International Journal of Software Engineering and Knowledge Engineering

On the Effect of Exploiting GPUs for a More Eco-sustainable Lease of Life

Giuseppe Scanniello, Ugo Erra, Giuseppe Caggianese
Università degli Studi della Basilicata
Dipartimento di Matematica, Informatica ed Economia
Potenza, 85100, Italy
giuseppe.scanniello@unibas.it, ugo.erra@unibas.it, giuseppe.caggianese@unibas.it
http://www2.unibas.it/gscanniello,www.unibas.it/erra

Camine Gravino
Università degli Studi di Salerno
Dipartimento di Studi e Ricerche Aziendali
Fisciano(SA), 84084 , Italy
gravino@unisa.it
http://http : //www.unisa.it/docenti/gravino/index

Received (12 Nov 2013)
Revised (20 Feb 2014)
Accepted (15 Jul 2014)

It has been estimated that about 2% of global carbon dioxide emissions can be attributed to IT systems. Green (or sustainable) computing refers to supporting business critical computing needs with the least possible amount of power. This phenomenon changes the priorities in the design of new software systems and in the way companies handle existing ones. In this paper, we present the results of a research project aimed to develop a migration strategy to give an existing software system a new and more eco-sustainable lease of life. We applied a strategy for migrating a subject system that performs intensive and massive computation to a target architecture based on a Graphics Processing Unit (GPU). We validated our solution on a system for path finding robot simulations. An analysis on execution time and energy consumption indicated that: *(i)* the execution time of the migrated system is less than the execution time of the original system; and *(ii)* the migrated system reduces energy waste, so suggesting that it is more eco-sustainable than its original version. Our findings improve the body of knowledge on the effect of using the GPU in green computing.

Keywords: Green Computing; Greening; GPU; Path Finding Robot Simulation; Migration.

1. Introduction

There are many views on the evolution of software systems and the development process. Basili and Musa classified this evolution into the following eras: functional, planning, cost, and quality [1]. The functional era concerns the 1960's. The main goal of the development process was to produce software able to meet the functional

requirements of both public institutions and industry. With this goal in mind, special development processes were defined in the functional era. The 1970's constitutes the planning era. The goal was to propose development processes that promoted software development in a planned, scheduled, and controlled manner. The notion of the software life cycle and the identification of phases and tasks that characterize them were introduced in the planning era. The 1980's is considered the cost era. At that time, the goal of reducing software development and evolution costs was seen as fundamental. Cost and effort estimation models and validation methods were introduced to address this problem. Quality played the leading role in the 1990's (referred to as the quality era). Software engineers started to quantify the quality of software systems in this era.

The last decade has seen the start of a new era thanks to the spread of the Word Wide Web. This era could be called the global era. The main goal was to design and develop distributed software systems (e.g., web-based or service oriented) using distributed development models [2].

In recent years, the power consumption of servers, data centers, and electronic devices has become a major concern [3]. For example, the power consumption of businesses in the USA doubled between 2000 and 2006 [4]. In order to tackle this problem and in the context of an increasing desire for eco-sustainable development, a new era is being born that could be called the "green era". Its relevance is widely recognized in information technology as shown by the existence of new conferences and workshops (e.g., IEEE/ACM GreenCom, HotPower, and GreenS).

Very often in the past, the shift from one era to another has resulted in adapting, porting, migrating, and re-engineering existing software systems to adapt them to new business needs [5, 6]. Then, it is easy to imagine that in the next few years an increasing interest in green technology could be manifested in the definition of methods, techniques, and tools which will offer to the existing systems a new lease of life to satisfy the desire for energy reduction and environmental sustainability.

In this paper, we outline the results of a research project aimed to develop a migration strategy to make more eco-sustainable existing software systems. We have applied here a strategy and a process to migrate a subject system, performing intensive and massive computation, to a target environment based on a Graphics Processing Unit[a] (GPU). We validated our solution migrating a system for multi-robot path finding simulations developed at the University of Basilicata within the research project mentioned above. The migrated system (from here on, greened system on the basis of our empirical evaluation) has been compared with the original one with respect to energy consumption and execution time. We used an energy logger tool to get the magnitude of the alternating current needed for the execution of both systems. We considered both execution time and energy consumption be-

[a]They are graphics chips that provide fine-grained and coarse-grained data and task parallelism. Modern GPUs are designed as computational accelerators or companion processors optimized for scientific and technical computing applications (e.g., [7]).

cause of the nature of the GPU. In fact, a possible reduction in execution time could not lead to a reduction of energy consumption (e.g., [8, 9]). Based on the experience gained in our study, we have also outlined possible implications for our research from both the practitioner and the researcher perspectives and some lessons learned for dealing with existing software in the context of energy reduction when using a target architecture based on the GPU.

The work presented in this paper is built on that presented in [10]. Our new main contributions can be summarized as follows:

- The strategy and the process have been better described.
- A deeper and more comprehensive discussion of related work and background has been added.
- The experimental part has been completely re-designed and re-executed. For example, a more accurate energy logger tool has been used to measure energy consumption.
- Implications from the researcher and the practitioner perspectives have been presented here for the first time. Also, lesson learned is new in this paper.

The remainder of the paper is organized as follows: In Section 2, we present motivation, background, and related work. In Section 3, we discuss possible benefits and issues in the application of a greening process to give existing software a more eco-sustainable lease of life. In this section, we also introduce our migration strategy and process. In Section 4, we report the design of our empirical evaluation. In Section 5, we present and discuss the obtained results and their implications from the researcher and the practitioner perspectives. In Section 6, final remarks, lesson learned, and future direction for our research are highlighted.

2. Motivation, Background, and Related Work

2.1. *Ecological Life Cycle and Greening*

Murugesan [11] proposes a holistic approach to fully and effectively address the impact of computers on the environment. This approach is focused on the entire software life cycle and addresses the problem of environmental sustainability using the following four steps:

Green Use. Reducing the energy consumption of computers and other information systems and using them in an environmentally sound manner;
Green Disposal. Reusing, refurbishing, and recycling electronic devices (e.g., personal computers and laptops);
Green Design. Designing energy-efficient computers, servers, cooling devices, and data centers with low environmental impact;
Green Manufacturing. Building electronic components, computers, and other hardware devices with low environmental impact.

Although the sustainability issue is important and socially relevant, the costs and risks engendered by the replacement of a software system which has evolved over the years through the input of significant economic and human resources cannot be met by the software companies themselves. These companies are not motivated to develop software systems for hardware architectures with a potential low environmental impact, for example, integrated GPUs or cloud computing[b]. Therefore, the most reasonable solution for these companies could be the migration of their systems to a more eco-sustainable hardware architecture.

In this way, the eco-sustainable development of software would have a greater impact in building a society and economy of green software. In addition, companies would benefit from an economic return due not only to save energy, but also to green disposal and manufacturing. Therefore, it is evident that a new step has to be added to those proposed by Murugesan [11]. We propose here the step: ***Green Software Development***. This step concerns sustainability and energy efficiency into all the phases of the software lifecycle. For the software maintenance, the main goal is the migration (or greening) of existing software systems to reduce energy consumption and improve sustainability. Greening will also preserve past investments and all the costs related to the development and the past maintenance operations performed.

2.2. *Total Cost of Ownership*

In the TCO (Total Cost of Ownership), the energy consumption is a financial estimate for direct and indirect costs of a product or system. Today, TCO is considered a dominant factor also for software systems (e.g., [13]). It also includes additional costs such as the cooling infrastructure and provisioning. Energy reduction represents one of the main drivers toward environmental sustainability (see for example the case of the Google data center in Alaska). To a first approximation [14], both cooling and provisioning costs are proportional to the average energy consumed, therefore improvements in energy efficiency should reduce all related costs.

2.3. *Save Energy Today*

In 1992 Energy Star[c], a voluntary labeling program, was launched by the U.S. Environmental Protection Agency. This program was designed to promote and recognize energy-efficiency in monitors, climate control equipment, and other electronic devices. Since the launch of this program, academia and industry have shown an increasing interest in the design and development of eco-sustainable systems, and then the term "green computing" has been coined.

