Optimizing Task-Driven Offloading in LLVM

Jan Kraus, Joachim Jenke, Christian Terboven
Chair for High Performance Computing, RWTH Aachen University, Germany

Motivation

- **GPU acceleration** can benefit many **scientific applications** that have to use **compute intensive kernels**.
- Common applications: **PDE solvers**, **particle simulations**, and **neural network training and inference**.
- **OpenMP** offers a **portable abstraction** for various **accelerator programming models** (i.e. **CUDA** and **ROCm/HIP**).
- The **OpenMP abstraction** can lead to **additional overhead** that should as **minimal as possible** to make it a viable alternative.

Contribution

- With OpenMP 4.0, the `target` directive was introduced to integrate offloading kernels within OpenMP.
- OpenMP 4.5 extended this directive by introducing **target tasks**: Offloaded kernels are enclosed by an implicit task. Offloaded kernels can seamlessly integrate with the general task flow.
- **Current LLVM implementation**: The target task implementation is fragmented over multiple libraries and communicates asynchronous kernel progress by **polling** a handle. This may lead to unnecessary CPU overhead in between tasks offloaded to accelerators to handle scheduling and dependencies and thus reduce GPU utilization.
- **Proposal**: Having more fine-grained control of kernels within the LLVM OpenMP runtime by **detaching** target tasks can reduce overhead and improve efficiency.

Workflow

Polling: Asynchronous task T is associated with a handle H that is polled for the kernel status. Schematic overview of the implementation in LLVM in Fig. 1.

- T is allocated and queued.
- On first invoke, the task routine is run where handle H is assigned to a value.
- T is not freed until H is deleted again.
- As long as H has a value, T is re-enqueued. During invocation, the routine is skipped and only H is queried.
- When the kernel is finished, H is deleted again, and T is freed.

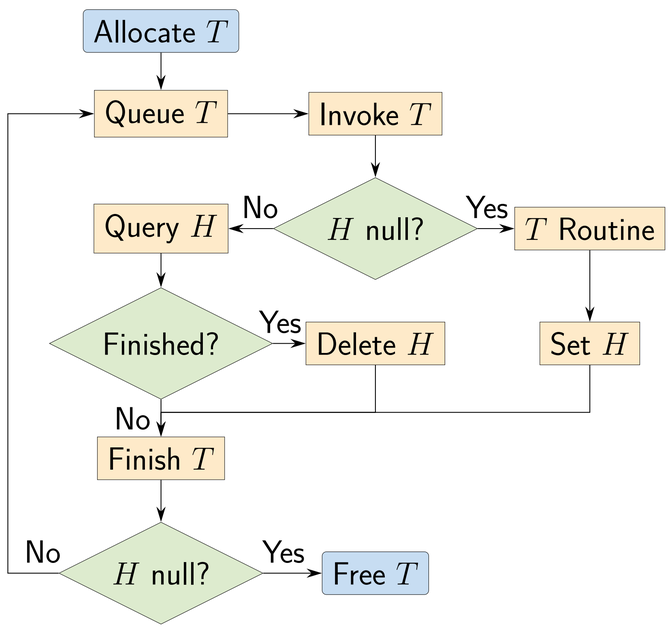

Figure 1: Schematic Representation of the Polling Workflow.

Detaching: Asynchronous target tasks are marked as detachable by default. A detachable task T is associated with an event E that has to be explicitly fulfilled. Schematic representation in Fig. 2.

- During allocation, T is marked as detachable and is assigned an event E.
- In the routine, the event fulfillment for E is queued after the kernel.
- Finish checks if E is already fulfilled, and if not detaches T.
- While detached, T is dormant.
- The plugin runs the kernel and fulfills E afterwards, so T can be freed.

