TECHNISCHE
UNIVERSITÄT
DRESDEN

Center for Information Services and High Performance Computing (ZIH)

Scalable Critical Path Analysis for Hybrid MPI-CUDA Applications

The Fourth International Workshop on Accelerators and Hybrid Exascale Systems, May 19th 2014

Felix Schmitt, Robert Dietrich, Guido Juckeland

ZIH
Center for Information Services &
High Performance Computing

Outline

1. Motivation
2. CUDA Dependency Patterns
3. MPI-CUDA Critical Path Analysis
4. Use Cases
5. Outlook and Conclusion

- **Motivation**
- CUDA Dependency Patterns
- MPI-CUDA Critical Path Analysis
- Use Cases
- Outlook and Conclusion

Motivation

- CUDA established for using general-purpose graphics-processing units in HPC [1]
- Increasing complexity of hybrid HPC programs requires sophisticated performance-analysis tools
- Problem: no current tool for automated analysis of execution dependencies in MPI-CUDA programs
  - Scalasca: scalable MPI critical-path analysis
  - HPCToolkit: MPI-CUDA profiling, no intra-device dependencies
  - NVIDIA Visual Profiler: CUDA optimization guidance, no MPI

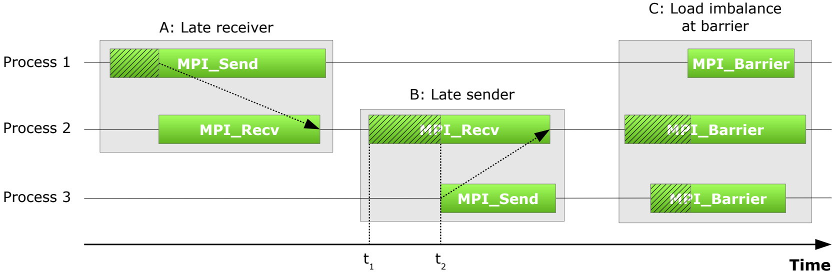
Goals

Guide the developer to optimization targets in hybrid MPI-CUDA programs

- Scalable critical-path analysis based on trace files
- Analyze host/device and device/device dependencies and inefficiencies
- Visualize analysis results in Vampir
- Order activities by their potential optimization influence

Preliminaries: Wait-State Analysis

- *Event Stream*: stream of ordered events, e.g. MPI process, CUDA stream
- *Wait State*: time period at which an event stream is blocked [2], result of inter-stream dependencies and load imbalances
- *Blame* (HPCToolkit) or *cost of idleness* (Scalasca): attributed to the cause of a wait state

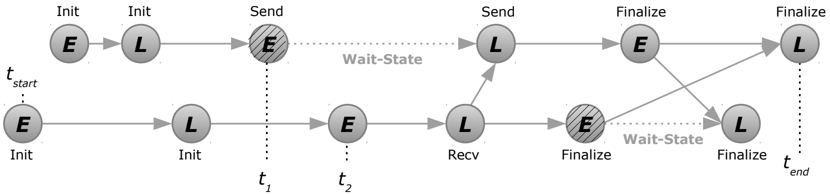


Examples for MPI Wait-States

Preliminaries: Critical Path

Event Dependency Graph (EDG): directed acyclic graph

- Nodes are the events of parallel event streams
- Edges model the *happens-before* relationship and are weighted with the duration between events [3]

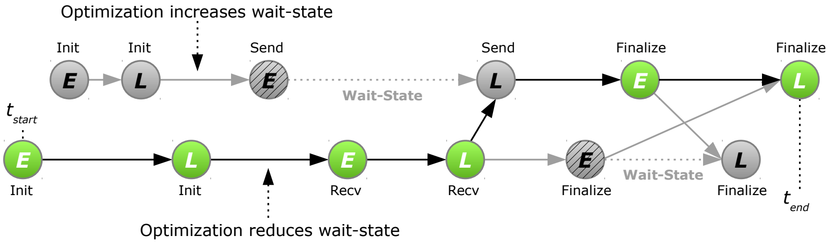


EDG for simple MPI example
(`MPI_Init`, `MPI_Send/Recv`, `MPI_Finalize`)