Today, this phenomenon is very prevalent in the database community, where the focus is on how traditional query processing and optimization techniques can

[b]Cloud computing is a new paradigm in which computing resources such as processing, memory, and storage are not physically present at the user's location [12]. Instead, a service provider owns and manages these resources, and users access them via the Internet.
[c]http://www.energystar.gov/

be extended to contexts where energy consumption is a key issue. Therefore, this community is concentrating its efforts on two relevant areas: hardware and software. The most appropriate hardware differs substantially depending on whether raw performance or energy efficiency is the primary goal. For example, Rivoire et al. [8] propose a first benchmark for energy-aware database systems. The literature also provides examples and results on the use of flash memory to extend RAM (Random Access Memory) or to extend/replace disk storage [15]. As for software, one trend consists in the implementation of query optimizers that compare alternatives based on energy consumption, and that manage the allocation of data structures according to energy consumption [16]. Another approach is proposed by Govindaraju et al. [17] who propose an implementation of various common database operations on the GPU, thus continuing the trend of the last few years, which has seen more and more computational power given to desktop PCs through the GPUs.

Modern GPUs are not only powerful graphics engines, but also a highly-parallel programmable processor featuring peak arithmetic and memory bandwidth that outpaces its CPU counterpart. For example, Rofouei et al. [18] performed a study into the power and energy cost of the GPU operations and carried out a cost/performance comparison with a CPU-only system. The results indicated that GPU-based systems reduce energy consumption if the performance gain is above a certain limit. That is, reducing execution time does not mean reducing energy consumption. The results obtained by [18] are consistent with those we achieved in the study presented in this paper. The most substantial difference with respect to our proposal is that we introduce here a migration strategy and analyzed the data gathered from the execution of both the existing system and migrated one.

Energy reduction is also becoming relevant in servers, cloud computing, and embedded systems [14, 19]. This phenomenon is called energy-proportional design. It is aimed to save energy and potentially double server efficiency in real-life use. Barroso and Hölzle [14] state that achieving energy proportionality will require significant improvements in the energy usage profile of every system component. This is particularly true for memory and disk components. Cloud computing can also be used to save energy. For example, computation offloading and virtualization could save energy for mobile applications [19]. However, cloud computing does not represent the panacea for eco-sustainability [20].

2.4. *Dealing with Existing Software Systems*

Software migration is the process of moving a given software from one operating environment (hardware and/or software) to another operating environment (i.e., the target environment), while retaining the original system data and functionality [21]. Migration as intended here does not mean installing a given system onto an hardware operating environment that has been implemented using low-power techniques (e.g., multiple voltage planes, clock gating, or dynamic voltage-frequency scaling). In fact, in such scenario the characteristics of the original system are not taken into

consideration and so all the chances to improve its sustainability are somewhat not evaluated.

Migration is very relevant in software maintenance [22]. In fact, during the past two decades several migration approaches have been presented [23–28]. Some of them have been proposed to convert the data of a legacy database and to enable the communication between the existing source code and the newly developed database (e.g., [27,28]). Other approaches migrate existing software systems to distributed architectures such as client-server [29], distributed objects [30], web-based, and service-oriented (SOA) [5]. Whatever the approach, reverse engineering and reengineering methods are needed. Furthermore, due to the risk involved in migration projects [5,29,31] the greater part of these approaches are incremental [22] and decompose legacy systems into smaller chunks that can be then: *(i)* encapsulated; *(ii)* reengineered; or *(iii)* redeveloped. All these approaches have not been conceived to deal with issues related to the energy consumption.

Encapsulation is the least costly and risky alternative because needs a minimum number of changes [32]. Encapsulated components can be batch programs, online transactions, programs, or even just simple blocks of code [29]. The provided functionality is accessed thorough a wrapper that implements the interface that the newly developed source code uses to access existing and encapsulated components. Therefore, wrappers are responsible for passing input parameters to the encapsulated component, capturing the output data, and returning them to the requester. Different wrapping approaches have been presented in the literature, which differ in the technology they use to encapsulate source code at different granularity levels (e.g., [33–35]). Differently from the migration strategies and approaches proposed in the past [22], encapsulation can be marginally applied in the migration of a given system to a low consumption target environment. In this context, encapsulation has to be rethought. In fact, the most reasonable approach to reduce energy waste seems to recover individual software components and identify those that can be reengineered or redeveloped and successively encapsulated. Similarly to the approaches proposed in the literature (e.g., [29,33]), incremental strategies seem the best solution to deal with existing systems and to preserve past investments.

2.5. *Related Work*

Eco-sustainability is new for software engineering (e.g., GreenS [3], one of the most specific workshops on green software engineering, is at its second edition [36]). For example, Sahin et al. [37] present a preliminary empirical study that investigates the impacts on energy usage of applying design patterns. The results of the data analysis indicate three main findings: *(i)* applying design patterns can both increase and decrease the amount of energy consumed by a system; *(ii)* design patterns within a category do not impact energy usage in similar fashion; *(iii)* it is improbable that energy usage can be estimated by only taking into consideration design level artifacts; and *(iv)* most research exploring the relationship between software design

and power consumption is speculative rather than based on empirical evidence. According to these results, it seems relevant to define approaches for the recovery of "energy wasting" software components both at coarse-grained and fine-grained level. A first step in that direction could consist in adapting existing approaches for the recovery of software architecture [38, 39] and prototypical micro-architectures (i.e., design patter instances) [40]. These research topics are longstanding and relevant in the software engineering. For example, Adritsos and Tzerpos [38] present LIMBO, a hierarchical algorithm for software clustering. The clustering algorithm considers both structural and non structural attributes to reduce the complexity of a software system by decomposing it into clusters. More recently, Scanniello et al. [39] propose an approach to recover the implemented architecture of a software system with a hierarchical structure. The approach is based on the combination of structural and lexical information. The structural dimension is used to decompose a software system into layers, while the lexical dimension to partition each layer into components. As far as the recovery of design patter instances, De Lucia et al. [40] present an approach for recovering instances of structural design patterns from object-oriented source code. Design pattern instances are recovered at a coarse-grained level and then candidate instances are validated by a fine-grained source code analysis phase. Several approaches for architecture recovery (based on static and/or dynamic analyses) have been proposed in the literature [41, 42], but till now, none has pursued eco-sustainable goals.

3. Migration

3.1. *Benefits*

In the past, the shift from one era (see Section 1) to the next one, together with methodological and/or technological innovations, resulted in the need to migrate software systems [6]. Therefore, it is reasonable to assume that there will be increasing interest from academia and industry in defining methods, techniques, and tools to migrate software systems that help to promote the construction of a sustainable economy. The definition of these new technologies may offer the following benefits: *(i)* preserving the value of past investments; *(ii)* reviewing marketed systems; *(iii)* meeting the new needs of the software market; *(iv)* reducing energy wastage and possibly increasing the performance of new systems; and *(v)* promoting a new business engine for IT companies.

3.2. *Issues*

The key issues of migration are managerial and technical. Management issues may include: alignment with customer priorities, staffing, and estimating costs. Technical issues may be related to a limited understanding of the system, impact analysis, testing, and identification of the target environment. In a real migration project a trade-off is very often made between management and technical issues [5]. In the

following discussion, we focus on the points that seem most interesting to green existing software systems.

3.2.1. *Management Issues*

The transfer of innovative technologies and tools to practitioners has long been investigated in IT [43]. Technology transfer benefits from empirical studies that show advantages deriving from the application of new technologies [44, 45]. For example, the adoption of methods, techniques, and tools could be promoted through experimentations to estimate the effort and the cost for migrating a software system to a target environment [46]. New approaches are also needed to estimate electrical energy consumption for migrated systems. These could be used to verify the energy saving in the case where the migrated system will be developed and used.

3.2.2. *Technological Issues*

An important role in the migration of software systems is played by reverse engineering and approaches for program comprehension. The maintenance community classifies these approaches as static and dynamic depending on the kind of analysis they perform. Static approaches analyze source code without executing it, while the source code is executed in the case of dynamic approaches [41]. Static and dynamic analysis approaches could be integrated (e.g., [47]), so obtaining hybrid approaches.

Static analysis may be used to recover object architectures [48] and design patterns [40] thus decomposing the original systems into loosely coupled subsystems. Each subsystem could then be migrated to an hardware component corresponding to the service it implements and possibly adapted to this hardware. In this scenario, computation and data management could be distributed across computers, servers, cooling devices, and data centers with low environmental impact. Modern computing paradigms (e.g., cloud computing and service oriented architectures) could be adopted. Approaches have also been defined to statically locate concepts and features in source code [49]. These approaches could be exploited to identify classes which can be replaced by components that implement similar features, but are more energy-efficient. Dynamic approaches could be used, for example, to recover the dynamic behavior of software systems [50]. The invocation of methods or procedures may be useful in understanding which parts of the system can be migrated to a more eco-sustainable target environment. For example, if a part of the source code performs intensive computation, this is a candidate for migration.

