Heterogeneous GPU reallocation

James Gleeson, Eyal de Lara
{jgleeson,delara}@cs.toronto.edu
University of Toronto

Abstract

Emerging cloud markets like spot markets and batch computing services scale up services at the granularity of *whole VMs*. In this paper, we observe that GPU workloads underutilize GPU device memory, leading us to explore the benefits of reallocating heterogeneous GPUs within *existing VMs*. We outline approaches for upgrading and downgrading GPUs for OpenCL GPGPU workloads, and show how to minimize the chance of cloud operator VM termination by maximizing the heterogeneous environments in which applications can run.

1 Introduction

Cloud infrastructure is becoming increasingly heterogeneous, offering new tradeoffs in energy, performance, and total cost of ownership (TCO). Microsoft investigated a custom FPGA deployment and its efficacy in accelerating Bing web search workloads, and were able to obtain a 95% increase in throughput at ranking servers for the same latency, 30% increase in TCO, and $< 10\%$ increase in power [25]. Beyond infrastructure, heterogeneous devices such as GPUs are now being offered to VM instances [21, 15, 8].

In order to reduce idle resources in the data center, Amazon has begun offering two new services to its cloud users. As an economic alternative to hourly-pricing, *spot markets* allow users to bid on resources, with machines being allocated to the highest bidder, with the lower bidder being terminated. For tasks with dynamically changing resource requirements that wish to avoid paying for resources they do not use, Amazon offers a *batch computing service* that monitors and automatically scales batch computation tasks by allocating/terminating VMs as computational load changes over time [4, 5].

However, the key issue with today's services aimed at reducing idle resources is that resources are allocated at the granularity of *whole VMs*. This leads to several negative side effects. When scaling down through VM termination, any progress made in the VM so far is lost and must be recomputed. When scaling up through VM allocation, a computationally bound workload may benefit most from a more powerful GPU, so a user will not want pay for additional VM resources (e.g. RAM); further, any unused but allocated resources prevent a cloud operator from achieving high user-to-machine density.

In this paper, we propose solving these issues for heterogeneous GPUs through the *reallocation* of GPUs within an *existing VM*. First, we observe GPU underutilization for a set of OpenCL and OpenGL applications with less than than 37% of GPU device memory used for a 3016 MiB Quadro K4000.

We show how we can extend the OpenCL GPGPU virtualization solution Crane [14] to support reallocation of heterogeneous GPU models by hiding the binary format of compiled OpenCL programs and reinjecting GPU state into a reallocated GPU of a different model. We design two approaches to GPU reallocation. ***GPU overcommit*** allows transparent GPU upgrade and downgrade without application modifications, but can lead to poor GPU reallocation choices. We propose a ***reconfiguration API*** where applications specify minimum and preferred GPU resource requirements needed to make correct GPU reallocation decisions, maximizing the heterogeneous GPUs a workload can execute with to minimize the chance of VM termination by a cloud operator. Finally, we show preliminary results that illustrate a hypothetical cloud scenario where a cloud operator downgrades a video encoder workload to a sufficient capacity GPU, ensuring performance after reallocation and freeing up high-capacity GPU resources for other users.

2 GPU heterogeneity in the cloud

First, we establish that GPU device memory is underutilized for both OpenCL and OpenGL workloads, which gives cloud operators the opportunity to dynamically

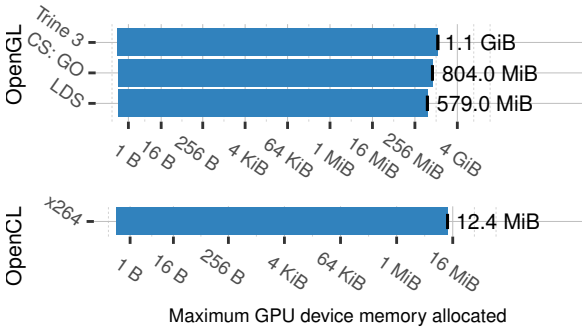

Figure 1: Maximum GPU device memory allocated by OpenCL and OpenGL programs for a Quadro K4000 with 3016 MiB. Less than 37% of GPU device memory is used by GPU workloads.

downgrade GPU workloads to a GPU with sufficient capacity. Next, we investigate existing cloud services that benefit from heterogeneous GPU device reallocation, for both the *cloud operator* and the *cloud user*.

