SD3 ParaDIME Application Selection Document

D5.1 Applications Selection Document, For Target Success Criteria, see SD1

Version 1.0

Document Information

Contract Number	318693
Project Website	www.paradime-project.eu
Contractual Deadline	Month 6 (30 March 2013)
Dissemination Level	Public
Nature	Report
Author	Anita Sobe (UNINE), Derin Harmanci (UNINE), Yaroslav Hayduk (UNINE)
Contributors	Thomas Knauth (TUD), Oscar Palomar (BSC), Arindam Mallik (IMEC), Adrian Cristal (BSC), Osman Unsal (BSC), Pascal Felber (UNINE), Wojciech Barczynski (AoTerra), Gina Alioto (BSC)
Reviewer	Oscar Palomar / Gina Alioto (BSC)
Keywords	benchmark, real-world application, selection matrix

Notices:
The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under the ParaDIME Project (www.paradime-project.eu), grant agreement n° 318693.

Change Log

Version	Description of Change
v1.0	Initial Draft released to the European Commission (based on internal v15).

Table of Contents

1 Introduction and executive summary

The SD3 ParaDIME Application Selection document originates from the D5.1 Application Selection Document; however, it does not include Target Success Criteria. Instead the Target Success Criteria for all layers of the stack have been consolidated in a separate deliverable, SD1.

The objective of this document is to identify the initial benchmark and application that we will use to evaluate the ParaDIME programming model and associated methodologies employed at the runtime (WP4), hardware architecture (WP3) and device levels (WP2). More specifically, the document defines which applications and benchmarks we will initially utilize to evaluate the overall energy savings and performance trade-offs for the complete ParaDIME computing stack as well as to determine the level of success attained with respect to the targets described in SD1 Target Success Criteria Document. A secondary objective of the document is to briefly explain the process and related criteria used to make this initial selection and potential additional selections in the future. This document does not discuss in detail the individual benchmarks that will be used to evaluate the individual components of the ParaDIME stack; these benchmarks will be explained as a part of the evaluation documents for their respective components.

2 Benchmark

2.1 Initial Approach

We began our selection process by surveying a wide range of algorithms and their available implementations (as benchmarks) with respect to a set of high level, "non-starter" criteria. In other words, we would not further consider any benchmark that did not meet these initial criteria which can be summarized as follows:

1. The algorithms (implemented as benchmarks) and applications must be parallelizable while at the same time easily implementable in shared memory.
2. The benchmark / application must have few enough dependencies in order to be able to apply message passing or the actor model.
3. Energy must be a key factor; the benchmark / application must be deployable on an energy-critical platform that would allow us to clearly measure and demonstrate a power / performance trade-off.
4. The benchmark / application allows for approximate data types (necessary for approximate computing).

These criteria significantly narrowed the scope of available applications and allowed us to focus on other important factors, including the importance of having direct access to the source code and, if possible, good access to the team working on the implementation if possible. For the benchmarks, we also focused on portability; in other words, the more implementations that already exist, the better - particularly when they exist as a part of a benchmark framework. Additionally, we looked at the configurability of the application with respect to input sizes and execution times which would ensure the best coverage possible for testing the various methodologies across the ParaDIME Stack. From this initial surveying process, we came up with

four possible candidates for benchmarks. These candidates are described in detail in the sections to follow.

2.1.1 K-means

The *K-means algorithm* groups objects in an N-dimensional space into K clusters. It begins by picking a random set of cluster centers and performs the following operations to assign points to clusters in an iterative manner: (i) compute the distance between every point and every cluster center, (ii) assign each point to the cluster with the closest center, (iii) re-compute the new center for each cluster as the mean of all the points in the cluster. The majority of execution time is spent for calculating the new cluster centers and data sharing. In its multi-core implementation, the amount of contention among threads depends on the value of K, with larger values resulting in less frequent conflicts. Since threads only occasionally update the same center concurrently, this algorithm is not embarrassingly parallel and may benefit from optimistic concurrency. Also, Chakradhar and Raghunathan [1] have shown the use of the algorithm for image segmentation under approximate computing. The K-means algorithm is implemented and available for transactional memory as **kmeans** in the STAMP benchmark suite (C, Java) [2] and for GPUs (OpenMP and OpenCL implementations) in the Rodinia benchmark suite (a suite proposed for heterogeneous platforms [3] — especially those including GPU accelerators). Moreover, there are implementations using message passing (MPI).

