A Scalable Runtime for Heterogeneous Exascale Machines

P, K, I, D
Queen's University Belfast
University Road
Belfast, United Kingdom
{p.harvey....}@qub.ac.uk

ABSTRACT

Exascale computation is the next target of high performance computing. In the push to create exascale computing platforms, simply increasing the number of hardware devices is not an acceptable option given the limitations of power consumption, heat dissipation, and programming models which are designed for current hardware technologies. Instead, new hardware technologies coupled with improved programming abstractions and more autonomous runtime systems are required to achieve this goal.

This position paper presents the design of a new runtime for a new heterogeneous hardware platform being developed to explore energy efficient, high performance computing. By combining a number of different technologies

In particular, this work seek to explore the use of FPGAs to achieve both the power and performance goals of exascale, as well as appropriate programming abstractions to enable effective use of the hardware platform.

CCS Concepts

•Computer systems organization → Distributed architectures; Heterogeneous (hybrid) systems; •Software and its engineering → Language features;

Keywords

ACM proceedings; LaTeX; text tagging

1. INTRODUCTION

There are more applications than ever that require high performance computing platforms. As the need for more detailed weather simulations, oil field simulations, or smart city applications increases so to does the need to move towards hardware platforms which can perform a billion billion calculations per second: exascale computing.

In order to meet this demand, simply increasing the number of hardware devices is not a scalable option given the relative non-linear scaling of power consumption, heat generation, space requirements, and cost. For example, extrapolating from the top HPC systems, such as China's Tianhe-2, it is estimated that sustaining exaflop performance requires a substantial 1GW of power. Consequently, more power efficient hardware architectures are going to be required.

In addition to the changes required to the hardware platforms, there are also a need to support better programming models, and their associated runtimes, in order to enable developers to take advantage these platforms. Although the default MPI + X programming approach, where X is a programming language of choice, has been and still is an efficient programming option, the low level at which MPI operates is an ongoing barrier to programmers across the spectrum of abilities. Furthermore, as HPC systems are becoming increasingly heterogeneous, using either parallel or reconfigurable hardware architectures (accelerators), there is requirement to move away from semantically broken programming models which are aimed at current computing technologies towards those better able to support developers for these heterogeneous distributed platforms.

This paper outlines the design of a runtime system for a new heterogeneous hardware platform. In order to ease adoption and uptake, this runtime is based on the task and data parallel models found within the OpenCL programming framework, but expands upon this in order to transparently leverage the partitioned global address space (PGAS) model that it provides across a number of devices. The runtime will extend the conventional runtime abstractions to include new abstractions, enabling automated placement of data and computation across a heterogeneous hardware platform, with a prticular focus on FPGAs.

The runtime is being developed as part of the Ecoscale Project[1] and will execute on a new hardware platform being developed in parallel to address the requirements of energy efficient, scalable computing in order to meet the future needs of an exascale compute platform[2]. Additionally, the runtime and the hardware platform are being developed in tandem with a number of industrial applications to provide meaningful use cases to both guide and exercise the work.

Key research questions that we are looking to answer The key contributions of this work are:

- Design of extensions to the OpenCL programming model

[1] www.ecoscale.eu

[2] Ecoscale is part of a trio of projects, all looking to creating the building blocks of an exascale platform - http://www.hpcwire.com/2016/02/24/eu-projects-unite-exascale-prototype/

WOODSTOCK '97 El Paso, Texas USA

DOI: 10.475/123_4

to support automated scheduling of kernels and data across multiple heterogeneous hardware platforms.

- The design of a runtime system to support the scheduling of data and computation across multiple heterogeneous hardware platforms.

The remainder of this paper is organised as follows: Section 2 provides an overview of the state of the art to place this work in context, add more in here

2. BACKGROUND

there is going to have to be a robust discussion of what has already been done to make our point

There have been a number of different approaches to address the creation and execution of HPC applications. In order to establish a context for this work the following is a summery of the related literature, and has been separated in ********* equivalence classes.

