VINETALK - SIMPLIFYING SOFTWARE ACCESS AND SHARING OF FPGAS IN DATACENTERS

S. Mavridis, M. Pavlidakis, I. Stamoulias, C. Kozanitis, N. Chrysos, C. Kachris, D. Soudris, A. Bilas

CHALLENGES OF ACCELERATORS IN DATACENTERS

The recent widespread of FPGAs in datacenters brings a number of challenges to both developers and cloud providers.

- The high programming effort.
- Interfacing IP cores to software applications.
- Lack of sharing mechanisms in FPGAs

VINETALK

A software layer between FPGAs and Apps that reduces the complexity of communication and allows sharing of FPGA IP's cores. We envision to split FPGA accelerated apps to: :

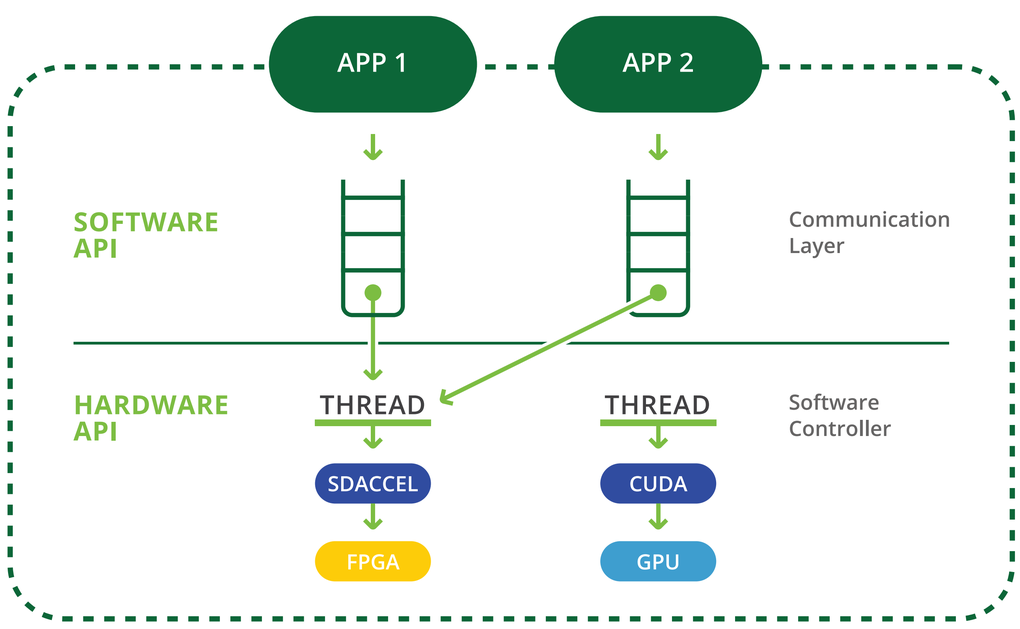
1. Highly optimized IP cores, developed by domain experts. (written in OpenCL, VHDL etc)
2. Compute intensive applications that will use these routines developed by high level developers. *(e.g. Image classification, Speech recognition etc*)

VineTalk User Code	SDAccel	VineTalk Hardware Code
	// User buffers float *input = (float *) malloc(input_size_bytes); float *output = (float *) malloc(output_size_bytes);	
vine_talk_init(); vine_tq tq = vine_tq_get(FPGA); vine_kernel krnl = vine_kernel_get("Kernel");	**// Accelerator specific initialization** const char *target_vendor = "Xilinx"; const char *target_device_name = "xilinx:adm-pcie-ku3:1ddr:3.0"; const char *target_kernel = "Kernel"; const char* xclbinFilename = "Kernel.xclbin"; *world = xcl_world_single(CL_DEVICE_TYPE_ACCELERATOR, target_vendor,target_device_name); *krnl = xcl_import_binary(*world, xclbinFilename, target_kernel);	
vine_buffer_s vine_input = VINE_BUFFER(&input, input_size_bytes); vine_buffer_s vine_output = VINE_BUFFER(&output, output_size_bytes);	**// Accelerator buffer allocation** cl_mem sda_input = xcl_malloc(*world, CL_MEM_READ_ONLY, input_size_bytes); cl_mem sda_output = xcl_malloc(*world, CL_MEM_WRITE_ONLY, output_size_bytes); // Input buffer copy xcl_memcpy_to_device(world,sda_input,input,input_size_bytes); clSetKernelArg(krnl, 0, sizeof(cl_mem), &sda_input);	VT2Accel(task);
vine_task task = vine_task_issue(tq,krnl,input,output);	**// kernel execution** xcl_run_kernel3d(world, krnl, 1, 1, 1);	xcl_run_kernel3d(world, krnl, 1, 1, 1);
vine_task_wait(task);	**// Copy of results - will block until execution completes** xcl_memcpy_from_device(world, output, sda_output, output_size_bytes); **// Free Accelerator memory** clReleaseMemObject(bufs->spot); clReleaseMemObject(bufs->strike);	Accel2VT(task);
vine_talk_exit();	**// Undo Accelerator initialization** clReleaseKernel(*krnl); xcl_release_world(*world);	
	// Free user buffers free(input); free(output);	