2.1 GPU device memory is underutilized

We profiled the maximum GPU device memory usage of both OpenCL and OpenGL applications over their lifetime. For our experiments, we used a Quadro K4000 with 3016 MiB of GPU device memory. For OpenCL, we used the VLC video player's x264 video encoder library to encode a 4.7 GiB, 360p video from uncompressed YUV4MPEG into MKV format. For OpenGL, we looked at selection of popular games [1]. We used a screen resolution of 1024×768, and used default graphics settings; the default settings for this powerful a GPU were the highest graphics settings.

For *OpenGL*, Figure 1 shows that all three games use less than 37% of the available GPU device memory. Given that video games are designed to run with a minimum set of system requirements, the fact that the games do not use the entire GPU comes as no surprise. For *OpenCL*, x264 only requires as much GPU device memory as is required to fit the image frames that it is encoding in a given batch, which is only 12.4 MiB.

Given that GPU workloads underutilize their GPU, there is an opportunity for data center operators to *downgrade* workloads to sufficient capacity GPUs. Next, we consider how heterogeneous GPU reallocation can benefit emerging cloud services like *spot markets* and *batch computing*.

[1]We abbreviate Counter-Strike: Global Offensive (CS:GO) and Lovers in a Dangerous Spacetime (LDS) in Figure 1

2.2 Improving cloud services through heterogeneous GPU reallocation

Amazon spot markets allow users to bid on otherwise idle resources at a reduced price to hourly-priced VMs, but risk revocation through VM *termination* when the spot price rises above their bid. In order to support revocation, applications checkpoint their state as they run, and perform recomputation when being resumed. Revocation costs threaten to devalue the spot market; if a spot server only offers 50% of the performance, then any spot price $> 50\%$ is worse than simply purchasing a non-revocable EC2 VM instance [27].

Amazon's batch computing service (AWS Batch) allows users to define batch computing tasks, which are a dependency graph of computational jobs, where each job corresponds to a script/executable/container that takes data inputs from stages computed earlier in the graph [12, 6]. AWS Batch allows automatically managed configurations that assess runtime load by monitoring queued jobs and scales up by *allocating additional VM instances* on your behalf within a predetermined maximum limit [7]; VMs are automatically terminated on your behalf as load decreases.

The critical limitation of both of these services is that they cannot *reallocate* resources to an existing VM.

In ***spot markets***, rather than terminating VMs that are outbid by other customers, VMs should be *downgraded* by reallocating a heterogeneous GPU model to the VM; reallocation can be arranged by migrating the VM to a machine with a sufficient capacity GPU. Users have more incentive to bid a higher price since it increases the likelihood they will receive a better GPU when the VM is initially spawned, or even an *upgrade* to a more powerful GPU. Higher bid prices also benefit data center operators, since they ensure a competitive spot market.

Batch computing scales up by allocating entire VMs, but cannot scale up existing VMs. Rather than only monitor the number of queued jobs, AWS Batch could monitor the resource utilization of existing jobs. If a job is computationally bound by the GPU, it will benefit from *upgrading* to a more powerful GPU. By avoiding spinning up new VMs, the data center operator gives the user higher performance at reduced cost, and reserves CPU and memory resources for other users needed for achieving high user-to-machine density.

3 Background

We begin by explaining why GPU reallocation is made challenging by the OpenCL API's assumption that available GPUs and hardware limits remain constant. Next, we describe Crane [14] and how it enabled vendor independent migration between machines with homoge-

neous GPU models, and how we can extend Crane to be a mechanism for GPU reallocation between heterogeneous models.

3.1 GPU hardware limits

A *GPU hardware limit* is a model-specific hardware limit queryable through the OpenCL API, and used to configure an application's execution. The most common GPU hardware limits a user queries are the total GPU device memory, and the amount of data-parallelism exposed by the GPU.

GPU hardware limits complicate reallocation of heterogeneous GPUs, since both the OpenCL API and the applications assume that the GPU remains the same throughout execution. There are three ways in which each GPU hardware limits may change when moving between heterogeneous GPUs.