2.1 OpenCL

OpenCL is a programming framework for heterogeneous and parallel computing. It is standardised and is managed by the Khronos working group[3]. In OpenCL, users are required to think in terms of host and device code, where a host is a coordinator application on the CPU, and a device is an *accelerator*. An accelerator may be a CPU, GPU, FPGA, or co-processor such as the Xeon Phi [5].

spir/vulkan??

Given the increasing simplicity with which developers are able to program accelerators with OpenCL, it is a prime candidate to explore exascale programming.

2.1.1 OpenCL Configuration

In OpenCL, the host is tasked with setting up, dispatching, and collecting results from a device. OpenCL is accessed through an API, which enables relatively low-level access to data types and functions in order to program and interact with one or more accelerators.

Creating an OpenCL environment consists of first querying the hardware at runtime to determine the available vendor *platforms* and the *devices* available in each platform. Platforms are essentially drivers provided by the hardware vendor, and the devices represent the actual accelerators. Then, a `context` must be created. A context is an umbrella structure that holds the device(s) to be used, as well as other runtime software constructs. A `command_queue` is then associated with each device and placed within the context. A `command_queue` is used to issue commands to a device. Commands include device queries, memory management operations, and kernel (Section 2.1.2) invocations. After this, a user creates a `program` with the kernel source file, and compiles it at runtime. The specific function to be executed within the compiled source is then used to create the `kernel` object. At this point the OpenCL environment has been constructed.

From here the user allocates memory on the device and then copies host data into this memory. The device memory is then associated with the correct position in the kernel arguments. Then, the number of dimensions upon which the kernel should work is calculated, and the kernel is launched on the device, with this information, via the `command_queue`. Usually, the host then blocks attempting to read data back from the device once it has finished its computation. Once all computation is complete and the device is no longer required, there are appropriate destructor functions. The device itself is treated simply as a functional unit. Data and code are passed to the device, the device executes this code, and the results are read back by the host.

[3]https://www.khronos.org/opencl/ - Accessed 5 January 2015

```
1    __kernel void square(__global float* input,
2    __global float* output,
3    const unsigned int count){
4    int i = get_global_id(0);
5    if(i < count)
6    output[i] = input[i] * input[i];
7    }
```

Listing 1: OpenCL Kernel to Compute the Square of An Input Array

2.1.2 Kernels

A device runs a special piece of code known as a *kernel*. An OpenCL kernel is written in a C-like syntax and represents the logic of a single thread. The number and groupings of threads are supplied during the configuration stage on the host. These values are known as the `local` and `global` worksizes, and are used to optimise the allocation of threads to the underlying hardware for a given dataset. Within a kernel, the currently executing thread may be identified via the API. This can be used to customise application logic. The kernel is expressed as a function with parameters. Information for the actual computation is passed to this function as arguments by the host.

The OpenCL model uses a memory hierarchy in which memory is split into `global`, `local`, `private`, and `constant` regions. This is a direct mapping to the hardware configuration of memory found in GPUs, however the same model is applied to all hardware devices. Global memory is shared amongst all threads, local memory is shared between a specified group of threads, and private memory is specific to a thread. Global and local memory are subject to unsynchronised modifications, although there are mechanisms to synchronise access. Constant memory is shared by all threads, but is read only. Listing 1 shows a simple kernel.

2.2 Location Transparent Accelerators

OpenCL was designed for single machines which would contain one or more accelerators. Each of these accelerators would be explicitly interacted with to both manage and execute kernel computations. However, as we mentioned previously, the modern applications require an access to a much larger pool of resources than a single machine can provide. To this end a number of projects have already explored the idea of expanding the OpenCL runtime to harness the resources available in a cluster environment.

SnuCL [6] extends the OpenCL framework by transparently presenting remote devices as though they were local. The OpenCL API (v. 1.1) is largely unchanged, except for a minimal number of optional extensions, with the runtime intercepting all functions and redirecting them as appropriate. MPI [2] is used as the remote communications library with an instance of OpenCL runtime being run on each machine as an MPI node.

