H2020-FETHPC-3-2017 - Exascale HPC ecosystem development

EXDCI-2

European eXtreme Data and Computing Initiative - 2

Grant Agreement Number: 800957

D4.5

Assessment for legacy code and software modernisation

Final

Version: 1.0

Author(s): G. Colin de Verdière, CEA

Date: 17/02/2020

Project and Deliverable Information Sheet

<table>
<tr><td rowspan="7">EXDCI Project</td><td colspan="2">Project Ref. №: FETHPC</td></tr>
<tr><td colspan="2">Project Title: European eXtreme Data and Computing Initiative - 2</td></tr>
<tr><td colspan="2">Project Web Site: http://www.exdci.eu</td></tr>
<tr><td colspan="2">Deliverable ID: D4.5</td></tr>
<tr><td colspan="2">Deliverable Nature: Report</td></tr>
<tr><td rowspan="2">Dissemination Level:
PU</td><td>Contractual Date of Delivery:
29/02/2020</td></tr>
<tr><td>Actual Date of Delivery:
17/02/2020</td></tr>
<tr><td colspan="2">EC Project Officer: Evangelia Markidou</td></tr>
</table>

* - The dissemination level are indicated as follows: **PU** – Public, **CO** – Confidential, only for members of the consortium (including the Commission Services) **CL** – Classified, as referred to in Commission Decision 2991/844/EC.

Document Control Sheet

<table>
<tr><td rowspan="6">Document</td><td colspan="2">Title: Assessment for legacy code and software modernisation</td></tr>
<tr><td colspan="2">ID: D4.5</td></tr>
<tr><td>Version: 1.0</td><td>Status: Final</td></tr>
<tr><td colspan="2">Available at: http://www.exdci.eu</td></tr>
<tr><td colspan="2">Software Tool: Microsoft Word 2013</td></tr>
<tr><td colspan="2">File(s): EXDCI-2-D4.5-V1.0.docx</td></tr>
<tr><td rowspan="4">Authorship</td><td>Written by:</td><td>G. Colin de Verdière, CEA</td></tr>
<tr><td>Contributors:</td><td>Maike Gilliot, ETP4HPC Office
François Bodin, IRISA</td></tr>
<tr><td>Reviewed by:</td><td>Paul Carpenter, BSC
Serge Bogaerts PRACE</td></tr>
<tr><td>Approved by:</td><td>MB/TB</td></tr>
</table>

Document Status Sheet

Version	Date	Status	Comments
0.1	24/01/2020	Draft	Initial version
0.2	28/01/2020	Draft	Revision of Maike Gilliot and Jean-Philippe Nomine
0.3	05/02/2020	Draft	After revision of Francois Bodin, this version will be submitted for internal review
1.0	17/02/2020	Final version	Reviews taken into account

Document Keywords

Keywords:	EXDCI-2, Legacy code, SLOC, FORTRAN, C++ OpenMP, MPI, Cuda, Viscosity

Table of Contents

List of Tables

References and Applicable Documents

[1] http://www.exdci.eu

[2] http://www.prace-ri.eu

[3] http://www.etp4hpc.eu

[4] https://en.wikipedia.org/wiki/Programming_complexity

[5] S. Wienke, J. Miller, M. Schulz, and M. S. Muller, “Development effort estimation in HPC,” in High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for. IEEE, 2016, pp. 107–118.

[6] B. Boehm, C. Abts, and S. Chulani, “Software development cost estimation approaches — a survey,” Annals of Software Engineering, vol. 10, no. 1, pp. 177–205, 2000.

[7] Nguyen, V., Deeds-Rubin, S., Tan, T., & Böhm, B. (2007). A SLOC Counting Standard.

[8] https://www.archive.sercuarc.org/wp-content/uploads/2014/05/Software-Cost-Estimation-Metrics-Manual-for-Defense-Systems.pdf

[9] Ramin Moazeni, Daniel Link, and Barry Boehm. 2013. Incremental development productivity decline. In Proceedings of the 9th International Conference on Predictive Models in Software Engineering (PROMISE '13). ACM, New York, NY, USA, Article 7, 9 pages. DOI=http://dx.doi.org/10.1145/2499393.2499403

[10] https://www.linuxfoundation.org/events/2008/10/estimating-the-total-cost-of-a-linux-distribution/

