** implements the runtime Framework for operating system kernel which in turn communicates the open CL Framework in the application layer introduced by Tu C-H, Lin T-S. [30]. The framework, designed to facilitate general-purpose in-kernel accelerations using different types of processors. This framework provides high-level programming interfaces for Linux kernel module developers to offload compute-intensive tasks on various hardware accelerators without managing platform-specific computing and memory resources.

- simplifies programming efforts by offering platform management and memory models that systematically manage heterogeneous hardware resources. This simplification enables developers to offload tasks onto different hardware accelerators with ease, enhancing the performance of user-space applications.
- The framework supports one- and zero-copy data-buffering schemes, ensuring high performance on platforms with diverse memory architectures. This feature allows offloaded tasks to deliver optimal performance regardless of the underlying hardware setup.Configurations.

This brings Mini security overheads that it calls between the openCL and OS kernel. Very few applications require kernel (OS) execution. Mainly network data applications and data drives with encrypted data. the encryption and decryption algorithms can be executed on the GPUs in a better way, So device drivers and Network Stack of the operating system should provide a means to use GPU for program flow requiring higher performance. Currently GPGPU systems utilise the job submission to their schedule as only from the application layer, but the discrete jobs submitted by kernel components required higher priority and software preemption.

A workload-aware GPU multiprocessing framework designed for modern GPUs, enabling concurrent kernels from different processes to share GPU devices efficiently called Slate was introduced by Allen T, Feng X, and Ge R [31]. Slate aims to address the issue of inefficient GPU utilization by implementing advanced kernel scheduling and resource sharing techniques, promoting workload-aware runtime design for better performance. Slate offers an open-source alternative to NVIDIA MPS, providing a cost-effective solution for GPU multiprocessing research and enhancing GPU utilization in real-world applications.

- Selects kernels at runtime that have complementary resource needs to minimize interference.
- Dynamically adjusts kernel sizes for efficient resource sharing and reclaiming.
- Manages data transfer and synchronization overhead.

UHCL-darknet, Liao L, Li K, Li K, Yang C, Tian Q. [38] proposed in this paper, is an OpenCL-based deep neural network framework designed for heterogeneous computing clusters. It aims to efficiently run deep learning models across various hardware architectures found in these clusters. The evaluation focuses on scalability, performance, resource management, and OS support. UHCL-darknet utilizes parallelization, task scheduling, and memory management techniques to optimize DNN execution. The paper identifies areas for further research, including optimization for diverse hardware configurations, dynamic resource allocation, and improving energy efficiency in heterogeneous clusters.

Yu C, Bai Y, Yang H, et al. [41], presented SMGuard managing GPU resources across multiple co-location applications. and also the proposed CapSM, a capacity based GPU resource model which provides fine grained granularity among GPU applications. SMGuard supports evicting red blocks to release the resources and remapping into uncompleted thread blocks and also avoids relaunch of Kernels. This also reduces the necessity of preemption of kernels.

Conclusion: Opportunities for Job Scheduling in Fused CPU-GPU Systems

This paper highlights the limitations of existing job scheduling algorithms for heterogeneous systems with tightly-coupled CPU-GPU architectures. While numerous frameworks and schedulers cater to multi-core systems with discrete GPUs, there's a gap in solutions designed for fused CPU-GPU SoCs with shared memory and caches.

This work proposes key characteristics and requirements for effective job scheduling in such systems.

We argue against solely GPU-based execution, especially when the CPU remains idle. A lightweight scheduling algorithm is needed to leverage both CPU and GPU capabilities, considering the CPU's higher clock speed and potential benefits.

The tight integration of CPU and GPU in fused SoCs opens doors for co-execution of applications. While existing work by Kali explores machine learning for job scheduling, it requires modification to dynamically accept jobs and optimize resource utilization.

Real-time embedded systems typically rely on priority and preemption algorithms. However, hardware preemption in GPUs often incurs high overhead. This work suggests exploring software-based preemption by introducing scheduling points during program execution, either at compile time or job submission. OpenCL applications, while designed for portability across CPU and GPU, often underperform compared to CPU-specific implementations. Modern CPUs equipped with SIMD units can achieve high throughput for certain applications. Developing applications to leverage both CPU and GPU SIMD capabilities can lead to better performance and resource utilization. Additionally, the scheduler should consider data size when making execution decisions. Smaller datasets may benefit more from CPU execution due to memory transfer overhead on the GPU.

The presented KOCL framework demonstrates a runtime framework within the operating system kernel that interacts with the OpenCL framework in the application layer. This offers minimal security overhead compared to traditional OS-to-OpenCL communication. While few applications require in-kernel execution, network data processing and encrypted data drives are examples where GPUs can excel for cryptographic algorithms. Operating system device drivers and network stacks should explore enabling GPU utilization for performance-critical tasks.

Currently, GPGPU systems primarily rely on application-layer job submissions for scheduling. However, discrete jobs submitted by kernel components often require higher priority and software preemption capabilities.

