On-line Application-specific Tuning with the Periscope Tuning Framework and the MPI Tools Interface

Isaías A. Comprés

Technical University of Munich, Germany

Petascale Tools Workshop

Madison, August 4th 2014

Overview

Introduction

Periscope Tuning Framework

MPI Tools Interface

MPICH Patches

MPIT Plugin

Performance Results

Conclusion

Introduction

MPI Runtime parameters:

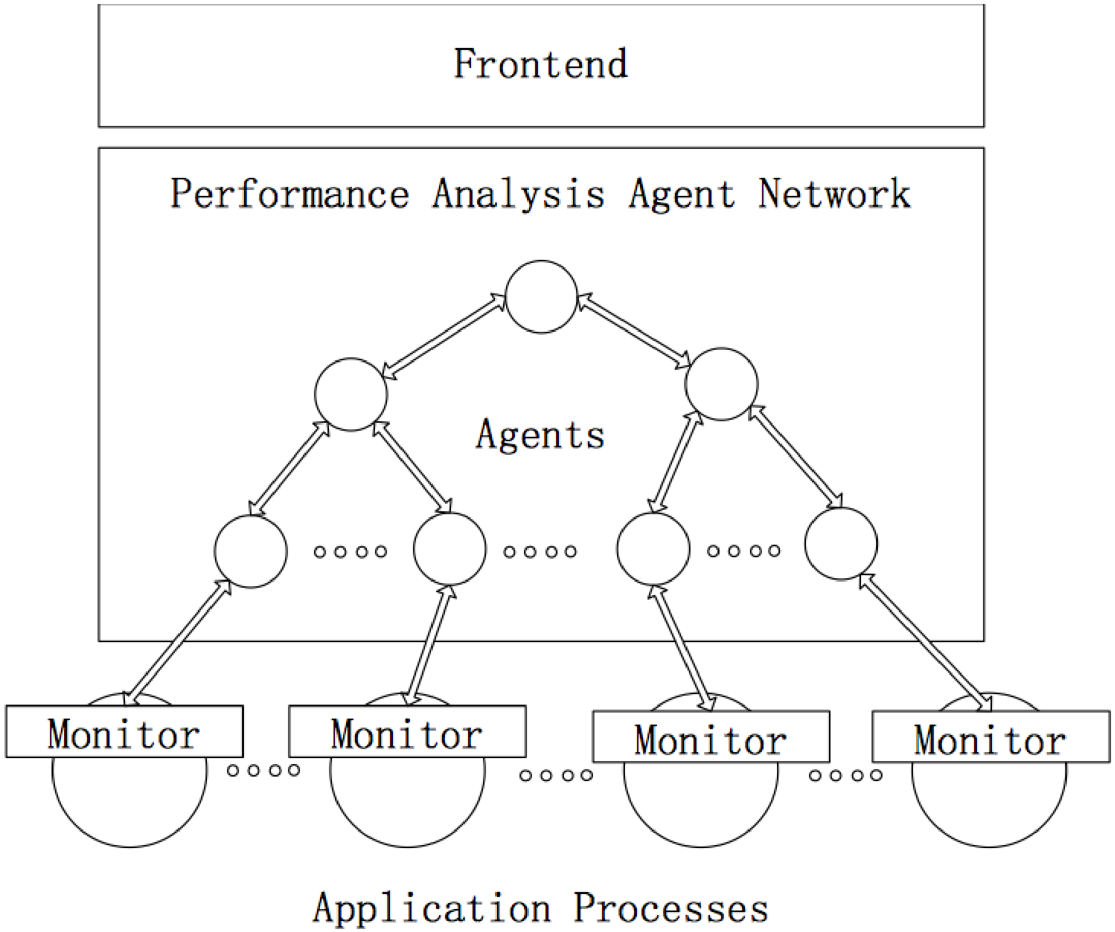
- Certain parts of MPI implementations are already configurable
  - Protocol thresholds, collective algorithms, buffer and queue sizes, ect.
- These parameters can be adjusted for better performance

Advantages:

- The modification of application sources is not necessary
- MPI runtime parameters can be changed without the need to recompile or relink the applications
  - Additionally, these changes can be performed quickly when online access is available (through MPI-T or proprietary interfaces)

Overview

Periscope Tuning Framework



Periscope components:

- Frontend
- Communication Agents
- Analysis Agents
- Monitoring Request Interface (MRI)
  - Replaced with Score-P

