MPI wrapper

Paul Müller

April 30, 2009

1. MPI Wrapper
   - Purpose
   - Goals
   - Trace Format
2. Components
   - Overview
   - HDTraceWritingCLibrary
   - HDMPIwrapper
   - Project Description Merger
3. Concepts
   - Topology
   - Files
   - Communicators
   - Datatypes
   - Non-blocking Requests
4. Future Work

Content

1. MPI Wrapper
   - Purpose
   - Goals
   - Trace Format
2. Components
   - Overview
   - HDTraceWritingCLibrary
   - HDMPIwrapper
   - Project Description Merger
3. Concepts
   - Topology
   - Files
   - Communicators
   - Datatypes
   - Non-blocking Requests
4. Future Work

Purpose

Trace the essential parts of the execution of an MPI program.

Purpose

Trace the essential parts of the execution of an MPI program.

Goals

- Human-readable xml output
- usable with Open MPI and MPICH
- primary use: PIOsim
- no interference with the program's MPI communication

Purpose

Trace the essential parts of the execution of an MPI program.

What is essential?

- Function calls
- Interaction information (communicators, tags)
- Timing (or “computation time”)
- Datatypes → transmission size
- Files

optional

- nested calls

Not important

- actual data
- program logic

local data

- Function calls
- *Time*

Local logging; no further modification of logfile

shared data

- Files
- MPI communicators
- *Datatypes*

Local logging; Data is assembled during postprocessing

Files that compose the trace

- program_node01_0_0.trc ... program_node03_9_0.trc
- program.proj

.trc files

- local data
- naming:

  <program name>_<hostname>_<rank>_<thread>.trc

.proj files

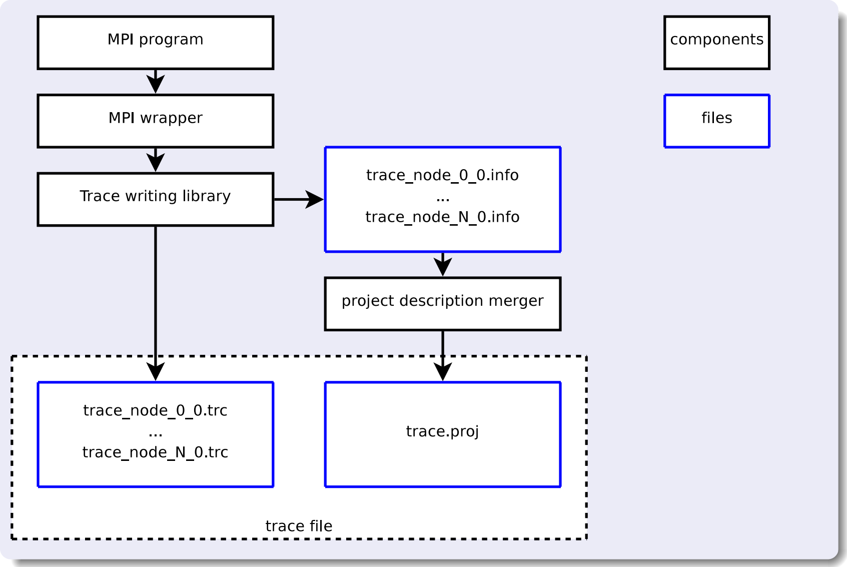
- which files belong to the trace
- which resources are used
- naming:

  <program name>.proj

Content

1. MPI Wrapper
   - Purpose
   - Goals
   - Trace Format
2. Components
   - Overview
   - HDTraceWritingCLibrary
   - HDMPIwrapper
   - Project Description Merger
3. Concepts
   - Topology
   - Files
   - Communicators
   - Datatypes
   - Non-blocking Requests
4. Future Work

- HDTraceWritingCLibrary Writing formatted log files (in collaboration with Stephan Krempel)
- HDMPIwrapper intercepting MPI calls
- Project Description Merger Generate a project description from individual trace files


MPI program
MPI wrapper
Trace writing library
trace_node_0_0.info
...
trace_node_N_0.info
project description merger
trace_node_0_0.trc
...
trace_node_N_0.trc
trace.proj
trace file
components
files

