www.vineyard-h2020.eu

D5.1: Accelerator Deployment Models

DOCUMENT ID	**D5.1**	**CONTRACT START DATE**	**1st FEBRUARY 2016**
DUE DATE	**31/01/2017**	**CONTRACT DURATION**	**36 Months**
DELIVERY DATE	**30/01/2017**		
CLASIFICATION	**Confidential**		
AUTHOR/S			
DOCUMENT VERSION	**2.0**		

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba

European Commission

Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement nº 687628

1 EXECUTIVE SUMMARY

The current data volume and processing requirements in a modern datacenter outgrow the performance increase offered by CPU technology scaling. In light of this trend, datacenter operators have started exploiting application-specific accelerators in order to provide the performance that is needed, without exceeding power consumption constraints. The resulting heterogeneity of resources raises the question of what type of datacenter composition is most useful and under what circumstances.

In this deliverable, we explore this question by studying accelerator deployment models. Under accelerator, we understand for example application-specific GPUs or specially programmed FPGAs. A deployment specifies types, amount, and connectivity of accelerators in a datacenter. With these definitions in mind, we created a theoretical model of the datacenter, its components, expected workloads, and finally, its possible deployments.

We have developed VineSim, a software simulator of a datacenter, based on the aforementioned theoretical modeling. VineSim takes as inputs a workload and a deployment description and outputs performance metrics of interest, such as job latency and resource utilization. In VineSim, one can configure several parameters, including how tasks are allocated to nodes, and estimations of how fast they execute on different accelerators. VineSim can be used to explore how different deployments respond to different kinds of workloads, thus allowing one to determine how to best compose a datacenter based on particular workload, performance, or budgeting requirements.

The present deliverable includes indicative uses of VineSim that explore the performance of different datacenter design options. Our main results are the following:

- Certain workloads can benefit dramatically from the presence of suitable accelerators.
- The overhead incurred by data transfers plays an increasingly important role as the performance of accelerators increases.
- In order to reap the benefits of accelerators, it is important to have heterogeneity-aware schedulers that can assign tasks to the most appropriate accelerators available.
- Optimizing scheduling for data locality may conflict with heterogeneity-awareness.
- For scale-out task-parallel jobs it is important to have as many accelerators available as the number of concurrent tasks.
- The best choice of deployment is heavily workload-specific.

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba

European Commission
Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

CONTRIBUTORS

Name	Organization
Eleni Kanellou, Nikolaos Chrysos, Angelos Bilas	FORTH
Christoforos Kachris	ICCS

PEER REVIEWERS

Name	Organization
Christoforos Kachris	ICCS

REVISION HISTORY

Version	Date	Author (Organization)	Modifications
0.1	01/12/2016	N. Chrysos (FORTH)	Initial version, Sections, structures
0.8	23/01/2017	Eleni Kanellou,N. Chrysos, Angelos Bilas (FORTH)	All sections but Section 4
1.0	31/01/2017	Eleni Kanellou, N. Chrysos, Angelos Bilas (FORTH), Christoforos Kachris (ICCS)	Included Section 4, internal reviews incorporated

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba

European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

Table of Contents

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement nº 687628

Table of Figures

Table of Tables

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

(Page intentionally blank)

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba

European Commission
Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement nº 687628

2 Introduction

Datacenters are the computers of the World Wide Web and act as the server-side of the current internet ofthings (IoT). Their architecture is currently changing in response to the new problems they are called to address. These machines serve traditional world-wide-web problems, such as web search or advertisement, but are also used in order to serve emerging applications, for example, artificial intelligence acting on big-data used for image recognition and speech processing[1]. Another type of emerging services is cloud computing, which mainly consists of moving enterprise-side functions to the cloud. Apart from commercial applications, such as the aforementioned ones, datacenters can also be of use in scientific or engineering applications, in order to process large amounts of data quickly and efficiently, i.e. in high-performance computing (HPC).

Traditionally, datacenters consisted of clusters of general-purpose processors. With the end of Dennard scaling, the evolution of server architectures cannot benefit from technology improvements alone, as it did in the past. On the other hand, the industry of datacenters expects the previously sustained performance increase to keep on the same pace. Therefore, emerging applications, with their particular characteristics and needs, require new architectures in order to be efficiently dealt with.

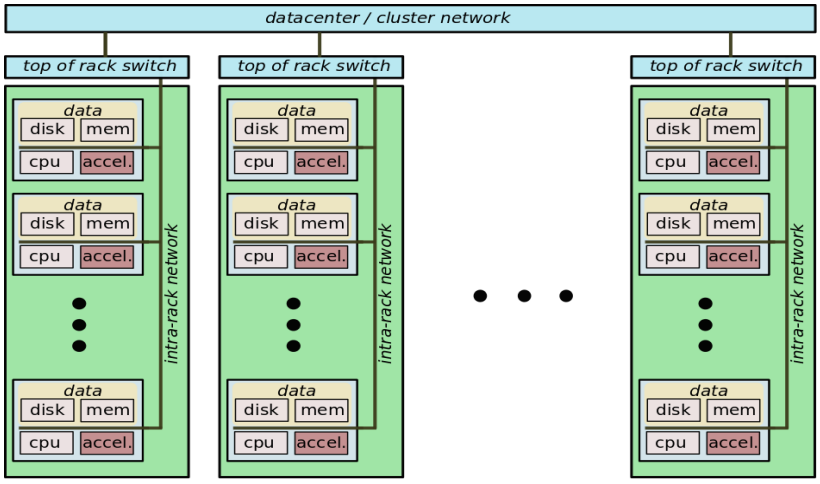
As we can no longer rely on general-purpose processors, the industry currently explores application-specific accelerators, such as GPUs, FPGAs, etc. For instance, while moving to petascale supercomputing was feasible with general-purpose processors, going to exascale architectures is not possible without significant use of accelerators. These technology directions imply that future datacenters will increasingly rely on accelerators.

The heterogeneity that results from including accelerators opens new unexplored avenues in datacenter architecture. Figure 1 shows examples of architecture paradigms when integrating accelerators in a datacenter. A typical scenario is depicted in Figure 1(a). There, accelerators are placed close to a CPU server that uses them, as well as close to the data they will need. In Figure 1 (b), accelerators are placed in dedicated shelves or even racks and can be accessed through the network by any CPU in the cluster. Such a deployment can be beneficial in cases where, for example, the datacenter has the possibility of only including a few potent or highly specialized accelerators but has to make them available to various servers. Variations of the aforementioned options provide design diversity that can cover different datacenter needs.

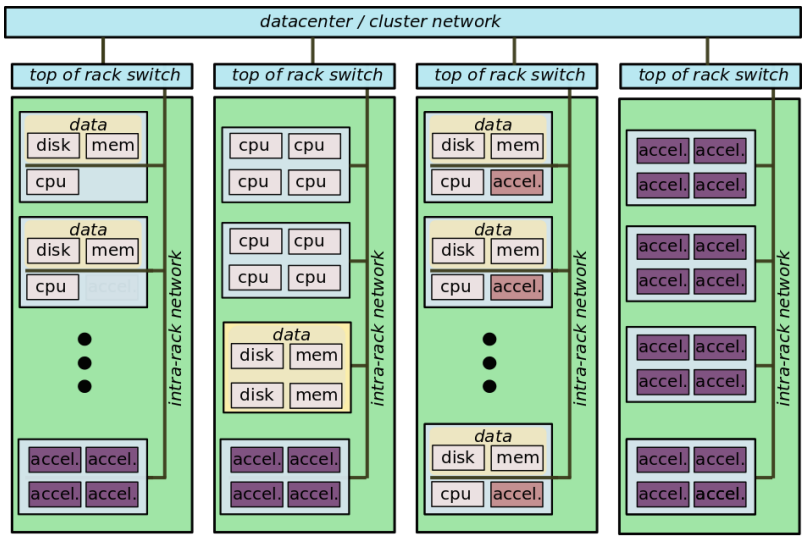
The resulting diversity makes research more challenging. At this time, we need tools to explore the fundamental properties of heterogeneous datacenters. The current deliverable describes this effort. We come up with a model that is used in order to explore alternative deployments. We also create VineSim, a tool for creating and studying these deployments under different workloads. Any such

[1] J. Hauswald *et al.*, "Sirius Implications for Future Warehouse-Scale Computers," in *IEEE Micro*, vol. 36, no. 3, pp. 42-53, May-June 2016.

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628



(a) dedicated accelerators per CPU



(b) accelerators as independent, shared resources in a disaggregated rack

Figure 1: Datacenter deployment paradigms.

study would be incomplete if it ignored the effects of scheduling. If the deployment is the flesh and body of a datacenter, its heart lies in scheduling. Thus, we complete this study by examining scheduling in datacenters.

Regarding alternative deployments, we find that data transfer overhead is a major issue in exploiting the potential of accelerators. Although the networks are getting faster, they cannot reach the throughput of on-chip datapaths of modern data-parallel processors and accelerators. Thus, as accelerator performance increases, the data transfer overhead increases as well.

The following sections of this deliverable delve into several research aspects. Section 3 sets some initial common ground by providing a brief overview of useful terminology. Section 4 extends this by presenting a survey of current use of accelerators in datacenters. Section 5 provides a theoretical modeling of the datacenter and formalizes some key concepts. This modeling is then

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission
Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

used in Section 6 to introduce VineSim, a flexible datacenter simulator. Section uses VineSim to experiment with different datacenter deployments and presents some of the insights gained when experimenting with GPUs and CPUs on real hardware. Section 9 deals with the effects of scheduling and work distribution over possible deployments. Section 10 summarizes the findings and results of this deliverable.

3 Background and Terminology

Following Deliverable D2.2, in modelling a datacenter workload, we adopt the Google paradigm[2] of considering that a workload consists of a set of user-submitted *jobs*, where each job consists of a set of identical *tasks*. (Notice that this might not necessarily be the view of authors that analyze workloads from datacenters other than Google-related.)A task is an element of computational work. It has a type and an associated input and produces an output. The input of a task can be data and/or code. We consider that tasks of a job can be executed in parallel.

We consider that a job arrives to the datacenter in the form of a request by some client application. The execution of a job can either be required to conform to a deadline (in which case the job is referred to as a batch job), or to provide some guarantee in terms of response time (in which case it may be referred to as a user-facing or latency-critical job).Batch jobs generally execute "in the background". Response readiness is not important, as long as the tasks that comprise the job are all completed by a certain deadline. In contrast, latency-critical jobs reflect real-time applications, where a user expects an online response. An example of a batch processing job type is that of MapReduce paradigm[3].

We define a *workload* as the set of jobs that arrive at a datacenter during a specified time period, e.g., an hour, a day, a week, etc. Apart from job types, job number, task per each job, but also, time frame, a workload has further characteristics, such as job inter-arrival time.

4 Accelerators in datacenter: a survey study

In the last couple of years, there are several research efforts towards an efficient framework for the deployment of the FPGAs in the datacenters. This section presents an overview of the most recent framework for the efficient utilization of FPGAs in the datacenters.

---

[2]Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M., & Wilkes, J. (2013). Omega: Flexible, Scalable Schedulers for Large Compute Clusters. *Proceedings of the 8th ACM European Conference on Computer Systems* (pp. 351-364). New York, NY, USA: ACM.

[3]Dean, J. a. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM, 51* (1), 107-113.

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

A general framework for integrating FPGAs into the cloud is proposed by IBM[4]. The framework proposes an accelerator pool (AP) that abstracts FPGA as a consumable resource while avoiding hardware dependencies of current FPGA technologies. In the AP abstraction, each FPGA has several pre-defined accelerators slots in which the hardware accelerators can be mapped. By utilizing the partial reconfiguration mechanism of the FPGAs, each slot can be considered a virtual resource that can be assigned for specific tasks. A cloud tenant can submit either pre-defined hardware accelerators that are hosted in central repository or can submit his own designs. However, in the latter case the cloud owner should perform the synthesis, place and route and generate the bitstreams for the FPGA slots.

To support the AP, a service logic (SL) module is developed and deployed inside the FPGA. The SL provided standard interface for the in-slot accelerators. Furthermore, it facilitates accelerators bandwidth and priority management. In the cloud datacenter, a control node selects the appropriate compute nodes that have the desired FPGA resources for the tenants and a central controller is used for the configuration of the FPGAs.

The proposed framework supports two different methods for translating address between the guest physical address of the virtual machines and the host physical address. The first method copies data between the VM memory and the host buffers. This method (*VM-copy*) is easy to implement but creates a data copy overhead. On the other hand, another method, called *VM-nocopy*, maintains a fixed mapping between the virtual and the host address space. This method does not introduce a data copy overhead but it requires several modifications to the host OS to reserve large blocks of physical memory, and modifications to the memory allocation methods of VM.

A prototype of the framework is implemented on an x86-based Linux-KVM environment with attached FPGAs and deployed in a modified OpenStack cloud environment. Four different accelerators are used for prototyping: Encryption (AES), Hashing (SHA), Stereo matching and Matrix-Vector Multiply. The performance evaluation shows that proposed framework allows the efficient utilization of the FPGA resources by the cloud tenants with less than 4 microseconds latency overhead of the virtualization.

The University of Toronto has proposed a new approach[5]for integrating virtualized FPGA-based hardware resources into cloud computing systems with minimal overhead. The proposed framework allows cloud users to load and utilize hardware accelerators across multiple FPGAs using the same methods as the utilization of Virtual Machines. The reconfigurable resources of the FPGA are offered to the users as a generic cloud resources through OpenStack.

An agent is introduced in this framework that implements the resource management of the OpenStack. In the proposed framework splits the FPA into several reconfigurable regions, each of

[4] F. Chen, Y. Shan, Y. Zhang, Y. Wang, H. Franke, X. Chang, and K. Wang, "Enabling fpgas in the cloud," in Proceedings of the 11th ACM Conference on Computing Frontiers, ser. CF '14. New York, NY, USA: ACM, 2014, pp. 3:1–3:10

[5] S. Byma, J. Steffan, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "Fpgas in the cloud: Booting virtualized hardware accelerators with open-stack," in Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on, May 2014, pp. 109–116

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

which is managed as a single resource. Therefore, instead of a single FPGA bitstream, a collection of partial reconfigurable bitstreams corresponding to the user hardware is passed to the agent. Again, as in the case of the IBM, the cloud provider must generate the partial bitstream for each accelerator and for each partially reconfigurable slot since the current technology requires specific bitstreams for each region of the FPGA.

A static logic module has been implemented that surrounds the virtual reconfigurable resources in the FPGA. The static module is under control of the cloud provider and is used to transfer the data between the host processor and the reconfigurable modules that have been utilized in the FPGA. The static logic module is also used to control the interfaces of the reconfigurable modules and thus to maintain some basic network security.

The proposed system can set up and tear down virtual accelerators in 2.6 seconds on average. The static virtualization hardware on the physical FPGA has a minimum overhead (only three clock cycles). The proposed system is implemented and evaluated in a NetFPGA-10 platform. The prototype system implemented four virtual reconfigurable modules on one device.

IBM Zurich proposes a Hyperscale datacenter framework[6]that allows cloud users to combine multiple FPGAs in the programmable fabric. This allows cloud operators to offer an FPGA to users in a similar way as a standard server.

The FPGA is split into three main parts: the user logic part used for implementing customized applications, the network service layer (NSL), which connects with the datacenter network, and the management layer to run the resource-management tasks. In the proposed framework multiple user applications can be hosted on a single physical FPGA, somehow similar to multiple VMs running on the same hypervisor. Each user can get a partition of the entire user logic and uses it to implement its own applications. This partitioning is achieved by utilizing partial reconfiguration. With partial reconfiguration it is possible to dynamically reconfigure a portion of the FPGA while the rest of the regions remain untouchable.

The network service layer (NSL) is a HW implementation of the physical, data link, network and transport layers and is used in a typical protocol layered architecture. Finally, the management layer (ML) contains a memory manager and a management stack. The memory manager enables access to memory assigned to vFPGAs and the management stack enables the vFPGAs to be remotely managed by a centralized management software.

The user first decides on the required number of vFPGAs and customizes them using their own custom hardware accelerators. The user can then define its fabric topology by connecting those customized vFPGAs. When a request from a cloud user arrives, the accelerator scheduler searches the FPGA pool to find a user logic resource that matches the vFPGA request. When the scheduler matches the request with the required FPGA then the interface for access to the vFPGA is offered to the user. Finally, the user rents the defined virtualized fabric from the IaaS vendor.

[6] J. Weerasinghe, F. Abel, C. Hagleitner, and A. Herkersdorf, "Enabling fpgas in hyperscale data centers," in 2015 IEEE International Conference on Cloud and Big Data Computing (CBDCom 2015), May 2015.

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

The proposed architecture has been integrated into the OpenStack framework and allows the renting of the FPGA resources on the cloud. The same architecture also allows the possibility to distribute their applications on a large number of FPGAs through an FPGA fabric. The integrated framework with multiple FPGAs is compared with a typical datacenter based on commodity processors. It is shown that if each system is based on 2048 module the FPGA-based system can provide 958 TFLOPS compared with the 442 TFLOPS offered by the commodity processors.

The Technical University of Dresden proposed a cloud hypervisor that integrates virtualized FPGA-based hardware accelerators into the cloud environment[7]. The hypervisor allows users to implement and execute their own hardware designs on virtual FPGAs. The hypervisor has access to a database containing all physical and virtual FPGA devices in the cloud system and their allocation status. Each device is assigned to its physical host system (node).

