HiPerDNO

High Performance Computing Technologies for Smart Distribution Network Operation

FP7 - 248135

Project coordinator: Dr Gareth Taylor, BU

Consortium Members: BU, EF, IBM ISRAEL, University of Oxford, EENL, UNION FENOSA, INDRA, GTD, KORONA, EG, Fraunhofer IWES

Document Title	Report on the initial testing, benchmarking and comparison of novel DMS functionality as supported by HPC
Document Identifier	HiPerDNO/2011/D3.1.3
Version	1.0
Work package number	WP3
Sub-Work package Number	WP3.1
Distribution	Public
Reporting consortium member	Oxford, Brunel, Indra
Internal reviewer & review date	Colin Axon (30/01/2012)

Report on the initial testing, benchmarking and comparison of novel DMS functionality as supported by HPC

Executive Summary

WP3 focuses on integration of HPC architecture to enable novel DMS applications which have been developed in WP1 & 2. Benchmarking is an important step towards the full integration in the final year of the project. The goal of benchmark tests is to evaluate the performance of the HPC platform developed in WP1.2 to meet the novel DMS applications' computational needs. Benchmarking is a challenging subject in its own right, which requires careful planning to be an effective tool to decide the best system for specific applications. Close collaboration between partners in WP1, 2 & 3 is required to design an effective mechanism to perform quantitative evaluation. In general, there are two ways to perform benchmarking, e.g., adopting standard benchmark suits and tailored benchmark tests. While there are various benchmark suites [1-5] adopted by the HPC community, it is preferable to design own benchmark tests since such tests are more representative to a typical system load. Hence, in our initial benchmark tests, we focused on benchmarking applications which have been developed in this project.

Language support for HPC is an active research area [6]. In this project, the HPC platform (WP1) aims at high-level compiled languages like C. As a high-level script language, MATLAB enjoys considerable popularity among scientific and engineering communities for its ease of use, code reusability, access to wide variety of toolboxes etc. Currently, most partners in this project have used MATLAB to develop their algorithms. Compiled MATLAB allows the use of MATLAB algorithms inside C applications. Thus it offers an opportunity in the project for the initial integration of algorithms on the HPC platform.

In this report, we will first discuss the benchmarking procedure and metrics used to evaluate the performance, and present the results of the initial benchmark tests for the Distribution State Estimator (DSE) and PD data pattern recognition applications. We will give a detail plan for tests of the Service Restoration algorithm (SRA) which will be conducted in year two. A tutorial on compiled MATLAB is provided as an appendix to this report.

Document Information

Project Number	FP7 - 248135	**Acronym**	HiPerDNO
Full Title	Report on the initial testing, benchmarking and comparison of novel DMS functionality as supported by HPC		
Project URL	http://www.hiperdno.eu		
Document URL	N/A		
Deliverable Number	D3.1.3	**Title**	Report on initial testing, benchmarking and comparison of novel DMS functionality as supported by HPC
Work Package Number	WP3	**Title**	Integration of HPC Architecture & Communications to Enable Novel DMS Functionality and to Provide Proof of Concept

Delivery	**Work Plan Date**	1/02/2012	**Actual Date**	25/01/2012
Status	Version 1.0			
Nature	Prototype ☐ Internal Report ☒ External Report ☐ Dissemination ☐			
Dissemination Level	Public ☒ Consortium ☐			
Author(s) (Partners)	IBM, Oxford, GTD			
Lead Author	**Name**	Rui Liao	**E-mail**	Rui.Liao@brunel.ac.uk
	Partner	Brunel	**Phone**	
Abstract	In this report, we first discuss the benchmarking procedure and metrics used to evaluate the performance. Results of initial benchmark tests for Distribution State Estimator (DSE) and PD data pattern recognition are given. We will give a detail plan for tests of Service Restoration algorithm (SRA) which will be conducted in year two. A tutorial on compiled MATLAB is provided as an appendix in this report.			
Keywords	HPC, Benchmarking			

Table of Contents

1 Introduction

WP3 is about integration of HPC architecture to enable novel DMS applications which have been developed in WP1 & 2. Benchmarking is an important step towards the full integration in the final year of the project. The goal of benchmark tests is to evaluate the performance of the HPC platform developed in WP1.2 to meet the novel DMS applications' computational needs. Benchmarking is a challenging subject in its own right, which requires careful planning to be an effective tool to decide the best system for specific applications. Close collaboration between partners in WP1, 2 & 3 is required to design an effective mechanism to perform quantitative evaluation. In general, there are two ways to perform benchmarking, e.g., adopting standard benchmark suits and tailored benchmark tests. While there are various benchmark suits [1-5] adopted by the HPC community, it is preferable to design own benchmark tests since such tests are more representative to a typical system load. Hence, in our initial benchmark tests, we focused on benchmarking applications which have been developed in this project.

Language support for HPC is an active research area [6]. In this project, the HPC platform (WP1) aims at high-level compiled languages like C. As a high-level script language, MATLAB enjoys considerable popularity among scientific and engineer communities for its ease of use, code reusability, access to wide variety of toolboxes etc. Currently most partners in this project have used MATLAB to develop their algorithms. Compiled MATLAB allows using algorithms developed in MATLAB inside C applications. Thus it offers an opportunity in the project for the initial integration of algorithms on the HPC platform. A tutorial on compiled MATLAB is provided as an appendix in this report.