In conclusion, this work identifies a shift in focus within the GPGPU field, with significant advancements happening in software frameworks like OpenCL being implemented across diverse hardware platforms. Limited research has explored the operating system component, as evidenced by frameworks like KOCL. While substantial progress has been made in hardware through fused architectures, shared virtual memory, and shared caches, software preemption techniques are proving successful in commercial applications like voice and facial recognition, medical image processing, and autonomous vehicle control. This paper highlights the need for further research in job scheduling algorithms specifically designed to exploit the unique capabilities and constraints of fused CPU-GPU SoCs.

REFERENCES:

1. Gabriel Campeanu, Jan Carlson, Severine Sentilles, Component-based ´ Development of Embedded Systems with GPUs, The Journal of Systems & Software (2019), doi: https://doi.org/10.1016/j.jss.2019.110488

2. Elsabbagh F, Tine B, Roshan P, et al, Vortex: OpenCL Compatible RISC-V GPGPU, arXiv (2020)
3. Valon Raca1 · Eduard Mehofer cluster CL: comprehensive support for multi-kernel data-parallel applications in heterogeneous asymmetric clusters, The Journal of Supercomputing, (2020), https://doi.org/10.1007/s11227-020-03234-w.
4. Akrem Benatia , Weixing Ji , Yizhuo Wang and Feng Shi Sparse matrix partitioning for optimizing SpMV on CPU-GPU, heterogeneous platforms, The International Journal of High Performance Computing Applications,(2019), 1–15
5. Paweł Russek, Ernest Jamro, Agnieszka D˛abrowska- Boruch and Kazimierz Wiatr, A study of the loops control for reconfigurable computing with OpenCL in the LABS local search problem, The International Journal of High-Performance Computing Applications, (2020), 1–12
6. Hou N, He F, Zhou Y, Chen Y An efficient GPU-based parallel tabu search algorithm for hardware/software co-design, Front. Comput. Sci., 2020, 14(5): 145316.
7. Yang SW, Qiu ZW, Chen YS, GPU swap-aware scheduler: Virtual memory management for GPU applications, ACM, 2020, https://doi.org/10.1145/3341105.3373866
8. Oka K, Kawakami S, Tanimoto T, Ono T, Inoue K, Enhancing a manycore-oriented compressed cache for GPGPU, ACM, 2020, MORC, https://doi.org/10.1145/3368474.3368491
9. Raúl Nozal ∗, Jose Luis Bosque, Ramon Beivide, EngineCL: Usability and Performance in Heterogeneous Computing, Future Generation Computer Systems, Future Generation Computer Systems 107 (2020) 522–537
10. Carvalho P, Cruz R, Drummond LMA, et al Kernel concurrency opportunities based on GPU benchmarks characterization, Cluster Computing, https://doi.org/10.1007/s10586-018-02901.
11. Kang JH, Lim JB, Yu HC Partial migration technique for GPGPU tasks to Prevent GPU Memory Starvation in RPC-based GPU Virtualization, *Softw: Pract Exper*. 2020;1–25.
12. Beheshti Roui M, Shekofteh SK, Noori H, Harati A Efficient Scheduling of Streams on GPGPUs, The Journal of Supercomputing, https://doi.org/10.1007/s11227-020-03209-x
13. M. Kiran Kumar, et al., Efficient automatic parallelization of a single GPU program for a multiple GPU system, Integration, the VLSI Journal, (2018) https://doi.org/10.1016/j.vlsi.2018.12.006
14. Gutiérrez-Aguado J, Claver JM, Peña-Ortiz R, Toward a transparent and efficient GPU cloudification architecture, The Journal of Supercomputing https://doi.org/10.1007/s11227-018-2720-z
15. Tang Y, Wang Y, Huawei LI, Xiaowei LI MV-Net: Toward real-time deep learning on mobile GPGPU systems, ACM Journal on Emerging Technologies in Computing Systems, Vol. 15, No. 4, Article 35, October 2019.