2.1.2 Genome

The *genome assembly algorithm* takes a large number of DNA segments and matches them to reconstruct the original source genome. There is a multi-threaded implementation available as **Genome**, one of the benchmarks of the STAMP suite [2] (C, Java). Genome first utilizes a hash set to create a set of unique segments (i.e., to remove duplicates). In the second step, each thread tries to remove a segment from a global pool of unmatched segments and add it to its partition of currently matched segments. Transactions may be used for the addition to the set of unique segments and for accessing the global pool of unmatched segments. Overall, these transactions are of moderate length with few reads and writes. Additionally, nearly all of the execution time is transactional, and there is little contention.

2.1.3 K-d tree

The *K-d tree algorithm* builds a k-d tree, which is a balanced tree, in *O(n*log(n))* time using random coordinate points. *K-d tree* is implemented in the Spec OMP 2012 (C++) as **K-d tree**. At a high level, the K-d tree application builds the k-d tree and then searches for points that are proximate to each point in the tree. The build phase is single threaded, but the search phase in this implementation is multi-threaded using the OpenMP task directive. Once the k-d tree is built, the k-d tree is walked to visit each point, and that point is used as a query point to search the k-d tree for all other points that lie within a specific radius of that query point. The default value for that radius is one-tenth the range of the random numbers. The total number of points found by using each point successively as a query point and the total execution time are reported. Walking and searching of the k-d tree imply two recursive traversals of the k-d tree. It is important to note that the Spec OMP 2012 (C++) provides energy-related benchmarks, which is an added benefit of this K-d tree implementation. However, the benchmark suite is not open source, and thus not freely available.

2.1.4 Heart wall tracking

The **Heart wall tracking application** tracks the changing shape of the walls of a mouse heart over a sequence of 104 ultrasound images, each with a resolution of 609×590 pixels. The application has several stages. In the first stage, the application performs various image processing passes, including edge detection, SRAD despeckling, morphological transformation, and dilation. In its final stage, the program tracks the changing shapes of the two heart walls by detecting the movement of certain sample points throughout the sequence of images. Heart wall tracking presents a pattern of braided parallelism which is a mixture of data and task parallelism. The application is coarsely parallelized according to independent tasks (TLP); each task is then finely parallelized according to independent data operations (DLP). The processing of a frame is implemented as a single GPU kernel in order to successfully implement braided parallelism and avoid kernel launch overhead. This structure requires the inclusion of some non-parallel computation into the kernel, leading to a slight warp under-utilization but overall greater performance. Additionally, the use of image processing may be interesting in terms of the use of approximate computing. The application is part of Rodinia benchmark suite [3] and is available in C, OpenMP, CUDA and OpenCL.

2.2 Final Selection

Having selected the initial candidate benchmarks, we next worked with representatives from the other layers of the ParaDIME Stack to define a set of additional requirements based on the methodologies or "features" that we hoped to implement and test. We assigned an A to all "must comply" requirements as a means of disqualification; those applications and benchmarks that do not comply with A requirements have been marked in gray. In addition, we assigned a B to all "nice-to-comply" requirements and applied points for each. Then we scored each benchmark and application's respective ability to meet each requirement. A summary of the scoring for each of the candidates can be found in the table below with the associated discussion to follow.