In this report, we will first discuss the benchmarking procedure and metrics used to evaluate the performance. Results of initial benchmark tests for Distribution State Estimator (DSE) and PD data pattern recognition are given. We will give a detail plan for tests of Service Restoration algorithm (SRA) which will be conducted in next few months.

2 Benchmarking Procedure and Metrics

In order to decide the best setting of HPC systems for the required application, we need to evaluate the system's potential performance using benchmarks. In general, parallel computing can improve performance in a few different ways. First, it allows many independent serial jobs to run at once. For example, in order to study the health of the assets, we need to perform pattern recognition task for each individual asset. Secondly, it can speed up the calculation by parallelising the algorithms. Finally, large problem can be solved by running parallel versions of the algorithms which allow access to multiple processors and memories. In this section, we give the initial benchmark test procedure and evaluation metrics.

2.1 Benchmarking Procedure

At the beginning of each test, the algorithm under test resides on the compute nodes, while the data reside separately on the HPC Data Storage. The overall application consists of the Pelican server (work balancer) and the Pelican pipelines (algorithm, implemented the DSE). When requested, the scheduler will launch both the Pelican server and the pipeline processes on the compute nodes, which all connect to the Data Server. Table 3.1 gives the HPC benchmark test environment. In the tests, 'one pipeline' means that all zones are computed serially on one compute node and 'four pipelines' means that one pipeline is run on each compute node of the cluster; "sixteen pipelines" means that four pipelines are run inside each of four compute nodes (one pipeline per core).

Name		Specification	Name	Specification
Node	Processor	Inter Xeon (R) CPU 5150 @ 2.66GHz, 4 cores	Cluster	6 nodes cluster: 1 data node, 1 head node and 4 compute nodes
	RAM	4 GB memory		
Development Language		C	Operating System	64bit Debian Squeeze 6.0

Table 3.1. HPC benchmark test environment

Overall computation time is measured on the HPC Data System, starting with the first data retrieval from the Data System and ending when the output data is stored in the Data System. This give an

indication of the overall system throughput including the initialisation time the Pelican server used to access data before starting the pipelines, data access and computational costs. Code instrumentation complexity is also reduced. We believe that the procedure is conceptually correct as it measures the wall clock time elapsed between input data access and output becoming available. Overheads can be difficult to quantify. On our development system, the main effects are:

1. Initialisation: Pipeline launching and Pelican server data access are asynchronous.
2. Data access contention, particularly on outputs from the pipelines. The locations where each pipeline stores data on the Data System are independent.
3. Unbalanced load on the compute nodes due to different computational costs for different computational jobs.
4. Inherent serialisation in the Pelican server.

2.2 Benchmarking Evaluation Metrics

As discussed in D1.2.3 [7], there are two ways to measure the parallel performance of a given application, which are strong and weak scaling respectively. In this report, we focus on the strong scaling test. To test the performance of strong scaling, the size of the problem is fixed while the number of processing units is increased. We also study the effect of multi-threading setting using multi-core processor. The performance metric used for evaluation are listed as follows.

- Execution Time $T(n,p)$

Given a problem size n and the number of processors p, execution time is the amount of time a computation job requires.

- Scaling Factor $S(n,p)$: the ratio of execution time using one processor and using p processor.

$$S(n,p) = \frac{T(n,1)}{T(n,p)}$$

- Efficiency Ratio $E(n,p)$: the ratio of speed-up to the number of threads.

$$E(n,p) = \frac{S(n,p)}{p}$$

Most partners in this project used MATLAB to develop algorithms. Compiled MATLAB allows the use of algorithms developed in MATLAB, inside C applications. The process of using compiled MATLAB exists for Windows and Linux includes installing MATLAB compiled runtime MCR, compiling MATLAB scripts and creating the compile C application. The detail procedure can be found in the appendix.

3 Benchmark Tests Results

In this section, we give preliminary results for the benchmark tests for the application of Distribution State Estimator (DSE), Patter Recognition of PD data. For the application of Service Restoration Algorithm (SRA), we give a detail plan for the tests which will be implemented in the next few months.

3.1 Benchmark Test for Distribution State Estimator

As a result of close collaboration between EDF R&D and Oxford, the DSE algorithm has been ported to the HPC platform. The algorithm code (in MATLAB) has been compiled and integrated with the Pelican framework (in C++) so that it can be deployed on the HPC Platform. As discussed in D1.2.3 [7], as in most non-linear optimisation problems, it is impossible to provide bounds on computational intensity, e.g., number of operations, memory transfers, etc. This implies weak scaling analysis is very difficult to perform. In this section, we study the strong scaling test of the DSE algorithm on HPC where the performance results obtained from test problems of different sizes employing an increasing number of Pelican pipelines. The tests consist of network made up by zones. Table 1. gives the information of the number of zones and nodes per zone for each test. The zones data were provided by EDF R&D.