Finally, one of the most challenging technological issues for software migration is the testing of the migrated system. Passing from a source operating environment to a target environment mainly produces a new software and then new bugs could be introduced by maintainers. A possible solution could consist in performing regression testing, where test cases could be derived from the original system and then used to exercise the migrated system [5].

3.3. *The Used Migration Strategy*

We adopted here the generic migration strategy proposed by De Lucia et al. [5]. This strategy is composed of the following steps:

(1) *Assessing the Current System.* The first step to migrate a subject software system is to assess it from both the technical and management perspectives. All the available documentation and resources (e.g., use cases, source code, fault history, operational profile, and developers' knowledge) should be used.
(2) *Defining the Target Environment.* The target environment has to be chosen so that it faces with the migration needs.
(3) *Identifying Problems and Risks.* Changing a running system is always associated with risks. The possible migration problems and risks have to be identified and assessed in order to delineate possible alternatives to be undertaken in case a problem happens. Some problems/risks may be managed, while others may lead to the failure of a migration project.

Our current system (or subject system from her on) regards a massive robot simulation in the path planning of thousand of robots with respect to obstacles in any two dimensional environments. The system enables to identify all potential obstacles to find a suitable path for all robots given a set of start positions. This system has been developed at the University of Basilicata within a research project conducted in cooperation between the Department of Computer Science and the Department of Engineering and Environmental Physic. The objective of this project was to develop a simulation system to identify difficulties facing mobile robot navigation in several application scenarios, such as instance manufacturing, mining, military operations, search and rescue missions, and so on.

We decided to renew this system because additional regional funds were received to follow these goals: *(i)* increasing parallelization, *(ii)* improving performances, and *(iii)* reducing energy waste.

In the following subsections, we discuss how each step of the migration strategy proposed by De Lucia et al. [5] has been instantiated in our research.

3.3.1. *Assessing the Current System*

Since the documentation was incomplete, we analyzed the source code (static analysis) and the system execution behavior (dynamic analysis) and exploited the knowledge of the developers who implemented some of the software component of the system. We began with a face-to-face meeting were the system and the goal of the original project were informally presented and discussed. This phase was important to share knowledge among the original developers and the researchers and to discuss possible technical and management project issues. After this meeting, we established a communication channel with the original developers by means of instant messaging tools and e-mails.

To analyze the source code of the original system, we used some of the static analysis tool prototypes developed at University of Basilicata and others were adapted to handle the problem at the hand (e.g., [51, 52]). When possible general purpose commercial and open source tools we employed, such as Enterprise Architect and Borland Together.

The assessment of the system revealed that it had a good level of decomposability. For example, the C++ files implementing the presentation logic was separated from those implementing the application logic and data access layer. The dynamic analysis also showed that some of the components performed computationally intensive tasks. We performed a manual analysis on these components. This analysis revealed the massive usage of concurrent threads, so making these components suitable to be migrated or reengineered to meet the project requirements.

3.3.2. *Defining the Target Environment*

In the context of our research, the target environment should give improvement in term of energy efficiency and computational performances. Due to the results of past studies (e.g., [18,53]), our implementation relies upon the computational capability of a modern Graphics Processing Unit (GPU). In fact, with the advent of compute-capable integrated GPUs having a power consumption of tens of Watts, it is possible to save energy and outperform the CPUs of a multi-core system. In particular, we adopt NVIDIA's GPU architecture with its CUDA parallel programming model.

Modern NVIDIA GPUs are fully programmable multi-core chips known as CUDA processors [54]. In a GPU, a streaming multiprocessor (SMX) is a cluster of streaming core processors. Each core can only execute one thread at a time, but when a thread performs an operation with high latency it is put into a waiting state and another thread executes. This feature, and the application of a scheduling policy means that the core can run many threads at the same time. In the generation of GPUs (codenamed Fermi), the number of SMXs ranges from 1 to 16. The GPU used in our experiments (GF100) consisted of 15 SMXs, each containing 32 single-precision CUDA cores. Because each SMX is capable of supporting up to 1536 threads, this GPU manages up to 23048 resident threads. All thread management, including creation, scheduling and barrier synchronization is performed entirely in hardware by the SMX with virtually zero overheads.

In terms of the software model, CUDA C [55] (or simply CUDA) provides software developers with facilities to execute parallel programs on the GPU (Figure 1). To use CUDA, a programmer needs to write their code as special functions called kernels, which are executed across a set of parallel threads. The programmer organizes these threads into a hierarchy of blocks and grids. A thread block is a set of concurrent threads that can cooperate via shared memory (which has a latency similar to that of registers) and can be coordinated using synchronization mechanisms. A thread grid is a set of thread blocks executed independently. All threads have access to the same global memory space. Each thread block is mapped to an

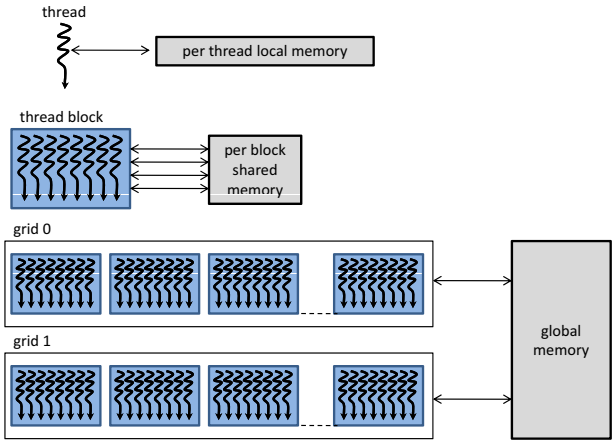


Fig. 1. Levels of parallel granularity and memory sharing on the GPU.

SMX and executes concurrently. SMX resources (registers and shared memory) are split across the mapped thread block. As a consequence, this limits the number of thread blocks that can be mapped onto one SMX.

The programming model of the CUDA language presents a number of similarities with the C programming language. This aspect may have practical implications. Beyond the programming model that is based on parallel programming which is completely different from sequential programming, a company could have developers with C knowledge to be easily skilled on CUDA. In this programming model, the CPU is called host and the GPU is called device. A CUDA program consists of several phases that are executed on either the host or a device. Host code exhibits little or no data parallelism while the device code exhibits rich amount of parallelism. The developer supplies a single source code encompassing both host and device code. The NVIDIA C Compiler separates the two. The host code is straight ANSI C code and is compiled with host standard C compilers and runs as an ordinary process. The device code is written in ANSI C extended with keywords for labeling data-parallel functions, namely the kernels. The CUDA threads are of much lighter weight than the CPU threads and they take very few clock cycles because they are scheduled in hardware with virtually no overhead. Differently, the CPU threads that typically take thousands of clock cycles to generate and schedule threads. The programmer develops the kernel and chooses when the execution is moved to a device and how many threads have to be generated.

3.3.3. *Identifying Problems and Risks*

The most important risk was concerned to the real improvement in term of energy efficiency and computational performances. For example, it could be possible that the migrated system does not meet the expected migration goals, namely the

migrated system is not more energy efficient than its original version.

Other issues could be related to the limited understanding of the system, impact analysis, and testing. To deal with the limited understanding of the system, we performed a meeting and established a communication channel with the original developers. Special conceived and general purpose tools were also used to comprehend the original system.

The complexity of the GPU architecture represents another risk for the migration. In this context, coding needs in-depth knowledge of resources available in terms of thread and memory hierarchy. Although approaches have been proposed to deal with migration risks (e.g., [45]), there is a lack of approaches in the greening context and regarding the estimation of the energy efficiency. In particular, formal and hybrid methods are not available due to the unavailability of historical data. Therefore, only expert opinion based methods can be applied. To deal with the migration risks of our project, we exploited our experience in software maintenance and parallel computing.

