

REMOTE PROCEDURE CALL COMPILER FOR FIELD PROGRAMMABLE GATE ARRAYS

Abstract

RPC-FPGA implements the Open Network Computing ONC-RPC remote procedure call protocol specification for use in FPGA accelerators. The compiler generates a High-Level Synthesis (HLS) interface to call a hardware procedure and to stream the data between the processor and the FPGA. The major benefits of RPC-FPGA are:

- hardware procedures running on an FPGA accelerator become accessible for any client in the network,
- the development time of the communication interface between the processor and the FPGA is greatly reduced.

Using RPC-FPGA a speedup gain of 17 to 20 is demonstrated on a square matrix multiplication of N=4096 with network speeds 100 Mbps and 1 Gbps respectively.

Motivation

High-Level Synthesis (HLS) languages define the interface between Host and FPGA using proprietary pragmas (Vivado HLS) , language constructs (OpenCL) or message passing (MPI). Some limitations of current host/fpga interfaces are:

- the hw/sw interface has to be defined and managed by the user
- DMA data transfer requires explicit coding
- data type serialization support is limited
- interfacing leads to substantial programming overhead

Methodology

The interface is defined in a protocol description file (.x) consisting of the arguments declaration and the procedure prototype, following the ONC-RPC syntax. Example: see the protocol description of a dot product.

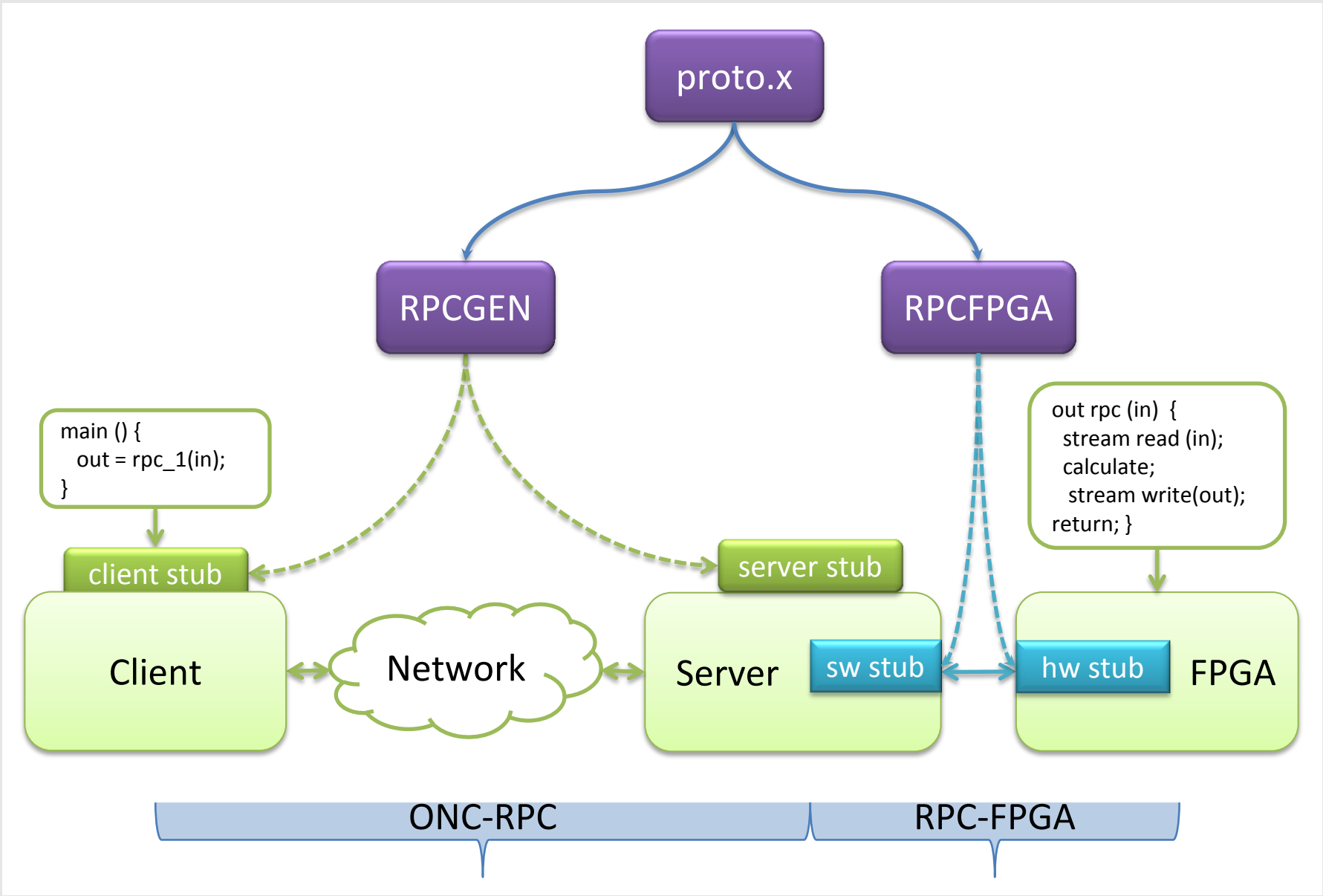
<pre>#ifndef RPC_HDR #define N 20 #endif typedef T float; struct data_in { T a[N]; T b[N]; }; struct data_out { T result; }; program prog { version proc_fpga { data_out dotproduct (data_in) = 1; } = 1; } = 0x31230000;</pre>	<p>1. common client/server header</p> <p>2. type definitions</p> <p>3. structure of I/O arguments</p> <p>4. remote procedure declaration</p>
--	--