Test	No. Zones	Nodes Per Zone
1	20 (different)	11-100
2	32 (identical)	181
3	441 (identical)	181

Table 1. Number of zones and per zone for each test.

In the first two tests, we switched off MATLAB's multithreading functionality. We aim to study the scalability of the DSE approach rather than absolute performance. The latter requires further investigation.

1. Test 1: A small distribution network

This test involves 20 small zones with small number of nodes each. Not unexpectedly, we observed poor scalability increasing the number of Pelican pipelines. There is simply not enough computation to overcome the overheads, as discussed below.

No. Pipelines	Execution Tim (sec)	Scaling Factor	Efficiency (%)
1	4.79	1	100
4	2.99	1.60	40.05
16	3.28	1.46	9.12

Table 2. DSE benchmark test performance for a small distribution network

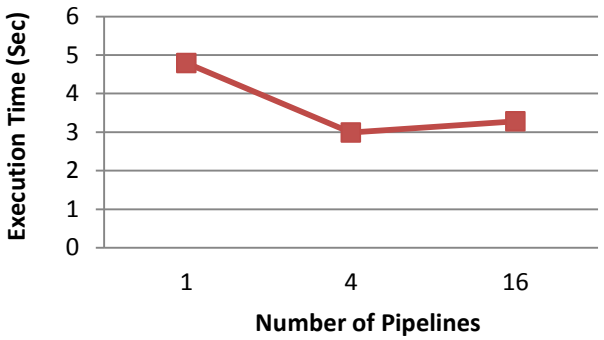

Figure 1. DSE execution time against number of pipelines for a small network

2. Test 2: A medium-size distribution network

We have observed better scalability than for the previous test. This confirms that overheads are relatively small with respect to the computation.

No. Pipelines	Execution Time (sec)	Scaling Factor	Efficiency (%)
1	84.69	1	100
4	23.16	3.65	91.41
16	8.5	9.96	62.27

Table 2. DSE benchmark test performance for a medium-size distribution network

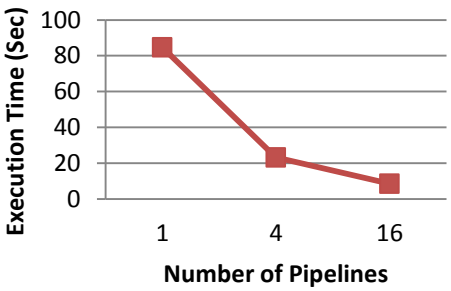

Figure 2. DSE execution time against number of pipelines for a medium-size network

3. Test 3: A large size distribution network

Figure and table 3 show the results for a large distribution network, consisting of about 80,000 nodes.

No. Pipelines	Execution Time [sec]	Scaling Factor	Efficiency [%]
1	1139.1	1	100
4	294.49	3.86	96.7
16	81.94	13.9	86.88

Table 3. DSE benchmark test performance for large-size distribution network

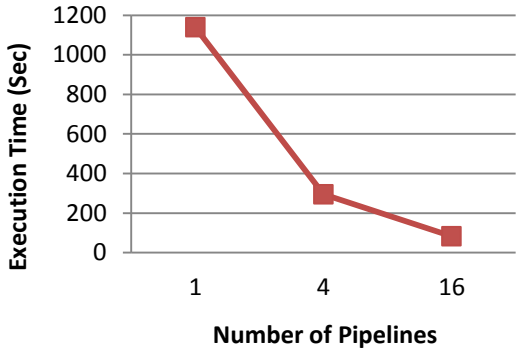

Figure 3. DSE execution time against number of pipelines for a large-size network

4. MATLAB Multi-threaded Test

The following Fig. 4 presents execution time of the second test set on one multi-core compute node. We compare effects of MATLAB's multithreading versus parallel execution of multiple Pelican pipelines. The first number on horizontal axis shows the number of pipelines on one node, the second whether MATLAB's multithreading is on or off.

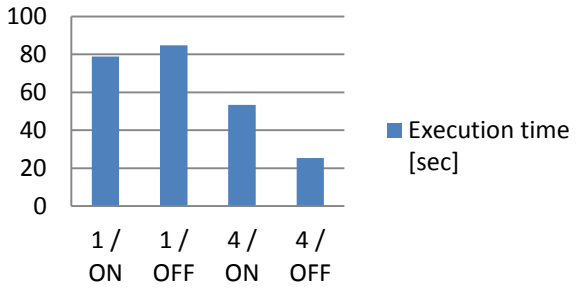

Figure 4. MATLAB Multi-threaded Test

3.2 Benchmark Tests of PD Data Pattern Recognition

In this section, we report on the results of initial testing of IPEC PD data analysis application on HPC. The purpose of the PD data analysis is to develop pattern recognition models which can be used to automatically detect the underlying structures within the data. The discovered patterns can be regarded as indicators to decide the integrity of the assets. The detail discussion of the algorithms can be found in deliverable D3.1.1 and D1.3.3 [8, 9]. In the benchmark test, we test a pattern recognition model to separate PD signals from multiple sources and noise. Pattern recognition in general contains two main steps which are feature extraction and data mining. Before proceeding to the discussion of the benchmark test, we first give a brief review of these techniques.

