H2020 FETHPC-1-2014

An Exascale Programming, Multi-objective Optimisation and Resilience Management Environment Based on Nested Recursive Parallelism
Project Number 671603

D4.5 – Resource Management Support

WP4: Unified Runtime System for Extreme Scale Systems

Version:	1.0
Author(s):	Pierre Lemarinier (IBM), Thomas Heller (FAU)
Date:	27/03/2018

Due date:	PM30
Submission date:	30/04/2017
Project start date:	01/10/2015
Project duration:	36 months
Deliverable lead organization	IBM
Version:	1.0
Status	Final
Author(s):	Pierre Lemarinier (IBM) Thomas Heller (FAU)
Reviewer(s)	Stephane Monte (NUM), Peter Thoman (UIBK)

Dissemination level	
PU	*PU - Public*

Acknowledgements

The work presented in this document has been conducted in the context of the EU Horizon 2020. AllScale is a 36-month project that started on October 1st, 2015 and is funded by the European Commission.

The partners in the project are UNIVERSITÄT INNSBRUCK (UBIK), FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN NÜRNBERG (FAU), THE QUEEN'S UNIVERSITY OF BELFAST (QUB), KUNGLIGA TEKNISKA HÖGSKOLAN (KTH), NUMERICAL MECHANICS APPLICATIONS INTERNATIONAL SA (NUMEXA), IBM IRELAND LIMITED (IBM).

The content of this document is the result of extensive discussions within the AllScale Consortium as a whole.

More information

Public AllScale reports and other information pertaining to the project are available through the AllScale public Web site under http://www.allscale.eu.

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	05/02/18	Initial version	Pierre Lemarinier
0.2	12/03/18	Runtime components description	Thomas Heller
0.3	20/03/18	Finalization of the draft	Pierre Lemarinier
0.4	26/03/18	Internal review	Peter Thoman
0.5	26/03/18	Modification based on review	Pierre Lemarinier
0.6	27/03/18	Internal review	Stephane Monte
1.0	27/03/18	Final version based on reviews	Pierre Lemarinier

Table of Contents

1 Contents

Executive Summary

This document provides an overview of the Application Programming Interface for hardware monitoring and configuration. It also provides the description of a power saving scheduling policy targeting minimal energy usage.
The hardware and configuration API provides the capability for the runtime to probe information regarding the current frequency used by the different cores, the hardware topology, and to change the frequency of cores and switch them on and off.
The power saving policy is a new scheduling policy implemented within the Allscale runtime environment that makes use of this hardware monitoring and configuration API to minimize the power consumption of the system. This document provides a description of this policy and an evaluation of this policy using one of the pilot application on a power8 machine.

2 Introduction

One of the objectives of the Allscale project is to provide a finer grain approach and to give different options on how to execute an HPC application. Power consumption of large-scale system has become a major issue. Thus, allowing the development of applications that limits their ongoing power consumptions is paramount. Within Allscale, we envisioned the capabilities for the runtime to monitor and configure dynamically hardware parameters such as core frequency to reduce the ongoing power consumption. This would enable the scheduler to dynamically select the set of resources and hardware configuration that achieves the user's requested objectives.
The targeted deployment systems are distributed memory architecture. As accelerators are not included with the Allscale project at this stage, the two main proposed mechanisms for configuring the platform are changing the number of powered resources and frequency scaling. We thus proposed an API for monitoring and configuring core's frequency and activation, and implemented this API inside the Allscale runtime. To demonstrate the use of this API in reducing the ongoing power consumption, we implemented a new power saving scheduling policy which dynamically switches off unused resources and reduces the core frequency to reduce power usage, and we measure using both benchmarks and one of the pilot applications of the project the benefit of this policy.

We present in section 3 the hardware monitoring and configuration API, we then detail the power saving scheduling policy in section 4 and the experiment we conducted on a Power 8 system to exhibit the potential of this policy in section 5. Finally, in section 6 we present the Allscale components involved and discuss the accelerators integration into the Allscale project.