HDTraceWritingCLibrary

- management of the .info file
- management of the .trc log file
- abstraction layer for writing .trc xml log
- ensure correct syntax
- log correct time using gettimeofday()

.info file

Information that needs to go into the project description.

writing the xml trace

- opening a trace structure

```
hdTrace trace = hdT_createTrace(node, topology);
```

- logging a state with attributes and elements

```
hdT_logStateStart(trace, "StateName");
hdT_logAttributes(trace, "cid='%d' , comm_id)
hdT_logElement(trace, "Info", "key='%s' value='%s'"
        , key, value);
hdT_logStateEnd(trace);
```

- closing the trace structure

```
hdT_finalize(trace);
```

sample output (.trc file)

```
<Program rank='0' thread='0'>
 ...
<Barrier cid='1'  time='1240837588.806989' end
    ='1240837588.806991'  />
<File_read fid='0' offset='0' size='16777216' count
    ='16777216' tid='1275068673'  time
    ='1240837588.806994' end='1240837588.816631'  />
<File_close fid='0'  time='1240837588.816638' end
    ='1240837588.816657'  />
<Allreduce size='8' cid='1' count='1' tid='1275070475'
 time='1240837588.816659' end='1240837588.816692'  />
 ...
 </Program>
```

nested calls

- What if an MPI function is implemented using other MPI functions?

writing the xml trace

- logging nested functions:

```
hdT_logStateStart(trace, "A");
// log attributes, elements of state A

  hdT_logStateStart(trace, "B");
  // log attributes, elements of state B
  hdT_logStateEnd(trace);

hdT_logStateEnd(trace);
```

sample output for nested calls (.trc file)

```
<Nested>
  <inner_mpi_function ... />
</Nested>
<outer_function ... />
```

.info files

- syntax for writing to the info file:

```
hdT_writeInfo(trace, format_string, ...);
```

- no further formatting, simple writing.

HDTraceWritingCLibrary

Also in the trace writing library (Stephan):

- statistics writing library
- topology library

HDMPIwrapper library

Task: Intercept MPI calls, log them

Method

- Create a static library that defines certain MPI functions.
- Link the library to an MPI program
- wrapper functions hide the original functions
- (Note: the wrapper depends on the implementation specific include files)

How is the call passed to MPI?

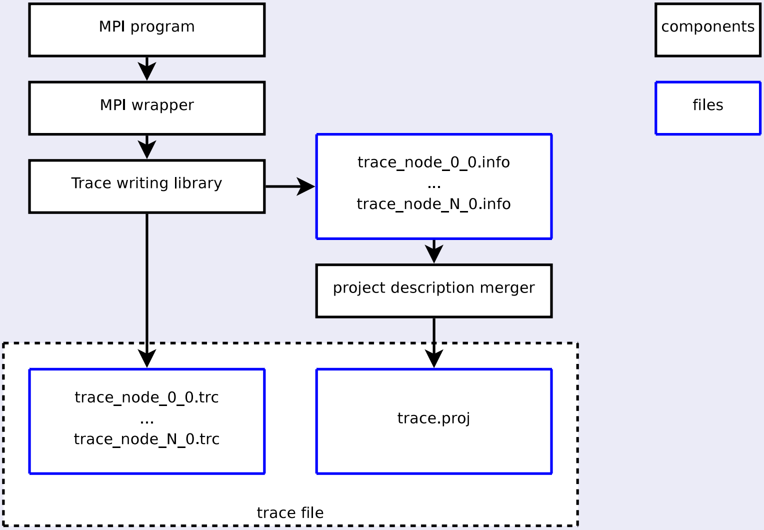
- MPI implementation: every function defined twice:
- MPI_ prefix
- PMPI_ prefix
- → hide the MPI_ function and call the PMPI_ function
- (if a program uses PMPI functions, the wrapper won't work)

typical wrapper function

```
int MPI_Send(Type1 v1, ..., TypeN vN)
{
  int ret;
  hdT_logStateStart(trace, "Send");
  ret = PMPI_Send(v1, v2, v3, v4, v5, v6);
  hdT_logAttributes(/* attributes */);
  hdT_logStateEnd(trace);
  return ret;
}
```

- Redundant structure → generate wrapper functions using a script
- Adjustable: attributes and elements, which function to log

reminder


MPI program
MPI wrapper
Trace writing library
trace_node_0_0.info
...
trace_node_N_0.info
project description merger
trace_node_0_0.trc
...
trace_node_N_0.trc
trace.proj
trace file
components
files

At program runtime

- Trace writing library + MPI wrapper produces complete .trc files.
- project description file: created after the execution.

