Lawrence Livermore National Laboratory

Implementation
MPI Tool Information Interface

Slides by Christof Klausecker

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

LLNL-PRES-508571

Overview

- First implementations of the new MPI Tool Information Interface

- Currently two implementations available:
  - MPICH2
    - *Dave Goodell* (ANL)
    - Patch submitted to MPI Forum Tools Mailinglist
  - Generic framework with focus on MVAPICH2
    - *Christof Klausecker* (LLNL & LMU)
    - To be released soon

MVAPICH2 based Implementation (LLNL/LMU)

- Based on the document from September 22nd (final version after formal reading)
- Implementation divided into
  - Generic framework
  - MPI implementation specific* part
- Focus on MVAPICH2 1.7 (CH3 based)
  - OFA-IB-CH3 (OpenFabrics libibverbs)
  - PSM-CH3 (Qlogic PSM)

* 3rd Version of the Channel Interface (CH3) – Layer to hide ADI-3 complexity

Implementation Structure

- Generic Framework

mpi_t.c	• Interface Implementation
mpi_t.h	• Interface Definition
mpi_t_types.h	• Internal Types

- Implementation dependent

variables.c	• Definition of Implementation Specific Variables

GENERIC MPI_T FRAMEWORK

Framework Implementation

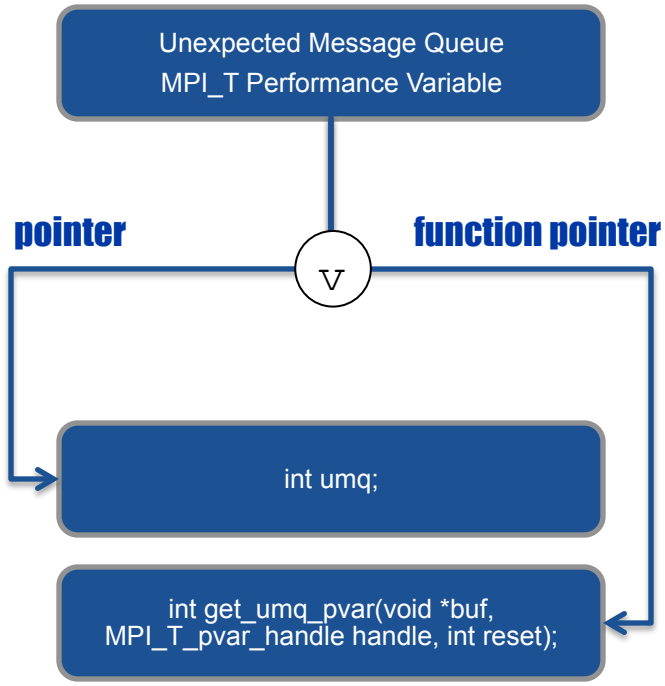
- Framework implementation takes care of managing control and performance variables
  - Handle allocation and object binding
  - Accessing variables
  - Starting/Stopping/Resting variables
  - Session Management
  - …
- Adding new variables can be accomplished fairly easy
  - Control and Performance Variables are exposed to the framework via pointers to global variables or functions

Implementation Structure

- Generic Framework

```
int MPI_T_pvar_get_num(int *num_pvar);

int MPI_T_pvar_get_info(int pvar_index, char *name, int
*name_len, int *verbosity, int *var_class, MPI_Datatype
*datatype, MPI_T_enum *enumtype, char *desc, int
*desc_len, int *bind, int *readonly, int *continuous);

int MPI_T_pvar_session_create(MPI_T_pvar_session
*session);

int MPI_T_pvar_session_free(MPI_T_pvar_session
*session);

int MPI_T_vvar_handle_alloc(MPI_T_pvar_session session,
int pvar_index, void *obj_handle, MPI_T_pvar_handle
*handle, int *count);

int MPI_T_pvar_handle_free(MPI_T_pvar_session session,

MPI_T_pvar_handle *handle);

int MPI_T_pvar_start(MPI_T_pvar_session session,
MPI_T_pvar_handle handle);

int MPI_T_pvar_stop(MPI_T_pvar_session session,
MPI_T_pvar_handle handle);

int MPI_T_pvar_read(MPI_T_pvar_session session,
MPI_T_pvar_handle handle, void* buf);

int MPI_T_pvar_write(MPI_T_pvar_session session,
MPI_T_pvar_handle handle, const void* buf);

...
```

- Implementation dependent



CONTROL VARIABLES

Control Variables

- Numerous variables
  - Available in unmodified MVAPICH2 source code
  - Typically exposed as environment variables
  - Just add description and pointer to global variables

