D4.5.2 –Microbenchmark Suite (Software)

WP4: Algorithms and libraries

Project Acronym	CRESTA
Project Title	Collaborative Research Into Exascale Systemware, Tools and Applications
Project Number	287703
Instrument	Collaborative project
Thematic Priority	ICT-2011.9.13 Exascale computing, software and simulation

Due date:	M15
Submission date:	31/12/2012
Project start date:	01/10/2011
Project duration:	36 months
Deliverable lead organization	USTUTT
Version:	1.0
Status	Final
Author(s):	José Gracia, Christoph Niethammer & Wahaj Sethi (USTUTT)
Reviewer(s)	Erik Lindahl (KTH), Mats Hamrud (ECMWF)

Dissemination level	
PU	*PU – Public*

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	22/11/2012	First version of the deliverable	José Gracia (USTUTT)
0.2	30/11/2012	Final draft, submitted for internal review	José Gracia, Christoph Niethammer & Wahaj Sethi (USTUTT)
0.3	14/12/2012	Document slightly revised	José Gracia (USTUTT)
1.0	15/12/2012	Final version of the deliverable	José Gracia (USTUTT)

Table of Contents

1 Executive Summary

Task 4.5 is concerned, amongst others, with the optimisation of collective communication operations. Collective operations involve multiple participants rather than only two as is found in point-to-point communication operations. Examples of collective operations are synchronisation barriers, or reductions over the full computational domain in order to find the sum/min/max of a particular quantity. Such operations are very common in most distributed applications.

This document briefly describes the software deliverable *Collectives Microbenchmark Suite* which is used within the CRESTA project firstly to assess progress of the optimisation work on collective operations, but also secondly as a tool to analyse the characteristics of the implementation of collectives. For the users or developers of parallel applications, the benchmark suite may help in assessing which implementation of collective should be chosen in a specific use-case.

After a short introduction and details on the availability of the source code, Section 4 describes the basic concepts and design of the benchmark suite. The document concludes with short usage instructions.

2 Introduction

The objective of work package WP4 is to address the limitations of existing algorithms and libraries, including communication libraries, for exascale computing systems on various levels. Task 4.5 *Realising collectives at extreme scale,* is particularly concerned with collective communication operations, i.e. those communication operations which involve multiple participants. Examples of collective operations are synchronisation barriers, or reductions over the full computational domain in order find the sum/min/max of a particular quantity. Such operations are very common in most distributed applications.

The approach is to initially investigate the limitations of existing collective communication libraries in order to identify the key areas where completely new approaches are necessary and where optimisation can be realised focused on the implementation level.

Based on this analysis, research on new approaches will be undertaken with the goal to support the work packages on application and User Tools. Our focus is on developing new implementations of collective operations, which will yield step-wise improvements in the application codes as we go along.

Part of this task is to provide a *Collectives Microbenchmark Suite*. This benchmark suite serves three purposes:

> Firstly, it is meant to track the progress over time of the work done in task 4.5 by deriving metrics that characterise the efficiency of an implementation of a given collective.
>
> Secondly, it allows the user to assess the respective efficiency of a set of implementations of a collective for his particular use-case (e.g. number of communicating processes, size of payload, rank layout and communication pattern) in order to choose the most suitable implementation.
>
> Lastly, it also allows the developer of a collective operation to analyse the characteristics and the efficiency of a particular under very controlled and reproducible test conditions.

While this *Collectives Microbenchmark Suite* is initially used to assess MPI collectives in the three ways mentioned above, it is not restricted to MPI, but rather designed in a way that should allow it to be used with other programming paradigm supporting collective communication operations.

Several other MPI benchmarks suites exist, as, for instance, the *Intel MPI Benchmarks*,[1] or the *Ohio State University Micro-Benchmarks.*[2] These benchmarks, however, measure the timing of communication operations (including collectives) in isolation only. In contrast, the Collectives Microbenchmark Suite described here explicitly embeds the collective operations into a given context, something we refer to as scenario. This allows us to mimic, to a certain degree, the context in which an application might use collective operations to derive metrics that model the performance of collective in a (application) context.

[1] http://software.intel.com/en-us/articles/intel-mpi-benchmarks

[2] http://mvapich.cse.ohio-state.edu/benchmarks/

3 Availability of the benchmark suite

At the time of preparing this document the source code of the benchmark suite is available through the CRESTA SVN code repository at

https://svn.ecdf.ed.ac.uk/repo/ph/cresta/wp4/microbenchmark_suite

Access to the CRESTA SVN is subject to the policies of the project. Instructions on obtaining credentials and access to the SVN are available on the project BSCW.

After a testing and evaluation period, the benchmark suite will be made available through the CRESTA project website or a public code repository and will be distributed under an open source licence.