Preliminaries: Critical Path (2)

Critical Path: [4]

- Longest path in an EDG without wait states
- Optimizing activities on this path *can* reduce execution time
- Optimizing other activities *can not* (directly)



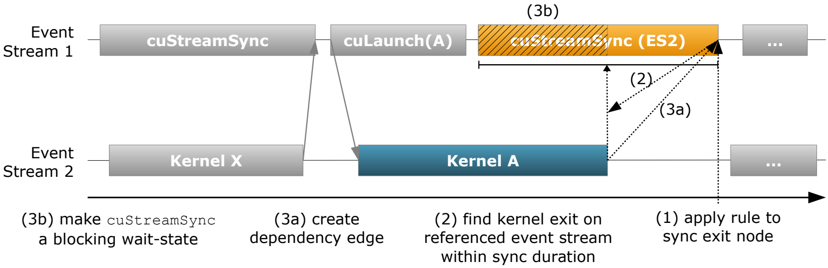
CUDA Wait-State Analysis

- Create a dependency/wait-state model for CUDA
- Two activity kinds: host (API) and device (kernels, memcpys)

New categorization of CUDA Inefficiency Patterns:

- **Blocking Synchronization**
- **Non-Blocking Synchronization**
- **Late Synchronization**
- **Inter-Stream Dependencies**

Rule-Based Pattern Detection



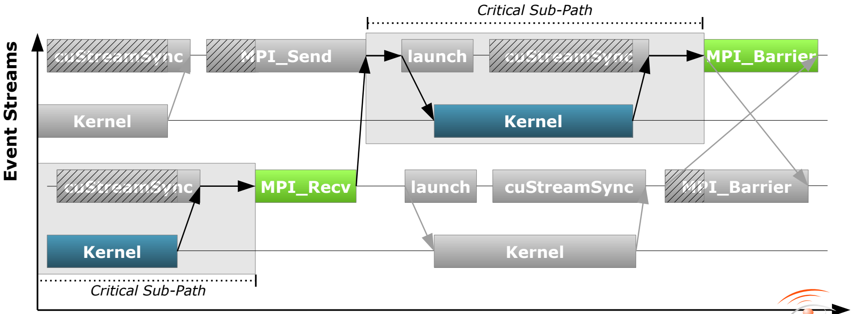
BlameKernelRule

Identifies blocking synchronization that is delayed by device activities.

- Motivation
- CUDA Dependency Patterns
- **MPI-CUDA Critical Path Analysis**
- Use Cases
- Outlook and Conclusion

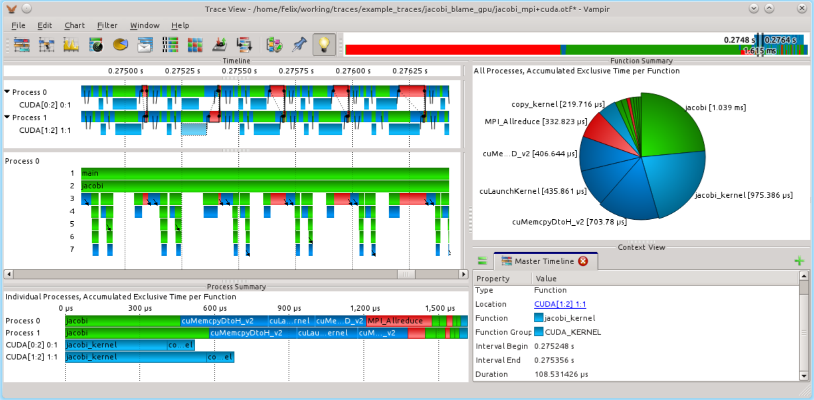
Critical Sub-Paths

- Combine MPI and CUDA critical path analysis
- MPI critical path detected using Scalasca's *parallel reverse replay* [5]
- Global CUDA critical path is dominated by MPI critical path
- → Determine critical sub-paths to *efficiently* and *concurrently* compute CUDA critical paths using OpenMP