    Example: variables can be found in ibv_param.h for OFA-IB-CH3
- Examples:
  - MV2_NUM_HCAS
    number of adapters to be used
  - MV2_USE_COALESCE
    enables message coalescing
  - MV2_IBA_EAGER_THRESHOLD
    eager protocol threshold

Control Variable - Fields

external fields (exposed via interface)	internal fields (for implementation only)
char name[255]	int count
int name_len	unsigned char type
int verbosity	
MPI_Datatype datatype	void *variable
int enumtype	int (*function)(void*, MPI_T_cvar_handle, int)
char desc[255]	
int desc_len	
int bind	
int scope	

Control Variable – Internal Fields

Implementation internal fields to store

- Type count in case it is fixed
- Internal type
  (always_on, pre_init, post_init)
- Pointer to the variable
  (NULL in case function)
- Pointer to the function providing the variable
  (NULL in case variable)

internal fields (for implementation only)

int count

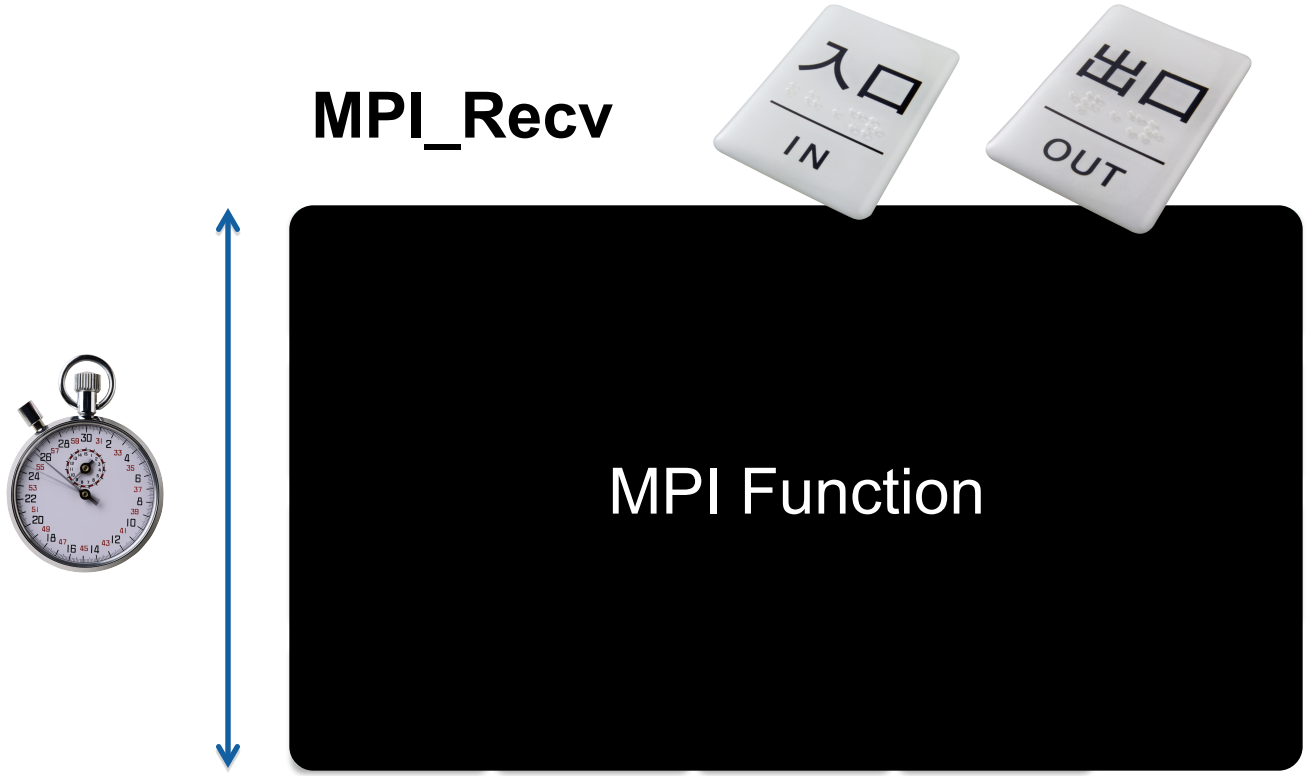
unsigned char type

void *variable

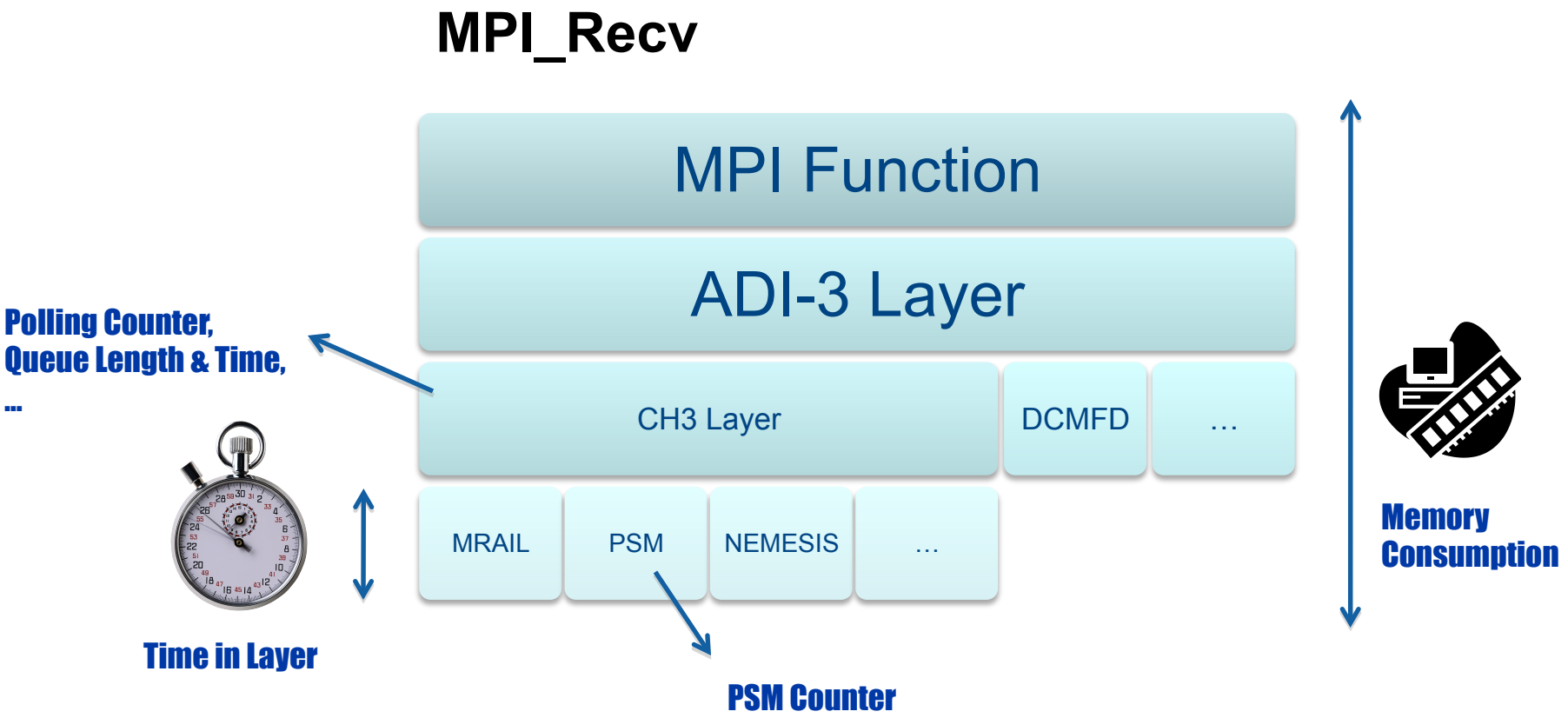
int (*function)(void*, MPI_T_cvar_handle, int)

PERFORMANCE VARIABLES

Granularity of PMPI Information



Granularity of MPI_T Information



Performance Variables

- Several MVAPICH2 performance variables can be exposed from the unmodified source code, some require changes
- Examples:
  - PSM Counters
  - MPI memory consumption
  - Unexpected message queue (UMQ)
  - Posted receive queue (PRQ)
  - Polling Counter for blocking operations
  - …

Performance Scaled Messaging (PSM) Interface

- Provides several internal performance counters
- Counters are activated by default - no additional runtime overhead

PSM COUNTERS
Total SENDS
Total RECVS
Total pre-posted receives
Total eager PUTS
Total eager GETS
Total rendezvous PUTS
Total rendezvous GETS
Total ACCUMULATES

Performance Variables – Fields

external fields (exposed via interface)	internal fields (for implementation only)
char name[255]	int count
int name_len	unsigned char type
int verbosity	
int var_class	int *allocated
MPI_Datatype datatype	int *started
int enumtype	MPI_T_pvar_handle handles;
char desc[255]	
int desc_len	void *variable
int bind	int (*function)(void*, MPI_T_pvar_handle, int)
int readonly	void *defaultvalue
int continuous	
int atomic	