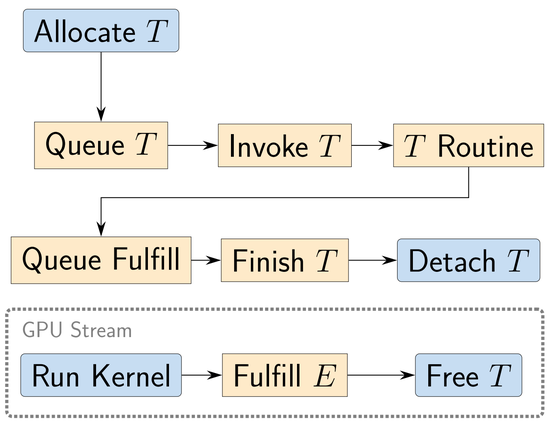

Figure 2: Schematic Representation of the Proposed Detaching Workflow.

Queuing the event fulfillment requires the plugin to return a thread after kernel execution. For CUDA it can be done via `cudaLaunchHostFunc`.

Evaluation

Independent Matrix-Matrix Multiplications

- Fig. 3 shows a naive matrix multiplication with OpenMP target offloading.
- Matrix sizes can be adjusted for different task granularities.
- Pin memory to enable asynchronous data transfers.

```
for (int t = 0; t < tasks; t++) {
  double *C = C_total + s1 * s3 * t;
  #pragma omp target teams distribute parallel for \
    map(from:C[0:s1*s3]) \
    map(to:A[0:s1*s2]) \
    map(to:B[0:s2*s3]) device(0) collapse(2) nowait
  for (int i = 0; i < s1; i++) {
    for (int j = 0; j < s3; j++) {
      C[i*s3+j] = 0;
      for (int k = 0; k < s2; k++) {
        C[i*s3+j] += A[i*s2+k] * B[k*s3+j];
      }
    }
  }
}
#pragma omp taskwait
```

Figure 3: Matrix-Matrix Kernel with OpenMP Target Offloading.

Matrix-Matrix Multiplication with Dependencies

- Identical kernels as Fig. 3.
- Adds linear dependencies in between target tasks as shown in Fig. 4.
- Number of chains C and length of chain L is adjustable to control strictness of dependencies.

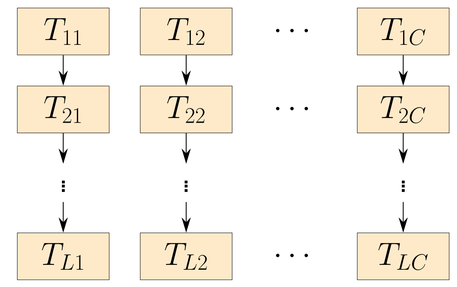

Figure 4: Representation of Linear Dependencies.

Results

Evaluated the total runtime speedup $\frac{T_{\text{Polling}}}{T_{\text{Detaching}}}$ on **CLAIX-2023** [ITC24] with **NVIDIA H100**.

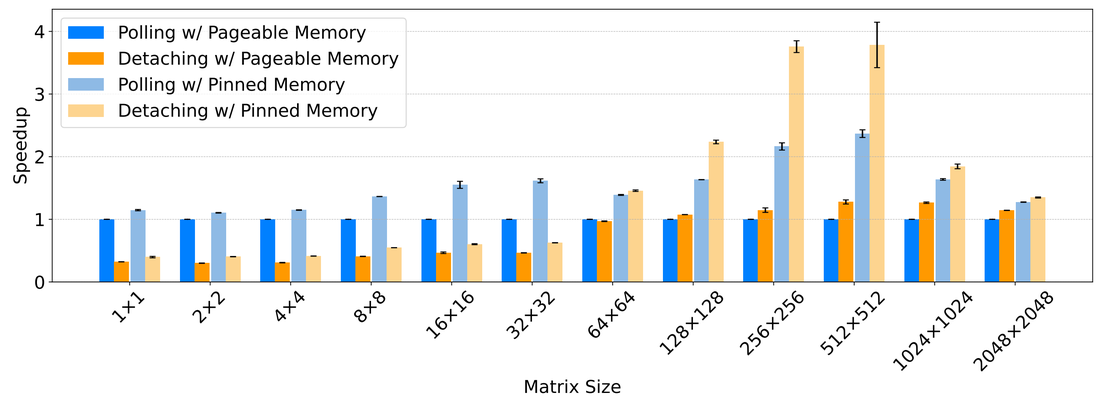

Figure 5: Total Runtime Speedup with 2048 Tasks to Polling w/ Pageable Memory for **Independent Kernels**.