[11] https://stripe.com/files/reports/the-developer-coefficient.pdf

[12] https://en.wikipedia.org/wiki/Technical_debt

[13] https://en.wikipedia.org/wiki/COCOMO

[14] http://www.r-ccs.riken.jp/aicssite/wp-content/uploads/2017/05/Analysis-of-the-Characteristics-and-Development-Trends.pdf

List of Acronyms and Abbreviations

AISBL	Association Internationale Sans But Lucratif (International Non-for-Profit Association)
CoE	Centres of Excellence for Computing Applications
CN	Code Number (in the tables of this deliverable)
D	Deliverable
EC	European Commission
ESLOC	Effective Source Line Of Code
EU	European Union
FET	Future and Emerging Technologies
H2020	Horizon 2020 – The EC Research and Innovation Programme in Europe
HPC	High Performance Computing
ISV	Independent Software Vendor
IT	Information Technology
KPI	Key-Performance Indicator
M	Month
OS	Operating System
PM	Person Month
PTC	PRACE Training Centre
Q	Quarter
R&D	Research and Development
R&I	Research and Innovation
RFP	Request for Proposal
ROI	Return On Investment
SLOC	Source Line Of Code
SRA	Strategic Research Agenda
SWOT	Strengths, Weaknesses, Opportunities and Trends
TRL	Technology Readiness Level
US	United States
WG	Working Group
WP	Work Package

Executive Summary

Many of today's applications running on large HPC systems are codes, that have been in development for more than two (or even three) decades, programmed partially in "old-fashioned" languages, such as Fortran, and which were not designed for scale and to parallelise for system architectures as we know them today in HPC. We refer to these codes as "legacy codes".

In the advent of a new generation of HPC systems, the question is whether these codes are still relevant to their owners, and if yes, how these codes can be rewritten to run efficiently on the future HPC systems, and what effort this would require. This deliverable casts a light on the European portfolio of legacy codes through the analysis of a survey that has been distributed amongst European code developers.

Between November 2019 and January 2020, the survey was distributed to many contacts in industry and in academia, leading to 19 answers. Although the answers are too few for a statistically sound analysis, some interesting lessons can be derived from the input. We will put in perspective a number of lessons learned from experience with respect to the picture and feedback from the ecosystem.

We first explain the rationale behind this study. Then, we introduce a new metric called the *viscosity* of codes before explaining in more detail the content of the survey.

The analysis of the surveys shows that:

- The European code base represents a significant amount of money that cannot or should not be wasted.
- The team sizes are too often too small to envision a full rewrite in a timely manner.
- The codes are critical to their owners and, therefore, can't be discarded through obsolescence due to a change of supercomputers.
- Rewriting a code from scratch takes many (> 2) years as soon as the code has a significant size.
- Still some, if not all, codes rely on the availability of FORTRAN (FORTRAN90). This raises the concern with respect to teaching, as FORTRAN is no more studied during the academic syllabus.
- Whereas some teams are eager to start their porting effort very soon, others plan to wait at least 5 years, if not more, before starting the work. Given the time needed for rewriting, one can expect these codes only to be ready for exascale in 10 to 15 years from now.

Conclusions of these findings are that some funding needs to be dedicated to code porting by teams in addition to the funds given to the Centres of Excellence (CoE). Given the size of the codes and their specificities, it is difficult to imagine that the CoEs will do all the jobs; hence the need to also directly fund code teams to do the ports.

Moreover, in the light of this analysis, the importance of the role of PTCs and CoEs is however confirmed, for the success of porting (open source) codes to the future machines.

And, most importantly, the proposed viscosity formula is a crude yet effective way to get a feel of the investments needed in the future to preserve the European portfolio of codes.

1 Introduction

In the race to the Exascale, most of the efforts (in terms of R&D and in terms of monetary effort) have been concentrated on hardware. It is not surprising, given the challenge of reaching 1 EFlop/s in less than 30 MW.

As of this writing, no unique solution for future system architecture has emerged as the silver bullet. Yet a general agreement concludes that exascale machines will exhibit some sort of acceleration to reach the proper level of GFlops/W.

It means that we will witness a paradigm shift in terms of programming those machines, moving away from CPU only[1] towards CPU "x" (you name it) plus potentially a multitude of different accelerators.

Be it a new hardware architecture or programming language, the adoption rate is always (way) below the expectations of the most enthusiastic early adopters or vendors, despite the expected benefits: One can wonder why GPU usage is not as widespread in Europe as envisioned with its potential performance boost. The same can be said about PGAS which, in theory, should simplify code development and in practice very few production codes rely on PGAS. This frustrating situation leads to a widening gap between promising prototypes and production systems.