Visualization in Vampir

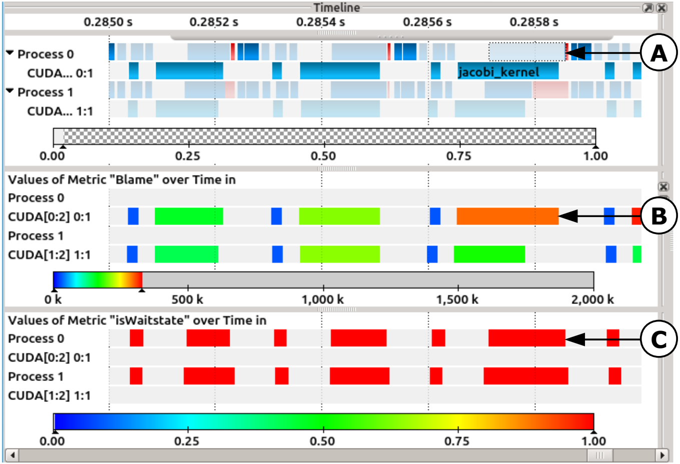
Vampir and *VampirServer* enable scalable visualization of hybrid applications, including timelines, profiles, message and data transfers and performance counters.



Trace View - /home/Felix/working/traces/example_traces/jacobi_blame_gpu/jacobi_mpi+cuda.otf* - Vampir

Property	Value
Type	Function
Location	CUDA[1:2] 1:1
Function	jacobi_kernel
Function Group	CUDA_KERNEL
Interval Begin	0.275248 s
Interval End	0.275356 s
Duration	108.531426 µs

Visualization in Vampir (2)



(A) Counter Overlay: blocking memory copy (implicit synchronization)
(B) Counter Timeline: the synchronized kernel is attributed blame
(C) Counter Timeline: blocking synchronization is marked as waiting time

Activity Optimization Order

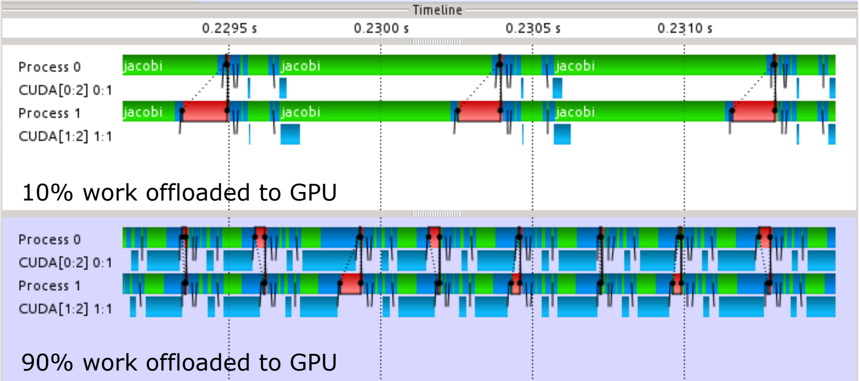
- Goal: Rank activity types by their potential influence
- Create an optimization order for activity types, add
  - normalized fraction of total critical-path duration (direct runtime impact)
  - normalized fraction of total blame (load-balancing impact)

→ Highest-rated activities are best optimization candidates

- Motivation
- CUDA Dependency Patterns
- MPI-CUDA Critical Path Analysis
- **Use Cases**
- Outlook and Conclusion

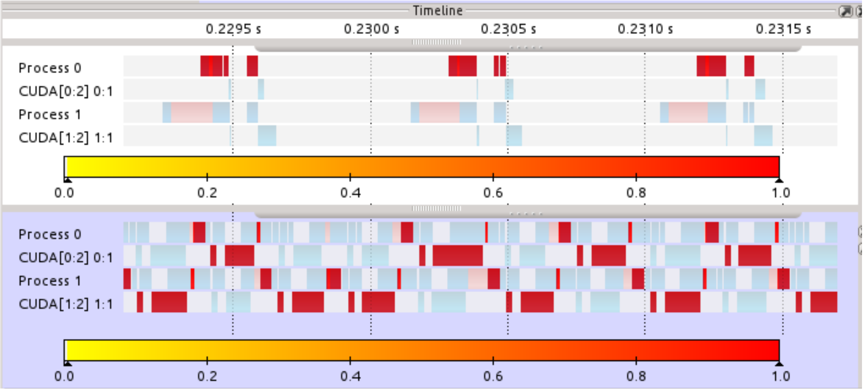
Correctness: Jacobi Method

MPI+CUDA application (two processes, one CUDA stream each).
Executes two kernels in each iteration.



