H2020 FETHPC-1-2014

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

Project Number 671603

D4.6 – Multi-Objective Dynamic Optimizer (a)

WP4: Unified runtime system for extreme scales

Version: 0.5
Author(s): Khalid Hasanov (IBM)
Date: 20/01/17

Due date:	PM16
Submission date:	day/month/year
Project start date:	01/10/2015
Project duration:	36 months
Deliverable lead organization	IBM
Version:	0.5
Status	Final
Author(s):	Khalid Hasanov (IBM)
Reviewer(s)	QUB, KTH

Dissemination level	
PU	*Public*

Disclaimer

This deliverable has been prepared by the responsible Work Package of the Project in accordance with the Consortium Agreement and the Grant Agreement Nr 671603. It solely reflects the opinion of the parties to such agreements on a collective basis in the context of the Project and to the extent foreseen in such agreements.

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 (NUMECA), 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	05/01/17	Frist draft	Khalid Hasanov
0.2	10/01/17	First review	Pierre Lemarinier
0.3	11/01/17	Second draft, addressing Kostas's feedback	Khalid Hasanov, Kostas Katrinis
0.4	17/01/17	Third draft – adressing feedback form QUB and KTH	Khalid Hasanov, Kiril Dichev, Xavier Aguilar
0.5	20/01/17	Final version	Khalid Hasanov

Table of Contents

Executive Summary

This document describes the status and structure of the early AllScale multi-objective dynamic optimizer (a) deliverable (D4.6). Instructions on how to obtain, build and run the prototype will be provided.

1 Introduction

The deliverable D4.6 is part of task T4.6 within WP4. T4.6 is all about designing and implementing a multi-objective dynamic optimizer component within the AllScale architecture. The objective of T4.6 is to research efficient ways of utilizing the available capabilities for steering applications towards fulfilling flexibly customizable trade-offs among a variety of (conflicting) optimization objectives, such as, minimizing execution time, energy consumption or resource utilization (=computational cost). This report covers only D4.6, which provides a multi-objective dynamic optimizer that is capable of effectively utilizing the multi-versioning feature of the AllScale environment. Namely, the scheduler controls the task granularity either by splitting tasks into sub-tasks or processing them without splitting. From the runtime point of view splitting a task means selecting a split variant of the work item and processing a task means selecting a process variant of the work item. In addition, the scheduler is able to dynamically control the number of cores utilized by throttling operation system threads.

The design and implementation of the multi-objective dynamic optimizer is aligned with the API specification of the deliverable D4.1 and extends the simple prototype scheduler as defined in D4.2 with a resource- and time-aware multi-objective scheduling policy.

The remainder of this document will provide instructions on how to build the prototype as well as the design and implementation details.

2 Build instructions

Since the AllScale runtime system is based on HPX, we have an installation of HPX as a prerequisite. Information on how to install HPX can be found here: http://stellar-group.github.io/hpx/docs/html/hpx/manual/build_system.html
Once HPX is installed, the runtime system prototype is expected to have all needed dependencies.

The multi-objective dynamic optimizer is part of the AllScale runtime and extends its prototype scheduler (D4.2). Thus, the source code of the prototype can be found in the AllScale GitHub repository at the URL below:
https://github.com/allscale/allscale_runtime

Once the sources have been obtained either via the Git SCM or a downloadable snapshot, the build process is configured using CMake, which interfaces with the native build tools of the developer's platform (make, Visual Studio, Eclipse, etc.) and the runtime system can be built together with the unit tests as well as an accompanying example.

The code is organized as follows:

- <project root>
  - allscale – all headers needed to interface with the runtime system
  - src – The source files that contain the implementation of the runtime system

- example – Examples showing the use of the runtime system
- tests – Unit tests to ensure a functional runtime system
- CMakeLists.txt – cmake build script
- README.md – short summary and usage hints

3 Design of Multi-Objective Dynamic Optimizer for Shared Memory Systems

3.1 The Objective Function

The main responsibility of the dynamic optimizer is to minimize an objective function provided by the user. The user can provide his or her preference using the application command line options. These options can be self-descriptive for the end user and can be any from the predefined parameter list below: *time, resource, energy, time_resource, time_energy, resource_energy, time_resource_energy*. The default option is *time.* The user provided objectives are interpreted by the run-time as the following objective function:

$$t^n e^m r^k$$

Here, t is the total execution time, e is the total energy requirement, and r is the total resource usage. This function is general enough to cover either one or more of the given objectives by manipulating the weights within their domains: $0 \leq n, m, k < 2, n, m, k \in \mathbb{N}$. For example, a user provided *time_resource* objective is interpreted by the run-time with the weights of $n = 1, m = 0, r = 1$, that is, the dynamic optimizer should find the best trade-off between the total execution time and the total resource usage. Such an objective function can be provided as in the example below:

./fibonacci <args> --hpx:threads=<num_threads> --hpx:ini=allscale.objective!=time_resource

The fibonacci example can be found in the example folder within the git repo of the runtime.

3.2 The Scheduler Design

The scheduler is organized in two-level hierarchical design:

- A top layer, ***strategic scheduler*** – this layer interprets the user provided objective function and runs asynchronously independent of the application on each node; it evaluates the objective function periodically and re/configures the available resources accordingly. The configurations may include throttling the number of cores, disabling/enabling hardware threads, changing the frequency, and/or voltage of the compute units, etc.