As a consequence, the main impact of exascale machines will be on software. The purpose of this deliverable is to focus on this impact and try to quantify the associated costs of migration of application codes. For doing so, we introduce the concept of "viscosity", a simple, but yet efficient metric for assessing the expected migration effort for a given code.

In the following chapter, we will introduce the notion of code viscosity first. In chapter 3, we will proceed with the analysis of the answers to our survey. The last chapter will conclude this study. The survey itself is in annex 1 and the raw values of the survey are available in annex 2.

2 Concept of code Viscosity

2.1 Applications evolutions

In this strategic landscape, when speaking of applications, the focus is on programming models and tools[2]. The different roadmaps try to understand what standards to promote and what kind of tools to develop to help the developers in their daily tasks. But the cost of developing or porting codes to new machines is seldom studied nor included as a specific item in European funding. The main reason is that those costs are viewed as being part of everyday developer's activities. Therefore, it is considered as "already accounted for". This view comes from the fact that code changes are most of the time incremental and part of some research. The only exceptions to this incremental process were the migration from SMP machines to cluster of SMPs in the '90s and the port to GPUs, for the codes which actually did it, in the '10s.

If some codes do not really need to evolve for the coming years, most will be under the pressure of evolving user needs. Simulations are more and more precise both in terms of

[1] Despite the fact that the Top500 is dominated by accelerated machines, the bulk of production on European supercomputers is done on homogeneous x86 systems, even if some codes have been ported to GPUs.
[2] See the SRA for example http://www.etp4hpc.eu/sra.html

physical models (introducing more complexity in the programs) or in representation (order of magnitude more elements to process). Datasets are getting always larger due to the increasing number of possible inputs (new sensors, new feeds, finer simulations...). Classical HPC codes will merge with AI to provide extended simulation tools. Workflows are getting more complex. Thus changes will be forced on codes. This observation is valid for codes coming from ISV or industrials as well as from an academic origin.

2.2 A simple metric for code assessment: Viscosity

With the announced arrival of European exascale machines around 2022+, the question of the costs and efforts[3] of porting the codes arises and must be addressed at the European level. What would be the consequences of underestimating this question? Updating large codes takes time: it will be a multi-year effort for the largest ones. Underestimating those efforts will have multiple consequences: important delays to achieve the porting, loss of leadership or attractiveness of European research teams, economic losses for European industrials in an ever more competitive world, to name a few.

In order to get a clearer view of the situation for HPC application codes, we propose to introduce a metric called "the code viscosity" that will capture the ability of a code to be ported to a new environment, or its reluctance for it, by analogy to the viscosity of a fluid that quantifies its resistance to movement.

The viscosity of a code is a consequence of the IDPD (Incremental Development Productivity Decline) that has been studied by Moazeni in [8].

The more viscous a code, the more expensive it will be to do the port, both in terms of time to completion and in the manpower needed, the less it is prone to the adoption of newer technologies. It is a crude approximation yet it should be sufficient to exhibit trends and help to determine extra migration costs. We can list some of those:

- Additional resources are needed to accompany the scientist who can't absorb the additional load due to the porting;
- Complexity of software is the root cause of the difficulty of migration. Studies are needed to help developers to master this complexity. A possible path is to develop the usage of DSLs. The goal here is to reduce the complexity as much as possible;
- Most scientists developing codes are not computer scientists (CS). Training scientists to the intimacy of future hardware won't work or divert them from their primary mission. Future development teams should be a mix of scientists of a given subject (physics, chemistry, climate, medicine...) and computer scientists;
- Many codes will require a change of software architecture (and possibly a change of programming language) to cope with the new hardware offerings. While being a pure CS job, it will be the opportunity to make the code rely on highly optimized libraries provided by the vendors;
- Verification and validation of the codes take lots of time, especially if the number of options is large. Depending on the test suites, this period can span a full year, incurring associated costs (manpower, computer access time...).

[3] Costs (machines, software, compute time…) and efforts (team size) are two linked notions. A large amount of money for a small team won't solve the problem of code porting. The critical size for a development team is a subject for further studies since it has not been touched by this study.

2.3 Proposed metric

As said above, the viscosity will give some indications on the migration cost of codes. The more viscous a code, the more expensive it will be to do the port, both in terms of time to completion and of manpower needed, the less it is prone to the adoption of newer technologies. In other words the more viscous the higher is the technical debt [12] of the code that will have to be paid sometime in the future.

