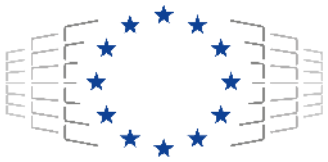
The VEF traces framework: collecting traffic traces from modern, highly-demanding applications

Jesus Escudero-Sahuquillo, Pedro J. García, UCLM, Spain
Francisco J. Andujar, UVA, Spain



EuroHPC
Joint Undertaking

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955776. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Greece, Germany, Spain, Italy, Switzerland.

Choose your own adventure

- VEF Traces will be presented in **three different talks** during the HiPEAC week.
- It depends on the **audience** the contents that will be delivered at each of these talks.
- **Initial questions**:
  - How many of you already know something about the VEF traces?
  - How many of you have attended the tutorials about VEF traces organized by the 3-SEA family?

Agenda

- The VEF Traces Framework
- VEF Traces file format
- Generating VEF Traces
- Static analysis
- Using TraceLib in 3rd party simulation tools
- Open discussion

Agenda

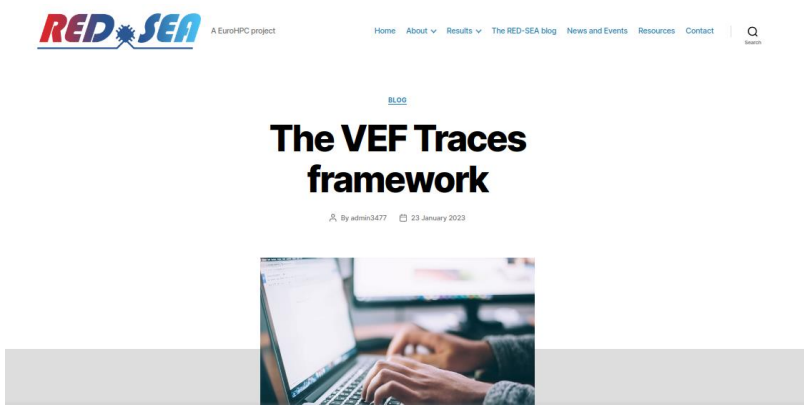
- **The VEF Traces Framework**
- VEF Traces file format
- Generating VEF Traces
- Static analysis
- Using TraceLib in 3rd party simulation tools
- Open discussion

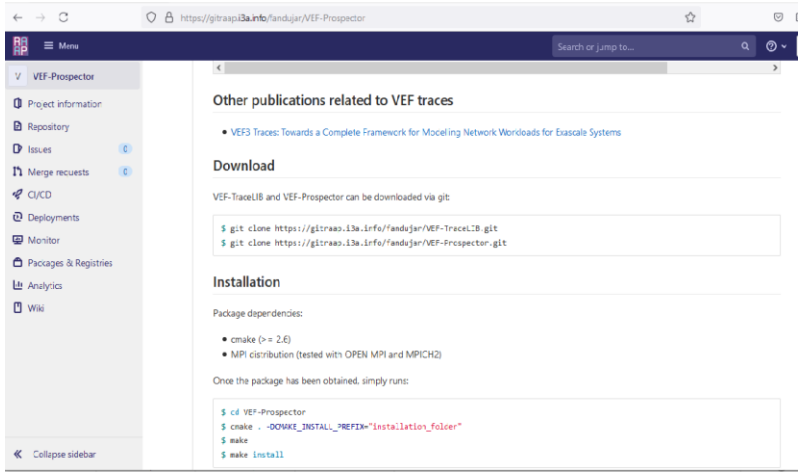
What is the VEF Traces framework?

- An **open-source framework to generate traces from MPI workloads**.
- An **off-line simulation environment** to supply workloads to interconnection network simulators
- A framework **independent of the simulation platform**:
  - Currently integrated in multiple interconnection network simulators
- It provides a **self-related trace model**, which assumes that:
  - Communications do not depend on absolute timestamps (although we obtain them for statistical purposes).
  - It is highly probable that communications generating packets in the network depend on received packets generated by previous communications

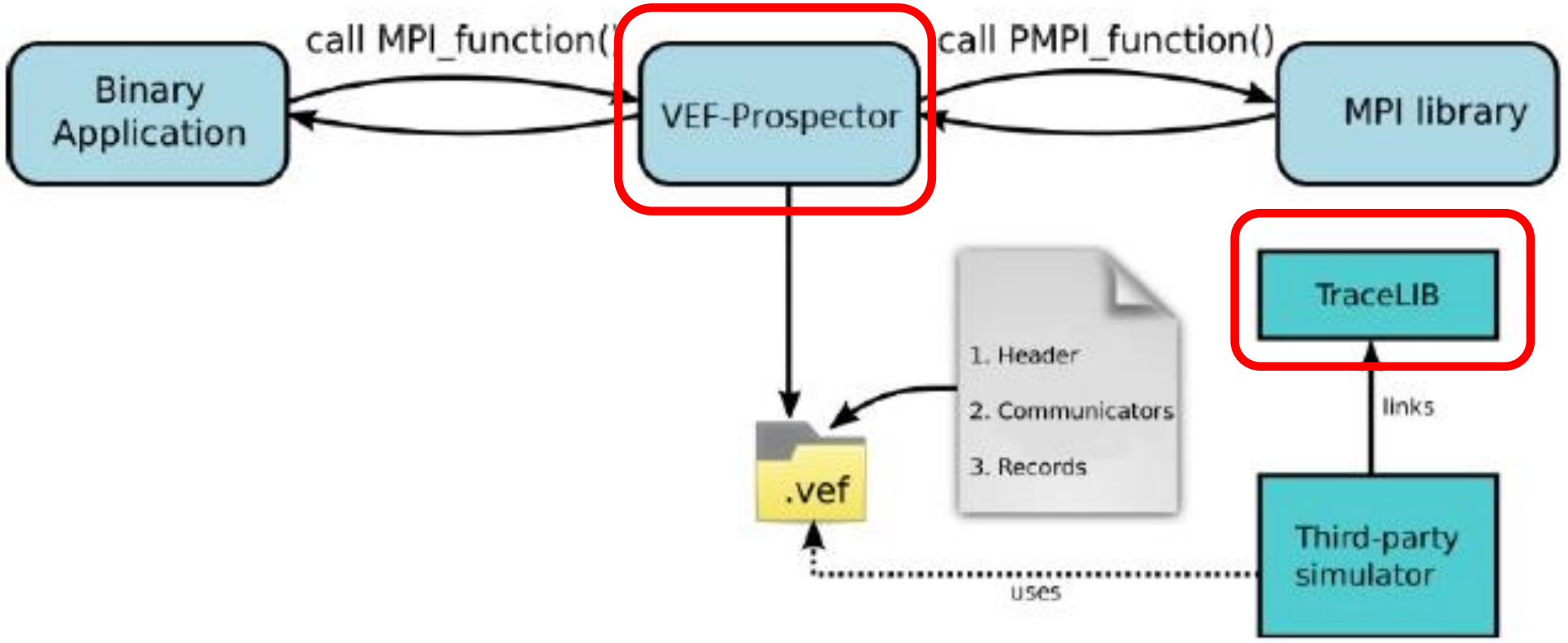
What is the VEF Traces framework?

- The VEF Trace framework comprises two open-source C libraries:
  - **VEF Prospector**: captures the trace directly from the MPI application using the MPI standard profiling interface (PMPI)
  - **VEF TraceLIB**: feeds the network simulator using VEF traces
  - VEF Traces framework tools are available through our webpage and GIT repositories:
    - https://redsea-project.eu/the-vef-traces-framework/
    - https://gitraap.i3a.info/fandujar/VEF-TraceLIB
    - https://gitraap.i3a.info/fandujar/VEF-Prospector
- VEF Traces public repository gathering traces from RED-SEA and DEEP-SEA projects:
  - https://gitraap.i3a.info/jesus.escudero/vef-traces-repository





VEF traces framework overview



What is a self-related trace?