The resource manager tries to minimize the number of active vFPGAs and to maximize the utilization of physical FPGAs in order to reduce energy consumption. The user can allocate wither a complete physical FPGA, which has to be marked separately in the device database or can allocate portion of the vFPGA. In the case of vFPGA allocation the configuration is performed by utilizing partial reconfiguration (PR).

The proposed architecture included also a Reconfigurable Cloud Computing Framework (RC2F), that is used for the realizing the vFPGA concept and allows integration of user cores. The main part of the RC2F framework consists of a controller managing the configuration and the user cores as well as the monitoring of status information. The RC2F is implemented into the FPGA offered to the cloud and the resource utilization is less than 3% for a Virtex 7 XC7VX485T device. The RC2F host API interacts with the RC3E hypervisor and provides access to the user-allocated resources without a direct user interaction with the device files. The proposed framework supports the required security by protecting the device files using access rights. This additional virtualization layer allows concurrent users to interact with their allocated devices without influencing each other.

The proposed framework is prototyped using a matrix multiplication application. The matrix multiplication offers both high amounts of data and computational complexity. The host application starts individual parallel user threads sending matrices to the cores, measures runtime and calculates the throughput. The prototype evaluation demonstrates the utilization of the FPGA resources into a cloud environment in a very efficient way.

Recently, Amazon announced the availability of FPGAs in the Amazon EC1 services[8]. Amazon EC2 F1 is a compute instance with field programmable gate arrays (FPGAs) that the users can program to create custom hardware accelerations for their application. Once the FPGA design is complete, the user can register it as an Amazon FPGA Image (AFI), and deploy it as F1 instance.

Amazon EC2 F1 instances are currently in preview in two different instance sizes that include up to eight FPGAs per instance. F1 instances include the latest 16 nm Xilinx UltraScale Plus FPGA. Each

[7] O. Knodel and R. G. Spallek, "RC3E: provision and management of reconfigurable hardware accelerators in a cloud environment," in 2nd International Workshop on FPGAs for Software Programmers, 2015
[8] Amazon EC2 F1 Instances, https://aws.amazon.com/ec2/instance-types/f1/

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

FPGA includes local 64 GiB DDR4 ECC protected memory, with a dedicated PCIe x16 connection. Each FPGA contains approximately 2.5 million logic elements and approximately 6,800 Digital Signal Processing (DSP) engines. Just like other Amazon EC2 On-Demand Instances, the user pay for F1 compute capacity by the hour with no long-term commitments or upfront payments.

5 Theoretical Model

In order to better study datacenters, we first come up with a theoretical model that describes them. This model then can serve as a common language in expressing their characteristics and formulating observations about their performance.

5.1 General

We consider processing units that have a type and a cost. These processing units can be CPUs, GPUs, FPGAs, or other, as defined by their type. A storage unit is a type of medium, typically a hard disk that allows for the permanence of storage of data, in the absence of purposeful modifications.

A datacenter typically contains rack cabinets of servers and storage. We denote by R the number of rack cabinets in a datacenter. Each rack contains shelves. We denote by S the maximal number of shelves a rack cabinet may contain. We assume the existence of a numbering such that each rack cabinet bears a unique number from 0 to $R - 1$. This is referred to as the rack number. Within each cabinet, we assume that shelves are also numbered, from 0 up to $S - 1$, starting from the bottom-most shelf. This is referred to as the shelf number.

The datacenter hosts N processing units and D storage units. We consider that processing and storage units are affixed inside the datacenter racks. Therefore, each storage unit or processing unit u in the datacenter is further described by a pair of coordinates (rn_u, sn_u). In those coordinates, rn_u indicates the rack number of the rack that the unit is mounted on, while sn_u indicates the shelf number.

Let a and b be two processing or storage units in a datacenter with R racks of up to S shelves. Let (rn_a, sn_a), $0 \leq rn_a \leq R - 1$, $0 \leq sn_a \leq S - 1$ be the coordinates of a and (rn_b, sn_b), $0 \leq rn_b \leq R - 1$, $0 \leq sn_b \leq S - 1$, be the coordinates of b. We define the horizontal distance $h(a, b)$ between a and b as:

$$h(a, b) = | rn_b - rn_a |.$$

Similarly, we define the vertical distance $v(a, b)$ between a and b as:

$$v(a, b) = | sn_b - sn_a |.$$

As communication between two units in a datacenter we understand the exchange or transfer of data or control information. Communication between two units in a datacenter may be either local or remote, in which case it can be intra-rack or inter-rack. Local communication can occur between units that are directly attached to each other, i.e. units that have horizontal and vertical distance equal to 0. We denote the maximal bandwidth that this communication can achieve by B_L.

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

Communication between units located in the same rack, i.e. units that have a horizontal distance equal to *0*, is considered intra-rack. We denote the maximal bandwidth that it can achieve by B_R. Communication between units that are located in different racks, i.e. units that have a horizontal distance other than *0*, is considered inter-rack. We denote the maximal bandwidth that it can achieve by B_I. We assume that:

$B_L > B_R > B_I$.

5.2 Deployment

We define the processor vector of a datacenter as a vector of N elements, $pv = [u_1, u_2, \dots, u_N]$, where each element u_i is a record that contains the attributes of a unique processing unit of the datacenter. Those attributes are the type of the processing unit, the unit's rack number, and its shelf number. The processor vector defines the datacenter's deployment.

5.3 Tasks, Jobs, Workload

A task is an element of computational work. It has a type and an associated input and produces an output. The input of a task can be data and/or code. A job is composed of several tasks. We consider that a job arrives to the datacenter in the form of a request by some client. Ideally, a job will scale out when executed over several processing units of the datacenter. We define a workload as a set W where each element $w \in W$ is a record that contains the attributes of a job, such as the invoking client, the job type, locators for the data required for the job etc.

5.4 Cost functions

Given a set T_t of task types and a set A of processing unit types, we assume the existence of an *affinity mapping*:

$F_a : \{ T_t, A \} \rightarrow [0, \infty)$.

This mapping indicates the throughput that a processing unit of type $a \in A$ can produce when it executes a task of type $t \in T_t$.

We further assume the existence of a power mapping:

$F_p : \{ T_t, A \} \rightarrow [0, \infty)$.

This mapping indicates the power that a processing unit of type $a \in A$ consumes when it executes a task of type $t \in T_t$.

5.5 Task Execution

We consider that a datacenter has an incoming task queue, where tasks of the arriving jobs are enqueued. We consider that tasks of the same job are contiguous in this queue. In order to execute a task, the scheduler must choose an available processing unit for it. It then transfers the task data

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

from their storage unit to the chosen processing unit. The execution time for a task t on a processing unit *p* is given by the inverse of their affinity mapping. The task response time is defined as the sum of the task execution time and the task data transfer. The job latency is the time that transpires between the moment a job arrives at the datacenter, i.e. each of its tasks is enqueued in the task queue, to the moment its last task is completed.

5.6 Utility function

Given a processor vector *pv* and a workload *W*, we consider that a scheduling policy assigns tasks in *W* to available processing units in *pv*. Let S_{pv} be the set of all possible processor vectors, S_w be the set of all possible workloads, and S_{sch} be the set of all possible scheduling policies. A utility function can be used to express the optimality of a configuration described by those three parameters. We define the utility function as:

$U : \{ S_{pv} , S_w , S_{sch} \} \rightarrow [0, \infty).$

We can consider that the utility function is optimized, when its inputs maximize or minimize its output value, depending on the interpretation we give to it. For instance, the utility function can be interpreted as the time that is needed to execute the workload, or the average job latency. One can use the utility function in order to find the best deployment for a given workload and scheduling policy. Alternatively, for a given deployment and workload, one can find the best scheduling policy. Moreover, by combining the above options, one can find both deployment and scheduling policy that optimize the utility function for a given workload.

6 VineSim: a flexible datacenter simulator

As the complexity of datacenter design as well as of expected workloads increases, it becomes crucial to be able to estimate expected trade-offs between cost and performance even before setting up a certain deployment. An analytical model such as that presented in Section 5, can aid in this process. For this purpose, the utility function detailed in Section 5.6 can be used. Specifically, a designer can choose some fixed values for some of the parameters – depending on the needs of the datacenter under design – and then use the other parameters as variables, on which to perform the exploration in order to find how to optimize the utility function.

Apart from analytical models, popular tools for such a purpose are software datacenter simulators, since they provide a cost-effective but also flexible way of doing any necessary predictions and estimations.

In this section, we present VineSim, our custom datacenter simulator. VineSim offers the possibility of testing out the performance of different workloads over different simulated deployments that contain accelerators in different configurations.

In order to serve this purpose, given a workload and a deployment, VineSim calculates and outputs performance metrics such as average and percentile latencies of jobs as well as tasks in the workload, cost of purchase of the given deployment, and utilization of processing units, both in percentages over the entire run of the workload, as well as in a time series. VineSim is further

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

accompanied by tools that facilitate batch execution of experiments, so that e.g. a single workload can be run over several deployments or vice versa.

Apart from offering batch experiment possibilities, VineSim also offers flexibility with respect to its internal configuration. Several characteristics are parameterizable, including but not limited to scheduling policies, accelerator characteristics, job arrival rates, etc. Each run of VineSim accepts configuration of those parameters, offering another dimension of custom configuration, apart from that of the workload and deployment.

6.1 Operation Overview

VineSim implements in software the theoretical concepts that are detailed in Section 5. It is an event-driven simulator, which models the notion of the passage of time by (a) maintaining a global time counter, restarted for each simulation run, (b) assigning time-stamps to each event, and (c) ordering events, based on their time-stamp, with the help of a heap data structure. The backbone of its operation consists in popping an event from the heap, processing it, and possibly, pushing a new event onto the heap. Processing an event advances the simulator's global time to the event's time-stamp.

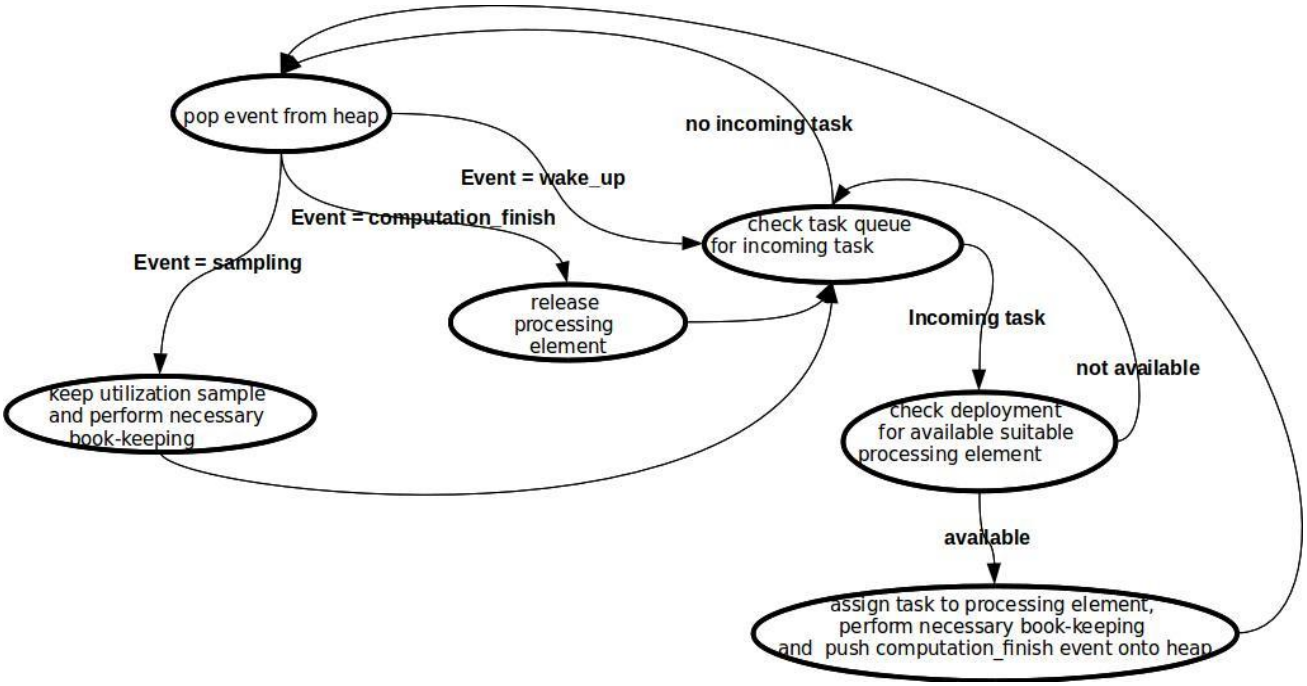


Figure 2: VineSim operation overview diagram.

A rough visual overview of the operation of VineSim is presented in *Figure 2*. Three types of events are recognized by VineSim, namely WAKE_UP, COMPUTATION_FINISH, and SAMPLING. Independently of how the deployment or incoming jobs and tasks are modeled, VineSim pops events from the heap and processes them according to their type, as follows.

- **WAKE_UP**: VineSim checks whether there are incoming tasks. If an incoming task is found, then VineSim attempts to find a suitable processing element to assign the task to. If such an

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

element can be found, then VineSim assigns it the task and performs any necessary book-keeping and computations, in order to calculate the point in simulation time that the task will be completed on that processing element. Using this information, it then pushes onto the heap a COMPUTATION_FINISH event, which has the task termination moment as its time-stamp. All these steps are performed until there are either no more incoming tasks to process or no available processing elements to assign them to. In those cases, VineSim pops a further event from the heap.

- **COMPUTATION_FINISH**: A COMPUTATION_FINISH event models the termination of processing of a specific processing unit P. Therefore, when VineSim processes such an event, it first marks P as idle, i.e. available. Then, before processing any further event, it performs the same steps as for a WAKE_UP event.
- **SAMPLING**: VineSim offers the optional sampling function. If it is switched on, it periodically samples the utilization of the datacenter processing units. This is done by scheduling SAMPLING events at regular time intervals. When such an event is popped from the heap, the utilization of the elapsed sampling interval is calculated and recorded and a new SAMPLING event is scheduled.

6.2 Inputs and Outputs

VineSim simulates the execution of a workload over a specific datacenter configuration. In order to do so, it receives as inputs the workload to execute and the datacenter configuration on which to execute it. The characteristics of a datacenter configuration include the number and types of accelerators that are used in the datacenter, as well as their (relative) location and therefore, their connectivity. The characteristics of a workload, on the other hand, refer to the jobs that have to be executed on the datacenter, i.e. how many tasks are generated by each job, what type of computation these tasks perform, what amount of data each task needs, where the data is located, the time each job arrives at the datacenter, etc.

In order to receive the above information, VineSim is provided with three input parameters, namely a deployment configuration file, a trace description file, and an inter-arrival rate. The deployment configuration file details the amounts and types of the different accelerators that comprise the datacenter and provides information of their location. This is the equivalent of the processor vector, introduced in Section 5. The trace description file contains a sequence of tasks in the order that they are executed at the datacenter, while the inter-arrival rate specifies intervals of arrival of the jobs that these tasks belong to. These two inputs are the equivalent of the workload, introduced in Section 5.

6.2.1 Inputs

Deployment description

The deployment description file is a text file that consists of lines where each line contains 3 fields, separated by space. Each line in a deployment description file details some characteristics of a processing unit. The fields are as follows, in the order they appear in a line.

- type

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

An integer that currently takes values between 0 and 5, boundaries included. Those values correspond to the following abstract processing unit types: CPU (by Intel), CPU (by ARM), GPU, FPGA, Multi-core CPU, and Unspecified processor. Those types are abstract in the sense that they represent a category and do not refer to a specific model. The values that the type can take, can be extended, whenever VineSim has to take into consideration a new processing unit category.

- x coordinate of processing unit

An integer that takes values starting at 0, included. It represents the rack in which the processing unit is located.

- y coordinate of processing unit

An integer that takes values starting at 0. It represents the shelf in which the processing unit is located.

Trace description

The trace description file is a text file that consists of lines where each line contains 7 fields, separated by space. Each line in a trace description file details the characteristics of a task. The fields are as follows, in the order they appear in a line.

- task type

An integer that takes values between 0 and 6, boundaries included. Those values correspond to the following abstract task types: Integer computation (INT), floating-point computation inefficient on GPUs (FP_BAD), floating-point computation efficient on GPUs (FP_GOOD), computation heavy on memory accesses (MEM), computation heavy on I/O accesses (IO), arbitrary computation type (ARB), which is implied to be suitable for custom accelerator implementations on FPGAs, and unspecified task type (UNSPEC_TASK).

- data size

An integer that takes values starting at 0, included. The value represents the amount of data in Bytes that the task operates on.

- x coordinate of data

An integer that takes values starting at 0, included. It represents the rack in which the data required by the task is located.

- y coordinate of data

An integer that takes values starting at 0. It represents the shelf in which the data required by the task is located.

- number of instructions

An integer that takes values starting at 0, included. It represents the number of operations that comprise the particular task. Operations are abstract and are implied to be of the same type as the task.

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

- Preferred accelerator type

An integer that currently takes values between 0 and 5, boundaries included. Those values correspond to the aforementioned abstract processing unit types. VineSim may interpret this field in one of two possible manners: It is either considered to declare the processing unit type on which this particular task executes most efficiently or to declare the only type of processing unit that this particular task can be executed on.