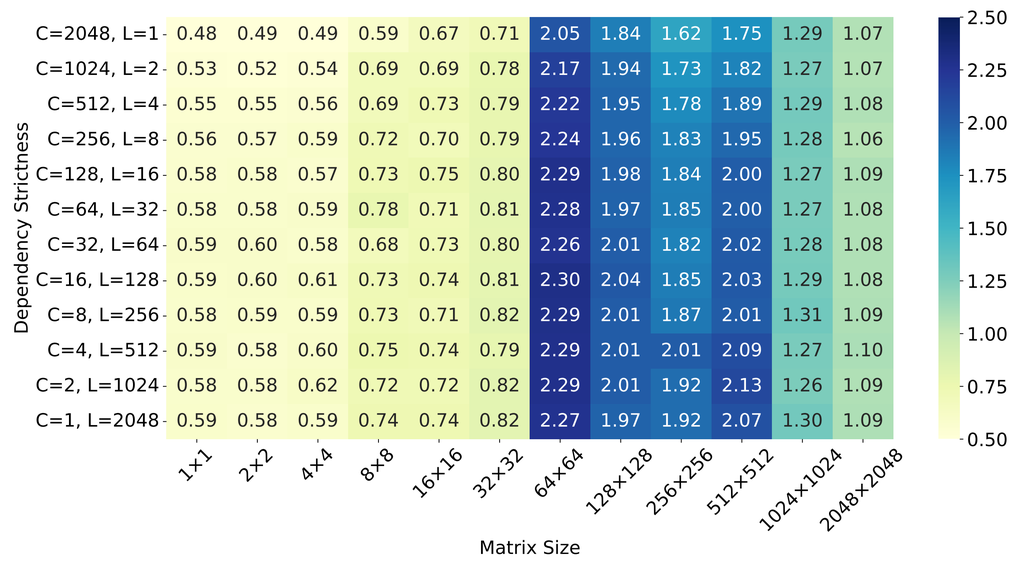

Figure 6: Total Runtime Speedup of Detaching to Polling w/ Pinned Memory for **Kernels with Dependencies**.

- **Detaching** tasks yields a speedup up to **175%** from size **128×128**. (Fig. 5)
- Adding **dependencies** lowers threshold to **64×64**, reaching up to **230%** speedup. (Fig. 6)
- **Stricter dependencies** correlate with **higher speedups**. (Fig. 6)

Further Work

- Evaluate event handling with more realistic work scenarios like Cholesky factorization and applications that use OpenMP offloading with dependencies like ExaHyPE [Rei+20].
- Improve dependency handling even further adding event-based stream management.

References

[ITC24] Janin Iglauer, Christian Terboven, and Tim Cramer. *CLAIX-2023: New Supercomputer at the RWTH IT Center Blog*. Feb. 9, 2024. URL: https://blog.rwth-aachen.de/itc/en/2024/02/09/claix-2023/ (visited on 08/15/2025).

[Rei+20] 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. "ExaHyPE: An engine for parallel dynamically adaptive simulations of wave problems". In: *Computer Physics Communications* 254 (2020), p. 107251.

Additional Data

Source Code

https://github.com/RWTH-HPC/llvm-project/tree/SC25-Poster

Artifacts

https://github.com/RWTH-HPC/LLVM-Detachment-SC25-Poster-Artifact

{kraus,jenke,terboven}@itc.rwth-aachen.de
https://www.i12.rwth-aachen.de/