Let’s consider tasks A and B doing a “ping-pong”:

```
1: // TaskA
2: while true do
3:    SEND(MessageA, TaskB)
4:     // Computation
5:     RECV(MessageB, TaskB)
6:     // Computation
7: end while
```

```
1: // TaskB
2: while true do
3:    RECV(MessageA, TaskA)
4:     // Computation
5:     SEND(MessageB, TaskA)
6:     // Computation
7: end while
```

The problem of absolute timestamps

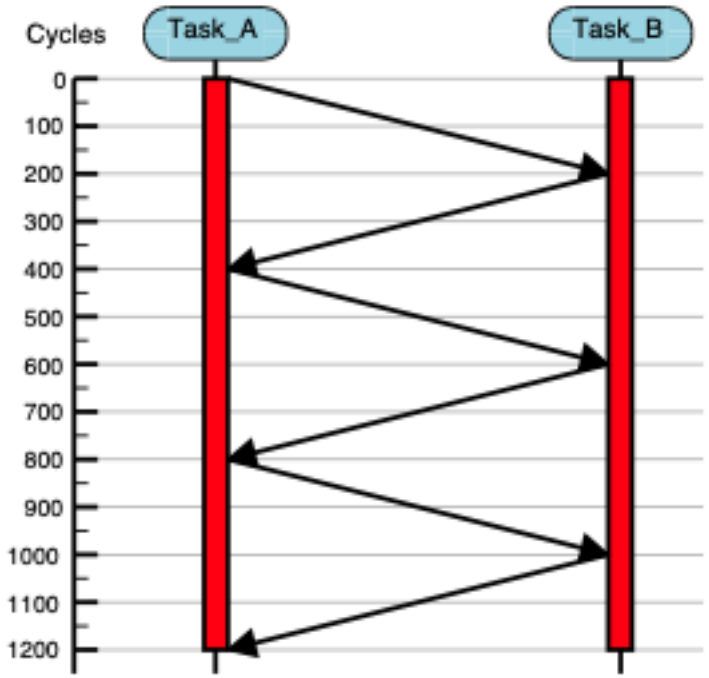
Considering absolute timestamps and ignoring the message size, a possible trace is:

SOURCE	DESTINATION	TIMESTAMP
A	B	0
B	A	200
A	B	400
B	A	600
A	B	800
B	A	1000

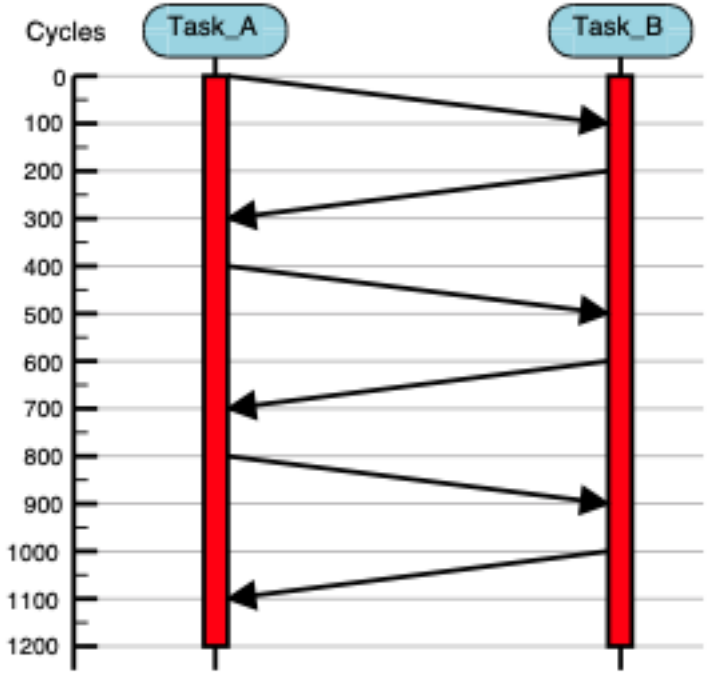
...

The problem of absolute timestamps

The execution time only depends on the last message in the trace!!!



200-cycle message delivery

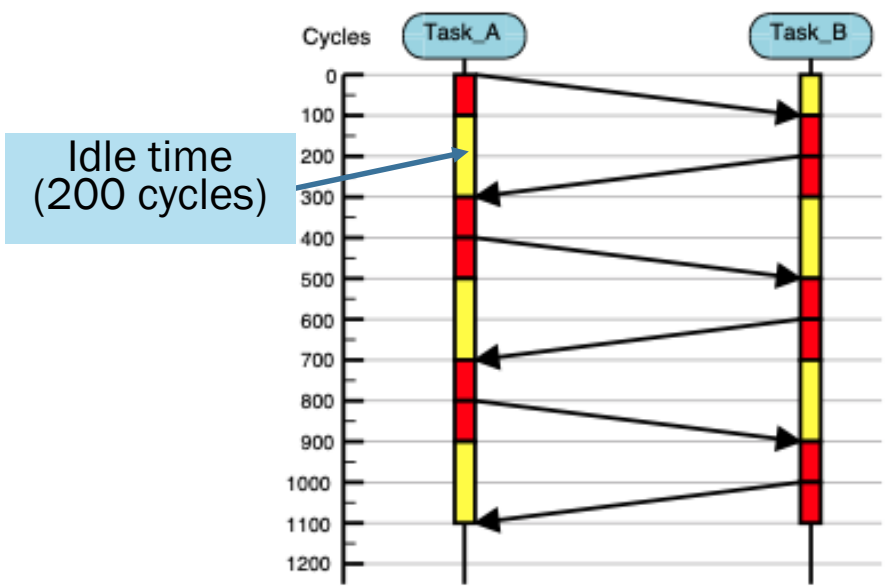


100-cycle message delivery

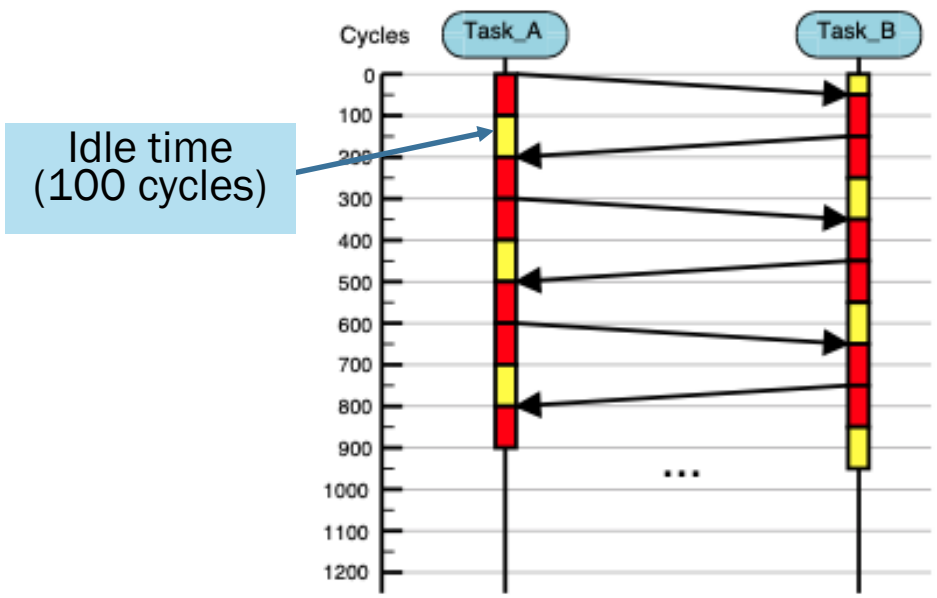
Self-related trace

Self-related record: Its execution depends on the execution of a previous record

The network latency and throughput impacts on the execution time!!!



