VEF Traces Framework Tutorial

Francisco J. Andújar
Universidad de Valladolid

Jesús Escudero-Sahuquillo, Pedro J. García, Francisco J. Alfaro, José L. Sánchez and Francisco J. Quiles
Universidad de Castilla-La Mancha

jesus.escudero@uclm.es

This work has been supported by the Spanish Ministerio de Ciencia e Innovación, under the project **PID2019-109001RA-I00**

Agenda

- The VEF Traces Framework
- VEF Trace Format
- Obtaining VEF Traces
- Using TraceLib in 3rd party simulation tools
- Extending the VEF Traces framework

Agenda

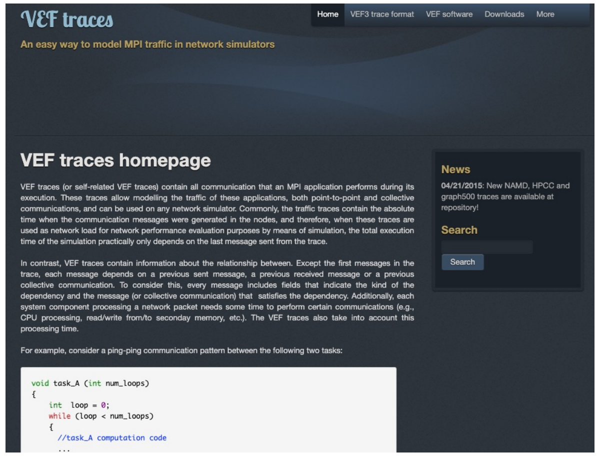
- **The VEF Traces Framework**
- VEF Trace Format
- Obtaining VEF Traces
- Using TraceLib in 3rd party simulation tools
- Extending the VEF Traces framework

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:
  - http://www.i3a.uclm.es/VEFtraces/
  - https://gitraap.i3a.info/fandujar/VEF-TraceLIB
  - https://gitraap.i3a.info/fandujar/VEF-Prospector



An easy way to model MPI traffic in network simulators

VEF traces homepage

VEF traces (or self-related VEF traces) contain all communication that an MPI application performs during its execution. These traces allow modelling the traffic of these applications, both point-to-point and collective communications, and can be used on any network simulator. Commonly, the traffic traces contain the absolute time when the communication messages were generated in the nodes, and therefore, when these traces are used as network load for network performance evaluation purposes by means of simulation, the total execution time of the simulation practically only depends on the last message sent from the trace.

In contrast, VEF traces contain information about the relationship between. Except the first messages in the trace, each message depends on a previous sent message, a previous received message or a previous collective communication. To consider this, every message includes fields that indicate the kind of the dependency and the message (or collective communication) that satisfies the dependency. Additionally, each system component processing a network packet needs some time to perform certain communications (e.g., CPU processing, read/write from/to seconday memory, etc.). The VEF traces also take into account this processing time.

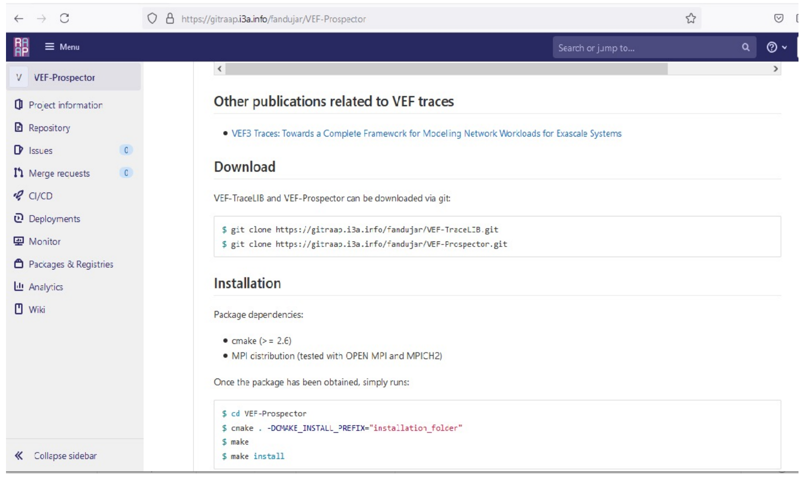
For example, consider a ping-ping communication pattern between the following two tasks:

```
void task_A (int num_loops)
{
    int  loop = 0;
    while (loop < num_loops)
    {
      //task_A computation code
      ...
```

News

04/21/2015: New NAMD, HPCC and graph500 traces are available at repository!

Search



Other publications related to VEF traces

- VEF3 Traces: Towards a Complete Framework for Modeling Network Workloads for Exascale Systems

Download

VEF-TraceLIB and VEF-Prospector can be downloaded via git:

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

Installation

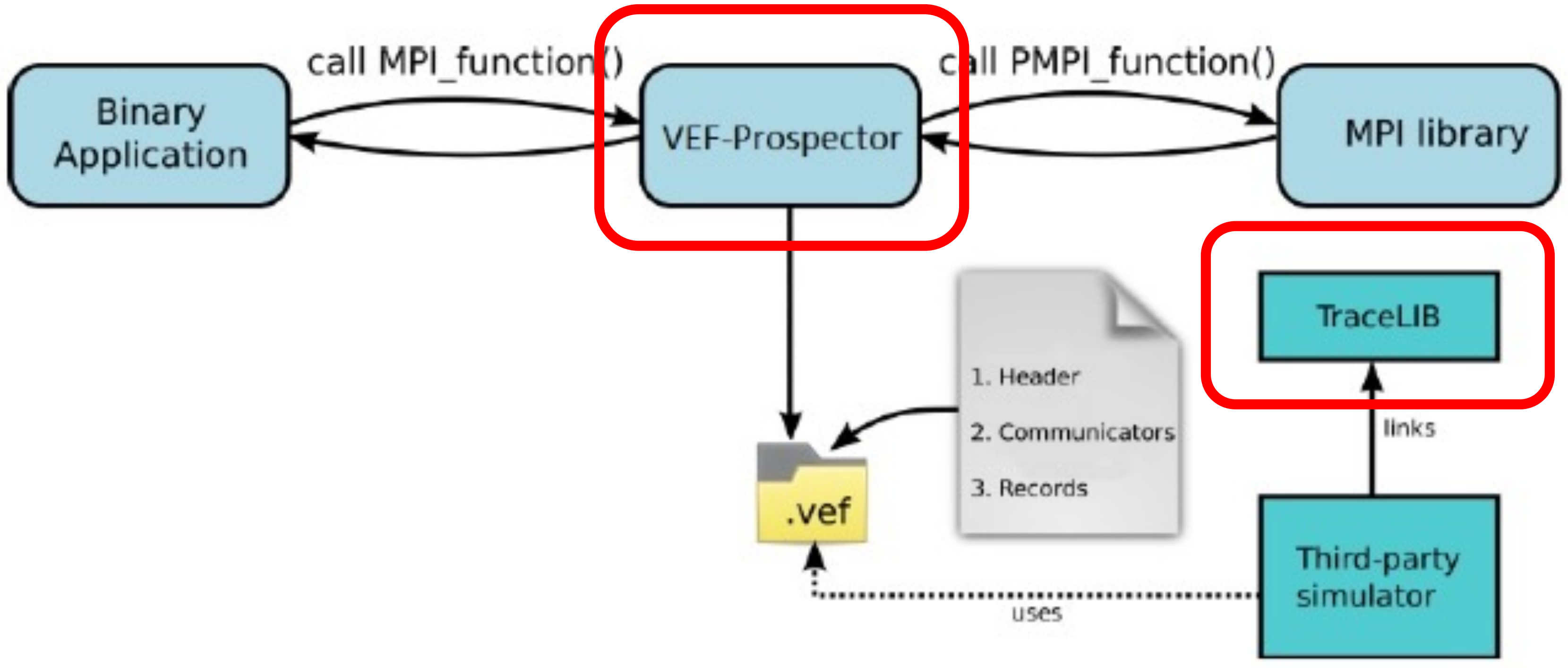
Package dependencies:

- cmake (>= 2.6)
- MPI distribution (tested with OPEN MPI and MPICH2)

Once the package has been obtained, simply runs:

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

Obtaining VEF traces



What is a self-related trace?

Let’s consider tasks A and B:

```
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

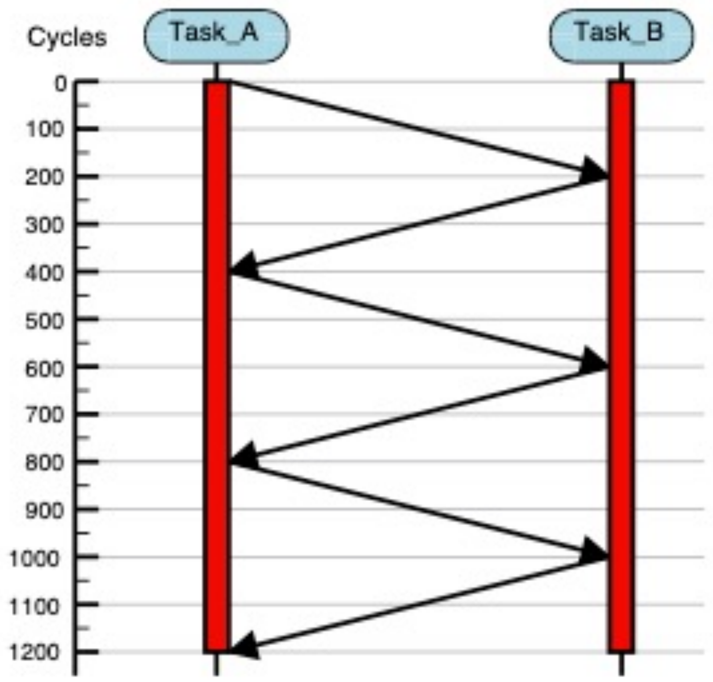
Considering absolute timestamps and ignoring the message size, a possible trace is:

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

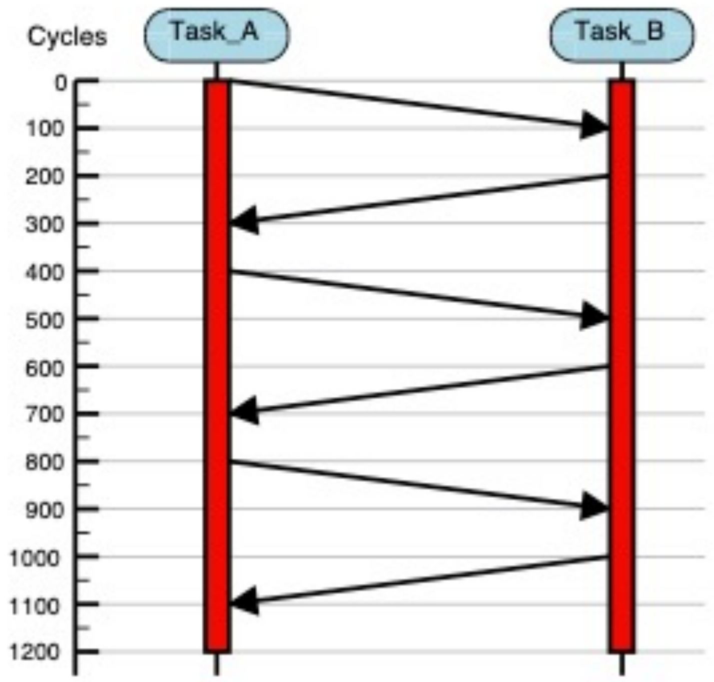
...

The problem of absolute timestamps

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



200-cycle message delivery

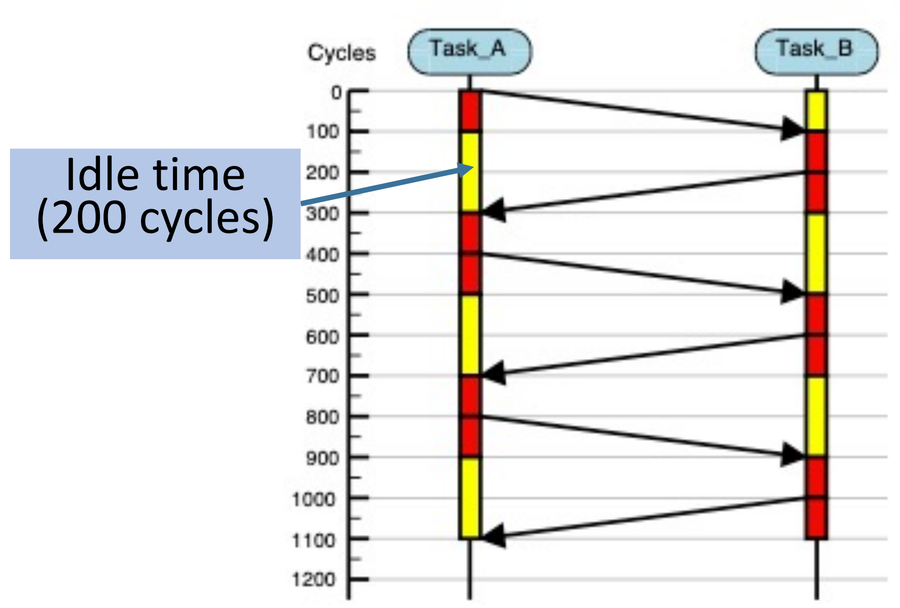


100-cycle message delivery

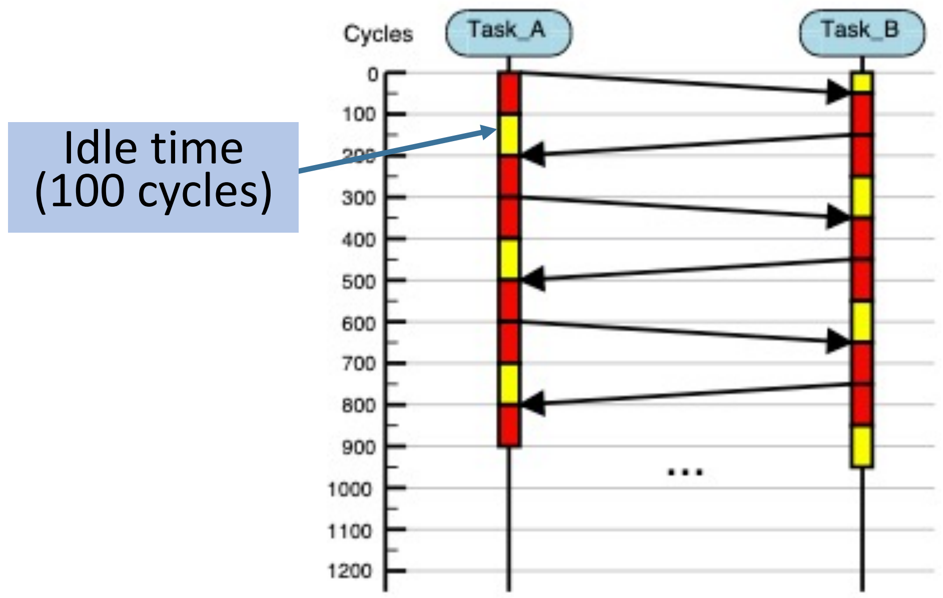
Self-related trace

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

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



100-cycle message delivery



50-cycle message delivery

Agenda

- The VEF Traces Framework
- **VEF Trace Format**
- Obtaining VEF Traces
- Using TraceLib in 3rd party simulation tools
- Extending the VEF Traces framework

VEF trace 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 trace 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

VEF trace 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)

VEF trace 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

VEF trace 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

VEF trace 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 express the self-relationship between records:

VEF trace 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 express the self-relationship between records:

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

VEF trace 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 express the self-relationship between records:

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

VEF trace 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

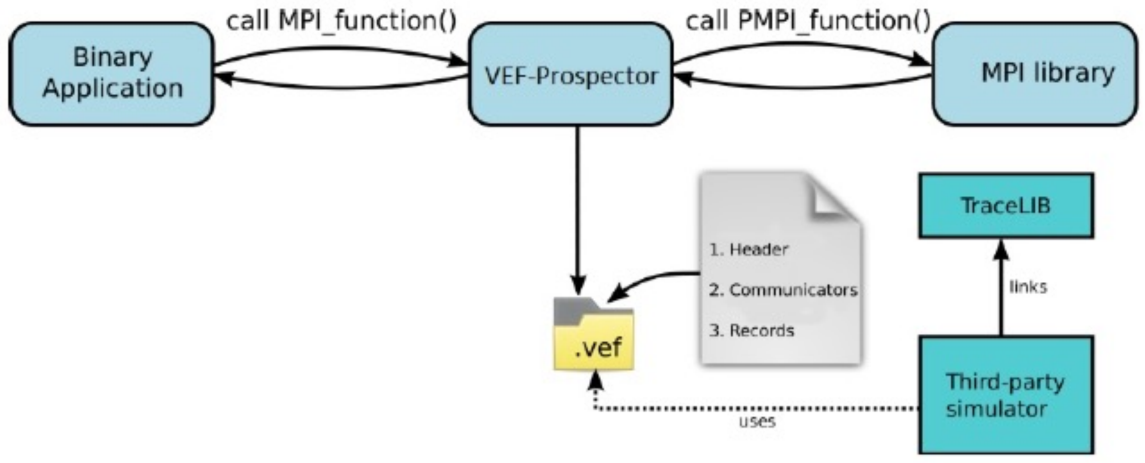
The last three fields express the self-relationship between records:

- 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 Trace Format
- **Obtaining VEF Traces**
- Using TraceLib in 3rd party simulation tools
- Extending the VEF Traces framework

Obtaining VEF Traces



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

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

- **Package dependencies**:
  - cmake (>= 2.6)
  - MPI distribution (tested with OPEN MPI and MPICH2)

Obtaining VEF Traces

- **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`

Obtaining VEF Traces

Executables in the VEF-Prospector repo

- **vef_mixer**: mix 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**: Update 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

Obtaining 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.

Obtaining 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**.

Obtaining 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
```

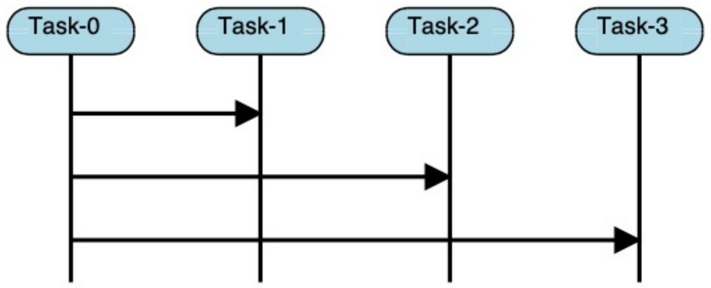
- This finishes the process to generate a VEF trace.
- VEF Traces can be tested using the `tracetor` tool, provided with the **TraceLIB library**.

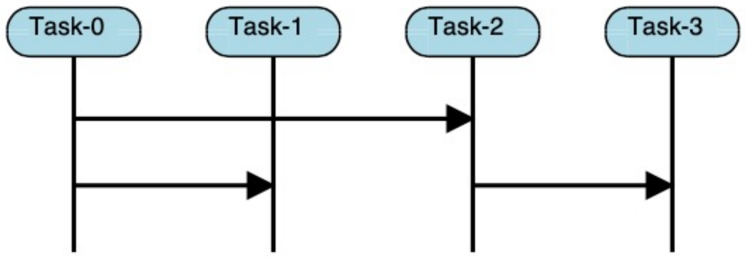
Agenda

- The VEF Traces Framework
- VEF Trace Format
- Obtaining VEF Traces
- **Using TraceLib in 3rd party simulation tools**
- Extending the VEF Traces framework

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.


Broadcast O(n)


Broadcast O(logn)

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);
    /* ...                                                */
}
```

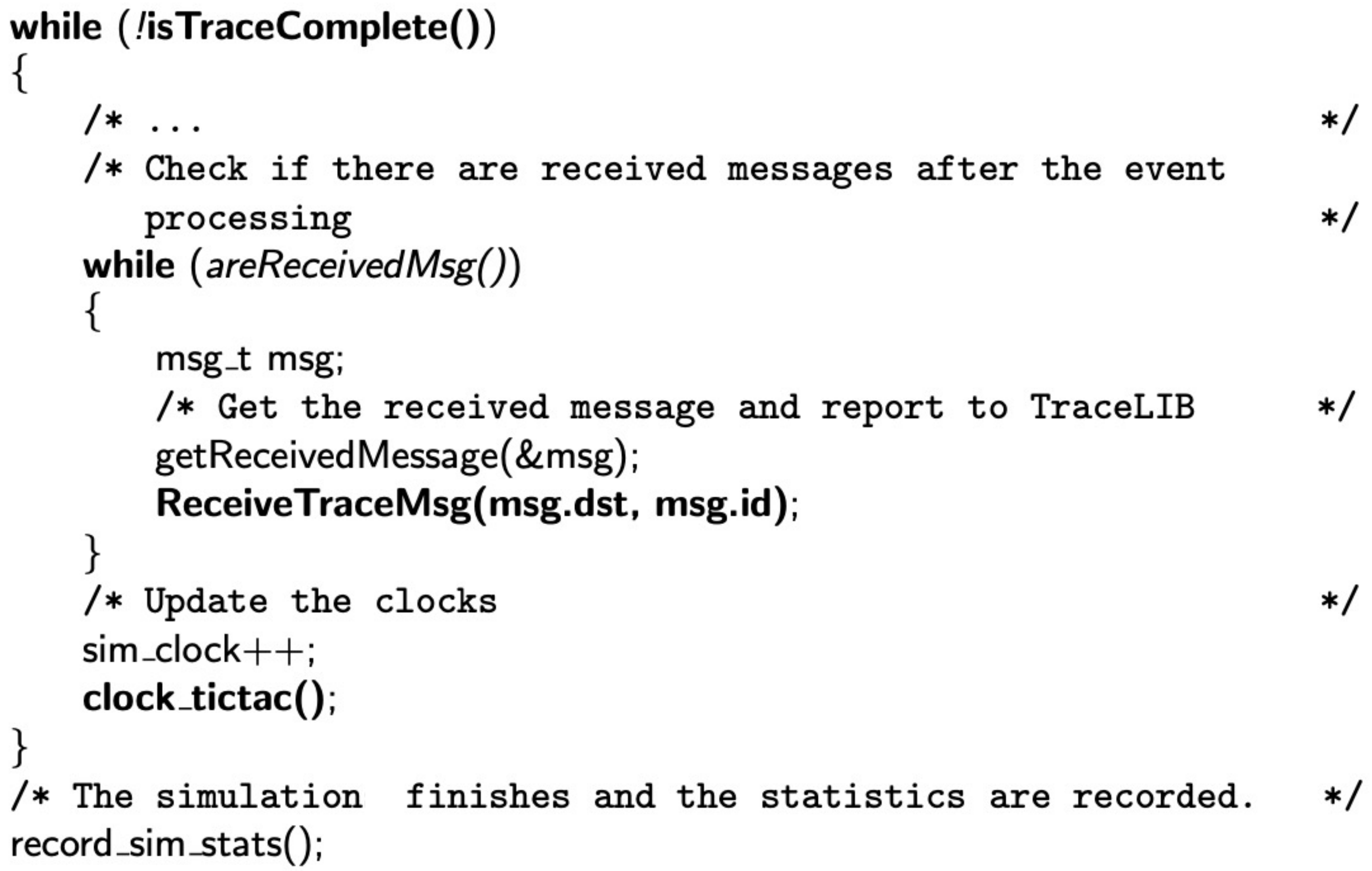
Using VEF TraceLIB in your simulator

- **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
```

Agenda

- The VEF Traces Framework
- VEF Trace Format
- Obtaining VEF Traces
- Using TraceLib in 3rd party simulation tools
- **Extending the VEF Traces framework (work-in-progress)**

Extending VEF traces (WIP)

- Data-center networks (DCNs) exhibit different network workloads than HPC applications based on MPI
  - Web search, social network, streaming, etc.
- These applications require to exchange large amount of data
- Normally, we do not have a datacenter to instrumentalize the network and to model its traffic (as we do with MPI-based applications).
- Modeling DCNs workloads is crucial.
- How can implement DCN workloads in the VEF trace framework?

Extending VEF traces (WIP)

- How can implement DCN workloads in VEF trace framework?
  - We have added a new trace format: The **Extended VEF** trace format:
    - An extended header register defines the type of traffic model
    - One or more body registers define the characteristics of the traffic model
    - This scheme can be used to add more traffic models, not only DCN workloads
  - **VEF-TraceLIB has been modified** to read these new trace format:
    - Only the code of the trace reader has been modified.
    - The new trace reader generates internal messages using the same structures than the MPI traces, avoiding changes in other parts of the library.
    - All the library features are available for extended traces
    - And no changes in the library interface are required.
    - From the user's point of view, there are no differences between simulating a MPI VEF trace or an Extended VEF trace.
    - In fact, MPI and Extended VEF traces can be simultaneously simulated

Extended VEF trace format (WIP)

Type of records:

1. Trace header
   - Same format than MPI traces

```
VEF3 64 200000 0 0 0 0 1000
EXT DCNW
GOOGLE_SEARCH GAUSS LOAD 0.5
```

Extended VEF trace format (WIP)

Type of records:

1. Trace header
    - Same format than MPI traces
2. Extended Header
    - ***EXT*** indicates that this is an extended trace
    - ***DCNW*** indicates the workload model

```
VEF3 64 200000 0 0 0 0 1000
EXT DCNW
GOOGLE_SEARCH GAUSS LOAD 0.5
```

Extended VEF trace format (WIP)

```
VEF3 64 200000 0 0 0 0 1000
EXT DCNW
GOOGLE_SEARCH GAUSS LOAD 0.5
```

Type of records:

1. Trace header
   - Same format than MPI traces
2. Extended Header
   - ***EXT*** indicates that this is an extended trace
   - ***DCNW*** indicates the workload model
3. Body records
   - Configures the model.
   - Depends on the workload models
   - Its number depends on the workload model

DCN traffic models for Extended VEF traces (WIP)

- The DCN traffic models are based on date obtained from published papers.
- We support two DCNs traffic models:
  - **DCNW model:** Simulates different workload models reported by several companies like Facebook or Google
    - Based on the workloads collected by Montazery et al. in their work **“Homa: a receiver-driven low-latency transport protocol using network priorities”**
  - **FLOWS model:** Simulates multiples traffic classes to define different services in the DCN. Each traffic class comprises several flows of data with different characterizations
    - Based on the workloads defined in the paper **“Modeling Traffic Workloads in Data-center Network Simulation Tools”** by Gonzalez-Naharro et al, HPCS 2019.
- The traces can be manually created according their format or using an interactive application provided with VEF Trace-LIB

DCNW model (WIP)

```
VEF3 64 200000 0 0 0 0 1000
EXT DCNW
FB_HADOOP GAUSS LOAD 0.5
```

Characterization of DCNW model

1. The trace header indicates the number of tasks and the number of messages to send.
2. Model: specifies the message size distribution, based on files with the accumulated probability
   - Multiple distributions, such as Facebook Hadoop servers, Google search servers, etc.
   - Also allows custom message size distributions
3. Message generation distribution: Depends on the Model selected. It can be a Pareto or an exponential distribution
   - The default load can be modified to increase or reduce the network load.
4. Destination distribution: specifies the distribution of the message destination
   - Uniform, gaussian and poison distribution available

FLOWS model (WIP)

Characterization of FLOWS model

1. The trace header indicates the number of tasks and the number of flows to send.
2. Main configuration: indicates the number of classes, the maximum time to generate the flows, and the default destination distribution (same distributions than DCNW model)

```
VEF3 64 30000 0 0 0 0 1000
EXT FLOWS
CLASSES 2 TIME 2000000  POISSON
0 MAX 100 MIN 10 PERC 45.5
1 MIN 1000 PERC 54.5 GAUSS MAX 10000
```

FLOWS model (WIP)

```
VEF3 64 30000 0 0 0 0 1000
EXT FLOWS
CLASSES 2 TIME 2000000  POISSON
0 MAX 100 MIN 10 PERC 45.5
1 MIN 1000 PERC 54.5 GAUSS MAX 10000
```

Characterization of FLOWS model

1. The trace header indicates the number of tasks and the number of flows to send.
2. Main configuration: indicates the number of classes, the maximum time to generate the flows, and the default destination distribution (same distributions than DCNW model)
3. Traffic class configuration:
   - Class identifier
   - Minimum and maximum message size, measured in Kbytes. The message size distribution is a uniform distribution between this two bounds.
   - Destination distribution: overwrites the default destination distribution
   - Percentage of flows generated. This percentage and the maximum time determines the message generation, that follows an exponential distribution.

VEF Traces Framework Tutorial

Francisco J. Andújar
Universidad de Valladolid

Jesús Escudero-Sahuquillo, Pedro J. García, Francisco J. Alfaro, José L. Sánchez and Francisco J. Quiles
Universidad de Castilla-La Mancha

jesus.escudero@uclm.es

This work has been supported by the Spanish Ministerio de Ciencia e Innovación, under the project **PID2019-109001RA-I00**