Energy-aware Self-Adaptive Middleware for Heterogeneous Parallel Architectures

Richard Kavanagh
School of Computing. University of Leeds
Leeds, UK
Richard.E.Kavanagh@leeds.ac.uk

Karim Djemame
School of Computing. University of Leeds
Leeds, UK
Karim.Djemame@leeds.ac.uk

***Abstract*—Hardware in HPC environments in recent years has become ever more heterogeneous in order to improve computational performance and as an aspect of managing power and energy constraints. This increase in heterogeneity requires middleware abstractions to eliminate additional complexities that it brings. In this paper we present a self-adaptation framework which includes aspects such as automated configuration, deployment and redeployment of applications to different heterogeneous infrastructure. This therefore not only mitigates complexity but aims to take advantage of the existing heterogeneity. The overall result of this paper is a generic event driven self-adaptive system that manages application QoS at runtime, which includes the automatic migration of applications between different accelerated infrastructures. Discussion covers when this migration is appropriate and quantifies the likely benefits.**

***Index Terms*—Self-adaptation, energy modelling, middleware, heterogeneous hardware architectures, application deployment.**

I. Introduction

Advances in distributed computing research have in recent years resulted in considerable commercial interest in utilising heterogeneous hardware architectures (e.g. CPUs, GPUs, FPGAs), with the intent of improving performance and reducing overall power and energy consumption. This heterogeneity of computer systems adds complexity to using the infrastructure that must be managed by the software in order to take full advantage of the available hardware.

Added to this complexity computer systems have faced significant power consumption challenges over the past 20 years. This dual challenge of both power and performance has in recent years shifted from the devices and circuits level, to their current position as first-order constraints for system architects and developers. A common theme is the need for low-power computing systems that are fully interconnected, self-aware, context-aware and self-optimising within application boundaries [1]. Thus, power saving, performance and fast computational speed are key requirements in application development. A key aspect for any future system is therefore to abstract away complexities of the heterogeneity and to support power and energy awareness through automatic reconfiguration at the level of the application. In doing so this handles the impact of heterogeneity which is rapidly increasing, the need for innovative architectures, algorithms and specialized environments to efficiently use these new and mixed/diversified parallel architectures. In this paper we address these issues with a energy-aware self-adaptive framework for heterogeneous parallel architectures. The main contributions of this paper are:

- a software framework for the adaptation of jobs on heterogeneous HPC infrastructure
- an algorithm for the redeployment of jobs onto alternative hardware configurations
- guidance on the redeployment of jobs on heterogeneous hardware.

The remainder of the paper is organised as follows. In Section III we present the overall architecture that supports energy awareness and self-adaptation. In Section IV, the middleware and self-adaptation engine is discussed. In Section V the experimental setup is outlined, followed by experimentation and evaluation in Section VI. The related work is then presented in Section II and Section VII summarises the research and provides plans for future work.

Work here was supported by the European Union's Horizon 2020 research and innovation programme under grant agreement RIA 687584.

II. Related Work

Due to increasing systems complexity in recent years, there has been a trend towards self-adaptive systems (SASs) to address issues such as maintenance, configuration and Quality of Service (QoS) compliance in such complicated environments. Self-adaptation requires the answering of fundamental questions of "When, Why, Where, do we have to adapt?", "What kind of change is needed?", "Who performs the adaptation?" and "How is the adaptation performed?" [2].

Krupitzer et al. [3] presents a taxonomy of self-adaptive systems and their inspiration, while R. deLemos et al. [4] identifies research challenges when developing, deploying and managing self-adaptive software systems. These challenges result from the dynamic nature of self-adaptation, which brings uncertainty. Adaptation commonly follows a Monitor Analyses Plan and Execute (MAPE) [5], [6] approach. An extended architecture of the MAPE-K loop as a reference model for the design of self-adaptive systems is found in [7], assuming that the system has a central controller with a central MAPE-K loop. The proposal consists in continuously evaluating adaptation steps concerning their actual effect and adaptation mechanisms concerning their applicability and efficiency in the case of topology changes.

Research effort has focused on the exploitation of hardware accelerators in cloud computing environments by addressing the challenge of programming such systems and making them easily accessible in a virtualized environment. One common approach is propose methods to offload computations on heterogeneous hardware components. A solution for the efficient exploitation of specialised computing resources of a heterogeneous system is found in [8]. Other works have proposed heterogeneous architectures that combine high-performance and low-power servers in order to achieve better overall energy proportionality and energy efficiency [9]. The mapping problem between compute resources and application configurations is explored in [10], considering throughput in the context of cloud. This is similar in idea to the work presented in this paper, though the exact context and approach differs regarding the heterogeneity that is being utilised. The ASCETiC project [11] holds similarities in that it worked upon energy efficiency and software adaptation but in the context of clouds. A model for developing applications, exploiting hardware heterogeneity in cloud data centres while considering the aspect of energy efficiency is presented in [12]. In this model applications are expressed as interconnected microservices which are automatically scheduled for execution on the most suitable heterogeneous computing elements. This leaves the open problem of handling of applications that do not follow a microservices pattern such as in HPC environments.

The Legato project [13] identifies power as a key concern and while software-stack support for heterogeneity is relatively well developed for performance, is seen to remain an open question for power and energy-efficiency, which is an aspect that this paper contributes towards. StarPU [14] is one such early work that uses abstractions to allow workloads to be placed upon various different accelerator based platforms, selecting between various different accelerators, to improve computational performance and not energy. EcoScale [15], [16] uses hardware performance monitors and models to project runtime and power consumption in heterogeneous environments, with the aim of dynamically selecting and distribute software functions to either hardware acceleration or to software based execution. This paper in comparison to existing literature brings together key aspects of performance, QoS and energy management, with application migration on heterogeneous architectures. The Antarex project [17], [18], focuses on energy efficient systems and in particular on producing a tool chain that tunes code to run efficiently on heterogeneous infrastructures. The Manago project [19] is equally similar to work presented in this paper but with a particular focus on time-predictability with trade-offs with power and energy efficiency. Dutot et al. [20] advance upon power-capping and consider energy budgets with a principle focus on scheduling.

From a technology viewpoint hardware accelerators, such as GPGPUs and FPGAs still need the use of power reduction techniques such as Dynamic Voltage and Frequency Scaling (DVFS) and partial reconfiguration for FPGAs to keep power consumption under control [21]. Many approaches to adaptation and energy/power optimisation concentrate at the hardware level, such as utilising task scheduling coupled with GPU-specific DVFS and dynamic resource sleep (DRS) mechanisms, as a means to minimise the total energy consumption [22]. Our work in this paper compliments such hardware based strategies given the similarity of goals, yet utilises software based approaches to minimise power and conserve energy.

III. Architecture

The architecture's (Figure 1) aim is to control and abstract underlying heterogeneous hardware architectures, configurations and software systems.

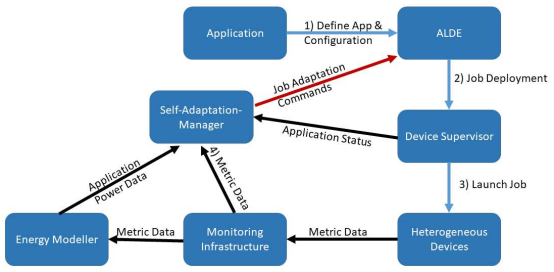

Fig. 1. Architecture

The overall flow in the architecture starts with *Application Life cycle Deployment Engine (ALDE)*. It manages the overall lifecycle of an application. It introduces for each application the entities: Executable (a specific implementation), Execution Configuration (resource requirements and setup). The ALDE submits jobs to the *Device Supervisor* which provides scheduling capabilities during application deployment and operation, where it maps tasks to appropriate Heterogeneous Parallel Device (HPDs). The *Monitor Infrastructure* provides metrics along with historical statistics for the devices and applications (e.g. power, energy consumed and performance). The *Self-Adaptation Manager (SAM)* during the operation phase manages the runtime based adaptation strategy applied to applications and HPDs (see Section IV-A). This includes aspects such as initiating redeployment to another HPD, restructuring a workflow task graph or dynamic recompilation. Furthermore, the component provides functionality to guide the deployment of an application to a specific HPD through predictive *energy modelling* capabilities and polices.

IV. Adaptation Framework

A. Self-Adaptation Management

The SAM's principle role is to manage adaptation at runtime, managing trade-offs between energy, power and performance within the framework. It is event driven, deciding for each event what adaptation to take and where it should be applied. It works through a series of listeners that monitor the physical infrastructure, the jobs that are launched and the system clock for cron based events. This therefore requires interaction with the *Monitoring infrastructure* (for system and application based metrics) as well as the *Device Supervisor* and *ALDE* (for application based information). The listeners act as triggers generating events which through a sequence of rules then map to actuators that perform the required adaptation.

Events - The first step in adaptation is a notification event which derives from the listeners. Events principally contain the following information:

- *Time:* the timestamp of the event.
- *Value:* a raw value representing how large the QoS breach is, i.e. the measured value of the violation.
- *Event Type:* This is either a "violation" if the violation is detected, a "warning", or an informative indicator such as a event driven by the system clock has occurred.
- *Agreement Term:* the metric to be monitored.
- *Guarantee Id:* an identifier for each QoS constraint.
- *Operator:* such as greater than, less than, equal.
- *Guaranteed Value:* the value of the threshold.

Events (Host, Application and Clock) dependent upon their source must contain additional information. *Host* events additionally must contain the hostname, thus indicating the events' origin. *Application* based events must additionally record the application's name, a reference to the exact application instance and a reference to any application configuration information and specific firing rules as defined by the ALDE. *Clock* events, must hold a map for additional settings. This allows clock events to mimic host or application based events, facilitating features such as un-pausing an application after a set period of time. Events such as the following have the potential to trigger adaptation:

- *Boundary conditions on measurements:* provide a reactive response to a QoS breaches, by setting constraints on application and host metrics.
- *Idle host detection:* enables responses such as increasing application's resource utilisation or switching off under-utilised resources. Enhanced with accelerator detection, it can discover opportunities for redeploying and reconfiguring applications.
- *Host's failing/failed or in drain state:* allows for self-healing, where applications can be reconfigured and redeployed on the remaining infrastructure. Draining hosts of existing jobs can be sped up.
- *Applications approaching deadline:* This allows applications to be check-pointed close to completion in order to preserve work before eviction.
- *Application starting/completion:* Useful to constrain execution to set times of the day, ensuring power hungry applications with low QoS requirements can be launched as required but run later.
- *Cron based events:* create triggers based upon schedules, increasing flexibility, e.g. events such as un-pausing jobs at a set time later on.

On event notification the SAM works in *two phases*. The first considers the mapping between the type of notification and the actuators to use essentially the type of adaptation to make such as: redeploying an application to use accelerators or pausing an application. The second phase indicates the exact nature of this adaptation to take such as which application should be adapted and by how much?

Adaptation Rules - The first phase utilises adaptation rules that can be specified as a tuple of: $\langle$Agreement Term, Comparator, Response Type, $\{$Event Type$\}$, $\{$Lower Bound$\}$, $\{$Upper Bound$\}$, $\{$Parameters$\}$ $\rangle$ which is utilised to determine the form of adaptation to take. Two examples of this are:

$\langle$IDLE_HOST+ACCELERATED,EQ,
RESELECT_ACCELERATORS $\rangle$ and
$\langle$ IDLE_HOST+ACCELERATED, EQ,
RESELECT_ACCELERATORS, WARNING, 0, 0,
KILL_PREVIOUS=TRUE ;application=gromacs $\rangle$.

The latter optional values allow for stronger granularity ensuring the adaptation behaviour considers the scale of the notification event, providing the flexibility to do things such as:

- Responding to warnings, in a different fashion to breaches or informative notifications.
- Observe the difference between the guaranteed value and the measured value and providing a stronger response if the deviation is further away (i.e. the lower bound and upper bound values).
- Parametrising the rules, so applications can further indicate how adaptation should occur e.g. clock based events such as "it is out of working hours" can specify through parameters application information, thus allowing lower priority jobs to run.

Application and resource based events, derived from measurements utilise a threshold value, which determines how many events are required before a rule fires. This ensures that the temporary reporting of minor breaches can be ignored (e.g. if power consumption goes too high due to a short burst of CPU utilisation).

Once a rule has fired a recent history log prevents the same rule firing in rapid succession, thus avoiding over adaptation. After a short configurable amount of time (e.g. last minute), the rule can then be re-fired. The rules can optionally be set into a hierarchy so that if one rule cannot be applied additional rules that match the criteria may be used instead. This generates the prospect of either having fall back options for adaptation or an intensification of the adaptation response.

Decision Engines - The second phase involves the usage of a decision engine that decides upon the location and scale of adaptation. This considers various parameters, such as the application configuration, QoS goals and the current environment to achieve this. Decision engines handle cases where information is lacking on how to adapt including cases such as host based events and their transformation into actions applied to applications. The transformation process for host based events can be achieved: *randomly*, based upon the applications power consumption, or based upon the *last application instantiated* on the originating host. Clock events can be transformed into either host or application based events dependent upon the additional parameters attached to the event. They are transformed in order of precedence by:

1) Event data with *application* details attached, which can happen for example when a pause action has specified when to resume.
2) Event data with *host* details attached. Similar to above but originates from a host based events instead.
3) The decision rule contains the *host* or *application* data.

Actuators - The actuators that are available to be utilised are:

- *Increase/Reduce Wall Time:* Increases or reduces the wall time by a fixed increment as specified in the rule-set.
- *Minimize Wall Time of Similar Apps:* Sets wall times closer to the average job completion time, aiming to aid back-filling by the device supervisor.
- *Pause/Unpause App:* This pauses an application, optionally can trigger unpause actions after a set period of time.
- *Pause/Unpause Similar Apps:* pauses/resumes many similar applications at once, for example if they are low priority and a power cap needs to be achieved.
- *Oversubscribe App:* allows pending jobs to be scheduled alongside each other on the same physical host.
- *Exclusive App:* ensures pending jobs are exclusively scheduled to physical hosts.
- *Reselect Accelerators:* This examines the configurations available to run an application and redeploys the application where necessary to improve on the current deployment's configuration (e.g. less energy, lower power, faster completion time).
- *Kill App:* This terminates a given application
- *Kill Similar Apps:* This kills a series of instances of the same application
- *Startup/Shutdown Host:* Actions to start and stop physical hosts
- *Increase/Reduce Power Cap:* This adjusts the power cap of the cluster by an incremental amount.
- *Set Power Cap:* sets the power cap to a defined value.

One actuator stands out as being more complex than the rest, "Reselect Accelerators" (see Figure 2). Its primary aim is to choose an application configuration that is better than the existing configuration. This may for example be switching from a single threaded CPU bound executable to a GPU accelerated version of the same application. This could be done to improve the accelerator utilisation. The algorithm firstly filters out configurations that are already running and thus have a head start upon any new instance starting. The second phase in the algorithm selects the new instance to launch. This works by ranking each configuration by either, power, energy or completion time. The best configuration is then selected so long as its power/energy or completion time is better than the existing running configuration. This ranking is performed based upon pilot jobs that are executed beforehand. The pilot jobs have a fixed workload (or a sequence of workloads that is repeated uniformly against each configuration), ensuring that each application configuration is compared fairly. This comparison gives a relative ranking between the configurations based upon the current hardware setup. It is considered that each application configuration has a relative affinity to each of the available resources on the testbed, therefore if the pilot jobs are repeated several times the likely improvement between configurations is going to be realised. This process enables the ratio of improvement between the configurations to be determined. This includes aspects such as the likely energy consumption and average power consumption for running a pilot job or job (by relative ratio between configurations), which reflects complex aspects such as which resources a particular job was submitted to.

B. Energy Modelling

Adaptation inside the framework requires guidance, one such aspect regards application power and overall energy consumption. The self-adaptation manager needs to know the likely consequences of its actions in regards to aspects such as the power and energy consumed by an application or physical host. Application power consumption cannot directly be measured and is synthetic in nature, based upon attributing power consumption to an application dependent upon workload. Adaptation of applications based upon power consumption therefore requires a model to attribute this power.

The energy modeller (EM) [23] considers the major power consumers such as CPUs and other accelerators. In order to do this it has various models that may be used to attribute power consumption to an application. Two models have been specifically designed for physical hosts with accelerators, namely the `CpuAndAcceleratorEnergyPredictor` that utilises neural networks to apply a fit to the available calibration data and the `CpuAndBiModalAcceleratorEnergyPredictor` that determines power usage of an accelerator assuming an unutilised and heavily utilised state. The latter adaptor being useful in cases where the quality of calibration data is poor which causes calibration to fail, yet it still offers an estimate that can give a guide to any adaptation.

The `CpuAndAcceleratorEnergyPredictor` works as an additive model in which the CPU and the accelerator's utilisation is considered separately. The CPU is considered as polynomial fit of order 2, this has been chosen because if the model turns out to be linear then it will still provide a good fit, yet offers flexibility in cases where pure linearity does not hold [23].

The accelerator based calibration is written in such a way as to be as flexible as possible. It utilises a multilayer perceptron network with a single hidden layer. The amount of inputs is based upon the size of the calibration data gathered providing a single output. The size of the hidden layer is scaled to be $\sqrt{inputsize + outputsize}$, this ensures its size is sufficient but not so large as to cause it to be overly trained. The emphasis is therefore placed upon gathering training data of sufficient quality for the network to train correctly, ensuring that the parameters chosen have sufficiently strong influence on the power consumption.

The second predictor `CpuAndBiModalAccelerator EnergyPredictor` also works in an additive fashion, with

```
public void reselectAccelerators (String appName, String deploymentId, boolean killPreviousApp,
    RankCriteria rankBy) {
        AppConfig currentConfig = getCurrentConfigurationInUse(appName, deploymentId);
        ApplicationDefintion appDef = getApplicationDefintion(currentConfig);
        \\a check on if resources are available and the executables required are compiled
        AppConfig validConfigs[] = getValidConfigurations(appDef);
        validConfigs[] = removeAlreadyRunningConfigurations(validConfigs[]);
        selectedConfiguration = selectConfig(validConfigs[], appDef, currentConfig, rankBy);
        startAndStopNewAndOldJobs(selectedConfiguration, appDef, currentConfiguration);
}

private CurrentConfig selectConfig(AppConfig validConfigs[], ApplicationDefintion appDef,
        AppConfig currentConfig, RankCriteria rankBy) {
    sort(validConfigs[],rankBy);
    If(first(validConfigs[]).isRunning()) {
        \\ensures a running configuration is not restarted, as its already made progress
        return null;
    }
    If(compareRankToCurrentInstance(first(validConfigs[]), currentConfig) {
        return first(validConfigs[]); \\ensures new configuration dominates previous config
    }
    return null; \\no better solution so exit
}
```

Fig. 2. Reselection of Accelerators Algorithm

two sub models, one for the CPU and another for the accelerators in use. The CPU sub-model uses the same polynomial model as the `CpuAndAcceleratorEnergyPredictor` model. This accelerator model performs clustering assuming two distinct states, one at the higher end of usage while the accelerator is active and another assuming the accelerator is idle. This model acts as an approximation for situations when the accelerator is only partially observable, has limited training data, or when the training data for accelerator utilisation does not correlate well to power consumption. An example of limited observability and correlation is with Nvidia GPUs, whereby the clock frequency of a stream multiprocessor (SM) may be used as a substitute for utilisation. This due to Nvidia-smi's utilisation value reporting the percentage of time in a given interval where at least one SM is active [24], which has limited direct correlation to power consumption. Alternative routes such as application profiling which provide performance counters for each application remain impractical, given overheads and requirements to attach to every application running.