The pattern recognition model under test contains feature extraction (FE) and unsupervised clustering analysis. In the process of FE, Principal Component Analysis (PCA) is shown to be the suitable dimension reduction technique by extracting the majority of the variation in the original data set. In the process of cluster analysis, DBSCAN [10] has been found to be a suitable algorithm as it can be used to find clusters with any arbitrary shapes. Inputs of the model are the $m \times n$ data matrix X and the fixed number of clusters. Each data point will then be provided a label which indicates the cluster it belongs to. The pseudo-code of the model is given as follows.

Algorithm 3.2.1 Pattern Recognition Model to Separate PD Signals

Comment: Given a $m \times n$ data matrix X and k numbers of clusters, the algorithm is to provide labels for each data point.

Convert matrix X to Y:	$\mathrm{Y} = \frac{1}{\sqrt{\mathrm{n}}}\mathrm{X}^{\mathrm{T}}$;
Singular Value Decomposition of Y :	[U S V] = SVD(Y);
Best rank approximation:	d
Principle components:	PCs=Y*(the first d columns of V).
DBSCAN clustering:	Class=DBSCAN(PCs, k)

To study the performance of computational metric related to parallelism, we distinguished two scenarios. The first scenario is multi-threaded parallelism benchmark test in MATLAB running on a desktop PC with multi-core processors. MATLAB versions after MATLAB 7.4 (R2007a) support multi-threaded computation for a set of functions and expressions which are combinations of element-wise functions. Multi-threaded benchmark test will be used to show if the algorithm benefit from the multi-core processors. The second scenario is distributed parallelism benchmark test in HPC. The CM application involves analysis of PD data recorded for multiple numbers of assets. In order to

study the integrity of each individual asset, we need to perform analyse of the historical data for each asset. Such analysis is parallel in nature since each analysis is independent of each other.

1. Test 1: Multi-threaded Benchmark test for one-month and on-year data for a single asset

This test involves one month worth of historical data for one asset. The pattern recognition analysis for each asset can be considered as a serial job which can be implemented in one single pipeline. In Fig. 5 and 6 we can see that a dual core machine can reduce execution time by adopting multi-threaded parallelism. However, when the chosen number of threads >=2, performance begins to deteriorate. In Table 5 and 6 we can see increasing the number of threads from 2 to 4 actually reduces scaling factor which in turn gives a low efficiency.

Number of Threads	Execution Time (s)	Scaling Factor	Efficiency (%)
1	6.80	1.00	100
2	5.48	1.24	62
3	5.57	1.22	41
4	5.13	1.33	33

Table 4. PD data pattern recognition benchmark test performance for a small dataset

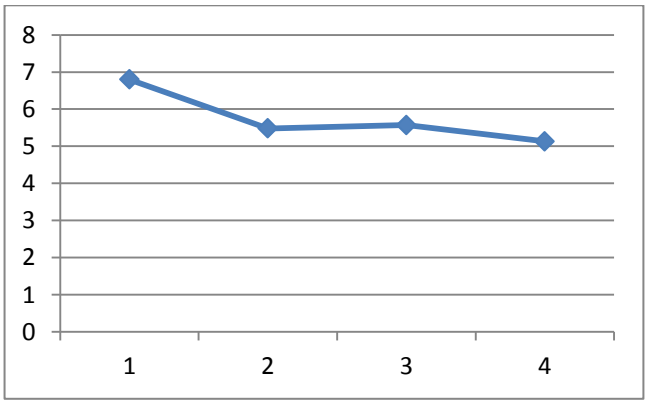

Figure 5. PD data pattern recognition execution time against number of threads for a small dataset

Number of Threads	Execution Time (s)	Scaling Factor	Efficiency (%)
1	221.18	1	100
2	170.38	1.3	65
3	165.07	1.34	45
4	153.72	1.44	40

Table 5. PD data pattern recognition benchmark test performance for a large dataset

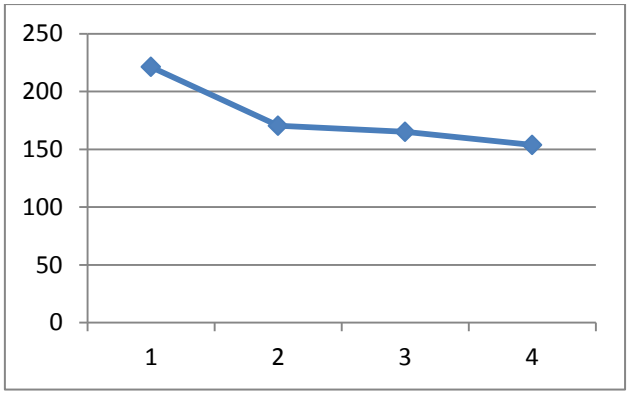

Figure 6. PD data pattern recognition execution time against number of threads for a large dataset

2. Test 2: Multi-threaded Benchmark test for a single asset with varied size of datasets

This test aims to study how the performance of multi-threaded setting in one pipeline scales with varied size of datasets.