16. Wang J, Jiang L, Ke J, Liang X, Jing N A sharing-aware L1.5D cache for data reuse in GPGPUs, IEEJ Transactions on Electrical and Electronic Engineering, *IEEJ Trans* 2019; **14**: 862–869
17. Chiou LY, Yang TH, Syu JT, Chang CP, Chang YJ Intelligent Policy Selection for GPU Warp Scheduler, 2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)
18. Escobar JJ, Ortega J, Díaz AF, González J, Damas M, Energy-aware load balancing of parallel evolutionary algorithms with heavy fitness functions in heterogeneous CPU-GPU architectures, *Concurrency Computat Pract Exper*. 2018; e4688.
19. M. Khairy, A.G. Wassal and M. Zahran, A survey of architectural approaches for improving GPGPU performance, programmability and heterogeneity, Journal of Parallel and Distributed Computing (2019), https://doi.org/10.1016/j.jpdc.2018.11.012
20. Do CT, Choi HJ, Chung SW, Kim CH A novel warp scheduling scheme considering long-latency operations for high-performance GPUs, The Journal of Supercomputing https://doi.org/10.1007/s11227-019-03091-2
21. Wang G, Wada K, Yamagiwa S, Optimization in the parallelism extraction algorithm with spanning tree on a multi-GPU environment, IEEJ Transactions on Electrical and Electronic Engineering *Ieej Trans* 2019; **14**: 862–869
22. Liao SW, Kuang SY, Kao CL, Tu CH, A Halide-based Synergistic Computing Framework for Heterogeneous Systems, 47th International Conference on Parallel Processing, August 13–16, 2018, Eugene, OR, USA. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3225058.3225107
23. Pérez B, Stafford E, Bosque JL, et al, Auto-tuned OpenCL kernel co-execution in OmpSs for heterogeneous systems, 0743-7315, 2018 Elsevier, https://doi.org/10.1016/j.jpdc.2018.11.001
24. Khalid YN, Aleem M, Ahmed U, Islam MA, Iqbal MA, Troodon: A machine-learning based load-balancing application scheduler for CPU–GPU system, Journal of Parallel and Distributed Computing 132 (2019) 79–94
25. Kohl N, Hötzer J, Schornbaum F, et al, A scalable and extensible checkpointing scheme for massively parallel simulations, The International Journal of High-Performance Computing Applications, 2019, Vol. 33(4) 571–589
26. Sadrosadati M, Ehsani SB, Falahati H, et al ITAP: Idle-Time-Aware Power Management for GPU Execution Units, ACM Transactions on Architecture and Code Optimization, Vol. 16, No. 1, Article 3, February 2019.
27. Huang L, Lü Y, Ma S, Xiao N, Wang Z, SIMD stealing: Architectural support for efficient data parallel execution on multicores, Microprocessors and Microsystems 65 (2019) 136–147
28. Christoph Hartmann, Ulrich Margull, GPUart - An Application-Based Limited Preemptive GPU Real-Time Scheduler for Embedded Systems, Journal of Systems Architecture (2018), doi: https://doi.org/10.1016/j.sysarc.2018.10.005

29. Navarro A, Corbera F, Rodriguez A, Vilches A, Asenjo R, Heterogeneous parallel, for Template for CPU–GPU Chips, Int J Parallel Prog,https://doi.org/10.1007/s10766-018-0555-0
30. Tu C-H, Lin T-S, Augmenting Operating Systems with OpenCL Accelerators, *ACM, Trans. Des. Autom. Electron. Syst.* 24, 3, Article 30 (March 2019), 29 pages.
31. Allen T, Feng X, Ge R Slate, Enabling workload-aware efficient multiprocessing for modern GPGPUs, 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), DOI 10.1109/IPDPS.2019.00035
32. Nozal R, Perez B, Bosque JL, Beivide R, Load balancing in a heterogeneous world: CPU-Xeon Phi co-execution of data-parallel kernels, J Supercomput,https://doi.org/10.1007/s11227-018-2318-5
33. Amaris M, Lucarelli G, Mommessin C, Trystram D, Generic algorithms for scheduling applications on heterogeneous platforms, *Concurrency Computat Pract Exper*. 2018;e4647.
34. Cruz RAQ, Bentes C, Breder B, et al, Maximizing the GPU resource usage by reordering concurrent kernels submission, *Concurrency Computat Pract Exper*. 2018;e4409.
35. Zhao Y, Chen L, Xie G, Zhao J, Ding J, GPU implementation of a cellular genetic algorithm for scheduling dependent tasks of physical system simulation programs, J Comb Optim DOI 10.1007/s10878-016-0007-y
36. K. Li, Scheduling parallel tasks with energy and time constraints on multiple manycore processors in a cloud computing environment, Future Generation, Computer Systems (2017), http://dx.doi.org/10.1016/j.future.2017.01.010
37. Liu W, Ma S, Huang L, Wang Z, The Design of NoC-Side Memory Access Scheduling for Energy-Efficient GPGPUs, Int J Parallel Prog (2018) 46:722–735 https://doi.org/10.1007/s10766-017-0521-2
38. Liao L, Li K, Li K, Yang C, Tian Q, UHCL-darknet: An OpenCL-based deep neural network framework for heterogeneous multi-/many-core clusters, J Sign Process Syst, DOI 10.1007/s11265-017-1283-1
39. A. Reuther, C. Byun, W. Arcand, D. Bestor, B. Bergeron, M. Hubbell, M. Jones, P. Michaleas, A. Prout, A. Rosa, J. Kepner, Scalable system scheduling for HPC and big data, J. Parallel Distrib. Comput. (2017), http://dx.doi.org/10.1016/j.jpdc.2017.06.009
40. Kim H, Patel P, Wang S, (Raj) Rajkumar R, A server-based approach for predictable GPU access with improved analysis, Journal of Systems Architecture 88 (2018) 97–109.
41. Yu C, Bai Y, Yang H, et al SMGuard: A Flexible and Fine-Grained Resource Management Framework for GPUs, DOI 10.1109/TPDS.2018.2848621, IEEE