Downgrading a GPU hardware limit can lead to application failure; if the application expects 4 GiB of GPU device memory, but after reallocation can only use 1 GiB, the application is not written to handle this failure and will terminate. *Upgrading* a GPU hardware limit will not lead to application failure, but the application will remain unaware of additional GPU computing resources available to it, leaving the GPU underutilized. *Equivalent* GPU hardware limits are not negatively affected, since they do not affect application correctness, nor do they lead to GPU resource underutilization.

3.1.1 Data-parallelism hardware limits

While execution throughput varies between GPU models, the data-parallelism hardware limits the OpenCL programmer interacts with when configuring a GPU kernel are *equivalent* even between a consumer-grade GTX 480 and a high-end Tesla K80. Hence, data-parallelism hardware limits aren't an issue when performing GPU reallocation. However, GPU device memory is a hardware limit the programmer frequently depends on that varies widely between GPU models, and so is the focus of this work.

For each GPU model, NVIDIA assigns the GPU a compute capability, which determines the hardware limits the GPU has, including data-parallelism [24]. An OpenCL programmer must explicitly decide two data-parallelism hardware limits when running a kernel on the GPU: (1) local work size, and (2) global work size.

Local work size affects the number of concurrent threads that will run on a compute unit, where the compute unit is capable of running 32 threads in lock-step with the same instruction but on different data (SIMD); this is referred to as the warp size.

Global work size determines the total number of work groups, and should be at least the number of compute units to ensure the GPU is fully occupied [24]. The number of work groups chosen is often a function of the total data input size which is much larger than the number of compute units, so the GPU compute units are usually fully occupied.

For local work size, to ensure compute units are fully occupied and have an opportunity to hide latency if a warp blocks on memory access or synchronization, programmers are encouraged to use local work sizes on the order of 64, 128, or 256 [24]. The local work size limit is the same across most GPU models; the consumer-grade GTX 480 (compute capability 2.0) and the high-end Tesla K80 card offered in GPU VM instances [9, 16, 22] (compute capability 3.7) both have a local work size limit of 1024. The global work size limit is the largest `size_t` available on the platform, which also does not vary between GPU models.

Hence, the data-parallel hardware limits chosen by the OpenCL programmer are *equivalent* across different GPU devices, and are not an issue for GPU reallocation.

In order to avoid the negative affects of *downgrading* and *upgrading* when hardware limits vary between GPUs, Crane must design policies for handling differences in GPU hardware limits.

3.2 Crane

Crane [14] is technique for performing GPU virtualization of OpenCL GPGPU applications that benefits from **Passthrough performance:** allowing VMs with a passthrough GPU to obtain within 5.25% of a passthrough GPU, **Vendor independence:** virtualization is implemented using only OpenCL API commands without any OS or hypervisor modifications, **VM migration:** both stop-and-copy and live migration modes.

The key technique used in Crane to achieve *vendor independence* was to construct a universal GPU state *extraction/reinjection* mechanism built entirely from portable OpenCL APIs. Using this mechanism, GPU device memory could be *extracted* and preserved to DRAM prior to initiating migration, and *reinjected* into the new GPU after migrating to the new host.

Crane focussed on enabling migration between *homogeneous* GPU models. However, with minor modifications that ensure a binary independent extraction format, this technique can be extended to support *heterogeneous GPU reallocation* by reinjecting into a different GPU model.

4 Design

We describe how to extend Crane [14] to support heterogeneous GPU reallocation by hiding binary details of compiled OpenCL programs. Next, we break the design space for GPU reallocation into two approaches. *GPU overcommit* transparently allows GPU reallocation, but may need to reallocate a GPU on-demand for applications that exceed the currently allocated GPU. *Reconfiguration* is non-transparent since it notifies applications to reconfigure their execution to fit the reallocated GPU; this approach benefits from minimizing the chance of VM termination by a cloud operator by maximizing the number of heterogeneous environments in which it can run.

4.1 Enforcing portable OpenCL binaries in Crane

The OpenCL API provides functions for compiling an OpenCL program into a binary. The format of the binary is left entirely up to the GPU vendors [18, 19, 20]. This is a barrier to GPU reallocation, since it prevents Crane from reinjecting binary independent GPU state (Section 3.2). To enforce portable OpenCL programs across heterogeneous GPUs, Crane transparently prevents the OpenCL API from returning vendor OpenCL binaries to the application. Crane forces OpenCL APIs for querying program binaries to return OpenCL source code instead. Correspondingly, Crane forces OpenCL APIs for loading binaries to interpret the "binary" argument as source code, and compile it to a vendor binary. OpenCL applications that cache compiled binaries on disk to reduce startup time can continue to use this programming pattern without application modification.