Figure 1: The Runtime Architecture of OpenCL

The advantages of SnuCL are: its relative simplicity with respect to the operating model, its ability to run standard OpenCL applications unchanged, and the optimizations implemented to address the problems arising from executing in the distributed environment.

It differs from our approach in that it only supports OpenCL version 1.1[6]. As a consequence it lacks the support of the shared virtual memory (SVM) that the newer versions of the OpenCL standard provide. Furthermore each device must be configured and managed individually, which ultimately limits the scalability of such approach.

VOCL [11] is a similar system to SnuCL. It also implements OpenCL specification 1.1 and equally employs the MPI for the internode communication. It is made of a host node and proxy nodes.

Given its similarity to SnuCL their advantages are quite similar too. Notable differences are that VOCL is positioned as a fully comprehensive virtual framework, with support for such features as a live device migration.

The differences between VOCL and our approach are quite similar to the ones of SnuCL. Whilst it is a step closer to our work in that each device is virtualized, the mapping between OpenCL (soft) device and a hardware device is still one-to-one, which once agains limits its scalability.

Although, it may initially seem a small step to extend a system designed for CPU/GPU to use a CPU/FPGA platform, this is not the case. Specificall there are

2.3 Scheduling of Kernels

FluidCL [8] is a system that dynamically schedules a kernels between a GPU and CPU in order to improve performance. When the kernels are compiled in the host, the runtime will compile the specified kernel for both devices. When dispatched, the kernel computation is split into many smaller computations which are dispatched on both the CPU and GPU at the same time. Each *subkernel* will then read data from each end of a single data set. The idea is that each device will perform as much work as possible, with the more suitable device (CPU/GPU) processing faster and consuming more work. The work experiments with the optimal size of work chunk and notes that it is dependant on whether the kernel is data or compute-intensive.

In order to ensure consistency between the two devices, this work requires that a complete duplicate of the data be present on each device. This is highly inefficient, and for large computations with large datasets, such as those being targeted in Ecoscale, makes this an infeasible approach. Also, the approach requires developers to manually alter their kernel code to include coordination points to synchronise data between the two devices. There is no support for synchronisation primitives. Finally, the authors note that FluidiCL is not designed for distributed computations or devices other than GPU/CPU, and is therefore not appropriate for the hardware described in this paper.

Shepard [7] is a framework with similar goals to this work. Shepard aims to decouple application development from the target platform in systems containing multicore CPU and GPU setups. The assumption is that developers will have access to a standard library of kernels for a range of operations. Developers will group these kerenls into tasks. The compiler more. Based on these costs, the runtime scheduler will decide if the kernel should launched on a CPU or a GPU.

Shepard schedules at the level of entire kernels.

Yan et al. [12] extend the OpenMP [**?**] and OpenACC [10] pragmas to support scheduling entire kernels executions on different devices.

Wen et al. [9] describe a scheduler for OpenCL kernels. The system use machine learning

This work is designed for systems of multiple applications, where as this work is targeting HPC systems which execute a single application. Their work is deigned for systems with GPUs or CPUs, not FPGAs. Also, this system is designed for one or two networked devices, as opposed to a cluster or super computer. However, the runtime described in this paper will attempt to apply a number of these techniques in an HPC setting.

A related and similar system has been developed by Grewe et al. [3]. This approach also uses machine learning, but with a greater emphasis on predicting scheduling of kernels, rather than online learning. This approach schedules entire kernels on GPU/CPU systems.

2.4 HPC Languages

There are a number of different HPC languages, both old and new. cite things: cilk, chapel, fortran, UPC, UPC++

2.5 Data Partitioning

One of the greatest challenges/bottlenecks in computation across different hardware domains is the placement of data.