REF	PRIORITY	METHODOLOGY	REQUIREMENT SCORING	K-means	Genome	kdtree	Heartwall tracking
1	Non-starter 1	Efficient message passing	ProgModel: Is the code parallelizable? (Y=1)	1	1	1	1
2	Non-starter 1	Efficient message passing	HW: Is there a Shared Memory version that we can use (Y=2) or can it be easily impemented in Shared Memory? (Y=1)	2	2	2	2
3	Non-starter 3	GENERAL	Is the application deployable on an energy-critical platform that would allow us to clearly measure and demonstrate a power / performance trade-off? (Y=1)	1	1	1	1
4	Non-starter 4	Approximate computing	ProgModels: Does it allow appoximate data types? (Y=1)	1	1	1	1
5	Non-starter 2	GENERAL	ProgModel: Does the application demonstrate limited dependencies / low contention (limited sharing of global data)? (Y=1)	1	1	1	1

REF	PRIORITY	METHODOLOGY	REQUIREMENT SCORING	K-means	Genome	kdtree	Heartwall tracking
6	A	GENERAL	ProgModel: Is the level of contention low enough; what is the level of contention? (low=1, med=0, high=-1)	1	1	0	1
7	A	GENERAL	Is the benchmark well-known (in terms of notoriety and understanding) by Project Participants and external communities at all concerned layers (Programming Models, Runtime and HW) for the Computing Stack? (Y=1, APP=NA)	1	0	1	0
8	A	GENERAL	ProgModel: Is the language easy-to-port to the Programming Model (Easy/Java=2, Moderate/C=1, Difficult/C++=0)?	2	2	0	1
9	A	GENERAL	ProgModel: Is the SW opensource (Y=1) or do we have direct access to the sourcecode (Y=1), Can we purchase the sourcecode (Y=0)?	1	1	0	1
10	A	GENERAL	ProgModel: What is order magnitude of lines of code of the non-GUI SW? (For BM<600 = 2, BM<1000 = 1, BM>1000 = 0, For APP<600=0, APP<1000 = 1, APP>1000=2)	2	1	0	0
11	A	GENERAL	HW: Does the app stress the memory bandwidth enough? (BM Y=1, APP=NA)	1	1	1	0
12	A	Efficient message passing	ProgModel: Can we configure multiple levels of nesting? (Y=1)	1	1	1	0
13	A	Heterogeneous computing (Accelerators)	HW: Is there data-level parallelism? (Y=1)	1	1	1	1
14	A	Carbon-aware scheduling	Is the minimum application execution time at least several minutes - hours? (Y=1)	1	1	1	1
15	B	GENERAL	HW: Can the benchmark support vectorized instructions out of complex patterns? (BM Y=1, APP=NA)	0	0	1	0
16	B	Efficient message passing	ProgModel: Is the algorithm already implemented in a message passing programming model (MPI)? (Y=1)	1	0	0	0
17	B	GENERAL	SysSW: Are the sizes of the reference input set(s) "BIG DATA", filling multiple disks, in order of TB? (Y=1)	0	0	0	0
18	B	GENERAL	HW: Is the benchmark Memory Latency bound or not? HW would like an example of both so no scoring is required. (BM Y/N, APP = NA)	N	Y	Y	N
19	B	GENERAL	HW: Does the benchmark use complex data structures? HW would like an example of both so no scoring is required. (BM Y/N, APP = NA)	N	N	Y	N
			TOTAL	18	15	12	11

Per the scoring, there are several clear advantages to using **kmeans**. Kmeans is available in many different programming languages and benchmarks, and is well known in the targeted communities. In addition, it fulfills more of the requirements of the different ParaDIME layers than any other benchmark. On the whole, due to its relative size and complexity, combined with its portability, we believe it to be an excellent fit for evaluating ParaDIME.

Genome like **kmeans** is also part of the popular STAMP benchmark which suggests that it is portable; however, there are not as many supported languages for Genome as there are for k-means. Moreover, in comparison, the k-means application (as well as the underlying algorithm) is much better known than the *Genome* application.

K-d tree is of interest to the project, particularly due to the fact that the SPEC benchmark in general covers energy efficiency metrics. This is due to the fact that communication between threads is low, which means that it lends itself to low power implementation. That said, SPEC OMP is not freely available, which means that we would need to purchase a license for the sourcecode if we want to know the underlying implementation details to the extent that we need to know them for the project.

Finally, while the **heart wall tracking application** is also an interesting application, it is likely to be too complex for an initial use case. Implementations exist in C, OpenMP, CUDA, etc., however the GPU implementations include Lisp-Code, which could pose problems when porting to the programming model/API.