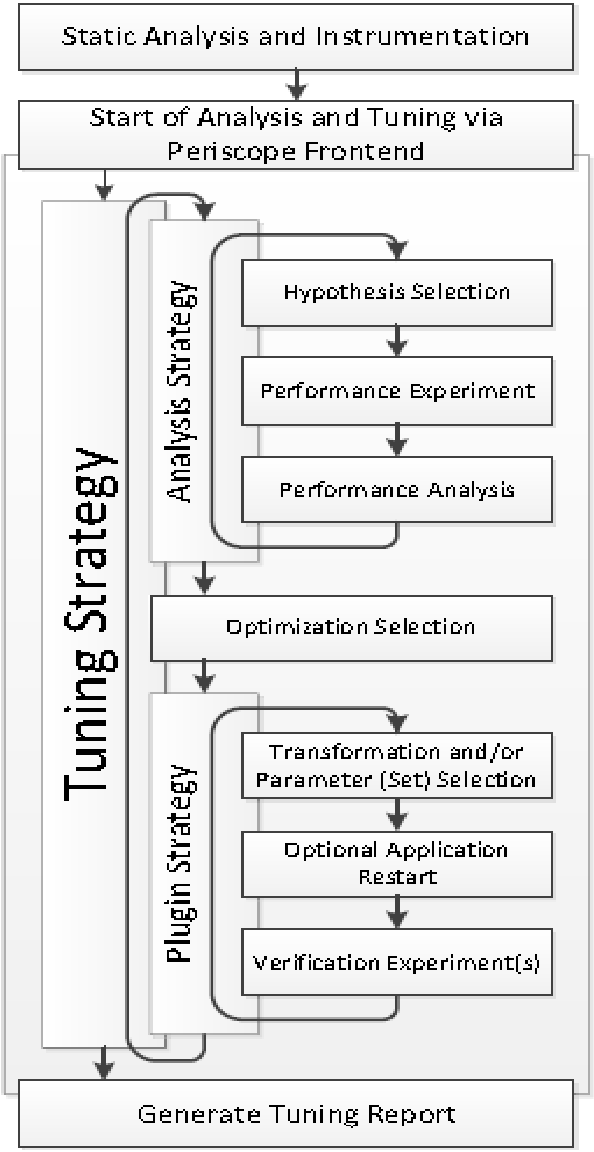
Periscope Tuning Framework

Analysis strategy:

- Runs the application and automatically detect areas of interest
- The type of application and analyses to be performed are dictated by the user and plugins

Tuning strategy:

- Programming model, hardware platform or performance aspect specific
- Can be model based, search based or a combination of both
- Generic flow with specific paths selected by loadable plugins

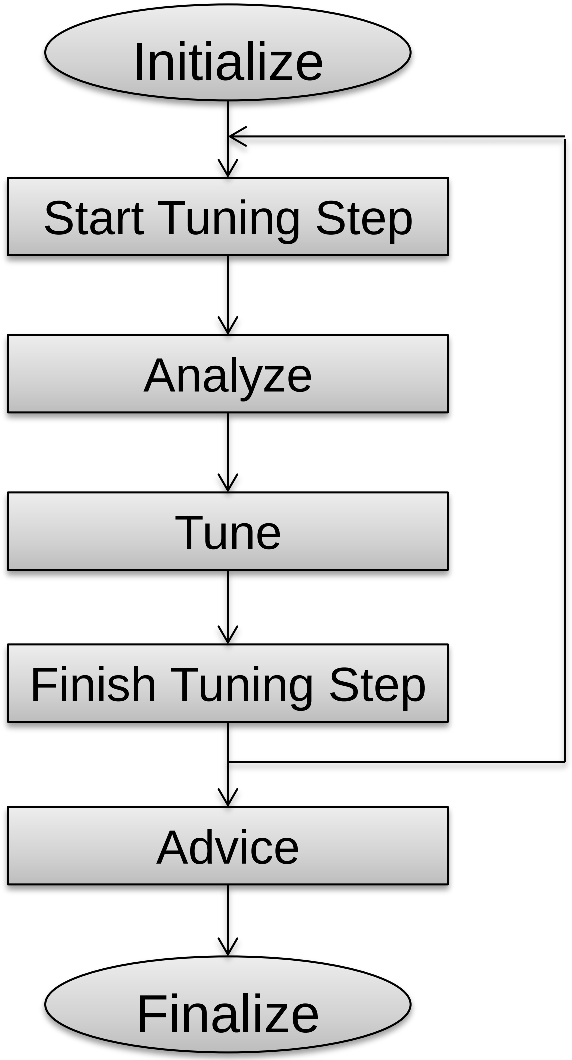


Plugin Flow

- Plugins are shared libraries that implement the plugin interface
  - They are implemented in C++
  - Need to be compiled with compatible tool chains