3.4. *The Adopted Migration Process*

According to the shown migration strategy, our migration process is based on the following phases:

(1) *Reverse Engineering*. In this phase suitable models of the subject system are produced. This phase has also the effect of increasing the comprehension of that system and its source code.
(2) *Reengineering*. Components that perform computation intensive tasks are identified and reengineered. The goal of this kind of step is mostly new in migration.
(3) *Integration*. The newly developed components are wrapped and integrated.
(4) *Testing*. The greened system is tested.

Details on these steps are provided in the following subsections.

3.4.1. *Reverse Engineering*

Reverse engineering tools are used to comprehend the system to be migrated and to reconstruct its architectural view. In particular, we used the CodeTrees environment [51] to ease the overall comprehension of the medium-sized software system (14,472 total lines of code, 1,076 comment lines, and 58 source files) to be greened.

3.4.2. *Reengineering*

The components that performed intensive computation were identified. We planned to incrementally migrate the system to reduce the failure risk. A prioritization of the components to be migrated was defined on the basis of the information gained in the Reverse Engineering phases. The knowledge of the original developer involved in the project should be also taken into consideration.

We focus here on the component that has been recognized as the most risky to be migrated. That is, the component implementing the A* search algorithm [56,57]. The input of this algorithm is: a set of pairs start - goal positions one for each robot and a grid map for the search space representation. For each position s, a heuristic $h[s]$ is used to estimate the goal distance, which is the cost of a minimal path from the position s to a goal state. Classical heuristics are based on Manhattan, diagonal, or Euclidean distance calculations. During its execution, A* maintains two values, $g[s]$ and $f[s]$. The value $g[s]$ is the smallest cost of any discovered path from a start position s_{start} to position s. The value $f[s] = g[s] + h[s]$ estimates the distance from s_{start} to the goal position via s. At each iteration, A* expands the states from which all adjacent states have been explored and tries to update the g-value of each visited state with a lower value. At the end, the g-value of each visited position s will be the distance from the start position s_{start} to position s. The main drawback of this algorithm is the computation time. This makes the algorithm unsuitable for massive robots path planning in large state spaces where the simulation must be performed in real-time. A possible solution to deal with this issue is to use a multi-core CPU. Since the CPUs have a limited number of cores this approach offers a partial solution when the number of robots increases. Conversely, GPUs offer the execution of many number of threads enabling to perform concurrently much more A* searches. To reengineer software components, we used here the CUDA 4.0 programming language.

3.4.3. *Integration*

In the GPU parallel path finding, a robot is a CUDA thread. Before beginning the computation, the input data have to be copied from the CPU (host) to the GPU (device). The input is a grid map that represents the search space and includes the start and goal positions of each robot in the simulation. In our case, the kernels are four. One kernel is in a charge of executing the A* algorithm. Other two kernels are for the initializations needed for the algorithm execution. The latter kernel concerns the output to be visualized in the graphical user interface of the new system.

For each kernel an execution configuration has to be specified as well. This configuration includes the number of threads in a block and the number of blocks in a grid. Then, the GPU in parallel executes the threads of each kernel. Kernels could be executed in sequence or in parallel. In our solution, we executed the kernels in sequential fashion due to the nature of the problem: the output of the A* algorithm is available only when its execution is concluded.

When the execution is concluded, the output is moved from the GPU to the CPU. For each robot the output consists in a path that is built backward from the goal node to the start node. The output is then visualized in the graphical user interface. Figure 2 shows a screenshot of that system. The dots represent the start and goal positions of the robots and their line paths. Since the graphical user interface is the same for both the original system and greened one, the end-user

should not perceive any difference except for the fact that the new greened system is faster as we experimentally observed.

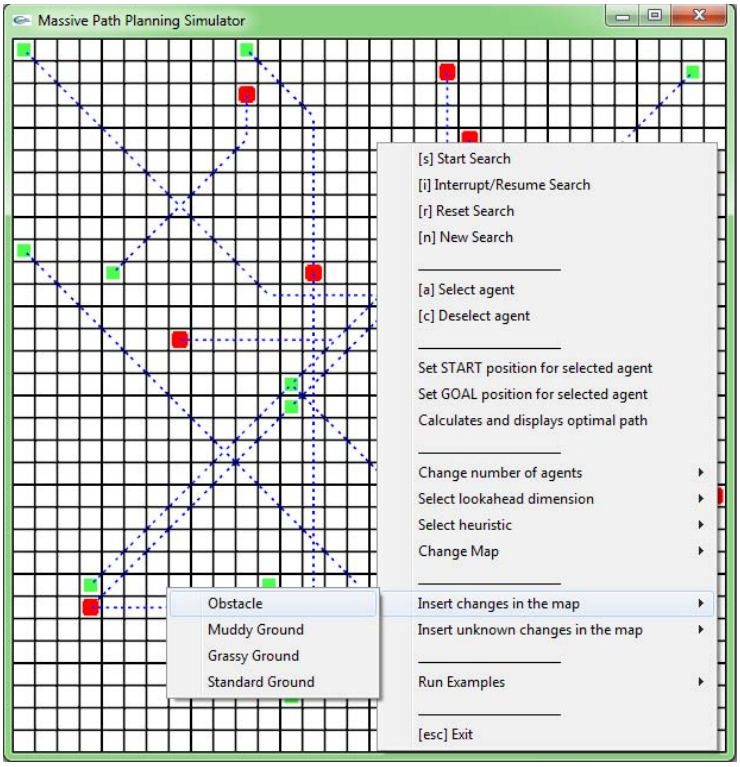


Fig. 2. A screenshot of the system.

Figure 3 shows an excerpt of the source code for invoking the implementation for the GPU of the A* algorithm. The statements from 2 to 5 are in charge of allocating the memory for the CPU, while those from 8 to 15 allocate the GPU memory. The copy from the CPU memory to the GPU memory is executed by the statements from 18 to 20. The statements from 23 to 32 are used for the starting configuration. The four kernels are invoked in the statements from 35 to 38. In particular, the A* algorithm implementation is invoked in the statement 36. Finally, the statement 41 deals with the moving of the execution output from the GPU to the CPU.

3.4.4. *Testing*

The software engineer performs testing to verify the introduction of new bugs in the greened system. In particular, regression testing has been executed using test cases derived from the original system. Test cases included: input and output. To identify possible differences in the behavior, the input has been used to exercise

```
1. //CPU memory allocation
2. int *h_map = (int *) calloc(map_width * map_height, sizeof(int));
3. int *h_start = (int *) calloc(robot_number, sizeof(int));
4. int *h_goal = (int *) calloc(robot_number, sizeof(int));
5. int *h_path = (int *) calloc(robot_number * map_width, sizeof(int));
6.
7. //GPU memory allocation
8. int *d_map, *d_path, *d_start, *d_goal;
9. size_t size_map = map_width * map_height * sizeof(int);
10. size_t size_robot = robot_number * sizeof(int);
11. size_t size_path = robot_number * map_width * sizeof(int);
12. cudaMalloc((void**)&d_map, size_map);
13. cudaMalloc((void**)&d_start, size_robot);
14. cudaMalloc((void**)&d_goal, size_robot);
15. cudaMalloc((void**)&d_path, size_path);
16.
17. //Copy data from the CPU to the GPU
18. cudaMemcpy(d_map, h_map, size_map, cudaMemcpyHostToDevice);
19. cudaMemcpy(d_start, h_start, size_robot, cudaMemcpyHostToDevice);
20. cudaMemcpy(d_goal, h_goal, size_robot, cudaMemcpyHostToDevice);
21.
22. //Execution configuration
23. int threadForBlock = 128;
24. int block_x = threadForBlock;
25. int block_y = 1;
26. int block_z = 1;
27. int grid_x = 1;
28. int grid_y = 1;
29. block_x = threadForBlock;
30. grid_x = (robot_number + (threadForBlock - 1)) / threadForBlock;
31. dim3 dimBlock(block_x, block_y, block_z);
32. dim3 dimGrid(grid_x, grid_y, 1);
33.
34. //Kernels invokation
35. initialize_gpu_memory<<<dimGrid, dimBlock>>>();
36. initialize_a_star<<<dimGrid, dimBlock>>>();
37. a_star_algorithm<<<dimGrid, dimBlock>>>();
38. build_paths<<<dimGrid, dimBlock>>>();
39.
40. //Copy output from the GPU to the CPU
41. cudaMemcpy(h_path, d_path, size_path, cudaMemcpyDeviceToHost);
```

Fig. 3. Source code to invoke the greened component.

the greened system and the output is compared with output of the new system. Currently, we do not provide any specific support to derive test cases and perform regression testing. This point is subject of future work.

