Enabling Graph-Based Profiling Analysis using Hatchet

Student: Ian Lumsden[1]

Mentors: Stephanie Brink[2], Michael R. Wyatt II[1], Todd Gamblin[2] , Michela Taufer[1]

Introduction

- Profiling is a way to measure the performance of code and how code runs on systems in high-performance computing (HPC)
- Numerous tools for HPC profiling (e.g. TAU, Caliper, HPCToolkit) have **custom** data format and analysis tools
  - Users are locked into types of analysis dictated by the provided tools
- Hatchet [1] is a new, **general** data analysis tool that can read HPC profiling data from **different profilers**
  - Store the raw performance data into a *pandas* DataFrame
  - Represent the relational *caller-callee* data with a directed acyclic graph

- Hatchet restricts users to table-based analysis of the raw performance data
- Hatchet does NOT support analysis of the relational data collected by HPC profilers

Research Goals

Augment Hatchet to enable analysis using the relational data collected by HPC profilers:

- Design a new graph-based filtering query language to enable the use of relational data collected by profilers in analysis
- Integrate the graph query language into Hatchet analysis
- Use the augmented Hatchet to analyze the performance of different MPI calls in HPC benchmark applications

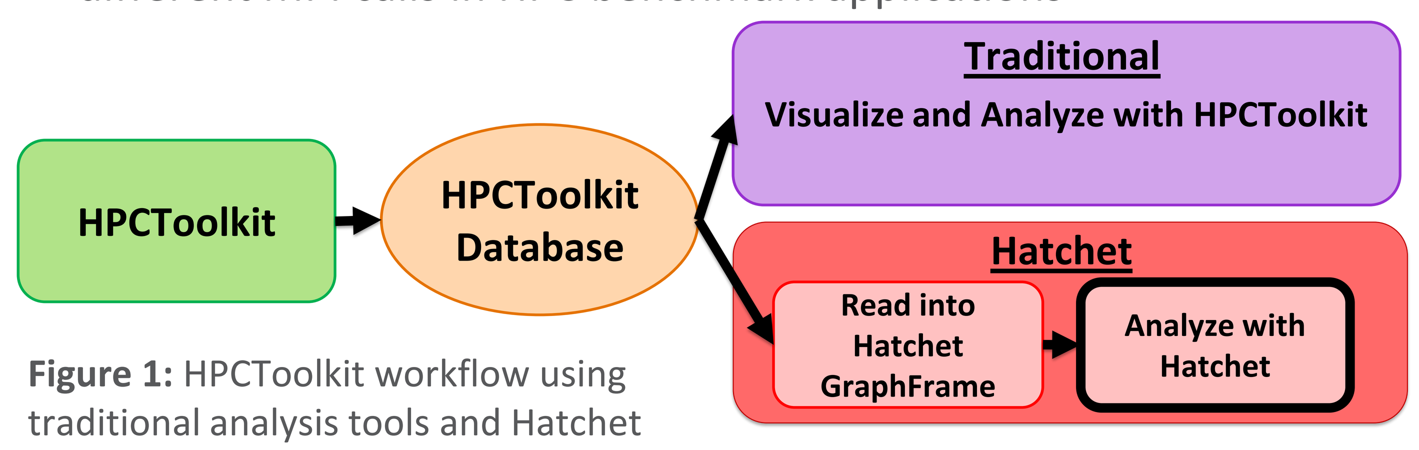


Figure 1: HPCToolkit workflow using traditional analysis tools and Hatchet

Methods

Our graph-based filtering query language consists of:

- **User Input**: A Query Path represented as a list of abstract graph nodes
- **Algorithm**:
  - Read and parse the user's query path
  - Match real nodes in the graph being filtered to the abstract nodes in the query path
  - Collect all graph paths that match the full query path
  - Create a new graph containing only the nodes in the matched paths