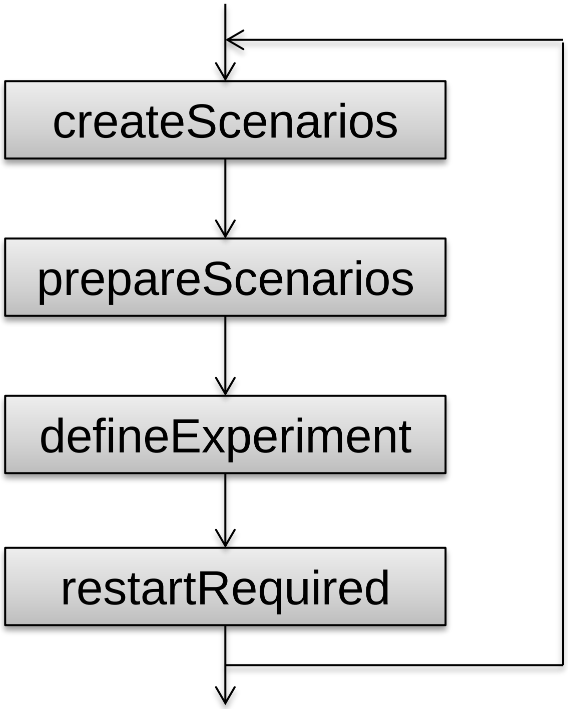
- Plugins indicate which path to follow in the analysis and tuning process
  - Most operations are optional

- Important when developing a plugin:
  - Relevant analyses
  - Modeling possibilities
  - Parameters that impact performance
  - Required application restarts



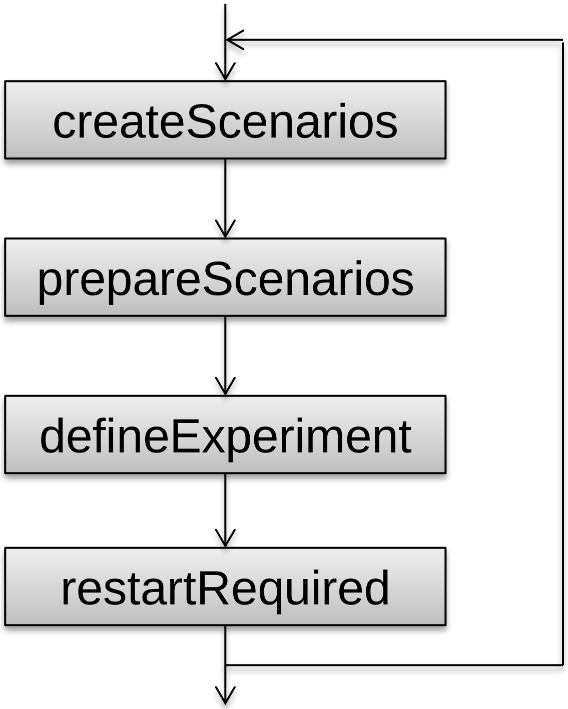
Plugin Flow

- Scenarios
  - Data objects that contains relevant data for an experiment:
    - Concrete values for relevant parameters
    - Where to set these parameters
    - Which performance properties to detect
    - Where to measure the effects
  - Scenario creation and manipulation is one the main tasks of a plugin
- Scenario lifetime
  - Scenarios move between pools as they are processed by plugins and the Frontend
    - Created (CSP), prepared (PSP), experiment (ESP), and finished (FSP) scenario pools



Plugin Flow

- createScenarios
  - Scenarios are created purely based on the number of parameters and their ranges
  - This operation can be offloaded to a search algorithm
- prepareScenarios
  - Preparation for the scenarios, if necessary
  - Examples include: compilation, environment settings, runtime options, ect.
- defineExperiment
  - The scenarios are mapped to the execution environment
  - This includes processes or threads, code regions, ect.
- restartRequired
  - Request a restart if required by the experiment



Periscope Tuning Framework (PTF) Summary

- Runtime environment
  - Hierarchy of communication agents
    - Leaf and aggregation agents
  - Monitoring library linked to the application
- Generic framework
  - Allows for different types of automatic tuners
  - Mixed analysis, modeling and empirical approaches possible
- Plugin interface
  - Predefined general flow; specific path selected by plugins
  - Plugins are loadable components
    - Can be implemented by third parties and be closed source
  - Plugins dictate which operations to perform, depending on the aspect to tune
    - *Parameters can be set online or require a restart*

Overview

MPI Tools Interface

MPI-3 interface that allows for:

- Performance variables (PVAR)
  - State of the MPI library
    - Message queues
    - Configuration parameter values
  - Performance counters
    - Aggregated traffic
    - Current set parameter values
- Control variables (CVAR)
  - Configurable at runtime
    - Mostly restricted to before MPI_Init
  - Isolation through sessions, where applicable
- Implementers are free to decide what to expose with MPIT

