MPIT: A New Interface for Performance Tools in MPI 3

(A Proposal from the MPI-3 Tools WG)

Martin Schulz, CASC/ LLNL

&

MPI-3 Working Group on Tools

Lawrence Livermore National Laboratory, P. O. Box 808, Livermore, CA 94551

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

Tool Interfaces in MPI

- Standardized interfaces for tools are essential
  - Otherwise: hard to create portable tools
  - Many ad-hoc solutions
- Positive example: PMPI
  - Standardized interposition mechanism for MPI
  - Required by the standard / Available everywhere
  - Used by a large number of tools
- BUT: PMPI only covers part of what tools need
  - Only intended as a first party interface
  - Application level abstraction
  - No insight into the MPI library
- **Chance to change this in MPI-3**

Charter of the Tools Working Group

- Define interfaces for development tools
  - Performance analysis, debuggers, correctness tools
  - Insight into MPI library
  - Understand interactions between MPI and application
  - Determine application execution environment
- Motivation:
  - Provide the basis for reliable and portable tools
  - Provide new functionality not covered by PMPI
- All efforts are intended as an extension to PMPI

https://svn.mpi-forum.org/trac/mpi-forum-web/wiki/MPI3Tools

Current Activities

- MPI Performance Information Interface (MPIT)
- Documentation of the current MPIR interface
  - Automatic Process Acquisition Interface
- Additional 3rd party interfaces
  - New, scalable version of an APAI for MPI-2/3
  - Debugger DLL naming
  - Handle query interfaces
- Potential future items
  - Extensions of the MPI_Pcontrol mechanism
  - Piggybacking (together with the FT WG)
  - Low level tracing option in MPIT

MPIT: A Performance Information Interface for MPI

- Enable tools access to internal MPI information
  - Information about MPI configuration
  - Internal performance information
    - Queue management
    - Memory consumption
- Usage models
  - Portable configuration tools
  - PMPI tools with access to internal data
  - Self monitoring applications & Autotuning
  - Portable communication tool <-> application
  - PAPI-C component

Central Concepts

- An MPI implementation decides what to expose
  - No restrictions on the implementation
  - Flexibility to expose implementation specific details
  - Option to provide production and debug versions
  - Requirement: API to provide additional information in a portable way to convey some semantics
- Variables
  - Data is conveyed through a set of variables
  - Configuration and performance variables
  - Separate type system to avoid initialization problems
- C bindings only / No Fortran bindings

Three + One Sections

Configuration variables

- Settings for runtime behavior (typical: env. vars.)
- Options to read / to set?

Performance variables

- Internal performance information of the MPI library
- Virtualized start/stop counters

Initialization, finalization, and type system

- Separate MPIT from MPI
- Separate sessions for performance variables

Initially planed, but put aside: low-level tracing interface

- Log internal events to ring buffer

Configuration Variables

- User controlled configuration mechanisms
  - Often in the form of environment variables
  - Tailor behavior to a machine or algorithm
  - Typically barely documented, hard to understand
- First section of MPIT allows access to those variables
  - Query all existing variables
  - Query default and current settings
  - Ideally: application specific control
- Sample use cases
  - Tool to list all configuration options
  - Document execution parameters
  - Configuration and optimization all the way to autotuning

Detecting Configuration Variables

- Two step process
  - Find all available variables
  - Query detailed information about a variable
- Step 1: Iterator approach to discover all variables
  - Initialize iterator with a constant
  - Pass the same iterator to repeating calls until done
  - Each invocation returns one variable name
  - Gather or search for variable names
- Step 2: Query additional information
  - Detailed description
  - Type and size information

API Calls for Variable Detection

- **MPIT_CONFIG_FIND**
  - IN/OUT: iterator
  - OUT: name/namelen (string)
  - IN: maxverbosity
- **MPIT_CONFIG_QUERY**
  - IN: name
  - OUT: datatype/count
  - OUT: description/length (string)
  - OUT: default value
  - OUT: verbosity
- **MPIT_CONFIG_VARS_CHANGED**
  - OUT: flag

Verbosity Levels

- MPIs could return large number of variables
  - Restrict the number of returned variables
  - Categorize all variables based on use case
- Verbosity levels in increasing order
  - **MPIT_VERBOSITY_USER_BASIC**
  - **MPIT_VERBOSITY_USER_DETAILED**
  - **MPIT_VERBOSITY_TUNER_BASIC**
  - **MPIT_VERBOSITY_TUNER_DETAILED**
  - **MPIT_VERBOSITY_MPI_DEVELOPER**
- Query call only return variables that at the requested verbosity level or lower

String Interface

- Many MPIT calls rely on return strings
  - Widely different lengths (name vs. descriptions)
- Currently not handled well in MPI
  - Predefined maximal length for a string as a constant
  - MPI returns actual length copied into buffer
  - Not efficient for widely varying string sizes
  - Danger of buffer overflows
- Proposal to use a different approach
  - IN/OUT parameter for length to indicate size of buffer
  - If zero is passed in, MPI returns actual size of string
  - Automatic truncation if buffer is too small