3 Application

3.1 Initial Approach

We took the same initial approach to selecting a real-world application as in defining a benchmark by again narrowing our selection utilizing the baseline criteria. We surveyed a wide range applications that met the same set of high level criteria and from this initial surveying process, we came up two possible candidates. These candidates are described in detail in the sections to follow.

3.1.1 Simulation of the hydraulic properties of the subsurface

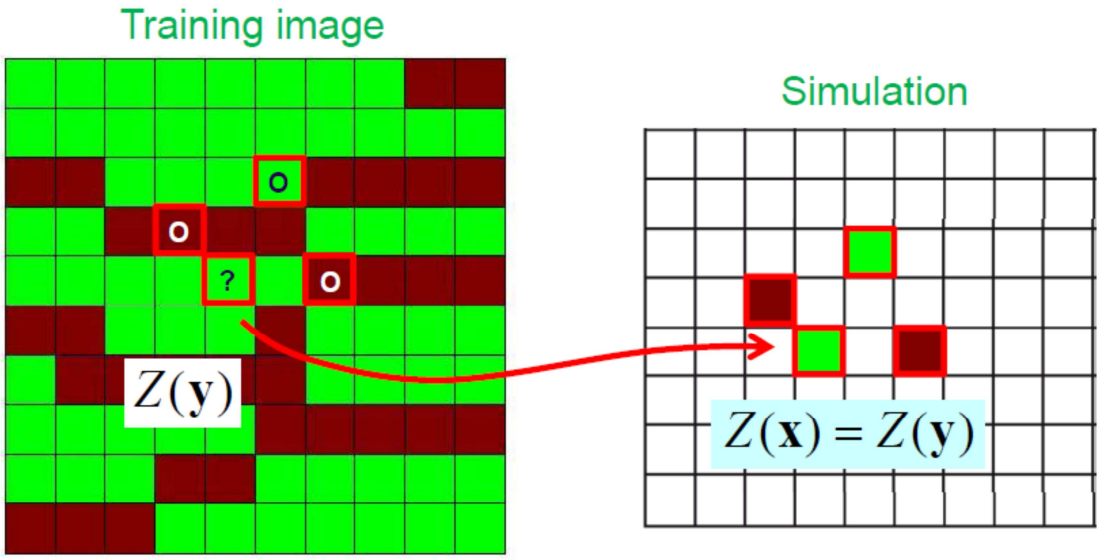
(In cooperation with Prof. Philippe Renard at University of Neuchatel, Switzerland)

Multiple-point geostatistics [4] is a prominent tool that has proven effective for performing geostatistical simulations. At its core, the technique analyzes the relationships between multiple variables in several locations at a time. In general, the cost associated with the deterministic determination of the hydraulic properties of the subsurface is prohibitively high. Hence, the aim of multiple-point geostatistical simulations is to simulate the hydraulic properties of the subsurface based on a given number of samples.

Multiple-point geostatistics uses the Direct Sampling [5] algorithm. As shown in the figure below, a simulation consists of a training image and a simulation image.

At startup, random samples are taken from the training image. More specifically, the algorithm selects a random starting point in the training image and then tries to find matching points in the simulation image. Here, the cells marked with “o” in the

training image match the same three points in the simulation image. The last point, $Z(\mathbf{y})$, of the training image will be copied to the simulation image to complete the pattern ($Z(\mathbf{x}) = Z(\mathbf{y})$).



When the simulation is performed sequentially, it imposes unrealistic constraints on execution time, limiting its use in practice. Hence, a number of attempts have been made to reduce the execution time by parallelization and the use of message passing.

3.1.2 FREVO - (FRamework for EVOlutionary design)

(In cooperation with Prof. Wilfried Elemenreich, Alpen-Adria Universität Klagenfurt, Austria)

FREVO provides a component-wise separation of the key building blocks for evolutionary computing into: *problem*, *representation*, *optimization method* and *ranking*. Each component can be developed and tested separately and reused for new projects. FREVO comes with a graphical user interface allowing the engineer to pick the particular components for a project. This component concept also supports evaluation of different configurations, for example, in order to see which controller representation (e.g., neural network vs. finite state machines) works better for solving a given problem [6].