100-cycle message delivery



50-cycle message delivery

Agenda

- The VEF Traces Framework
- **VEF Traces file format**
- Generating VEF Traces
- Static analysis
- Using TraceLib in 3rd party simulation tools
- Open discussion

VEF traces file format

Type of records:

```
VEF3  4  5  2  3  10  0 1000
C0   0  1  2  3
C1  2  3
G0  C0  0  0  4  0  0  0  -1
G0  C0  0  1  0  4  0  0  -1
G0  C0  0  2  0  4  0  0  -1
G0  C0  0  3  0  4  0  0  -1
0  0  1  16  3  300  G0
1  1  0  32  3  250  G0
2  1  2  128  1  100  1
3  2  2  0  3  200  G0
4  2  1  4  2  100  2
G1  C1  0  2  4  0  1  250  4
G1  C1  0  3  0  4  3  800  G0
G2  C0  1  0  0  4  2  300  1
G2  C0  1  1  4  0  1  350  2
G2  C0  1  2  4  0  3  100  G1
G2  C0  1  3  4  0  3  100 G1
```

VEF traces file format

```
VEF3  4  5  2  3  10  0 1000
C0   0  1  2  3
C1  2  3
G0  C0  0  0  4  0  0  0  -1
G0  C0  0  1  0  4  0  0  -1
G0  C0  0  2  0  4  0  0  -1
G0  C0  0  3  0  4  0  0  -1
0  0  1  16  3  300  G0
1  1  0  32  3  250  G0
2  1  2  128  1  100  1
3  2  2  0  3  200  G0
4  2  1  4  2  100  2
G1  C1  0  2  4  0  1  250  4
G1  C1  0  3  0  4  3  800  G0
G2  C0  1  0  0  4  2  300  1
G2  C0  1  1  4  0  1  350  2
G2  C0  1  2  4  0  3  100  G1
G2  C0  1  3  4  0  3  100 G1
```

Type of records:

1. Trace header *(nTasks nMsgs nCOMM nCollComm nLocalCollComm noRecvDep)*

VEF traces file format

```
VEF3  4  5  2  3  10  0 1000
C0   0  1  2  3
C1  2  3
G0  C0  0  0  4  0  0  0  -1
G0  C0  0  1  0  4  0  0  -1
G0  C0  0  2  0  4  0  0  -1
G0  C0  0  3  0  4  0  0  -1
0  0  1  16  3  300  G0
1  1  0  32  3  250  G0
2  1  2  128  1  100  1
3  2  2  0  3  200  G0
4  2  1  4  2  100  2
G1  C1  0  2  4  0  1  250  4
G1  C1  0  3  0  4  3  800  G0
G2  C0  1  0  0  4  2  300  1
G2  C0  1  1  4  0  1  350  2
G2  C0  1  2  4  0  3  100  G1
G2  C0  1  3  4  0  3  100 G1
```

Type of records:

1. Trace header *(nTasks nMsgs nCOMM nCollComm nLocalCollComm noRecvDep)*
2. Communicators (or COMMs)

VEF traces file format

```
VEF3  4  5  2  3  10  0 1000
C0   0  1  2  3
C1  2  3
G0  C0  0  0  4  0  0  0  -1
G0  C0  0  1  0  4  0  0  -1
G0  C0  0  2  0  4  0  0  -1
G0  C0  0  3  0  4  0  0  -1
0  0  1  16  3  300  G0
1  1  0  32  3  250  G0
2  1  2  128  1  100  1
3  2  2  0  3  200  G0
4  2  1  4  2  100  2
G1  C1  0  2  4  0  1  250  4
G1  C1  0  3  0  4  3  800  G0
G2  C0  1  0  0  4  2  300  1
G2  C0  1  1  4  0  1  350  2
G2  C0  1  2  4  0  3  100  G1
G2  C0  1  3  4  0  3  100 G1
```

Type of records:

1. Trace header *(nTasks nMsgs nCOMM nCollComm nLocalCollComm noRecvDep)*
2. Communicators (or COMMs)
3. Point-to-point message records

VEF traces file format

```
VEF3  4  5  2  3  10  0 1000
C0   0  1  2  3
C1  2  3
G0  C0  0  0  4  0  0  0  -1
G0  C0  0  1  0  4  0  0  -1
G0  C0  0  2  0  4  0  0  -1
G0  C0  0  3  0  4  0  0  -1
0  0  1  16  3  300  G0
1  1  0  32  3  250  G0
2  1  2  128  1  100  1
3  2  2  0  3  200  G0
4  2  1  4  2  100  2
G1  C1  0  2  4  0  1  250  4
G1  C1  0  3  0  4  3  800  G0
G2  C0  1  0  0  4  2  300  1
G2  C0  1  1  4  0  1  350  2
G2  C0  1  2  4  0  3  100  G1
G2  C0  1  3  4  0  3  100 G1
```

Type of records:

1. Trace header *(nTasks nMsgs nCOMM nCollComm nLocalCollComm noRecvDep)*
2. Communicators (or COMMs)
3. Point-to-point message records
4. Collective communication records

VEF traces file format

```
VEF3   4   5   2   3   10   0 1000
C0    0   1   2   3
C1   2   3
G0   C0   0   0   4   0   0   0   -1
G0   C0   0   1   0   4   0   0   -1
G0   C0   0   2   0   4   0   0   -1
G0   C0   0   3   0   4   0   0   -1
0   0   1   16   3   300   G0
1   1   0   32   3   250   G0
2   1   2   128   1   100   1
3   2   2   0   3   200   G0
4   2   1   4   2   100   2
G1   C1   0   2   4   0   1   250   4
G1   C1   0   3   0   4   3   800   G0
G2   C0   1   0   0   4   2   300   1
G2   C0   1   1   4   0   1   350   2
G2   C0   1   2   4   0   3   100   G1
G2   C0   1   3   4   0   3   100 G1
```

Type of records:

1. Trace header *(nTasks nMsgs nCOMM nCollComm nLocalCollComm noRecvDep)*
2. Communicators (or COMMs)
3. Point-to-point message records
4. Collective communication records

The last three fields of every record indicate the relationship between records:

VEF traces file format

```
VEF3   4   5   2   3   10   0 1000
C0    0   1   2   3
C1   2   3
G0   C0   0   0   4   0   0   0   -1
G0   C0   0   1   0   4   0   0   -1
G0   C0   0   2   0   4   0   0   -1
G0   C0   0   3   0   4   0   0   -1
0   0   1   16   3   300   G0
1   1   0   32   3   250   G0
2   1   2   128   1   100   1
3   2   2   0   3   200   G0
4   2   1   4   2   100   2
G1   C1   0   2   4   0   1   250   4
G1   C1   0   3   0   4   3   800   G0
G2   C0   1   0   0   4   2   300   1
G2   C0   1   1   4   0   1   350   2
G2   C0   1   2   4   0   3   100   G1
G2   C0   1   3   4   0   3   100 G1
```

Type of records:

1. Trace header
2. Communicators (or COMMs)
3. Point-to-point message records
4. Collective communication records

The last three fields of every record indicate the relationship between records:

- Dependency type (0:independent, 1:send, 2:receive or 3:collective comm)

VEF traces file format

```
VEF3   4   5   2   3   10   0 1000
C0    0   1   2   3
C1   2   3
G0   C0   0   0   4   0   0   0   -1
G0   C0   0   1   0   4   0   0   -1
G0   C0   0   2   0   4   0   0   -1
G0   C0   0   3   0   4   0   0   -1
0   0   1   16   3   300   G0
1   1   0   32   3   250   G0
2   1   2   128   1   100   1
3   2   2   0   3   200   G0
4   2   1   4   2   100   2
G1   C1   0   2   4   0   1   250   4
G1   C1   0   3   0   4   3   800   G0
G2   C0   1   0   0   4   2   300   1
G2   C0   1   1   4   0   1   350   2
G2   C0   1   2   4   0   3   100   G1
G2   C0   1   3   4   0   3   100 G1
```

Type of records:

1. Trace header
2. Communicators (or COMMs)
3. Point-to-point message records
4. Collective communication records

The last three fields of every record indicate the relationship between records:

- Dependency type (0:independent, 1:send, 2:receive or 3:collective comm)
- Dependency time for record execution

VEF traces file format

```
VEF3   4   5   2   3   10   0 1000
C0    0   1   2   3
C1   2   3
G0   C0   0   0   4   0   0   0   -1
G0   C0   0   1   0   4   0   0   -1
G0   C0   0   2   0   4   0   0   -1
G0   C0   0   3   0   4   0   0   -1
0   0   1   16   3   300   G0
1   1   0   32   3   250   G0
2   1   2   128   1   100   1
3   2   2   0   3   200   G0
4   2   1   4   2   100   2
G1   C1   0   2   4   0   1   250   4
G1   C1   0   3   0   4   3   800   G0
G2   C0   1   0   0   4   2   300   1
G2   C0   1   1   4   0   1   350   2
G2   C0   1   2   4   0   3   100   G1
G2   C0   1   3   4   0   3   100 G1
```

Type of records:

1. Trace header
2. Communicators (or COMMs)
3. Point-to-point message records
4. Collective communication records

The last three fields of every record indicate the relationship between records:

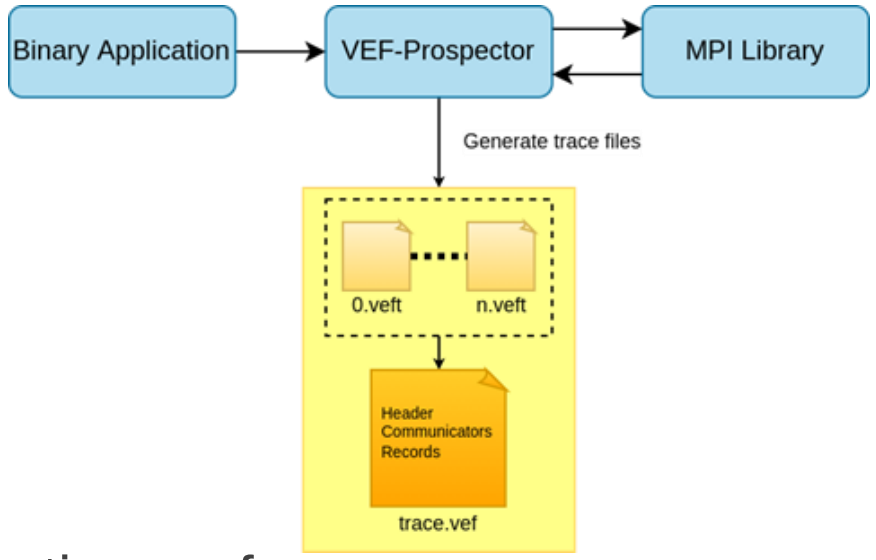
- Dependency type (0:independent, 1:send, 2:receive or 3:collective comm)
- Dependency time for record execution
- Dependency ID: Identifier of the previous record which the current record is waiting for.

Agenda

- The VEF Traces Framework
- VEF Traces file format
- **Generating VEF Traces**
- Static analysis
- Using TraceLib in 3rd party simulation tools
- Open discussion

Generating VEF Traces

- **VEF Prospector**:
  - Open-source tool that profiles an MPI application and generates VEF-traces files.
  - GNU General Public License.



- Designed to minimize the impact on application performance:
  - First stage: The MPI communications are captured (using MMAP, /tmp folder, etc.)
  - Second stage: Once the application is completed, the VEF trace is generated

Generating VEF Traces

- **Download**:
  - `$ git clone https://gitraap.i3a.info/fandujar/VEF-Prospector.git`
- **Package dependencies**:
  - cmake (>= 2.6)
  - MPI distribution (tested with OPEN MPI and MPICH2)
- **Installation** (after the repository has been cloned):

```
$ cd VEF-Prospector
$ cmake . -DCMAKE_INSTALL_PREFIX="installation_folder"
$ make
$ make install
```

- For the installation folder, we recommend `$HOME/opt`
- If the installation folder is not set using `-DCMAKE_INSTALL_PREFIX`, the tools are installed by default in `$HOME/local/vef_prospector`

Generating VEF Traces

Executable files in the VEF-Prospector package

- **vef_mixer**: mixes the temporary files generated by the instrumentation library to obtain the VEF traces.
- **vef_tmp_reader**: allows to read in "human" format the temporary files.
- **vef2_updater**: Updates VEF traces in format VEF2 to VEF traces in format VEF3
- **prospector_debugger**: used for development purposes, it is not required to obtain VEF traces.
- **vmpirun**: wraps the mpirun command and loads the instrumentation library.
- **vfmpirun**: wraps the mpirun command and loads the "full-version" of the instrumentation library.
- **repasaVEF**: reads a VEF trace and characterizes its traffic in several data files

Generating VEF Traces

Libraries

- **libvefprospector.so** (MPI Instrumentation library). It captures the MPI calls supported by the VEF traces. When a non-supported call is captured, the process to capture the MPI calls may be aborted, depending on the environment variable `VEFP_IGNORE_NON_MODELLED_CALLS`.
  - If this variable is not set or set to zero, when a non supported call is captured the MPI application is aborted, notifying also the unsupported call that causes the failure.
  - If this variable set to an integer greater than zero, the unsupported calls are ignored. However, it is possible that the temporary files could not be merged in a VEF trace. See "Modelled MPI calls" for detailed information.
- **libvefprospector_full.so**: "full" MPI Instrumentation library. This library captures the most important MPI calls, although these calls are not supported by the VEF traces (e.g. *MPI_Waitany()* or *MPI_ReduceScatter()*, or all the non-blocking collective communications of MPI 3). It's useful to get more information in the temporary files, but there is no advantages in use the full version to generate the VEF traces.

Generating VEF Traces

- To obtain VEF traces, replace the `mpirun` command by the `vmpirun` command. For example:

```
$ vmpirun -np 16 ./my_mpi_app
```

- A new folder will be generated using the application name and a timestamp. Following the previous example, the library will generate a folder called:

```
my_mpi_app-AAAA-MM-DD-HH-MM-SS
```

- Once the MPI application ends, the trace folder will be contain multiple `.veft` and `.comm` files (one *.veft* file and one .comm file per MPI task).
  - The veft files contain the mpi calls of the task,
  - while the *comm* files contain all the MPI communicators used by this task.
- The information of these files can be read using the executable **vef_tmp_reader**.

Generating VEF Traces

- Another file called `VEFT.main` will be used to generate the VEF traces, using the **vef_mixer** executable:

```
$ vef_mixer -i VEFT.main -o output_trace.vef
```

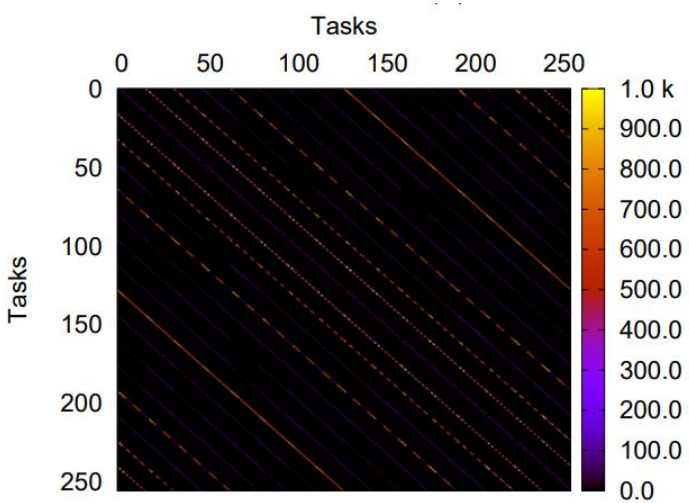
- This finishes the process to generate a VEF trace.
- VEF Traces can be tested using the tracetor tool, provided with the TraceLIB library.