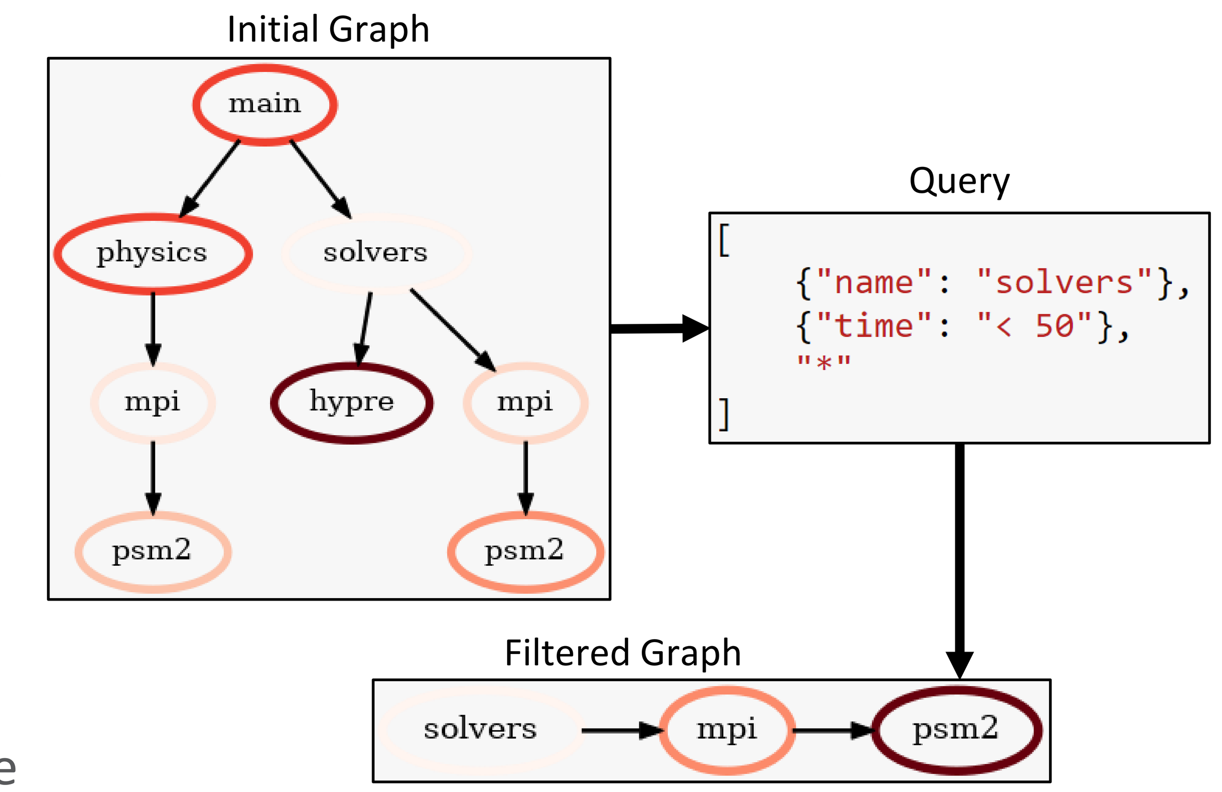


Figure 2: Example of filtering a graph using the new query language

Case Study: Application to MPI Benchmarks

1. Collect Data

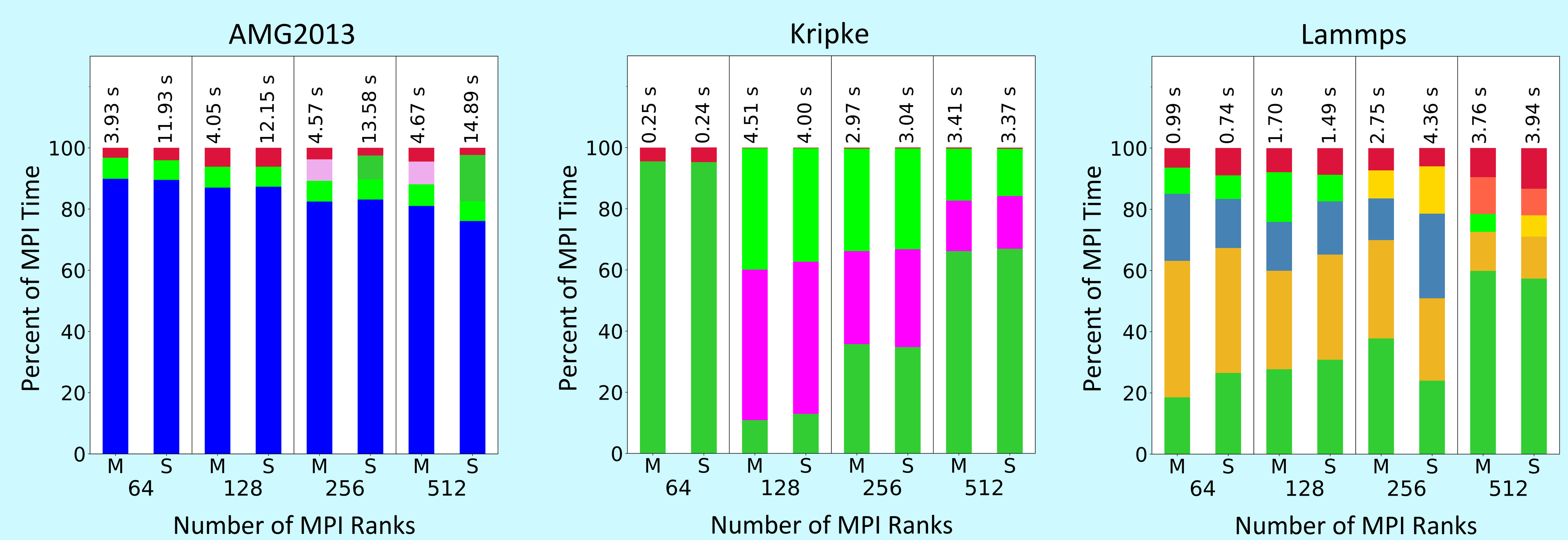
- Run each of the following benchmarks with MVAPICH2 (**M**) and Spectrum-MPI (**S**) using 64, 128, 256, and 512 MPI ranks, and profile the runs with HPCToolkit
  - AMG2013
  - Kripke
  - Lammps
- Load the generated profiles into Hatchet
- Perform the benchmarking on LLNL's Lassen supercomputer