By no means, it is a measure of people willingness to advance their codes but rather a way to show that code evolutions need to be taken into account in parallel or (better) before any system change is envisioned and provisioned accordingly.

By purpose, the parameters chosen for assessing viscosity can be determined easily by a software development team, and no specific tool nor any highly sophisticated (and time consuming) analysis is required. Again, our goal is to get a broad picture of the application code landscape by gathering this data from a wide range of industrial and academic application codes. This metric is simpler than the methodology proposed by Wienke [5] since it doesn't require access to the source code by researchers to run various tools.

$$V = \frac{A * L}{TC * P}$$

The different parameters are:

- **A** = age of the code in years
- **L** = length of the source code, in millions of lines, **excluding** third party libraries such as BLAS, FFTW.
- **TC** = team capability/confidence to develop accelerated codes
- **P** = percentage of acceleration (1 = code fully ported to accelerators, 0 = not ported)

We select the aforementioned parameters for the following reasons:

A: As a code ages, without a constant rewrite that almost nobody could afford, its implementation doesn't benefit from the advances in programming languages (new standards), new hardware or in development practices (new algorithms, new data representation, increased parallelism...). The age parameter takes also into account the change of programmers across time that can occur, incurring a natural loss of intimate knowledge of the source code. The older the code is, the higher probability that the original authors are not around anymore and that nobody really masters the innards of the source code. Therefore making a technology update is seen as a major risk, once again slowing down the evolution process and thus undertaken only under high pressure. This aging process contributes to the cost of updating a code to the latest technologies (hardware and/or software) – see again [8].

TC: **T**eam **C**apability/confidence to migrate the code to accelerated/exascale systems (0 cannot, 1 have the expertise and team to perform the jobs).

If the previous parameters are structural to any code development, the last parameters try to capture the complexity of a code from a technical point of view.

L: The complexity of a code is an obvious contributor to the viscosity of a code yet not the only factor. A number of metrics have been proposed to characterize code complexity ([4], [6] or [7] for a survey of development costs or [13] for a model of code complexity). In many cases, the values of those metrics are unknown to the development teams. To compare different codes (and especially commercial codes that can't disclose too many features of their developments for obvious reasons) we would have to agree on a set of metrics that would be meaningful in every cases, such as for FORTRAN and C++ codes which exhibit quite different sets of complexities by nature. The easiest, yet crudest, metric we propose here is to

simply count the number of source line of the code (SLOC). This is the easiest value to get in all cases since it requires no (possibly expensive) tool[4]. We think that this parameter is nonetheless relevant since the larger the code is the more difficult it becomes to change: the number of possible failures due to source modifications increases dramatically with its size as well as the time to run validation tests. These difficulties prevent many teams to move forward by big steps.

Based on our experience a normal developer really masters more or less 100 000 lines of code in a single program. By this we mean that the purpose of the code is fully known to the developer, the algorithms are clear in his/her mind and can be explained to others. It also implies that the data structures and the parallelism (intra and inter node) are well understood so that a complete rewrite to adapt the said code to a different platform or language will not take years. We probably should introduce this factor to link the team size to the code size (parameter **L**) and take productivity into account.

A second observation from the field shows that 80 % of the SLOC of a code don't account for the actual computation time. It means that the difficulty of porting a code is concentrated on the 20 remaining percent.

For a first version of the metric, we choose to ignore the two previous remarks for the sake of the simplicity of the measurements done by the different teams and rely solely on the **L** parameter.

P: To refine the **L** parameter, we propose to take into account the amount of porting towards accelerators already done. Obviously, a 1M SLOC code already running fine on an accelerated petascale machine will require far less efforts to run on an exascale system than a code which hasn't done the adaptation. Therefore, we introduce the **P** factor that ranges from 0 to 1 (0 meaning unfit for accelerators, 1 means fully using accelerators) to capture this investment in adapting the code to multiple parallelism paradigms (distributed memory + accelerators) already done. The **P** parameter implies that the architecture of a true exasflop computer will be based on some sort of accelerator (GPUs are the most probable option). It means that if **P**=0 then $V=\infty$ or, in other words, that moving the code to an accelerated exaflopic system will be a major costly challenge due to a huge technical debt.

[4] `wc -l *.[cCf]` and its variations, would often do the trick.