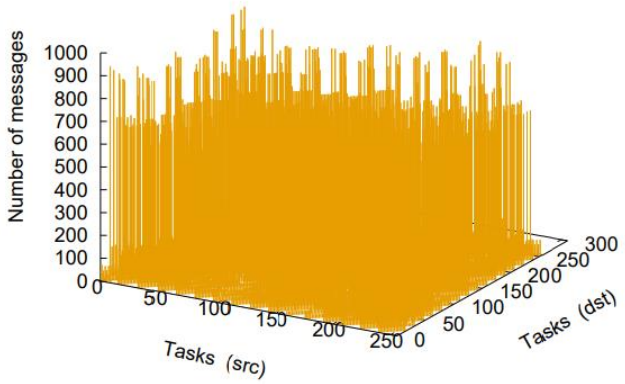
Agenda

- The VEF Traces Framework
- VEF Trace file format
- Generating VEF Traces
- **Static analysis**
- Using TraceLib in 3rd party simulation tools
- Open discussion

Static analysis of VEF Traces

- VEF-Prospector also offers an off-line analyzer for the VEF traces
  - Point-to-point and collective communications
  - Generates reports in plain text, graphs using GnuPlot and PDF that integrates all graphs using LaTex





Static analysis of VEF Traces

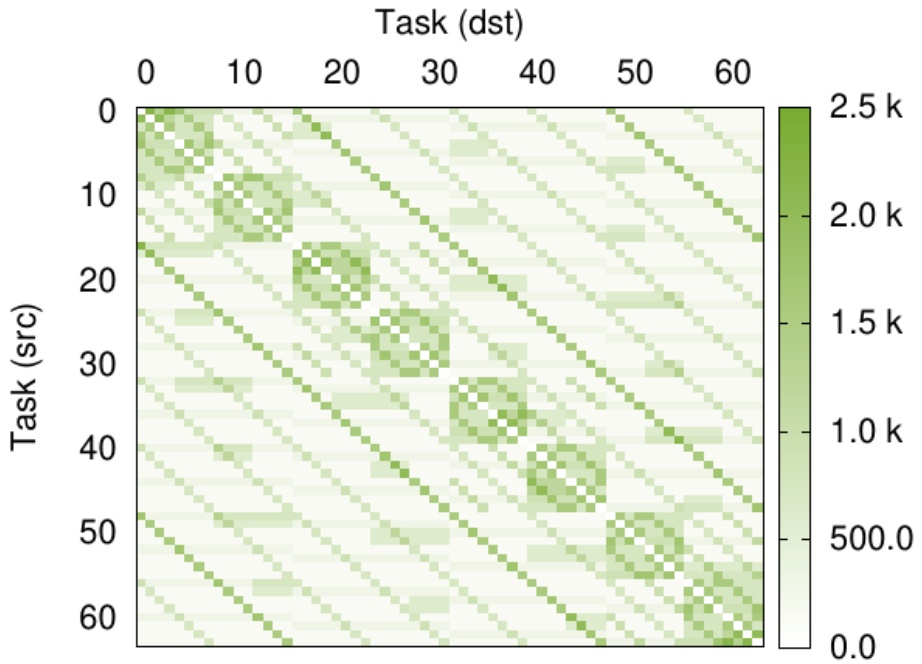
- We need to add all the traces into the same folder
- Then we run the following command:

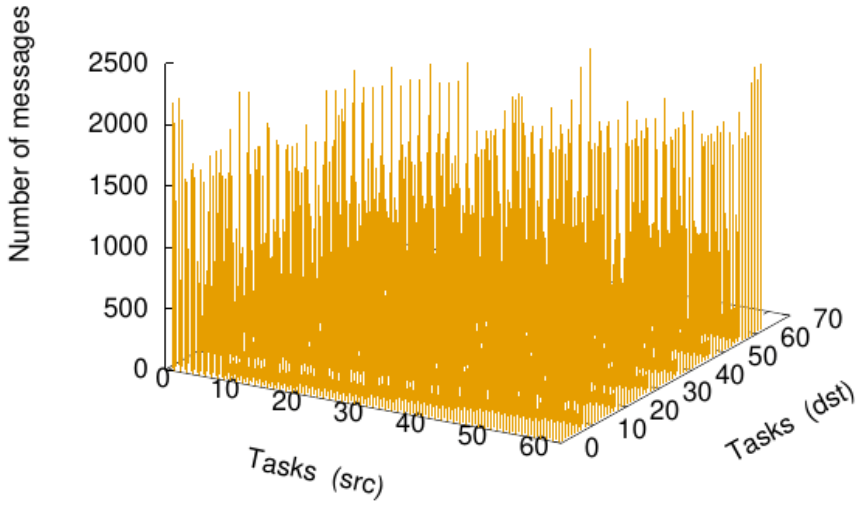
```
$ offline-vef-analysis.sh "path/to/vef" "file".vef
```

- Then, all the resulting files are stored in the $HOME/VEF-Traces folder

Example: LAMMPS (64 tasks)

