HLRN User Workshop 3-6 Nov 2020

MPI Correctness Checking and Debugging

Klaus-Dieter Oertel

intel®

Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation. Learn more at intel.com or from the OEM or retailer.

Your costs and results may vary.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804. https://software.intel.com/en-us/articles/optimization-notice

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. See backup for configuration details. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

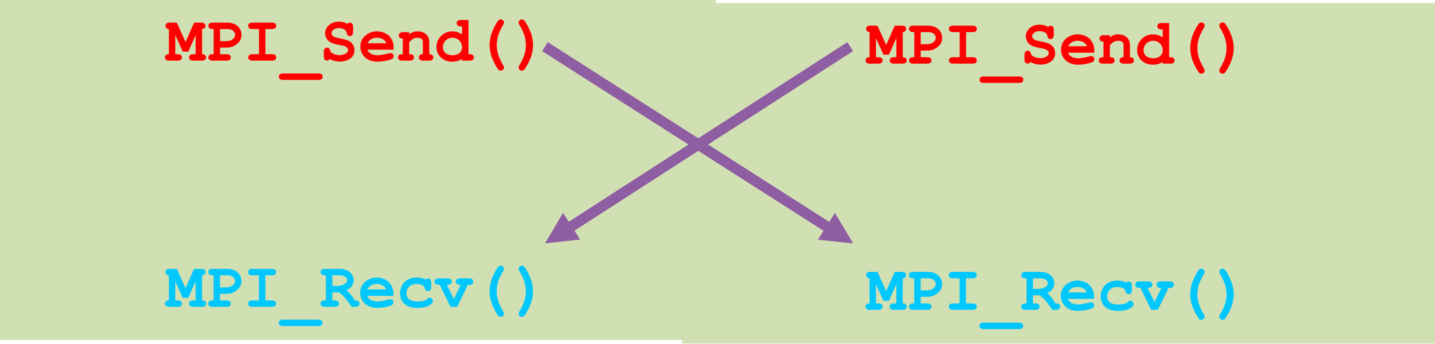
No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Common MPI Problems

Common MPI Problems

Deadlocks because communication relies on buffering in MPI:



```
MPI_Send()        MPI_Send()

MPI_Recv()        MPI_Recv()
```

May or may not work, depending on the message size and MPI implementation!

- Characterized as “unsafe” in the MPI Specification!
- In a “safe” code all MPI_Send calls can be replaced by synchronous MPI_Ssend calls

Common MPI Problems (cont.)

Memory reused in concurrent MPI operations

```
message =...
MPI_Isend(message)
message = ...
MPI_Isend(message)
...
MPI_Wait()
MPI_Wait()
```

Violates the MPI specification!

Can lead to message corruption due to unfinished buffering!

MPI Correctness Checking

MPI Correctness Checking

A technology which validates MPI correctness

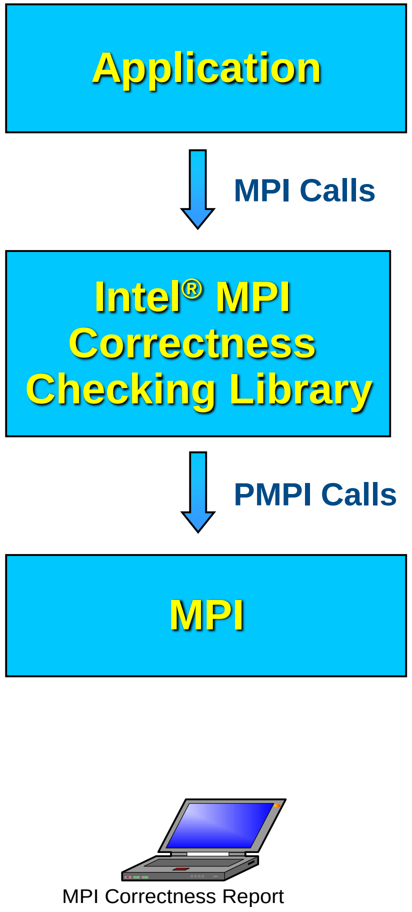
- Detects over 50 distinct MPI programming and run-time errors:
  - ○ Issues with data types, buffers, communicators,
  - ○ point-to-point messages and collective operations,
  - ○ deadlocks, and data corruption.