- parent job id

An integer that takes values starting at 0, included. It declares the numeric ID of the job that this particular task belongs to.

6.2.2 Outputs

For the given workload and given deployment, VineSim calculates the following outputs.

Job latency in various percentiles

Given the input workload and specified IAT, VineSim calculates the latency of each job. The results are used in order to produce percentiles of latency over all the jobs in the workload. Commonly, the 99.9%-ile, 99%-ile, and the 50%-ile, i.e. the average latency, are of use to a designer.

Task latency in various percentiles

Same as previous, but calculates latencies of individual tasks instead of jobs.

Processor utilization

Recall that a deployment may contain processing units of several different types. For a given run, VineSim calculates the percentage of time that each individual processing unit was active, i.e. was executing some task. Using these results, VineSim then outputs percentiles of overall processor utilization. Furthermore, VineSim can classify these results into utilization per processing unit type, i.e. accelerator type. Optionally, VineSim can produce utilization as a time series. This means that, for a given run, it outputs the percentage of utilization (both over all processors as well as by accelerator type) at regular intervals of the simulation. The length of interval is a variable parameter that can be changed for each experiment.

Cost of purchase of deployment

Based on the deployment that is specified in the deployment description file, VineSim calculates the monetary cost of purchasing the equipment. This calculation is based on abstract "price lists" that VineSim receives as a parameter. Future extensions of the simulator would aim at also estimating the cost of operation, such as power consumption, for a given workload.

6.3 Parameterization

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

VineSim is designed to be flexible in its mode of operation as well as the production of results. This is achieved by incorporating parameterizable characteristics in its structure and function. The current parameters, their meaning, and their values are detailed below.

6.3.1 Affinity specification

We have defined the affinity of an accelerator type X for a task type Y as the number of operations per time unit that X can perform for a task of type Y. Given that the affinity value for task type Y on accelerator X is A, then if a task of type Y consists of n operations, its computation on accelerator X will finish in (n / A) time units. This corresponds to the affinity mapping function defined in Section 5.

The affinity table is a core instrument of VineSim, critical in speeding up simulation times for large workloads in large datacenter deployments. At the same time, the affinity table allows the simulator to flexibly adapt to new technologies and accelerator capabilities, but also to new workloads. Below we summarize how the user can feed the desired values in the current implementation of VineSim. We further analyze the role of the affinity table in Section 7.

VineSim receives a file that contains the affinity's table values that the user wants to use in a two-dimensional table. Rows are interpreted as accelerator types, while columns are interpreted as task types. Therefore, each element helps calculate the affinity of the accelerator type of its row, for the task type of its column.

Task-type / Accel. type	Integer	Floating point (GPU hostile)	Floating point (GPU friendly)	Memory	I/O	Arbitrary
CPU (Intel)	100000	0.6	0.6	0.1	0.01	1
CPU (ARM)	500	0.1	0.1	0.01	--	1
GPU	1200000	0.01	1	0.001	0.001	0.3
FPGA	1000	0	0	0.001	0.0000000001	1000

Table 1: Indicative affinity matrix used as VineSim parameter.

There are two generic ways to fill in the values in the affinity table. The first one is to profile particular tasks, and find its computation time on different platforms. Then, all one has to do is define these tasks as single-operation tasks, to create a new column for this particular task, and fill in the table the corresponding column with the execution time from the profiling. We used this option in Section 8.4, when we evaluated scale-out version of Neurasmus workloads. We analyze further the role of the affinity table in Section 7. The second way is more abstract, and does not

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

depend on profiling particular tasks. Instead, columns now have a generic type, such integer intensive, memory intensive, etc. Here, we estimate the affinity of different accelerators – task-type pairs using the peak INT FLOPS performance of the accelerator, times a slowdown factor. The affinity values in Table 1 uses this notation, which we further describe below. Note that in our simulation, we use a subset of this matrix with values contained in Table 2.

The first column of the table indicates the affinity that each accelerator type has for INT type tasks. The values in this column are integers, expressing the maximum FLOPS performance of different accelerators, as advertised by the manufacturer of the device or the IP. The Intel CPU is assumed to deliver 100 GFLOPs on integer instructions, i.e. levels reachable by current, multi-core, multi-issue CPUs. For comparison, a low-power single-core, single-issue ARM core is assumed to deliver half a GFLOPS, whereas a GPU 1.2 TFLOPS.

For task types other than INT, affinity is calculated as follows. Consider an accelerator of type X and a task of type Y and let x and y be the integers that correspond to the accelerator type's row and the task type's column, respectively, further assuming that $y \neq 1$. Let I be the affinity of accelerator type X for tasks of type INT and let Z be the value of element (x, y) in the table. The Z value is treated as the slowdown factor on the peak INT performance expected of this accelerator on the particular task type. Therefore, the affinity of X for tasks of type Y is given as I*Z. Given this definition, it follows that the values contained in columns other than the first one of the table must be positive but may be decimal. For example, an Intel CPU is estimated as delivering 10% of its peak INT performance when executing a memory intensive application, i.e. an affinity value of 10000 operations per time unit.

6.3.2 Scheduling policy

VineSim assigns incoming tasks to available processing units. The manner in which this can be achieved is manifold and it is specified by the present parameter. The tool assumes that processing units are ordered by incrementing order of their (x, y)-coordinates, although whether this influences the scheduling depends on the policy. Scheduling policies implemented by VineSim are the following.

- FLAT: the processing units in the deployment are parsed in random order and the incoming task is assigned to the first accelerator that is not busy. This scheme assumes that any task can be executed on any processing units.
- HIGH: the processing units in the deployment are parsed and among those that are not busy, one with the highest affinity to the incoming task type is chosen. This scheme also assumes that any task can be executed on any processing units.
- PREF: the processing units in the deployment are parsed and among those that are not busy and that are compatible with the preferred type specified by the incoming task is chosen.
- CLOSER_TO_DATA: the processing units in the deployment are parsed and among those that are not busy, the one that is closest to task data is selected.

6.3.3 Utilization Sampling

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

VineSim offers the possibility of producing the utilization of the processing units in the deployment as a time series. In order to do so, the tool samples the utilization at regular time intervals during a simulation run. The present parameter acts both as a switch to turn this option on or off, as well as a value that indicates the sampling interval. More specifically, if the value of this parameter is set to -1, then sampling is disabled. However, VineSim as a time interval in microseconds, which is then used as sampling interval, interprets any other positive integer value.

6.4 Overall Function

VineSim produces the aforementioned outputs by performing calculations that are executed by several building blocks, which are detailed in this section. The overall structure of VineSim is sketched out in Figure 3. VineSim is written in C++ and is used by a scripting infrastructure in order to run experiments in batches.

6.4.1 VineSim classes and data types

The software that makes up VineSim relies on a multitude of classes and data types for its operation. In the following, we detail those that directly relate to the theoretical concepts introduced in Section 5. The essential notions in modeling a datacenter, namely processing elements, jobs, and tasks, are represented in VineSim by the following types.

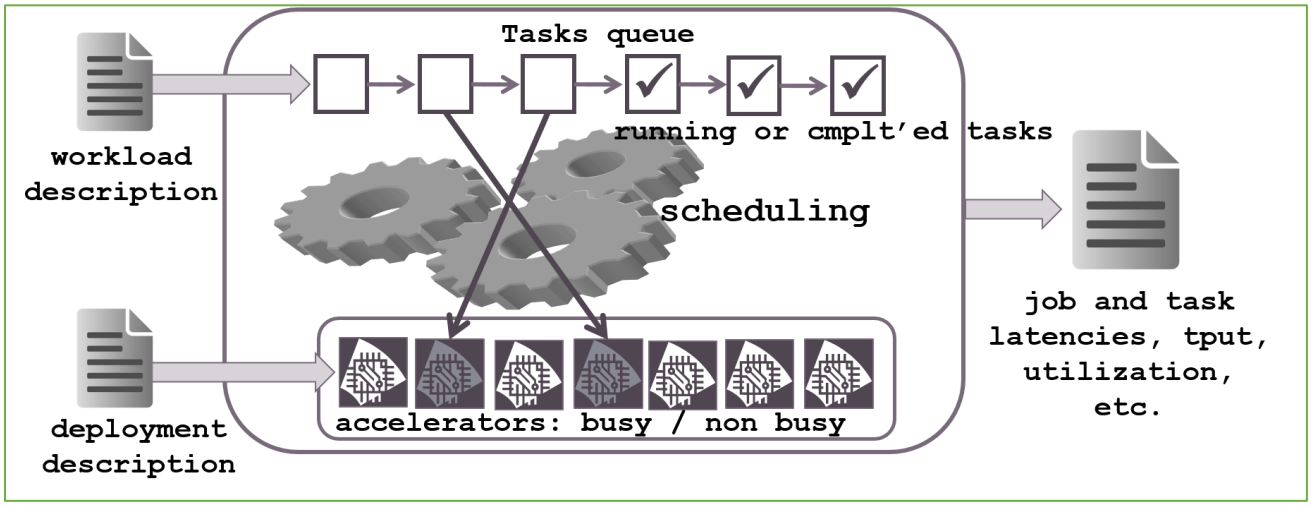


Figure 3: Structural overview of VineSim.

VineProcessor

`VineProcessor` is a class that serves for the implementation of processing units that comprise the datacenter. Important fields of the class include the following.

- `proc_type`: A field of type `ProcessorType`, which is a custom type of VineSim that contains all processing units types recognized by the simulator (see "processing unit types"

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

in Section 6.2.1). For an instance of **VineProcessor**, this field is set at the beginning of the simulation and remains unchanged.

- **proc_coordinates**: A field of type **Coordinates**, which is a custom type of VineSim that specifies the x- and y-coordinates of an instance of the **VineProcessor** class (see "x coordinate of accelerator" and "y-coordinate of accelerator" in Section 6.2.1). For an instance of **VineProcessor**, this field is set at the beginning of the simulation and remains unchanged.
- **cost_of_purchase**: An integer field that represents the monetary cost of an instance of **VineProcessor**. It is set in accordance with field **proc_type** in the beginning of the simulation and remains unchanged, as well.
- **is_busy**: A boolean field that indicates whether the instance of **VineProcessor** is currently busy executing a task or available for taking on an incoming task.
- **t_active**: A field of type **Time**, which is a custom type of VineSim that represents time in us. **t_active** is a running counter, which sums the amount of time an instance of **VineProcessor** is busy during a simulation run. It is used in order to calculate processor utilization at the end of a run.
- **interval_active**: A field of type **Time**. As **t_active**, this is also a running counter which sums the amount of time an instance of **VineProcessor** is active. However, it is not used for the entirety of a simulation run. Instead, it is used only when the sampling option is activated and a sampling interval is indicated in the VineSim parameters. In this case, it is used to sum the running total of the time an instance of **VineProcessor** is active during a sampling interval.

VineJob

VineJob is a class that is used for the representation of jobs arriving at a datacenter. Important fields of the class include the following.

- **job_id**: An integer field that receives a value unique for each instance of a simulation run, just for the purpose of distinguishing different job instances from one another. For an instance of **VineJob**, this field is set once and then remains unchanged during a simulation run.
- **type**: A field of type **TaskType**, which is a custom type of VineSim that contains all task types recognized by the simulator (see "task types" in Section 6.2.1). For an instance of **VineJob**, this field is set once and then remains unchanged during a simulation run.
- **number_of_tasks**: An integer field that indicates the number of tasks that an instance of **VineJob** contains. For an instance of **VineJob**, this field is set once and then remains unchanged during a simulation run.
- **arrival_time**: A field of type **Time** that represents the time point during a simulation run that an instance of **VineJob** arrives at the datacenter. For an instance of VineJob, this field is set once and then remains unchanged during a simulation run.
- **start_time**: A field of type **Time** that represents the time point during a simulation run that an instance of **VineJob** starts being executed by accelerators in the datacenter. Notice that this corresponds to the moment the first task of the job starts being executed by an accelerator. For an instance of **VineJob**, this field is set once and then remains unchanged during a simulation run.

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

- `finish_time`: A field of type `Time` that represents the time point during a simulation run that an instance of `VineJob` terminates its execution in the datacenter. Notice that this corresponds to the moment the last active task in the job terminates. For an instance of `VineJob`, this field is set once and then remains unchanged during a simulation run.
- `executed_tasks`: An integer field that is used to count the running total of tasks of an instance of `VineJob` that have finished executing. The instance of `VineJob` is considered terminated when `number_of_tasks = executed_tasks`.

VineTask

`VineTask` is a class that is used for the representation of tasks that comprise the jobs arriving at a datacenter. Important fields of the class include the following.

- `type`: A field of type `TaskType` (cf. with corresponding `VineJob` field). For an instance of `VineTask`, this field is set at the beginning of the simulation and then is not modified. Notice that all `VineTask` instances that belong to the same job must have the same type as each other and as the `VineJob` instance they correspond to.
- `assigned`: A Boolean field that indicates whether an instance of `VineTask` has been assigned to an accelerator at any given moment of a simulation run.
- `data_size`: An integer field that indicates the size of data in Bytes that the instance of `VineTask` is supposed to operate on.
- `data_coordinates`: A field of type `Coordinates` (cf. with corresponding `VineProcessor` field), which, for an instance of `VineTask`, indicates the location of the data that the task has to operate on.
- `number_of_operations`: An integer field which states the number of operations that an instance of `VineTask` is made up of. This field is set at the beginning of the simulation and then is not modified.
- `preferred_acc_type`: A field of type `ProcessorType` (cf. with corresponding `VineProcessor` field), which indicates the preferred accelerator type of an instance of `VineTask`. This field is set at the beginning of the simulation and then is not modified.
- `parent_job_id`: An integer field which states the id of the `VineJob` instance that a `VineTask` instance belongs to. This field is set at the beginning of the simulation and then is not modified.
- `generation_time`: A field of type `Time`, which is set once in a simulation run for an instance of `VineTask` and which indicates in us the moment of the simulation when the instance was generated by its parent job.
- `task_init_time`: A field of type `Time`, which is set once in a simulation run for an instance of `VineTask` and which indicates in us the moment of the simulation when the instance is assigned to an accelerator.
- `start_time`: A field of type `Time`, which is set once in a simulation run for an instance of `VineTask` and which indicates in us the moment of the simulation when the instance starts executing on the accelerator it is assigned to.
- `finish_time`: A field of type `Time`, which is set once in a simulation run for an instance of `VineTask` and which indicates in us the moment of the simulation when the instance finishes executing on the accelerator it is assigned to.

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

6.4.2 VineSim Data Structures and Objects

In order to simplify the presentation of VineSim, we detail the main data structures necessary for its function, omitting secondary or auxiliary ones that serve the purpose of software engineering. The essential building blocks of VineSim, as also indicated by Figure 3, are the following.

Processor Vector

This object is a vector that consists of elements of type **`VineProcessor`**. At startup, VineSim parses the deployment description file and uses each line in order to create an element in the processor vector. The processor vector represents the datacenter deployment.

Task Queue

This object is another vector, optionally used by VineSim as queue, which consists of elements of type **`VineTask`**. At startup, VineSim parses the trace description file and uses each line in order to create an element in the task queue. The task queue encodes the workload trace of a given simulation run.

Time Machine

VineSim implements event-based simulation. For this purpose, an events heap, referred to as time machine, is used by the simulator. During the course of a simulation run, events are pushed to and popped from the time machine. Each event has a time-stamp, specifying the moment in the simulation that it must take place. The time-stamp is used for the ordering of events in the heap. When an event is popped from the heap, it advances the time of the simulation to its time-stamp. Event types include:

- Task arrival: A task is removed from the task queue and assigned to an accelerator. From this moment on, the accelerator is marked as busy (**`is_busy`** field of **`VineProcessor`** class) and the task is marked as assigned (**`assigned`** field of VineTask class).
- Computation finish: Whenever a scheduling event is processes, the corresponding freeing event is created and pushed into the time machine. When such an event is processes, the processing element that it refers to is marked as idle (**`is_busy`** field of **`VineProcessor`** class).
- Sample utilization: If the sampling option is switched on in the parameters of VineSim, then such events are pushed onto the time machine in the specified sampling intervals. When such an event is processed, it calculates the utilization of the datacenter in the sampling interval that just elapsed.

6.4.3 VineSim Operation and Timing

At startup, the trace description file and deployment description file are read by VineSim and are used to create the processor vector and task queue, before initializing the time machine. Events are scheduled and later processed based on the order imposed by their time-stamps.

At the core of the simulator lies the assigning of tasks to processing elements. This is done by following the parameter-specified scheduling policy (cf. Section 6.3). Once an available processing element has been selected for an incoming task, VineSim can calculate the duration of the task, by

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

using the number of operations specified by the task and the affinity specified for the selected processing element. It then schedules an even to free the processing element at the point in time the task is calculated to have finished.

