Optimizing Task-Driven Offloading in LLVM

Jan Kraus
kraus@itc.rwth-aachen.de
Chair for High-Performance Computing i12, RWTH Aachen University
Aachen, Germany

Joachim Jenke
jenke@itc.rwth-aachen.de
Chair for High-Performance Computing i12, RWTH Aachen University
Aachen, Germany

Christian Terboven
terboven@itc.rwth-aachen.de
Chair for High-Performance Computing i12, RWTH Aachen University
Aachen, Germany

Abstract

We investigate an inefficiency in the LLVM OpenMP runtime related to accelerator offloading. The current implementation manages asynchronous GPU tasks by polling *async handles*, which introduces CPU overhead. We propose replacing this polling model with an event-driven approach that detaches target tasks by default. In our design, each asynchronous task is associated with an event that is fulfilled once the GPU kernel completes, allowing the task to yield execution. This eliminates repeated polling and reduces scheduling overhead. We implemented this mechanism using existing features in the LLVM OpenMP runtime, relying on a host callback function provided by CUDA. Experiments on NVIDIA H100 GPUs show runtime improvements of up to 75% for independent tasks once matrix sizes exceed 128×128, with benefits appearing at even smaller sizes when task dependencies are present. For large kernels, the effect diminishes as execution time dominates.

CCS Concepts: • **Software and its engineering** → **Runtime environments**; • **Computer systems organization** → *Heterogeneous (hybrid) systems*; • **Computing methodologies** → Parallel computing methodologies.

ACM Reference Format:
Jan Kraus, Joachim Jenke, and Christian Terboven. 2025. Optimizing Task-Driven Offloading in LLVM. In *Proceedings of The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '25).* ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

With the introduction of target tasks in OpenMP 4.5 [5], it has become possible to integrate accelerator offloading into the general tasking model. Programmers can write code that mixes CPU and GPU work with minimal boilerplate, relying on the runtime to orchestrate execution. However, we observed a shortcoming in the current LLVM [3] implementation. The runtime communicates the progress of asynchronous offloaded tasks by repeatedly polling an *async handle*. This polling-based design may lead to unnecessary CPU overhead between tasks that are intended to run on an accelerator.

In this paper, we explore whether the polling model could be replaced with a mechanism that avoids overhead, without compromising correctness or introducing undue complexity using features already supported by the LLVM OpenMP runtime. The idea we pursue is to detach target tasks by default, allowing them to yield execution. This preserves the logical structure of asynchronous task execution while giving the runtime faster feedback on a kernels completion.

2 Workflow

The easiest way to understand our approach is to contrast it with the status quo.

Polling. In the current workflow, a target task T is bound to a handle H representing the state of the GPU kernel. The runtime enqueues T and, on each scheduling pass, checks whether H indicates completion. If the kernel is still running, the task is re-enqueued and polled again in the next cycle. This continues until the handle signals completion, at which point the task's resources are released and H is deleted. This design may adds significant scheduling overhead in between target tasks.

Detaching. In our *detaching* workflow, we remove the handle and replace it with an event-based model. When an target task T is created, it is marked as detachable. Instead of associating it with a handle, it is now associated with an event E that must be fulfilled before the task can be freed. During task invocation, the GPU kernel is queued as usual, but an event-fulfillment operation is also inserted into the GPU stream to run immediately after the kernel completes. Once the kernel and event-fulfillment are enqueued, the target task is detached, meaning the task stops consuming CPU cycles lays dormant. The task remains idle until the target plugin, i.e., NVIDIA CUDA [4] or AMD ROCm/HIP [1], signals