In-place analysis

- Collects and analyzes MPI event data as the application runs
- Reports errors to console as they are detected
- Shows error location in GUI
- Can trigger debugger breakpoints for in-place analysis

Availability

- Packaged with Intel® Trace Analyzer and Collector (ITAC), for usage with Intel® MPI



MPI Correctness Checking - Goal

Solves two problems:

- Finding programming mistakes in MPI application which need to be fixed by the application developer (e.g. illegal buffer re-usage)
- Detecting errors in the execution environment
  (e.g. by checksum comparison of sent/received message)

Two aspects:

- *Error detection* – done ***automatically*** by the tool
- *Error analysis* – manually by the user based on
  - information provided about an error
  - GUI and knowledge of source code, system, ...

MPI Correctness Checking - Usage

The application runs in the MPI correctness checking mode by running:

```
mpirun -check_mpi ...
```

Or by library pre-loading (mc for "message checking"):

```
mpirun -genv LD_PRELOAD libVTmc.so ...
```

alternative:

```
export LD_PRELOAD=libVTmc.so[:libmpi.so]
mpirun ...
```

or `srun ...` for SLURM

Or by static linkage with **`libVTmc.a`**

Prerequisite: Setup the Intel® Trace Analyzer and Collector environment

At best, the application is compiled with **`-g`** so that errors get a source code reference.

MPI Correctness Checking - Output

Usage with temporary LD_PRELOAD:

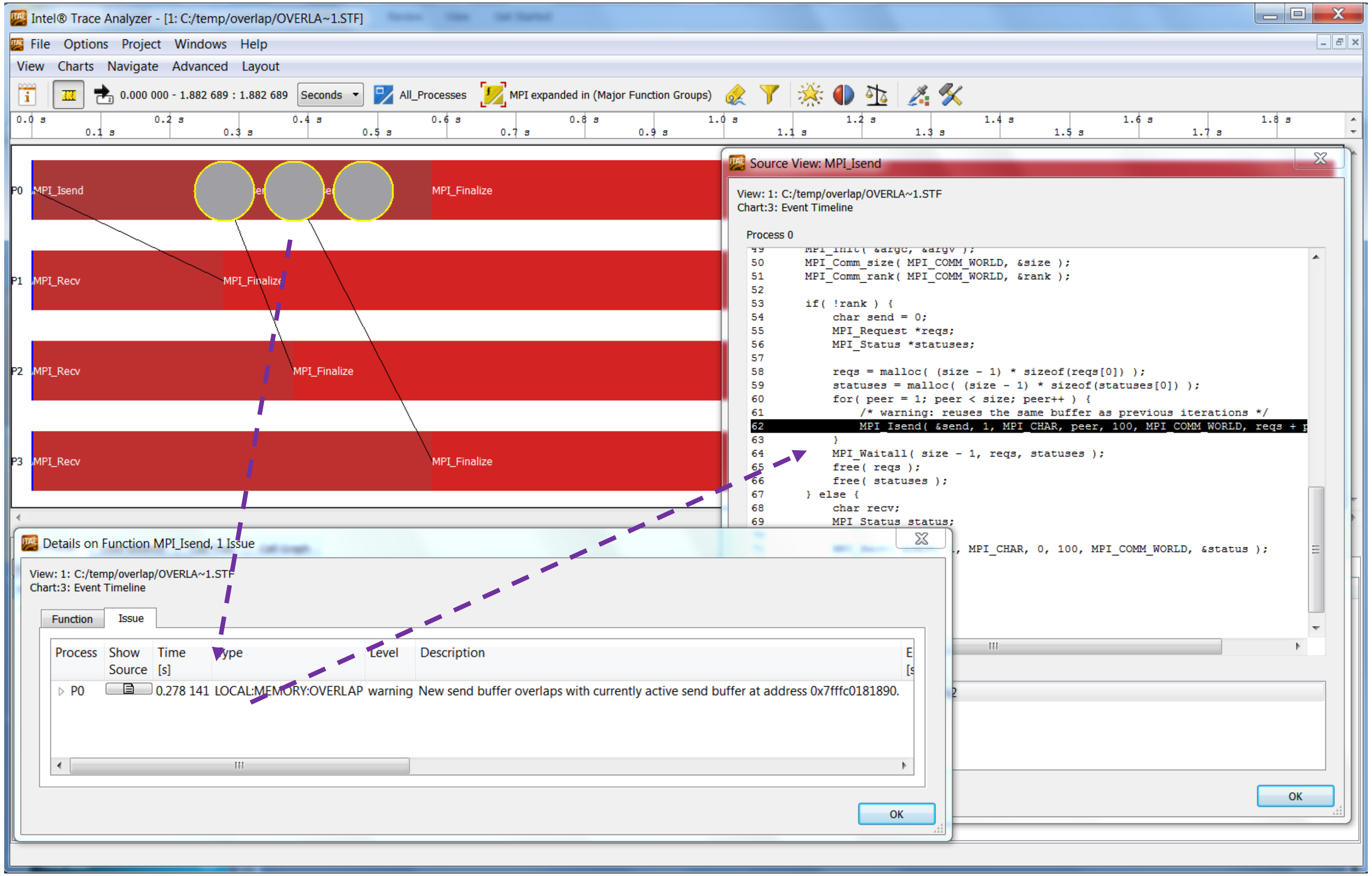
```
$ env LD_PRELOAD=libVTmc.so:libmpi.so srun -n 2 overlap
[...]
[0] WARNING: LOCAL:MEMORY:OVERLAP: warning
[0] WARNING:    New send buffer overlaps with currently active send
buffer at address 0x7fbfffec10.
[0] WARNING:    Control over active buffer was transferred to MPI at:
[0] WARNING:       MPI_Isend(*buf=0x7fbfffec10, count=4,
datatype=MPI_INT, dest=0, tag=103, comm=COMM_SELF [0],
*request=0x508980)
[0] WARNING:       overlap.c:104
[0] WARNING:    Control over new buffer is about to be transferred to
MPI at:
[0] WARNING:       MPI_Isend(*buf=0x7fbfffec10, count=4,
datatype=MPI_INT, dest=0, tag=104, comm=COMM_SELF [0],
*request=0x508984)
[0] WARNING:       overlap.c:105
```

