H2020 FETHPC-1-2014

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

Project Number 671603

D5.2 – On-Demand, On-Line Monitoring Infrastructure (a)

WP5: Cross layer resilience and online analysis for non-functional parameters

Version: 1.0
Author(s): Xavier Aguilar (KTH)
Date: 13/01/17

Due date:	PM16
Submission date:	15/01/2017
Project start date:	01/10/2015
Project duration:	36 months
Deliverable lead organization	KTH
Version:	1.0
Status	Final Version
Author(s):	Xavier Aguilar (KTH)
Reviewer(s)	Thomas Fahringer (UIBK), Kiril Dichev (QUB)

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 (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	16/12/16	Frist draft	Xavier Aguilar
0.2	10/01/17	TF feedback	XA, TF
0.3	13/01/17	KD feedback	XA, TF, KD
1.0	13/01/17	Final version	XA, TF, KD

Table of Contents

Executive Summary 5
1 Introduction 6
2 Build Instructions 6
3 Component Description 7
3.1 HPX Hooks Used by the Component 7
3.2 Metrics Measured by the Component 7
3.3 Online Performance Introspection 7
3.4 Performance Reports at the End of a Run 8
4 Future Work 9
5 Bibliography 10

Index of Figures

Figure 1: Work item dependency graph. The graph is coloured using time from yellow (low value) to red (high value)..... 9

Index of Tables

Table 1: Work items and their execution times..... 8

Executive Summary

This document describes the status and structure of the first prototype of the AllScale performance monitoring infrastructure (D5.2). The document contains instructions on how to obtain and build the prototype as well as a description of its implementation.

1 Introduction

Deliverable D5.2 is a source code deliverable that provides the first prototype of the AllScale performance monitoring infrastructure. This prototype captures performance data for AllScale Work Items, which are defined in D4.1 and implemented in D4.2. The monitoring infrastructure also provides means for performance introspection, that is, capabilities to access the performance data online while the application runs. Thereby, it can help the AllScale scheduler in its decision making process. In addition, the monitoring infrastructure generates performance profiles that can be used by AllScale users to analyze their applications after program execution.

2 Build Instructions

The first prototype is the basis of development for the AllScale performance introspection tool, and provides a starting infrastructure that can be used by the other collaborators in the project. On one hand, it empowers the performance analysis of the code generated by the AllScale compiler. On the other hand, it provides insightful information on the performance achieved by the scheduler and the runtime.

The AllScale monitoring component is embedded within the AllScale runtime and on top of HPX, therefore we need to install HPX first. Information on how to install HPX can be found here:
http://stellar-group.github.io/hpx/docs/html/hpx/manual/build_system.html

In addition to the installing instructions that can be found in the previous link, HPX has to be built with the flag –DHPX_WITH_THREAD_IDLE_RATES=On.

If the user wants to collect PAPI counters (Mucci 1999), PAPI needs to be installed too:
http://icl.cs.utk.edu/papi/

The source code of the prototype can be currently found in our internal git repository at this URL: http://10.10.5.114/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 platform used. The monitoring component builds together with the AllScale runtime as well as its 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 monitoring infrastructure and 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 Component Description

3.1 HPX Hooks Used by the Component

The monitoring component relies on several hooks implemented by the AllScale runtime system to intercept work item execution. The prototype presented here catches the following actions:

- Work item execution start and end
- Work item enqueued and dequeued in scheduler queues
- Final result is propagated through each work item in the treeture (defined in D4.1).

When one of these actions is intercepted, the monitoring component takes control and performs the proper tasks needed to collect the performance data.

3.2 Metrics Measured by the Component

The metrics collected by the first prototype are execution time and PAPI hardware performance counters. For each work item, the monitoring component captures exclusive execution time and inclusive execution time. Exclusive time refers to the time spent within the work item itself, and inclusive time is the time spent by all the work items spawned from a work item. In other words, the inclusive time is the time spent since the work item starts until it has the final result ready.

For PAPI counters, the user defines the counters to be collected via the environment variable MONITOR_PAPI, e.g., *MONITOR_PAPI="PAPI_TOT_INS, PAPI_L2_TCM"* would be used to collect total instructions and total L2 cache misses. Once the variable is defined, the monitoring component measures such counters per work item.

3.3 Online Performance Introspection

This prototype defines and implements an API that can be used to obtain performance data on-demand while the application still runs. This data is extremely useful, for instance, to help the scheduler in its decision making process. The current prototype provides the introspection within a single shared memory node. The second prototype will implement introspection capabilities at a global level across nodes.

The Introspection API contains the following calls:

```
// Returns the exclusive time for a work item with ID w_id
double get_exclusive_time(std::string w_id);

// Returns the inclusive time for a work item with ID w_id
double get_inclusive_time(std::string w_id);

// Returns the average exclusive time for a work item with name w_name
double get_average_exclusive_time(std::string w_name);

// Returns the minimum exclusive time for a work item with name w_name
double get_minimum_exclusive_time(std::string w_name);

// Returns the maximum exclusive time for a work item with name w_name
double get_maximum_exclusive_time(std::string w_name);

// Returns PAPI counters for a work item with ID w_id
double *get_papi_counters(std::string w_id);

// Returns the PAPI counter with name c_name for the work item
// with ID w_id
double get_papi_counter(std::string w_id, std::string c_name);
```

These Introspection calls return 0 in case the work item does not exist or it has not finished its execution yet.