V. Experimental Design

To evaluate the feasibility of the adaptation features as outlined in Section IV, the experimentation is presented in the context of the energy efficient HPC environment presented in Section III as implemented by the TANGO project [1].

The objective of the experimentation is to ascertain if the self-adaptation when monitoring applications in operation achieves dynamic energy management from the middleware of a HPC software stack. First an outline of the experimentation is given followed by the testbed and application setup.

The experimentation centres around the prospect of hardware becoming available, this might be the completion of another job for example. This presents the opportunity to trigger adaptation and redeploy an application in order to obtain an improvement and may include changing accelerators in use. A similar scenario is the loss of resources, such as a node failure, whereby the reselection process for jobs may be required.

The experimentation was performed on a nova S5 cluster, using a subset of a bullx blade system. The testbed was composed of the following heterogeneous hardware resources.

4 bullx 515 nodes equipped with: 2 Intel Xeon E5-2470 (Sandy Bridge) at 2.3GHz, 12 X 16GB DDR3-1600 ECC SDRAM and 2 X 256GB SATA3 flash SSDs. Additionally two nodes ns50-51 with 2 Nvidia Kepler K20X GPUs each and nodes ns52-53 with 2 Intel Xeon Phi 5100 series (rev 11) KNC each. In addition to these nodes there are: 3 bullx B520 double compute blades (ns55-57), each equipped with: 2 Intel Xeon E5-2690 v3 (Haswell) at 2.6GHz with 16 X 16GB DDR4-RDIMM 2133DDR and 2 X 256GB SATA3 flash SSDs.

The experimentation utilises the GROMACS (http://www.gromacs.org/) application, an open source and widely utilised molecular dynamics simulation package. It is used to generate load within the testbed and provides a realistic application that can be compiled into various alternative implementations such as Message Passing Interface (MPI) and CUDA.

VI. Evaluation

The following section discusses the performance of the self-adaptation presenting an analysis of the experimental results.

This experiment illustrates the use of multiple application implementations of HPC applications in order to deploy, monitor and adapt an application so that the most efficient implementation is executed, given the resources that are available at the time of execution. The workflow, as shown in Figure 1, is as follows:

1) The Gromacs application is defined in the ALDE and the configurations available

2) Job Deployment, A CUDA instance is started
3) Launch Job, The device supervisor launches the job onto the infrastructure
4) The SAM receives an event indicating a host has become free with an accelerator
5) The SAM compares Gromacs implementations and relaunches the most efficient version of the application
6) The Gromacs application completes

In the following experimentation two different versions of the Gromacs mini-app are prepared one using MPI (using 16 threads) and the other CUDA. For each configuration, previous pilot runs are performed where execution time and energy consumption are measured for a given fixed workload. This presents a relative ranking of each configuration of the application upon the available hardware. This ranking of each application configuration will have an affinity towards a particular set of resources. A CUDA job for example must be launched on a subset of resources that have GPUs. The following table show these initial measurements, where we can see that depending on the configuration, the Gromacs simulations can achieve different performance and energy consumption. The most efficient configuration in terms of time and energy for this execution is using the MPI implementation of Gromacs. The variance in execution time and energy consumption is low, in this case, making it particularly suitable for determining speed up between configurations.

TABLE I
RUN OF PILOT JOBS TO DETERMINE POWER, ENERGY AND COMPLETION TIME RANKINGS

Name	Run Count	Total Energy (all runs)(J)	Average Energy (J)	Total Time	Average Time Per Run (s)	Average power (W)
cuda	3	22,835	7,611.67	87	29	**262.47**
mpi 16	3	**19,217**	**6,405.67**	**67**	**22.33**	286.82

What can be seen is that the MPI application has the lowest energy consumption overall at 6,405.67J per run. This is principally due to the lower runtime of the MPI application, 67s as compared to 87s. The average power consumption of the CUDA application is however less. This set of configurations therefore offers either:

a) A lower power consumption that runs for longer and consumes more energy.
b) A higher power consumption that runs for a shorter period of time and therefore uses less energy.