MPI Correctness Checking - Output

Recommended usage is via `–check_mpi`:

```
$ mpirun –check_mpi –n 2 overlap
[...]
[0] WARNING: LOCAL:MEMORY:OVERLAP: warning
[0] WARNING:    New send buffer overlaps with currently active send
buffer at address 0x7fbfffec10.
[0] WARNING:    Control over active buffer was transferred to MPI at:
[0] WARNING:       MPI_Isend(*buf=0x7fbfffec10, count=4,
datatype=MPI_INT, dest=0, tag=103, comm=COMM_SELF [0],
*request=0x508980)
[0] WARNING:       overlap.c:104
[0] WARNING:    Control over new buffer is about to be transferred to
MPI at:
[0] WARNING:       MPI_Isend(*buf=0x7fbfffec10, count=4,
datatype=MPI_INT, dest=0, tag=104, comm=COMM_SELF [0],
*request=0x508984)
[0] WARNING:       overlap.c:105
```

MPI Correctness Checking in the GUI


Intel® Trace Analyzer - [1: C:/temp/overlap/OVERLA~1.STF]
File Options Project Windows Help
View Charts Navigate Advanced Layout
0.000 000 - 1.882 689 : 1.882 689
Seconds
All_Processes
MPI expanded in (Major Function Groups)
0.0 s 0.1 s 0.2 s 0.3 s 0.4 s 0.5 s 0.6 s 0.7 s 0.8 s 0.9 s 1.0 s 1.1 s 1.2 s 1.3 s 1.4 s 1.5 s 1.6 s 1.7 s 1.8 s
P0
MPI_Isend
MPI_Finalize
P1
MPI_Recv
MPI_Finalize
P2
MPI_Recv
MPI_Finalize
P3
MPI_Recv
MPI_Finalize
Source View: MPI_Isend
Details on Function MPI_Isend, 1 Issue
Function
Issue
OK
OK

Source View: MPI_Isend

View: 1: C:/temp/overlap/OVERLA~1.STF
Chart:3: Event Timeline

Process 0

```
    MPI_Init( &argc, &argv );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );

    if( !rank ) {
        char send = 0;
        MPI_Request *reqs;
        MPI_Status *statuses;

        reqs = malloc( (size - 1) * sizeof(reqs[0]) );
        statuses = malloc( (size - 1) * sizeof(statuses[0]) );
        for( peer = 1; peer < size; peer++ ) {
            /* warning: reuses the same buffer as previous iterations */
            MPI_Isend( &send, 1, MPI_CHAR, peer, 100, MPI_COMM_WORLD, reqs + p
        }
        MPI_Waitall( size - 1, reqs, statuses );
        free( reqs );
        free( statuses );
    } else {
        char recv;
        MPI_Status status;

                        , MPI_CHAR, 0, 100, MPI_COMM_WORLD, &status );
```

