Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Improvement of the Uncertainty Analysis Platform URANIE for High Performance Computing

Miroslaw Kupczyk[a*], Damian Kaliszan[a], Vincent Bergeaud[b]

a Poznan Supercomputing and Networking Center, Dabrowskiego 79a, 60-529 Poznan
[b] CEA Saclay, DM2S, STMF, LGLS, 91191 Gif-sur-Yvette Cedex, France

Abstract

The work undertaken in the PRACE project consists of making the software evolve so that it is suitable for exploitation of design of experiments implying thousands of cores in multiple contexts: serial codes, parallel codes, coupled simulations.
The main work which is focused on a serial code context is about to optimize the application to process more efficiently on a bigger number of cores and the corresponding amount of input data. We rely on a strategy that uses the fork mechanism provided by the Linux kernel to deploy different computations and checking finalization of each of the forked processes.
Due to URANIE's internal architecture, which makes a single computation indivisible, it is difficult to properly examine the scalability of the system in terms of the strong and weak scalability definition. However, the promising results were obtained after tens of test-runs. The work carried out in this project allowed running URANIE codes on thousands cores on Tier-0 architecture. The scenario with processing a huge amount of data on a very limited number of cores was the starting point and the reason to tackle with the optimization.
After enhancements of a certain part of the code, it was tested on up to 4096 cores, the maximum core number we have been granted access to. The tests showed that the URANIE code is ready to be run on Tier-0 machine.

1. Introduction

Over the last decade, the improvements of computer hardware and software have brought a significant change in the capabilities of simulation software in the field of nuclear applications. New computer power has facilitated the emergence of simulations that are more realistic (complex 3D geometries being treated instead of 2D ones), more complex (multi-physics and multi-scale being taken into account), and more meaningful (with propagation of uncertainties).

* Corresponding author. E-mail address: miroslaw.kupczyk@man.poznan.pl

In order to treat uncertainty analysis in this constantly evolving framework, CEA has developed a software platform named URANIE [1] that provides tools for validation, optimization, uncertainty analysis, and model calibration for high performance computing codes.
The URANIE codes denoted for revision run on tens of cores; however, we expect that it should run on a bigger number of cores due to the communication schema of the application. The plan was to look into the code listings, put tags and special debugging pragmas and analyse the application behaviour using various inputs. We decided not to use the external code analysing tools.

2. Presentation of URANIE

The "Uncertainty and Sensitivity" platform URANIE developed by the CEA aims to regroup methods and algorithms about Uncertainty and Sensitivity (US) and Verification and Validation (VV) analyses in the same framework. URANIE is based on the data analysis framework ROOT [2], (http://root.cern.ch) an object-oriented and petaflopic computing system developed by CERN. This framework offers several useful features as advanced visualization, powerful data storage and access in several formats (binary, SQL, distant access), a C++ interpreter, and so on.
Mathematical models, designated to simulate complex physical processes, are often used in scientific and engineering studies. URANIE is designed to simulate mathematical models of physical systems and is built in a modular way with several libraries (Figure 1) devoted for a particular task in the US&VV analyses. We also wrapped external libraries for special treatments:

- Mixmod (MIXture MODelling) is a library written by the French institute INRIA to estimate the Gaussian mixture parameters through maximum likelihood with the EM (Expectation Maximisation) or SEM (Stochastic Expectation Maximisation) algorithms;
- Opt++ is an object-oriented class library written by the Computer Science and Mathematics Researchers (CSMR) at Sandia National Laboratories for the resolution of nonlinear optimization problems;
- And Club, developed by the French institute CNES, is a library which implements low level C++ utilities input-output services for the files treated by the simulations codes (replace the input parameters values in the input files and recover the output values in the output files after the completion of a “code evaluation”).

In this section, we then describe the libraries of the URANIE platform.