3 Hardware Monitoring and Configuration API

We present hereafter the resulting API for hardware monitoring and configuration. As accelerators are not included at this stage in the Allscale project, the API focuses on CPUs. Nonetheless, this API could easily be extended to accelerators in the future. The resources monitoring and configuration such as the number of cores and threads used on each node is integrated in the main Allscale runtime API, thus this hardware monitoring and configuration API focuses on CPU frequency scaling. We detail in this section the different functions that the API provides to further exhibit its capabilities.

3.1 Presentation of the proposed API

Monitoring

By default, CPUs frequency scaling is on-demand, which means that their frequency scales based on the ongoing load. In order to set the frequency manually, the CPU's governor needs to be set to user based instead of on-demand.
The following function returns the different possible governor policies for a targeted core.

```
std::vector<std::string> get_governors(unsigned int cpu);
```

In order to monitor what is the current frequency of a core, this API follows the semantic of cpufreq and provides two functions, one that returns the current frequency as seen by the kernel, and one that returns the actual hardware frequency. The later requires root access:

```
unsigned long get_kernel_freq(unsigned int cpu);
unsigned long get_hardware_freq(unsigned int cpu);
```

A user may need to obtain the different possible frequencies a CPU can run on to later choose which one she selects. The following function returns the vector of all possible frequencies in KHz for a specific CPU.

```
std::vector<unsigned long> get_frequencies(unsigned int cpu);
```

The following function returns the number of CPUs currently set on a specified frequency:

```
unsigned int num_cpus_with_frequency(unsigned long frequency);
```

The last function regarding frequency monitoring returns the latency in nanoseconds for changing the frequency of a targeted CPU:

```
unsigned long get_cpu_transition_latency(unsigned int cpu);
```

The last monitoring function provides the user the capabilities to get the current CPU resources online and available to Allscale. Its current implementation is based on HWLOC:

```
hw_topology read_hw_topology();
```

Configuration

The second section of the proposed API enables configuring the local hardware by setting the requested frequency and offlining CPUs. Here after we present the relevant different functions to reach this objective.

The main function for frequency scaling enables setting a target frequency to a sequence of cores:

```
int set_frequency(unsigned int min_cpu, unsigned int max_cpu, unsigned long
target_frequency);
```

The following function set the frequency of all cores to the next *freq_step* higher, if *dec* is false, or otherwise lower available frequency:

```
void set_next_frequency(unsigned int freq_step, bool dec = true);
```

In order for a user to amend the frequency scaling, she must first change accordingly the frequency scaling gorvernor for these CPUs:

```
int set_freq_policy(unsigned int cpu, cpufreq_policy policy);
```

The API finally provides two functions for respectively switching on and off selected cores:

```
void make_cpus_offline(unsigned int min_cpu_id, unsigned int max_cpu_id);
void make_cpus_online(unsigned int min_cpu_id, unsigned int max_cpu_id);
```

3.2 Integration in the Allscale Runtime

The proposed API has been integrated into the Allscale runtime. Its current implementation relies on HWLOC for the resource discovery, and on cpufreq for monitoring and configuring the frequency of CPUs. The power consumption monitoring is provided by the monitoring component of the Allscale runtime. The resulting development is used by the Allscale scheduler for its power saving policy presented hereafter, and for the current development of the multi-objective scheduling policy.

4 Power Saving Scheduler Policy

The proposed power saving scheduling policy makes use of the hardware monitoring and configuration API for setting the platform to a minimum power using system. As accelerators are not considered and the runtime is still progressing towards a distributed memory set-up, only investigation on a shared memory system is presented in this document. Nonetheless, this scheduling policy serves both as a demonstrator of the use of the hardware monitoring and configuration capabilities and as a base for the power saving goals of the future multi-objectives scheduling policy. As such, it reduces gradually and dynamically the power consumption of the machines, enabling the measurement of both the power reduction and the time degradation as the CPUs frequency scales down.