Details on Function MPI_Isend, 1 Issue

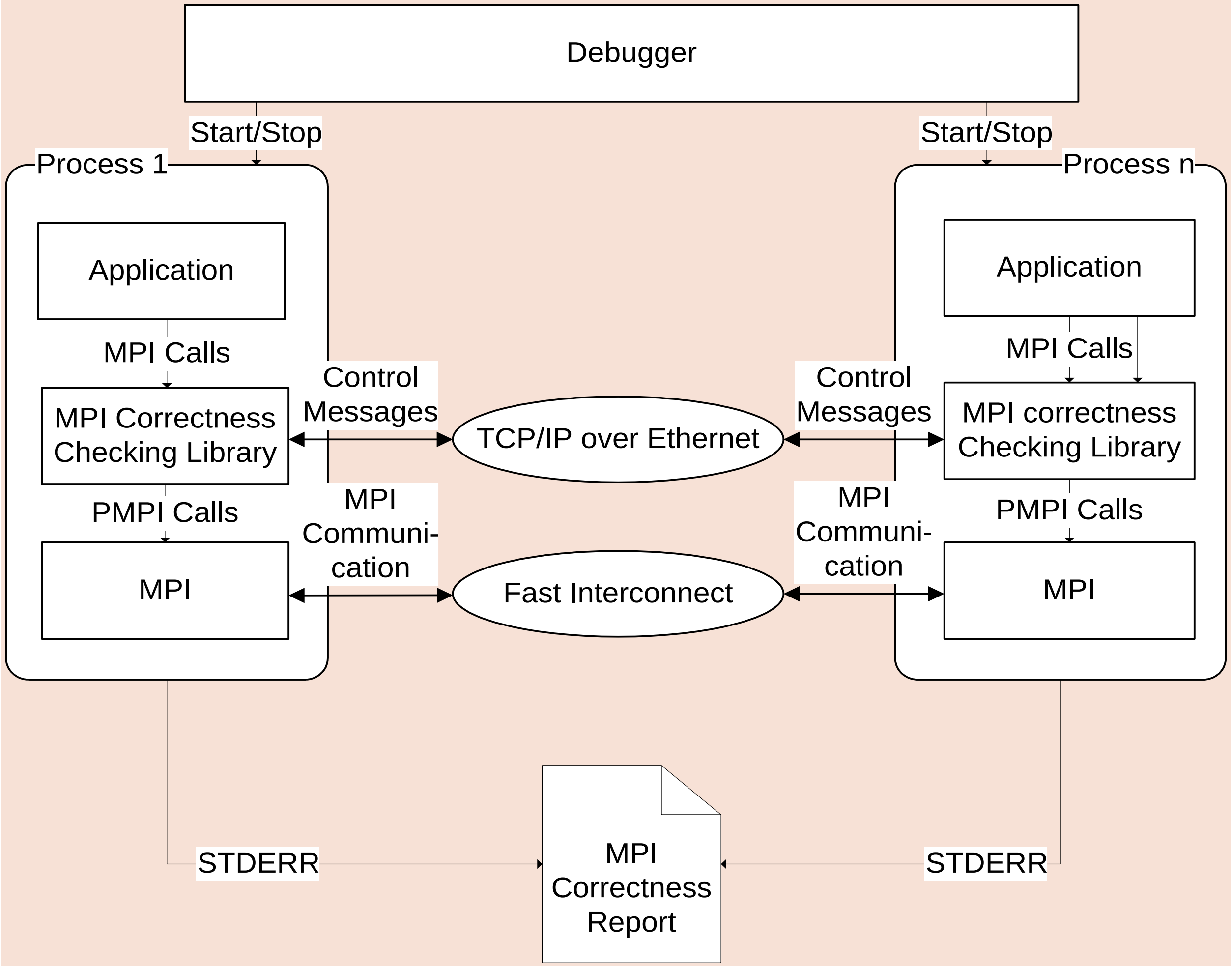
View: 1: C:/temp/overlap/OVERLA~1.STF
Chart:3: Event Timeline

Process	Show Source	Time [s]	Type	Level	Description	E [s
▷ P0	🗎	0.278 141	LOCAL:MEMORY:OVERLAP	warning	New send buffer overlaps with currently active send buffer at address 0x7fffc0181890.	

Set **CHECK-TRACING on** to enable writing of trace file.

MPI Correctness Checking: How it works

- All checks are done at runtime in MPI wrappers.
- Detected problems are reported on stderr immediately in textual format.
- A debugger can be used to investigate the problem at the moment when it is found.



Supported Checks

Two different categories:

- ***Local checks:*** only need information available in the process itself and thus do not require additional communication between processes
- ***Global checks:*** information from other processes is required

Different levels:

- ***Warning:*** application can continue
- ***Error:*** application can continue, but almost certainly not as intended
- ***Fatal error:*** application must be aborted

Some checks may find both warnings and errors, for example:

- Invalid parameter in MPI_Send() => message cannot be sent => **Error**
- Invalid parameter in MPI_Request_free() => resource leak => **Warning**

Local Checks

Unexpected process termination, e.g.

