[1]H2020 FETHPC-1-2014

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

Project Number 671603

D4.4 – Data Management Support

WP4: Unified Runtime System for Extreme Scale Systems

Version: 1.0
Author(s): Arne Hendricks (FAU), Thomas Heller (FAU)
Date: 09/04/18

[1]

Due date:	30
Submission date:	09/04/18
Project start date:	01/10/2015
Project duration:	36 months
Deliverable lead organization	FAU
Version:	1.0
Status	Draft
Author(s):	Arne Hendricks (FAU) Thomas Heller (FAU)
Reviewer(s)	Herbert Jordan (UIBK), Fearghal O Donncha (IBM)

Dissemination level	
PU	*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
1.0	09/04/18	Draft	
1.0	25/04/18	Final version	

Table of Contents

1 Introduction

One of the key aspects of the AllScale Runtime System is the efficient support of data management, i.e. the handling of distributed data items.
Efficient data management is also needed to accomplish goals like dynamic load balancing and resiliency as these features also depend on the efficient handling of data dependencies of work items.
The AllScale Runtime System is built on top of the HPX runtime system and makes use of the AGAS concept used by HPX.
AGAS, short for Active Global Address Space, extends the PGAS, short for Partitioned Global Address Space, model used by frameworks such as X10, Chapel, UPC and Co-Array Fortran.
But in contrast to PGAS, which partitions one global address space statically into separate logical blocks, AGAS also supports dynamic addition or subtraction of hardware resources as well as migration of globally named objects, making it a perfect suit for a dynamic and resilient foundation for our runtime system.

2 Implementation

The support for distributed computing comes in two forms. First of all, the general support of HPX for distributed computing that is implemented as the Active Global Address Space (AGAS) and then the specific implementation of data item lookup to support the specific use case of the AllScale Environment.

2.1 HPX AGAS

HPX's AGAS translates a global identifier (GID) to global addresses.
This global address can then be used to access an object remotely. Every address with a global address is called a global object. Localities in the AllScale Runtime System are modelled as global objects, and thus possess unique identifiers referencing them. The GIDs managed by AGAS are stored in address translation tables. The address translation table is implemented as a fully distributed lookup table supporting transparent migration.
Several components of the AllScale Runtime System are modeled using HPX components with GIDs, including the locality identifier for localities storing fragments of data items.

2.2 A Strategy for Efficient Data Item Management

AllScale data items are closely modeled with respect to the AllScale work items.
The region associated with a specific data item fragment is determined by the accompanying split operation of a work item. This builds up an hierarchical logical

structure of parent and child relationships which can be exploited for efficient lookup.
During the recursive splitting of regions and the following distribution of child work items to compute nodes by the scheduler, the data dependencies are also split, and as such, every data item has child regions, which are required for child tasks. This can be modeled with a binary tree, where each node spawns a left and right child node, consisting of the split regions.

The DataItemManager exposes this information in a direct action **locate**, which uses a layered approach to location.

1. It first checks if a requested region is present in its locally owned regions region, **owned_regions**, this can be done very efficiently with region operators **difference** and **intersect** (see D2.6(b))**.**

2. If the requested region is not present in the **owned_regions** region, a lookup cache is used. The cache enables fast lookup of a region to the locality where the fragment resides. This cache is being maintained and updated once the information is acquired in the next step.

3. If the required region is not found in the cache the requests needs to be branched out to the related nodes, i.e. left and right children, using HPX dataflow mechanisms. The order is determined the reverse order of the split information which is used to spawn and distribute work items. This allows for efficient retrieval of required information in logarithmic complexity. Since during that step, the associated caches are updated, remote lookup is only rarely required, an observation which was visible during evaluating the benchmarks and example applications.

In the case of load balancing, the DataItemManager is able to migrate fragments to other nodes updating the caches along the way. The effect of this migration is that work items requesting specific fragments need to re-locate which then might change to a worst-case complexity of O(Nodes) lookups.

In general, especially for long running simulations, it is to be expected that case 1 and 2 are predominant and the extensive lookup only needs to be performed during the start of an application or in the presence load balancing events.

3 Evaluation

The performance of the implementation described in this document is evaluated by a two-dimensional stencil benchmark as well as the IPIC3D prototype. It was evaluated on the Haswell compute nodes of the Cori Supercomputer which consists of two 16-core Intel CPUs interconnected with a Cray Aries Dragonfly network, 128 GB of memory per node, dedicated L1 and L2 caches, with 64 KB (32 KB instruction cache, 32 KB data) and 256 KB, respectively, as well as 40-MB shared L3 cache per socket.

The stencil benchmark was ported from the MPI versions of the Intel Parallel Research Kernels[i] using weak scaling. **Error! Reference source not found.** shows the performance of this benchmark. For regular accesses as performed by a stencil computation, the AllScale Runtime System is able to outperform the reference implementation due to the ability to perform asynchronous communication operations allowing to hide the needed communication completely behind computational phases.