4 Design of the benchmark suite

The benchmark suite is a *framework* to measure and assess *arbitrary* implementations of collective operations under specific, reproducible scenarios. At this stage in the project it is being used to test and analyse implementations of MPI collectives. However, collectives in other programming paradigms, as for instance PGAS-like models, should be supported as well.

In order to be as general as possible, we treat the collective operation as a black-box and restrict ourselves at measuring only time, specifically the global completion time of a scenario (see below). Further, in order to minimise effects that might influence the completion time, the benchmark should be as unbiased towards, for example, cache effects or memory latency.

The benchmark suite defines:

- Which time difference to measure, i.e. location of start and completion timestamps within the code.
- Standard scenarios (think communication pattern) to be timed.
- A set of parameters to be varied. We distinguish parameters of the collective operation and parameters of the scenario.

4.1 Set of parameters for the collective operation

The following parameters for collective operations have been chosen. Some of them might take a range of values and the suite will take care to sweep through them. Others take a single specific value for simplicity without loss of generality. The parameters are:

- ***Number of participants (P)*** in the collective operation. A single run of the benchmark will sweep the number of participants from a user specified minimum to the maximum specified by the user. In MPI runs, the maximum is taken to be size of the MPI communicator at program start.
- ***Number of repetitions (nrep)*** of the timing loop. This is specified by the user.
- ***Message size (m)*** refers to the number of elements in the outbound message buffer (if applicable). It is swept from user-specified minimum to a maximum value in powers of two.
- The ***data type*** of the message is fixed to the single value of byte. This makes the total size of the payload unambiguous. Message buffers are contiguous.

4.2 Scenarios

In a nutshell, a scenario just specifies a sequence of actions taken by each participant; note that sequence of actions may not be the same across participants.

The following actions are possible: start a timer (*start*), stop the timer (*stop*), issue a collective operation (*issue-op*), wait on the completion of a collective operation (*sync*), do useful calculations (*comp*) between issuing a collective and waiting for its completion, or be delayed (*delay*) before issuing the collective operation. Note, that scenarios may use any collective operation. Also note that for blocking collective operations the *sync-op* is a null action.

We define three basic scenarios:

- Scenario 1: collective only
- Scenario 2: collective with delay
- Scenario 3: collective overlapping with computation

More advanced scenarios will be defined later as required. Possible examples are, for instance, a combination (sequence) of collectives, different (non-blocking) collectives issued concurrently, etc.

Scenario 1: all participants execute the sequence actions

start – issue-coll – sync – stop.

No further scenario parameters are specified.

Scenario 2: a specific participant D executes the sequence of actions

start – delay(t_{delay}) – issue-coll – sync – stop.

All other participant execute the same sequence as in scenario 1, i.e.

start – issue-coll – sync – stop.

The scenario parameters D and t_{delay} are user-specified. D can be either D=1 or D=P, i.e the first or last participant; t_{delay} sweeps from a user-specified minimum to a user-specified maximum value in powers of two; values are usually in the range of microseconds.

Scenario 3: all participants execute the sequence of actions

start – issue-coll – calc(t_{calc}) – sync – stop.

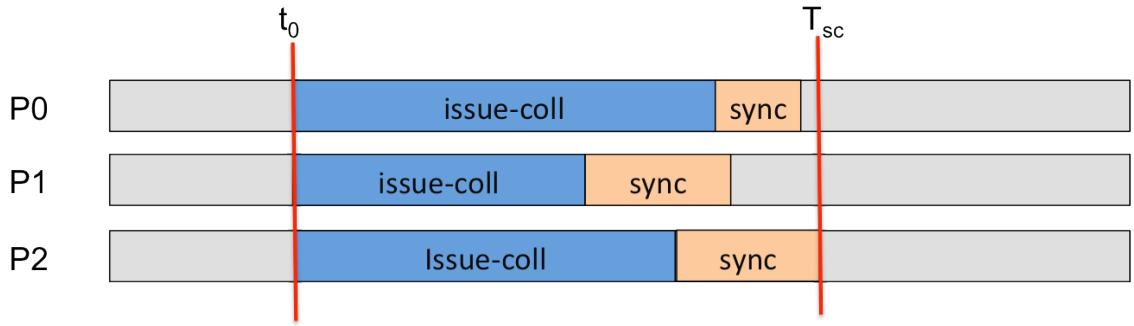
The scenario parameters t_{calc} sweeps from a user-specified minimum to a user-specified maximum value in powers of two; values are usually in the range of microseconds.