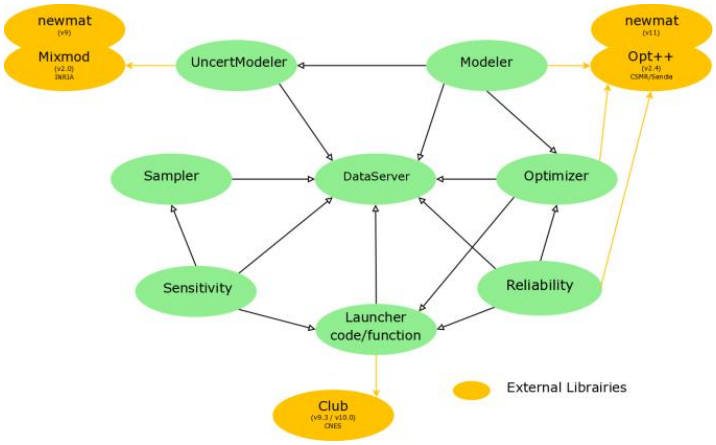

Figure 1 : Functional diagram of URANIE

The main library, the DataServer library, defines the TDataServer object which contains all the information about the uncertain variables for the US&VV analyses. Then, this TDataServer object flows through the other libraries in order to apply the methods of the study.

The second library, the Sampler library, is devoted to generate a design of experiment (deterministic/statistical) from characteristics of the uncertain variables. Several methodologies are implemented:

- qMC ("quasi Monte-Carlo") sequences (Sobol, Halton);
- SRS (“Simple Random Sampling”), LHS (“Latin Hypercube Sampling”), ROA ("Random Orthogonal Array"), Archimedean Copulas;
- MCMC ("Markov Chain Monte-Carlo") method for Gaussian mixture.

The Launcher library is devoted to manage the computation on a desktop (sequential) or on a cluster (distributed). The goal is to construct the Y matrix jointed in the X matrix of the design of experiment. We can launch either the original simulation code, or an analytical function like surrogate models. The surrogate models can be built by the modeller library or written by the user following a prototype.

The Modeler library is devoted to build a surrogate model from input to output attributes contained in a database. The surrogate models implemented in URANIE are polynomial, polynomial Chaos expansion and neural networks. We plan to implement the kriging methodology soon. After building the surrogate model, it can be saved in several languages like C/C++, Fortran, PMML (“Predictive Model Markup Language” is the leading XML standard for save statistical and data mining models and supported by over 20 vendors and organizations) for using it in a US&VV analyses, instead of the computation code, with a very low CPU time computation.

The optimizer library is devoted to perform a Verification and Validation code or to find the optimum of a computation code or analytical function. We can also perform multi-criteria optimization with Genetic Algorithms.

The goal of the UncertModeler library is to examine how well a sample of data agrees with a given distribution as its population with goodness-of-fit techniques.

The Reliability library is devoted to perform a reliability analysis. At present time, this library is not implemented, but we plan to implement the SORM/FORM methodology.

The last library, the Sensitivity library, contains several methods to perform Sensitivity Analysis between the two X and Y matrixes like Regression method (Pearson – "SRC", Spearmann – "SRRC" coefficients), Screening method (Morris) and Sobol indexes (Sobol's methodologies, FAST).

ROOT (and therefore URANIE) has a C++ interpreter which allows us to integrate easily new algorithms or to customize complex treatment with several objects in a function with few parameters. The C++ interpreter is the first level of the user interface. The second level user interface is an XML file; this file contains a description of the uncertain variables and the different steps to apply for US&VV analyses. The XML interface allows us to integrate easily URANIE modules in an industrial platform.
URANIE is based on the realisation of DOEs (Design Of Experiments), sets of simulations in which the same code is executed with slight modifications in the input files and parameters, so that the uncertainty range of the input variables is covered.

3. Description of the project

The key point of URANIE's inner architecture is that a single simulation (job) is indivisible. The number of possible simulations is linearly correlated with the number of used processor cores. In this way increasing the range of input variables causes the number of required hardware resources also needs to be increased. The most used ratio is either 1 or 2 (a single simulation uses 2 cores).

In order to accommodate various middlewares and launch codes as black boxes, the URANIE launcher has the following strategy:

- a single job is allocated,
- the master node runs a control process,
- the control process launches children of the control process, each child running a *mpirun* script which runs one computation in the DOE,
- the control process checks for the state of the children in order to decide when to run new children.