- **EXIT:BEFORE_MPI_FINALIZE**

Buffer handling, e.g.

- **MEMORY:ILLEGAL_MODIFICATION**

Request and data type management, e.g.

- **REQUEST:ILLEGAL_CALL**

Parameter errors found by MPI, e.g. wrong types

NB: Full name qualified with leading ”**LOCAL:**”

Global checks for point-to-point and collective operations

Pending messages

- **MSG:PENDING**

Data type issues

- **MSG/COLLECTIVE:DATATYPE:MISMATCH**

Corrupted data transmission (can detect HW issues)

- **MSG/COLLECTIVE:DATA_TRANSMISSION_CORRUPTED**

For collective operations only

- **COLLECTIVE:ROOT_MISMATCH**

NB: Full name qualified with leading ”**GLOBAL:**”

Deadlock Detection

Very useful is the global check for detection of deadlocks:

- **DEADLOCK:HARD**
- **DEADLOCK:POTENTIAL**
- **DEADLOCK:NO_PROGRESS**

The default time (60s) for deadlock detection can be adjusted:

DEADLOCK-TIMEOUT 10s

Meaning: After 10s with

no progress in the MPI communication

on all processes the application will be interrupted and debug diagnostics shown.

Configuration

Each MPI correctness checking run writes a protocol file <executable>.prot which lists all default or explicit settings

Specify configuration file “-genv VT_CONFIG <configfile>”

- Easily derived from the protocol file <executable>.prot of a previous run

Each check can be turned on and off individually (`CHECK ** ON`), e.g.:

```
CHECK GLOBAL:DEADLOCK:POTENTIAL ON/OFF
```

Number of warnings and errors that are printed are configurable, defaults are:

- Abort immediately at first real error

  ```
  CHECK-MAX-ERRORS 1
  ```

- Keep running regardless how many warnings are issued

  ```
  CHECK-MAX-REPORTS 0
  ```

- Print at most 20 warnings/errors of each type

  ```
  CHECK-SUPPRESSION-LIMIT 20
  ```

Static Linkage

A code linked statically against the ITAC library libVT (by flags –trace or –tcollect for the linking) cannot immediately be run with MPI correctness checking

Reason: Calls to the MPI library are already intercepted by the static library.

Solution: Re-link without ITAC tracing or,
when API functions (like VT_traceon/off) have to be resolved, by:

```
-L${VT_SLIB_DIR} -lVTnull
```

Intel® MPI Debug Output

Intel® MPI Debug Output

Use environment variable I_MPI_DEBUG to print out debugging information

I_MPI_DEBUG	Debug information provided
1	Verbose error diagnostics
2	Confirm which I_MPI_FABRICS was used
3	Effective MPI rank, `pid` and node mapping table
4	Process pinning information
5	Intel® MPI-specific environment variables
6	Show defaults of MPI collectives
>6	Add extra levels of debug information

Intel® MPI Debug Output (contd.)

Add comma separated list of flags to fine tune debug output:

```
I_MPI_DEBUG=<level>,<flags>
```

<flags>	Debug information provided
pid	Process id for each debug message
tid	Thread id for each debug message for the multi-threaded library
host	Host name for each debug message
flock	Synchronize output from different processes or threads
nobuf	Do not use buffered I/O for debug output
…	…

Intel® MPI Debug Output (contd.)

Use environment variable I_MPI_DEBUG_OUTPUT to re-direct the output to a file (or stderr instead of stdout).

- Annotation of the file name with “%r” will produce one output file per rank!
- Use format strings “%p” or “%h” to add pid or host name to the file name

Compiling with „-g“ will print out additional debug information

Hydra process manager and Libfabric

Enable debug output from the Hydra process manager:

- mpirun --verbose
- I_MPI_HYDRA_DEBUG=1|on|enable|yes

For Intel MPI 2019+ use debug functionality of libfabric:

- FI_LOG_LEVEL=Warn, =Trace, =Info, =Debug
- See https://ofiwg.github.io/libfabric/master/man/fabric.7.html
- Simple tool to query for fabric interfaces:
  fi_info [-l]

Summary

The MPI correctness checking from the Intel® Trace Analyzer and Collector (ITAC) detects MPI programming and run time errors

Intel® MPI provides built-in debug output for the MPI communication

Use Libfabric commands and flags for debug output from the provider level intel®