These results give an indication of how quickly a replacement replica should start in order to consume less energy overall, assuming the overall application workload is similar to that of the pilot jobs. To find the relative rank between configuration options requires each application configuration to remain proportional in its energy usage to the other potential configurations. In the case of the pilot job executions shown in Table I, an MPI implementation could be started in the first 4s of the CUDA instance's execution and still use less energy. This is derived from:

avg(cuda_run_energy) - avg(mpi_run_energy) = Δenergy_between_run_types
7611.67 - 6405.67 = 1206J
Δenergy_between_run_types / avg(mpi_run_power) = migration exploitation window size
1206 / 286.82 = 4.02s

Given the executions above are short lived the benefits of restarting jobs with different accelerators is limited, however if the workload is increased then the migration exploitation window size will also increase. Practically as this mechanism only provides the relative ranking between application configurations, without a priori knowledge of the runtime (to work out scale differences from the pilot jobs), then it is possible to use recent job submissions as guidance on current expected execution durations. The MPI and CUDA jobs both linearly scale (so far as tested), leaving the ratios between their durations and energy consumption comparable. However, due to gradient differences causing divergence (Figure 3, CUDA increasingly uses more energy in comparison to the MPI implementation, therefore to save both energy and time, larger jobs should favour the MPI implementation. The migration exploitation window size, scales linearly with the job size.

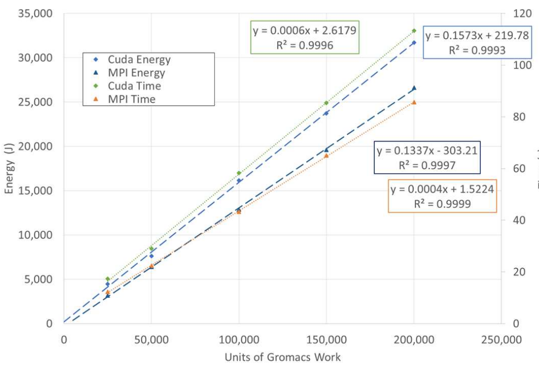

Fig. 3. CUDA vs MPI Job Workload Scaling

In addition applications using this calculation may be ranked to consider which ones have the most difference between the available configurations options. This therefore finds the application which is most likely to benefit from adaptation.

To illustrate this a rule is added that causes the detection of idle resources with accelerators. The SAM therefore considers if any Gromacs instance may be accelerated. The rule used is: $\langle$IDLE HOST ACCELERATED, EQ, RESELECT ACCELERATORS, WARNING, 0, 0, KILL PREVIOUS=TRUE;application=gromacs $\rangle$ which follows the format: $\langle$ Agreement Term, Direction, Response Type, Event Type, Upper Bound, Lower bound, Parameters $\rangle$.

The reselect accelerators actuator compared the deployment options available and found it was possible to execute an instance that consumes a lower amount of energy. The sequence of events generated is therefore that an IDLE HOST ACCELERATED event triggers the adaptation, which terminates the CUDA instance and launches a new MPI instance. The result of this is shown in Figure 4. The CUDA job runs on node ns51 and is then replaced by an MPI job on node ns55. Followed by the MPI job then completing at 27s, 2 seconds earlier than the CUDA job could have been expected to complete. Although this is a small benefit 6%, with longer running jobs the benefit is likely to be more substantial.

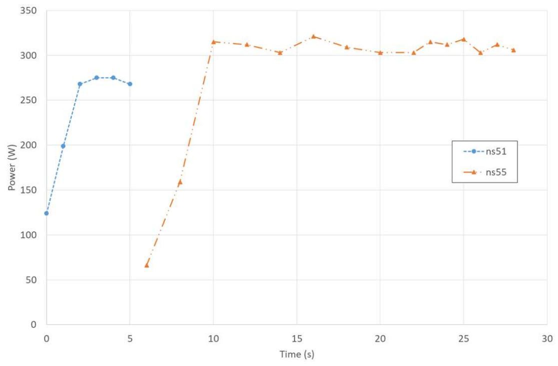

Fig. 4. MPI Job Launching and Cancelling CUDA Job

This job redeployment mechanism allows for multiple possibilities, when considering events that the SAM will act upon. For example job or node failure could cause the job redeployment to the most appropriate available resources, thus acting as a recovery mechanism. If ranking by power consumption it would aid power capping mechanisms by ensuring the mix of jobs running is most likely to have a sufficiently low power consumption. This mechanism also ensures accelerators where appropriate are more likely to be utilized.

In Figure 3 it seems counter intuitive that the CUDA implementation given the vast parallelism of the GPU does not perform better than the MPI implementation. This results in the transfer of the job from ns51 to ns55 in Figure 4. The difference is caused by the overall efficiency of the host. The explanation can be seen in Figure 5, which shows the energy consumption of a series of Gromacs jobs, both on ns51 (default choice for GPUs) and ns55 (default choice for MPI jobs), along with MPI implementation on ns51. We omit here the graph for job completion time for purposes of clarity though it is similar to Figure 3 and 5.