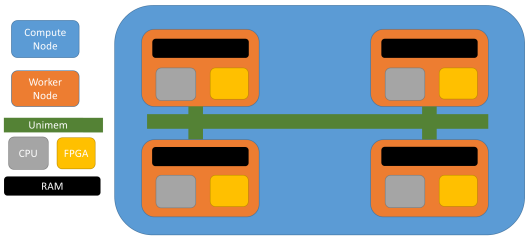

Figure 2: Ecoscale Hardware Unit Architecture add in the elipse to show multiple units, and lines for multicore

This is not an issue for compute intensive tasks which do not use large amounts of input or output data.

Both compute and data intensive applications are present in HPC compute, as shown by most standard benchmarks containing a mix of both workloads. The applications in this work are a mix of compute and data intensive.

Although there are a number of approaches to attempt to automatically classify, most languages leave this task to the developer and provide tools to simplify this action [**?**].

The extensions by Yan et al. [12] to OpenMP and OpenACC also support specifying how data should be partitioned between

attached to loop contrcuts to include patterns which can be applied to the placement of data. Specifically,

2.6 Summary

From the literature review in this section, the following conclusions can be drawn:

- There are no dynamic scheduling approaches for OpenCL kernels which target FPGA hardware
-

Given these conclusions, there is a clear gap for exploration of something...

3. ECOSCALE HARDWARE

As part of the Ecoscale project, a new hardware platform is being developed. An abstracted view of the overall hardware architecture can be seen in Figure 2. The new platform builds on the hardware architecture developed in the EUROSERVER project [1], and will combine low power multicore ARM CPUs, as well as FGPAs. The use of 3D interposer integration enables a high level of compute density. A pair of CPUs and FPGA will be located within a **WORKER** (orange box). Each WORKER will contain around fooGB of RAM. 12 WORKERS will then be located within a **NODE** (blue box).

One of the key elements of the new hardware is that both within and between WORKERs is the UNIMEM interconnect. UNIMEM is essentially a hardware routing fabric that enables either CPUs or FPGAs to access the memory located in a RAM block either locally in a WORKER or remotely in a different worker. These remote direct memory accesses (RDMA) will be done in hardware, removing a number of software layers, thus decreasing latency.

Each of these compute nodes will be networked together using the fancy hardware fabric that iovakis taked about

4. ECOSCALE FRAMEWORK

The Ecoscale framework is a combination of a runtime and language annotations for the OpenCL framework and is being designed for breadth and flexibility. The goal for this runtime is to be able to support a range of different application categories including both traditional compute-intensive and physics applications as well as data-intensive applications.

5. LANGUAGE

From a linguistic perspective, this work will be based on the default OpenCL programming model. As noted in Section 2.1, OpenCL is a popular approach to programming and executing computation mainly on CPU/GPU accelerators. Given this popularity, this choice enables this work to be compatible with existing applications, and require minimal effort from developers in terms of uptake. This work will initially support the framework in the C language, but will be as compatible as OpenCL currently is with other languages.

5.1 Abstractions

Currently, OpenCL provides a software abstraction for an accelerator known as a device, where each of these software devices have a 1-to-1 relationship with the underlying hardware device. The current range of software devices include `DEFAULT`, `CPU`, `GPU`, `ACCELERATOR`. The work intends to extend this device abstraction to include aggregation of multiple devices. Specifically, the following:

- **WORKER** : Abstraction of the devices found in a worker.
- **NODE** : Abstraction of the devices found in a node.

The motivation for these new devices it to enable a developer to submit work to a software device and allow the runtime to best schedule the kernel computation amongst the hardware devices abstracted by the software device. Scheduling is discussed further in Section 6.2.

Additionally, a developer will be able to express the types of accelerator which will be available within a WORKER or a NODE. For example, a user my wish to have only CPUs, or only FPGAs. Equally, the user may wish to express the locality of kernel execution, such as within a `WORKER`. This will be achieved by extending the existing OpenCL API.