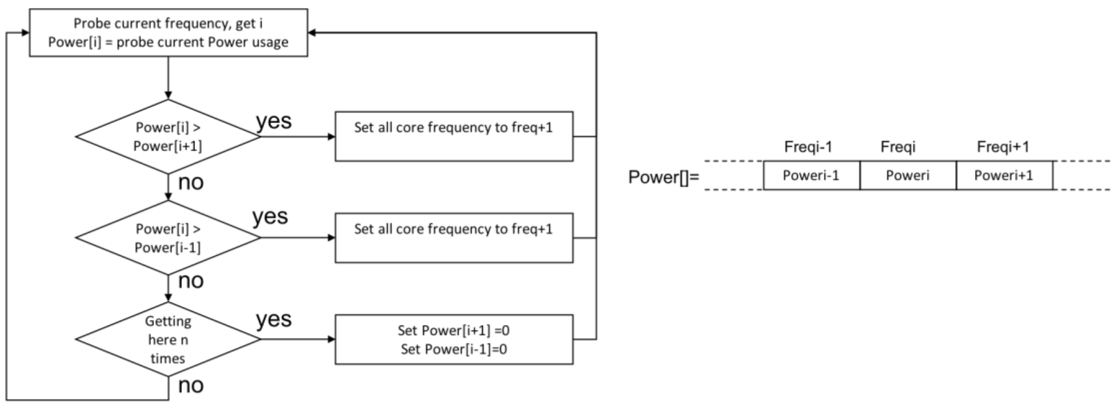

Figure 1- Overview of the Power Saving Scheduling Policy

Figure 1 presents an overview of the power saving scheduling policy. When the Allscale runtime is initializing, all unused computing resources are put offline. The frequency governor of used CPUs is set to *user* mode and these CPUs are in turn set to their maximum possible frequency before executing the application.
While the application is executing, the scheduler enquires regularly the current frequency the CPUs are currently running on, along with the current power usage of the board of the machine and compares to the power usage it would get by running at the next lower or higher frequency. Note that initially the scheduler does not possess the information on the power usage for the different possible frequency. This information is initialized to 0 power consumption for all frequencies. If to its knowledge the power consumption would diminish by running to the next lower or higher frequency, the scheduler requests the change of frequency using the relevant call from the hardware monitoring and configuration API. Finally, in order to adapt to a change in the workload, the information on the power consumption for the neighboring frequencies are regularly reset to 0, forcing the scheduler to change the current CPUs frequency and thus measure the resulting power consumption associated to them.

5 Preliminary Results

The presented scheduling policy has been implemented and integrated with the Allscale runtime. We present hereafter the obtained performance evaluation using this policy compared to using the standard performance-oriented scheduling policy of Allscale. This evaluation was conducted on a Firestone Power8 machine equipped with a 2x10 cores Power8 processors. The provided results were obtained by using the boost library 1.59 and gcc-7.2 for generating the Allscale runtime and applications.

5.1 Energy and Power Monitoring

The ongoing power consumption is a metric that will eventually be reported by the monitoring component of the Allscale runtime. At the time of this evaluation, this metric was not available from the monitoring component yet. A workaround implementation was thus included into the hardware monitoring API and its implementation to provides this metric to the scheduler. Its proposed implementation is Power8 architecture specific and relies on the capabilities from the Power8 On Chip Controller (OCC: https://www.ibm.com/developerworks/community/wikis/home#!/wiki/Power%20Systems/page/Overview%20of%20the%20OCC%20for%20POWER8) component to report different sensor metrics with a small time sampling and exposes them either out of band through BMC or in-band. The metric that the scheduler regularly requests corresponds to the board's current power usage.

5.2 Benchmark results

The first set of experiments consists in executing a benchmark implemented with Allscale called *fine_grained*. This benchmark computes a simple stencil operation using a *pfor* operation.