Protocol description file dotproduct.x

RPC-Model

We use a 2-stage Remote Procedure Call:

- 1st stage: connect network client and server via ONC-RPC
- 2nd stage: connect server with attached FPGA via RPC-FPGA

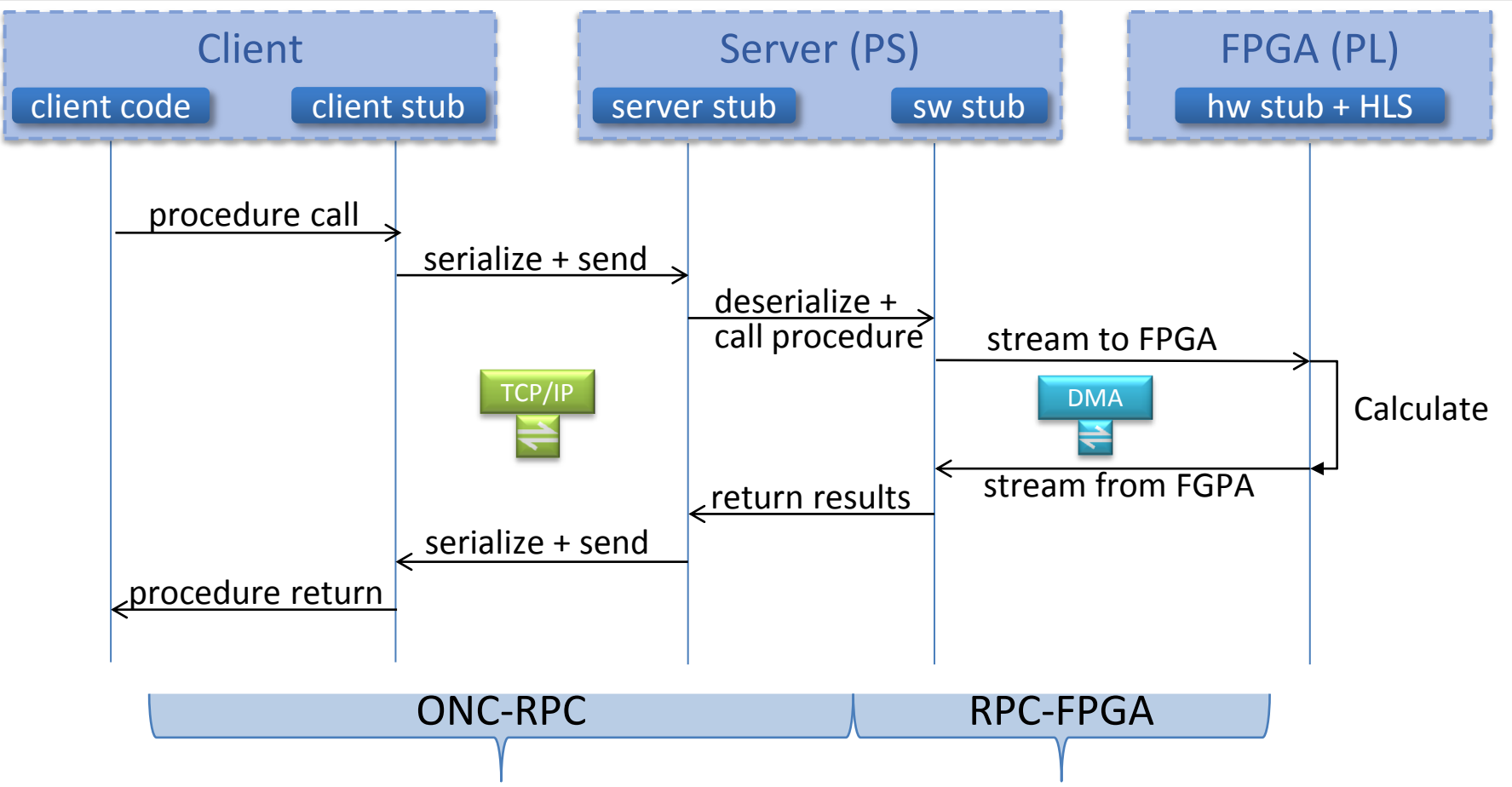


Protocol compilers RPCGEN and RPCFPGA

The client procedure call is relayed to a server with an attached FPGA by stubs generated from the protocol file proto.x using a) **RPCGEN** for the network and b) **RPCFPGA** for the reconfigurable hardware core.

Interfaces

- TCP/IP or TCP/UDP transport between network client and server
- DMA channels (AXI4) between processing system (PS) and programmable logic (PL)



Stubs and transport layers

RPCFPGA

HLS Template

- defines the hardware interface between PS and PL
- declares hw data types and structures of the procedure arguments
- creates logic for streaming data transfer
- manages marshalling of data types and serialization of arrays
- optimizes pipelined throughput to stream at 1 element per cycle
- includes TODO mark to insert procedure body

Example: the HLS template dotproduct.c generated from dotproduct.x

<pre>void dotproduct(uint_t *in, uint_t *out) { // streaming interface definition #pragma HLS INTERFACE axis port=in #pragma HLS INTERFACE axis port=out #pragma HLS INTERFACE ap_ctrl_none port=return // argument and marshalling variables int tmp_int; T tmp_T; data_in data_in; data_out data_out; // read data_in; type conversion int u0, u1; for(u0 = 0; u0 < N; u0++) #pragma HLS PIPELINE data_in.a[u0] = (tmp_int = *(in++) * ((T *) &tmp_int)); for(u1 = 0; u1 < N; u1++) #pragma HLS PIPELINE data_in.b[u1] = (tmp_int = *(in++) * ((T *) &tmp_int)); /** TODO insert procedure body ****/ /******* // write data_out; type conversion *out++ = (tmp_T = data_out.result, *(int *) &tmp_T); return; }</pre>	<p>Interface definition</p> <p>Declarations</p> <p>Stream input</p> <p>Calculate</p> <p>Stream output</p>
--	---

Generated HLS hw stub for the programmable logic (PL)

Server code linking network to FPGA

- service procedure on the server calls the hardware procedure on behalf of the ONC-RPC client
- I/O arguments are sent as uint 32 or 64-bit words from DDR (PS) to BLOCKRAM (PL) using DMA

<pre>data_out * dotproduct_1_svc (data_in *data_in, struct svc_req *req) { // Open streams to/from FPGA int fdr = open("/dev/xillybus_read_32", O_RDONLY); int fdw = open("/dev/xillybus_write_32", O_WRONLY); // stream data "data_in" to FPGA write(fdw, (void *) data_in, sizeof(data_in)); // stream data "data_out" from FPGA read (fdr, (void *) &data_out, sizeof(data_out)); return &data_out; }</pre>	<p>DMA channels</p> <p>Stream PS → PL</p> <p>Stream PL → PS</p>
--	---

Generated sw stub for the server (PS)

RPC extension to multiple dimensions

- HLS optimizations operate on multidimensional arrays
- ONC-RPC supports only one-dimensional arrays

RPC protocol file	RPCGEN generates	RPCFPGA generates
T array<DIM>;	struct { int array_len; T *array_val; } array;	struct { int array_len; T array_val[DIM]; } array;
T mat<R><C>;	not allowed	struct { int mat_len[2]; T mat_val [R][C]; } mat;