3.4 Performance Reports at the End of a Run

Once the application ends, the performance monitoring component writes two reports. First, it outputs a table such as Table 1 with the metrics collected per work item. The table in this case contains exclusive and inclusive times. It also contains the % of such times regarding to total application time (application wall-clock time).

Work Item	Exclusive Time	% Total	Inclusive time	% Total
0.0 fib	3.00065	42.14	7.0042	98.35
0.0.0 fib	3.00.243	42.16	3.0054	42.20
0.0.0.0 fib	2.44×10^{-3}	0.03	2.74×10^{-3}	0.03
0.0.0.1 fib	1.002	14.07	1.004	14.10

Table 2: Work items and their execution times.

After writing the table, the monitoring infrastructure creates a file called "treeture.dot" that contains a graph in DOT format (Koutsofios 1991) with all the

work items and their dependencies. Each node of the graph is a work item and includes work item name, work item ID, exclusive time (time spent in such node), and inclusive time (total time spent in all children spawned from the node). The edges are the dependencies between work items. The graph is coloured by exclusive time with a gradient from yellow to red in order to highlight work items with long execution times. Figure 1 shows an example of the graph created after running the Fibonacci example included in the sources.

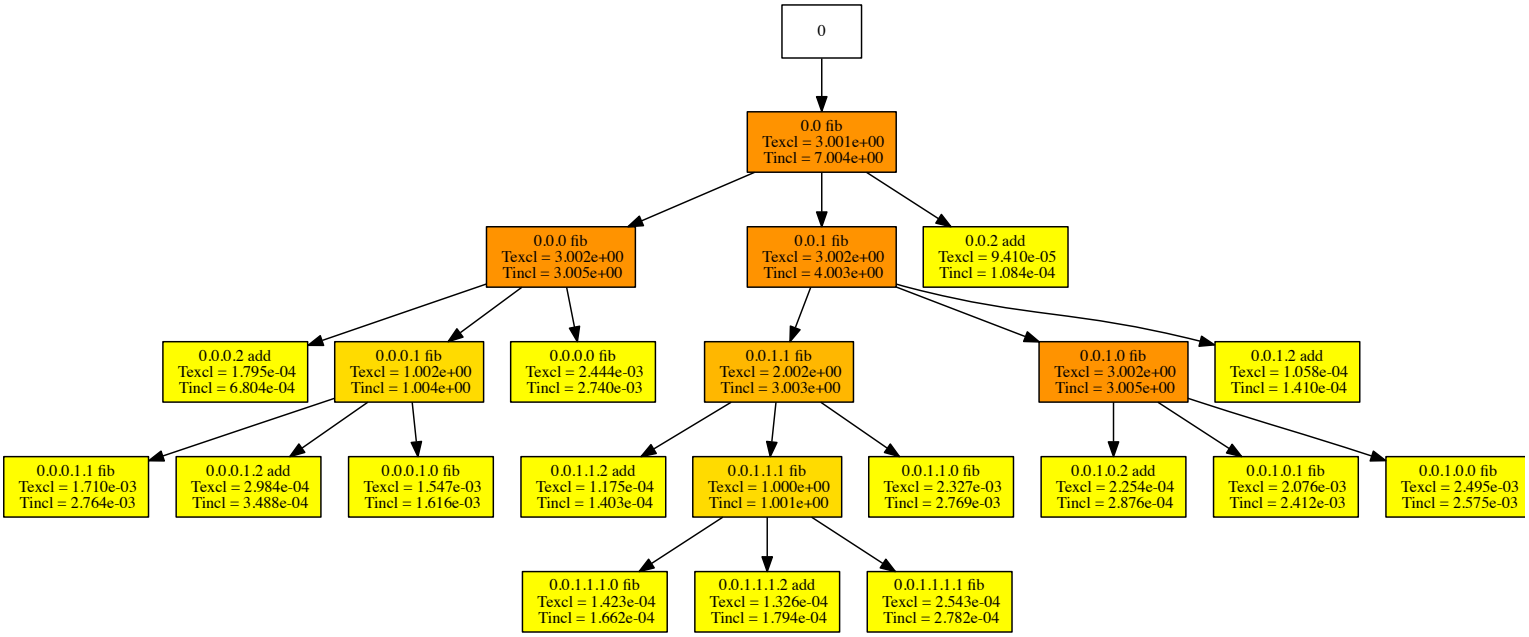

Figure 1: Work item dependency graph. The graph is coloured using time from yellow (low value) to red (high value).

4 Future Work

The first prototype presented in this document provides introspection capabilities within a shared memory node. Thus, the next step is to extend the prototype in order to provide a holistic view of the whole application across nodes in a distributed memory system. In addition, we will refine and add new metrics to the monitoring infrastructure, for instance, power and energy metrics.

The current prototype captures data for all work items. Future versions of the monitoring infrastructure will provide filtering mechanisms to select the work items monitored.

5 Bibliography

Koutsofios, Eleftherios, and Stephen North. "Drawing graphs with dot. Technical Report 910904-59113-08TM." AT&T Bell Laboratories, Murray Hill, NJ, 1991.

Mucci, Philip J., Shirley Browne, Christine Deane, and George Ho. "PAPI: A portable interface to hardware performance counters." *In Proceedings of the department of defense HPCMP users group conference.* 1999. 7-10.