Section of a trace in Vampir with two kernels:
jacobi_kernel and *copy_kernel*.

Correctness: Jacobi Method (2)



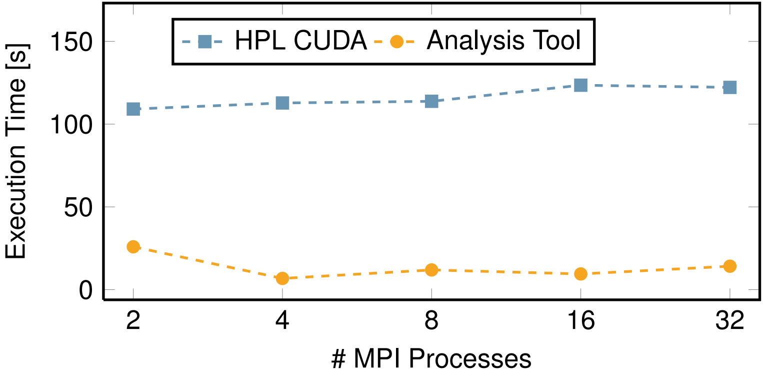
Analysis result in Vampir's performance radar (timeline overlay): CUDA kernels become critical activities (red) for high GPU offloading ratio due to blocking synchronization.

Correctness: Jacobi Method (3)

Activity (all instances)	Critical Path [%]	Blame [%]	Rating
jacobi_kernel	**40.69**	**35.34**	**0.7603**
cuMemcpyDtoH_v2	30.10	5.6	0.3570
MPI_Barrier	~0	35.62	0.3562
copy_kernel	5.04	9.59	0.1463
MPI_Allreduce	~0	12.78	0.1278
cuMemcpyHtoD_v2	10.15	0.0	0.1015

Activity optimization order for 90% work offloaded to the GPU.

Scalability: HPL CUDA



Scalability of HPL CUDA version and analysis [1].
Combining MPI parallel replay and CUDA dependency analysis still scales with the MPI operations of the input trace.

---

[1] 1 MPI process/node, NVIDIA K20X GPUs

- Motivation
- CUDA Dependency Patterns
- MPI-CUDA Critical Path Analysis
- Use Cases
- **Outlook and Conclusion**

Conclusion

Contributions:

- Comprehensive dependency model for CUDA activities
- Scalable tool for critical-path analysis of MPI-CUDA traces
- Identifies waiting time and the causing activities
- Visualization of all metrics in Vampir
- Generates a list of optimization targets, ordered by potential influence

Future Work

- Extend support to applications including OpenMP, CUDA and MPI (prototype available)
- Evaluate usage of hardware performance counters during optimization guidance
  → Which activities are easier to optimize?
- General CPU functions missing in this implementation (added in prototype)

Thank you for your attention!
Questions?

References

[1] Wu-chun Feng and Kirk W. Cameron.
The Green500 List - November 2013.
`http://www.green500.org/lists/green201311`, November 2013.

[2] Wagner Meira, Thomas J. LeBlanc, and Virgilio A. F. Almeida.
Using cause-effect analysis to understand the performance of distributed programs.
In *Proceedings of the SIGMETRICS symposium on Parallel and distributed tools*, SPDT '98, pages 101–111, New York, NY, USA, 1998. ACM.

[3] Leslie Lamport.
Time, clocks, and the ordering of events in a distributed system.
Commun. ACM, 21(7):558–565, July 1978.

[4] C.-Q. Yang and B.P. Miller.
Critical path analysis for the execution of parallel and distributed programs.
In *Distributed Computing Systems, 1988., 8th International Conference on*, pages 366–373, 1988.

[5] David Bohme, Felix Wolf, and Markus Geimer.
Characterizing Load and Communication Imbalance in Large-Scale Parallel Applications.
In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 2538–2541, 2012.