Current Implementations

Open MPI:

- Hundreds of parameters available
  - Point to point and collectives thresholds
  - Many other Open MPI specific parameters
- CVARs writable before MPI_Init only

MPICH:

- Several changes were done internally since 1.5 (documented in MPICH's website)
  - Improved efficiency
  - Conforms better to the standard
- Parameters relevant to point to point and collectives

Overview

MPICH Patches

- Motivation: quick online updates to parameters
  - Slight degradation of performance acceptable
  - Some correctness constraints handled by our tool
- Based on MPICH 1.5.x (unfortunately)
  - Most features were marked as experimental
  - YAML files processed at configuration
  - *Our patches no longer compatible with the trunk*
- Created extra CVARs
  - Explicit selection of internal collective algorithms
    - MPICH deals with thresholds instead
  - Can be changed at runtime
    - Restrictions handled by the plugin, where applicable

MPICH Patches

Point to point:

- These are in general safe to change per point to point operation
  - Sender specifies how to handle the payload in the message itself (active messages)
- These changes required extra care with the nemesis implementation
  - Some operations can be replaced by function pointers
    - Luckily, these replacements are done once, at initialization
  - Sets of CVARS for intra- and inter-node
    - Typically shared memory and network variants
- Eager and LMT thresholds, depending on the platform

MPICH Patches

Collectives:

- 8 blocking collectives, 8 non-locking collectives
- Algorithm selected explicitly, and not through thresholds
- Correctness ensured by the PTF plugin
  - Some algorithms have certain requirements for correctness
    - For example: powers of 2 process counts in the communicator
  - Internal algorithm must match in all participating processes

PVARS:

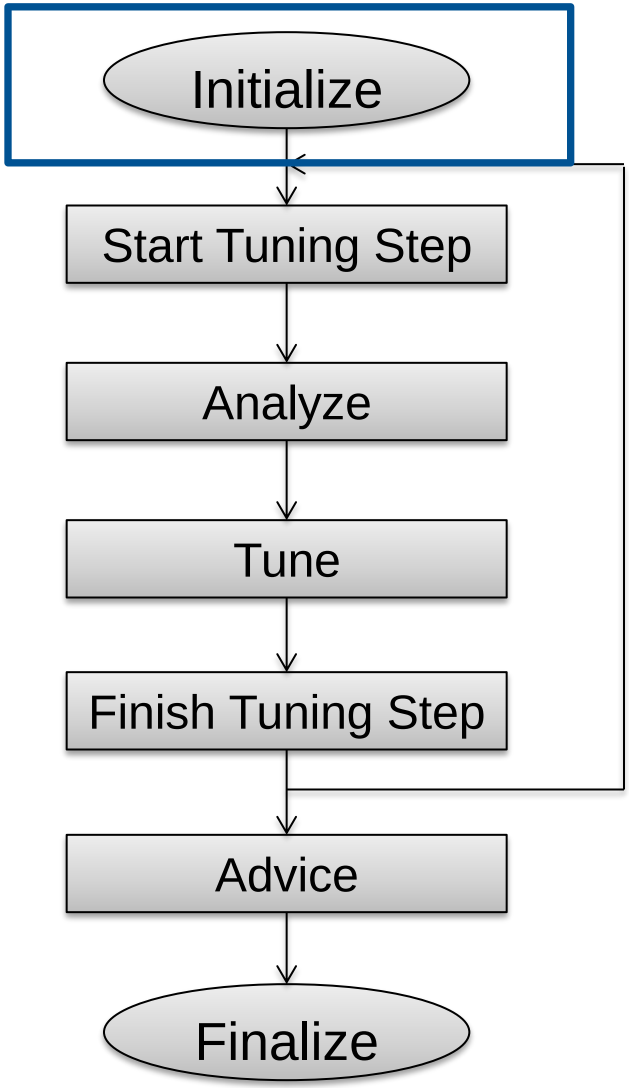
- Additional low level network traffic counters exposed
  - Expose data not visible at the PMPI level
    - Point-to-point traffic generated by collectives
    - Exact traffic generated at the BTL (Channel in MPICH), includding overheads

Overview

MPIT Plugin

Initialization:

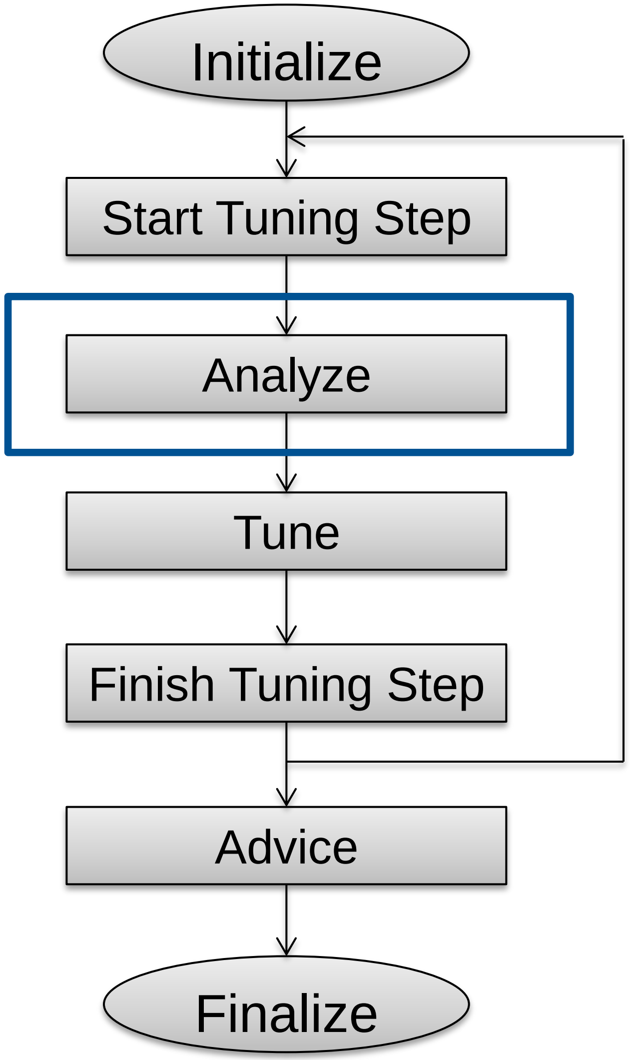
- The plugin acquires static information about the application
  - MPI_Init and MPI_Finalize locations
  - Point to point calls
  - Collective operation calls
  - Currently one-sided operations are not supported
- MPI_Init, MPI_Finalize and not supported operations are filtered out



MPIT Plugin

Analysis:

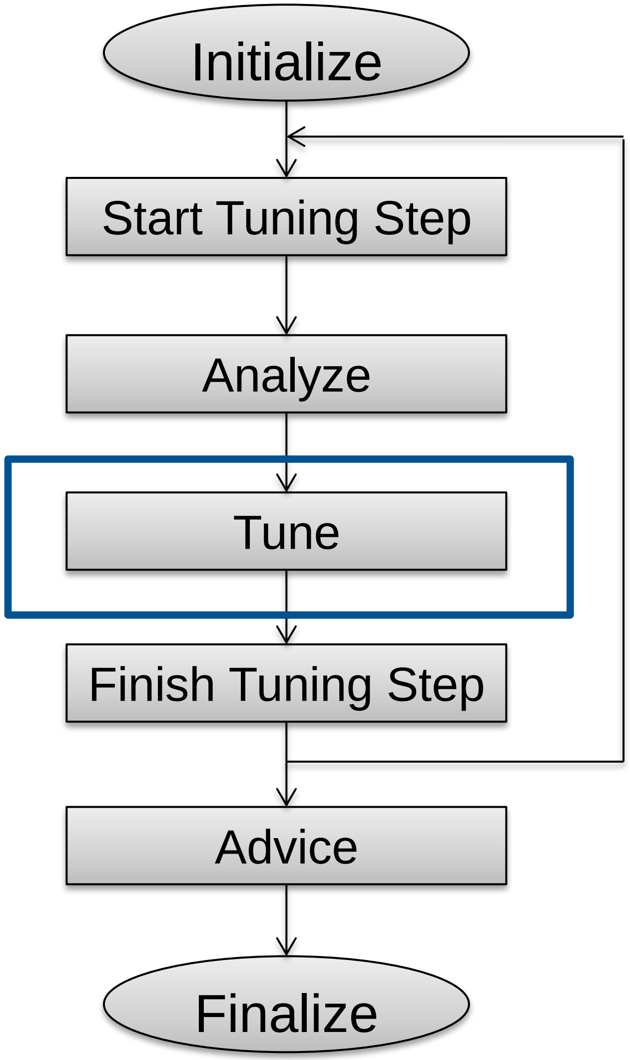
- An MPI analysis is requested, in order to evaluate the following performance properties of the application:
  - Near threshold dependency,
  - Frequent MPI call,
  - Late sender condition, etc.
- The analysis results are processed and a search space and related data structures are prepared
- *This analysis is only performed in the first tuning step, for this plugin*



MPIT Plugin

Tuning:

- Odd steps:
  - Tune point to point parameters
    - Inter- and intra thresholds
    - Eager, and LMT if enabled
- Even steps:
  - Each internal collective is evaluated
    - Per enabled call site (code region)
- In both cases:
  - Collect low level BTL byte values
    - In contrast to PMPI based tools
  - Update a communication matrix



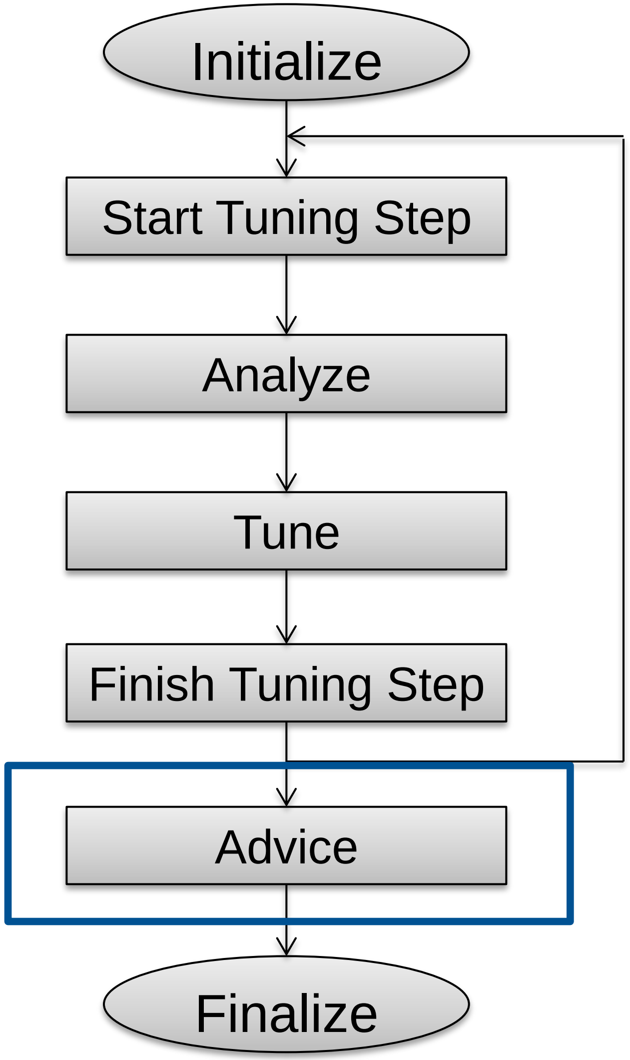
MPIT Plugin

Tuning steps:

- The number of pair of tuning steps is configurable:
  - Correlation between parameters for point to point communication and collectives
  - Measurement samples; variance reduction

Advice:

- Generates a summary of the search
- Posts the best found values for each MPI call explored
- Performs a bandwidth reduction on the communication matrix (with raw BTL values) and outputs a recommended topology (a hostfile for SuperMUC)



Overview

Test Systems

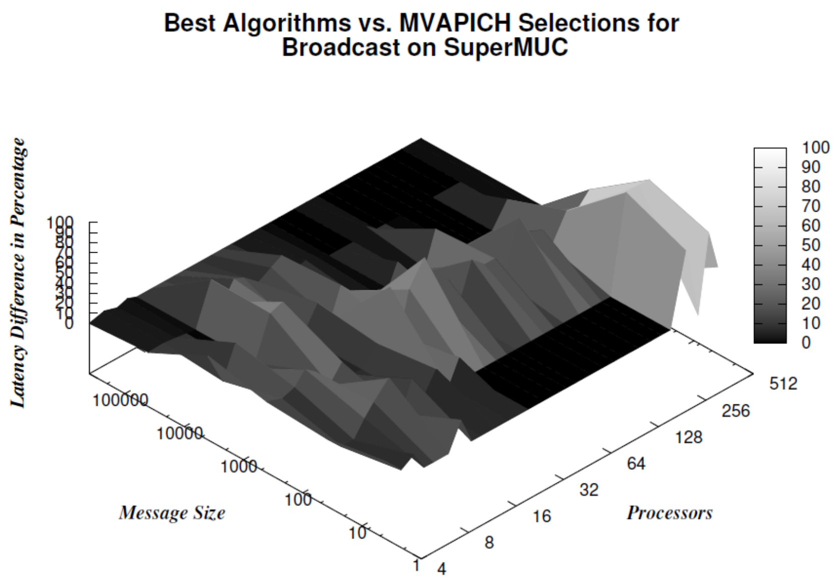
SuperMUC specifications:

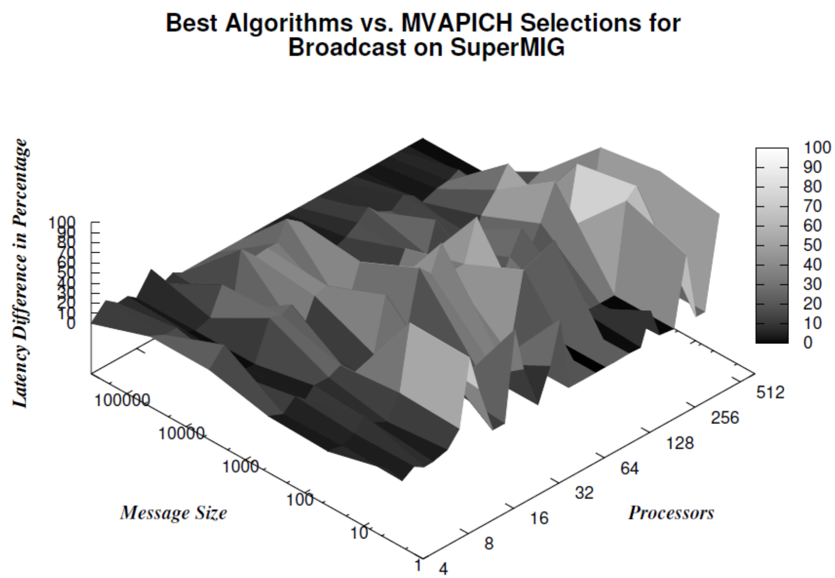
- CPU:
  - Xeon E5-2680
  - 20M Cache,
  - 8 Cores
    - 2.7 GHz
    - 3.5 GHz
- Thin nodes:
  - 2 * 8 cores
  - 32GB RAM
  - Infiniband FDR10

SuperMIG specifications:

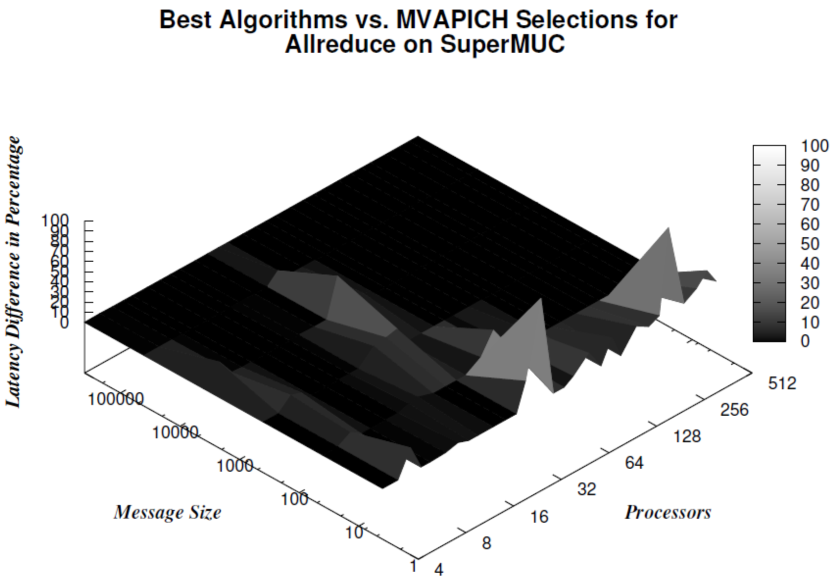
- CPU:
  - Xeon E7-4870
  - 30M Cache,
  - 10 Cores
    - 2.4 GHz
    - 2.8 GHz
- Fat nodes:
  - 4 * 10 cores
  - 256GB RAM
  - Infiniband FDR10