3 Survey analysis

3.1 Survey collection

The detailed version of the survey can be found in Annex 2: Survey content. Between November 2019 and January 2020, the survey was distributed to many contacts in industry and in academia via the Centers of Excellence (CoE) as well as ETP4HPC's network. We got **19 answers** to our requests. We are very grateful to those who took the time to answer our request for input. 19 is too small a number to really have a statistically sound database. Nonetheless some important lessons can be derived from what was provided to us.

In this document, in order to keep the anonymity of the codes and the development teams, we transformed the code names to serial numbers and we removed any information related to the owners (cf. Annex 1: Raw values of survey for full details of the survey outcome).

3.2 Analysis

3.2.1 Assumptions

For the analysis we make a number of assumptions:

1. The productivity of a developer has been evaluated to 300 *effective source lines of code* (ESLOC) per month [8], where *effective* means fully tested, debugged, commented and documented which is a lengthy and tedious process. From the previous entry we estimate the cost of an ESLOC to be 10€, assuming that a developer is paid 3000€ per month. Your mileage may vary here but 10€ is easy to manipulate;
2. A full rewrite of a code will lead to a contraction (hopefully) of the number of ESLOC. A conservative factor of 2 is taken here, assuming that the rewrite is done at iso-functionalities. This also takes into account that part of the knowledge has not to be re-discovered, with the assumption that the numerical methods do not need to be changed due to the new possibilities offered by an Exascale machine[5];
3. We assume that the team size won't change in the future;
4. For question on the expected duration for rewriting the code (question 15), when the answer was a range, we took a mid-point for our evaluations. The raw values used are listed in Table 13 of the Annex 1: Raw values of survey;
5. A developer typically masters approximately 100 000 lines of code in a single program.

3.2.2 Viscosities

From the surveys we can first compute the viscosities associated to each code. Table 1 summarizes the values. We can observe that we have 3 classes:

1. 9 codes have a viscosity below 20
2. 5 are between 20 and 1000
3. And 5 are above 1000

[5] In some cases, this last point will be proven plainly wrong and may a) increase the total code size b) slow down the development process since new methods require a lot of validations.

If we focus on codes that are planned to be ported to Exascale systems, codes 3, 13 and 14 (highlighted in grey) are the most viscous because they are old (13, 26 and 42 years respectively), fairly large (more than 0,5 Million SLOC) and the confidence that the team has the required expertise is only average. Code 18 benefits from a larger team, ready for the coming challenges, even if the code is large (2 M SLOC) and old (25 years). This pinpoints the need to work on the skills of the teams and the need to have regular training sessions to develop those skills such as the ones provided by the PTCs, given the fast evolution of technology.

CN	1	3	4	5	7	9	10	11	12	13	14	15	16	18	2	6	8	17	19
Viscosity	4	39000	0.83	18.57	17.97	0.13	86.25	156	1.70	2600	7000	1.40	134.40	50.00	1.20	0.25	513000	160	1500

Table 1: Computed viscosities for the different codes.
The yellow cells indicate codes that won't be ported to Exascale systems.

3.2.3 Total cost of the codes

Using assumption 2, we can evaluate the implicit cost for developing these codes over the years. Most of the time, those costs are implicit and included as part of the manpower costs. Yet codes are part of the immaterial capital of the owners and it was important, as part of this deliverable to evaluate this capital, at least through the answer we had. As a consequence, the real capital is far more important, thus stressing the importance of legacy codes. Table 2 shows that most of them represent a significant amount of money. We highlighted in grey values above a million €. Only 5 out of 19 are "worth" less than a million €.

CN	1	3	4	5	7	9	10	11	12	13	14	15	16	18	2	6	8	17	19
Cost	5 000	15 000	500	1 000	5 000	50	6 000	3 000	1 700	5 000	10 000	100	12 000	20 000	10	250	19 000	1 000	10 000

Table 2: Estimated cost (in k€) of the code according to our assumption.

The authors wish they had had access to more surveys to demonstrate the European software capital. This is especially true with large community (open source) codes, where the costs are diluted amongst many institutions.

3.2.4 Importance of the codes

We asked the owners of codes what was their feelings about the importance of their codes for the activity of their organisation 14 out of 15 declare that they value it at 4 or more on a scale of 5 as can be seen in Table 3. It means that these codes will have to be ported to new generations of computer, one way or the other.

CN	1	3	4	5	7	9	10	11	12	13	14	15	16	18	2	6	8	17	19
Imp - 9	5	4	3	5	5	5	4,5	4	3	5	3	2	5	5	2	5	5	4	5

Table 3: Importance of the codes according to their owners.

This table in itself justifies this study on legacy codes.