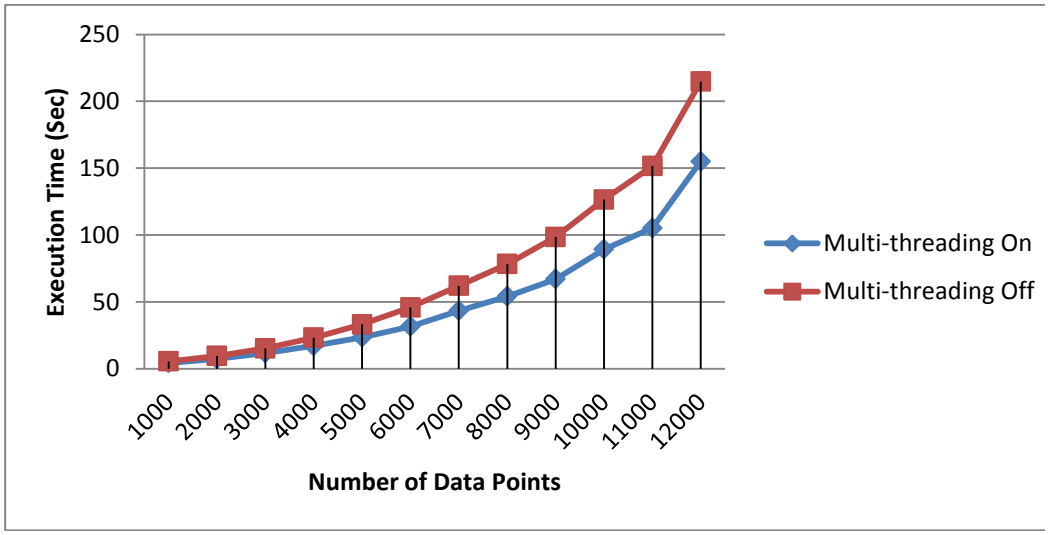

Figure 7. PD data pattern recognition execution time against number of data points

In Fig. 7, we change the size of the problem and plot the corresponding execution time with multi-threading setting on or off. The gap between the red curve and blue curve becomes larger with increasing size of data matrix. That is, the benefit of using multi-threaded parallelism for computation becomes more obvious. Hence increasing of execution time in a single core machine is faster than in a multi-core machine when dealing with large datasets.

3. Test 3: Benchmark test for one-year historical data of 50 assets

In this test, we perform the test on one-month data of 50 assets. The 50 independent jobs are run in the same time. The performance results in table 7 and figure 7 are obtained by varying the number of pipelines. During this test, multi-threading setting was switched on. In Fig 7, we can see execution

time reduces significantly when using multiple pipelines while the efficiency remains high as shown in Table 7.

Number of Pipelines	Execution Time (s)	Scaling Factor	Efficiency (%)
1	141.39	1.00	100
2	73.96	1.91	96
3	51.53	2.74	91
4	40.78	3.47	87

Table 6. PD data pattern recognition benchmark test performance for one-year historical data of 50 assets

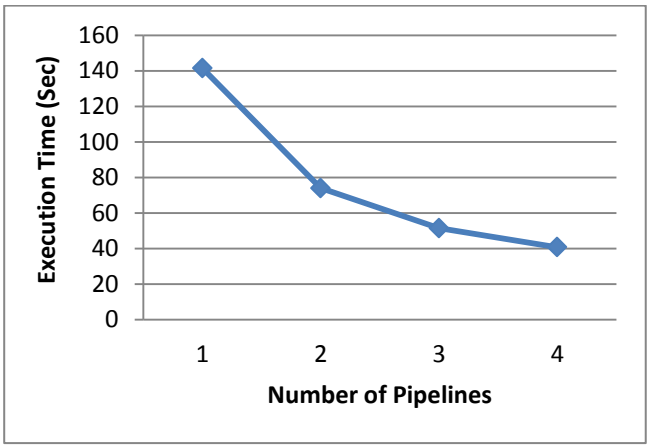

Figure 8. PD data pattern recognition execution time against number of pipelines

3.3 Benchmark Tests Plan for Service Restoration Algorithm

A prototype for a novel Service Restoration Algorithm (SRA) was developed in WP2.3. Over this initial prototype, low level HPC capabilities are under development, including using OpenMP and MPI features. A complete description of initial prototype can be found on deliverable D2.3.1 [11], a description of initial approach of HPC integration can be found on deliverable D3.2.3 [12], and a complete technical description of this integration will be included in deliverable D2.3.3.

On the other hand, full tests are planned to be performed in WP 4.2. These tests are designed to check both results of the process itself and performance of the algorithm. Regarding the performance, four versions of SRA will be compared, depending on how deep HPC integration they have:

- SRA with no HPC capabilities.

- SRA with OpenMP multithreading.
- SRA with MPI multiprocess.
- SRA with both OpenMP and MPI.

The tests will consist on repetitive launches of an executable (SRAE), developed ad-hoc for these tests. This executable reads some files containing the needed static and dynamic information about the fault scenario (network, switches, load scenario from State Estimator, etc.), and calls the SRA library that computes the restoration process. The scenarios files will be generated from UF corporate systems used for the trials (SEPLO), but the executable is intended to be run over an HPC platform which provides multithreading, MPI run time environment and Data Server. In this case OeRC form Oxford University will be the HPC environment used, by means of a remote ssh (Secure Shell) access from UF for execution, and a remote uploading of needed files to data server by means of scp (Secure Copy).