In order to perform the necessary calculations, VineSim uses its parameters and the class attributes of `VineTask`, `VineJob`, and `VineProcessor` as follows. Let `A` be the affinity of the available processing element for a task `K` of type `T` that consists of `N` operations. Then, since affinity is defined as number of operations a processing element can execute per time unit (in our case, per microsecond), the time that `K` will take to be completed once it starts being executed will be:

```
t_comp(K) = number_of_operations / A
```

Assume that `J`, the parent job of `K`, arrives at the datacenter at time `t_arr`. Then, for `K` it will hold that `arrival_time = t_arr`. Assume that `K` is the first of `J`'s tasks to be assigned to a processing element and assume that this occurs at time t_asgn. Consequently, for `J` it will hold that `start_time = t_asgn`. For all tasks pertaining to `J`, it will hold that `generation_time = t_asgn`. However, for `K` it will also hold that `generation_time = task_init_time = start_time = t_asgn`, assuming that no data transfer is necessary for K. In case data transfer is necessary, it will hold that `generation_time = task_init_time`, and assuming that the data require `t_trans` time to reach the processing element where `K` is to be executed, then for `K` it will hold that `start_time = task_init_time + t_trans`. For `K`, once its duration on the assigned processing element is calculated, it will hold that:

```
finish_time = start_time + t_comp(K)
```

VineSim then schedules an event with this `finish_time` as time-stamp in the time machine. This event frees the processing element that was executing `K` and marks it as idle, making it once again available to other incoming tasks.

6.5 Related Work

Given the cost and volume of a datacenter, it borders on the impossible to test out different deployment or performance hypotheses in practice. Instead, as posited by the creation of VineSim, datacenter simulators are a viable alternative. However, datacenters are systems with a multitude of possible characteristics to study, a fact which is further exemplified by the variety of aspects that different datacenter simulators focus on.

Several tools focus on modeling and simulating datacenter power consumption. Some of them are concerned with moderating datacenter power consumption by examining physical characteristics of a datacenter, such as cooling and airflow. CoolSim[9] and TileFlow[10] are two such tools. Other simulators are merely concerned with modeling the behavior of a datacenter network, as is the case

[9] http://www.coolsimsoftware.com/
[10] http://tileflow.com/

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

with DCNsim[11] or DIABLO[12]. However, our concern is to determine trends regarding resource utilization and job latency.

Seeing as accelerator integration and resource heterogeneity are relatively emerging practices, several previous simulators, such as BigHouse[13], attempt to answer such performance or utilization questions without being concerned with GPUs or FPGAs as resources. Specifically, BigHouse performs discrete-event queuing simulation, using stochastic models to approximate workload behavior. MDCSim[14], another simulation tool, models the datacenter as a three-tier system, roughly considering that requests that arrive at a datacenter go through a user-level (which models the services made available by the datacenter), kernel level (which includes scheduling algorithms), and communication level (which models interconnect characteristics). This tool, however, also does not explicitly deal with heterogeneity in hardware resources and in how different deployment choices affect performance and resource utilization.

A tool which does deal with heterogeneity, albeit in the context of cloud computing and IaaS, is CloudSim[15]. Although CloudSim also does not deal with accelerators explicitly, it offers the possibility of modeling so-called hosts, i.e. entities that correspond to physical processors, with different computational powers. However, CloudSim is mostly concerned with modeling provisioning techniques for virtual machine (VM) allocation, instead of examining the effect on performance that is caused by the inclusion of accelerators in a datacenter deployment.

A simulation tool that does study datacenters in a way that comes quite close to VineSim is CactoSim[16], a datacenter simulator that is designed to take into account the heterogeneity of datacenter hardware and that had been developed under the European 7th Framework project Cactos. Contrary to our custom-build VineSim, CactoSim is designed as an add-on to the Eclipse development framework and builds on Palladio, an existing Software Architecture Simulation tool. Furthermore, CactoSim is designed to integrate with a physical, real-life datacenter and focuses on

---

[11] Nongda Hu, Binzhang Fu, Xiufeng Sui, Long Li, Tao Li, and Lixin Zhang. 2013. DCNSim: a unified and cross-layer computer architecture simulation framework for data center network research. In *Proceedings of the ACM International Conference on Computing Frontiers* (CF '13). ACM, New York, NY, USA, , Article 19 , 9 pages.

[12] Zhangxi Tan, Zhenghao Qian, Xi Chen, Krste Asanovic, and David Patterson. 2015. DIABLO: A Warehouse-Scale Computer Network Simulator using FPGAs. *SIGARCH Comput. Archit. News* 43, 1 (March 2015), 207-221.

[13]David Meisner, Junjie Wu, and Thomas F. Wenisch. 2012. BigHouse: A simulation infrastructure for data center systems. In *Proceedings of the 2012 IEEE International Symposium on Performance Analysis of Systems & Software* (ISPASS '12). IEEE Computer Society, Washington, DC, USA, 35-45.

[14]S. H. Lim, B. Sharma, G. Nam, E. K. Kim and C. R. Das, "MDCSim: A multi-tier data center simulation, platform," *2009 IEEE International Conference on Cluster Computing and Workshops*, New Orleans, LA, 2009, pp. 1-9.

[15]Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. 2011. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.* 41, 1 (January 2011), 23-50.

[16] Östberg, P.-O., Groenda, H., Wesner, S., Byrne, J., Nikolopoulos, D. S., Sheridan, C., Krzywda, J., Ali-Eldin, A., Tordsson, J., Elmroth, E., Stier, C., Krogmann, K., Domaschka, J., Hauser, C. B., Byrne, P. J., Svorobej, S., McCollum, B., Papazachos, Z. C., Whigham, D., Ruth, S. & Paurevic, D. (2014). *The CACTOS Vision of Context-Aware Cloud Topology Optimization and Simulation.*. *CloudCom* (p./pp. 26-31), : IEEE Computer Society. ISBN: 978-1-4799-4093-6

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement nº 687628

using so-called live data obtained from datacenter operation, in order to assist in design decisions and equipment management through simulated predictions.

Contrary to this tool, as well as to other tools and methodologies which may focus on specific request types or applications, our aim with VineSim is to come up with a more generic tool, that is not bound to a limited application type, hardware infrastructure, or physical aspect of the datacenter. At the same time, our aim is not to come up with results that are as close to reality as possible (e.g. to accurately model the expected latency of a specific application) but instead, to observe trends in certain output metrics as the datacenter infrastructure changes.

7 Affinity specification for simulation

The concept of affinity was introduced in Section 5. Recall that there, the affinity mapping is defined as a mapping which takes pairs consisting of a task type T and an accelerator type A, and maps them to a number, which represents the throughput that an accelerator of type A can produce when executing a task of type T. In this section, we take a more detailed look on affinity.

The affinity concept is intended as a way of modeling the computational power of an accelerator as well as its suitability for a particular type of computation. We use this approach to modeling because it provides us with an important advantage: *It allows for fast simulation of the execution of a task on an accelerator, since the duration of the task can be calculated by a simple division*. In essence, this avoids the necessity of having to implement both the task and accelerator behavior in simulation software. As mentioned, the affinity of an accelerator for a particular task is a number. We identify two ways of coming up with this number. The configurability of the set of task types, accelerator types, and the affinity among them is one of the characteristics that makes VineSim flexible.

Method 1: affinity numbers can be determined in practice. Assuming that alternative implementations of a task on different accelerators are available, a designated (abstract) task type is assigned to the task of interest, then it is executed on different accelerators. Measurements of the throughput of each accelerator for the task define the accelerator's affinity for it. Using this method, we created the affinity values in Table 3 in order to evaluate scale-out workloads consisting of tasks Neurasmus.

Method 2: Alternatively, a task of type T is considered to consist predominantly of operations of that type. For example, a task of type INT would predominantly contain integer arithmetic, whereas a task of type FP would predominantly contain operations on floating-point numbers, etc. The affinity of an accelerator A for a task of type T expresses how many operations of type T the accelerator is able to carry out in one time unit in one microsecond. Let us now shed more light into how we filled using this method Table 1.

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

- **Task types**: We used INT, FP_BAD, FP_GOOD, MEM, IO, and ARB types[17]. INT, FP_BAD, and FP_GOOD reflect applications that are CPU-intensive, although in different ways. Specifically, the INT task type models applications that mainly rely on integer arithmetic (e.g. some pseudo-random number generators[18]). The FP_* categories model those CPU-intensive applications that contain mostly floating point operations rather than integer arithmetic. Many such applications can be divided into highly parallel components that take advantage of the GPU architecture. This application class (containing e.g. FFT[19] or gray-scale conversion[20]) is modeled by the FP_GOOD task type. Conversely, floating-point-heavy applications, where the computation is however not readily parallelizable, as are for example many graph algorithms, may not perform as well on GPUs and are modeled by the FP_BAD task type. Type MEM models the kind of application that causes high and/or frequent memory traffic. Type IO models the kind of application that causes high and frequent disk I/O accesses.

  The ARB type is an auxiliary task type that is kept abstract, in order to allows us to define application classes that are specifically designed or targeted for a particular type of accelerator.

- **Accelerator types and their affinities:** Arbitrary many classes of accelerators could be defined here as well. In order to cover two of the basic accelerator types, in our simulations, we have modeled the types of Intel CPU and GPU. The Intel-type CPU was chosen as an accelerator class that models multi-core CPUs that offer high performance. The GPU accelerator type is meant to model state-of-the-art GPUs and their performance potential.

  Table 2 contains the affinity matrix that we used in simulations other than for Neurasmus workloads. In order to come up with the values in this table, we have consulted the potentials of the different technologies that the accelerator types are meant to represent. Recall that the affinity of processor type X for task type Y expresses the number of operations of type Y that type X can perform in a microsecond.

Task-type / Accel. type	Integer	Floating point (GPU hostile)	Floating point (GPU friendly)
CPU (Intel)	100000	0.6	0.6
GPU	1200000	0.01	1

Table 2: Indicative affinities used as VineSim parameter.

17 Similar classifications are encountered in works such as: Manu Awasthi, Tameesh Suri, Zvika Guz, Anahita Shayesteh, Mrinmoy Ghosh, and Vijay Balakrishnan. 2015. System-Level Characterization of Datacenter Applications. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering* (ICPE '15).

18 Volodymyr Kindratenko. 2014. *Numerical Computations with Gpus*. Springer Publishing Company, Incorporated.

19 e.g. https://developer.nvidia.com/cufft

20 e.g. https://code.google.com/archive/p/cuda-grayscale/

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

Current multi-core CPUs (Intel's technology is used as indicative example) can reach upwards of 100 Gigaflops (e.g. Intel i5 processor capabilities[21]). Guided by this, we specified an affinity of 100000 INT-type operations per microsecond, i.e. 100 * 10^9 operations per second.

According to manufacturer specifications, state-of-the-art GPUs can reach performances upwards of 2 TFLOPS (e.g. the Tesla series of Nvidia[22] [23]). Opting for a conservative technology modeling, we opted for endowing the GPU type with an affinity of 1200000. This means that, in simulations using these values, a GPU is capable of performing 1200000 operations of type INT per microsecond. This corresponds to 1.2 * 10^{12} (1.2 TFLOPS) INT operations per second, a value that is consistent with current technology, albeit not state-of-the-art accelerators.

As also described in Section 6.3.1, for the calculation of the affinity to other task types, we opt for a scaling factor representation.

For example, following the affinity values specified in Table 2, we have that the affinity of a GPU for tasks of type FP_GOOD is equal to the INT affinity (1200000 ops/usec) multiplied by the factor in cell [GPU / Floating point (GPU friendly)], which is 1. This yields affinity of 1200000 ops/usec (1.2 TFLOPS) also for this type of tasks. On the contrary, the affinity for FP_BAD type tasks is worse, since the multiplication factor in cell [GPU / Floating point (GPU hostile)] is 0.01, yielding an affinity of 12000 ops/usec. Similarly, the affinity of a CPU for tasks of type FP, both friendly and hostile to the GPU, is determined by the factor of 0.6, yielding 60000 ops/usec or 60 GFLOPS.

8 Exploration of indicative datacenters deployment

In this section, we examine the effects that different deployments have on performance and cost of a datacenter. Deployments may vary from one another on several factors, such as the following.

- Percentage of accelerators in the datacenter.
- Homogeneity or heterogeneity in accelerator type.
- Type of accelerator.
- Location of accelerators.

Apart from the hardware composition of a datacenter, another important factor that ends up influencing the deployment is the scheduling policy that is followed when assigning tasks to processing elements. Scheduling is a category in itself, since it can greatly affect whether the full potential of the datacenter can be taken advantage of.

On a secondary level, the different types of deployments described up to here, produce further variations in conjunction with the needs of different workloads. Such workload-induced variation factors include:

- Location of data with respect to where a task is assigned.

21 http://download.intel.com/support/processors/corei5/sb/core_i5-3500_d.pdf
22 http://www.nvidia.com/object/tesla-k80.html
23 http://www.nvidia.com/object/tesla-p100.html

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

- Availability of processing elements as influenced by job inter-arrival time.

In the following, we outline selected experimental evaluations of deployments, performed with VineSim, and formulate conclusions reached through this process.

In order to take advantage of the tool, a datacenter designer would have to have an estimate of one or more of the following.

- The type of workload that the datacenter is expected to service.
- The available budget of purchase and/or of operation.
- The worst performance acceptable for one or more given metrics.
- The best performance required for one or more given metrics.

Based on these parameters, the experimental setup to examine specific questions can be determined. Indicative such uses of VineSim are detailed below.

8.1 Best deployment for given workload mix

In the following example, VineSim is used to explore datacenter composition versus average job latency, for different workloads. Task types that are explored are floating-point that are GPU-affine (FP-GOOD) and floating-point that are not GPU-affine (FP-BAD). Deployments consist of 40 processing units in total, and workloads consist of 500 jobs in total. The job sizes are identical, containing 30 tasks each. Indicatively, we assume 3,000,000 operations in each task. Tasks are assigned to the most suitable accelerator available, depending on their type, i.e. we use a best fit scheduling policy (referred to as "HIGH" in Section 6.3.2). The processing unit affinity matrix that is assumed, is that presented in Table 1.

Figure 4 presents our results. The x-axis shows a varying percentage of GPU content in the 40 processing units of the deployment. The y-axis shows average job latency (in logarithmic scale). Time units are microseconds. The deployments that are explored vary the proportions of GPUs versus CPUs, starting with a CPU-only deployment and increasing the GPU percentage by steps, until ending up with a GPU-only deployment, as follows:

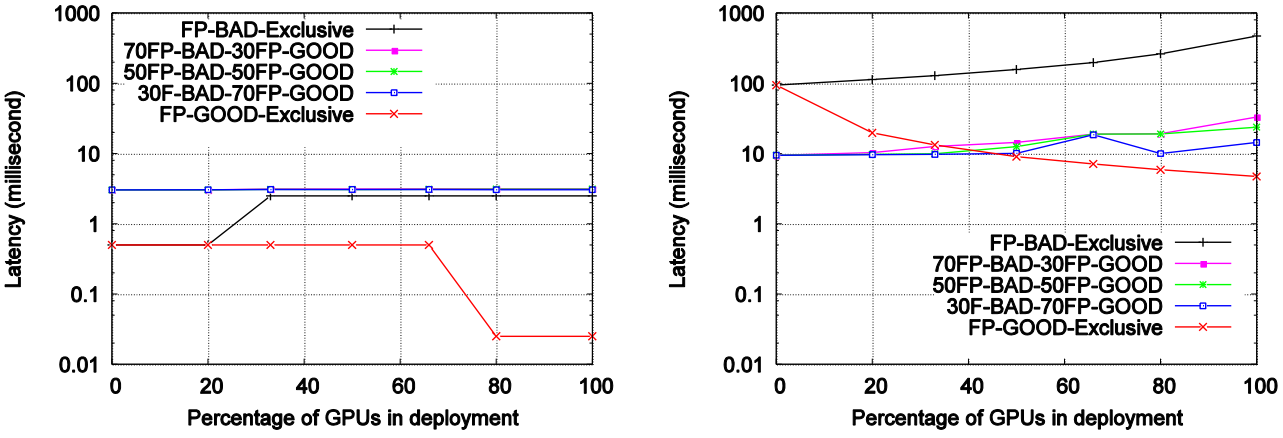
- 0% GPUs,
- 20% GPUs,
- 33% GPUs,
- 50% GPUs,
- 66% GPUs,
- 80% GPUs,
- 100% GPUs.

We test five different workload compositions, labelled in the graphs as follows:

- **FP-BAD-Exclusive**: The only task type this workload contains is FP-BAD.
- **70FP-BAD-30FP-GOOD**: 70% of the tasks in this workload are of type FP-BAD, while the remaining 30% are of type FP-GOOD.
- **50FP-BAD-50FP-GOOD**: 50% of the tasks in this workload are of type FP-BAD and the remaining 50% are of type FP-GOOD.

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

- **30FP-BAD-70FP-GOOD**: 30% of the tasks in this workload are of type FP-BAD, while the remaining 70% are of type FP-GOOD.
- **FP-GOOD-Exclusive**: The only task type this workload contains is FP-GOOD.



(a) Low load *(b) High load*

Figure 4: Average job latency tendencies for deployments of various GPU content.

Figure 4 presents two scenarios: A low load (Figure 4 (a)) and a high load (Figure 4 (b)) situation. In the first scenario, if a job has tasks that have no interdependence, it essentially executes alone in the datacenter, i.e. without interference by other jobs from the same workload. The 30 tasks per job and the fact that each deployment contains 40 processing unit, means that each task of a job can be assigned to a processing unit upon arrival, without having to queue. Thus, they execute in parallel. In deployments that contain a mix of processing unit types, this means that the latency of a job will be that of the task executed on the "slowest" processing unit for that task type, i.e. the processing unit that has less affinity with the task type. If tasks have interdependence, the average job latency may even increase, as tasks of the same job may wait for another before starting execution.