3.2.5 Code sizes

To get a feel of the codes base, we asked the owners the size of their codes **excluding** third parties libraries. Table 4 summarizes the values with a highlight on 1 million and above.

CN	1	3	4	5	7	9	10	11	12	13	14	15	16	18	2	6	8	17	19
SLOC	0,50	1,50	0,05	0,10	0,50	0,01	0,60	0,30	0,17	0,50	1,00	0,01	1,20	2,00	0,00	0,03	1,90	0,10	1,00

Table 4: Code sizes expressed in million lines of source (SLOC).

Excluding the libraries is important because the given SLOC is the real added value poured in the code by the team. It is in those lines that the knowledge of the group has been captured. From the table, we notice that 10 out of 19 codes are at least half a million lines long, not counting the associated documentation. This also explains the values presented in section 3.2.4: the larger the code, the more important it is perceived by the developers.

3.2.6 Team sizes

Table 5 lists the current size of the teams, with an average team size of 10.2. Only 5 of these are above 10 people with two exceptional ones (30 and 35 people).

CN	1	3	4	5	7	9	10	11	12	13	14	15	16	18	2	6	8	17	19
Cur team size	12	15	10	35	30	4	5	4	4	10	10	5	8	16	2	8	10	4	2

Table 5: Current team size

The consequence is that the developers are in charge of quite a large number of SLOC (81609 on average, as computed from Table 4). Table 6 has been computed by dividing the number of SLOC by the current team size. Therefore it depicts the number of lines that they have to maintain on average per person.

CN	1	3	4	5	7	9	10	11	12	13	14	15	16	18	2	6	8	17	19
SLOC/dev	41 667	100 000	5 000	2 857	16 667	1250	120 000	75 000	42 500	50 000	100 000	2 000	150 000	125 000	500	3 125	190 000	25 000	500 000

Table 6: Number of SLOC per developer for each code.

As said above, from our experience, a developer has an intimate knowledge of 10 000 SLOC and is able to cope with 100 000 SLOC. Above this last value, knowledge of the idiosyncrasies of the code is lost to the developer, especially in case of a massive rewrite.

The table shows also that the current (maintenance) teams are too small to face a massive change in their codes, at least for codes 3, 10, 14, 16, 18, 8, 19; where the SLOC/developers varies from 41 000 up to 500 000.

3.2.7 Age

Table 7 lists the different ages of the codes. 10 out of 19 are older than 10 years with one very old (42 years).

CN	1	3	4	5	7	9	10	11	12	13	14	15	16	18	2	6	8	17	19
Age	8	13	7	13	23	3	23	26	8	26	42	7	14	25	6	4	27	4	3

Table 7: Age of the codes covered by the survey, in years.

Table 8 shows that some, if not all, codes rely on the availability of FORTRAN (FORTRAN90), noted Ftn for short in the table. The only noteworthy exception is code 13 which is old (26 years) but mostly C++ with a small amount of FORTRAN, indicating that this code has seen a significant amount of rewrite along the years. This raises the concern

about teaching FORTRAN to young scientists since FORTRAN is no more studied during the academic curriculum.

CN	1	3	4	5	7	9	10	11	12	13	14	15	16	18	2	6	8	17	19
Age	8	13	7	13	23	3	23	26	8	26	42	7	14	25	6	4	27	4	3
Languages	**Ftn**/C++/cuda/ Mpi	**Ftn**/C++/MPI/ OenMP	**Ftn**/MPI/ OpenMP	**Ftn**/MPI/ OpenMP/Cuda	**Ftn**/C/MPI/ OpenACC	Python	C/Ftn/MPI/ OpenMP	C++/MPI/ Python	C++	C++/Ɛ Ftn	**Ftn**	**Ftn**	C/C++/MPI/ OpenMP	C/C++, CUDA, OpenCL, MPI,	script	python	C++	javascript/ python	**Ftn**/MPI/ OpenMP

Table 8: Age in years of the different codes and the programming languages used.

Not surprisingly most of the codes rely on the classical list of programming languages: FORTRAN, C, C++ with parallelism being done with MPI and OpenMP.

We can also notice that most of the codes have already adopted the MPI+X programming model, X being either OpenMP for in node parallelism or OpenACC/CUDA for acceleration. This is good news in the perspective of future accelerated machines.

3.2.8 Rewrite capabilities

As mentioned in section 3.2.2, 5 out of the 19 codes are – according to the code owners – not planned to be ported to exascale architectures (codes 2, 6, 8, 17 and 19).