4.2 GPU overcommit: transparent reallocation

Some GPGPU applications only use a *portion* of the GPU computing resources available to them (Figure 1). For these types of applications, it is practical to provide transparent GPU *upgrade* and *downgrade* functionality.

Since OpenCL applications configure themselves to use a single device once at startup, the only way to provide the option of *upgrading* to a device with more memory is to *overcommit* GPU hardware resources. When the application selects a device and queries for GPU hardware limits, we return a limit *larger* than is currently available on the current local GPU. For example, a local GPU has only 1 GiB of GPU device memory, but a remote GPU has 4 GiB, so we report 4 GiB. When the application attempts to allocate resources beyond what are available on the local host, we can block the OpenCL API request temporarily as we make a decision about how to fulfill request.

There are three options for fulfilling a blocked request for an overcommitted GPU:

- **Live migrate to a new GPU:** migrate to a new host with adequate GPU resources.
- **Block-and-wait:** the OpenCL API call remains blocked while waiting for an available GPU.
- **Suspend-and-wait:** the VM is suspended while waiting for a new GPU.

The best choice depends on: (1) currently available remote GPUs, (2) competing demand from other remote VMs that wish to make use of unused local machine resources.

Live migration is advantageous since a VM can immediately move to available GPU resources when they become available. **Block-and-wait** is advantageous if a GPU is likely to become available soon, or there are other non-OpenCL applications in the VM that can continue to make progress in the meantime. **Suspend-and-wait** is advantageous if relinquishing the current GPU would enable a separate VM to make forward progress using the freed resources (e.g. the GPU, or additional RAM).

Unfortunately, the maximum GPU resource requirements of an application cannot be known without executing it in its entirety. GPU reallocation must estimate resource requirements; incorrect guesses lead to additional GPU reallocations.

4.3 Reconfiguration API: non-transparent reallocation

GPU overcommit is not a practical approach for OpenCL applications that use all GPU resources available to them, since they will immediately trigger migration to a remote host with the most powerful GPU. Instead, these applications must be able to reconfigure themselves to use the maximum capacity available on the local GPU, and advertise that they would *prefer* a larger GPU if one is available.

In the reconfiguration API, OpenCL applications specify *minimum* and *preferred* GPU hardware limits, which allows the cloud service operator to make GPU reallocation decisions on their behalf. Minimum GPU hardware limits ensure that an application is never given a GPU with too few resources for it to execute. Preferred GPU hardware limits provide an upper bound on additional resources an application would benefit from having; additional resources beyond this will go unused.

When a GPU is reallocated to a VM, it notifies the application using callbacks. The extra burden imposed on the developer would be:

R.1 Checkpointing progress: application-specific logic for checkpointing progress made so far.

R.2 Cleanup: releasing existing GPU specific resources.

R.3 GPU configuration: reconfigure the application based on newly available GPUs.

R.4 Resuming execution: resume execution from the checkpointed state.

Developers are already required to write code for both **R.2** and **R.3**, so the changes required will be minimal.

In GPU overcommit, checkpointing OpenCL state (**R.1**), releasing GPU resources (**R.2**), and recreating GPU resources (**R.4**) are handled transparently by Crane. However, unlike GPU overcommit, the reconfiguration API approach maximizes *GPU heterogeneity*, since applications can be told to execute in a smaller GPU than they are currently using. Since applications reconfigure their workload to fit the currently allocated GPU, they can always make forward progress using their current GPU, and never block attempting to use overcommited resources. Maximizing heterogeneity minimizes VM termination by a cloud operator, since the VM can simply be placed elsewhere to run.

5 Evaluation

We extended Crane to support GPU reallocation by leveraging its universal state extraction/reinjection mechanism and support for live migration (Section 3.2), and enforced OpenCL binary compatibility by recompiling OpenCL programs after migrating (Section 4.1). Our prototype is still incomplete, since we must implement GPU reallocation policies needed for cloud scenarios (Section 4.2 and 4.3). However, the x264 GPU device memory requirements (12.4 MiB, Figure 1) are below both cards, making the prototype sufficient for measuring performance before and after GPU reallocation.