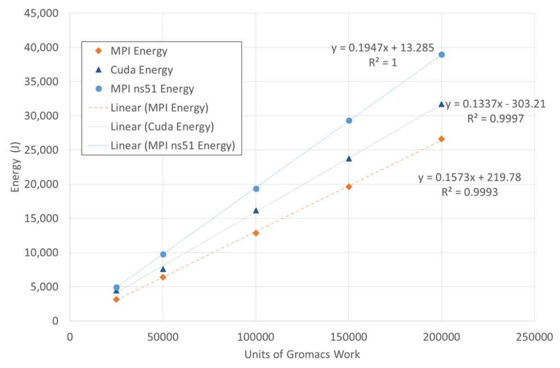

Fig. 5. MPI vs CUDA on Multiple Host Types

The CUDA implementation is more efficient than the MPI implementation of ns51, however the MPI implementation may also run on ns55 which is generally more efficient than ns51 due to CPU generational differences, such that the MPI implementation is the best choice should the more efficient nodes be available.

The strategy employed here to determine the relative ranking of jobs has many considerations that must be made, which are going to be split into categories and discussed in turn. In terms of comparing various different configurations of an application then there are two key aspects: temporal and power.

Job duration is important, simply the longer it runs the greater the energy consumption. However ranking of jobs may be subject to changes in throughput over time for a given configuration, dependent upon the size of the workload (Figure 3 and 5). This can be checked with a series of jobs of different sizes and examining if the application scales under workload changes. If an application scales in a predictable fashion, recently completed jobs can be used as a guide to estimate both completion time and the migration exploitation window size. Thus offering an estimate of the likely speed up/energy saving between configurations. One strategy of tackling throughput changes would be to make pilot jobs of a similar size to "typical" jobs thus mitigating scaling differences.

The second aspect is the power consumption of a job and if its power consumption is consistent (with obvious exception of the start and end of jobs). In the case of Gromacs (MPI) on ns55 the power consumption is consistent in the range of 303-320W with an average of 310W excluding earlier lower power consumption at the start of the applications execution. If it varies through time, the following questions can be considered. Does it act in phases with any particular recognisable fashion? and more importantly does the average power consumption stay the same with job length?

Applications may be made up of several stages. These from the perspective of ranking configurations only need be considered in cases where the average power consumption varies or the duration varies because of some underlying change in actions. Examples of this might include transferring data vs compute work, or different types of compute work on different accelerators/instruction sets. The steps may change size, altering average power consumption (e.g. in cases where a larger input file is needed). Data transfers may alternatively act as bounding behaviour on the power consumption, for example streaming where only a certain amount of power consumption can be achieved during transfer.

An additional aspect is the heterogeneity of the hardware. An application's configuration is unlikely to specify specific hardware, but it may provide an affinity to a given subset of hardware or explicitly exclude some resources. An example of this is a CUDA job that needs a GPU. It may be for a given job configuration that the resources available to use are not very heterogeneous in terms of power and compute capacity (i.e. temporal/duration of job given the same workload). It is likely that this affinity of a given configuration towards different subsets of the infrastructure can be discovered by several runs of a given pilot job. Thus giving an average case for the likely speed up between configurations can be discovered.

This strategy has its limitations and is dependent upon other workloads running at the same time, as well as the scheduler in use and size of the job to be rescheduled. The variance in time and energy consumption can be considered a measure of the reliability of the reselection method. The variance would be reduced if the pilot run dataset could be filtered to records only considering the types of resources available.

VII. Conclusion

This paper has shown a self-adaptive framework for heterogeneous hardware, including more specific aspects such as the automatic redeployment of applications and power capping. The criteria for selecting an application for migration has been discussed, with particular focus on a window of opportunity for migration. Opportunities for reducing power and energy consumption are shown to be further complicated by trade-offs between GPU acceleration and host CPU architectural versions, thus ensuring use of accelerators is not always best. In a wider context of self-adaptation mechanisms have been demonstrated that can aid power capping behaviour and raise the intelligence of such an approach from the hardware to the middleware/application level. In future work this will be extended to include check-pointing as part of migration along with extending adaptation into domains such as Internet of Things.

References

[1] K. Djemame, et. al., "Tango: Transparent heterogeneous hardware architecture deployment for energy gain in operation," in *Proceedings of the First Workshop on Program Transformation for Programmability in Heterogeneous Architectures*, S. Tamarit, G. Vigueras, M. Carro, and J. Marino, Eds., Barcelona, Spain, March 2016.

[2] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, no. 2, p. 14, 2009.

[3] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker, "A survey on engineering approaches for self-adaptive systems," *Pervasive and Mobile Computing*, vol. 17, pp. 184 – 206, 2015.

[4] de Lemos, Rogério, et. al., *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–32.

[5] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, jan 2003.

[6] IBM, "An architectural blueprint for autonomic computing," p. 34, 2005. [Online]. Available: http://www-03.ibm.com/autonomic/pdfs/AC Blueprint White Paper V7.pdf

[7] V. Klos, T. Gothel, and S. Glesner, "Adaptive Knowledge Bases in Self-Adaptive System Design," in *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*, aug 2015, pp. 472–478.