RPCFPGA handling of variable length multidimensional arrays

Results

- Case study: dense NxN square matrix multiplication
- Zynq 7020 SoC Zedboard ARM (PS) + FPGA (PL)

PL performance

- Available BlockRAM limits matrix size to N=128
- Maximum from HLS report: 5.08 GFLOPS (only PL)
- Measured on PetaLinux: 2.09 GFLOPS (includes driver overhead)

Block matrix computation

PS+PL algorithm

- Increases matrix size up to N=4096 on Zedboard
- Block matrix multiply on PL (blocks of 128x128)
- Summation of blocks on PS

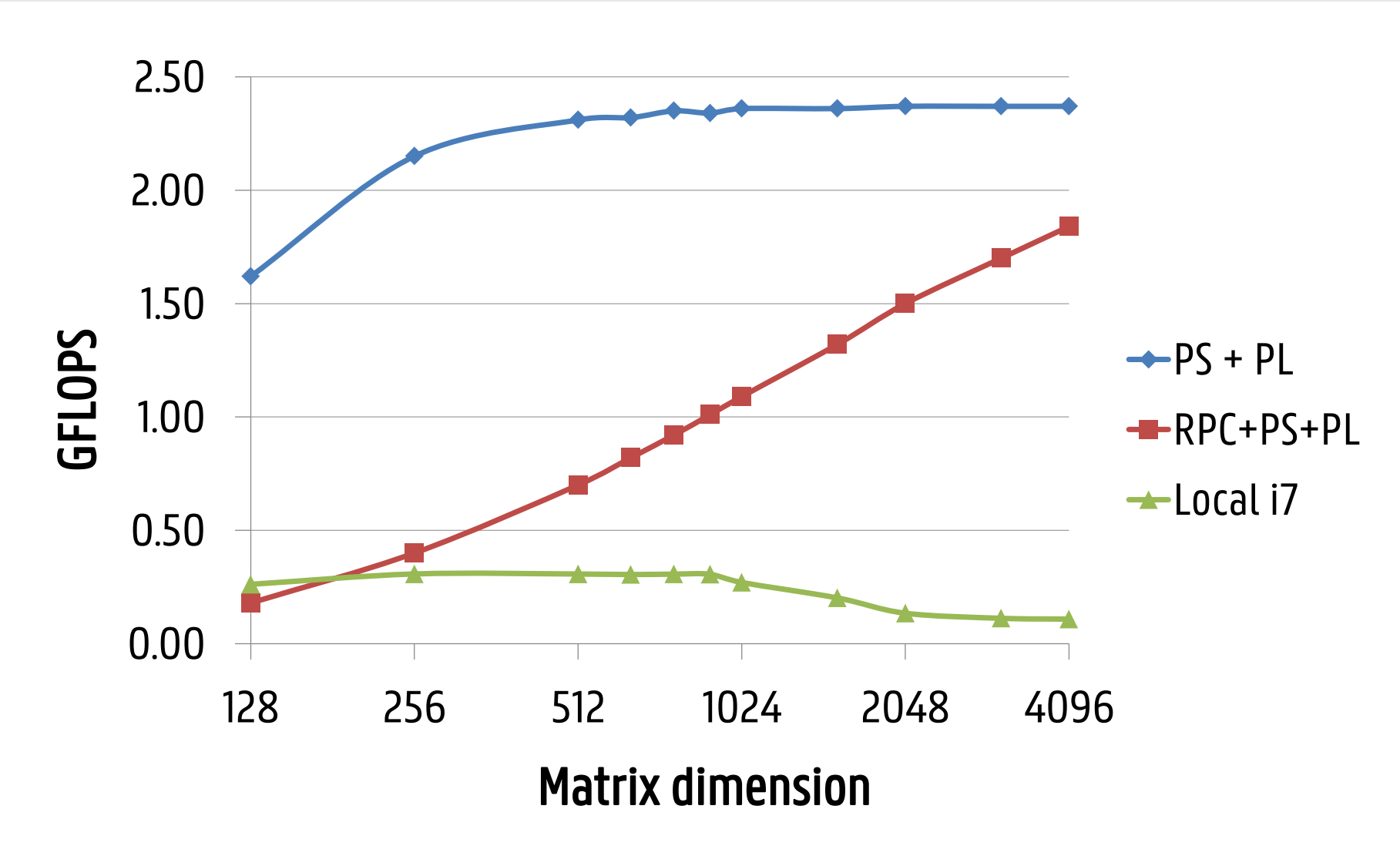
$$C_{ij} = \sum_{k=0}^{m-1} A_{ik} \cdot B_{kj}$$