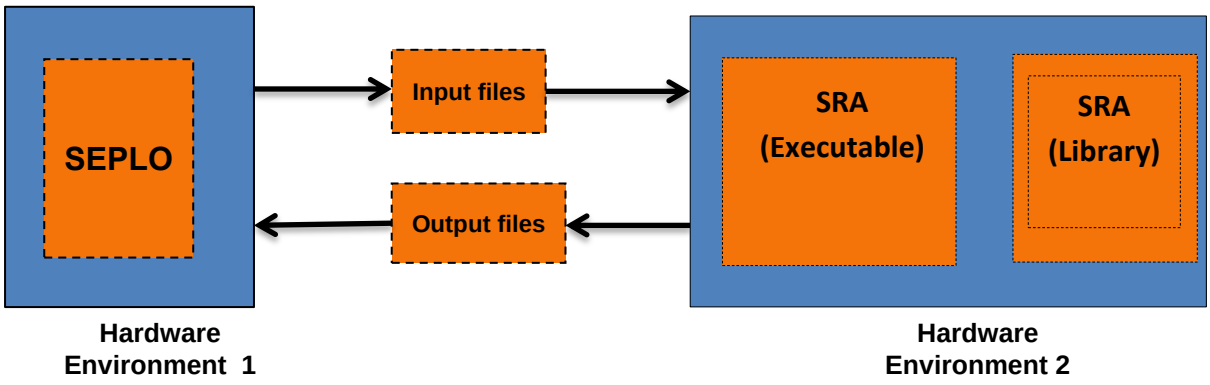

Figure 9. SRA benchmark test processes.

One key factor about SRA is that it is based on heuristic evolutionary algorithm of genetic type. This means that on every execution, even using the same inputs, the result could be slightly different, and the time needed can change significantly, depending on chance trying different solutions. For this reason, a unique execution is not enough to characterize performance on solving a specific input, but some statistical average values must be considered for a complete benchmarking of different versions of SRA.

Taking into account these considerations, SRAE performs the following operations:

- Reads the input files from data server.
- Calls SRA library n times, where n is a parameter defined in the call by the user.

- Retrieves the solution of SRA in each call to output files, and records general parameters of execution: time, number of iterations, final objective function achieved (cost and number of maneuvers), number of load flows computed, etc.
- Computes some statistics over general parameters, in order to retrieve some average indicators about performance and quality of results, and outputs these results to a different output file.

The number of executions in each launch will be adjusted in the preliminary phase of the trials. The detailed output files will be used to analyze the quality of solution in terms on how the restoration is proposed, and the other ones will be used for the process between the different versions.

Regarding HPC features, next additional considerations will be included:

- On OpenMP versions, the number of cores or sharing memory processors available is a key factor for drawing conclusions about how it affects performance.
- On MPI versions, the number of processors available and the number of them specified on MPIrun call is a key factor for drawing conclusions about how it affects performance.
- On all cases, the previous system load should be stable and equivalent on every execution.

An initial design of the trials was described in deliverable D3.2.2 [13], but final approach is still under discussion and will be detailed on deliverable D4.2.2. HPC platform description and design can be found in deliverables D1.2.1 [14] and D1.2.2 [15]

4 Discussion and Conclusion

Measurement results of these preliminary benchmark tests are in line with our expectations. For the DSE algorithm, we perform benchmark tests for three types of networks, i.e. small networks, medium networks and large networks. We can see the algorithm scales well for problems dealing with large data, i.e. the medium and large networks. It performs less well for a smaller network. This is consistent with the overheads discussed in Section 2. It is to be noticed that the Pelican framework allows for dynamic load balancing: zones will be processed by the first pipeline available rather than any fixed schedule. The last test shows that there is little scope to employ multi-threading in the MATLAB DSE code. Multithreading is not directly implemented in MATLAB, but through built-in functions. In the case of this application, the use of MATLAB built-in functions is too modest for multithreading to make any difference, hence it is much more efficient to switch MATLAB multi-threading off, in order to schedule more Pelican pipelines.

For the PD data pattern recognition, we perform multi-threaded tests for varied size of datasets. It shows that execution time can be reduced using multi-threading. However, efficiency will be reduced significantly when having more than two threads. For the DSE algorithm, it has been shown that it is more efficient to use more pipelines and switch off MATLAB multi-threading. In the case of PD data analysis, currently the pattern recognition algorithm for each asset is a serial job. Hence, running analysis for a single asset can only employ one pipeline, in which case having multi-core processor in the compute node is beneficial as multi-threading functionality can be applied. In the test on one-month data of 50 assets, by varying the number of pipelines the test shows clear that the execution time was reduced significantly when using multiple pipelines while the efficiency remains high.

In the light of our results, the effects of overheads can be summarised as follows.