Performance Variables – Internal Fields

Implementation internal fields to store

- Type count in case it is fixed
- Internal type
  (always_on, pre_init, post_init)
- Counter, tracking times allocated
- Counter, tracking times started

internal fields (for implementation only)

int count

unsigned char type

int *allocated

int *started

MPI_T_pvar_handle handles;

void *variable

int (*function)(void*, MPI_T_pvar_handle, int)

void *defaultvalue

Performance Variables – Internal Fields

Implementation internal fields to store

- List of related allocated handles
- Pointer to the variable
  (NULL in case function)
- Pointer to the function providing the variable
  (NULL in case variable)
- Pointer to a defaultvalue

internal fields (for implementation only)

```
int count
unsigned char type

int *allocated
int *started
MPI_T_pvar_handle handles;

void *variable
int (*function)(void*,
MPI_T_pvar_handle, int)
void *defaultvalue
```

Example:

ADDING A PERFORMANCE VARIABLE

Adding PVAR UMQ MAX to MPI_T Framework

void add_pvar_umq()

```
pvaritem_t tmp;
int id;
char name[] = “Unexpected Message Queue (MAX)";
char desc[] = “…";
tmp.name_len = LENGTH(name);
memcpy(tmp.name, name, tmp.name_len);
tmp.verbosity = MPI_T_VERBOSITY_USER_ALL;
tmp.var_class = MPI_T_PVAR_CLASS_ HIGHWATERMARK;
tmp.datatype = MPI_INT;
tmp.enumtype = MPI_T_ENUM_NULL;
tmp.desc_len = LENGTH(desc);
memcpy(tmp.desc, desc, tmp.desc_len);
```

```
tmp.bind = MPI_T_BIND_NO_OBJECT;
tmp.readonly = 1;
tmp.continuous = 0;
tmp.atomic = 0;

tmp.count = 1;
tmp.variable = &pvar_umq_max;
tmp.function = NULL;

tmp.allocated = &pvar_umq_max _allocated;
tmp.started = & pvar_umq_max _started;
tmp.defaultvalue = &pvar_default_int_zero;

id = add_pvar(tmp);
add_cat_pvar(cat_mapper[MPI_T_CAT_ALL], id);
```

Adding PVAR UMQ MAX to MVAPICH2 CH3 Layer

- Simple Counter
  - Minimal overhead

    For more intrusive variables „started" field can be used to enable/disable
  - MPI_T framework manages copies

    necessary for startable, stoppable and resettable variables

FDx – find deqeue
AEx – allocate enqueue
U – Unexpected Queue
P – Posted Queue

```
int pvar_umq_current = 0;
int pvar_umq_max = 0;

MPIDI_CH3U_Recvq_FDU{
   …
   pvar_umq_current--;
   …
}
MPIDI_CH3U_Recvq_FDU_or_AEP(){
   …
   pvar_umq_current--;
   …
}
MPIDI_CH3U_Recvq_FDP_or_AEU(){
   …
   pvar_umq_current++;
   if (pvar_umq_current > pvar_umq_max)
      pvar_umq_max = pvar_umq_current;
   …
}
```

CATEGORIES (3 OPTIONAL SLIDES)

Categories

external fields (exposed via interface)	internal fields (for implementation only)
char name[255]	int *controlvars
int name_len	int num_controlvars_alloc
char desc[255]	
int desc_len	int *perfvars
int num_controlvars	int num_perfvars_alloc
int num_perfvars	
int num_categories	int *categories
	int num_categories_alloc

Categories

Implementation internal fields to store the indices of

- Performance Variables
- Control Variables
- Subcategories

internal fields (for implementation only)

```
int *controlvars
int num_controlvars_alloc

int *perfvars
int num_perfvars_alloc

int *categories
int num_categories_alloc
```