Some of the above is evidenced in Figure 4(a) by the FP-GOOD-Exclusive workload: As implied by the affinity matrix of Table 1, a task in that workload executes 20x faster on GPUs than on CPUs. We consider that tasks of this workload are not interdependent. Even so, as long as a deployment contains less than 30 GPUs, the average job latency is limited to that, that a CPU can provide for this task type. At 80% GPUs in the deployment, however, the average job latency shows an intense drop, because 80% GPUs translates to 32 GPUs in the deployment. Thus, all tasks of a job can be executed on GPUs, reducing the average job latency by a factor of ~20, as expected.

The inverse is the case for the FP-BAD-Exclusive workload, which contains task types that execute 5x faster on CPUs than on GPUs, based on the affinity matrix of Table 1. As soon as the content of CPUs in the deployment drops below 30, as is the case in the 33% GPUs deployment, the latency increases a fivefold.

In Figure 4(b), a high load scenario is presented. There, no inter-arrival time is assumed: tasks must queue before a processing unit becomes available to execute them. For the FP-GOOD-Exclusive workload we observe that the average job latency drops, i.e. performance improves the more GPUs are included in the deployment. This stems from the fact that, the more GPUs are

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

included in the deployment, the more "fast" processing units are available. The queuing time for an individual task decreases, leading to the overall decrease of average job latency for jobs.

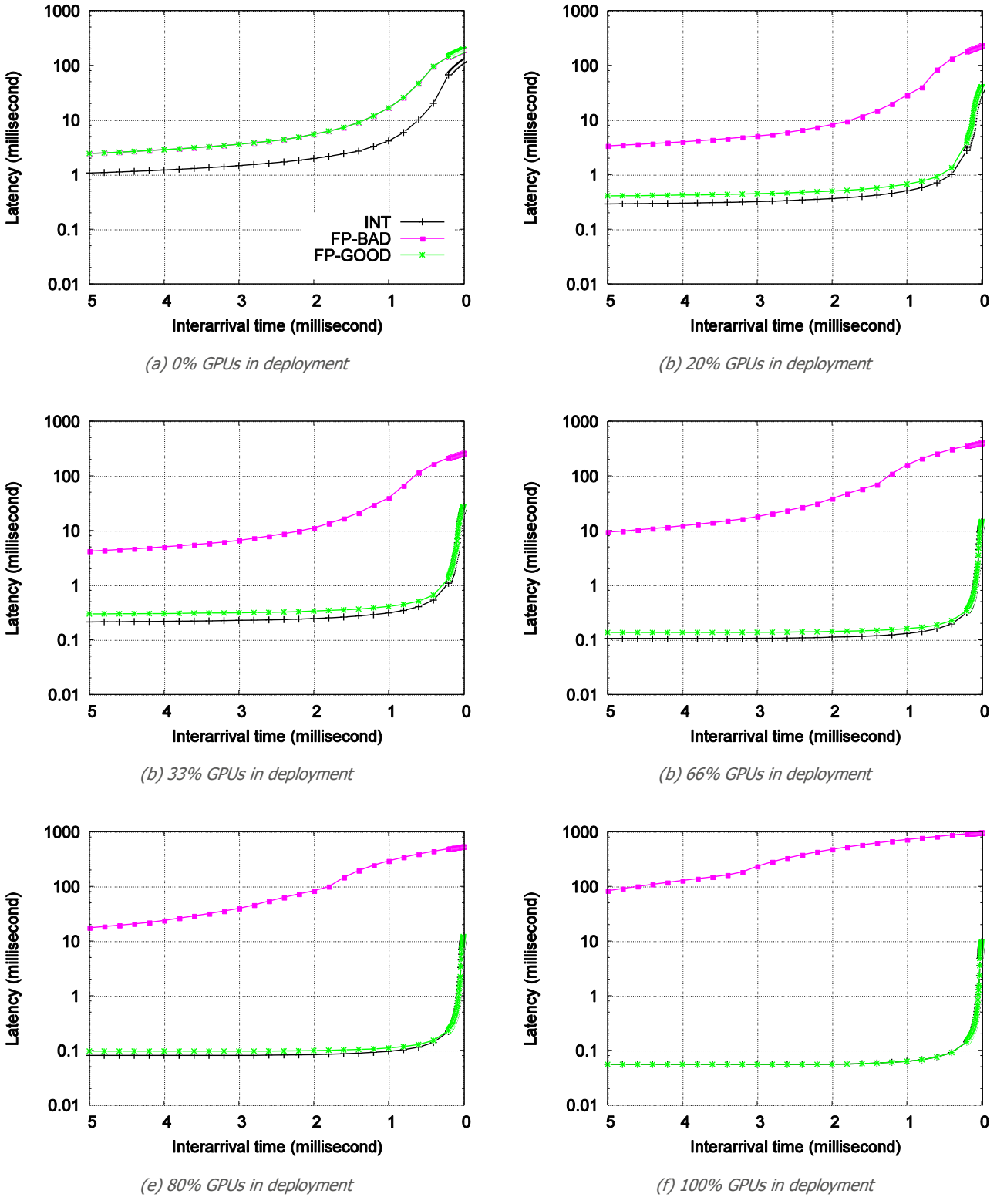
Conversely, for the FP-BAD-Exclusive workload, average job latency increases as the GPU content of the deployment increases, as an individual task may have to queue longer before being assigned to a processing unit. Notice that at high load, the choice of scheduling policy is obliterated by the fact that restricted types of processing unit are available upon job arrival.

For workloads that are between the aforementioned two categories, results in terms of average job latency vary. This can become even more complicated when IAT is not uniform and job sizes vary. A datacenter designer can decide on a specific upper bound on cost of purchase for the datacenter, as well as on an acceptable highest value for average job latency (or other metrics), test representations of the expected workloads on VineSim, and select a datacenter composition based on the tendencies observed in the results.

8.2 Number of accelerators in deployment versus workload type

In a similar vein as in the above simulation run, we run workloads consisting exclusively of one task type on deployments that have different content of GPUs versus CPUs. We test the INT, FP-GOOD, and FP-BAD task types. In this case, we examine the behavior for different values of the IAT input parameter, thus increasing the load that the datacenter has to process. Deployments consist of 40 processing units in total, and workloads consist of 500 jobs in total. The job sizes vary, following a distribution of 90% small jobs (up to 20 tasks) and a remaining 10% of medium-sized and very large jobs, following size distributions observed in real-life datacenter traces (cf. with Deliverable D2.2). We use identical job and task amount and identical traces in what job size and job sequence is concerned and only vary the task type from one trace to another. Indicatively, we assume 30,000,000 operations in each task. Tasks are assigned to the most suitable accelerator available, depending on their type, i.e. we use a best fit scheduling policy (referred to as "HIGH" in Section 6.3.2). The processing unit affinity matrix that is assumed, is that presented in Table 1.

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628



(a) 0% GPUs in deployment

(b) 20% GPUs in deployment

(b) 33% GPUs in deployment

(b) 66% GPUs in deployment

(e) 80% GPUs in deployment

(f) 100% GPUs in deployment

Figure 5: Comparison between different workloads and amount of GPUs in deployment.

Figure 5 plots of the average job latencies resulting from the simulation. The x-axis shows a varying inter-arrival rate for incoming jobs. The y-axis shows average job latency (in logarithmic scale). Time units are microseconds. Figure 5(a) represents workload behavior in a deployment that consists entirely of CPUs. Notice that based on the affinity matrix, a task of type FP-BAD and FP-

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission
Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

GOOD runs 0.6 times slower than a task of type INT. Even though the workload job sizes are mixed, this is reflected on the average job latency plotted in the figure. Notice that, given also that the traces are identical save for the task type, the graphs for the FP-BAD and FP-GOOD workloads coincide. On the other hand, consistently with the affinity matrix, the INT workload shows lower average job latency.

Figure 5 (b), on the other hand, is for a deployment where 20% of the processing units are GPUs. There, the FP-GOOD and FP-BAD workloads no longer show similar behaviors in terms of average job latencies. The 20% increase in more potent accelerators translates to a tenfold decrease in average job latency at low loads for the GPU-affine workloads, i.e. INT and FP-GOOD, while average latency of the FP-BAD workload marginally increases.

When it comes to high loads, i.e. high inter-arrival rates, meaning low inter-arrival times, Figure 5(a) through (f) reveal that going from a deployment with only CPUs to a deployment with only GPUs results in an approximate tenfold performance increase when it comes to average job latency. An exception to that is FP-BAD, i.e. the workload type that is not GPU-affine. Its behavior in Figure 5 (a) through (f) indicates that average job latency is such, that high load, which translates to increased task queuing time, does not affect average job latency as much, possibly because even under low load, queuing time due to the bad performance of the specific task type on any kind of accelerator, approaches the actual task execution time.

In comparison to other workload types, the GPU-hostile workload remains relatively less affected by inclusion of more GPUs in the datacenter, than the other workload types benefit from it. The average job latency increase suffered when the deployment changes from including no GPUs to including all GPUs, is roughly tenfold, compared to the roughly 1000-fold improvement in performance, i.e. drop in latency that the other workload types can show under low load for the same deployment change.

We explore this behavior further by running the experiment for lower loads, in order to approach a 0 load scenario. The result is presented in Figure 6. Assuming a 100% GPU deployment, we confer with the affinity matrix (Table 1), where we observe that, on a GPU, a task of type FP-BAD executes 100x slower than tasks of type INT and FP-GOOD. Thus, at a 0 load scenario, we expect workloads of those types to show 100x lower latency on average, than the FP-BAD workload. This is confirmed by Figure 6, where a 0 load scenario is approximated.

Notice that, when lowering the load, INT and FP-GOOD workloads reach a steady average job latency sooner than the FP-BAD workload. Recall that the job size distribution in our workloads is mixed. However, to interpret the FP-BAD behavior, we calculate the tasks per job average, coming up with 50 tasks per job. Given the deployments of 40 processing units, we see that even under a close to zero load, i.e. task inter-arrival times that allow each job to finish executing alone in the datacenter, and assuming that a job consists of the calculated average of 50 tasks, 10 of those tasks will have to queue for at least as long as a task of type FP-BAD will take to execute on the fastest processing unit. Since a job completes only when all its tasks have finished executing, this means that on average, at 0 load and when at most 40 tasks may execute in parallel, a job will require the double of an individual task execution time in order to complete. This, in combination

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

with the bad affinity for GPUs, means that a much lower inter-arrival rate must be used in order to achieve close to zero load scenario.

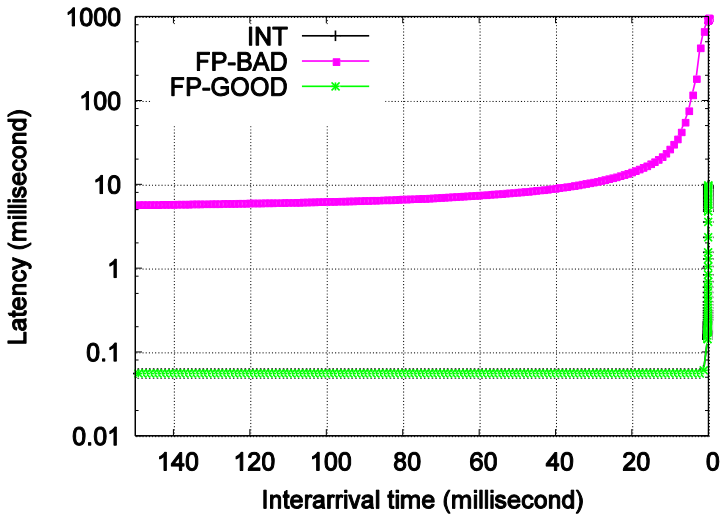


Figure 6: Exclusive workloads on 100% GPU deployment when approaching zero load.

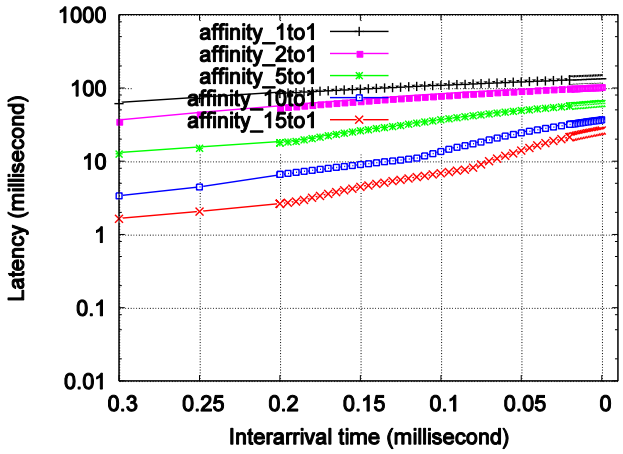
8.3 Number of accelerators vs computational power

A reasonable assumption is that processing elements are more expensive the higher their computational power, a datacenter designer might wonder whether to invest in few but potent accelerators or whether it would be better to equip the datacenter with many, less powerful ones. We construct an experiment where the workloads contain jobs and tasks of floating-point computation type that is efficient on GPUs. We use the FP_GOOD workload from the experiments of Section 8.2, i.e. a workload of 500 jobs in total, where job sizes are mixed following a realistic distribution. As with the previous experiment, we test out deployments of 40 processing units, varying the percentage of GPUs, and using the best fit scheduling policy.

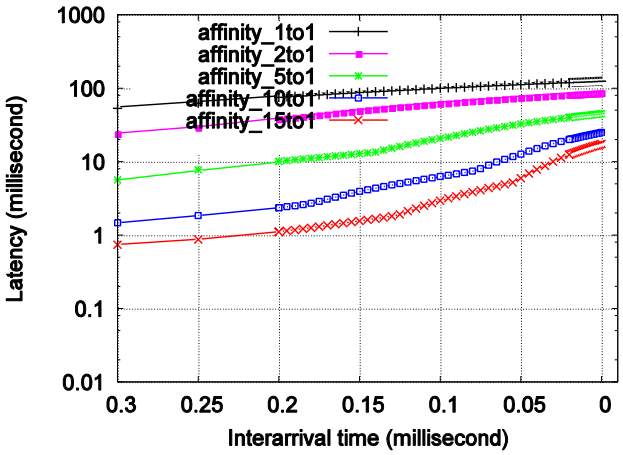
However, instead of using the affinity matrix indicated in Table 1, we come up with five affinity matrices, representing five different relative strengths of GPUs versus general-purpose processors. Recall that, by the way the affinity matrix is designed, it suffices to modify the INT column of each of the five affinity matrix versions we have, in order to specify the computational power of a GPU. We test out situations where the ratio of the computational power of a GPU with respect to a CPU is 1 to 1, 2 to 1, 5 to 1, 10 to 1, and 15 to 1. We come up with the following affinity matrices, as labeled in the following plots:

- **affinity_1to1**: CPU INT affinity is 100,000 and so is GPU affinity.
- **affinity_2to1**: CPU INT affinity is 100,000, GPU affinity is 200,000.
- **affinity_5to1**: CPU INT affinity is 100,000, GPU affinity is 500,000.
- **affinity_10to1**: CPU INT affinity is 100,000, GPU affinity is 1,000,000.
- **affinity_15to1**: CPU INT affinity is 100,000, GPU affinity is 15,000,000.

