Directive-Based Pipelining Extension for OpenMP

Xuewen Cui, Thomas R. W. Scogland , Bronis R. de Supinski and Wu-chun Feng

SyNeRGy Lab @ Virginia Tech

Abstract

Heterogeneity continues to increase in all kinds of computing applications, with the rise of accelerators such as GPUs, FPGAs, APUs, and other co-processors. Programming models, such as CUDA, OpenACC and OpenCL are designed to offload compute-intensive workloads to co-processors efficiently.

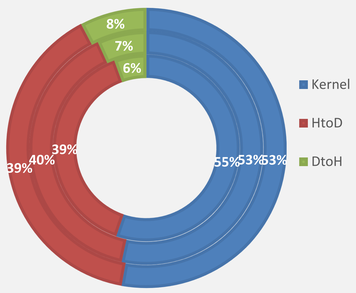
The problems of naive offload model
- Synchronously copying and executing, in sequence is inefficient.
- Manually pipelining these activities reduces programmability.

We propose a directive-based pipelining extension for OpenMP.
- Our extension offers a simple interface to overlap data transfer and kernel computation with an auto-tuning scheduler.
- We achieve performance improvements between 40% and 60% for a Lattice QCD application.

Motivation and Goal

Offload computation consists of 3 phases:

Drawbacks:
- Compute resources are wasted during data transfers.
- Requires all the data to be copied; fails without enough memory.

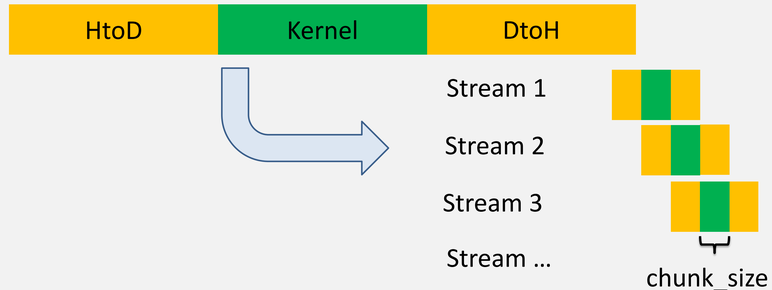


Time distribution of kernel and data transfer for Lattice QCD code.
(3 problem sizes, 16,24,32 inside out)

Goal: A directive-based pipelining extension that improves:
- Programmability
  - ✓ Directive-based pipelining extension for user to define task splitter and scheduler.
- Performance
  - ✓ Overlap kernel computation and data transfer
- Memory management
  - ✓ Detect the remaining on-chip memory before launching new sub-tasks to avoid out of memory errors

Task Splitter and Auto-Tuning Scheduler

- Splitting the task into small chunks and then launching a GPU stream to execute each chunk.
- The chunk size and number of streams can be set statically or adapted at runtime. The scheduler also adjusts the chunk size to fit in limited memory.



Proposed Extension Syntax

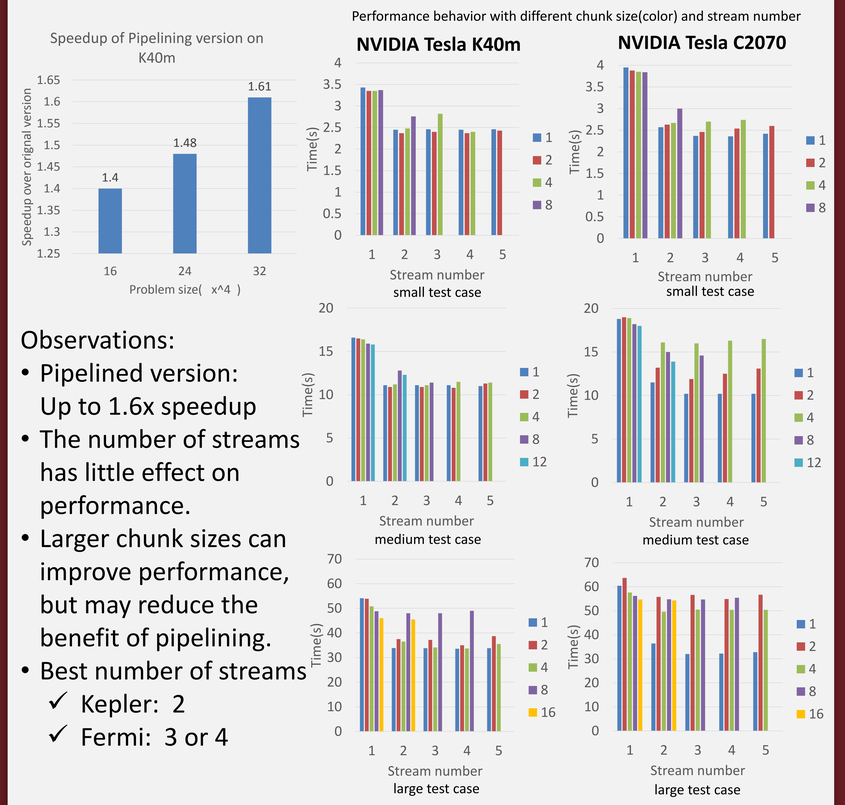
```
#pragma target teams distribute parallel for\
  pipeline(<scheduler>(<chunk_size>,<num_stream>))\
  pipeline_copy(in/out)(<var>(size)[<cond>:<num>])\
pipeline_shadow(in/out)(<var>(<offsets…>:<cond_roll>))\
  pipeline_mem_limit(<mem_size>)
```

pipeline() inputs	
<scheduler>	Scheduler to use for this region(static, adaptive)
<chunk_size>	Sub-task chunk size
<num_stream>	Stream number to launch on GPU
pipeline_copy() and pipeline_shadow() inputs	
<in/out>	Input data or output data
<var>	Variable(array) to copy
<size>	Size of each “item” in the array/matrix
<cond>	Whether this dimension is the outer split loop
<num>	Number of items in this dimension
<offsets…>	Shadow element offsets
<cond_roll>	If the shadow should wrap around
<mem_size>	Maximum memory usage

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-POST-675851

Performance Results and Conclusion

Case Study: Lattice QCD code on NVIDIA GPUs



Observations:
- Pipelined version: Up to 1.6x speedup
- The number of streams has little effect on performance.
- Larger chunk sizes can improve performance, but may reduce the benefit of pipelining.
- Best number of streams
  - ✓ Kepler: 2
  - ✓ Fermi: 3 or 4

Future Work

- Test and study the behavior of more scientific applications
- Support for complex memory layouts
- Translator support for directive-based extension
- Auto-tuning model

References

1. Scogland, T., Feng, W., Rountree, B., de Supinski, B., CoreTSAR: Core Task-Size Adapting Runtime. In *Int’l Supercomputing Conf.*, 2014.
2. Scogland, T, Rountree, B., Feng, W., de Supinski, B., Heterogeneous Task Scheduling for Accelerated OpenMP. In *26th IEEE Int’l Parallel & Distributed Processing Symp.* 2012.