TOOL INTEGRATION

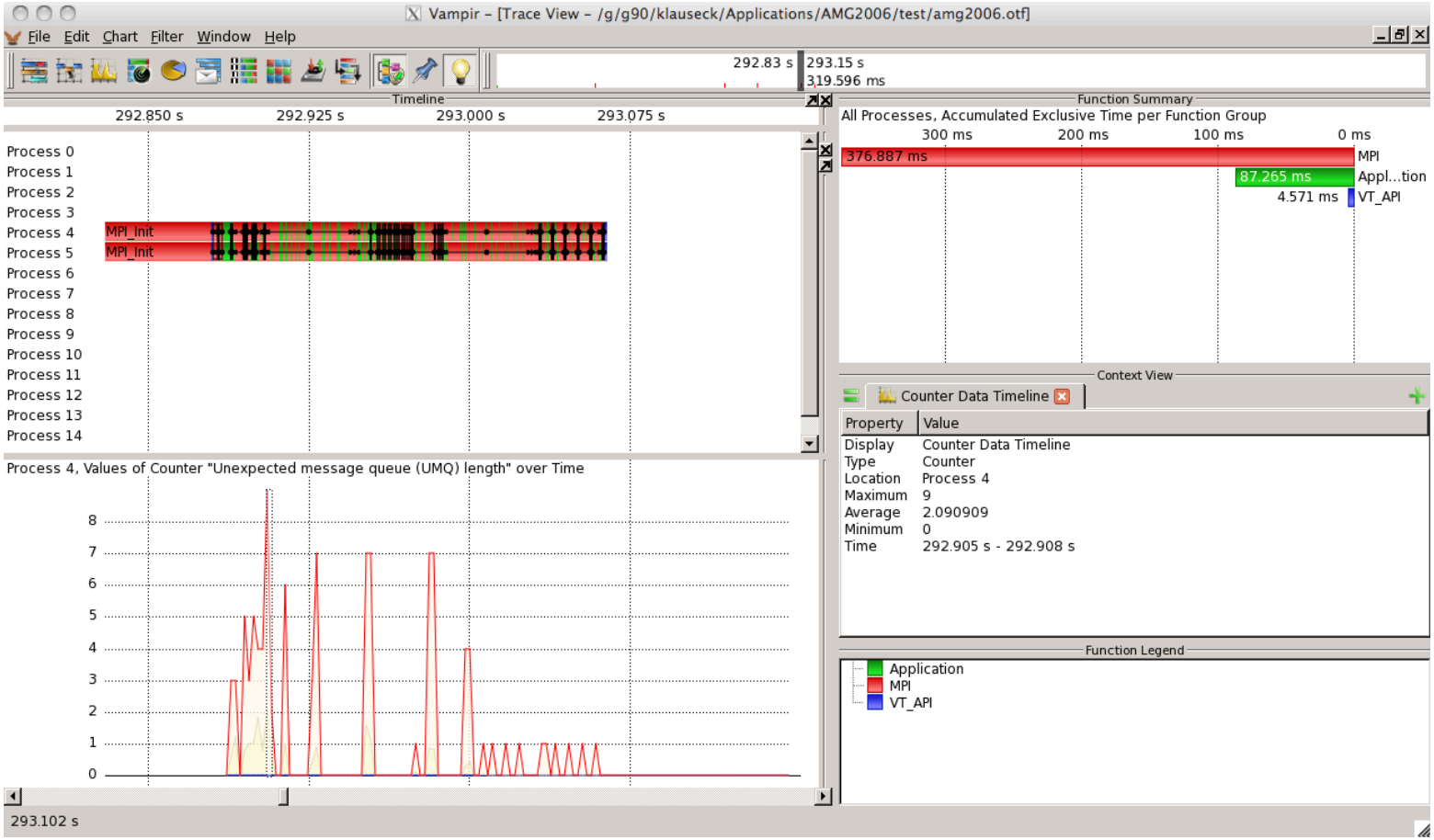
Tool Integration

- Simple PMPI Tool
  - Outputs CVARS on Rank 0 during MPI_Init
  - Starts Session and PVARS during MPI_Init
  - Outputs PVARS during MPI_Finalize (no bind vars)
- Tracing
  - VampirTrace **MPI_T functionality not available in official builds!!!**
    http://www.tu-dresden.de/zih/vampirtrace/
- Profiling
  - IPM **MPI_T functionality not available in official builds!!!**
    http://ipm-hpc.sourceforge.net/

Tool Integration

- VampirTrace
  - MPI_T variables stored as OTF counter events
  - Control Variables
    - Collected at MPI_Init
    - To review settings used while tracing
  - Performance Variables (selection)
    - Collected during selected MPI calls
- IPM
  - MPI_T variables (selection) gathered during MPI calls
  - Summarized data stored per callsite
    - MIN
    - MAX
    - AVG
    - Adaptive histogram

Vampir and MPI_T data



Summary

- First two MPI tool information interface implementations available (for MPICH2 and MVAPICH2)
- Impact on performance negligible
- As proof-of-contept, the MPI tool information interface has been integrated into two performance analysis tools
  - Profiling Tool - IPM
  - Tracing Tool - VampirTrace