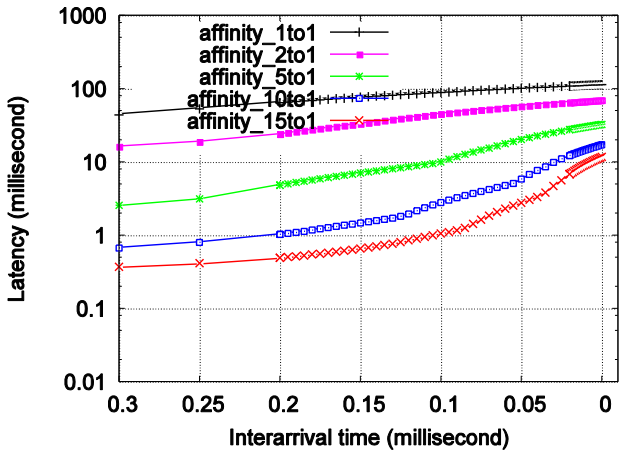
PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628



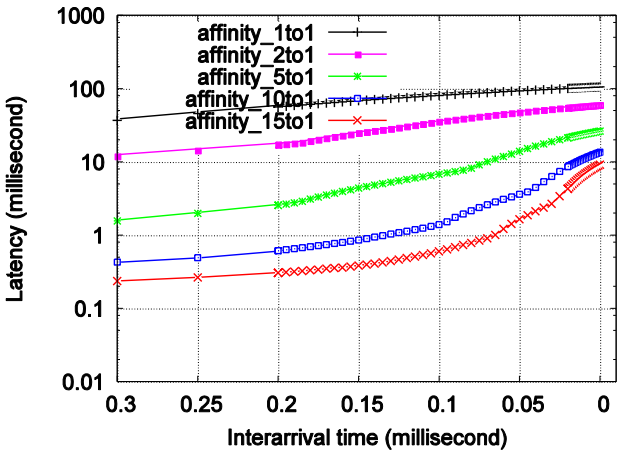
(a) 20% GPUs in deployment



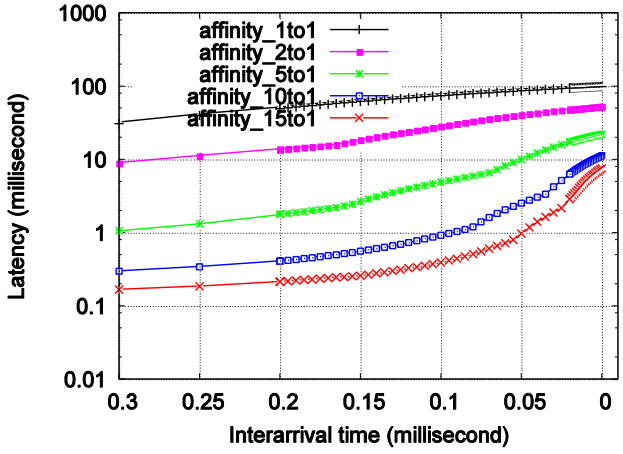
(b) 33% GPUs in deployment



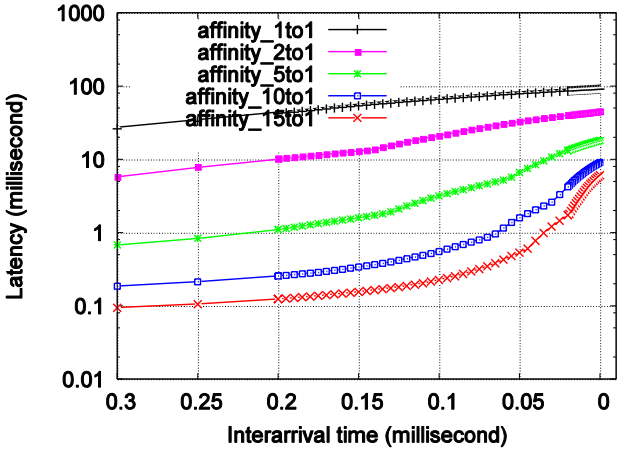
(c) 50% GPUs in deployment



(d) 66% GPUs in deployment;



(e) 80% GPUs in deployment



(f) 100% GPUs in deployment

Figure 7: Comparison between relative affinities and amount of GPUs in deployment.

In order to determine whether the potency or the amount of GPUs is of higher importance, we test out those different affinity relations in deployments that vary in the GPU content. The results of the

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission
Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

experiment are presented in *Figure 7*. The x-axis shows a varying inter-arrival rate for incoming jobs. The y-axis shows average job latency (in logarithmic scale). Time units are microseconds.

By observing *Figure 7*, where we have a deployment consisting 100% of GPUs, we notice that an increase by a factor of 15 in the affinity of the GPU, we obtain a decrease by a factor of 1000 in the average job latency. By comparing *Figure 7*(f) with *Figure 7*(a), we further observe that an increase in the percentage of GPUs in the deployment by a factor of 5, results in the decrease of average job latency by a factor 10. We further compare the behavior of affinity_15to1 of the GPU deployment of *Figure 7*(b) with the affinity_5to1 GPU deployment of *Figure 7*(f) and notice equivalent trends in average job latency. A similar observation holds for *Figure 7*(a) and (d).

This shows that, when used for GPU-affine workloads, a deployment with many less powerful GPUs can perform equivalently to a deployment that has few but very powerful GPUs. This shows that, where average job latency is concerned, a datacenter designer has flexibility when it comes to choosing a deployment. For example, if budgeting constraints forbid the purchase of GPUs that correspond to the affinity_15to1 scenario, desired performance goals may be reachable with the purchase of more affinity_5to1 type GPUs.

8.4 Evaluation of Neurasmus workloads on alternative deployments

In this section, we use VineSim to evaluate how a brain simulation workload might perform on different deployments. Neurasmus has evaluated their brain simulator tool on different accelerators[24] [25], coming up with execution numbers for computation kernels of brain networks, namely RGJ, NGJ, and SGJ. The tested accelerators are GPUs, Xeon PHI, and Maxeler DFEs. As outlined in Table 3 below, this study showed that the optimal accelerator type depends on the kernel type and the (brain) network size.

Using VineSim, we examined how a Neurasmus-inspired workload will perform on the four deployments listed below:

- 30 GPUs + 10 CPUs
- 30 PHI + 10 CPUs
- 30 DFEs + 10 CPUs
- 10 GPU + 10 PHI + 10 DFE + 10 CPUs

We considered a workload that consists of 2500 jobs, each consisting of one task that executes a single RGJ, SGJ or NGJ kernel. All kernels occur with the same probability in the workload.

[24] Georgios Smaragdos, Georgios Chatzikonstantis, Sofia Nomikou, Dimitrios Rodopoulos, Ioannis Sourdis, Dimitrios Soudris, Chris I. De Zeeuw, Christos Strydis, "Performance analysis of accelerated biophysically-meaningful neuron simulations". ISPASS 2016: 1-11

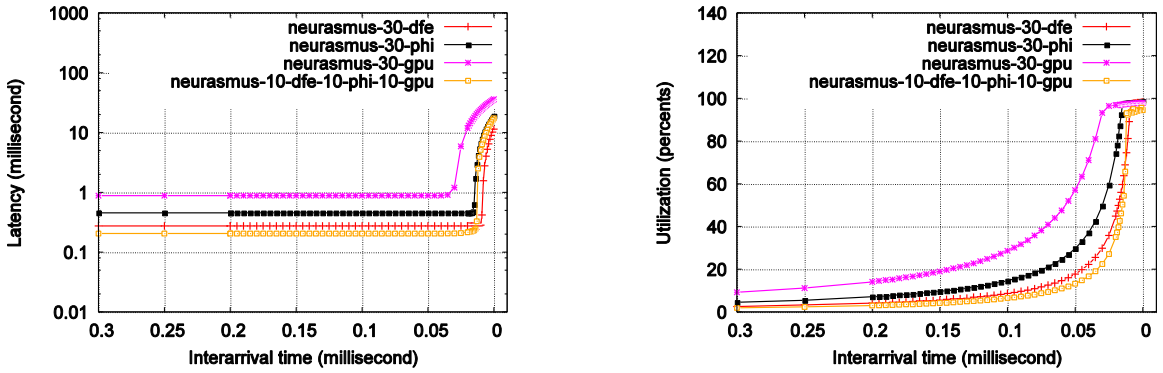
[25] Georgios Smaragdos, Georgios Chatzikonstantis, Rahul Kukreja, Harry Sidiropoulos, Dimitrios Rodopoulos, Ioannis Sourdis, Zaid Al-Ars, Christoforos Kachris, Dimitrios Soudris, Chris I. De Zeeuw, Christos Strydis: BrainFrame: A node-level heterogeneous accelerator platform for neuron simulations. CoRR abs/1612.01501 (2016)

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

Task / Workload	SGJ	RGJ	NGJ
PHI	0.5 ms	0,75 ms	0,1 ms
DFE	0,3 ms	0,5 ms	0,01 ms
GPU	0,1 ms	2,4 ms	0,06 ms

Table 3: Execution time of Neurasmus tasks on different accelerators for maximum network size.

Our results are presented in Figure 8. Using Figure 8(a), we can find the optimal (performance-wise) by comparing the average job latency. Considering the exclusive-only deployments, the configuration with DFEs performs best, then comes the one with Xeon PHI, and last the one using GPUs. Although GPUs outperform the other accelerators on kernel SGJ, they perform really bad with RGJ, which is reflected on this results. Nevertheless, the best configuration among the ones we tested is the mixed one which includes all kinds of accelerators, GPUs, DFEs and PHIs. In such a heterogeneous datacenter, a good scheduler (the best-available in our simulations) can match each task with the most responsive accelerator.



(a) Average job latency

(b) Resource utilization

Figure 8: Performance of Neurasmus workload on deployments using only DFEs, Xeon PHI and GPUs, as well as on mixed deployments.

We remark that the present experiments constitute an indicative instance of the behaviour of the Neurasmus brain simulation tools, as well as of how we can evaluate their performance in large datacenters using VineSim. This may nevertheless be useful for operators of datacenters on which users spawn brain simulation jobs. For simplicity, this first study neglected the overhead of data transfers. Additionally, we used CPUs only to coordinate tasks execution, not to execute kernels. In future work, we will add more dimensions in this study, including different brain network sizes and variable connectivity between neurons.

8.5 Lessons learned from Vinetalk studies

In Deliverable D2.2, we evaluated the execution time of various computational kernels (i.e. tasks) when running on CPU and on GPU. In these tests, we used a first working version of the VineController, which is responsible for virtualizing the accelerators present in a node, and of the

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

Vinetalk API, which allows applications to issue tasks to the VineController and to collect the results in a transparent way. Our first results were obtained from a single node configuration, consisting of a machine with 64 AMD Opteron(TM) 6272 processor cores, 256 GB memory, and equipped with a Quadro K2200 NVIDIA GPU. The Quadro GPU has 640 CUDA cores, 4 GB memory, and is connected to the main memory with 16 lanes of PCI Express 2.0, offering an aggregate bandwidth of 8 GByte/s in each direction.

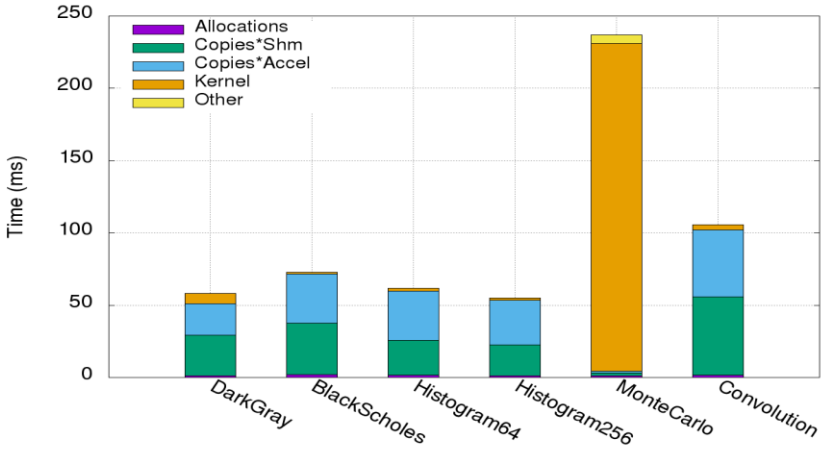


Figure 9: Latency breakdown when executing various computation kernels in GPUs.

Figure 9 presents the latency breakdowns we collected when executing kernels on GPUs. The blue boxes represent the time it takes to copy data to and from the GPU and the orange boxes represent the actual kernel execution time. As can be seen, for many of the task types that we considered in this evaulation, the transfer time exceeds the actual execution time by 1-2 orders of magnitute. One exception among our tests is "MonteCarlo", which by nature is a very compute intensive benchmark that requires iterating through all the data several times in order to produce the result.

Our findings from the tests run in D2.2 are summarized below:

- The speedup offerings of a GPU depend greatly on the task data size and on the kernel complexity. When a kernel does a single pass over the input data, e.g. as in kernels with O(N) complexity, then the time to move data may exceed the execution time on a accelerator that is capable of processing multiple data at the same time using multiple execution engines, such as a GPU.
- Despite the cost of data transfers, GPUs may significant speedups when compared to CPUs. In our tests, these ranged from 8xto 34x. Super large speedup values of 4Kx were observed in MonteCarlo.

8.5.1 The penalty of data transfers with modern accelerators

Figure 10(a) depicts schematically the data transfer problem that we discuss here. Up to now, in order to accelerate a task on an accelerator, we needed to move data from main memory to accelerator memory using some form of DMA. Effectively, we needed to serialize the data over serial links, that nowadays typically offer less than 16 GByte/s transfer speed. At the receiving

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

end, FPGAs and GPUs may run thousands threads concurrently, at GHz speed, thus collectively passing through data at TBytes/s speed. Thus, it becomes obvious that for tasks that do not iterate over the input data multiple times, the transfer time may dominate the task completion time.

Note that the transfer time penalty may occur on the PCIe links that connect the device with the processor and the main memory. It does not necessarily increase significantly when offloading a task to a remote accelerator, in a different machine, as the cluster network can be considered as "one additional hop" in the path from main memory to the accelerator: a clever DMA engine at the receiving NIC can move incoming data to the targeted accelerator memory[26]. As shown in Figure 10(b), a remote accelerator adds a network round-trip time latency, which will be prohibitive only for super latency-sensitive applications. The situation may change of course if the network has very small bandwidth, e.g. 1 Gb/s, or when it becomes congested.

8.5.2 Hiding the heterogeneity inside the processor package

As more transistors are becoming available inside the chip, the computer architecture community and the processor industry are looking for efficient ways to utilize it. For some time now, accelerators have been considered as candidates to occupy this dark silicon inside the chips[27].

Recently, Intel announced its new offerings that combine CPUs with FPGA accelerators inside the same package[28]. Additionally, NVIDIA combines ARM processors with GPUs[29]. In these systems, the accelerators may access task data directly from main memory, obviating the need and the overheads for additional data transfers.

Additionally, these new systems offer the possibility to access the main memory from the accelerator side using virtual addresses, leveraging the System Memory-Management-Units (SMMU). The benefit of using virtual addresses is that pointers are valid at the accelerators, which enables the accelerator to traverse complex data structures that use pointers to connect memory segments. Finally, in these systems, it may become possible that the accelerator maintains caches that are coherent with the CPU caches, e.g. using a special protocol such as CAPI.

A prominent upcoming example that can take advantage of on-chip or on-board communication is that of FPGAs that implement accelerators. A benefit that these FPGAs offer is that they allow programming close to data whenever necessary and therefore, they can overcome the issue of distance in cases it poses a significant problem.

---

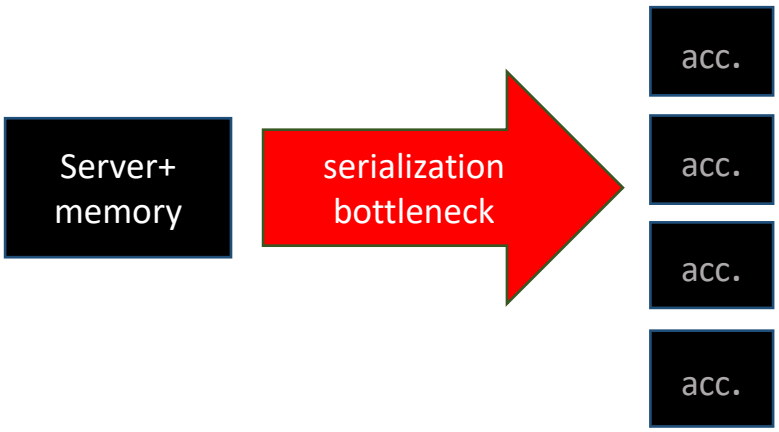
[26] www.mellanox.com/related-docs/prod_software/PB_GPUDirect_RDMA.PDF

[27] H Esmaeilzadeh, Dark silicon and the end of multicore scaling, ACM SIGARCH Computer Architecture News, 2011

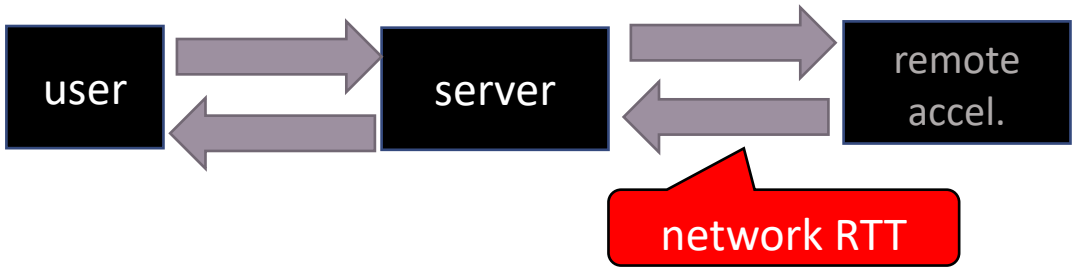
[28] www.pcworld.com/article/3023055/hardware/the-first-fruits-of-intels-biggest-buy-ever-will-come-this-quarter.html

[29] http://www.nvidia.com/object/tegra-4-processor.html

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628



(a) Serializing data from main memory to accelerator memory, over e.g. PCI, may slow down the performance of multi-threaded accelerators, such as GPUs with thousands of cores.



(b) Accessing a remote accelerator over an uncongested network may not necessarily increase the transfer time, as compared to accessing the accelerator locally through PCI, but introduces an latency round-trip time overhead, which is independent of the task data size.

Figure 10: Communication overheads when transferring data to a multi-threaded accelerator.

9 Distributing work in heterogeneous datacenters

A datacenter or compute cluster consists of many interconnected compute nodes that are shared among users that spawn jobs. Each node is typically equipped with disk devices that store application data, while a cluster-wide file system, such as GFS, allows accessing any file from any node in the cluster. Clusters are typically designed such that any user job can be executed on any node, thus increasing the statistical multiplexing, thus also the utilization and the user-perceived performance.

Jobs typically consist of tasks. Although in principle tasks may communicate with each other, in practice, many data-intensive jobs consists of (embarrassingly-) parallel tasks that can run independently with low or no inter-task communication traffic. In this environment, we need a cluster scheduler to allocate machines to jobs and to time-schedule tasks.