- A bottom layer, ***tactical scheduler*** – this layer is responsible for effectively utilizing the resources provided by the top layer; the decision of whether to split the work items or process, the selection of the work item variants and assigning them to compute units, and moving data item fragments are all responsibility of this layer. Each node has their own tactical scheduler and they mostly work independently of each other, except when they need to move data across the nodes for load-balancing

purposes, or satisfy the requirements imposed by the resiliency manager. **Figure 1** represents the interaction of the two layers.

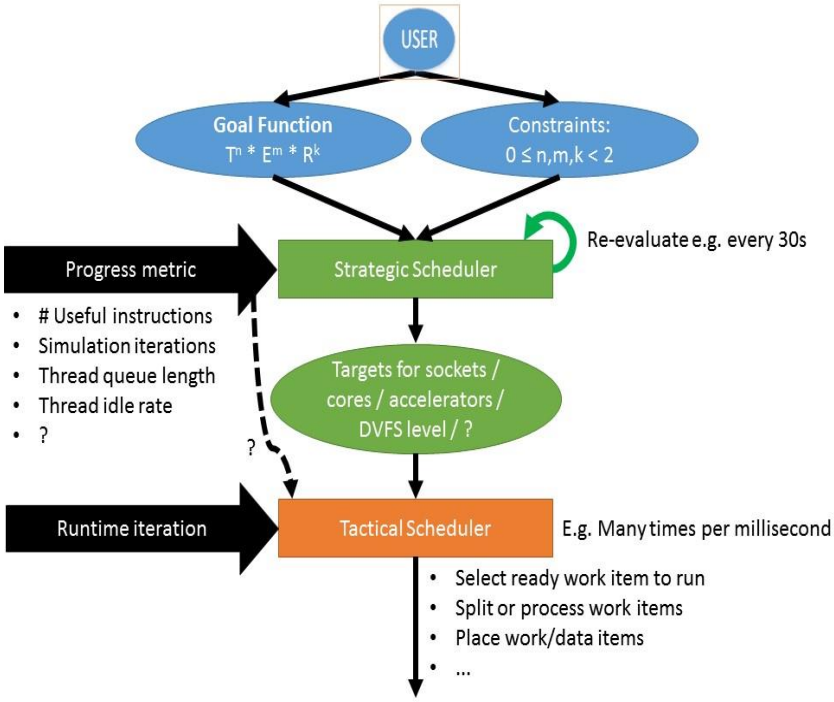

Figure 1. Multi-Objective Dynamic Optimizer

3.3 The Strategic Scheduler Implementation

As the AllScale project is based on the HPX runtime, the strategic scheduler utilizes the HPX APIs underneath to implement the desired functionalities. To implement a scheduling policy that can find an efficient trade-off between the total execution time and the total resource utilization, the run-time should provide a meaningful performance counter through its monitoring interface. Depending on the value of the utilized performance counter the number of cores either suspended or resumed while trying to find an optimal balance between the execution time and the number of cores utilized. This scheduling policy will also act as the foundation for implementing energy and power aware scheduling policies in later deliverables.

The performance counter that is being used currently is provided by the application. More precisely, after each iteration of the application its execution time is exported as an HPX performance counter. The scheduler reads such performance counter periodically and depending on its value the number of cores is either suspended or resumed. The number of threads to suspend or resume at each time is one. The thread to suspend is elected randomly and the thread to resume is always the first one in the suspended list of threads. It is worth mentioning that, the design is not limited to any specific performance counter and any meaningful performance counter can be used for this purpose. In the future, the performance counters provided by the monitoring interface (D5.2) will be incorporated into the scheduler. While we will do more research

on usability of different performance counters, it is enough at this stage to show the capability and the design of the multi-objective dynamic optimizer. The implementation can be found in the ***periodic_throttle*** method in the ***scheduler_component.cpp*** file:
https://github.com/allscale/allscale_runtime/blob/master/src/components/scheduler_component.cpp

3.4 The Tactical Scheduler Implementation

The implementation of the tactical scheduler depends on the HPX performance counters as well. In this case, two performance counters, namely, the length of the queue on each worker thread, and the idle rate of each worker thread are being used to decide when to split or process work items. If there are few tasks in the system but the idle rate is high, or if there are many tasks in the system but the idle rate is low, then the work item needs to be split, otherwise it is processed without being split. The implementation can be found in the ***do_split*** method in the ***scheduler_component.cpp*** file:
https://github.com/allscale/allscale_runtime/blob/master/src/components/scheduler_component.cpp

4 Future Work

After the first prototype of the shared memory version of the multi-objective dynamic optimizer has been completed, its capabilities will be extended to distributed memory systems (T4.6). It will incorporate additional objectives of energy usage, power dissipation, and objectives imposed by the resilience management component (T5.5). In addition, the dynamic optimizer will provide migration of data and tasks between hardware components for load balancing issues (T4.4), the management of hardware resources and their properties (T4.5). Furthermore it will utilize information collected and aggregated by the monitoring infrastructure (T4.3 and T5.2) to retrieve status information characterizing the managed environment.