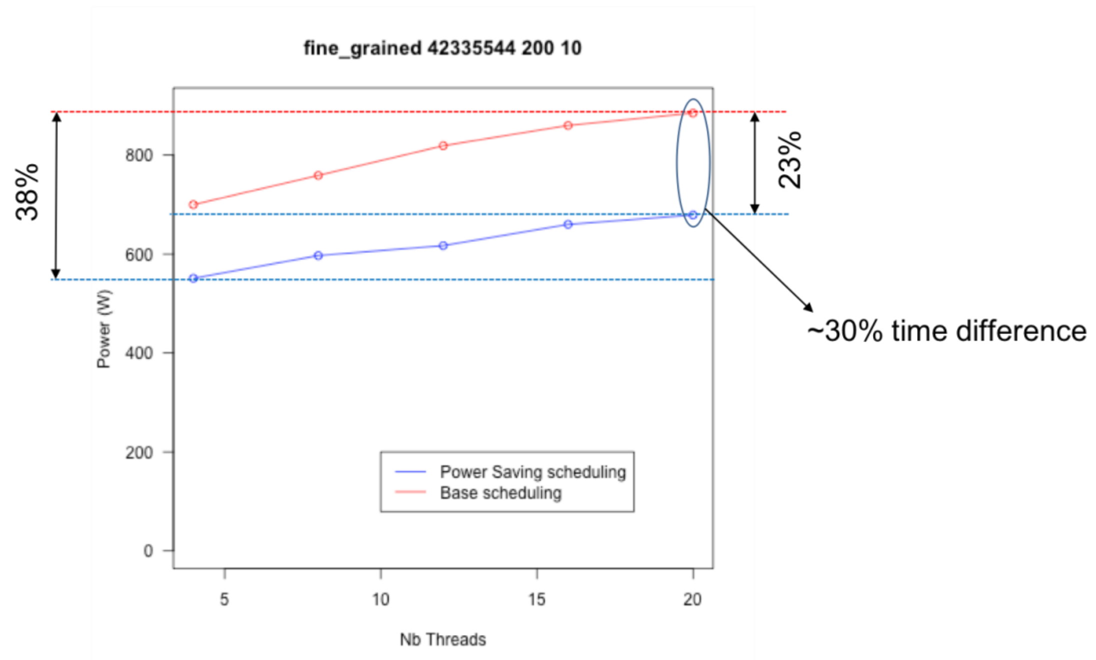

Figure 2 - Scaling of the Fine_grained benchmark using both power saving scheduling policy and the default allscale scheduling policy.

These experiments consist in executing fine_grained using the Allscale runtime with a predefined number of threads from ranging from 4 to 20. For each predefined number of threads, two different types of execution are made: one using the power saving scheduling and one using the default scheduling policy.
When running with the default policy, the Allscale runtime let all used resources in the default configuration they are in, which implies that all CPU frequency governors are let "on-demand". This implies that the OS governs the core frequency based on the workload and keeps all core online. The reported power measurement is the average power being consumed during the execution. Figure 2 presents the experiments results obtained on the Power8 machine. The average power consumption is reduced by more than 20% when using the power-saving scheduling policy. While this policy does not change the number of resources used by the application, close to 40% average power consumption reduction could be achieved by allowing the power saving scheduling policy to also reduce dynamically the number of resources. This will be a feature of the final multi-objectives scheduling policy for the extreme case of targeting minimal energy usage as the sole objective.

5.3 Pilot application IPIC3D

The second set of experiments was conducting using one of the 3 pilot applications of the Allscale project: IPIC3D. this pilot application simulates the interaction of solar wind with the earth's magnetosphere at the particle level.
Two different application configurations were studied, one called U which executes this simulation on a uniform distribution of particles, and B which simulates a beam of such particles. Both configurations assumed 100000 particles.