4. Design of the Experiment

The main goal of our experiment is to evaluate possible benefits deriving from the application of the proposed migration strategy on the considered system in term of execution time and the possible reduction of energy consumption. To this end, we compare the original system and the migrated one with respect to these criteria. We considered both execution time and energy reduction because they are not directly related as we introduced in Section 2.3 and as we will discuss in Section 5.2.

In this section, we present the design of our experiment following the guidelines proposed by Wohlin et al. [58]. For replication purposes, the experimental package and the raw data are available on the web[d]. The executable files of both the original and the greened systems are also available for downloading.

[d]www2.unibas.it/gscanniello/Greening/

4.1. *Definition*

The original and the greened systems were executed on the same input to understand whether or not differences in terms of execution time and energy consumption were present. In this context, the baseline was the original version of the system.

To ensure that important aspects were defined before the planning and the execution of the experiment took place [58], we used the GQM (Goal Question Metric) template [59]. According to that template, the goal of our experimentation can be defined as follows:
Analyze *the greened system* ***for the purpose of*** *comparing it with the original one* ***with respect to*** *execution time and energy consumption* ***from the point of view of*** *the maintainer* ***in the context of*** *the massive robot simulation in the path planning, and* ***from the point of view of*** *the researcher* ***in the context of*** *the validation of the proposed migration strategy and process.*

We have then defined and investigated the following two research questions:

RQ1 Does the greened system outperform the original system in terms of execution time?

RQ2 Is the energy consumption of the greened system lower than that of the original one?

4.2. *Context*

The comparison was performed over a number of different input configurations on two grid maps of 128×128 and 256×256 tiles. We choose these two kinds of grid maps on the basis of the actual use of the original system. Grid maps either smaller than 128×128 or larger than 256×256 were practically never used.

Each configuration input was constituted of a number of robots and of a number of obstacles that have to be avoided by these robots. The number of robots were: 5,120, 51,200, 512,000, 5,120,000, and 51,200,000. The number of obstacles for each row of the grid map was: 0, 5, 10, 15, 20, and 25. The number of robots and obstacles is a consequence of the chosen dimension for the grid maps. Robots are positioned on a grid map (more than one robot can be placed on each point on the map) and obstacles are placed on each row of that map randomly.

To get a deeper understanding of how execution time and energy consumption vary with respect to the size of a grid map and the number of robots and obstacles, we have considered all their possible combinations as input configurations. Note that only these three aspects could be chosen by the user of the original and the migrated software systems.

4.3. *Variable Selection and Design*

The main factor of the study is Implementation (the variable we manipulated). It is a nominal variable with two possible values: CPU and GPU. CPU is referred to the original system, while GPU to the greened one. We also considered: Map, Robot,

and Obstacle. Map is the dimension of the grid map, while Robot and Obstacle are the numbers of robots and obstacles, respectively. In our experiment, these variables are ordinal and their levels are those we report in Section 4.2.

According to the defined null hypotheses, we selected and used the following dependent variables:

- **Time**. It indicates the computation time and it is measured in seconds.
- **Consumption**. It indicates the total energy consumption measured in Joule.

We considered and analyzed 60 samples (or units) in terms of Time and Consumption with respect the original and greened systems. In other words, the two implementations are applied to each of 60 samples and the pair of treatments are applied to the same samples.

4.4. *Preparation and Execution*

The empirical evaluation was performed on a PC equipped with an Intel Core 2 Duo E7400 2.80Ghz processor, a NVIDIA Fermi GTX 480 1.5GB video card, and Windows 7 as the operating system.

To get the execution time of the original and the greened systems on each configuration input, we instrumented source code. To measure the Joules, we used the energy logger tool *Watts up?*[e]. We opted for this logger because it has been successfully used to measure energy consumption for GPU based implementations (e.g., [60,61]). The use of an energy logger was needed because there are not accepted metrics to determine energy usage when dealing with the GPU based implementations. This is not completely true for those implementations based on CPU (e.g., [37,62]). Therefore, to conduct a comparison as much as possible fair between the two considered implementations, we used the energy logger also for the CPU implementation.

It is worth mentioning that the GPU acts as co-processor and then in the total energy consumption there is also the consumption related to the idle state of the CPU. We also point out that each test was ran five times. We presented here the worst results for both the original and greened systems. Note that the results in terms of computation time and energy consumption among these runs were similar.

4.5. *Threats to validity*

A threat is related to the energy logger tool used to measure energy consumption. In fact, the accuracy of the measurements might affect results. However, the use of the chosen energy logger tool equally affects the measurements for both the implementations. To increase our awareness in the achieved results, we plan future replications by exploiting at least one different energy logger tool. Another possible threat is due to configuration input we used to perform the comparison between the two different implementations of the system.

[e]www.powermeterstore.com/crm_uploads/wattsup.pdf

The quality of the system implementation might affect the validity of results: it underwent few maintenance operations. This from one side made the migration easier and from the other side might reduce the differences from the original system to the greened one with respect to execution time and energy consumption. Then, to improve our confidence in the achieved results, our migration process should be applied on other software systems. This will also allow a better characterization of which kind of system would benefit more from our solutions.

5. Results

5.1. *Descriptive Statistics*

Some descriptive statistics (i.e., min, max, mean, median, and standard deviation) on Time and Consumption are reported in Table 1. These statistics show that the execution time of the greened system is lower than that of the original one. The same holds for energy consumption. Another remarkable finding is that the standard deviation values for Time and Consumption are always lower for the greened system. In other words, the data points for both Time and Consumption tend to be closer to their mean values in case of the greened system, thus suggesting that its execution and energy consumption are more predictable.

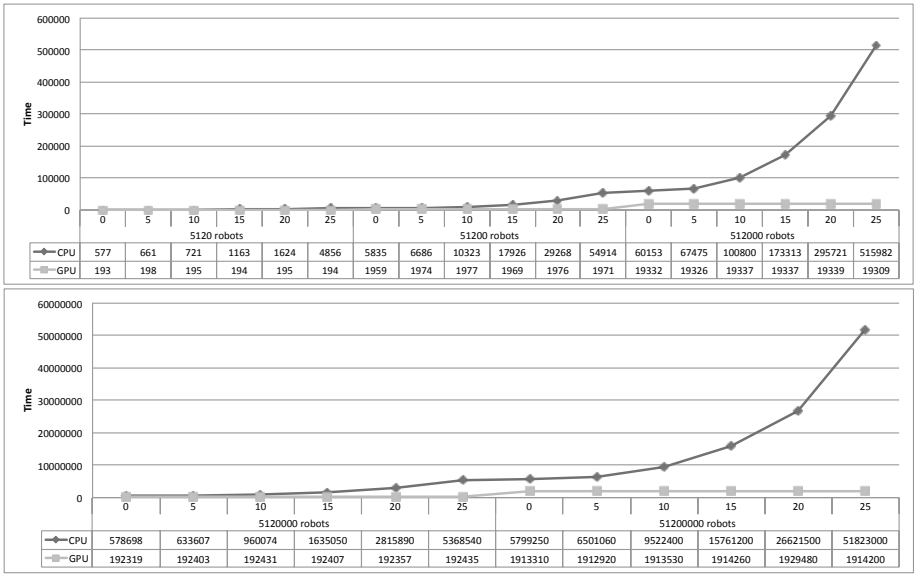
Table 1. Descriptive statistics on Time (seconds) and Consumption (Joule)

Variable	Implementation	Min	Max	Mean	Median	De.St.
Time	CPU	577	61130000	5979000	202700	13441613
	GPU	193	4985000	766700	34610	1526421
Consumption	CPU	57	6430000	628000	21610	1408505
	GPU	29.8	1066000	159600	6417	321328.1