The logical conclusion of this approach is to include a `CLUSTER` software device, which would contain `NODE`s. Although linguistically this is an easy goal to achieve, it is not clear if this would be an effective approach. Given that there are many cluster level scheduling tools, such as slurm [**?**], it is not clear if there would be the greatest benefit to not use existing technology to achieve cluster level scheduling. This is an open question and will be explored during the course of the work.

By expanding on the existing approach, this work will provide the user with extra functionality in a way builds upon existing idioms, rather than introducing lots of new syntax and concepts. This also has the advantage of increasing compatibility with legacy code.

5.2 Annotations

As noted in Section **??**, when using a software device which abstracts multiple accelerators it is the responsibility of the scheduler to partition the computation and data

```
1       #pragma eco <device type>
2       __kernel void square(__global float* input,
3       __global float* output,
4       const unsigned int count){
5       int i = get_global_id(0);
6       if(i < count)
7       output[i] = input[i] * input[i];
8       }
```

Listing 2: OpenCL Kernel to Compute the Square of An Input Array

```
1
2    }
```

Listing 3: OpenCL Kernel to Compute the Square of An Input Array

placement as best is can for the given criteria. However, as with some other efforts noted in Section **??**, it should be possible for the developer to provide suggestions to the runtime to assist with these task. To this end this work proposes a limited set of annotations as follows:

The first set of annotations are for the kernel declarations. The purpose of these is to indicate to the runtime system which accelerator that this kernel is most suitable to, as determined by the developer, line 1 in Listing 2. The current options include `FGPA` and `CPU`. Previously OpenCL supported this concept by directly choosing the device to execute the kernel on, however, as a software device now abstracts multiple devices, this feature becomes useful. It should be noted, that one of the main goals of this work to identify the most appropriate accelerator automatically at runtime. This feature is conceptually similar to the `register` keyword in C.

The second set of annotations are to assist in partitioning the data to simplify the process of distributing the data across multiple devices at runtime. Listing 3 shows how the software coordinator can specify how the data should be partitioned.

It is important to note that all of these annotations are optional - i.e. the application will complete successfully without them. Thus, the burden on the developer is non-existent if desired. However, application performance in a number of dimensions will improve with their use.

put in something here or later noting about the outline of the runtime

5.3 Compilation

As the runtime will use OpenCL, kernels will be compiled at runtime for the CPU. To compile for the FPGAs, the Ecoscale project will be creating a custom compiler. As well as taking an OpenCL kernel and generating a bitstream to be encoded onto the FPGA hardware, the compiler will also generate metadata describing the power, performance, and space consumed on the FPGA hardware. To enable the scheduler to optimise for different metrics, the compiler will generate multiple bitstreams per kernel, each optimised for a different metric. These configurations will be collected together in a library and be made available to the runtime. Due to space limitations, details of the FPGA compiler is left to another discussion.

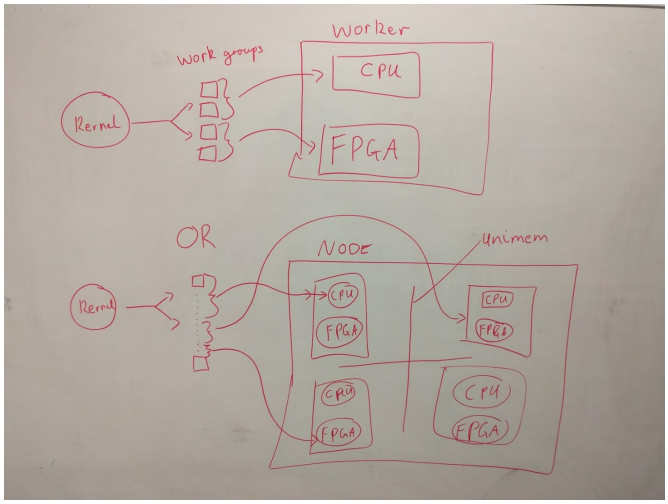

Figure 3: Scheduling of Automatically Partitioned Kernel Invocations

6. RUNTIME DESIGN

6.1 Software Architecture

- manage resources across multiple coherence island
- isolate and protect multiple workloads from each other

As described in Section 3, the hardware platform will consists of several WORKERs located within a NODE. Each worker will run an operating system and contain an Ecoscale rutime. The entire runtime is replicated on each node for robustness, which is discussed further in Section 7. The runtime will be based on the an open source implementation of OpenCL (POCL [4]).