[8] G. Durelli, M. Coppola, K. Djafarian, G. Kornaros, A. Miele, M. Paolino, O. Pell, C. Plessl, M. D. Santambrogio, and C. Bolchini, "Save: Towards efficient resource management in heterogeneous system architectures," in *Reconfigurable Computing: Architectures, Tools, and Applications*, D. Goehringer, M. D. Santambrogio, J. M. P. Cardoso, and K. Bertels, Eds. Cham: Springer International Publishing, 2014, pp. 337–344.

[9] V. Petrucci, M. A. Laurenzano, J. Doherty, Y. Zhang, D. Moss, J. Mars, and L. Tang, "Octopus-man: Qos-driven task management for heterogeneous multicores in warehouse-scale computers," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2015, pp. 246–258.

[10] M. Meredith and B. Urgaonkar, "On Exploiting Resource Diversity in the Public Cloud for Modeling Application Performance," in *CLOUD COMPUTING 2017, The Eighth International Conference on Cloud Computing, GRIDs, and Virtualization*. Athens, Greece: IARIA, 2017, pp. 66–72.

[11] K. Djemame, R. Bosch, R. Kavanagh, P. Alvarez, J. Ejarque, J. Guitart, and L. Blasi, "PaaS-IaaS Inter-Layer Adaptation in an Energy-Aware Cloud Environment," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 2, pp. 127–139, apr 2017.

[12] P. Ruiu, A. Scionti, J. Nider, and M. Rapoport, "Workload management for power efficiency in heterogeneous data centers," in *2016 10th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)*, July 2016, pp. 23–30.

[13] Legato Project, "LEGaTO Homepage," 2018. [Online]. Available: https://legato-project.eu/about

[14] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 187–198, 2011.

[15] I. Mavroidis, et. al., "Ecoscale: Reconfigurable computing and runtime system for future exascale systems," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 696–701.

[16] P. Harvey, K. Bakanov, I. Spence, and D. S. Nikolopoulos, "A scalable runtime for the ecoscale heterogeneous exascale hardware platform," in *Proceedings of the 6th International Workshop on Runtime and Operating Systems for Supercomputers*, ser. ROSS '16. New York, NY, USA: ACM, 2016, pp. 7:1–7:8.

[17] C. Silvano, G. Agosta, A. Bartolini, A. R. Beccari, L. Benini, J. Bispo, R. Cmar, J. M. P. Cardoso, C. Cavazzoni, J. Martinovič, G. Palermo, M. Palkovič, P. Pinto, E. Rohou, N. Sanna, and K. Slaninová, "Autotuning and adaptivity approach for energy efficient Exascale HPC systems: The ANTAREX approach," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 708–713.

[18] C. Silvano, G. Agosta, J. Barbosa, A. Bartolini, A. R. Beccari, L. Benini, J. Bispo, J. M. P. Cardoso, C. Cavazzoni, S. Cherubin, R. Cmar, D. Gadioli, C. Manelfi, J. Martinovič, R. Nobre, G. Palermo, M. Palkovič, P. Pinto, E. Rohou, N. Sanna, and K. Slaninová, "The ANTAREX tool flow for monitoring and autotuning energy efficient HPC systems," in *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2017, pp. 308–316.

[19] J. Flich, G. Agosta, P. Ampletzer, D. A. Alonso, C. Brandolese, E. Cappe, A. Cilardo, L. Dragić, A. Dray, A. Duspara, W. Fornaciari, G. Guillaume, Y. Hoornenborg, A. Iranfar, M. Kovač, S. Libutti, B. Maitre, J. M. Martínez, G. Massari, H. Mlinarić, E. Papastefanakis, T. Picornell, I. Piljić, A. Pupykina, F. Reghenzani, I. Staub, R. Tornero, M. Zapater, and D. Zoni, "MANGO: Exploring Manycore Architectures for Next-GeneratiOn HPC Systems," in *2017 Euromicro Conference on Digital System Design (DSD)*, 2017, pp. 478–485.

[20] P. F. Dutot, Y. Georgiou, D. Glesser, L. Lefevre, M. Poquet, and I. Rais, "Towards Energy Budget Control in HPC," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017, pp. 381–390.

[21] J. Nunez-Yanez, "Energy efficient reconfigurable computing with adaptive voltage and logic scaling," *SIGARCH Comput. Archit. News*, vol. 42, no. 4, pp. 87–92, Dec. 2014.

[22] X. Mei, X. Chu, H. Liu, Y. W. Leung, and Z. Li, "Energy efficient real-time task scheduling on cpu-gpu hybrid clusters," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017, pp. 1–9.

[23] R. Kavanagh, D. Armstrong, and K. Djemame, "Accuracy of Energy Model Calibration with IPMI," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. San Francisco, California: IEEE, jun 2016, pp. 648–655.

[24] NVidia Corporation, "NVML API Pages - For GPU Utilization," 2017. [Online]. Available: http://docs.nvidia.com/deploy/nvml-api/structnvmlUtilization__t.html#structnvmlUtilization__t.