9.1 Schedulers in homogeneous datacenters

Data locality has served as a central pillar for the scale-out compute paradigm, in which one can improve application performance by running it in a cluster with more nodes. State-of-the-art cluster schedulers typically try to enforce data locality by executing tasks on nodes that are physically close

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

to the storage devices that store the task data. In this way, data do not have to travel over the typically oversubscribed datacenter interconnect.

Thus, current schedulers try to match the tasks with the data they operate on. They typically do so by creating a preference index for each task-node pair that increases with the physical distance between the task data and the node. In practice, the schedulers prefer to execute tasks on

- the nodes that store the task data; for resiliency, GFS and HDFS store each data segment in more than one segment, thus more than one options may be possible.
- The nodes in the same rack with the task data -- the network inside the rack typically has more capacity available than the inter-rack network.

Previous work has identified a conflict between data locality and fairness in scheduling. Here, we identify the conflict between data locality and heterogeneity.

9.2 What changes due to heterogeneity

The heterogeneity of compute resources that comes with emerging datacenters introduces an additional optimization knob that we can twist to improve scheduling. Consider a datacenter consisting of general-purpose CPUs and GPUs. Additionally, consider that we can run any task either on a CPU or a GPU. Obviously, a task with lots of floating-point operations will run faster on a GPU; in contrast, a task that does many memory accesses may run faster on a CPU, due to the better memory subsystem of CPUs. Thus, in principle, the scheduler of a heterogeneous datacenter can improve the performance by executing each task on the most appropriate computing resource.

In many deployments, the best-performing accelerator will typically not be present on every node in the system, thus optimizing on accelerator performance reduces the number of nodes on which a task may run on. Effectively, tasks have two allocation preference vectors, which may conflict with one another. On one hand, we want to execute tasks close to their data. On the other hand, we want to execute tasks on "good accelerators", which may bring computation away from its data.

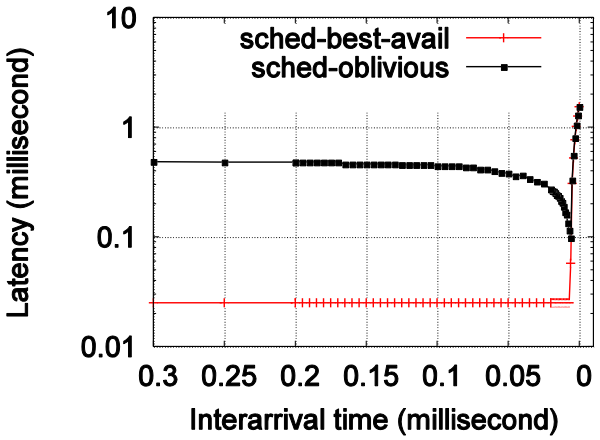


Figure 11: Average job latency versus job inter-arrival time (IAT) for two different scheduling policies.

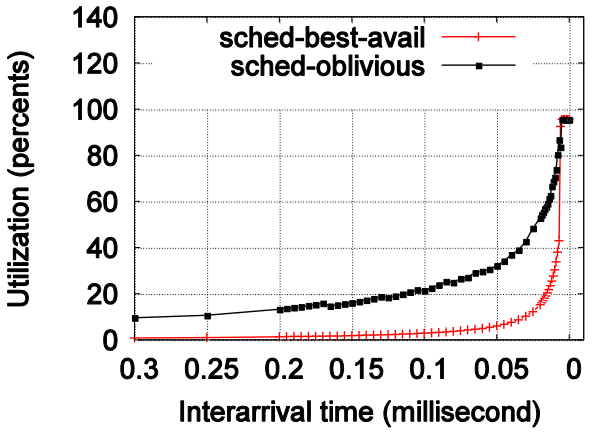


Figure 12: Resources utilization versus job inter-arrival time (IAT) for two different scheduling policies.

9.2.1 Questions Asked

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission
Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

Using VineSim, we have performed a number of experiments to study scheduling alternatives for heterogeneous datacenters. The questions that we address in these experiments are listed below:

- *should we schedule tasks close to data or on best-performing accelerator?*
- *should we avoid running tasks on slow-performing accelerators?*
- *what is the impact of data transfers?*

9.3 Scheduling alternatives

In our evaluations, we examine the following schedulers:

- The ***Oblivious*** (or FLAT) scheduler executes every task on the next available (i.e. not currently busy) node, overlooking task and accelerator types. Thus, this scheduler is unaware of the different node capabilities, and thus oblivious of the datacenter heterogeneity. The oblivious scheduler is also unaware of data location.
- The ***Best-Available*** (or HIGH) scheduler tries to match each task with the best available (i.e. not currently busy) accelerator in the system.
- The ***Preferred-Only*** scheduler follows the task description when choosing where to execute it. Each task defines a type of nodes on which it may run. Hence, given a task with preferred accelerator type *T*, the scheduler will scan all available (i.e. not currently busy) nodes trying to find an available accelerator of type *T*. If no accelerator of type *T* is available, then the scheduler will try to match the next task.

Note that the Oblivious and the Best-Available schedulers are work conserving, in the sense they will never leave resources idle while there is work to do. On the other hand, the Preferred-Only scheduler is non-work-conserving, since it will not execute a task until one of its preferred accelerators become available, thus possibly underutilizing resources.

Note also that while the Oblivious and the Preferred-Only schedulers execute tasks in-order, the Best-Available may execute them out of order. This happens when a task is blocked waiting for some accelerator to become free: subsequent tasks can be spawned if they work with different but available accelerators. Nevertheless, whenever one resource is free, we scan the task queue in order to see if we can match it with a waiting task; hence, in this way, we avoid starvation.

9.4 Experimental setup

In our experiments, we consider a datacenter consisting of 40 nodes; in our baseline configuration, these nodes are:

- 20 multi-core general-purpose servers running at 60 GFLOPs maximum speed
- 20 GPUs running at 1.2 TFLOPs.

We use two vastly unequal (performance-wise) compute nodes, in order to highlight the issues that may arise in heterogeneous clusters. Our (**500Jobs**) workload consists of:

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

- 500 jobs
- 5 tasks per job
- 30M operations per tasks.

In our experiments, we vary the job inter-arrival time (IAT) in order to model both low and high load conditions. We have purposely selected a simple workload in order to extract meaningful deep insights from the experiments that we run.

9.5 Why heterogeneity-aware scheduling matters

In our first experiment, we compare the Oblivious scheduler with the Best-Available scheduler. Our results for the average job latency are presented in Figure 11. The two schedulers compare differently to each other depending on the incoming load, which we vary along the x-axis by changing the inter-arrival time:

- At light load conditions (large IAT), the heterogeneity-aware scheduler outperforms the oblivious one by more than one order of magnitude. In particular, the best-available scheduler achieves 20x lower latency than the oblivious scheduler.
- At high loads (small IAT), the two schedulers achieve similar performances, implying that scheduling decisions do not matter that much.

In this experiment, a job consists of five tasks. The execution time of a job will thus be minimized when all of its tasks are executed in parallel on five GPUs. With oblivious scheduling, the type of the processor executing a task will be more or less a stochastic process: some tasks may be executed on a CPU and some on a GPU. Especially at low loads, when nearly all resources are idle, half of the tasks will be executed on GPUs and half in CPUs. In this case, the job execution time will be determined by the slowest task(s), i.e. those running on CPUs.

The task completion time on a CPU is equal to (number of operations per task *divided* by operations per sec) 30M/60 GFLOPs = 0.5 milliseconds. On the other hand, the task completion time on a GPU is equal to 30M/1200 = 0.025 milliseconds, i.e. 20 times smaller than on a CPU. Therefore, the oblivious scheduler may schedule correctly half of the tasks, but, in practice, the job-level performance will be dictated by the CPU time.

In contrast, at low loads the best available scheduler successfully schedules the tasks on their best-performing accelerator, as there are more GPUs than tasks per job. Effectively, at low loads, the scheduler is able to sustain the best possible performance levels.

At high loads, i.e., when the IAT approaches zero, the two schedulers perform identically. Under high load, it becomes very likely that the best available scheduler cannot find the GPUs needed for all job tasks; therefore, it will execute jobs in a set consisting of both GPUs and CPUs.

In this experiment, the number of tasks per job was smaller than the number of GPUs. If the number of tasks per job exceeded the number of GPUs, then, the best-available scheduler would perform similarly with the oblivious. One however can think of variations to Best-Available, which,

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

instead of allocating a task to one of the currently available schedulers, may decide to wait for a better match to become available later in time.

Independent of the scheduling policy, at low loads, there is no or very small queuing time: all jobs are executed as soon as they arrive in the system. The queuing time becomes apparent at very high loads, i.e. when the IAT approaches zero.

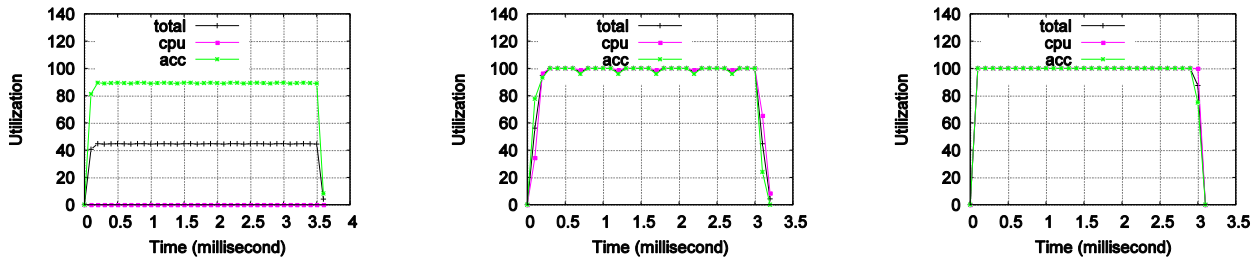
In fact, there is a region on the horizontal axis, for which, while the load increases, the latency of the oblivious scheduler actually decreases. Thus, it seems that as the load increases, the probability that all the tasks of a jobs are executed on GPUs increases as well. To explain this behavior, notice that it emerges for IAT values smaller than 0.2 milliseconds, because then jobs arrive faster than the task time on CPUs. Therefore, when a new job arrives, the previous job is still occupying some CPUs; effectively, the number of available GPUs becomes larger than the number of available CPUs. Thus, the probability that a job is executed entirely on GPUs increases, reducing the average job latency. As the load increases further, and the IAT approaches the GPU task time (0.03), this effect disappears. From that point and on, the queuing component of latency dominates.

Figure 12 presents the utilization of resources as a function of the IAT. As can be seen, the utilization with the oblivious scheduler is considerably lower for all IAT values, except those that close to zero. Thus, a better (heterogeneity-aware) scheduler can reduce the operational costs, while delivering similar or better performance.

9.6 Utilization can fire up for a very small increase in load

Figure 13 presents the utilization time-series separately for the GPUs and the CPUs for different inter-arrival times (0, 6 and 7 μsec), when using the best-available scheduler.

For IAT ≥ 7 μsec, the GPUs can handle the incoming load, without any help from the CPUs: the deployment has 20 GPU, which can handle 4 concurrent jobs with 5 tasks each. A busy GPUs becomes free after 25 μsec. Hence, if the workload generates at most 20 tasks (or 4 jobs) every 25 μsec, i.e.,$IAT \geq \left\lceil \frac{25}{4} \right\rceil = 7\ \mu sec$, the CPUs will remain idle, and thus be available for other work. This is shown in Figure 13(a).



(a) IAT = 7 μsec (b) IAT = 6μsec (c) IAT = 0 μsec

Figure 13: The utilization time-series for the best-available scheduler for various job inter-arrivals times (IAT).

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission
Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

Going from an IAT of 7 to 6 μsec, in Figure 13(b), the GPUs alone cannot handle the load; then, the best-available scheduler tries to explore the available CPUs. Effectively, all CPUs are now put to work in order to handle just a 16% load increase.

As shown in Figure 13(c), for an IAT=0, the workload is finished in about 3 milliseconds. The total number of tasks in this workload is 500 jobs x 5 tasks / job = 2500. If all 40 nodes were GPUs, all tasks would finish in (2500 / 40) x 0.025 = 1.562 msec. In reality, half of the nodes are CPUs, which essentially provide marginal help in reducing the total execution time. It follows that the expected time to finish the workload is slightly smaller than 2x1.562 = 3.24 milliseconds, as verified in the figure.

9.7 Non work-conserving schedulers

In previous examples, we have seen that in certain situations, putting CPUs to work increases resources utilization but with marginal improvements in performance. This happens especially with scale-out data-parallel jobs, which spawn independent tasks and wait for all of them to complete: spawning one or more tasks to "sluggish" nodes may cancel the fast responses that other nodes may produce. Could it then under certain circumstances be beneficial to stop allocating available compute nodes?

For instance, in our test case, 25 CPUs can handle the same number of tasks per time unit as a single GPU. Hence, one could spare 25 CPUs for just one GPU – this argument neglects the I/O and network bottlenecks of a single GPU node.

This question triggered us to consider the preferred-only *non work-conserving* scheduler, which matches every task to a resource of a predefined type. If this resource is not available, the scheduler will skip this task, although it could possibly schedule it to a resource of a different type. Consider that the same policy would be appropriate if for some tasks we have code to run it on only one type of compute nodes.

Our VineSim results are presented in Figure 14. As can be seen in Figure 14(a), the preferred-only scheduler that prefers GPUs performs identically with the best-match scheduler in terms of latency. The two schedulers however do not make exactly the same decisions. As shown in Figure 14(c), the preferred-only scheduler will never utilize any CPU, whereas the best available scheduler does so on very high loads. On the other hand, the naïve oblivious scheduler keeps CPUs utilized even at low load.

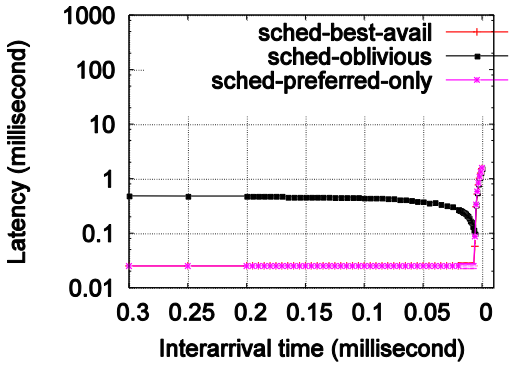
For completeness, Figure 14(b) examines the case when preferred-only scheduling is configured to select CPUs instead of GPUs. As can be seen, at low loads, the preferred-only solution now performs identically with the oblivious scheduler, as the job latency is dictated by the slow CPUs. As the load increases, the latency of the oblivious scheduler first drops, because it tends to allocate some jobs exclusively on GPUs --for the reason we explained above--, and then increases due to queuing. On the other hand, once jobs arrive faster than they complete on CPUs, the latency of the preferred-only scheduler increases due to queuing.

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

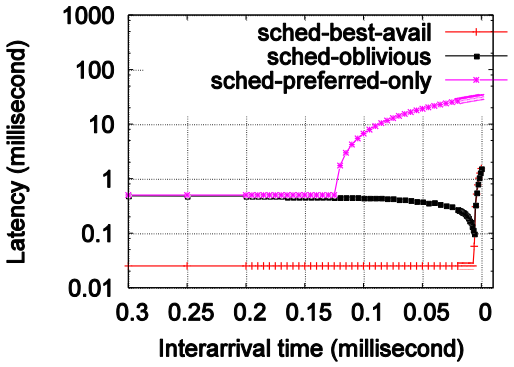
Note that the preferred-only scheduling we considered here is one of the many possible forms of work-conserving schedulers. Better solutions may wait for a given amount of type for a preferred resource to become free, similar to what delayed scheduling does to improve data locality[30].

9.8 The cost of data transfers

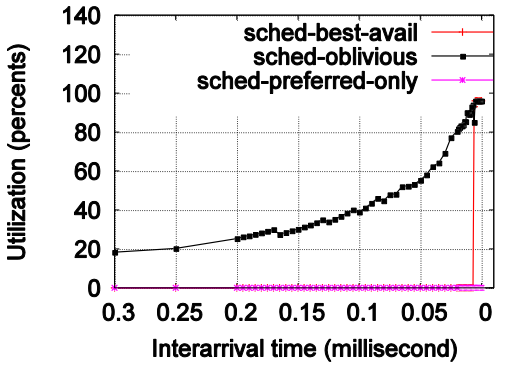
In our following heterogeneous experiments, we evaluate the implication of data transfers when scheduling in datacenters. For this purpose, we will consider different geographical distributions for task data and different task data sizes.



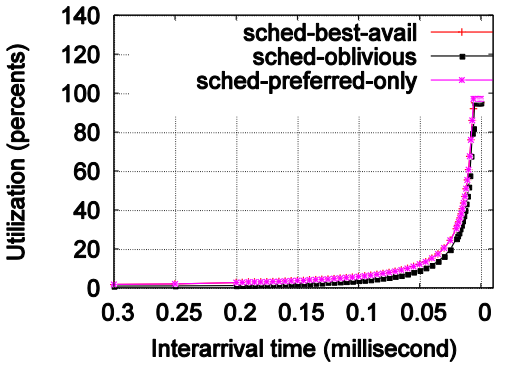
*(a) Average job latency versus job inter-arrival time (IAT) for three scheduling policies (**preferred = GPU**).*



*(b) Average job latency versus job inter-arrival time (IAT) for three scheduling policies (**preferred = CPU**).*



*(c) CPU utilization for the three scheduling policies (**preferred = GPU**).*



*(d) GPU utilization for the three scheduling policies (**preferred = GPU**).*

Figure 14: Performance of preferred-only scheduling vs oblivious and best available.