An ecoscale runtime will consist of a communication library, a scheduler, a log of performance metrics of previous executions, access to the FPGA bitstreams and their associated metrics (Section 5.3), and an online machine learning component.

When initialised, the runtime will be pointed to a library of FPGA bitstreams.

6.2 Scheduling

Within this work, the novel hardware architecture presents the opportunity to explore two aspects of scheduling. The first is hierarchical multi-level scheduling, and the second is individual scheduling within an OpenCL device.

Given the myriad contributes that have been made in the area of software scheduling, this work seeks to leverage as much existing technology as possible.

This said, the presence of FPGA accelerators does present a new and novel problem to be solved.

The scheduler will be responsible for the placement of execution and data around the system. Figure 4 shows the architecture of the scheduler.

6.2.1 Preemption

The Ecoscale runtime will use a *run to completion* model. There are two reasons for this. Firstly, although it is possible to provide preemption on FPGAs by time-multiplexing

Figure 4: Architecture of the Ecoscale Runtime

different workloads, the process of context switching is slow, and effect the performance of the execution thus nullifying the performance gains of preemption [**?**]. Secondly, it is the work groups of the kernels which will be scheduled, which have a shorter lifetime that the kernel as a whole. consequently, it will be possible to interleave the execution of different works groups as they complete, without requiring preeption.

we could say something about the benefits of using preemption and that given time we can test the slow down....

6.2.2 *Hierarchical*

The conventional OpenCL framework will schedule the entirety of a kernels execution on a single accelerator. As discussed in Section **??**, there have been attempts to decompose this and schedule parts of a kernels execution on multiple devices, however, this has been limited to single machines with CPU/GPU configurations. The goal of this work is partition a kernel's execution into work groups at runtime, and then distribute these across a number of devices.

6.2.3 *Unit of Schedulability*

Conventional OpenCL deploys an entire kernel execution on a single device. Given that an average OpenCL application has on the order of tens of kernels, this model will not enable full utilisation of a hardware platform with thousands of devices. To do so would require a programmer to manually break kernels into many smaller kernels, and manually schedule then.

Alternatively, this work proposes that in such situations, developers submit kernels to the new software devices proposed in Section **??** and that the runtime partition the kernels into worgroups. It is these work groups that will be scheduled across the different accelerators. This has two main advantages. Firstly, by using a runtime scheduler, the best hardware platform can be found to execute a workgroup for a given invocation. This placement may be different for different invocations. Secondly, the scheduler will be able to schedule multiple workgroups as units, should this be required, whereas this is a non-trivial task to accomplish manually in an application.

Also, workgroups represnt a compromise between scheduling individual work items and entire kerenls. This also has the advantage of being a metric which the programmer can specify at the application level.

The ability to dynamically schedule OpenCL kernel workgroups across heterogeneous hardware platforms is seen as a key research objective of this work.

6.2.4 *Device Level Scheduling*

As noted in section 5.1, Ecoscale will present the developer will be presented with two additional OpenCL device types. As these software devices it will be the responsibility of the runtime to schedule each kernel invocation on an accelerator. In order to do this, there are three primary metrics that will be considered: performance, power consumption, data locality.

Performance will be established in two ways. First, the runtime will monitor the execution of kerenls on different hardware platforms and use this information for future scheduling decisions. Secondly, something about estimating loads based on data input.

Power consumption will also be measured in two ways. As well as monitoring the power consumption on the CPU, for the FGPA, the compiler which generates the bitstreams will also provide power consumption data. As the FPGA is being precise configured, this information will be highly accurate.