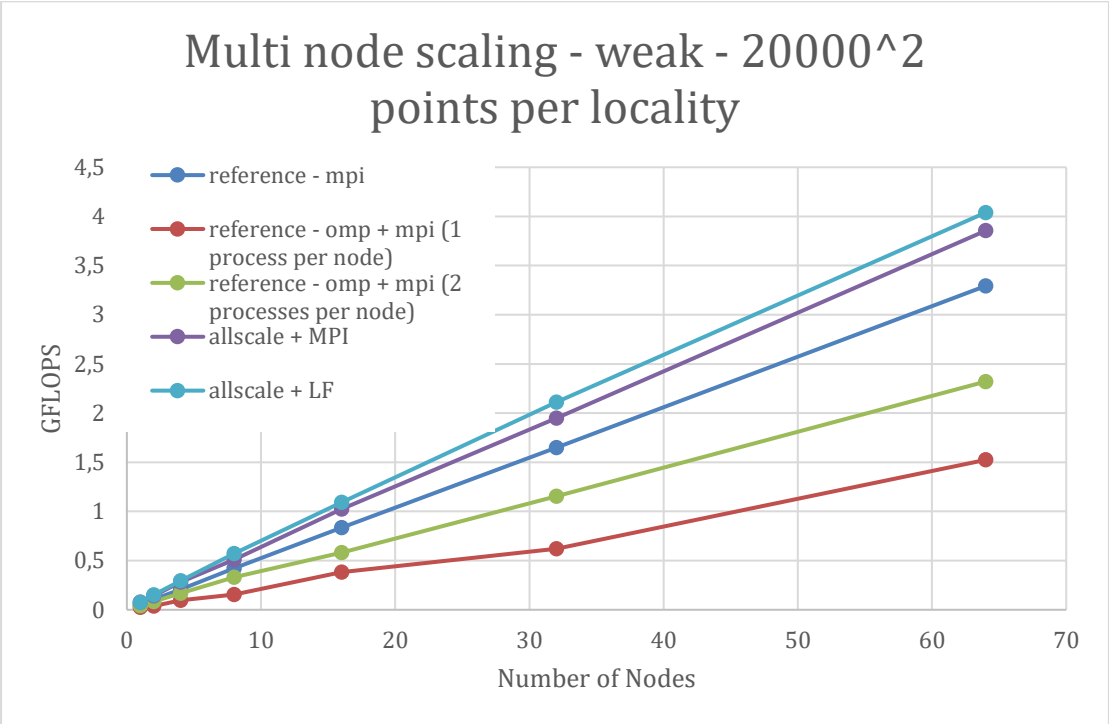

Figure 1 shows the weak scaling behavior of the AllScale Runtime System based Stencil implementation in comparison to the MPI+OpenMP reference implementation

The second evaluation of the implementation was performed with the IPIC3D prototype on the same Hardware (see Figure 2). The weak scaling results show a parallel efficiency of 95% using 16 nodes while the strong scaling capabilities drop to 20% for 16 nodes. The poor strong scaling is mainly a result of too little work available with more than 4 nodes.

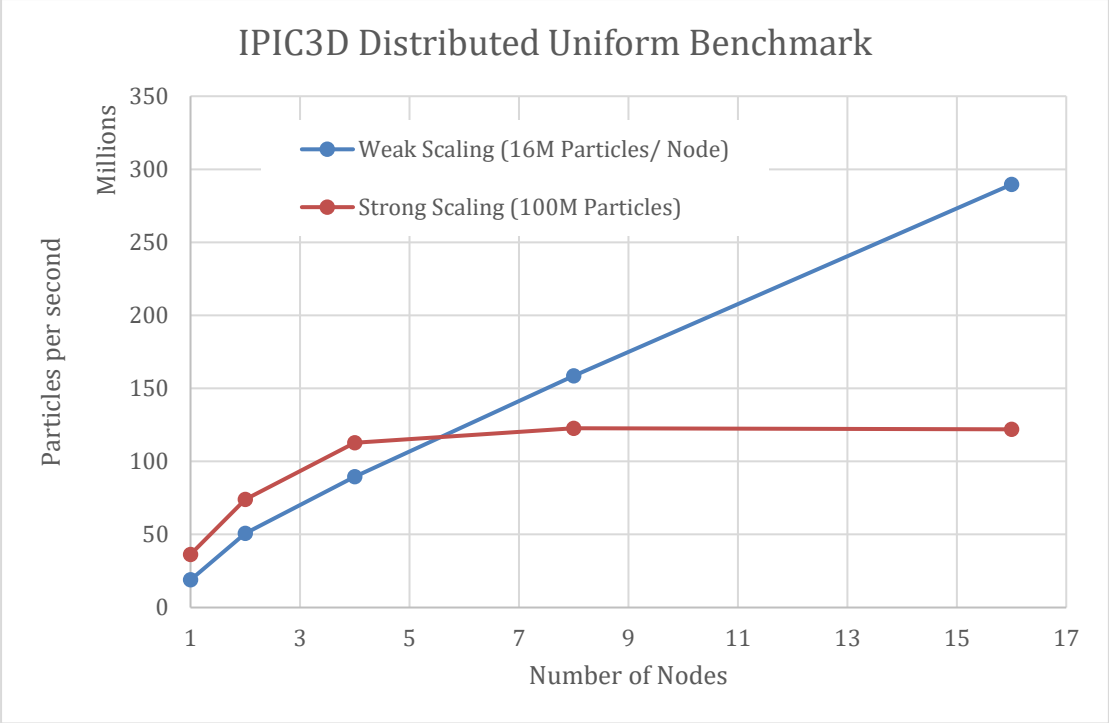

Figure 2 shows the performance of the IPIC3D prototype using a uniform distribution of particles over a different number of compute nodes.

The evaluation shows that the implementation of the DataItemManager provides a solid basis in terms of performance for further work based on our implementation.

4 Conclusions and Future Work

The infrastructure and implementation of data management support is in place and already used extensively by tests, examples and benchmarks. The performance evaluation is showing good results.
The implementation is continuously improved and adapted to arising use cases, and will be further evaluated regarding initialization overheads and chances for additional optimization in the next 3 months.

[i] https://github.com/ParRes/Kernels