Reading and Writing Configuration Variables

- Once configuration variables are identified
  - Ability to query configuration settings
  - Document default/current settings
  - Adjust configuration
  - Opportunity f or auto tuning
- **MPIT_CONIG_GET**
  - IN: name (string)
  - OUT: buf (with type/size as provided by query call)
- **MPIT_CONFIG_SET**
  - IN: name(string)
  - OUT: buf (with type/size as provided by query call)

Issues with Writing Configuration Variables

- Writing configuration variables has complications
  - Access from multiple tools
  - Changes may require global synchronization
  - WG currently favors avoiding writes
  - Strong concrete use cases?
- Options to avoid racing updates
  - Request exclusive write access during **MPIT_INIT**
  - Locking mechanisms
- Options to deal with globally synchronizing changes
  - Make **MPIT_CONFIG_SET** a collective call
  - Batching updates before applying them

Configuration Variables API Overview

MPIT_CONFIG_FIND (iter,name,maxverbose)

Iterator to find all configuration variables

MPIT_CONFIG_QUERY (name,desc,type/size,default,verbose)

Query description and type of a variable

MPIT_CONFIG_VARS_CHANGED (flag)

Notification that the list has changed

MPIT_CONFIG_GET (name,buffer)

Read the value of a configuration variable

MPIT_CONFIG_SET (name,buffer)

Write a value to a configuration variable

Performance Variables

- Access to MPI internal performance information
  - Queue length, memory footprint, matching time, …
  - MPI defines a set of variables that are available and updated by the MPI library at runtime
  - MPIT can query (and control) these variables
- Sample use cases:
  - Performance studies (find bottlenecks inside of MPI)
  - Scalability studies (e.g., wrt. Memory)
- Same query process as with configuration variables
  - Step 1: Iterator-based detection of available data
  - Step 2: Ability to query information for each variable

API for Variable Detection

- **MPIT_PERFORMANCE_FIND**
  - IN/OUT: iterator
  - OUT: name/namelen (string)
  - IN: maxverbosity
- **MPIT_PERFORMANCE_QUERY**
  - IN: name
  - OUT: class of the performance variable / type of information
  - OUT: datatype/count
  - OUT: description/length (string)
  - OUT: readonly (can not be reset)
  - OUT: continuous (can not be started/stopped
  - OUT: verbosity
- **MPIT_PERFORMANCE_VARS_CHANGED**
  - OUT: flag

Classes of Performance Variables

- **MPIT_VARCLASS_STATE**
  - Snapshot of a discrete state (e.g., work/message/wait)
- **MPIT_VARCLASS_UTILIZATION**
  - Utilization of a finite resource (e.g., buffer) (range 0.0-1.0)
- **MPIT_VARCLASS_RESOURCE**
- **MPIT_VARCLASS_HIGHWATERMARK**
- **MPIT_VARCLASS_LOWWATERMARK**
  - Absolute utilization of a resource in MPI (current/high/low)
- **MPIT_VARCLASS_COUNTER**
  - Monotonically increasing counter of a specific set of events (+1)
- **MPIT_VARCLASS_AGGREGATE**
  - Aggregate value over time of a specific event parameter (+N)
- **MPIT_VARCLASS_TIMER**
  - Aggregated time of a set of events

Handle API for Performance Variables

- Performance variables require active handles
  - Allocation and deallocation routines
  - Easier for optimizations
  - Access routines require handles

- **MPIT_PERFORMANCE_HANDLE_GET**
  - IN: name (string)
  - OUT: handle
- **MPIT_PERFORMANCE_HANDLE_FREE**
  - IN: handle

Counter API

- All counter access require a valid handle to the counter
- Start/Stop API
  - **MPIT_PERFORMANCE_START**
  - **MPIT_PERFORMANCE_STOP**
  - IN: handle (**MPIT_ALL_COUNTERS** would also be OK)
- Read/Write/Reset API
  - **MPIT_PERFORMANCE_READ**
  - **MPIT_PERFORMANCE_RESET**
  - **MPIT_PERFORMANCE_READRESET**
  - **MPIT_PERFORMANCE _WRITE**
  - IN: handle of counter
  - IN: application buffer of correct type (for read & readreset)

Performance Variables API Overview

MPIT_PERFORMANCE_FIND (iter,name,maxverbose)

Iterator to find all performance variables

MPIT_PERFORMANCE_QUERY (name,…)

Query description and type of a variable

MPIT_PERFORMANCE_VARS_CHANGED (flag)

Notification that the list has changed

MPIT_PERFORMANCE_HANDLE _GET(name,handle)

Request a handle for a variable

MPIT_PERFORMANCE_HANLDE_FREE (handle)

Release a handle

Performance Variables API Overview (cont.)