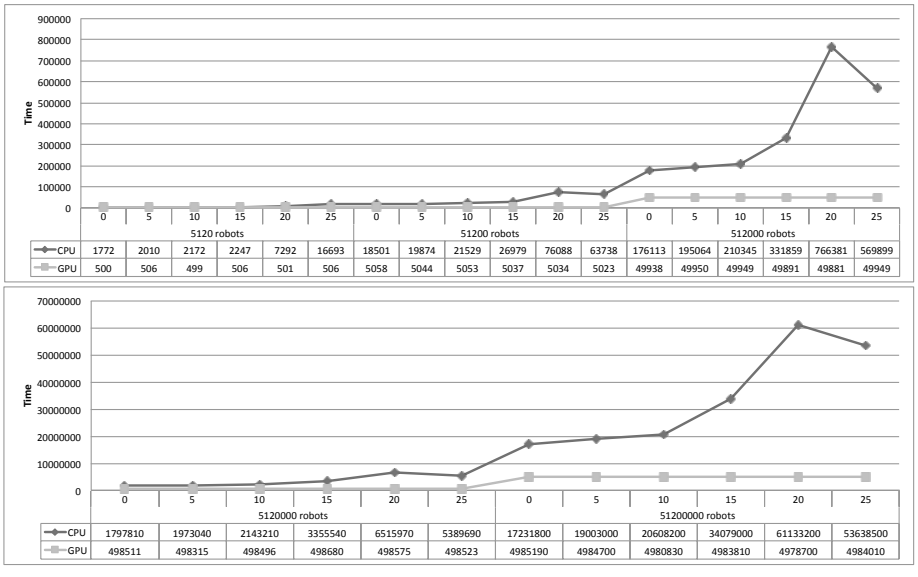
Figure 4 and Figure 5 show the results achieved in terms of execution time and energy consumption for all the input configurations considered. As for Time, Figure 4.a shows the results for CPU and GPU achieved on a grid map 128×128. On the top of Figure 4.a, we graphically summarize the results achieved with 5,120, 51,200, and 512,000 robots, while on the bottom those for 5,120,000 and 51,200,000 robots. We reported the results in this way because of the magnitude of the differences in the execution time achieved in the two groups of configuration input. Similarly, Figure 4.b shows the results achieved for a grid map 256×256. The energy consumption results are visually shown in Figure 5.a and Figure 5.b for the grid maps 128×128 and 256×256, respectively.

The execution time of the greened system is always lower than the original one as Figure 4.a shows. Moreover, the differences in the execution time are more evident when the number of robots and obstacles increases. Indeed, the number of obstacles affects the execution time only in the case of the original system.

Figure 5 suggests that the new system has a lower energy consumption with respect to the original one. This is true for all the input configurations. As for

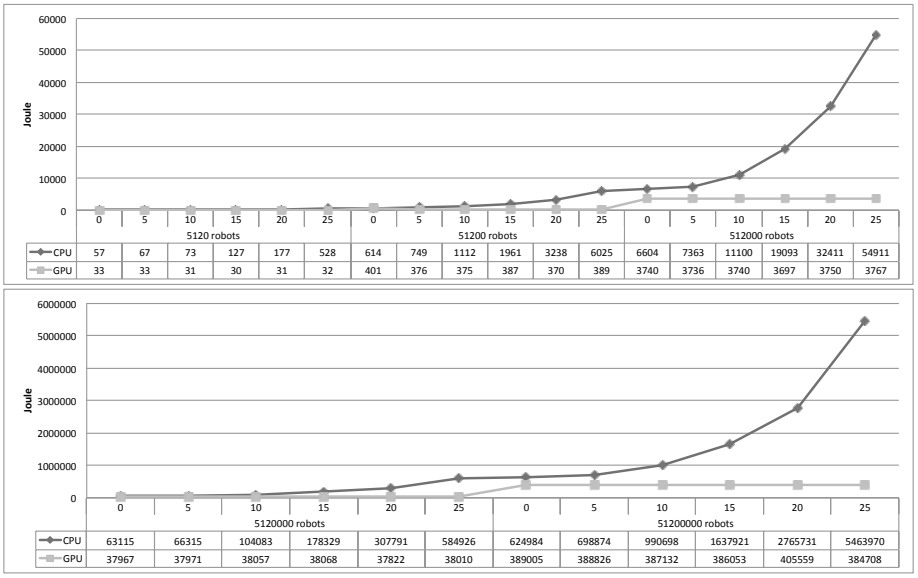


(a)

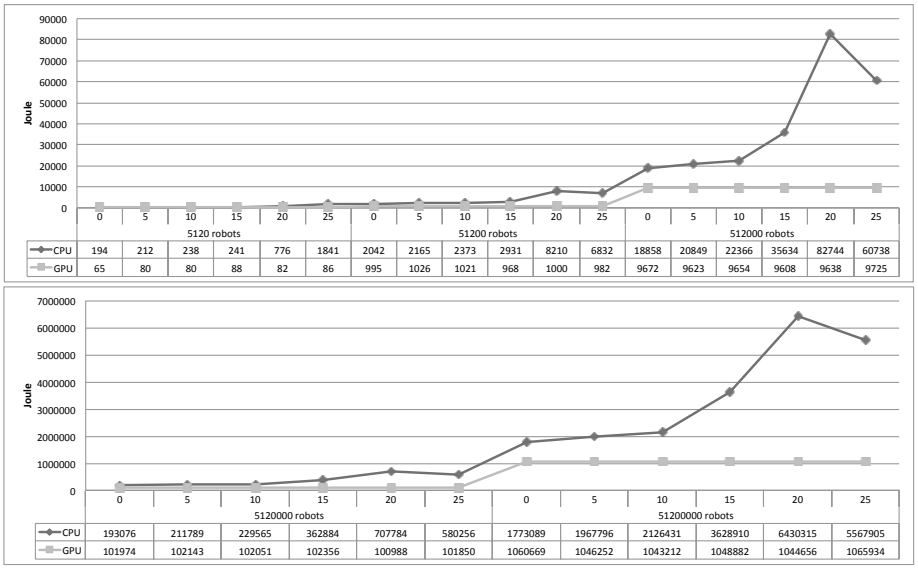


(b)

Fig. 4. Graphics for Time (in seconds) (a) 128×128 and (b) 256×256



(a)



(b)

Fig. 5. Graphics for Consumption (in Joule) (a) 128×128 and (b) 256×256

Time, the higher the number of robots and obstacles, the greater is the difference of energy consumption between the two systems. It is worth mentioning that the experiment design used and the plots shown make statistical analysis unnecessary.

5.2. *Discussion*

In the migration of a simulation and real-time software the use of the GPUs seems to make the difference in terms of execution time. The results shown in Figure 4 allow us to positively answer RQ1.

As for energy consumption, the difference between the two systems is not so clear as for execution time. On the basis of these results, we could not provide conclusive findings on whether the GPU reduces energy waste in our study. However, the results shown in Figure 5 allow us to postulate that the greened system is more eco-sustainable than the original system when the size of the input increases. Accordingly, we positively answered the research question RQ2 even if such a result should be interpreted with particular more caution with respect to that we obtained on execution time.

The results above suggest that somewhat execution time and energy reduction are not directly related, so confirming the findings in [8,9,18]. Therefore, the results of our work improve the body of knowledge on the effect of using the GPU in green computing. This is definitively one of the most important contribution of our study.

Another interesting result is that the number of obstacles on a grid map seems to have an effect on both execution time and energy consumption. This could be due to the fact that increasing the number of obstacles, the number of memory accesses to determine their positions increase. On the other hand, the effect of the number of robots seems to scarcely affect execution time and energy consumption. These two findings above hold for both the two grid maps considered here. Practically, both systems (i.e., original and greened) need more time and consume more energy to find paths for the robots on maps when the number of obstacles increases. As shown in Figure 4 and Figure 5, the number of obstacles affects more time and energy consumption in the case of the original system.

Execution time and energy consumption seem to depend on the combination of both the kind of implementation and the number of robots on the grid map (see Figure 4 and Figure 5). In addition, it seems that the number of robots affects more the execution time and energy consumption of the original system.