Finally, the third metric is data locality. As the scheduler is responsible for placing both the computation and the data, it knows exactly where the information will be for the next kernel. As it is easier for the scheduler to move a few kilobytes of data for the kernel, rather than the gigabytes of raw data for the computation, the scheduler will focus on keeping data in the same physical location for as long as possible. This is similar to the technique used in current GPU programming to reduce the cost of moving data.

something about perf....

Beyond these established metrics, the scheduler will also be optimising for power efficiency and FPGA space occupancy. In order to achieve this, machine learning techniques will be used to automatically detect the most appropriate accelerator for an execution. Given that the goal is to classify each schedulable unit to the most appropriate hardware accelerator, support vector machines [**?**] would be the most sensible approach. SVMs are suited to classifying data sets into two classes. Many SVMs combined can partition data sets into multiple classes. this is crap - word better

Although machine learning techniques have been used previously [3] in conjunction with OpenCL, these have been between a maximum of two devices, where each device was a CPU or a GPU. Given significant operational differences between a GPU and an FPGA, this work is targeting a well defined advancement in the state-of-the-art.

In the first instance, each kernel will be scheduled as an entire unit as is currently done. However, given the average number of kernels found in applications is on the order to tens, and the intended hardware platform will consist of thousands or more of hardware accelerators, this approach does not scale. Therefore, this work proposes to subdivide the work to be done by a kernel into multiple segments in order to take advantage of as many hardware accelerators as possible. This subdivision will be determined automatically by the scheduler at runtime.

6.3 Memory Architecture

As noted in Section 3, the Ecoscale hardware will enable memory located in different worker nodes to be accessed as remote direct memory accesses. This will be most relevant in

In conventional OpenCL applications, memory is an important consideration for the developer. When using an ac-

celerator, the developer must manually handle data movements between the host (usually a CPU) and the accelerator (usually a GPU). This is required as the host and accelerator have different physical memory regions. This data movement is often one of the greatest bottlenecks in terms of performance, and requires the careful attention of the developer.

The first phase of development, where entire kernels are the units of computation, will resemble the existing OpenCL framework.

One interesting aspect of this will be to explore scheduling of computation to minimise data movement. Currently, OpenCL developers will leave data on a device for as long as possible, sharing this data between invocations of different kernels on the same physical device to reduce movement. We intend to automatically have the scheduler perform this type of optimisation, although across multiple physical devices.

7. ROBUSTNESS

As the size of the computation and the compute platform increases, it is increasingly necessary to consider the effects of hardware, network, or software failures, and to provide some method of mitigation. The Ecoscale runtime is designed to address this in two ways.

7.1 Runtime Resilience

The Ecoscale runtime described in Section6 is a centralised system with one worker setup as a master, and the others as slaves. However, each worker will posses a full copy of the Ecoscale runtime meaning that each worker it capable of becoming the controller, even though it is operating as a worker.

It is the responsibility of the master worker to monitor the health of each slave, and to respond appropriately on failure detection. Failure is detected by either explicit failure messages communicated by a worker, such as kernel compilation failure, or implicit via the lack of heartbeat messages from a slave worker.

Given the number of possible failure paths and the space available to describe them, there are two scenarios discussed.

I don't know what i'm talking about here!

7.2 Checkpointing

The most common way to ensure the continuity of data in the presences of errors is via some distributed checkpoint/restart protocol [**?**]. This approach sees snapshots of data and computation being taken at different points of an execution. Should an error occur, the data from the last snapshot of the computation is recovered, and the execution restarted from the last know point. As similar approach will be taken in this work, building on existing efforts in this direction [**?**].