In Figure 2, we devise a hypothetical cloud scenario: a cloud service monitoring an x264 workload realizes that the GPU is underutilized, and decides to *downgrade* the workload from an enterprise-grade NVIDIA GRID K1 card (4036 MiB) to a consumer-grade NVIDIA GTX480 card (1474 MiB) with sufficient capacity on a separate physical machine. After the host migrates at 60 seconds in and the lower but sufficient capacity GTX480 is allocated, performance does not degrade; in fact, performance increases because the GTX480 happens to be clocked at a higher rate (1401 MHz) than the GRID K1 (849 MHz).

Each host has two Intel Xeon E5-2609 2.4 GHz quad-core processors, 32 GiB of RAM, and are connected by a 1 Gbps Ethernet link. The hypervisor is Xen 4.7, running CentOS 6 with Linux kernel 3.16.6 in dom0, Ubuntu 16.04 LTS with Linux kernel 4.4.0 in domU. The domU was configured with 2 GiB of memory and ran NVIDIA driver version 361 supporting OpenCL 1.2.

6 Related work

Existing work mentioning movement between heterogeneous devices focusses on *application transparency*. These works do not consider non-transparent approaches that benefit from allowing applications to reconfigure their execution, or the benefits of reallocation in cloud scenarios. Further, existing works either ignore loss of correctness after reallocation, or only allow movement between equivalent feature-sets which defeats the benefit of reallocating heterogeneous hardware.

Process checkpoint/resume: CheCUDA [30] and CheCL [29] integrate with the userspace BCLR framework [13] to allow checkpoint/resume of CUDA and OpenCL applications. The authors note that the checkpoint could be resumed on a machine with a different GPU, but fail to mention how this will affect application correctness after resume. This paper investigates transparent and non-transparent approaches for GPU reallocation that preserve correctness after reallocation.

Heterogeneous operating systems: There have been many works [10, 26, 23] that investigate redesigning existing modern monolithic OSes to better support highly heterogeneous architectures. Helios [23] and M^3 [10] recognize that today's OSes treat heterogeneous devices like a high latency I/O device that lacks fine-grained coordination mechanisms, leading them to a loosely coupled message passing design. It remains to be seen whether these OS designs can support the full fidelity of today's workloads. This paper investigates how to support reallocation of GPUs in a heterogeneous cloud, using OSes and the OpenCL GPGPU programming abstractions of today.

Shadow drivers: Kadav et al. [17] investigate enabling migration through device virtualization, which they call a shadow driver [28], that interposes a state-tracking layer at the driver-to-kernel API interface. The authors use this technique to virtualize NIC drivers, and note that it is possible to migrate between heterogeneous NICs using this approach, but limit it to migrating between identical feature-sets, or force identical feature-sets by masking out additional features. This work investigates allowing both *upgrades* and *downgrades* between heterogeneous devices using application transparent and non-transparent approaches.