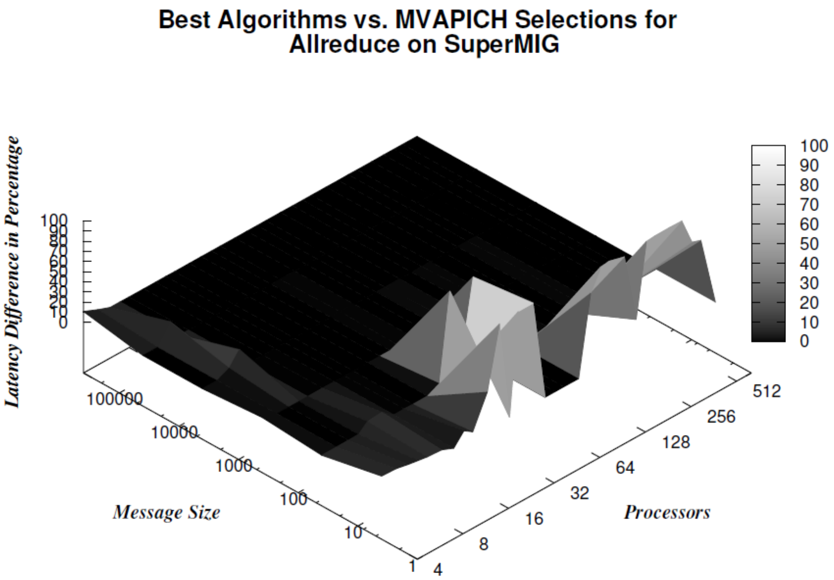
MPI_Broadcast



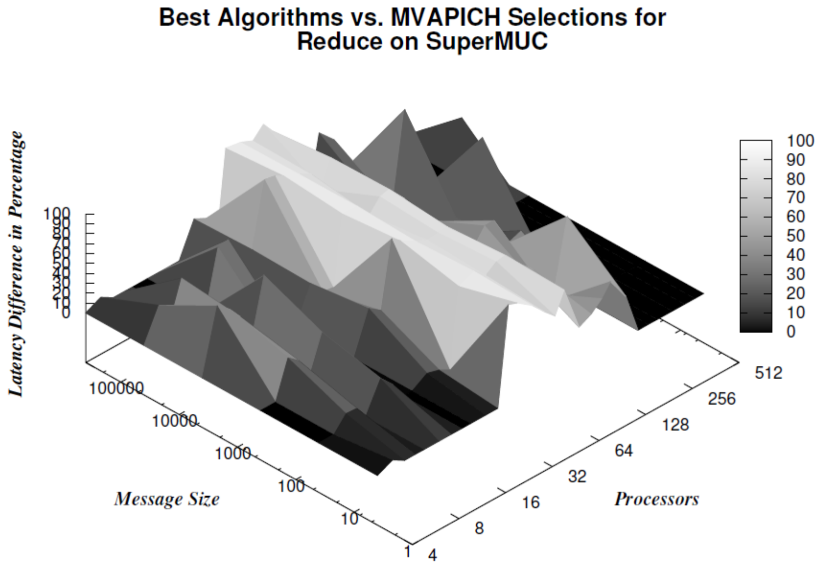


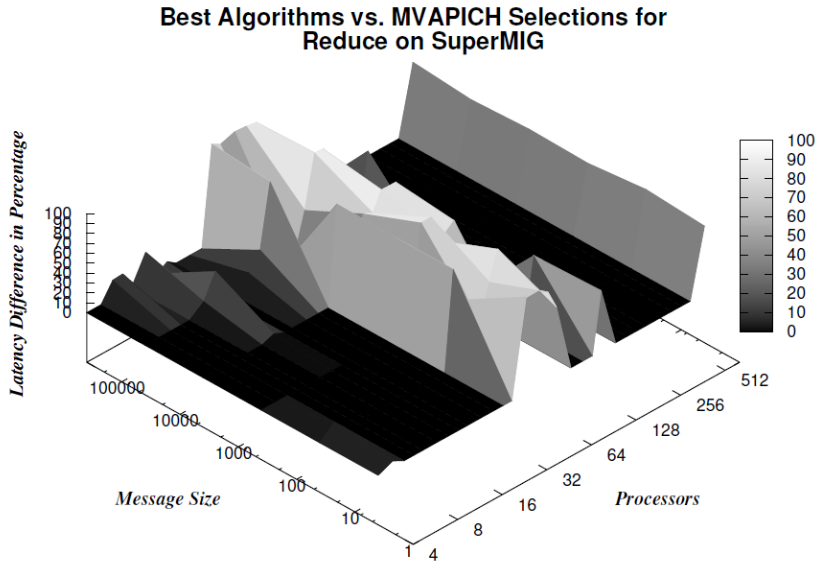
MPI_Allreduce





MPI_Reduce


Best Algorithms vs. MVAPICH Selections for
Reduce on SuperMUC
Latency Difference in Percentage
Message Size
Processors
100
90
80
70
60
50
40
30
20
10
0
100000
10000
1000
100
10
1
4
8
16
32
64
128
256
512


Best Algorithms vs. MVAPICH Selections for
Reduce on SuperMIG
Latency Difference in Percentage
Message Size
Processors
100
90
80
70
60
50
40
30
20
10
0
100000
10000
1000
100
10
1
4
8
16
32
64
128
256
512

Overview

Conclusion

MPI-T:

- The interface has been great addition for tool developers
- CVARS can be, in many cases, safely modified at runtime
  - This can be used to accelerate the search process of automatic tuners

Runtime parameters:

- Require no change to the application source code
- Once an application is built with a supporting MPI library, no additional recompiles or relinks are necessary
- Can have significant effects in performance, but only under certain conditions (as seen with the parameter sweeps)

Periscope Tuning Framework:

- Generic framework for automatic tuning
- Supports the mixed use of analyses, modeling and search approaches