Table 9 gives the time in years needed to rewrite the codes for a new architecture, as seen by the code teams. 10 out 18 will need 2 or more years of rewrite. The only outlier is where 100 years of rewrite is quoted, for which we have a doubt (is 100 years realistic?). Code 19 was not able to provide an estimation since no porting to a new architecture is planned.

CN	1	3	4	5	7	9	10	11	12	13	14	15	16	18	2	6	8	17	19
Rewrite - 18	1.50	10.00	2.00	7.00	0.75	0.75	2.00	1.00	10.00	3.00	5.00	1.00	2.00	1.00	2.50	1.00	100.00	1.50	??

Table 9: Time for complete (in years) of the code estimated by the teams.

The evaluation of the time to rewrite the code from scratch is a difficult exercise. Based on our assumptions, listed in section 3.2.1, we computed the time of rewrite using the size of the code (SLOC), the average productivity of a programmer (300 SLOC/month), the team size (TS), and the assumption that the rewritten code is only half the size of the original code:

$$Comp\ (years) = \frac{SLOC \times 10^6}{2 \times 300 \times 12 \times\ TS}$$

In Table 10, we compare the estimated duration for the rewrite as defined in the formula above with the team's estimated duration for the rewrite

CN	1	3	4	5	7	9	10	11	12	13	14	15	16	18	2	6	8	17
Comp. Rewrite	6	14	1	0	2	0	17	10	6	7	14	0	21	17	0	0	26	3
Difference Comp – est.	4	4	-1	-7	2	-1	15	9	-4	4	9	-1	19	16	-2	-1	-74	2

Table 10: Computed time to rewrite and the delta with the estimation[6]

For small codes, this formula looks too conservative. On the other hand for large codes, it says that the code team may be a bit optimistic. The computed results should probably not be taken at face value but rather as an indication that there is a risk of underestimating the cost, time and team size to rewrite a code from scratch.

It is also interesting to analyse the foreseen date of porting to Exascale systems (Table 11). There are clearly two classes of teams: the one of people eager to do the port now (now being in less than 2 years) and the one that will wait at least 5 years, if not more. A further study should investigate the reasons behind this difference of timing.

CN	1	3	4	5	7	9	10	11	12	13	14	15
When	ASAP	5 to 10	5	5 to 10	now	2	~2	?	2	5 to 10	10	Unknown

Table 11: When will your code be ported to Exascale machines?

3.2.9 Pertinence of the viscosity criterion

Is our viscosity formula valid?

Table 12 shows that for viscosities > 20 the rewrite time may be underestimated by the team (positive values of the last row). There is a strong correlation between the view given by the viscosity and the underestimated time of rewrite (under our assumptions); as soon as viscosities are above 20, the delta is positive.

Visc	4	39000	1	19	18	0	86	156	2	2600	7000	1	134	50	1	0	513000	160
Rew	2	10	2	7	1	1	2	1	10	3	5	1	2	1	3	1	100	2
Comp – estim.	4	4	-1	-7	2	-1	15	9	-4	4	9	-1	19	16	-2	-1	-74	2

Table 12: Pertinence of the viscosity criterion

A future study should keep track of the effective rewrite time to validate our model, should such a rewrite occur (at least code 14 is planning to).

[6] Values have been rounded to the closest integer (0 means less than half a year)

4 Conclusion

The real benefit of the exascale machines will depend on the novel results for industry and academia gained by their specific applications exploiting these exascale machines. In this context, the application codes are a determining element for Europe's exascale strategy.

We conducted a survey to gain some insight on the European application landscape: What do those codes look like, what are they made of, and how difficult it will be to prepare theses codes for the next generation of HPC systems. The proposed viscosity formula is a crude yet effective way to get a feel of the investments needed in the future to preserve the European portfolio of codes.

The results of this analysis brought to light a number of additional questions that could answered by further studies and hopefully on a larger base of codes. Yet, the analysis of the surveys shows that:

- The codes are important to their owners and, therefore, can't be discarded through obsolescence due to a change of supercomputers;
- This code base represents also a significant amount of money;
- The team sizes are too often too small to envision a full rewrite in a timely manner;
- Rewriting a code from scratch takes many (> 2) years as soon as the code has a significant size.

The remarks made in this document show the importance of the tools such as the PTC and CoEs for the success of porting (open source) codes to the future machines.