2. Extract MPI Layer

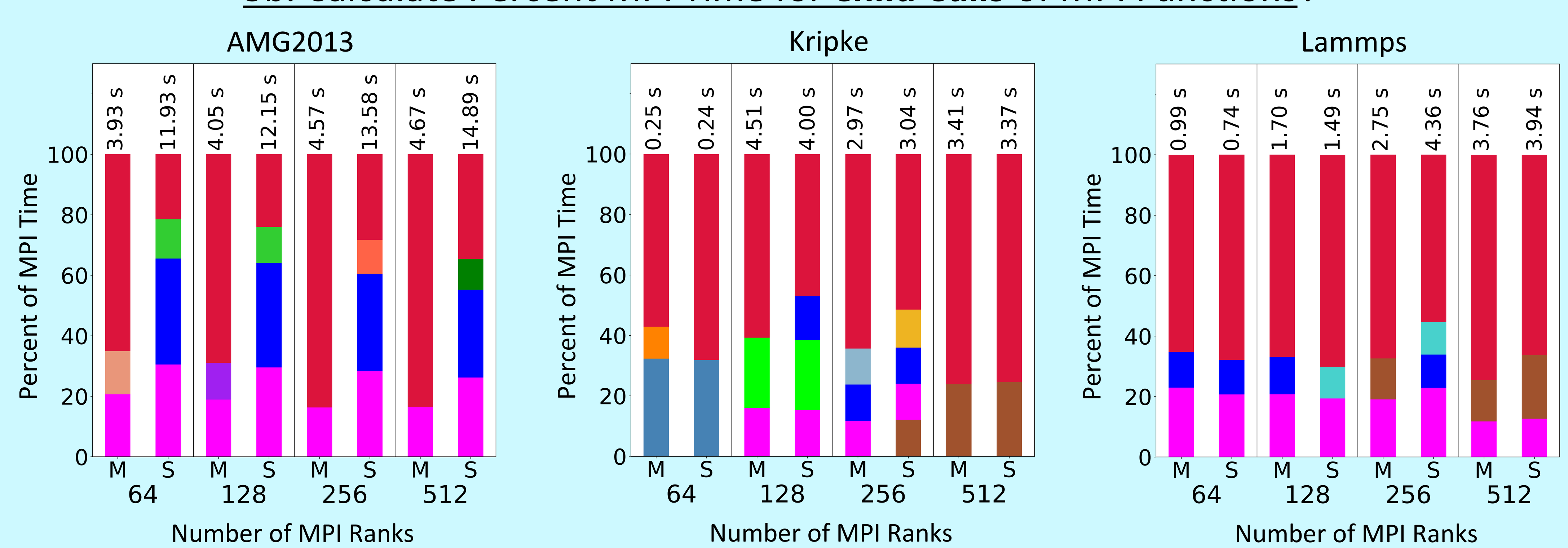
- Filter the *call graphs* to get subgraphs rooted at standard MPI function calls
- Query Path used to extract MPI Layer:

[{"name": "P?MPI_.*"}, "*"]

3a. Calculate Percent MPI Time for each *MPI Function*†

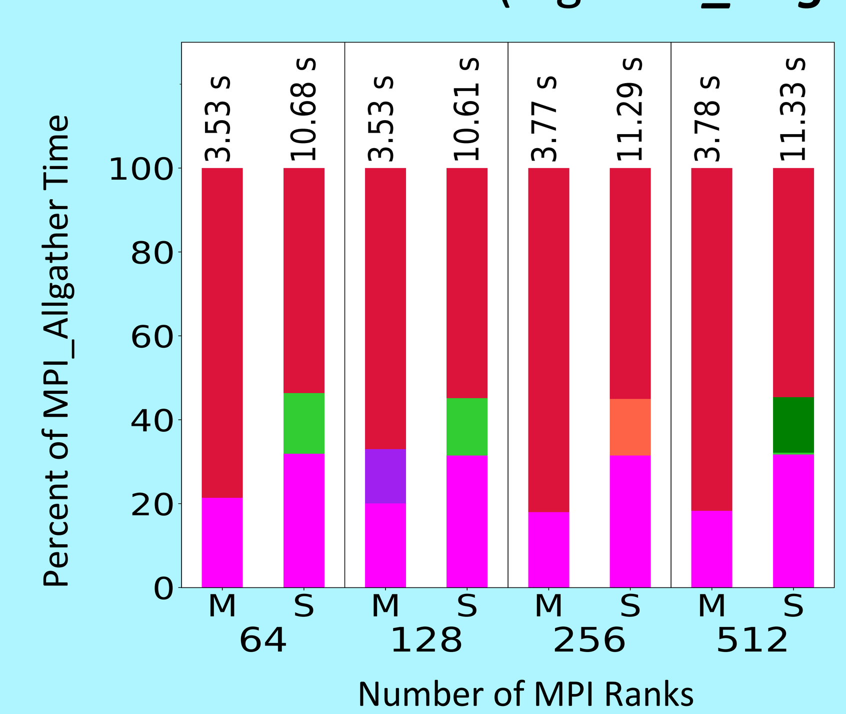


3b. Calculate Percent MPI Time for *Child Calls* of MPI Functions‡



4. Identify Slow-Down Causes‡

Zoom into specific benchmarks (e.g. ***AMG2013***) and examine the ***children*** of specific MPI Functions (e.g. ***MPI_Allgather***)



MPI Function Calls

MPI_Finalize	MPI_Send
MPI_Allreduce	MPI_Wait
MPI_Allgather	MPI_Waitany
MPI_Waitall	MPI_Alltoallv
MPI_Testany	Remaining MPI Time

Child Function Calls

<unknown file> [libopen-pal.so.3.1.0]:0	<unknown file> [libmlx5.so.1.0.0]:1133	<unknown file> [libmlx5.so.1.0.0]:0
<unknown file> [libpami.so.3.1.0]:0	pthread_spin_lock.c:26	syscall-template.S:81
syscall-template.S:82	pml_pami_send.c:0	pml_pami_init.c:0
cancellation.c:81	memset.S:1133	Geometry.h:0
stl_vector.h:0	malloc.c:0	Remaining MPI Time

Lessons Learned

Using the query language and Hatchet, we were able to:

- Extract all call paths specific to a given library
- Determine the performance contributions of function calls used by these libraries
- Correlate children function calls to specific important library API calls in an application
- Use this correlation to determine children function calls that contribute the most to the performance of the targeted library API call
- Compare the correlation of children and API calls across libraries to determine possible causes for performance differences

In our tests with MVAPICH2 and Spectrum-MPI, we learn that:

- The **pthread_spin_lock** function is consistently one of the most impactful in the performance on MPI functions for both libraries
- In AMG2013, the worse performance of MPI_Allgather in Spectrum-MPI can possibly be attributed to **pthread_spin_lock**

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Sponsors:

[1] Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN, USA
[2] Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA, USA

References

[1] A. Bhatele et al. 2019. Hatchet: Pruning the Overgrowth in Parallel Profiles. In *Proc. of SC19*

† "Remaining MPI Time" consists of all MPI functions that contribute less than 5% of the overall MPI runtime.
‡ "Remaining MPI Time" consists of all child functions that contribute less than 10% of the overall MPI runtime.

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

Release Number: LLNL-POST-812963

GCLab Website: