Debugging and Tracing of Many-core Processors

Simon Marchi

DORSAL Laboratory
Department of Computer and Software Engineering

POLYTECHNIQUE MONTRÉAL
AFFILIÉE À L'UNIVERSITÉ DE MONTRÉAL

Plan

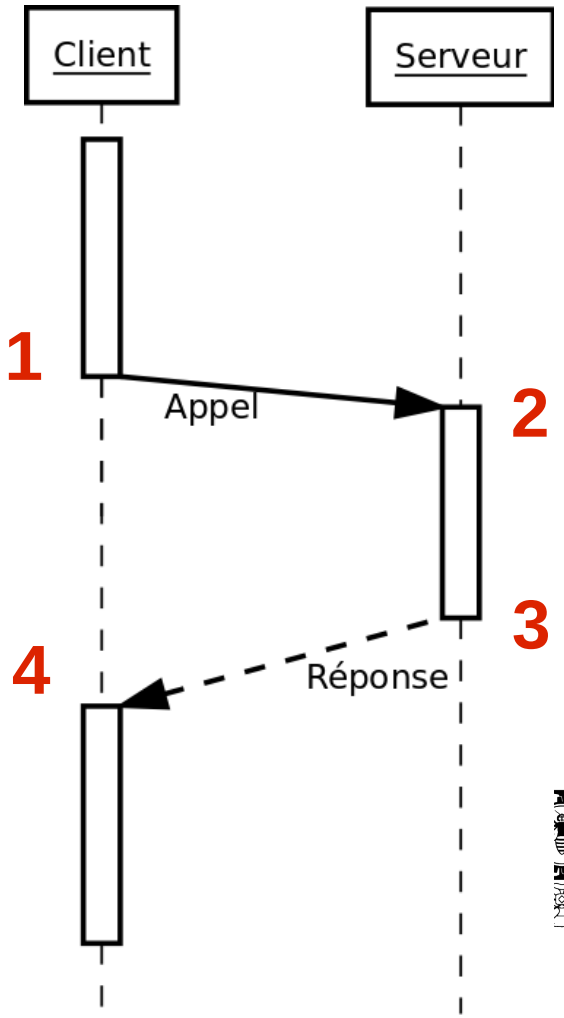
- Introduction
- Remote procedure calls debugging
- Tracing many-core processors
- Conclusion

Remote Procedure Calls (RPC)

- Calling functions in a different address space
- Define a logical thread across RPCs
- Syntax similar to regular function calls
- Hidden complexity of remote communications

Remote Procedure Calls, description

1. Encoding input parameters
2. Decoding by server
3. Encoding of output parameters after remote execution
4. Decoding of output parameters and continuation of normal operation



Remote Procedure Calls, examples

- Protocols: ONC (SunRPC), D-Bus, XML-RPC, Microsoft DCOM and WCF...
- Intel Xeon Phi co-processor: offload mode
- Custom implementation: LTTng, Transmission
- Other: API REST (Web applications)

Remote Procedure Calls Debugging, objectives

- Avoid the need to use two debuggers, see a partial story on each, and go from one to the other
  - Support different Remote Procedure Call paradigms
  - Hide the Remote Procedure Call Machinery

RPC Debugging, existing solutions

- Microsoft Visual Studio: DCOM and WCF
  - Automatically attaches to the server process
  - Application-specific proprietary solution
- Intel Xeon Phi: debugging offload mode
  - Based on GDB and Eclipse
  - Two instances of GDB created by Eclipse

RPC Debugging, with GDB ?

- Multi-process
- Non-stop mode
- Python extensions
  - ☻ Breakpoint and FinishBreakPoint
  - ☻ Callbacks on breakpoints
  - ☻ Listing threads and processes
  - ☻ Thread stack unwinding
  - ☻ Dynamically evaluate complex expressions
  - ☹ Control the execution of debugged threads
  - ☹ Changing the active thread in the interface

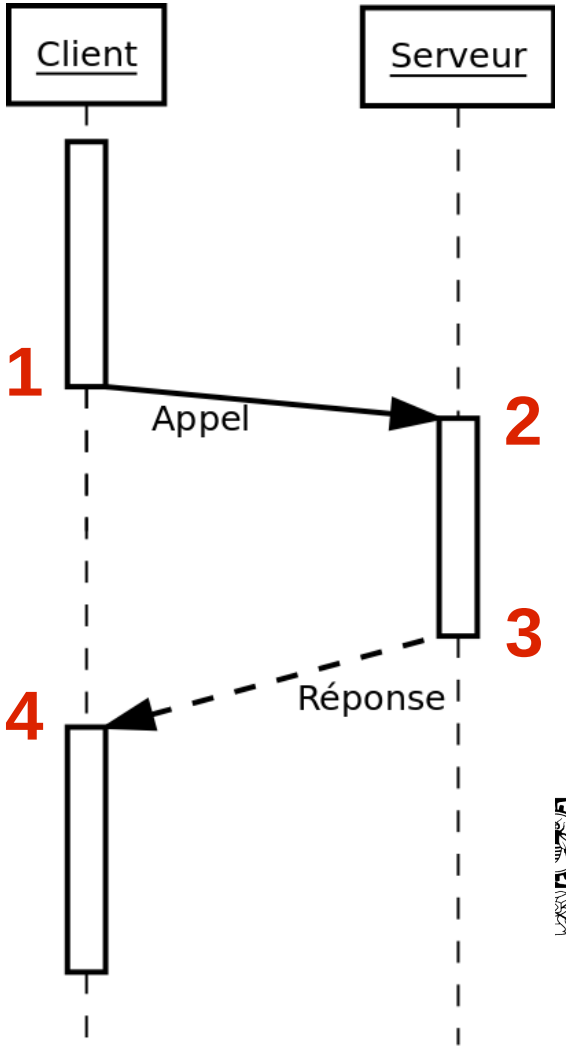
RPC Debugging, solution overview

- Python extensions in GDB
- Four new commands
  - step-rpc
    - Step checking if a remote procedure call is initiated.
  - finish-rpc
    - Complete the current remote procedure call and stop.
  - backtrace-rpc
    - Display the thread stack, combining the functions called within the server and the calling functions in the client, hiding the RPC layer.
  - bind-rpc
    - Specify a relation between a client and a server function
- Ex.: ONC, xmlrpc-c, gdbus, LTTng, Transmission.

RPC Debugging, inner working

Internal breakpoints

Step	Who	Event
1	Client	RPC initiation
2	Server	Server function execution
3	Server	End of server function
4	Client	Receiving the result and resuming normal execution



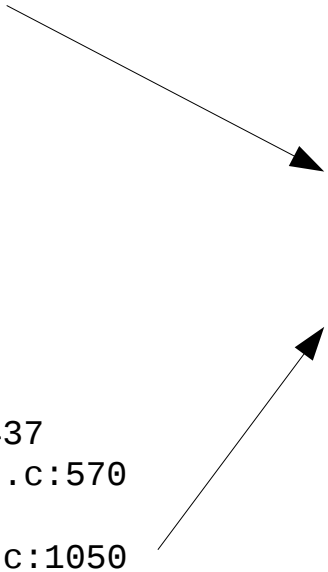
RPC Debugging, example backtrace-rpc

Server

```
#0  do_things at server.c:12
#1  sample_add at server.c:32
#2  callNamedMethod at registry.c:294
#3  xmlrpc_dispatchCall at registry.c:324
#4  xmlrpc_registry_process_call2 at registry.c:413
#5  processCall at xmlrpc_server_abyss.c:475
#6  handleXmlrpcReq at xmlrpc_server_abyss.c:610
#7  runUserHandler at server.c:541
#8  processDataFromClient at server.c:577
#9  serverFunc at server.c:629
#10 connJob at conn.c:39
#11 pthreadStart at thread_pthread.c:49
#12 start_thread at pthread_create.c:301
#13 clone at clone.S:115
```

Client

```
#0  __pselect at pselect.c:73
#1  waitForWork at xmlrpc_curl_transport.c:437
#2  finishCurlMulti at xmlrpc_curl_transport.c:570
#3  performCurlTransaction at
                      xmlrpc_curl_transport.c:1050
#4  performRpc at xmlrpc_curl_transport.c:1155
#5  call at xmlrpc_curl_transport.c:1376
#6  xmlrpc_client_call2 at xmlrpc_client.c:580
#7  clientCall_va at xmlrpc_client_global.c:147
#8  xmlrpc_client_call at xmlrpc_client_global.c:174
#9  add at client.c:36
#10 main at client.c:62
```



```
server - #0 do_things at server.c:12
server - #1 sample_add at server.c:32
client - #2 add    at client.c:36
client - #3 main at client.c:62
```

RPC Debugging, limitations

- GDB currently requires both processes to be on the same node
  - Integrate at the Eclipse CDT level?
  - Multi-target GDB?
- Network protocol timeouts may be triggered during interactive debugging sessions
  - Increase tremendously the default timeout values when debugging
- How to differentiate generated code from normal source code

Tracing Many-core Processors

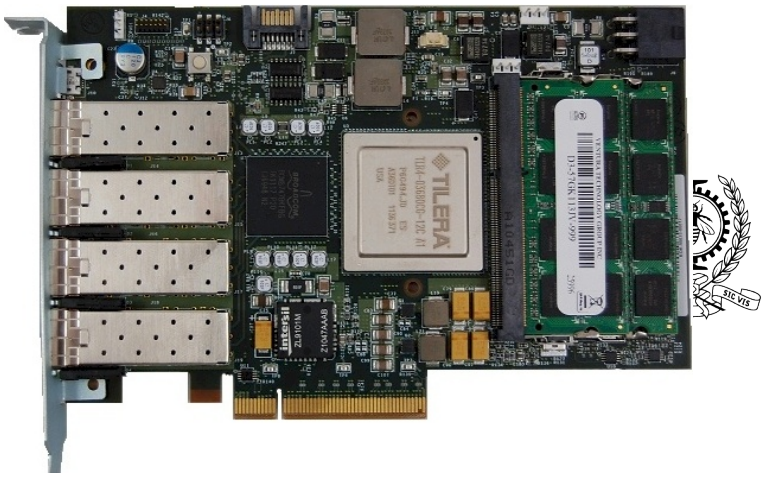
- Many-core processors with tens or hundreds of cores (240 virtual cores on Intel Xeon Phi).
- Highly parallel computing (e.g. GPGPU).
- Scalable performance, energy efficiency.

Many-core Processors Tracing

- Tracing, when debugging is too intrusive
- Insert tracepoints in kernel or user space code
- Event generated when a tracepoint is encountered
- Trace: collection of events
- Linux kernel tracers: LTTng, ftrace, SystemTap...

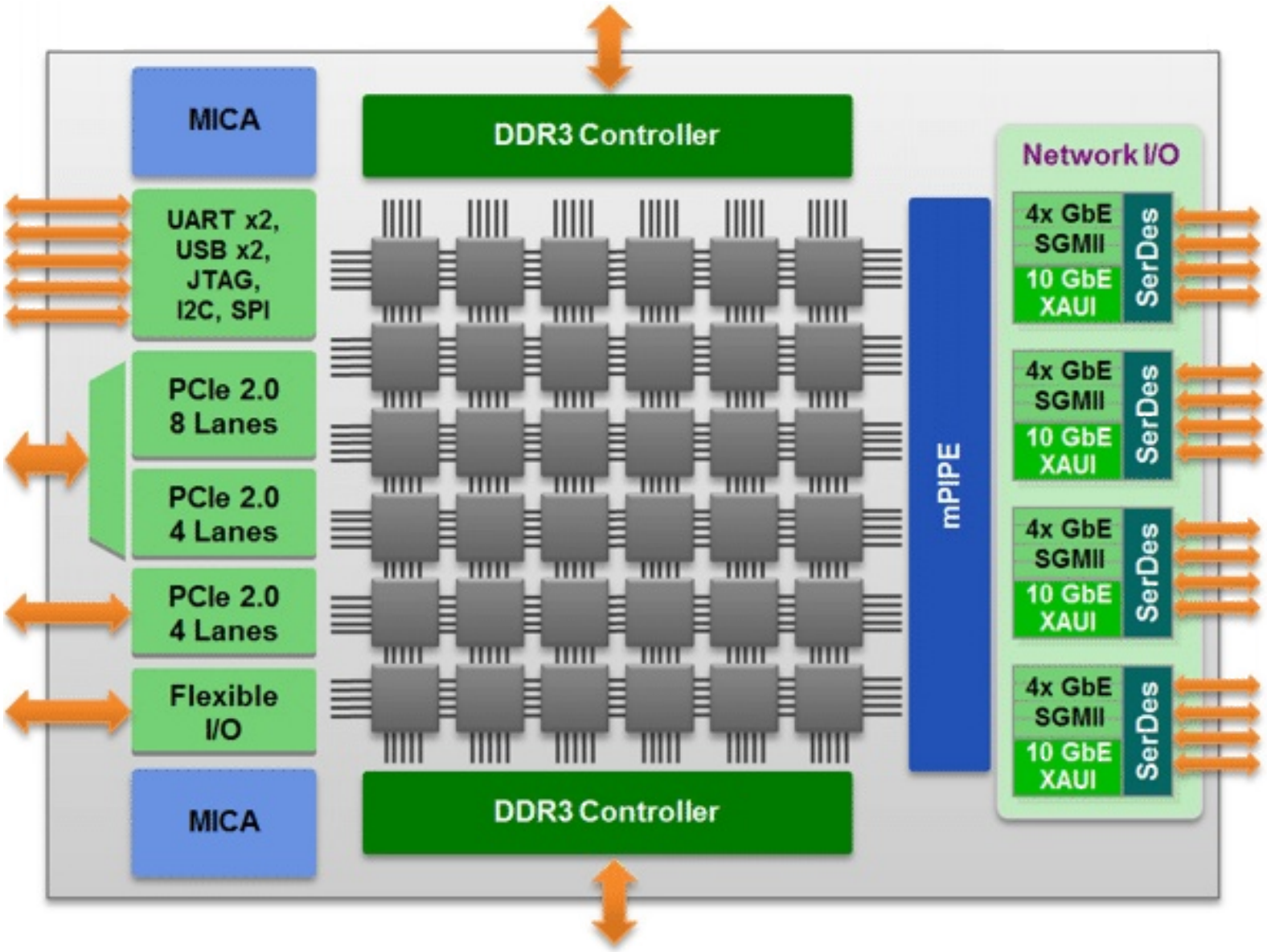
Tilera Many-core Processors

- Tilera TILE-Gx36 (36 cores)
- Standalone or co-processor board
- Typical applications:
  - Multimedia: video encoding and decoding
  - Security: hardware assisted encryption
  - Networking: packet inspection and routing
  - Cloud computing: distributed cache
- 8 Gibi RAM / FS



Source: http://regmedia.co.uk/2012/01/28/tilera_tileencore_adapter.jpg

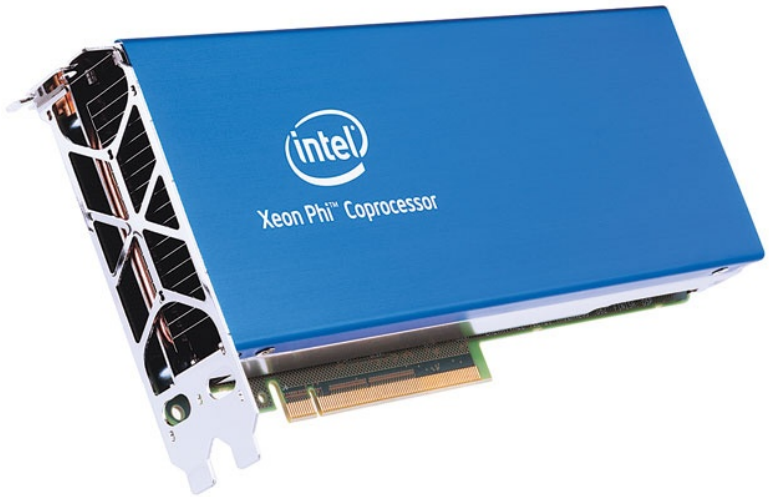
Tilera Many-core Processors



Source: http://www.wirmax.it/images/prodotti_immagini/mikrotik/tilera36.gif

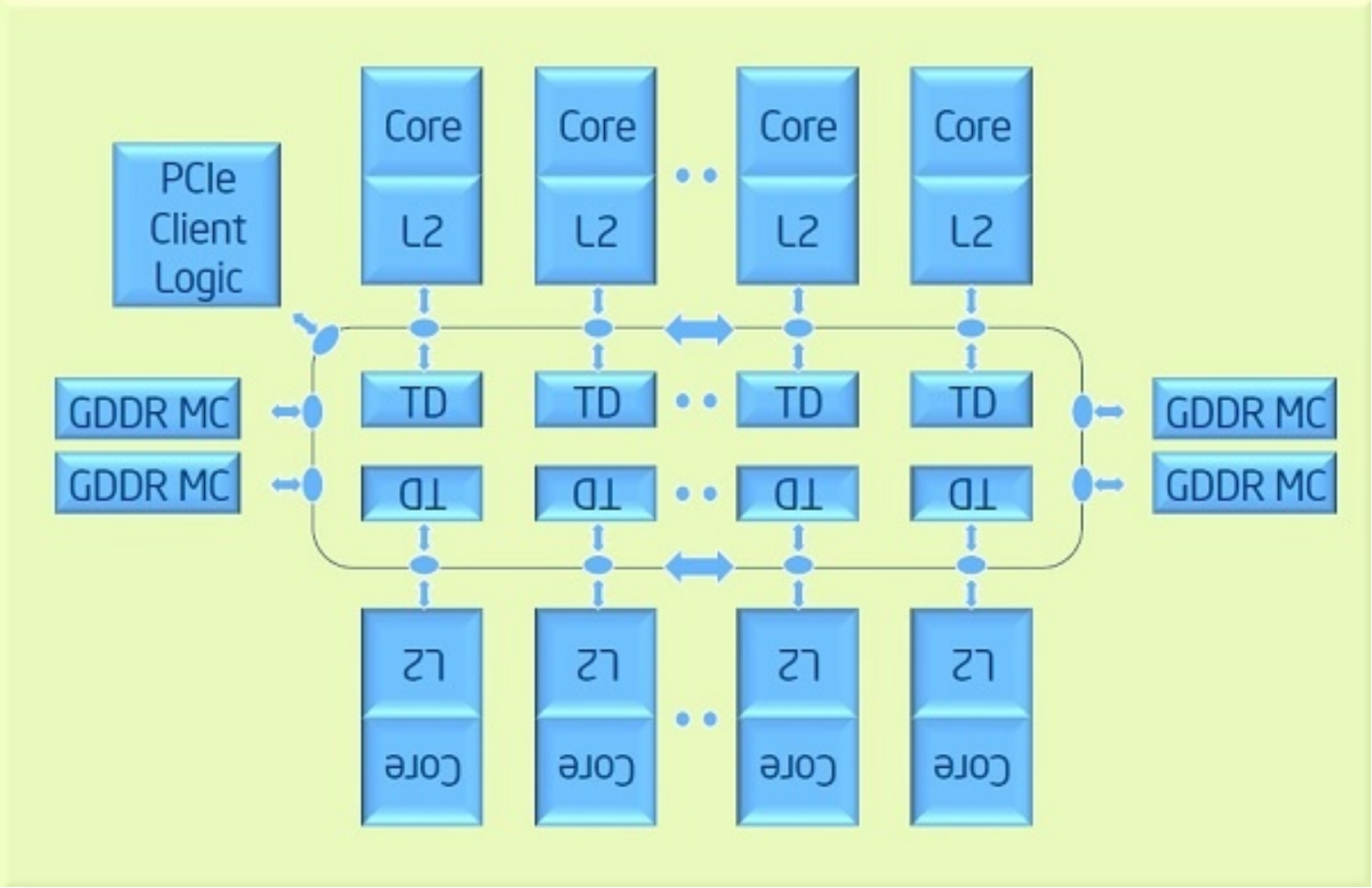
Xeon Phi Many-Core Processors

- Intel Xeon Phi (57 x 4 logical cores)
- Currently available only as co-processor
- Typical applications:
  - Scientific computing
  - Cluster
- 6 Gibi RAM / FS



Source: http://spectrum.ieee.org/img/08NIntelPhimaster-1374245245264.jpg

Xeon Phi Many-Core Processors



Source: http://img.hexus.net/v2/lowe/News/Intel/phi1.jpg

Tracing Many-core Processors, objectives

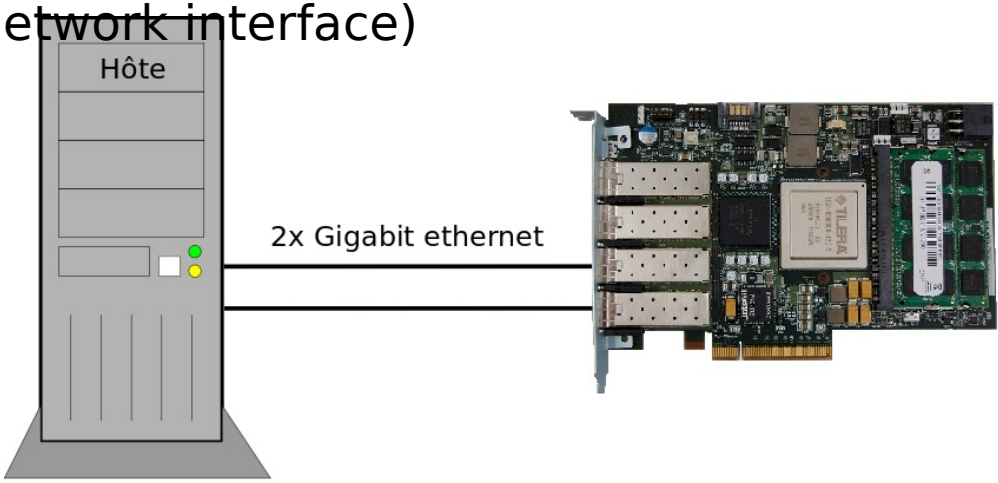
- Porting LTTng to Tilera TILE-Gx and Intel Xeon Phi.
- Evaluate the impact of LTTng on application performance for these many-core processors.

LTTng porting

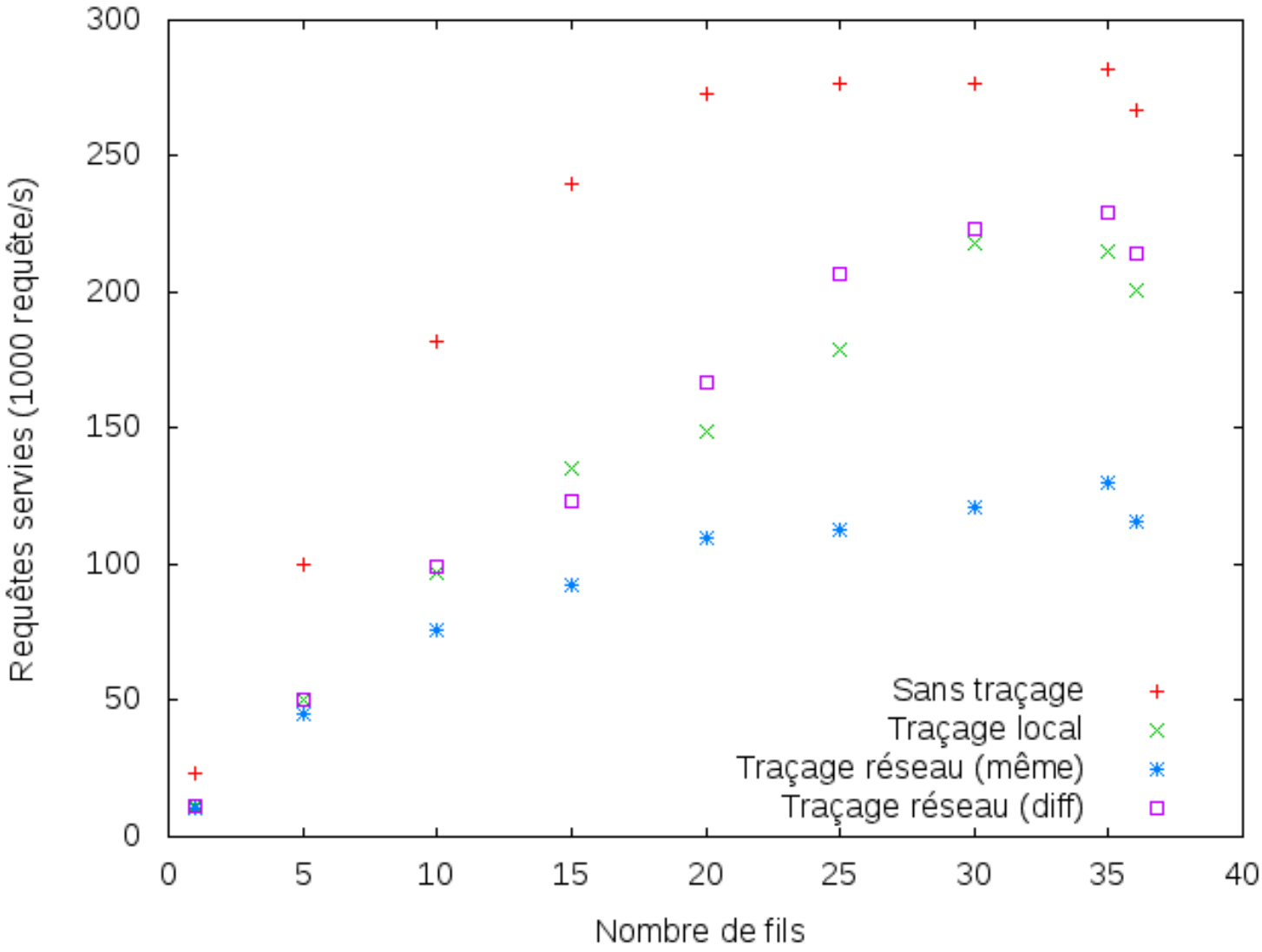
- Modifications
  - Linux kernel
    - System *tracehook* and system calls tracepoints for TILE-GX
  - LTTng
    - System calls descriptions for TILE-Gx
    - A few other minor issues
  - *userspace-rcu*
    - Redefine memory barriers for Intel Xeon Phi

Tilera Many-core Processors, performance tests

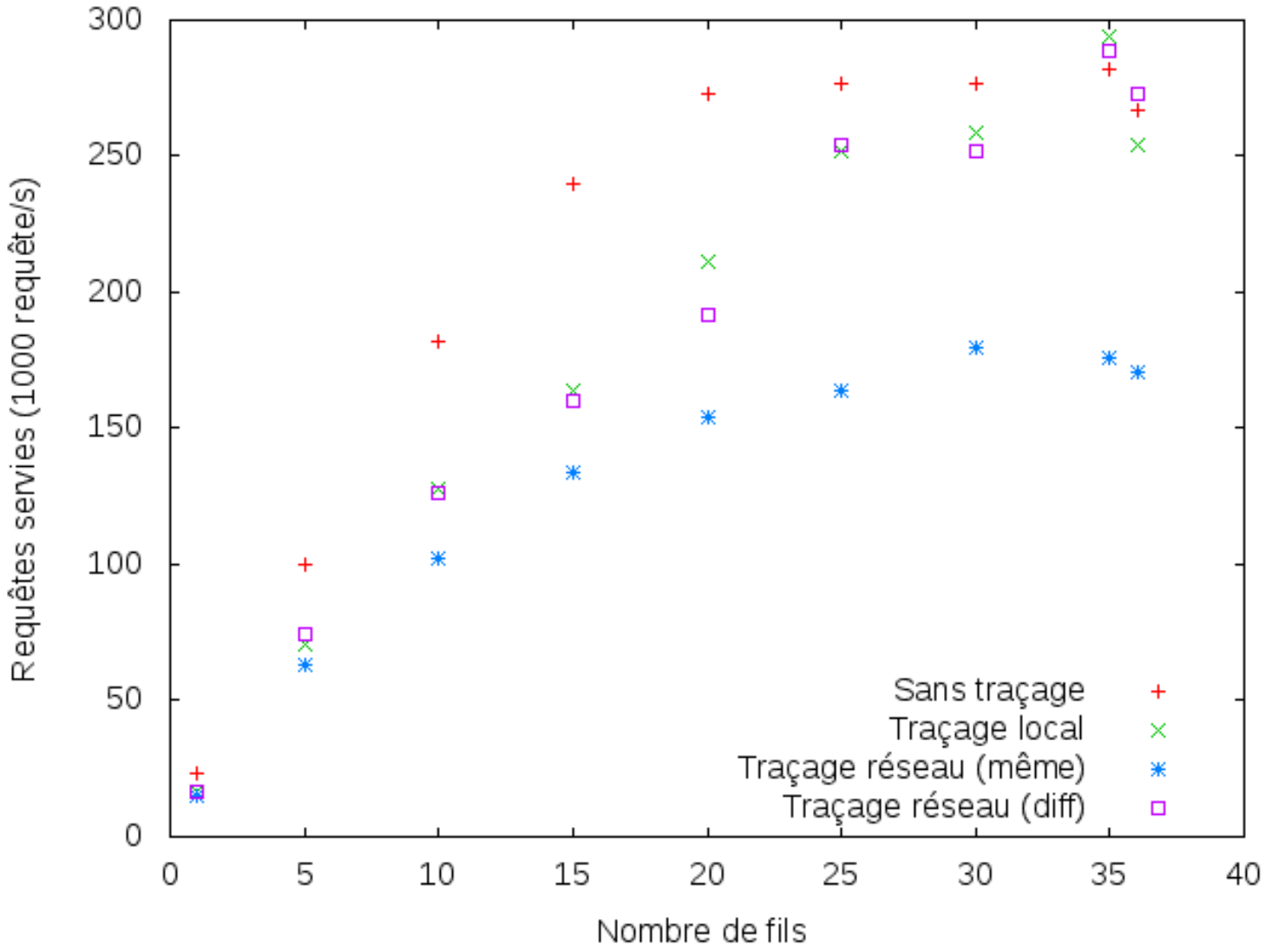
- memcached – distributed cache
- Host simulates numerous clients
- 4 tracing modes
  - No tracing
  - Local storage
  - Trace sent over network (same network interface as requests)
  - Trace sent over network (different network interface)
- 2 Events selected
  - All kernel events
  - Small selection of kernel events



Results – Tilera 1



Results – Tilera 2



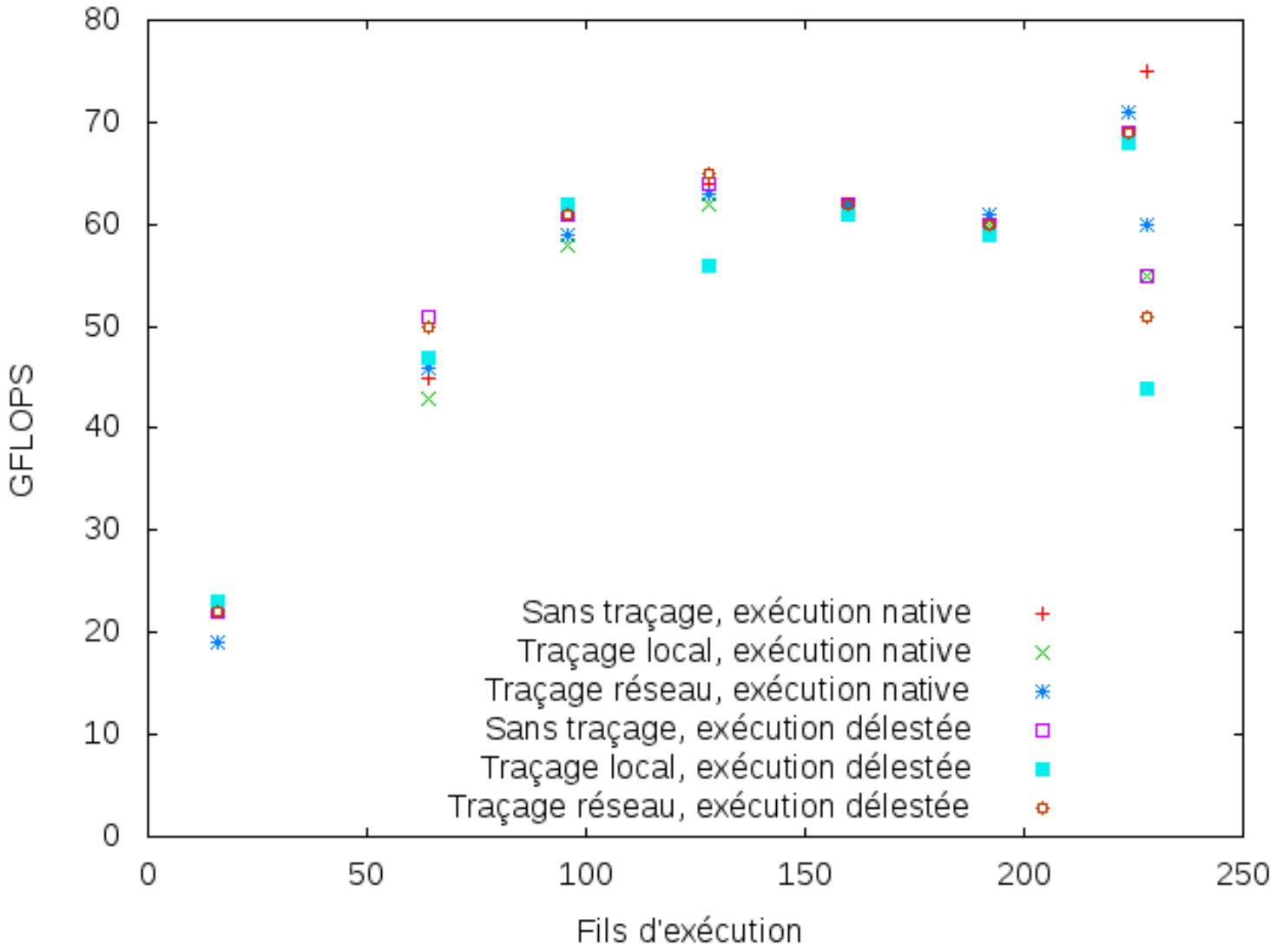
Tilera – Lost events

# fils	Sans traçage	Traçage local		Traçage réseau (même)		Traçage réseau (diff)	
		Tous	Réduite	Tous	Réduite	Tous	Réduite
1	-						
5	-			3378659		2456947	
10	-			9137452		9382781	
15	-			14320257		15588129	19310
20	-			18167592	3041	19922867	190202
25	-			20746223	11996	25355095	747052
30	-			23463356	180796	28863393	471267
35	-			23748875	307231	29604835	504430
36	-			24230644	611770	30314552	1164407

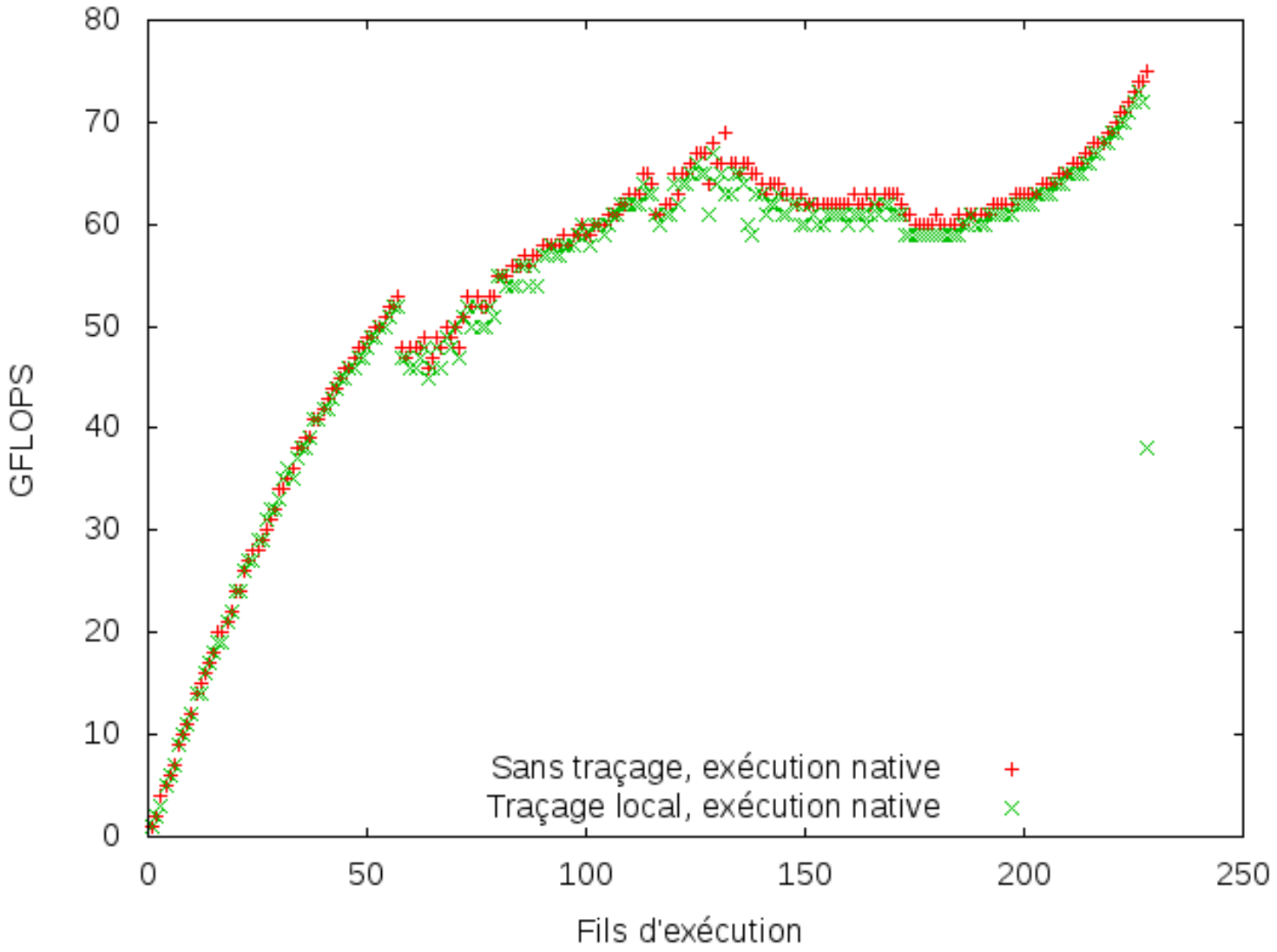
Xeon Phi Many-core Processors, performance tests

- Finite elements analysis, diffusion
- 2 execution modes
  - Native execution
  - Offload mode
- 3 tracing modes
  - No tracing
  - Local storage
  - Trace sent over the virtual network (PCI-express)

Results – Xeon Phi 1



Results – Xeon Phi 2



Xeon Phi – Lost events

# fils	Exécution native			Exécution avec délestage		
	Sans	Local	Réseau	Sans	Local	Réseau
16						
64						
96						
128					179139	
160						
192						
224			10158			
228		7102686	1247757		2774194	1392009

Tracing Many-core Processors, possible improvements

- Tilera TILE-Gx
  - Use the PCI-express link to transfer tracing data.
- Intel Xeon Phi
  - More demanding applications such as MPI
  - Combined host / target tracing