This strategy gives good flexibility and good performance on hundreds of processors, but the bottleneck on the master node can become a problem on large runs. Also, this strategy gives little control on the placement of processes in the context of coupled simulations.

The main goal of the project was to improve URANIE efficiency by minimizing the single simulation execution time for an increased number of DOEs.

Additionally, the goal was also not to re-create a large amount of source code as it was created for a substantial period of time and number of people.

4. Performed work

The main work was carried out on *curie.ccc.cea.fr* (CEA) and local PSNC clusters (CANE) we

had at our disposal. At first we installed URANIE software on CANE to try to reproduce the bottleneck problem with the number of threads but we decided to quit this setup due to a very long time that tasks spent in the queue before launching. After that all further work was carried out on *curie*.

With the system-based strategy we conducted lots of test-runs in which the odd situation was observed. After increasing the number of jobs above the experimental limit (approximately 450) lots of launched child processes started to become zombies, so that the master process which normally waited for their children to complete could wait forever because it would never receive the proper terminate signal from them.

The above was strongly related to the problem tracked in the source code. Originally the software authors did not check the return value and error code given by the *fork()* function which was called by the master process in the loop limited by the number of DOEs. In such cases, when -1 was returned (*fork()* did not succeed) the processes with such IDs were still added to the list of running processes, whereas they should not have. This was, in turn, causing other troubles when calling the *waitpid()* function on running threads invalid PIDs (-1) which were completely changing the logic of this part of the code.

It turned out that spawning children processes above the limit caused *fork()* to return *errno* equal to EAGAIN which literally meant it was not possible to create a new process because the caller's RLIMIT_NPROC resource limit was encountered. In this case it was decided to programmatically call the *setrlimit* function increasing the maximum number of threads (setrlimit(RLIMIT_NPROC, …)) on the one hand and decreasing the maximum size of the process stack setrlimit(RLIMIT_STACK, …) on the other hand. This sometimes helped a bit to increase the initial experimental limit but not much and not in each test-run case.

Finally, to let URANIE run large DOE it was decided to split the total number of launching children processes into groups with a size not exceeding the experimental maximum number. In this way the processes which finished their job were replaced by new ones performing other simulations. This, in turn, enabled the whole configuration (with a large number of threads) to work and not making the big job submitted to queuing system hanged out.

However, the bad impact of launching processes in such a way is that there is always a hard limit of few hundreds of running threads. Up to this limit it can be noticed that an increasing number of jobs (as the number of used threads and cores) does not influence the execution time. Moreover other processes have to wait for their turn causing the total time to draw out. In other words, this strategy shows that its results are strongly dependent on system limits set by a system administrator.

It was also noticed (see table and chart below) that the results have been obtained within 30-minute execution time limit in the wide range of used cores (lower or equal than 2048) which was not observed for the original codes.

After all tests were carried out it seems that some of the primary results provided by the software authors might have been incorrect due to the reasons described above.

Total execution time according to environment configuration (i.e. number of jobs = number of simulations, number of cores etc.) are given below in [Table 1]:

Table 1. Total execution times for different strategies (jobs and cores number etc.

		Serial				
		njobs = ncores			njobs = 2 * ncores	
		1GB mem			1MB mem	
No. of nodes	Ncores	noIO	noIO, emptyMaster	100 IO	noIO	noIO, delay(30)
1	16	00:02:36	00:03:52	00:03:48	00:06:24	00:06:40
2	32	00:02:35	00:03:47	00:03:58	00:06:22	00:06:39
4	64	00:02:38	00:02:37	00:03:57	00:06:28	00:06:32
8	128	00:02:37	00:02:40	00:03:50	00:06:28	00:06:32
16	256	00:02:44	00:02:44	00:03:53	00:06:40	00:06:45
32	512	00:02:53	00:02:54	00:04:18	00:07:15	00:07:07
64	1024	00:05:05	00:04:31	00:05:48	00:09:38	00:09:29
128	2048	00:09:06	00:09:02	00:10:52	00:16:47	00:16:24
256	4096	00:16:57	00:17:06	00:21:16	00:35:21	00:32:26

The explanation of the [Table 1] header:

Different versions vary in terms of:

- memory allocation per execution,
- I/O rate (no I/O, 100 write executions of the total memory),
- used strategy (temporization or emptyMaster),
- total number of jobs (equal to the number of cores or twice the number of cores).