To prepare Europe for exascale means also to prepare its applications for Exascale and not only the hardware. The effort necessary must be financially supported in the upcoming 10 years. This requires funding beyond the current efforts in the CoEs[7].
Other ways and means to support the application porting should be discussed with the developer communities and supported at European level to make the future exascale machines a success for Europe.

[7] Given the size of the codes and their specificities, it is difficult to imagine that the CoE will do all the jobs, hence the need to fund also the team to do the port by themselves.

5 Annex 1: Raw values of survey

This is the raw values of the received surveys used for computations. Rows labels refer to the survey questions of Table 14 by their numbers and we used a mnemonic for concision sake.

CN - 1	1	3	4	5	7	9	10	11	12	13	14	15	16	18	2	6	8	17	19
CTSO - 6	2	15	4	2	3	1	7	4	4	4	2	~5	6	3	54	4	8	1	30
CCTS - 7	12	15	10	35	30	4	5	4	4	10	10	5	8	16	2	8	10	4	2
Org - 8	2012	2007	2013	2007	1997	2017	1997	1994	2012	1994	1978	2013	2006	1995	2014	2016	1993	2016	2017
Age	8,00	13,00	7,00	13,00	23,00	3,00	23,00	26,00	8,00	26,00	42,00	7,00	14,00	25,00	6,00	4,00	27,00	4,00	3,00
Imp - 9	5	4	3	5	5	5	4,50	4	3	5	3	2	5	5	2	5	5	4	5
LOC -10	0,50	1,50	0,05	0,10	0,50	0,01	0,60	0,30	0,17	0,50	1,00	0,01	1,20	2,00	0,00	0,03	1,90	0,10	1,00
Acc - 13	1,00	0,00	0,60	0,35	0,80	0,30	0,40	0,10	0,80	0,01	0,01	0,10	0,50	1,00	0,50	0,50	0,01	0,01	0,20
Exa - 14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
When - 15	ASAP	5 to 10	5	5 to 10	now	2	~2years	?	2	5 to 10 years	10 years	Unknown	0	0		2			
Expert - 16	1,00	0,50	0,70	0,20	0,80	0,40	0,40	0,50	1,00	0,50	0,60	0,50	0,25	1,00	0,01	0,80	0,01	0,25	0,01
Rewrite - 18	1,50	10,00	2,00	7,00	0,75	0,75	2,00	1,00	10,00	3,00	5,00	1,00	2,00	1,00	2,50	1,00	100,00	1,50	??

Table 13: Raw values of the collected surveys.

To fit the table in the page, we used mnemonics that refer to the entries of the survey. The meaning of the mnemonics is: **CN** code number, **CTSO** code team size at origin, **CCTS** current code team size, **Org** start of code development, **Imp** importance of the code, **LOC** lines of code, **Acc** expertise of the team on accelerators, **Exa** the code will be ported to Exascale machines, **Expert** level of expertise of the team.

6 Annex 2: Survey content

The text of the survey sent to developers is in Table 14 below.

Question	Your answer	Comment		
1	Code Name:			
2	Owner organization:			
3	Field & method (short summary):		Will be useful to sort results by types of codes, as a secondary outcome of the survey	
4	Development method:	company/lab owned community development		☐
5	Code status:	Open source Licensed		☐
6	Code team size at origin:			
7	Current code team size:			
8	Development started in:		date	
9	How important is this application for your organization / for your work?		(1=useful to 5=highly critical)	
10	Current number of line of codes (excluding 3rd party libraries):		in (measured) millions of lines	
11	Language(s):		•	FORTRAN C C++ MPI OpenMP CUDA OpenACC Python, …
12	Major 3rd party libraries:		Will be useful as a secondary outcome of the survey	
13	Acceleration:		Quantify the	

Question	Your answer	Comment		
			adaptation of the code to accelerated petascale systems: range [0..1], 0 = unfit to accelerated computing, 1 = code running ne on an accelerated petascale machine, e.g. 0.5 = job half way done	
14	Are you planning to port your codes on an exascale system when available?		yes / no	
15	If yes in how many years?			
16	Do you think your team has the expertise to do a port on an accelerated exascale machine		Range [0..1], 0 = cannot, 1 have the expertise and team to perform the jobs, in-between building expertise and/or team to do so.	
17	Can you describe the reasons for a move to exascale class systems		•	more data analytics more compute capabilities (finer grain model, more elaborate models) code coupling new scientific instrument etc
18	Estimated time of rewrite for a radical change of computer architecture:			

Table 14: the questions asked in the survey