FREVO utilizes evolutionary algorithms to perform search in a high dimensional space, typically$\mathbb{R}^n \rightarrow \mathbb{R}^m$, with *n* being the number of input dimensions (e.g., the number of weights of a neural network) and *m* being the number of parameters to be optimized. In many cases, a fitness function $f: \mathbb{R}^m \rightarrow \mathbb{R}$ is applied to collapse the dimension of the output parameters. In problems with competitive evaluation, there is no absolute fitness function but a possibility to get a ranking of multiple candidates being matched against each other.

Since the search can allocate a lot of processing time, parallelization is advisable. The evaluations within a generation are easily parallelizable. However, for problems with competitive fitness, there are particular dependencies that require communication. Moreover, missing synchronization causes either a longer calculation time for a

correct selection or produces a non-optimal selection of the best genomes for the next generation.

Due to the robust nature of evolutionary algorithms, a small amount of error can be tolerated in the selection (i.e., approximate data structures are supported [7]). Moreover, it is possible to trade-off between accuracy and time in both sorting and selection of FREVO. Approximate sorting algorithms exist and can possibly be distributed [8]. That said, the execution time and success of an evolutionary algorithm depends on the quality of the selection algorithm.

Although the evaluation aspect (i.e., calculating the fitness function) of the evolutionary algorithms of FREVO can be highly parallelized, the complexity of determining the fitness for a given genome and a potentially high number of required population sizes times the number of generations may result in an excessively long completion time for an evolutionary algorithm applied to a complex problem.

3.2 Final Selection

Once again having selected the initial candidate benchmarks, we next worked with representatives from the other layers of the ParaDIME Stack to define a set of additional requirements and a means of scoring each benchmark or applications ability to meet each requirement. A summary of the scores for each of the candidates can be found in the table below with further discussion to follow.

REF	PRIORITY	METHODOLOGY	REQUIREMENT SCORING	FREVO	Hydraulic
1	Non-starter 1	Efficient message passing	ProgModel: Is the code parallelizable? (Y=1)	1	1
2	Non-starter 1	Efficient message passing	HW: Is there a Shared Memory version that we can use (Y=2) or can it be easily impemented in Shared Memory? (Y=1)	2	2
3	Non-starter 3	GENERAL	Is the application deployable on an energy-critical platform that would allow us to clearly measure and demonstrate a power / performance trade-off? (Y=1)	1	1
4	Non-starter 4	Approximate computing	ProgModels: Does it allow appoximate data types? (Y=1)	1	1
5	Non-starter 2	GENERAL	ProgModel: Does the application demonstrate limited dependencies / low contention (limited sharing of global data)? (Y=1)	1	1
6	A	GENERAL	ProgModel: Is the level of contention low enough; what is the level of contention? (low=1, med=0, high=-1)	0	1
7	A	GENERAL	ProgModel: Is the language easy-to-port to the Programming Model (Easy/Java=2, Moderate/C=1, Difficult/C++=0)?	2	1
8	A	GENERAL	ProgModel: Is the SW opensource (Y=1) or do we have direct access to the sourcecode (Y=1), Can we purchase the sourcecode (Y=0)?	1	1
9	A	GENERAL	ProgModel: Do we have direct access to the development team (BM = NA, APP: Y=1)	1	1