SC '25, St. Louis, MO

ACM ISBN XX-X-XXXX-XXXX-X/2025/11
https://doi.org/XXXXXXX.XXXXXXX

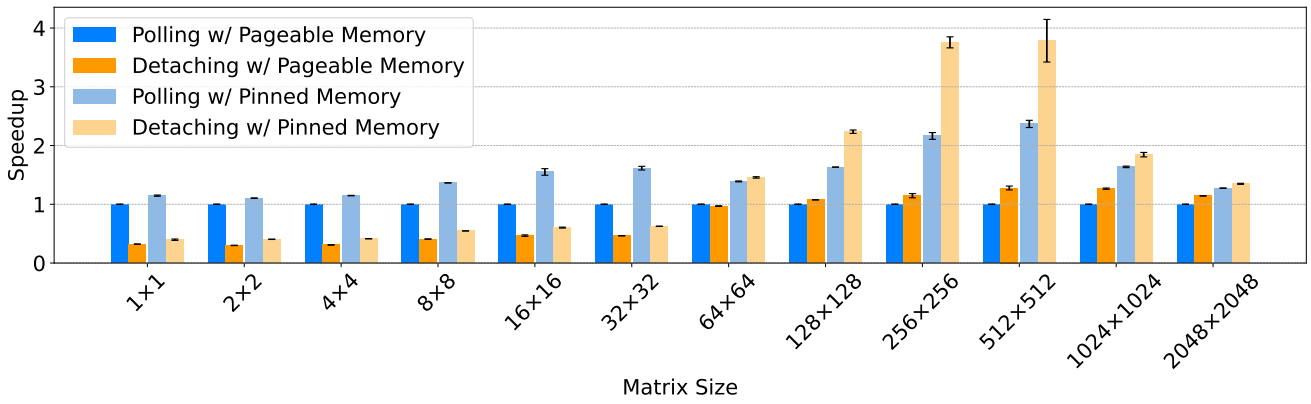

Figure 1. Total runtime speedup of the *detaching* implementation to the *polling* for independent asynchronous matrix multiplications using pageable and pinned memory. Speedup computed relative to *polling* with pageable memory.

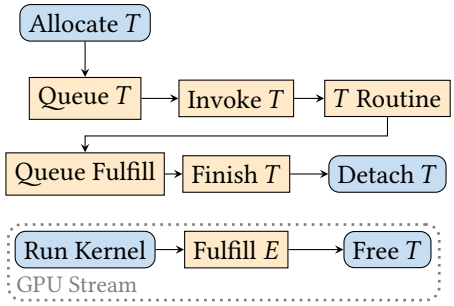

Figure 2. Schematic overview of the *Detaching* workflow.

event completion. A schematic overview of this workflow can be found in Figure 2.

En-queuing the event-fulfillment is handled by the target plugin. On CUDA it can be done via the `cudaLaunchHostFunc` call that can enqueue host functions on a GPU stream. HIP offers an identical call.

3 Evaluation & Results

Evaluation. To evaluate the impact of this approach, we conducted experiments on the CLAIX-2023 [2] system equipped with NVIDIA H100 GPUs. We designed two synthetic benchmarks to expose CPU overhead. The first consists of independent matrix-matrix multiplications, each offloaded asynchronously to the GPU. By varying the matrix size, we achieve different task granularities to estimate the overhead our implementation induces. The second benchmark introduces dependencies between multiplications, forming linear chains and allowing us to examine how detachment interacts with dependency scheduling.

Results. Figure 1 shows the speedup of our *detaching* implementation in comparison to the *polling* implementation for independent tasks. We can see, that detachment reaches a speedup of up to 175% for independent tasks once the matrix size reaches 128×128. When dependencies are present, the benefit appears at even smaller sizes, around 64×64, and correlates positively with increasing dependency strictness. For very large matrices (1024×1024 and above), the advantage diminishes as kernel execution dominates total runtime.

4 Conclusion & Future Work

Our results shows that detachment can be useful for medium to large task granularities. Continuing, we will focus to integrate detachment with the taskgraph extensions in OpenMP, enabling more efficient fine-grained scheduling. We also intend to evaluate detachment in realistic scientific applications such as the ExaHyPE PDE solver [6].

References

[1] AMD. 2025. *HIP Documentation — HIP 6.4.43484 Documentation.* https://rocm.docs.amd.com/projects/HIP/en/latest/

[2] Janin Iglauer, Christian Terboven, and Tim Cramer. 2024. *CLAIX-2023: New Supercomputer at the RWTH • IT Center Blog.* https://blog.rwth-aachen.de/itc/en/2024/02/09/claix-2023/

[3] C. Lattner and V. Adve. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004.* (2004-03). 75–86. doi:10.1109/CGO.2004.1281665

[4] NVIDIA. 2025. *CUDA Toolkit Documentation 13.0.* https://docs.nvidia.com/cuda/

[5] OpenMP Architecture Review Board. 2024. OpenMP Application Program Interface Version 6.0. https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-6-0.pdf

[6] Anne Reinarz, Dominic E. Charrier, Michael Bader, Luke Bovard, Michael Dumbser, Kenneth Duru, Francesco Fambri, Alice-Agnes Gabriel, Jean-Matthieu Gallard, Sven Köppel, Lukas Krenz, Leonhard Rannabauer, Luciano Rezzolla, Philipp Samfass, Maurizio Tavelli, and Tobias Weinzierl. 2020. ExaHyPE: An engine for parallel dynamically adaptive simulations of wave problems. *Computer Physics Communications* 254 (2020), 107251.