8. CONCLUSION

This paper presents the design for a runtime system to target heterogeneous high performance computer architecture, with an emphasis on power efficiency. By extending and enhancing the existing OpenCL framework, this work will enable the portable performance and ease of programming which is currently offered by OpenCL, combined with the energy efficiency, compute power, and reconfigurable computing offered by the Ecoscale hardware platform. Also, the ability to schedule OpenCL kernels at the level of worgroups across heterogeneous hardware platforms is a key innovation that will enable such OpenCL applications to take advantage of current and next generation HPC platforms.

In this way, our work will explore and realise a power and computationally efficient way to enable high performance computing, which is intended to lay a foundation for an exascale compute platform.

9. ACKNOWLEDGEMENTS

This research project is supported by the European Commission under the H2020 Programme and the ECOSCALE project (grant agreement 671632).

10. REFERENCES

[1] Y. Durand, P. M. Carpenter, S. Adami, A. Bilas, D. Dutoit, A. Farcy, G. Gaydadjiev, J. Goodacre, M. Katevenis, M. Marazakis, E. Matus, I. Mavroidis, and J. Thomson. Euroserver: Energy efficient node for european micro-servers. In *Digital System Design (DSD), 2014 17th Euromicro Conference on*, pages 206–213, Aug 2014.

[2] M. P. Forum. MPI: A message-passing interface standard. Technical report, Knoxville, TN, USA, 1994.

[3] D. Grewe, Z. Wang, and M. F. P. O'Boyle. Opencl task partitioning in the presence of gpu contention. In *26th International Workshop, LCPC 2013, San Jose, CA, USA, September 25–27, 2013.*, 2013.

[4] P. Jääskeläinen, C. S. de La Lama, E. Schnetter, K. Raiskila, J. Takala, and H. Berg. pocl: A performance-portable opencl implementation. *International Journal of Parallel Programming*, 43(5):752–785, 2015.

[5] J. Jeffers and J. Reinders. *Intel Xeon Phi Coprocessor High Performance Programming*. Newnes, 2013.

[6] J. Kim, S. Seo, J. Lee, J. Nah, G. Jo, and J. Lee. Snucl: An opencl framework for heterogeneous cpu/gpu clusters. In *Proceedings of the 26th ACM International Conference on Supercomputing*, ICS '12, pages 341–352, New York, NY, USA, 2012. ACM.

[7] E. O'Neill, J. McGlone, P. Milligan, and P. Kilpatrick. Shepard: Scheduling on heterogeneous platforms using application resource demands. In *Proceedings of the 2014 22Nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, PDP '14, pages 213–217, Washington, DC, USA, 2014. IEEE Computer Society.

[8] P. Pandit and R. Govindarajan. Fluidic kernels: Cooperative execution of opencl programs on multiple heterogeneous devices. In *Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '14, pages 273:273–273:283, New York, NY, USA, 2014. ACM.

[9] Y. Wen, Z. Wang, and M. F. P. O'Boyle. Smart multi-task scheduling for opencl programs on CPU/GPU heterogeneous platforms. In *21st International Conference on High Performance Computing, HiPC 2014, Goa, India, December 17-20, 2014*, pages 1–10, 2014.

[10] S. Wienke, P. Springer, C. Terboven, and D. an Mey. Openacc: First experiences with real-world

applications. In *Proceedings of the 18th International Conference on Parallel Processing*, Euro-Par'12, pages 859–870, Berlin, Heidelberg, 2012. Springer-Verlag.

[11] S. Xiao, P. Balaji, Q. Zhu, R. Thakur, S. Coghlan, H. Lin, G. Wen, J. Hong, and W. chun Feng. Vocl: An optimized environment for transparent virtualization of graphics processing units. In *In Proc. of the 1st Innovative Parallel Computing (InPar*, 2012.

[12] Y. Yan, P.-H. Lin, C. Liao, B. R. de Supinski, and D. J. Quinlan. Supporting multiple accelerators in high-level programming models. In *Proceedings of the Sixth International Workshop on Programming Models and Applications for Multicores and Manycores*, PMAM '15, pages 170–180, New York, NY, USA, 2015. ACM.