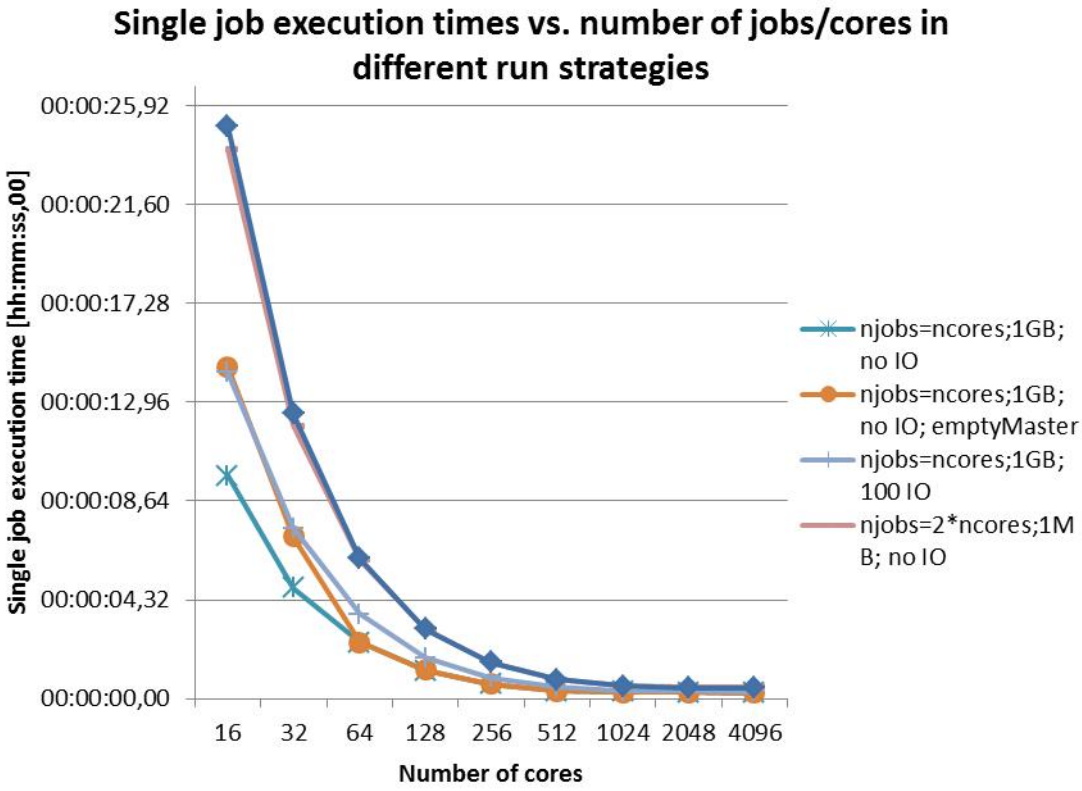

Figure 2 : Single job execution times in different run strategies

The timing characteristics are presented in [Figure 2]. It is difficult to calculate the speedup for given configurations because, as it was written above, the number of simulations is strictly related to the number of used processor cores (and also threads). The experimental runs showed the weak scalability of the improved codes. For a single simulation only a single processor would be used. However, we believe this will not make any sense.

Due to URANIE's internal architecture it was decided to normalize the above times to receive a single job execution (for different setups) time. Afterwards it can be noticed that the increasing number of simulations (jobs) along with the number of cores make the single job execution time shorter.

References

[1] Uncertainty and sensitivity analysis of the nuclear fuel thermal behaviour, Nuclear Engineering and Design 253 (2012) 200– 210.
[2] Data Analysis Framework ROOT; http://root.cern.ch

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-312763. The work is achieved using the PRACE Research Infrastructure resources: Tier-0 CURIE (CEA – France) and Tier-1 CANE (PSNC – Poland). PRACE Preparatory Phase Project, grant No. 1527.