5.3. *Implications*

To judge the implications of our experiment, we adopted a perspective-based approach [63]. Implications on the application of our migration process and strategy are also presented here. The reported implications are also founded on qualitative results gained from the application of our migration process and strategy. We based our discussion on the *practitioner/consultant* (simply *practitioner* in the following)

and *researcher* perspectives [64]. The main practical implications of our study can be summarized as follows:

- The greened system yields a reduction of execution time and energy consumption. Very rarely our greened system is comparable with the original one in term of energy consumption and execution time (i.e., when the number of robot is low). In all the other cases, the greened system outperforms the original one. This finding is relevant from the practitioner perspective because he/she could take into account the possibility of using a GPU to reduce execution time of existing systems to make them more eco-sustainable.
- Although CUDA presents a number of similarities with the C programming language, the adoption of our migration strategy and process could require a radical process change in an interested company. In fact, a deep knowledge of the GPU architecture is needed to fully exploit its computational power. This is relevant for the practitioner, who should be specifically trained on such a kind of architecture. Also, the researcher could be interested in this point conceiving new approaches and methods to automatically support the phases of our process and strategy.
- Using our strategy allows focusing only on a part of the entire system to be migrated. This could potentially reduce the effort and the risk to give legacy systems a more eco-sustainable lease of life. This is relevant for the practitioner. From the researcher perspective, it could be interesting to study potential benefits deriving from the application of our strategy.
- Execution time and energy reduction seem to be not directly related. From the researcher perspective, this point is interesting both in our research context and in different applicative contexts. Our results pose the basis for future work.
- Our results suggest that execution time and energy consumption are affected by the kind of input. This result is relevant for the researcher, who could be interested in studying how to improve the performance of a system and to reduce its energy consumption as well. That is, if one knows possible issues that affect energy consumption and execution time, he/she can act to solve them.
- The study is focussed on a desktop application for path finding robot simulations. From the researcher perspective, the effect of using our solutions on different kinds of applications (e.g., business and web based) represents a possible future direction.
- The original system has been developed in a research project and underwent few maintenance operations. The magnitude of the benefits deriving from the use of our solutions suggests that similar results could be obtained in different real-time software (e.g., games) developed in different kinds of projects. This is relevant from both the practitioner and the researcher perspectives.
- Energy consumption in a migration project has been analyzed for the first time in our study. This could be of interest for the researcher.

6. Conclusion

The replacement of a legacy system which has evolved during the years through the input of significant economic and human resources may introduce unmanageable costs and risks. Software houses are conservative and act only when they are forced to [65]. Therefore, the most reasonable solution for companies appears to be the migration of their systems to reduce energy consumption. This satisfies the need to be technically innovative and at the same time promotes environmental sustainability. To make existing software systems more eco-sustainable, there are management and technical issues to deal with. Possible research directions related to these issues have been highlighted and discussed in the paper.

In this paper, we have also presented a process to migrate an existing system to a target environment based on the GPU. We empirically validated our solutions on a system for massive robot simulations. The migrated system has been compared with the original one. The experimental results suggested that the greened system outperforms the original one with respect to execution time. In addition, the energy consumption of the greened system is lower so resulting more eco-sustainable than the original system.

6.1. *Lesson Learned*

Our goal here is to bring together lessons learned that can be useful to researchers and software practitioners. We gained lesson learned in the application of our migration process and strategy:

- Whatever is the language to program a GPU (e.g., CUDA), a deep knowledge of its architecture and of its capabilities is needed to fully exploit the computational power of this kind of processor.
- To get a significant speed-up of the computation time, the use of the memory has to be properly managed. In particular, to get peak memory performance, computation must be structured around sequential memory accesses because random access to the memory might slowdown computation.
- Moving data between the CPU (host) and the GPU (device) might slowdown computation time and negatively affect energy consumption. For small sized data, the overheads of data transfer overshadow improvements (time and consumption). To green existing software systems, the maintainer has to consider issues related to the data transferring from and to the GPU. Time can be saved delaying data transfer operations as much as possible. The best is to transfer data only once.
- The GPU is particularly useful to execute many closely-coupled and independent parallel threads: it is based on a single-program multiple-data programming model. The GPU processes many elements in parallel using the same program and each element is independent from the other elements. Furthermore, the power consumption of a GPU is higher than a multicore CPU. Then,

the greening of an existing software system using GPU is possible only in case computation can be parallelized.

6.2. *Future Work*

It would be worth measuring the energy consumption of each part of the target system, so having information to be used in the definition of load balancing techniques to optimize energy consumption. A possible future direction could also consist in the definition of models to estimate the energy consumption of a migrated system and the effort and cost due to its migration. This will allow estimating the consumption and the possible energy reduction of the target system before its actual migration to the GPU. Estimation models would support the management of a company to deal with the issue: is it worthwhile to green our existing system? The results of our research poses the basis in this direction.

References

[1] V. R. Basili and J. D. Musa, "The future engineering of software: A management perspective," *IEEE Computer*, vol. 24, no. 9, pp. 90–96, 1991.

[2] B. Sengupta, S. Chandra, and V. Sinha, "A research agenda for distributed software development," in *Proc. of International Conference on Software Engineering*. ACM press, 2006, pp. 731–740.

[3] P. Lago, R. Kazman, N. Meyer, M. Morisio, H. A. Müller, F. Paulisch, G. Scanniello, B. Penzenstadler, and O. Zimmermann, "Exploring initial challenges for green software engineering: summary of the first greens workshop, at icse 2012," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 1, pp. 31–33, 2013.

[4] S. Rivoire, M. A. Shah, P. Ranganathan, C. Kozyrakis, and J. Meza, "Models and metrics to enable energy-efficiency optimizations," *IEEE Computer*, vol. 40, no. 12, pp. 39–48, 2007.

[5] A. De Lucia, R. Francese, G. Scanniello, and G. Tortora, "Developing legacy system migration methods and tools for technology transfer," *Softw., Pract. Exper.*, vol. 38, no. 13, pp. 1333–1364, 2008.

[6] H. M. Sneed, "Planning the reengineering of legacy systems," *IEEE Software*, vol. 12, no. 1, pp. 24–34, January 1995.

[7] S. Yoo, M. Harman, and S. Ur, "Highly scalable multi objective test suite minimisation using graphics cards," in *Proc. of International Symposium on Search Based Software Engineering*, ser. Lecture Notes in Computer Science, vol. 6956. Springer, 2011, pp. 219–236.

[8] S. Rivoire, M. A. Shah, P. Ranganathan, and C. Kozyrakis, "Joulesort: a balanced energy-efficiency benchmark," in *Proc. of ACM SIGMOD international Conference on Management of Data*. ACM press, 2007, pp. 365–376.

[9] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey, "Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 451–460, 2010.

[10] G. Scanniello, U. Erra, G. Caggianese, and C. Gravino, "Using the GPU to Green an Intensive and Massive Computation System," in *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, March 2013, pp. 384–387.

[11] S. Murugesan, "Harnessing green it: Principles and practices," *IT Professional*, vol. 10, no. 1, pp. 24–33, January 2008.

[12] M. Creeger, "CTO roundtable: Cloud computing," *Queue*, vol. 7, no. 5, pp. 1:1–1:2, 2009.

[13] J. G. Koomey, "Estimating total power consumption by servers in the U.S. and the world," 2007.

[14] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *IEEE Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[15] G. Graefe, "The five-minute rule twenty years later, and how flash memory changes the rules," in *Proc. of International Workshop on Data Management on New Hardware*. ACM Press, 2007, pp. 6:1–6:9.

[16] R. Ramamurthy and D. J. Dewitt, "Buffer-pool aware query optimization," in *Proc. of International Conference on Innovative DataSystems Research*, 2005, pp. 250–261.

[17] N. K. Govindaraju, B. Lloyd, W. Wang, M. Lin, and D. Manocha, "Fast computation of database operations using graphics processors," in *Proc. of ACM SIGMOD International Conference on Management of Data*, 2004, pp. 215–226.

[18] M. Rofouei, T. Stathopoulos, S. Ryffel, W. Kaiser, and M. Sarrafzadeh, "Energy-aware high performance computing with graphic processing units," in *Proc. of International Conference on Power Aware Computing and Systems*. USENIX Association, 2008, pp. 11–11.

[19] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *IEEE Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[20] E. Davidson, E. Vaast, and P. Wang, "The greening of it: How discourse informs it sustainability innovation," in *In Proceedings of Conference on Commerce and Enterprise Computing*. IEEE Computer Society, 2011, pp. 421–427.

[21] J. Bisbal, D. Lawless, B. Wu, and J. Grimson, "Legacy information systems: Issues and directions," *IEEE Software*, vol. 16, no. 5, pp. 103–111, 1999.

[22] M. L. Brodie and M. Stonebraker, *Migrating legacy systems: Gateways, interfaces & the incremental approach*. Morgan Kaufmann Pub; 1 edition, 1995.

[23] G. Antoniol, R. Fiutem, E. Merlo, and P. Tonella, "Application and user interface migration from Basic to Visual C++," in *Proc. of International Conference on Software Maintenance*. IEEE Computer Society, 1995, pp. 76–85.