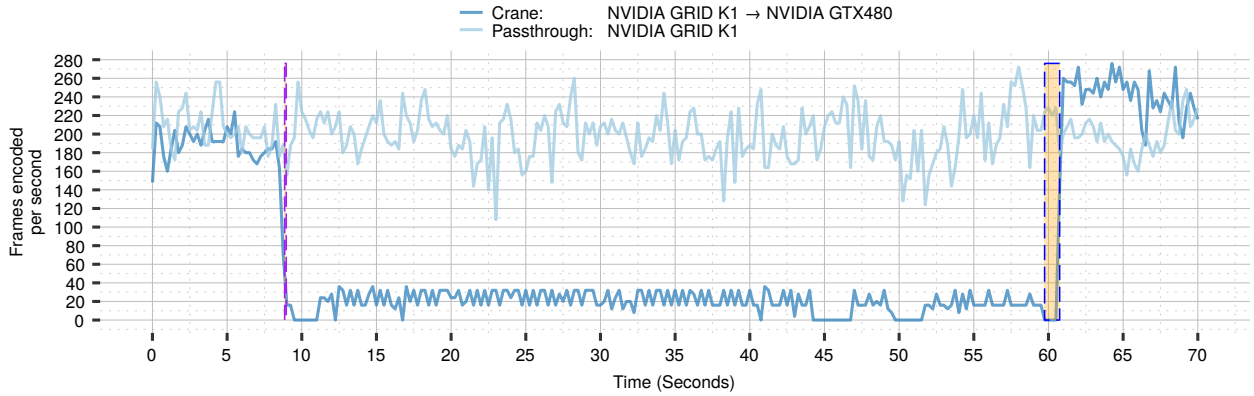

Figure 2: *Downgrading* the x264 workload from a NVIDIA GRID K1 (4036 MiB) to a sufficient capacity NVIDIA GTX480 (1474 MiB) by live migrating to a new host. The cloud operator issues a live migration command 8 seconds into running the application. Boxes denote stoppage before and after live migration [14].

7 Conclusions

In this paper, we show that GPU workloads underutilize GPU device memory, and propose heterogeneous GPU reallocation as a solution for improving idle resource utilization in spot markets and batch computing services. GPU reallocation avoids recomputation costs incurred when services terminate whole VMs. The transparent GPU overcommit approach supports existing OpenCL applications, whereas a non-transparent reconfiguration API minimizes VM termination by maximizing heterogeneous environments in which a VM can run.

8 Discussion topics

Here we cover controversial points and open issues that are important avenues for future work.

GPU underutilization: Our observation that existing workloads underutilize GPUs will be controversial. However, this is only a small sampling of GPU workloads, particularly for OpenCL GPGPU applications. In our future work, we intend to measure GPU workloads characteristic of cloud environments, such as training neural networks for personalized recommendation systems [11].

Extending Crane to OpenGL: We measured GPU device memory usage for popular games to assess GPU underutilization. However, Crane is currently limited to GPU reallocation for OpenCL GPGPU applications. Nevertheless, universal state extraction/reinjection techniques applied to OpenCL can also be applied to OpenGL, and are an important area for future work.

Measuring data centers in the wild: We outlined potential benefits to cloud operators and cloud users, but we have yet to quantify these benefits in cloud services. In particular, we would like to investigate:

- What *performance gain* can a cloud user expect at a given bidding price?
- What *cost savings* can a cloud user expect at a given level of performance?
- What *cluster utilization* can a cloud operator achieve with GPU reallocation?
- What *types of GPU workloads* use only a portion of the GPU, and how many use the entire GPU?
- How do *GPU overcommit* and *reconfiguration API* approaches compare across these GPU workloads?
- What is the *variablity of GPU performance* in heterogeneous GPUs available in today's Amazon EC2 instances? [2]

Heterogeneity beyond GPUs: OpenCL is a generic data-parallel programming API, and is not tied to executing only on GPUs. FPGAs [25] and even custom ASICs [1] are being used to accelerate cloud workloads, with FPGAs now available in Amazon EC2 instances [3]. Altera FPGAs already support a OpenCL-to-verilog compiler that can be used to run OpenCL code originally written for a GPU on an FPGA [2]. So long as different device types are programmed using the OpenCL API, both reconfiguration and overcommit approaches can be used to move between devices such as GPUs and FPGAs.

References

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA* (2016).

[2] Currently, Amazon has NVIDIA Tesla K80, NVIDIA GRID K520, NVIDIA Tesla M2050 available [8].

[2] ALTERA. Implementing FPGA design with the OpenCL standard. *Whitepaper* (2013).

[3] AMAZON. Amazon EC2 F1 Instances. https://aws.amazon.com/ec2/instance-types/f1/.

[4] AMAZON. AWS Batch FAQs. https://aws.amazon.com/blogs/aws/aws-batch-run-batch-computing-jobs-on-aws/.

[5] AMAZON. AWS Batch FAQs. https://aws.amazon.com/batch/faqs/.

[6] AMAZON. Compute Environments. http://docs.aws.amazon.com/batch/latest/userguide/compute_environments.html.

[7] AMAZON. Creating a Compute Environment. http://docs.aws.amazon.com/batch/latest/userguide/create-compute-environment.html.

[8] AMAZON WEB SERVICES. Linux Accelerated Computing Instances. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using_cluster_computing.html.

[9] AMAZON WEB SERVICES. EC2 Instance Types, 2017. https://aws.amazon.com/ec2/instance-types/.

[10] ASMUSSEN, N., VÖLP, M., NÖTHEN, B., HÄRTIG, H., AND FETTWEIS, G. M3: A hardware/operating-system co-design to tame heterogeneous manycores. *ACM SIGOPS Operating Systems Review 50*, 2 (2016), 189–203.