- High level developer has **to know only VineTalk**
- **Coding effort:** 30% fewer lines of code + semantically simpler routines.

VINETALK DESIGN

The Vinetalk design for a single server setup.



manospavl@ics.forth.gr

SOFTWARE-FACING API

It replaces the multitude of all platform-specific acceleration APIs, by providing functions that handle memory management and data and task transfers between apps and accelerators.

COMMUNICATION LAYER

It implements and manage virtual accelerators. Consequently, it provides accelerator virtualization.

SOFTWARE CONTROLLER

It is a process that controls all accesses to the underlying hardware. Moreover, it enables FPGA sharing across multiple apps.

HARDWARE-FACING API

It allows hardware designers to incorporate new kernels by using only two simple functions. For FPGAs this API is implemented in OpenCL and SDAccel.

INTEGRATION WITH SDACCEL

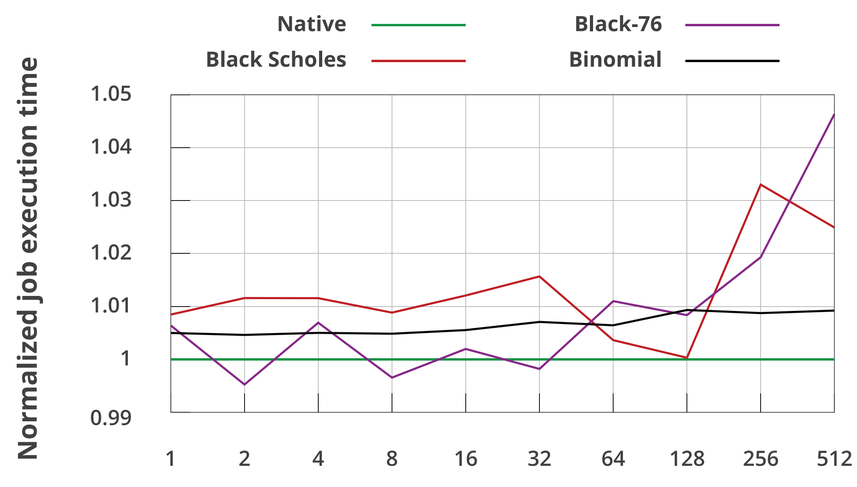
VineTalk intervenes between the application software side and the hardware side of SDAccel.

- It simplifies the development of applications that use FPGA accelerators.
- Any SDAccel compatible kernel can be used by applications using VineTalk, with no hardware dependencies.

EVALUATION

We use three financial applications, Black&Scholes, Black-76, Binomial, for our evaluation.

- **VineTalk overhead:** Between 0.9% and 4%



- **FPGA sharing overhead:** Less than 2%
  - Native setup (without sharing): 1 financial app, 2000 tasks
  - VineTalk setup (with sharing): 2 financial apps, concurrently, 1000 tasks each

ACKNOWLEDGMENTS

We thankfully acknowledge the support of the European Commission under the Horizon 2020 Framework Programme for Research and Innovation through the VINEYARD (H2020-ICT-687628) project. And Xilinx University Program for the kind donation of software and hardware platforms.