Block matrix CPU-FPGA computation

- Uses the ARM processor under PetaLinux
- PS-PL block communication is minimized by Gray code ordering
- Performance = 1.62 GFLOPS for N= 128 (PS+PL)
- Performance = 2.37 GFLOPS for N=4096 (PS+PL)

Speedup comparison

- PS+PL : execution on ARM+FPGA under PetaLinux
- RPC+PS+PL : execution on network using RPC call
- Local i7 : execution on i7@2.93GHz processor



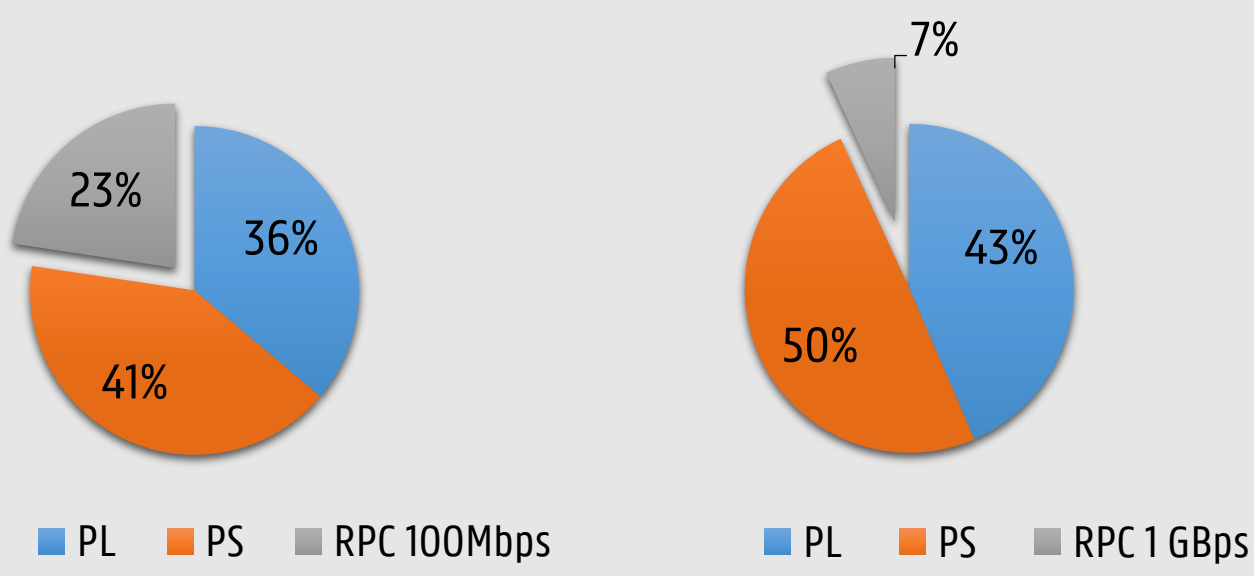
Performance analysis

PL = block matrix multiplication in FPGA

PS = sending and receiving block matrices between PS and FPGA;
summation of blocks matrices in PS

RPC = sending and receiving matrix between network client and server

Performance	100 Mbps	1 Gbps
N = 128	0.18 GFLOPS	0.21 GFLOPS
N = 4096	1.84 GFLOPS	2.21 GFLOPS



Impact of network bandwidth @ N=4096

References

- [1] Oracle, “ONC+RPC Developer’s Guide,” Oracle, 2016.
- [2] Xilinx, “PetaLinux Tools Documentation: Reference Guide,” 2016.
- [3] Erik D’Hollander, Bruno Chevalier and Koen De Bosschere “Calling hardware procedures in a reconfigurable accelerator using RPC - FPGA”, Proc. Field Programmable Technology, FPT 2017
- [4] Erik.D’Hollander, “High-Level Synthesis Optimization for Blocked Floating-Point Matrix Multiplication,” ACM SIGARCH Computer Architecture News, vol. 44, no. 4, pp. 74–79, 2016.