[11] BARCLAY, CHRIS. Orchestrating GPU-Accelerated Workloads on Amazon ECS, 2016. https://aws.amazon.com/blogs/compute/orchestrating-gpu-accelerated-workloads-on-amazon-ecs.

[12] BARR, JEFF. AWS Batch - Run Batch Computing Jobs on AWS. https://aws.amazon.com/blogs/aws/aws-batch-run-batch-computing-jobs-on-aws.

[13] DUELL, J. The design and implementation of Berkeley lab's Linux checkpoint/restart. *Lawrence Berkeley National Laboratory* (2005).

[14] GLEESON, J., KATS, D., MEI, C., AND DE LARA, E. Crane: Fast and Migratable GPU Passthrough for OpenCL applications. In *Proceedings of the 10th ACM International on Systems and Storage Conference* (2017), ACM.

[15] GOOGLE CLOUD PLATFORM. Graphics Processing Units (GPU) — Google Cloud Platform. https://cloud.google.com/gpu/.

[16] GOOGLE CLOUD PLATFORM. GPUs on Compute Engine, 2017. https://cloud.google.com/compute/docs/gpus.

[17] KADAV, A., AND SWIFT, M. M. Live migration of direct-access devices. *SIGOPS Oper. Syst. Rev. 43*, 3 (July 2009), 95–104.

[18] KHRONOS GROUP AND OTHERS. The OpenCL specification–version 1.0, revision 48. *Khronos OpenCL Working Group.* (2010).

[19] KHRONOS GROUP AND OTHERS. The OpenCL specification–version 1.2, revision 19. *Khronos OpenCL Working Group.* (2010).

[20] KHRONOS GROUP AND OTHERS. The OpenCL specification–version 2.2, revision 06. *Khronos OpenCL Working Group.* (2016).

[21] MICROSOFT AZURE. N-Series GPU enabled Virtual Machines. https://azure.microsoft.com/en-us/pricing/details/virtual-machines/series/#n-series.

[22] MICROSOFT AZURE. Pricing - Linux Virtual Machines, 2017. https://azure.microsoft.com/en-ca/pricing/details/virtual-machines/.

[23] NIGHTINGALE, E. B., HODSON, O., MCILROY, R., HAWBLITZEL, C., AND HUNT, G. Helios: heterogeneous multiprocessing with satellite kernels. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (2009), ACM, pp. 221–234.

[24] NVIDIA. OpenCL Programming Guide for the CUDA Architecture, Version 3.2. *CUDA SDK 3* (2010).

[25] PUTNAM, A., CAULFIELD, A. M., CHUNG, E. S., CHIOU, D., CONSTANTINIDES, K., DEMME, J., ESMAEILZADEH, H., FOWERS, J., GOPAL, G. P., GRAY, J., ET AL. A reconfigurable fabric for accelerating large-scale datacenter services. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on* (2014), IEEE, pp. 13–24.

[26] SCHÜPBACH, A., PETER, S., BAUMANN, A., ROSCOE, T., BARHAM, P., HARRIS, T., AND ISAACS, R. Embracing diversity in the barrelfish manycore operating system. In *Proceedings of the Workshop on Managed Many-Core Systems* (2008), p. 27.

[27] SUBRAMANYA, S., RIZK, A., AND IRWIN, D. Cloud spot markets are not sustainable: the case for transient guarantees. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)* (2016), USENIX Association.

[28] SWIFT, M. M., ANNAMALAI, M., BERSHAD, B. N., AND LEVY, H. M. Recovering device drivers. *ACM Transactions on Computer Systems (TOCS) 24*, 4 (2006), 333–360.

[29] TAKIZAWA, H., KOYAMA, K., SATO, K., KOMATSU, K., AND KOBAYASHI, H. CheCL: Transparent checkpointing and process migration of OpenCL applications. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International* (2011), IEEE, pp. 864–876.

[30] TAKIZAWA, H., SATO, K., KOMATSU, K., AND KOBAYASHI, H. CheCUDA: A checkpoint/restart tool for cuda applications. In *2009 International Conference on Parallel and Distributed Computing, Applications and Technologies* (Dec 2009), pp. 408–413.