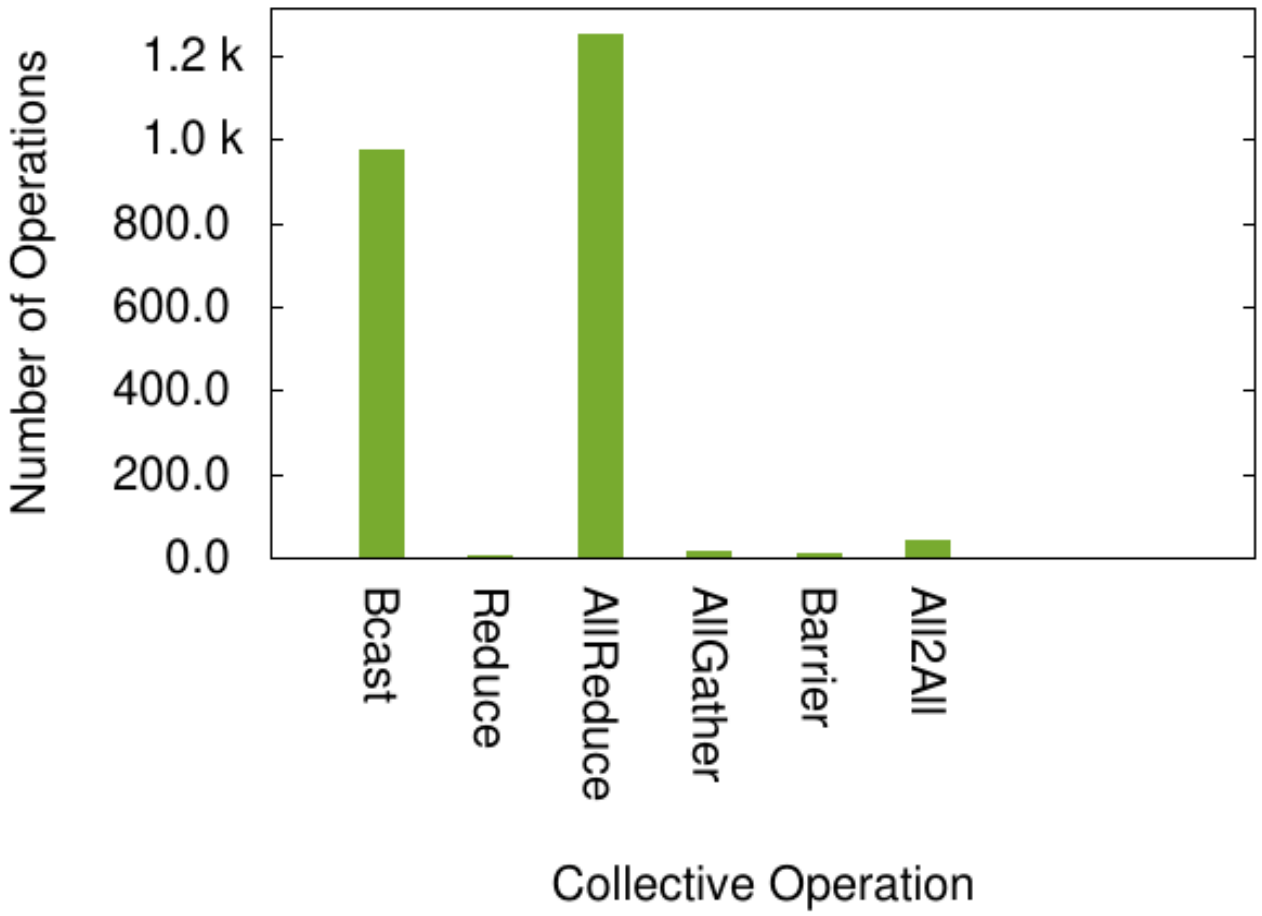
Point-to-point call matrix





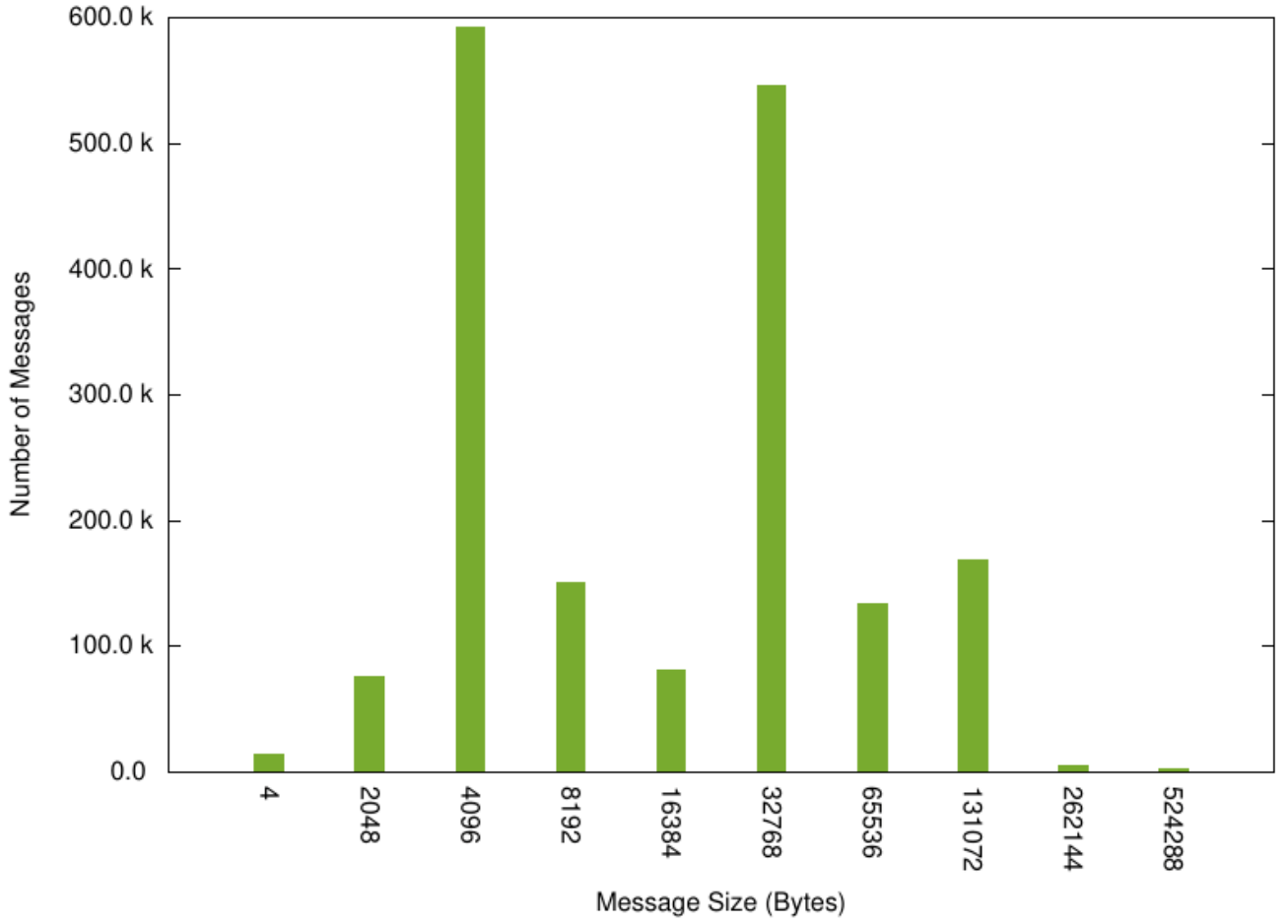
Example: LAMMPS (64 tasks)

Collective communication distribution by type



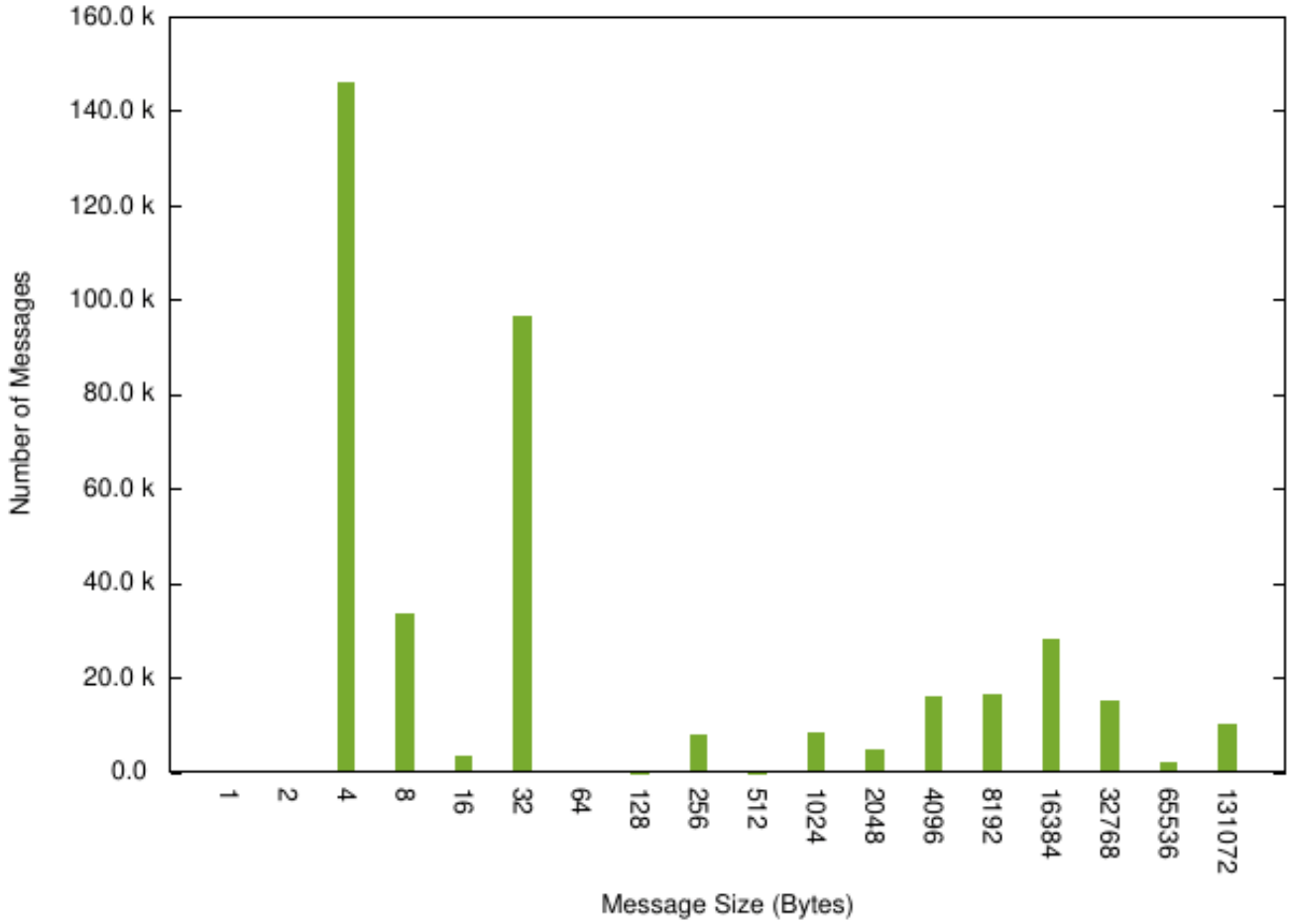
Example: LAMMPS (64 tasks)