1. Initialisation. We believe that this has a relatively minor effect, although very difficult to quantify. In any case, it depends to a large extent on the scheduler, resource manager, etc. that we employ in the development system. A fully developed system would ameliorate or altogether remove this.
2. Data access contention. The development system uses Gigabit Ethernet interconnects. That may lead to contention between Pelican server and pipelines as well as between pipelines and Data System. A fully developed Data System would employ multiple, parallel data servers for independent connection to the pipelines as well as possibly, multiple Pelican servers. It should

be noticed that we believe that the overheads caused by the Pelican server are minimal. This is also supported by our experience with Pelican in other fields.

3. Unbalanced loads. For the large and medium size networks, we used all identical zones; due to the limitations of the data we were provided. It should be noticed that allowing for scheduling of the zones as to maximise load balancing is not, in general, possible. Computation time for each zone depends not only on the size, but also on the mathematical and numerical structure of the problem: in all likelihood, this is not known in advance.

This deliverable has shown the initial work in benchmark test. More in-depth quantitative analysis will be performed in WP3 and WP4 in the final year of the project. Below we give the plan of further work.

1. MPI parallelism will be deployed for the DSE "overlapping zones". MPI parallelism is a more static approach to parallelism. It is expected to bring difficulties in load balancing, hence reduced strong scaling [7].
2. In Section 3.3, we have provided a detail plan for implementing SRA on HPC. SRA is a highly non-linear optimisation problem. OpenMP as well as hybrid MPI+OpenMP parallelisms will be expected to be deployed. We would expect reasonably good strong scaling, as at its core it relies on a genetic algorithm approach [7].
3. Due to the distributed nature of the assets, presently PD data analysis involves more trivial use of parallel computing, i.e., running multiple independent jobs in many pipelines. We expect correlation analysis will be implemented across assets which would require more sophisticated usage of parallel computing.
4. In [7], the Oxford team provides the following requirements for more in-depth benchmarking:
    - Identification of data transmission vs. computation bottlenecks. This may provide important information on system design, e.g. relative investment in interconnects and/or processing capabilities.
    - Limits on suitable parallelism, and identification of "time windows" within which the algorithms and applications proposed can provide answers. This could have some impact on DNOs deployment and use of HPC provided data.
    - Better and more accurate system sizing as currently. Note that current estimates are based on data provided from partners, not from direct observation on target systems.
    - Possible identification of algorithmic bottlenecks, with a view of optimising applications.

5 References

[1] Top500 at http://www.top500.org

[2] LINPACK at http://www.top500.org/about/linpack

[3] HPC Challenge at http://icl.cs.utk.edu/hpcc

[4] STREAM at http://www.cs.virginia.edu/stream/

[5] SPEC at http://www.spec.org

[6] C. Severance, K. Dowd, High Performance Computing, 2nd Edition, O'Reilly Media, July 1998

[7] D1.2.3 (M24): "Report on the interaction between software and hardware platforms and the impact of federating", University of Oxford, Jan, 2012

[8] D3.1.1 (M18): " Detailed specification report on HPC architecture and platform standardisation for developmental support of novel DMS functionality", Brunel, Sep, 2011

[9] D1.3.3 (M24): "Report on the initial testing, benchmarking and comparison of novel DMS functionality as supported by HPC", IBM, Jan, 2012

[10] M. Ester, H. Kriegel, J. Sander, X.W. Xu, " A DensityBased Algorithm for Discovering Clusters in Large Spatial Databases with Noise", The Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA, 1996

[11]. HiPerDNO report D2.3.1 (M12): "Functional Design, Specification and Prototype Development of an Algorithm for Distribution Network Service Restoration", INDRA, HiPerDNO, February 2011.

[12]. HiPerDNO report D3.2.3 (M24): "Working paper on proto-standards for data acquisition for processing with an HPC toolset", Brunel University, HiPerDNO, January 2012.

[13]. HiPerDNO report D3.2.2(M18): "Specification for standard network field trials carried out in WP4", Union Fenosa Distribución, July 2011.

[14]. HiPerDNO report D1.2.1 (M16): "Report on the architecture and performance criteria used for selection", University of Oxford, HiPerDNO, Version 2 – June 2011.

[15]. HiPerDNO report D1.2.2(M18): "Prototype deployment of selected architecture and necessary documentation", University of Oxford, HiPerDNO, June 2011.

6 Appendix Compiled MATLAB Tutorial

Author: Piotr Łopatka

1. Introduction

This document explains how to compile MATLAB scripts into C libraries and standalone applications. Currently in the HiPerDNO project the algorithms (WP2) are developed mainly in MATLAB, while the HPC platform (WP1) aims at high-level compiled languages like C. Compiled MATLAB allows using algorithms developed in MATLAB inside C applications. Thus it offers an opportunity in the HiPerDNO project for the first integration of algorithms on the HPC platform. Compiled MATLAB exists for Windows and Linux. This document focuses on the Linux version of MATLAB, because this is the target platform for the HPC system.