MPIT_PERFORMANCE_START(handle)

Start recording data of one/all variables

MPIT_PERFORMANCE_STOP(handle)

Stop recording data of one/all variables

MPIT_PERFORMANCE_READ(handle,buffer)
MPIT_PERFORMANCE_READRESET(handle,buffer)
MPIT_PERFORMANCE_RESET(handle)

Read and/or reset the value of a variables

MPIT_PERFORMANCE_WRITE(handle,buffer)

Set the value of a performance variable

Hierarchy and Group Information

- Variables should be structured
  - Semantic information describing variables
  - Grouping and hierarchical relationships
  - Needs to be provided by the MPI library
- Structured as hierarchical sets
  - Variables are grouped in sets
  - Sets are grouped in sets
- Routines to query set structure
  - Iterator based
  - Top-down and bottom-up options
  - Additional textual descriptions

Variable Taxonomy API

MPIT_TAXON_QUERY_SET_VARIABLES (iter,var,set,type)

Iterator to find all sets containing a variable

MPIT_TAXON_QUERY_VARIABLE_SETS (iter,var,set,type)

Iterator to find all variables contained in a set

MPIT_TAXON_QUERY_SET_SETS (iter,set,set)

Iterator to find all sets contained in a set

MPIT_TAXON_CHANGED (flag)

Taxonomy information has changed

MPIT_TAXON_DESCRIBE_SET(iset,desc)

Get additional information for a particular set

Initialization of MPIT

- Initialization separate from MPI itself
  - Not bound to MPI_Init / MPI_Finalize
  - New calls: **MPIT_Init**/**MPIT_Finalize** (similar semantics)
  - Can be run before Init and after Finalize
- Multiple, nested initialization
  - Allow multiple tools to gather MPIT data
  - Each usage requires separate Init/Finalize pair
  - Usage counter
- Open questions:
  - Is this sufficient for a clean co-existence of tools?
  - Is it necessary to have access to argv/argc?

Session Management

- How to enable multiple tools to gather performance data?
  - Identify which tool requested which counters
  - Provide consistent counter data
- Explicit sessions for performance counters
- **MPIT_PERFORMANCE_SESSION_CREATE**
  - OUT: session handle
  - Create session and return session specific handle
- **MPIT_PERFORMANCE_SESSION_DESTROY**
  - IN: session handle
  - End session and delete all handles
- Session argument for every **MPIT_PERFORMANCE_** call

MPIT Type System

- Type system separated from MPI
  - Not guaranteed to be initialized before MPI_Init
  - Complex datatypes unnecessary
- MPIT offers a set of basic types
  - **MPIT_BYTE, MPIT_SHORT, MPIT_LONG, MPIT_INT, MPIT_LONG_LONG, MPIT_CHAR, MPIT_FLOAT, MPIT_DOUBLE**
  - Each type has an equivalent MPI base type
- Enumeration types
  - Fixed number of values (similar to enum)
  - **MPIT_TYPECLASS_GET**
    - Returns whether a type is an enumeration type or not

MPIT Enumeration Types

- Each element of an enumeration carries semantics
  - MPI would benefit from being able to get such semantics
  - Query interface for descriptions of each element
- **MPIT_ENUMTYPE_QUERY**
  - IN: type (must be an enumeration type)
  - OUT: number (number of elements in the set)
  - OUT: name of the enumeration type
- **MPIT_ENUMITEM_QUERY**
  - IN: type (must be an enumeration type)
  - IN: number (item number to query)
  - OUT: description string

Init/Finalize/Types API Overview

MPIT_INIT (instance)

Initialize the interface for use

MPIT_FINALIZE (instance)

Finish using this instance of MPIT

MPIT_PERFORMANCE_SESSION_CREATE (session)

Create a new session for performance variables

MPIT_PERFORMANCE_SESSION_DESTROY (session)

Delete a new session for performance variables

MPIT_TYPECLASS_GET (type, class)

Determine whether a type is an enumeration type

MPIT_ENUMTYPE (type, number, name)
MPIT_ENUMITEM (type, itemno, desc)

Query information about an enumeration type

MPIT Status

- API has been presented to MPI forum before
  - January meeting
  - Feedback is being integrated
- Draft currently being reworked
  - Multiple initialization/finalizations
  - Adding session management
  - Group and hierarchy information
- Timeline
  - Final discussions in the next weeks
  - Presentation to the MPI Forum at October Meeting
  - First reading + votes after that

Conclusions

- With MPI-3 we have the chance to get more tools interfaces
  - Going beyond the successful PMPI interface
  - For debugging, performance analysis, correctness tools
- MPIT: a new way of getting performance data from MPI
  - Portable access to performance and configuration data
  - Based on a query interface (MPI library independent)
  - Support for interface virtualization
- **Feedback on MPIT needed now**
  - Extensions discussions in the WG
  - Presentation and buy in from MPI forum
  - Need more feedback from the tools community!