4.3 Time to completion

Each participant in the collective records timestamps at the start (t_{start}), and completion (t_{end}) of a scenario respectively. These timestamps are collected by the master and used to calculate the global time to completion of the scenario (T_{sc}) as:

$$T_{sc} = \max(<t_{end}>) - \min(<t_{start}>),$$

where the brackets <> denote the set of values collected from all ranks. The time to completion is then the timespan between the first participant entering the scenario up to the last participant completing the scenario. This is illustrated in the figure below for scenario 2 with three participants P0, P1, and P2. Note that the start time will not the same exactly for all participants.



5 Usage of the benchmark suite

This section very briefly addresses the usage of the benchmark suite. More specific instructions can be found in the README file that comes with the source files.

The benchmark suite is distributed with a *makefile* for easy building. The build process will generate an executable specific for a given collective that is specified by the argument *coll* as

```
make coll=barrier
```

This will generate the executable *mbs_barrier.out.* The argument *coll* currently can take any of the values *barrier, broadcast, gather, allGather, allReduce*.

The executable is started with *mpirun* or *aprun* respectively as

```
aprun –n P_max mbs_${coll}.out \
        P_min \
        nreps \
        m_min m_max \
        delay delay_P > out.log
```

where P_max and P_min are the maximum and minimum number of participants, nreps the number of repetitions, m_max and m_min the largest and smallest message size respectively, delay the duration delay for the late-arriving participant in scenario 2 in microseconds, and delay_P the participant number which is being delayed.

The output of the benchmark is quite lengthy and should be piped to a file as in the example above. The output is suitable for plotting with gnuplot (see output file for details).

A typical output, here for the broadcast collective with up to 256 participants and payload of 512 bytes, might look like:

Broadcast:

Data	ranks	average (us)	min (us)	max (us)	stdDev (us)
512	2	85.365891	37.193298	224.351883	4.814990
512	4	286.627841	234.127045	431.299210	7.668952
512	8	412.119293	355.243683	833.511353	13.162572
512	16	456.163192	403.642654	8327.484131	112.357419
512	32	474.721408	415.563583	601.530075	112.839889
512	64	496.664786	438.451767	609.636307	113.483498
512	128	523.060489	444.412231	640.392303	114.418232
512	256	577.194500	481.367111	781.536102	115.163906

The columns represent payload size, number of participants, average time to completion, minimum and maximum timing, and standard deviation of the measurements respectively. These measurements were made on cluster Laki consisting of Intel Nehalem nodes interconnected by an Infiniband network.

This textual output can be used to produce plots as the one below.

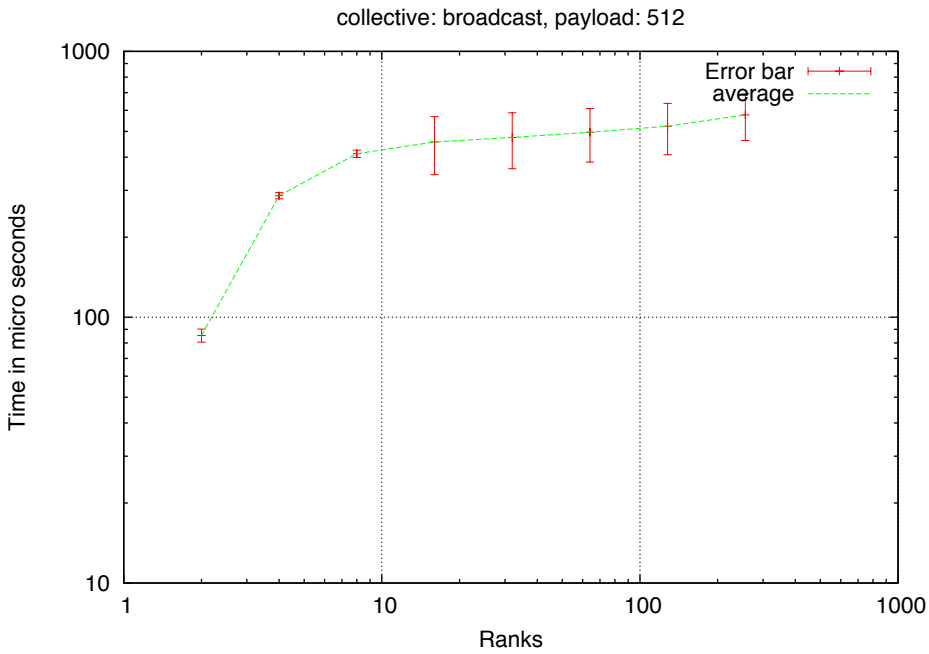


Figure 1 Typical plot generated from the textual output of the Collective Microbenchmark suite.

6 Conclusion

This document described the Collectives Microbenchmark Suite software deliverable. This is used within the CRESTA project firstly to assess progress of the optimisation work on collective operations, but also secondly as a tool to analyse the characteristics of the implementation of collectives. For the users or developers of parallel applications, the benchmark suite may help in assessing which implementation of collective should be chosen in a specific use-case.