2. MATLAB Compiled Runtime

MATLAB Compiled Runtime (MCR) is a standalone set of libraries that enables execution of MATLAB applications on computers without MATLAB installed [1]. It is a part of standard MATLAB installation, "disabled" by default, and it is stored in a compressed format in a MATLAB directory tree:

MATLABDir/R2010b/toolbox/compiler/deploy/glnxa64/MCRInstaller.bin

- Installing MCR

Let us call the machine where MATLAB is installed the "host". The machine on which the MCR will be installed is the "target" machine. It is possible that the host and the target are the same machine. If another machine is selected as target machine, it must have the same architecture (x86, amd64) and the same type of operating system installed (Windows or Linux). Even for Linux, we suggest using the same distribution to avoid potential problems. To install MCR type on a target machine:

sudo sh MCRInstaller.bin

Default options can be confirmed during the installation. The installation of MCR will claim more than 500MB of disk space. After installation, set up appropriately the environment variables *LD_LIBRARY_PATH* and *MATLAB_ROOT*. It can be placed in *.bashrc* file to make settings permanent for the user.

export MATLAB_ROOT=/opt/MATLAB/MATLAB_Compiler_Runtime/v714

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:

$MATLAB_ROOT/runtime/glnxa64/:

$MATLAB_ROOT/bin/glnxa64:

$MATLAB_ROOT/sys/os/glnxa64:

$MATLAB_ROOT/sys/java/jre/glnxa64/jre/lib/amd64/native_threads:

$MATLAB_ROOT/sys/java/jre/glnxa64/jre/lib/amd64/server:

$MATLAB_ROOT/sys/java/jre/glnxa64/jre/lib/amd64:

export XAPPLRESDIR=$MATLAB_ROOT/X11/app-defaults

3. Compiling MATLAB Scripts

The compilation of MATLAB code takes place only on the host machine with MATLAB installed. MATLAB scripts can be compiled into libraries or standalone applications.

- C Shared Library

In order to compile MATLAB into a shared library type:

mcc –B csharedlib:libName {list of .m files}

Even for small MATLAB scripts compilation takes significant amount of time. The result of compilation consists of:

library file: libName.so

C header files: libName.h

Other files: readme.txt, libName.c

In order to use the library a C application must be developed and linked to the generated library (see further sections).

- Standalone Application

In order to compile standalone application type:

mcc –m {list of .m files} –o outputName

The result of compilation consists of:

An executable: outputName

A calling script: run_outputName.sh

In order to run and test the compiled application, type:

sh run_outputName.sh $MATLAB_ROOT {args}

The *$MATLAB_ROOT* is an environment variable set before and *{args}* is a list of arguments to be passed to the application.

4. Creating and Compiling C Application

The application should be developed on the host machine with MATLAB installed. In the application source file (*ex. test.c*) header files (*ex. libName.h*) should be included. The C application must initialize the MCR environment before any call to the compiled library is made. Later, in this document, we present the example source code for MATLAB script and C application.

In order to compile a C application type:

mbuild test.c –L. libName.so –I.

It will generate an executable "test" which can be executed. In order to run the application on a target machine different from the host machine the executable and the library file must be copied to the target machine.

Example: Standalone application

The following code presents example standalone application.

Script FuncMain.m

```
fprintf('This is a standaalone App\n');
z=Func1(5,4);
fprintf('The result is %d\n',z);
```

Function Func1.m:

```
function [ c] = Func1( a,b )
c=a+b;
end
```

To compile the code, type: *mcc –m FuncMain.m Func1.m –o test*

To execute the code (no parameters are passed to the application), type:

sh run_test.sh $MATLAB_ROOT

Example C-MATLAB application

The following files present MATLAB and C code.

Function AddFunc.m:

```
function [ sum ] = AddFunc(x,y)
sum = x + y;
end
```

C code test.c:

```
#include <stdio.h>
#include "libTest.h"

int main(void)
{
    mxArray *mX,*mY;
    mxArray *mSum=NULL;
    double x[]={1.5,2.5};
    double y[]={3.5,4.5};
    double sum[]={0,0};
    mclInitializeApplication(NULL,0);
    if (!libTestInitialize())
    {
        printf("Cannot initialize the library properly\n");
        return -1;
    }
```

```
    mX=mxCreateDoubleMatrix(2,1,mxREAL);
    mY=mxCreateDoubleMatrix(2,1,mxREAL);
    memcpy(mxGetPr(mX),x,2*sizeof(double));
    memcpy(mxGetPr(mY),y,2*sizeof(double));
    mlfAddFunc(1,&mSum,mX,mY);
    memcpy(sum,mxGetPr(mSum),2*sizeof(double));

    printf("\nThe result of computation is =[%f,%f]\n",sum[0],sum[1]);
    libTestTerminate();
    mxDestroyArray(mX); mX=NULL;
    mxDestroyArray(mY); mY=NULL;
    mxDestroyArray(mSum); mSum=NULL;

    mclTerminateApplication();

    return 0;
}
```

The C code shows how:

The MCR environment is initialized from C

How the compiled library is initialized

How to create MATLAB arrays, C arrays, and how to copy the data between them

How to call a compiled MATLAB function from C

To compile the MATLAB script, type: *mcc –B csharedlib:libTest AddFunc.m*

To build the C application (requires MATLAB compilation step first):

mbuild test.c –L. libTest.so –I.

To run the application type: ./test