Project Description Merger

Python script that produces a project description file using the .info files.

What is stored in the project file

Topology Which log files belong to the trace?

File list Which files are used and how are they called by different processes?

Communicator list Which communicators are used and how are they called by different processes?

Datatypes Which datatypes are used?

- Format: xml

Content

1. MPI Wrapper
   - Purpose
   - Goals
   - Trace Format
2. Components
   - Overview
   - HDTraceWritingCLibrary
   - HDMPIwrapper
   - Project Description Merger
3. Concepts
   - Topology
   - Files
   - Communicators
   - Datatypes
   - Non-blocking Requests
4. Future Work

Topology

- For program traces, the topology is given by $(\mathrm{hostname}, \mathrm{rank}, \mathrm{thread})$
- Naming of the project file:

  <program name>.proj

- Naming of the trace files:

  <program name>_<hostname>_<rank>_<thread>.trc

project description file

```
<Topology>
 <Level name="Hostname">
  <Level name="Rank">
   <Level name="Thread">
   </Level>
  </Level>
 </Level>
 <Label value="node01">
  <Label value="1">
   <Label value="0" />
  </Label>
  <Label value="0">
   <Label value="0" />
  </Label>
 </Label>
</Topology>
```

Operations on files

open, close, delete, access

Project description file

List the filename and its initial size.

Wrapper

- assign ID on first access via filename
- use hash map to store name ↔ id relationship
- the file is always referred to via the id:

```
<File_open cid=’0’ name=’filetest_02.tmp’ flags=’4’
       fid=’1’ ... />
<File_write_at_all fid=’1’ offset=’0’ size=’1’
     count=’1’ tid=’1275068673’ .../ >
```

communicators

- assign id on first access
- need to know global rank ↔ local rank map and id that is used by every rank

.trc file

```
<File_open cid='0' name='filetest_02.tmp' ... />
```

project file

```
<CommunicatorList>
  <Communicator name="">
   <Rank global="1" local="1" cid="1" />
   <Rank global="2" local="0" cid="2" />
  </Communicator>
 ...
```

Datatypes

- Also referring to datatypes via process-internal id
- Problem: combined datatypes
- Solution: recursive unwrapping of datatypes using

  ```
  MPI_Type_get_envelope
  MPI_Type_get_contents
  ```

- every process has its own datatype definitions

project file representation

```
<NAMED id="1275069445" name="MPI_INT" />
<VECTOR id="−872415229" name="" count="5" blocklength=
    "6" stride="7" oldType="1275069445" />
```

Non-blocking requests

- MPI_I∗-Functions return a request structure that can be used to wait for completion etc.
- split collective (∗_begin ... ∗_end) calls. (Maximum of one open split collective operation per file handle)

- Assign an id to every used MPI_Request and MPI_File
- ∗_end functions are transformed to Waits for the corresponding request.

non-blocking

```
<Isend size='1' count='1' tid='1275068673' toRank='2'
    toTag='0' cid='0' rid='2' ... />
<Wait ... >
  <For rid='2' />
</Wait>
```

split collective

```
<File_write_at_all_begin fid='1' rid='0' ...  />
<Wait ... >
  <For rid='0' />
</Wait>
```

Content

1. MPI Wrapper
   - Purpose
   - Goals
   - Trace Format
2. Components
   - Overview
   - HDTraceWritingCLibrary
   - HDMPIwrapper
   - Project Description Merger
3. Concepts
   - Topology
   - Files
   - Communicators
   - Datatypes
   - Non-blocking Requests
4. **Future Work**

Future Work

- support threaded programs
- performance analysis
- synchronisation of timestamps