9.8.1 Uniform deployment

30M Zaharia, et al, Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling", Proceedings of the 5th European conference on Computer systems, 265-278

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

Figure 15(a) depicts a computing cluster consisting of two racks, each consisting of 20 nodes. Every node has a storage device; additionally, every second node contains a GPU. This may correspond to a deployment where the GPUs are attached to the PCIe slots of some compute nodes. Thus, Figure 15(a) represents a uniform deployment, where CPUs and GPUs are equally distant from data.

The nodes in the same rack are all connected to a top-of-rack (ToR) switch using 10 Gb/s links. The ToR switches of the two racks are connected to a spine switch. Our simulator does not model yet network contention, and thus will not reproduce congestion effects. To account for contention, we assume that the ToR-Spine links run at only 1 Gb/s, which is ten times slower than intra-rack links. Additionally, every switch introduces a cut-through (packet-level) latency overhead of 200ns. We assume minimal path routing, i.e., a two hops path when the source and the destination are located in the same rack, and four hops path when the source and the destination belong to different racks.

In our model, we currently do not pipeline transfer time with execution time. Hence, we assume that a resource is busy while task data are being transferred to it.

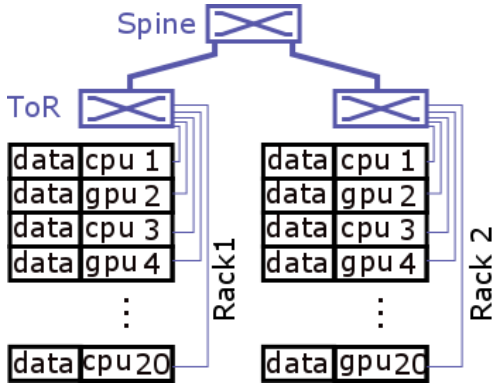


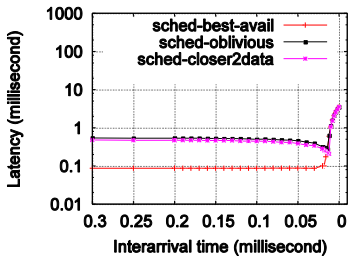
Figure 15: A uniform deployment of accelerators inside two racks.

We have modified our 500Jobs workload, so that each task has its data placed in a randomly selected node using a uniform distribution. In our experiments, we test task data sizes equal to8 KByte, 64 KByte, and 1 Mbyte. Table 4 summarizes how different factors affect task latency.

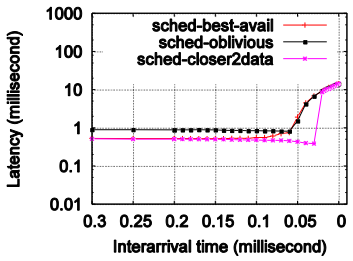
Task Execution time	Data transfer time for task data = 8KB	Data transfer time for task data = 64KB	Data transfer time for task data = 1MB
GPU 25 μsec	6,22 μsec inside rack	50 μsec inside rack	800 μsec inside rack
CPU 500 μsec	62,2 across racks	500 μsec across racks	8000 μsec across racks

Table 4: Comparing task transfer and execution times.

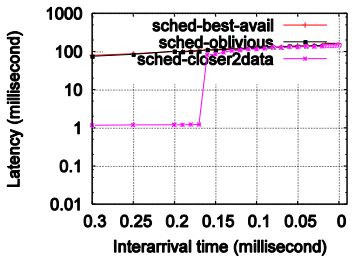
PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628



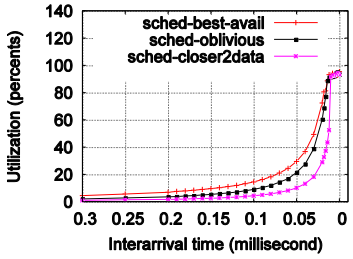
*(a) Average job latency versus job inter-arrival time (IAT) for 3 scheduling policies (**data size 8K**).*



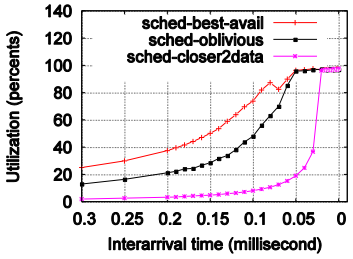
*(b) Average job latency versus job inter-arrival time (IAT) for 3 scheduling policies (**data size 64K**).*



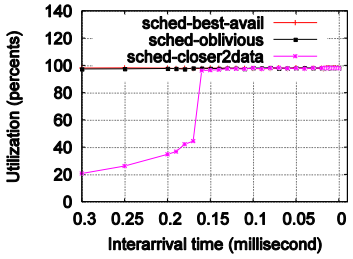
*(c) Average job latency versus job inter-arrival time (IAT) for 3 scheduling policies (**data size 1MB**).*



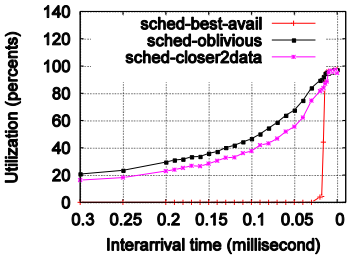
*(d) GPU utilization for 3 scheduling policies (**data size 8K**).*



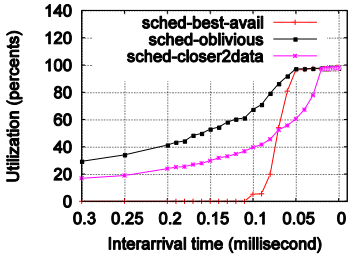
*(e) GPU utilization for 3 scheduling policies **(data size 64K).***



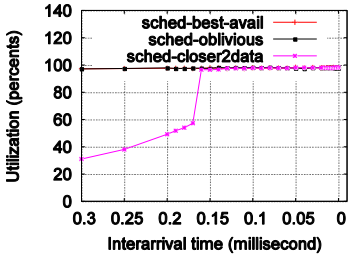
*(f) GPU utilization for 3 scheduling policies **(data size 1MB).***



*(g) CPU utilization for 3 scheduling policies (**data size 8K**).*



*(h) CPU utilization for 3 scheduling policies **(data size 64K).***



*(i) CPU utilization for 3 scheduling policies **(data size 1MB).***

Figure 16: Performance of best available, oblivious and closer-to-data scheduling for the 2-rack system and the data distribution depicted in Figure 15(a).

Note that the Closer-to-Data scheduler optimizes data locality but is blind to different computing capacities, similar to the Oblivious Scheduler. Conversely, the best-available scheduler optimizes compute affinity but ignores data locality.

In addition to best-available and oblivious, in the next experiments we also test a scheduler that minimizes data transfers. In particular, when the ***Closer-to-Data*** schedules a task, it scans all the available (not currently busy) nodes, giving the highest priority to the node keeping the task data, n_d, medium priority to nodes in the same rack with n_d, and the lowest priority to nodes located in different racks.

Figure 16 depicts the performance of the aforementioned schedulers when running the 500Jobs workload in deployment of Figure 15. In Figure 16(a,b,c) we present the job latency for different data transfer sizes. As can be seen in Figure 16(a), for small transfers (8KB), the best-available

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission
Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

scheduler achieves the lowest latency at low loads, whereas the closer-to-data behaves similar to the oblivious scheduler. As the load increases, the GPUs alone cannot handle the load, and the three schedulers yield the same latency.

In Figure 16(b), for 64KB task data, the transfer time across racks is roughly equal to the execution time on a CPU (see Table 4). For this reason, the closer-to-data yields similar latency with the best-available scheduler. At higher loads, the closer-to-data outperforms the best-available. This happens because the good performance of best-available depends on the availability of GPU nodes. Once these are all used, it has the same performance as the oblivious scheduler (see Figure 16(e)). On the other hand, the good performance of closer-to-data depends on the availability of any node, as long as this resides in the same rack with the task data. As can be seen in Figure 16(e,h), the closer-to-data has many candidates to choose from in order to optimize for data movement.

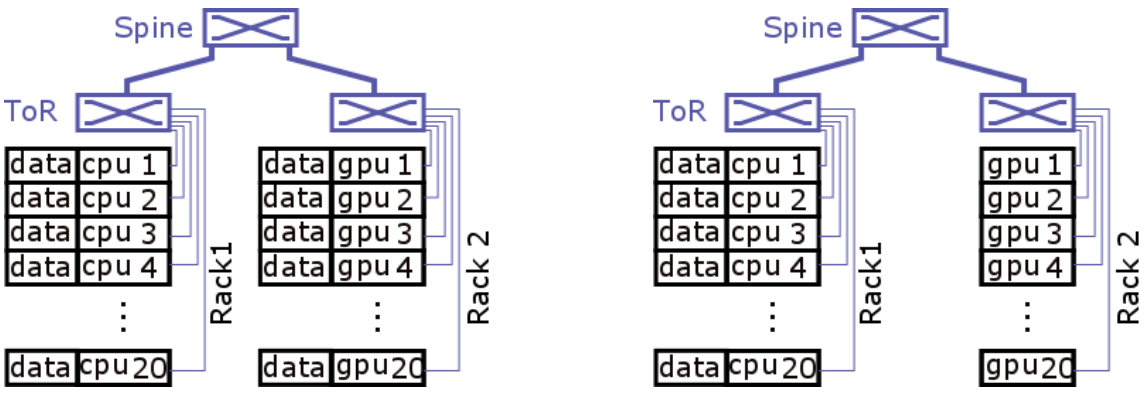
Closer-to-data scheduling shines for really large data transfers, in Figure 16(c,f,i). As shown in Table 4, the transfer time of 1 MB on inter-rack links (8ms) is 10 times larger than the transfer time inside the rack and 16 times larger than the execution time on a CPU. Hence, even when the closer-to-data scheduler takes the "wrong" decision to execute a task on a CPU, chances are that it scheduled the task inside the rack that holds the data, hence speeding up the data transfer by 10x.

For inter-arrival times below 0.15 milliseconds, all resources become busy. At this point, the decisions are no longer affected by the scheduling policy: once a node becomes free, any work-conserving scheduler will allocate it to the next task.

9.8.2 Unbalanced deployment

Figure 17 depicts an unbalanced computing cluster consisting of two racks. The first rack consists of 20 CPUs and the second rack consists of 20 GPUs. In Figure 17(a), the task data are distributed across all nodes, whereas in Figure 17(b) they are distributed only on nodes of rack 1.

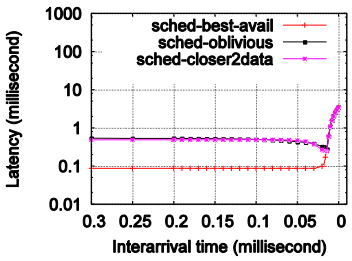
Our results for both deployments are presented in Figure 18. As can be seen, the best-available scheduler is optimal for small task data, and the closer-to-data scheduler for large task data. In all cases, the performance differences are visible only for low and medium input loads. At high loads, the queuing effect dominates. It is interesting that for 1 MB data sizes, keeping all data close to CPU (Figure 18(f)) yields significantly lower latency than having them distributed across both racks (Figure 18(c)). We are currently investigating the reasons behind this behavior.



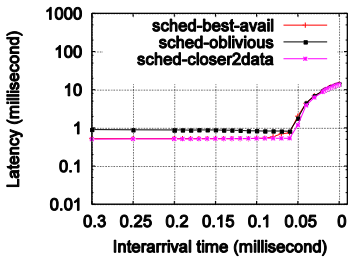
PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission
Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

(a) Data distributed at all nodes *(b) Data located only at nodes of rack 1*

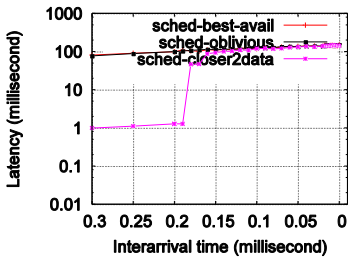
Figure 17: An unbalanced deployment of accelerators. Accelerators are located in a remote rack.



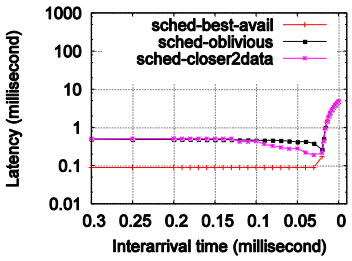
*(a) Average job latency versus IAT for 3 scheduling policies (**Figure 17(a),data size 8K**).*



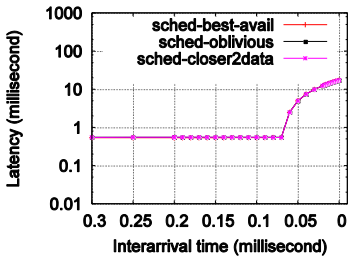
*(b) Average job latency versus IAT for 3 scheduling policies (**Figure 17(a), data size 64K**).*



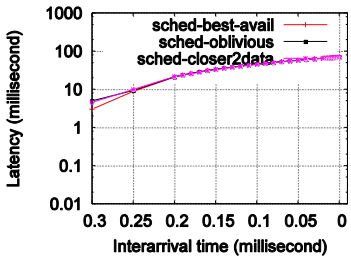
*(c) Average job latency versus IAT for 3 scheduling policies (**Figure 17(a),data size 1MB**).*



*(d) Average job latency versus IAT for 3 scheduling policies (**Figure 17(b), data size 8K**).*



*(e) Average job latency versus IAT for 3 scheduling policies (**Figure 17(b), data size 64K**).*



*(f) Average job latency versus IAT for 3 scheduling policies (**Figure 17(b), data size 1MB**).*

Figure 18: Performance of best available, oblivious and closer-to-data scheduling for the 2-rack system and data distributions depicted in Figure 17.

10 Summary

The present deliverable describes our effort to model and evaluate alternative deployments for heterogeneous datacenters. For this purpose, we first surveyed accelerator use in existing datacenters. We then distilled the key concepts into a theoretical model, which describes physical components, communication features, and connectivity characteristics of a datacenter, but which also abstracts the notion of workload and describes it by dividing it into subordinate elements of computation, i.e. tasks. We used this theoretical model in order to express observable characteristics of datacenter behavior, and to formulate it as the problem of optimizing a utility function, whose solution leads to determining the best deployment for a given workload.

We consider that such utility functions can aid the designer of a datacenter in deciding how to best plan a deployment, given the characteristics of the workloads that have to be served by it – and possibly also vice versa, i.e. given a deployment, what types of workloads is it suitable for. Analytically, this would be achieved by fixing the values of those parameters that express e.g. deployment characteristics and determining the values that other parameters have to have in order to minimize or maximize the function.

We also implemented VineSim, a flexible datacenter simulator. VineSim does not model a specific datacenter nor is it limited to modeling a specific datacenter application. Instead, it models a

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission | Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628

workload as consisting of jobs, which in turn contain tasks. A task is viewed as an elementary unit of computational work, which can perform well or worse on different processing elements that the datacenter may contain. This frees us from being restricted to modeling the behavior of an entire application as a whole, e.g. a map-reduce framework or web queries. Instead, VineSim offers the possibility of expressing a wider range of application types, provided they can be broken down into VineSim task types. Those task types are easily extendable, to include new categories that weren't taken into account during initial stages of development. The same adaptability holds for the processing units that a datacenter may contain. Further flexibility of VineSim stems from the fact that it allows for parameterization of several of its core aspects, such as what scheduling strategy will be used when assigning tasks to processing elements.

We used VineSim in order to evaluate indicative accelerator deployments in datacenters but also, basic scheduling strategies. Among other things, we find that:

- Datacenter performance characteristics are heavily workload-dependent.
- There is a fine balance between the performance benefits that can be gained by few but powerful or many but less powerful accelerators.
- Scheduling has a significant impact on average job latency, especially for heterogeneous datacenters, but these effects may be obliterated by high job arrival rates.
- Even small increases in load may significantly increase the processor utilization.
- For scale-out, task-parallel jobs, it is important to have as many accelerators available as the number of concurrent tasks.
- Tasks that process large amounts of data benefit from placement close to them.

We are currently working together with Vineyard partners on evaluating how Vineyard-relevant applications perform on different accelerator types. One can use these performance numbers in order to estimate through VineSim how those applications might perform in large-scale deployments. The first outcomes of our ongoing collaboration with Neurasmus were presented in the current deliverable.

Future avenues of research with VineSim will deal with exploring the behavior of more scheduling schemes. A further step is the inclusion of more accelerator and task types. Another intention is to integrate VineSim with real-life workload traces and datacenter deployments in order to use its predictive powers in more realistic settings.

PARTNERS
Institute of Communications and Computer Systems, Maxeler Technologies, Bull Systems, Queen's University of Belfast, Foundation for Research and Technology, The Hartree Centre / Science and Technologies Facilities Council, Neurasmus BV, Neurocom Luxembourg, Hellenic Exchanges SA, Holding, Clearing, Settlement and Registry, LeanXcale, Loba
European Commission
Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement n° 687628