Message size distribution of point-to-point messages



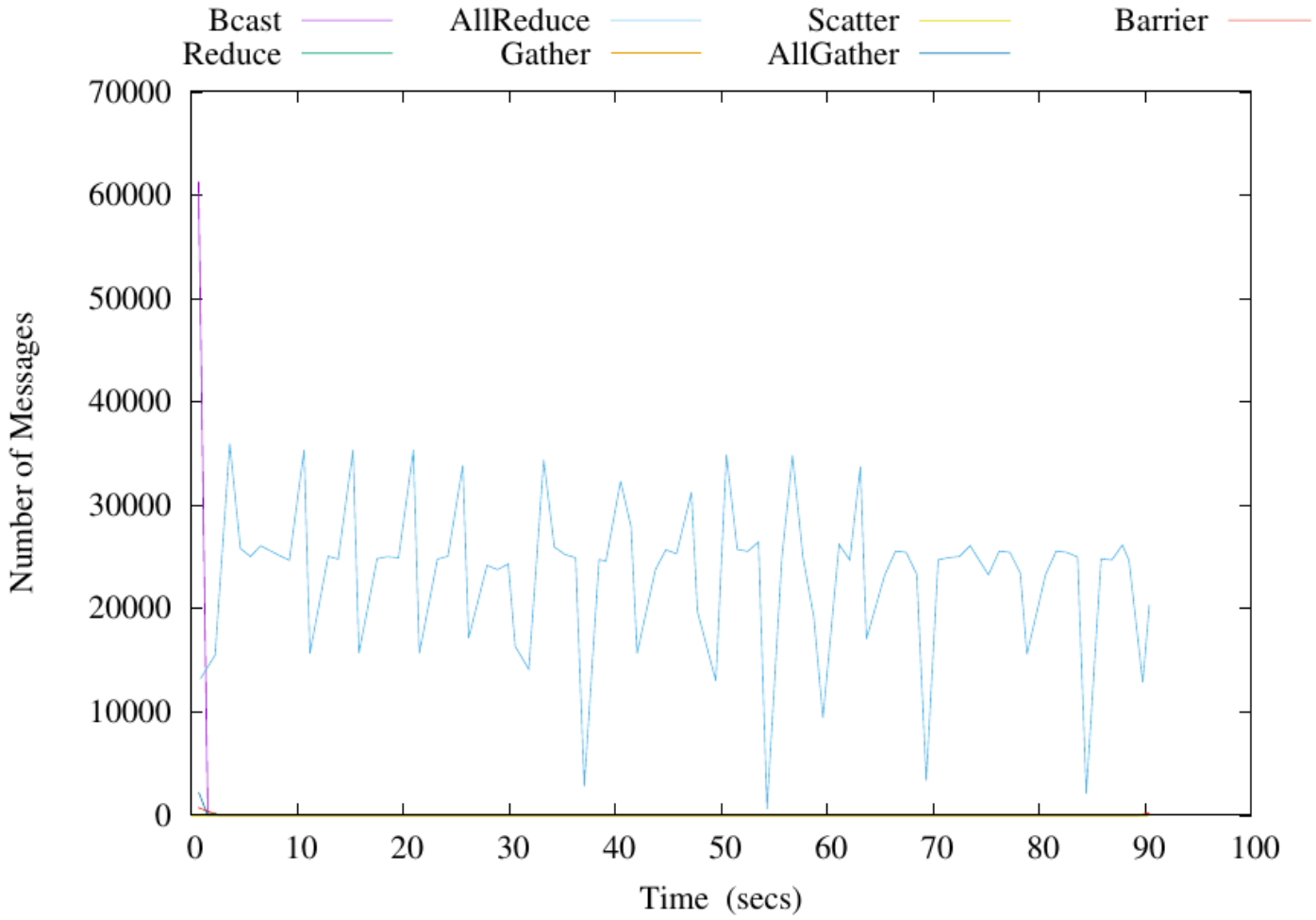
Example: LAMMPS (64 tasks)

Message size distribution of collective communication messages



Example: LAMMPS (64 tasks)

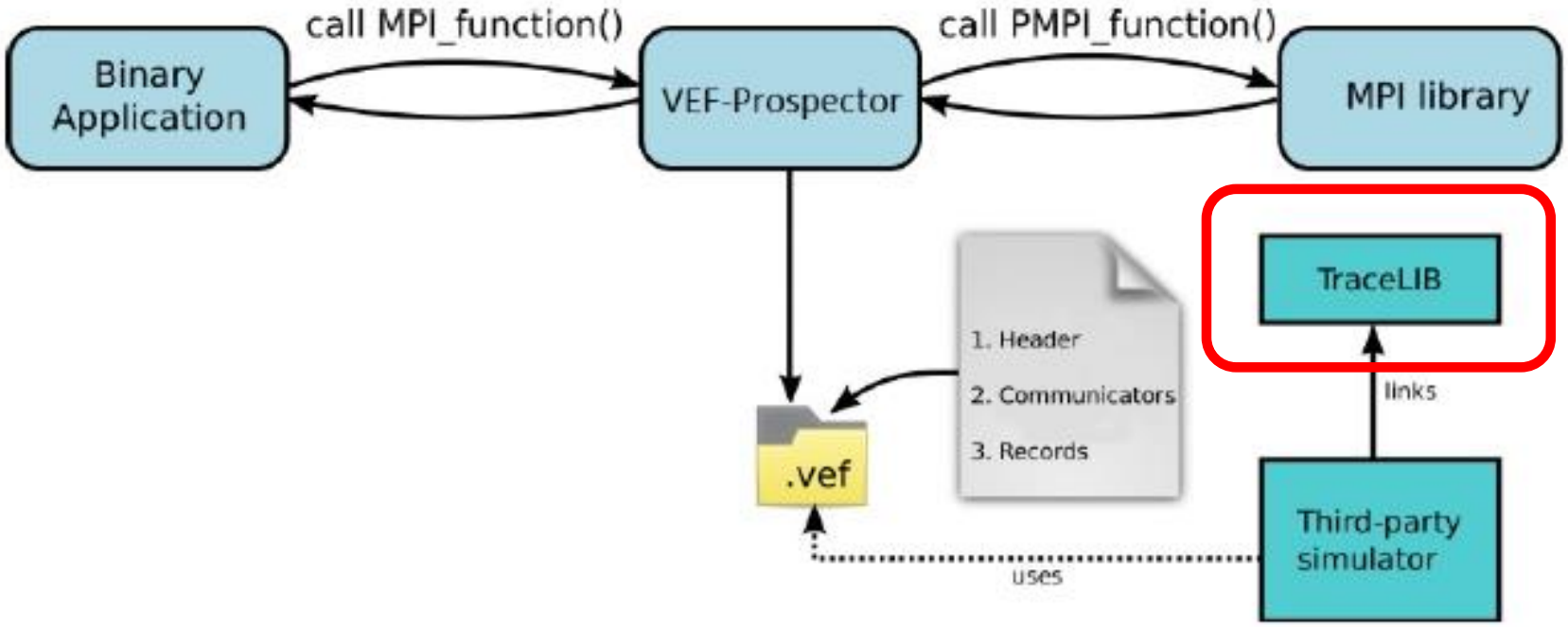
Number of collective communication messages over time (ideal network)