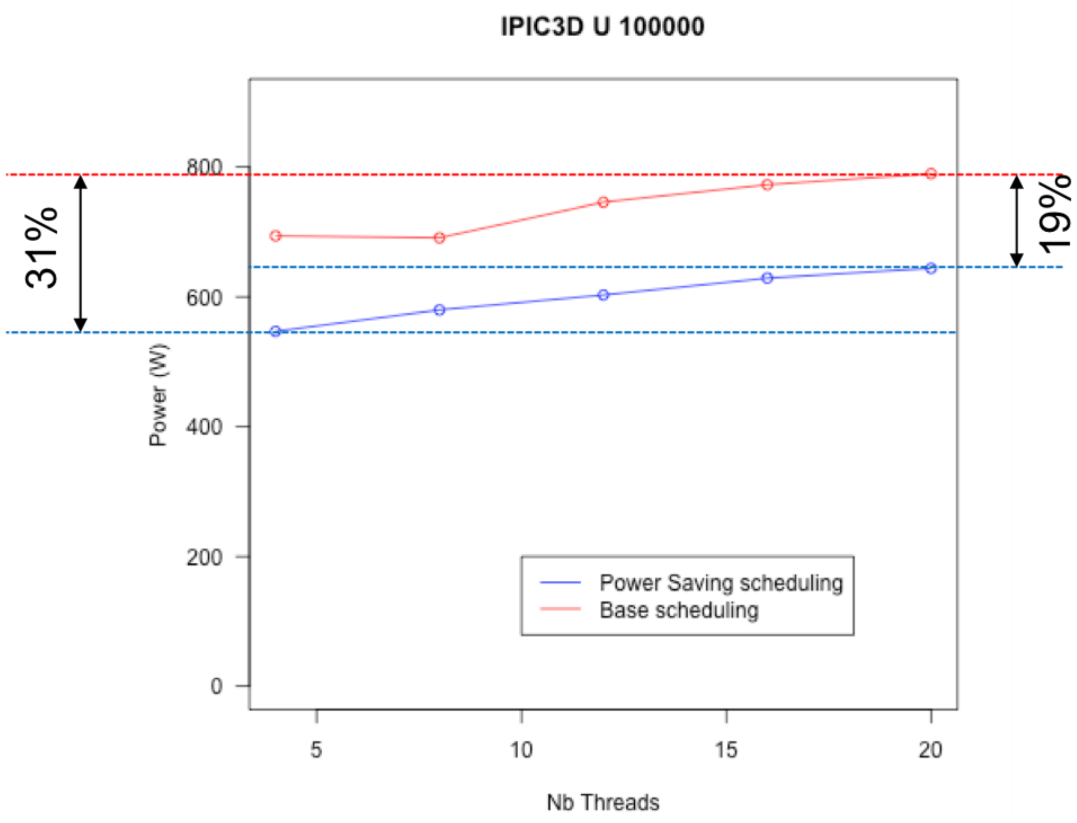

Figure 3 - IPIC3D on uniform particles distribution

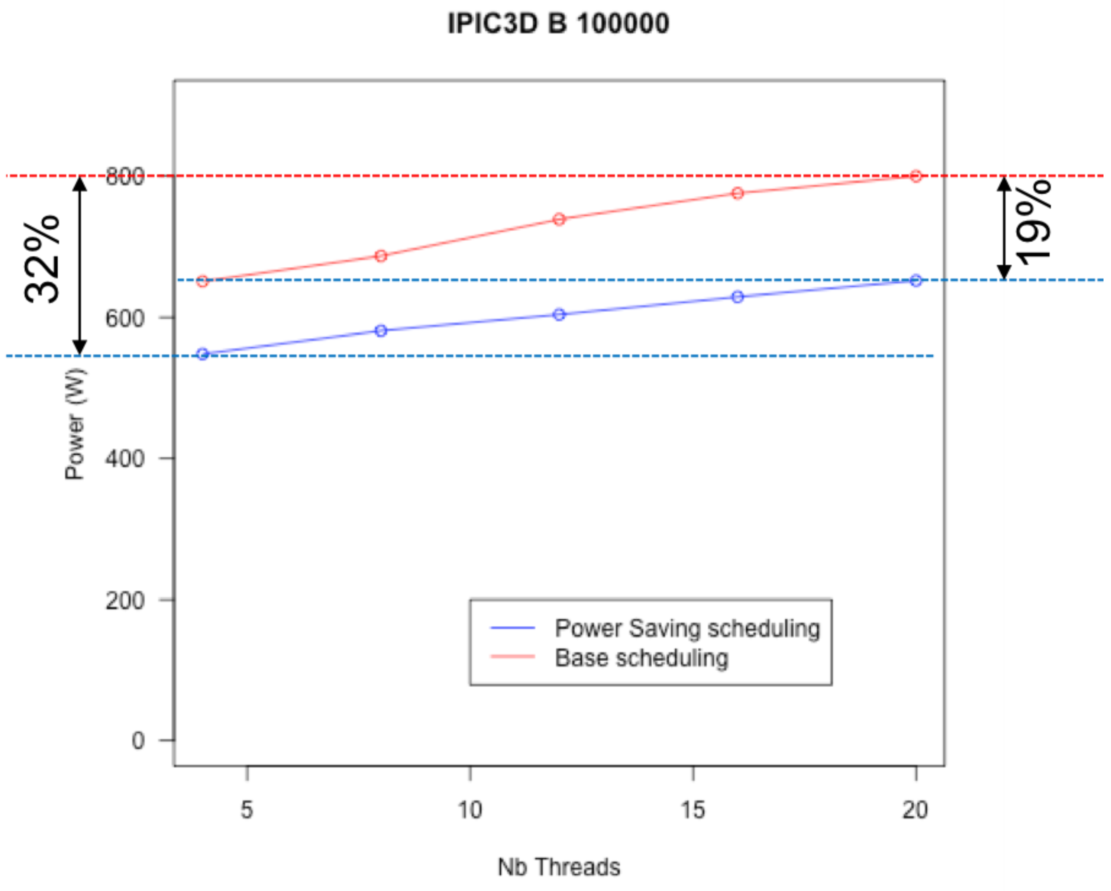

Figure 4 - IPIC3D on a beam shaped distribution of particles

Figure 3 and Figure 4 presents the obtained results when executing IPIC3D on Power8 machine with respectively a uniform and beam shaped distribution of particles. Compared to the *fine_grained* benchmark, the average power saved by the proposed scheduling policy is slightly lesser, reaching 19% compared to using the default scheduling policy. Moreover, the possible average power saving obtained by combining frequency scaling, setting unused CPUs offline and reducing the number of resources the application is using reaches 32%, which exceeds our objective of 25% difference in ongoing energy consumption between execution targeting maximizing power reduction execution targeting maximizing resource utilization.

6 Runtime Components and Accelerators

After a review of existing tools for accelerators in HPC, the AllScale consortium came to the conclusion that an inclusion of accelerators is not feasible at the moment. The prevalently used framework in HPC is CUDA. CUDA on one hand offers the promise to be able to seamlessly compile C++ code to GPU device code. Unfortunately, the compiler by NVIDIA is not mature enough to handle the sophisticated modern C++ techniques used by the AllScale environment. Furthermore, technological difficulties with memory handling is currently blocking further developments within the CUDA framework. Recent developments, both in compiler technology as well as unified memory handling, might lead to a successful GPU integration within the AllScale environment by the end of the project.

7 Conclusion

In this report we presented the hardware monitoring and configuration API that tackles frequency scaling, then ee presented a power saving scheduling policy that serves as a basis for the future multi-objectives scheduling policy. Finally, we presented the integration of both the API and the power saving scheduling policy into the Allscale project and reported experiment evaluations on a Power8 system of these newly added features using one of the pilot application.