REF	PRIORITY	METHODOLOGY	REQUIREMENT SCORING	FREVO	Hydraulic
10	A	GENERAL	ProgModel: What is order magnitude of lines of code of the non-GUI SW? (For BM<600 = 2, BM<1000 = 1, BM>1000 = 0, For APP<600=0, APP<1000 = 1, APP>1000=2)	0	2
11	A	Efficient message passing	ProgModel: Can we configure multiple levels of nesting? (Y=1)	0	1
11	A	Heterogeneous computing (Accelerators)	HW: Is there data-level parallelism? (Y=1)	1	1
12	A	Energy efficiency via increased utilization to deal with peaks	SysSW: Does the application include a mix of batch and interactive jobs? (BM=NA, APP Y=1)	0	1
13	A	Energy proportionality	SysSW: Can the application be extended to demonstrate networking with server and client to measure latencies or not? (BM=NA, APP Y=1)	0	1
14	A	GENERAL	SysSw: Does the application have a comand-line interface, or if it has a GUI, is it configurable to run with and without GUI - headless? (Y=1)	1	1
15	A	Carbon-aware scheduling	Is the minimum application execution time at least several minutes - hours? (Y=1)	1	1
16	B	GENERAL	General: Does the application / GUI lend itself for demonstration? (Y=1)	0	1
17	B	Efficient message passing	ProgModel: Is the algorithm already implemented in a message passing programming model (MPI)? (Y=1)	0	0
18	B	GENERAL	SysSW: Are the sizes of the reference input set(s) "BIG DATA", filling multiple disks, in order of TB? (Y=1)	0	0
			TOTAL	13	19

At the highest level, both applications lend themselves for use as a real-world application in the project. Both are scientific programs that are already implemented and currently available from developers with close ties to the consortium. This means that it is not necessary to implement either of them from scratch. Moreover, both applications are actively maintained which is important for the ParaDIME results to be of interest. However, at this point, the advantages of the applications diverge.

The **simulation of the subsurface** application is already implemented in CUDA and has a graphical user interface that allows for executing a mixture of batch and interactive tasks. Additionally, the base implementation of this application is of the appropriate size and complexity. Finally, the application lends itself for possible use in a data center, making it particularly desirable for testing the storage API which will be provided by the runtime. This is particularly important, because none of the aforementioned benchmarks is apt for testing this.

The main advantage of **FREVO** is that it is a framework, where programmers can define the agent behaviors that should be evolved. This means that we could vary the complexity of the program and the use of a graphical interface. Moreover, its size

would simplify the porting to the programming model / API. However, FREVO does not easily lend itself for use in a data center which makes it the lesser candidate.

4 In Summary

Per the discussions in previous sections, we favor **kmeans** as the initial benchmark to use with ParaDIME. It is available in many programming languages and is therefore easy to port. It is not very complex, i.e., there is no need to transfer complex data structures when performing message passing. Its input size is further configurable.

For the applications, we favor the **simulation of the subsurface** application, because it meets the data center-related requirements from the runtime (SysSW), which cannot be met by the benchmark. Furthermore, it is easier to implement a distributed version of this application in comparison to FREVO.

5 Bibliography

[1] S. T. Chakradhar and A. Raghunathan, "Best-effort computing: re-thinking parallel software and hardware," in *Proceedings of the 47th Design Automation Conference DAC*, 2010.

[2] C. C. Minh, J. Chung, C. Kozyrakis and K. Olukotun, "STAMP: Stanford Transactional Applications for Multi-Processing," in *Proceedings of the IEEE international symposium on workload characterization IISWC*, 2008.

[3] S. Che, J. W. Scheaffer, M. Boyer, L. Szafaryn, L. Wang and K. Skadron, "A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads," in *IEEE International Symposium on Workload Characterization IISWC*, 2010.

[4] F. Guardiano and R. Srivastava, "Multivariate geostatistics: beyond bivariate moments," in *Geostatistics Troia 92*, Kluwer Academic Publishers, Dordrecht, 1993, p. 133–144.

[5] M. Gregoire, P. Renard and J. Straubhaar, "The Direct Sampling method to perform multiple-point geostatistical simulations," *Water Resources Research*, vol. 46, no. 11, 2010.

[6] A. Sobe, I. Fehervari and W. Elmenreich, "FREVO: A Tool for Evolving and Evaluating Self-organizing Systems," in *Proceedings of the 1st International SASO Workshop on Evaluation for Self-Adaptive and Self-Organizing Systems*, 2012.

[7] Y. Matias, J. S. Vitter and N. E. Young, "Approximate Data Structures with Applications," *CoRR*, 2002.

[8] W. Elmenreich, T. Ibounig and I. Fehervari, "Robustness versus Performance in Sorting and Tournament Algorithms," *Acta Polytecnica*, pp. 7-18, 2009.