Agenda

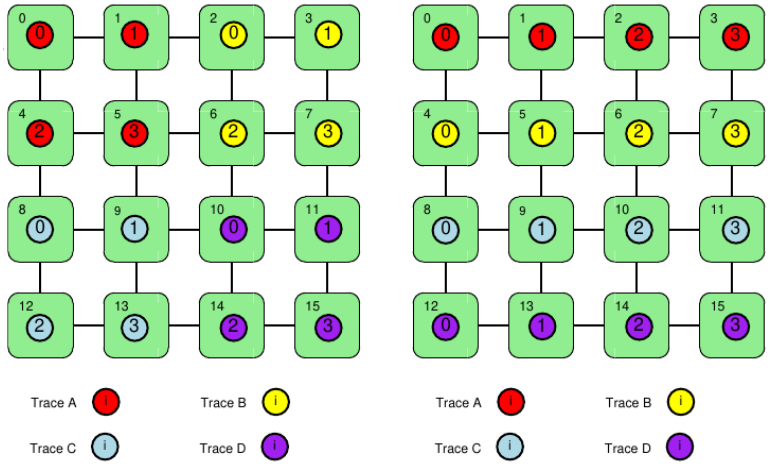
- The VEF Traces Framework
- VEF Trace file format
- Generating VEF Traces
- Static analysis
- **Using TraceLib in 3rd party simulation tools**
- Open discussion

VEF traces framework overview



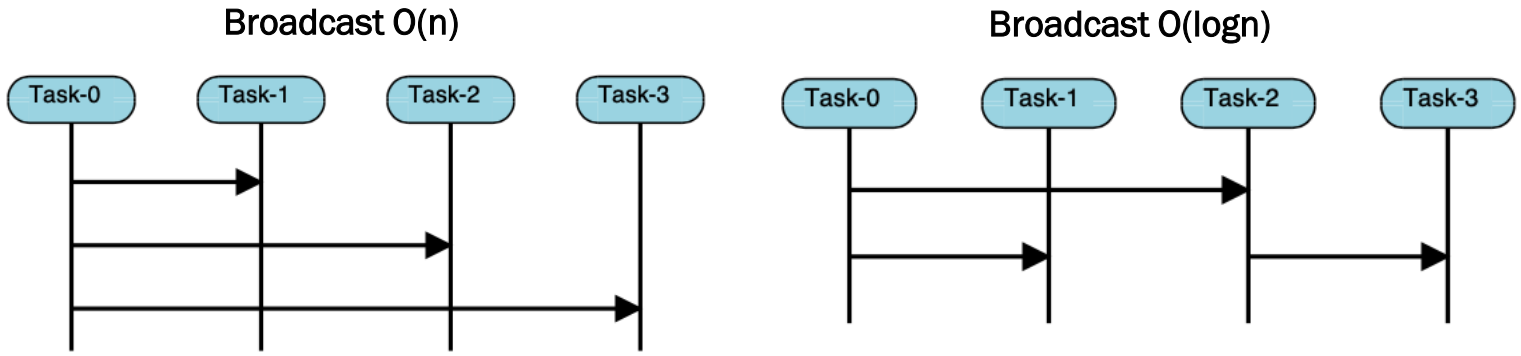
Features of VEF TraceLIB

- It **reads the VEF traces** and communicates with the network simulator.
- Simulation of **simultaneous traces**.
- **Flexible mapping** scheme of tasks to NICs.



Features of VEF TraceLIB

- It **reads the VEF traces** and communicates with the network simulator.
- Simulation of **simultaneous traces**.
- **Flexible mapping** scheme of tasks to NICs.
- **Models the message exchange** in collective communications:
  - The collective communication records only models the MPI collective comm.
  - Implementations based on the OpenMPI algorithms.
  - Possibility to implement their own collective communication functions.



Installing VEF TraceLIB

- **Download**:

```
$ git clone https://gitraap.i3a.info/fandujar/VEF-TraceLIB.git
```

- **Installation**:
  - Packet dependencies: cmake (>= 2.6)
  - Once the package has been cloned:

```
$ cd VEF-TraceLIB
$ cmake . -DCMAKE_INSTALL_PREFIX="installation_folder"
$ make
$ make install
```

- If the installation folder is not set using `-DCMAKE_INSTALL_PREFIX`, the tools are installed in `$HOME/local/vef_tracelib`

Using VEF TraceLIB in your simulator

- **Declaration and initialization of the VEF traces data structures:**

```
int main(int argc, char * const argv[])
{
    /* Simulator clock                                          */
    int sim_clock=0;
    /* Configure the simulator                                  */
    sim_config();
    /* TraceLIB configuration struct                            */
    conf_t my_conf;
    my_conf.simNodes = sim_num_nodes;
    my_conf.number_of_traces = 1;
    /* Set the fields of my_conf and initialize TraceLIB        */
    init_MultipleTraceManager(&my_conf);
    /* ...                                                      */
}
```

Using VEF TraceLIB in your simulator

- **Starting simulation and sending messages to the simulator:**

```
/* Start the simulation.  The simulation finishes when the
   TraceLIB completes the trace execution                    */
while (!isTraceComplete())
{
    /* TraceLIB Message struct                               */
    msg_t msg;
    /* Does TraceLIB have messages to inject at current cycle?
       */
    while (areMsgstoGet())
    {
        /* Get the message and program the generation event  */
        getMsgTrace(&msg);
        putEvent(GENERATION_EVENT, sim_clock, msg);
    }
    /* Process the events programmed for the current cycle   */
    processEvents(sim_clock);
    /* ...                                                   */
}
```

Using VEF TraceLIB in your simulator

- Comunicating a message reception to TraceLIB:

```
while (!isTraceComplete())
{
    /* ...                                                    */
    /* Check if there are received messages after the event
       processing                                             */
    while (areReceivedMsg())
    {
        msg_t msg;
        /* Get the received message and report to TraceLIB   */
        getReceivedMessage(&msg);
        ReceiveTraceMsg(msg.dst, msg.id);
    }
    /* Update the clocks                                      */
    sim_clock++;
    clock_tictac();
}
/* The simulation  finishes and the statistics are recorded.  */
record_sim_stats();
```

Using the `tracetor` tool

- It is **installed automatically** with the TraceLib library
- Optional arguments:

```
-n|-N : number of NICS
-f|-F : factor time. Default: 1
-c|-C : simulation clock. Default: no use (overwrite -F flag if this flag is specified)
-w|-W : size of window. Default: 1000
-nicSize: size of NIC in bytes. Default 0 (ideal NIC)
-nicBW: input bandwitch of NIC in bytes/cycle. Default 0 (ideal NIC)
-intra|-INTRA: bandwitch of intra Cards in bytes/cycle. Default: 0 (Ideal intra card.)
-MPI|-mpi :type of MPI groupComm. Default: 1 (OPENMPI basic O(N))
-cpuXnic: sets the CPUS attached per NIC
-NOC:  reads the .names file to automap the memory devices
-h|-H : show this text
```

Special Thanks!

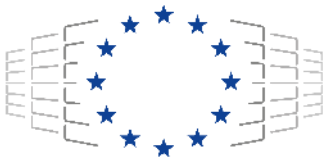
- Francisco J. Andújar, UVA, Spain
- Miguel Sánchez de La Rosa, UCLM, Spain
- Gabriel Gómez López, UCLM, Spain
- Carlos Medrano Navalón, UCLM, Spain
- Francisco J. Alfaro, UCLM, Spain
- Pedro J. García, UCLM, Spain
- **… and all the individuals from the 3-SEA projects that have contributed to improving the VEF Traces framework and populating the VEF Traces repository**

Agenda

- The VEF Traces Framework
- VEF Traces file format
- Generating VEF Traces
- Static analysis
- Using TraceLib in 3rd party simulation tools
- **Open discussion**

The VEF traces framework: collecting traffic traces from modern, highly-demanding applications

Jesus Escudero-Sahuquillo, Pedro J. García, UCLM, Spain
Francisco J. Andujar, UVA, Spain



EuroHPC
Joint Undertaking

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955776. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Greece, Germany, Spain, Italy, Switzerland.