[24] M. Ceccato, P. Tonella, and C. Matteotti, "Goto elimination strategies in the migration of legacy code to java," in *Proc. of European Conference on Software Maintenance and Reengineering*. IEEE press, 2008, pp. 53–62.

[25] J. Henrard, D. Roland, A. Cleve, and J. Hainaut, "An industrial experience report on legacy data-intensive system migration," in *Proc. of the International Conference of Software Maintenance*. IEEE press, 2007, pp. 473 – 476.

[26] F. Ricca and P. Tonella, "Using clustering to support the migration from static to dynamic web pages," in *Proc. of International Workshop on Program Comprehension*. IEEE press, 2003, pp. 207–216.

[27] G. Scanniello, A. De Lucia, M. Mennella, and G. Tagliamonte, "An approach and an eclipse based environment for data migration," in *Proc.of International Conference of Software Maintenance*. IEEE Computer Society, 2008, pp. 237–246.

[28] P. Thiran, J.-L. Hainaut, G.-J. Houben, and D. Benslimane, "Wrapper-based evolution of legacy information systems," *ACM Trans. Softw. Eng. Methodol.*, vol. 15, no. 4, pp. 329–359, 2006.

[29] H. M. Sneed, "Encapsulating legacy software for use in client/server systems," in *Proc. of International Working Conference on Reverse Engineering*. IEEE press, 1996, pp. 104–.

[30] J.-M. Lin, Z.-W. Hong, G.-M. Fang, H. C. Jiau, and W. C. Chu, "Reengineering windows software applications into reusable corba objects," *Information & Software Technology*, vol. 46, no. 6, pp. 403–413, 2004.

[31] L. Aversano, G. Canfora, A. Cimitile, and A. De Lucia, "Migrating legacy systems to the web: an experience report," in *Proc. of European Conference on Software Maintenance and Reengineering*. IEEE press, 2001, pp. 148–157.

[32] H. M. Sneed, "Risks involved in reengineering projects," in *Proc. of International Working Conference on Reverse Engineering*. IEEE press, 1999, pp. 204–.

[33] A. De Lucia, R. Francese, G. Scanniello, G. Tortora, and N. Vitiello, "A strategy and an Eclipse based environment for the migration of legacy systems to multi-tier web-based architectures," in *Proc. of International Conference on Software Maintenance*. IEEE Computer Society, 2006, pp. 438–447.

[34] H. M. Sneed, "Wrapping legacy COBOL programs behind an XML-Interface," in *Proc. of International Working Conference on Reverse Engineering*. IEEE press, 2001, pp. 189–197.

[35] P. Tonella, "Using the O-A diagram to encapsulate dynamic memory access," in *Proc. of International Conference on Software Maintenance*. IEEE Computer Society, 1998, pp. 326–335.

[36] P. Lago, N. Meyer, M. Morisio, H. A. Müller, and G. Scanniello, "2nd international workshop on green and sustainable software (greens 2013)," in *Proc. of the International Conference on Software Engineering*. IEEE/ACM, 2013, pp. 1523–1524.

[37] C. Sahin, F. Cayci, I. L. M. Gutiérrez, J. Clause, F. E. Kiamilev, L. L. Pollock, and K. Winbladh, "Initial explorations on design pattern energy usage," in *International Workshop on Green and Sustainable Software*, 2012, pp. 55–61.

[38] P. Andritsos and V. Tzerpos, "Information-theoretic software clustering," *IEEE Trans. Software Eng.*, vol. 31, no. 2, pp. 150–165, 2005.

[39] G. Scanniello, A. D'Amico, C. D'Amico, and D. Teodora, "Using the Kleinberg algorithm and Vector Space Model for software system clustering," in *Proc. of International Conference on Program Comprehension*. IEEE Computer Society, 2010, pp. 180–189.

[40] A. De Lucia, V. Deufemia, C. Gravino, and M. Risi, "Design pattern recovery through visual language parsing and source code analysis," *Journ. of Syst. and Softw.*, vol. 82, no. 7, pp. 1177–1193, 2009.

[41] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke, "A systematic survey of program comprehension through dynamic analysis," *IEEE Trans. Software Eng.*, vol. 35, no. 5, pp. 684–702, 2009.

[42] S. Ducasse and D. Pollet, "Software architecture reconstruction: A process-oriented taxonomy," *IEEE Trans. Software Eng.*, vol. 35, no. 4, pp. 573–591, 2009.

[43] S. L. Pfleeger and W. Menezes, "Marketing technology to software practitioners," *IEEE Software*, vol. 17, pp. 27–33, 2000.

[44] M. Colosimo, A. De Lucia, R. Francese, and G. Scanniello, "Assessing legacy system migration technologies through controlled experiments," in *Proc. of International Conference of Software Maintenance*. IEEE, October 2007.

[45] H. M. Sneed, "A cost model for software maintenance & evolution," in *Proc. of International Conference on Software Maintenance*. IEEE Computer Society, 2004, pp. 264–273.

[46] M. Colosimo, A. De Lucia, G. Scanniello, and G. Tortora, "Evaluating legacy system migration technologies through empirical studies," *Information & Software Technology*, vol. 51, no. 2, pp. 433–447, 2009.

[47] G. A. Di Lucca and M. Di Penta, "Integrating static and dynamic analysis to improve

the comprehension of existing web applications," in *Proc. of International Workshop on Web Site Evolution*. IEEE press, 2005, pp. 87–94.

[48] J.-F. Girard and R. Koschke, "A comparison of abstract data types and objects recovery techniques," *Sci. Comput. Program.*, vol. 36, no. 2-3, pp. 149–181, March 2000.

[49] A. Marcus, V. Rajlich, J. Buchta, M. Petrenko, and A. Sergeyev, "Static techniques for concept location in object-oriented code," in *Proc. of International Workshop on Program Comprehension*. IEEE press, 2005, pp. 33–42.

[50] L. C. Briand, Y. Labiche, and J. Leduc, "Toward the reverse engineering of UML sequence diagrams for distributed java software," *IEEE Trans. Software Eng.*, vol. 32, no. 9, pp. 642–663, 2006.

[51] U. Erra and G. Scanniello, "Towards the Visualization of Software Systems As 3D Forests: The CodeTrees Environment," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, ser. SAC '12. New York, NY, USA: ACM, 2012, pp. 981–988.

[52] G. Scanniello, M. Risi, and G. Tortora, "Architecture recovery using Latent Semantic Indexing and k-means: An empirical evaluation," in *Proc. of International Conference on Software Engineering and Formal Methods*, 2010, pp. 103–112.

[53] U. Erra, B. Frola, and V. Scarano, "BehaveRT: A GPU-based Library for Autonomous Characters," in *Proceedings of the Third International Conference on Motion in Games*, ser. MIG'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 194–205.

[54] D. B. Kirk and W.-m. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012.

[55] NVIDIA Corporation, "NVIDIA CUDA C Programming Guide," 2010, version 5.5.

[56] P. E. Hart, N. J. Nilsson, and B. Raphael, "Correction to 'A formal basis for the heuristic determination of minimum cost paths'," *SIGART Bull.*, no. 37, pp. 28–29, 1972.

[57] U. Erra and G. Caggianese, *Real-time Adaptive GPU multi-agent path planning*, GPU Computing Gems Jade Edition ed. Morgan Kaufmann Publishers Inc., 2011, vol. 2, ch. 22, pp. 295–308.

[58] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Springer, 2012.

[59] V. Basili, G. Caldiera, and D. H. Rombach, *The Goal Question Metric Paradigm, Encyclopedia of Software Engineering*. John Wiley and Sons, 1994.

[60] Y. Jiao, H. Lin, P. Balaji, and W. Feng, "Power and performance characterization of computational kernels on the GPU," in *Proc. of the International Conference on Green Computing and Communications & International Conference on Cyber, Physical and Social Computing*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 221–228.

[61] J. M. Cebrín, G. D. Guerrero, and J. M. Garcia, "Energy efficiency analysis of GPUs," in *Proc. of the International Parallel and Distributed Processing Symposium Workshops & PhD Forum*. IEEE Computer Society, 2012, pp. 1014–1022.

[62] S. Gude and P. Lago, "A survey of green it - metrics to express greenness in the it industry," VU University Amsterdam, Technical Report, 2010.

[63] V. R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, L. S. Sørumgård, and M. V. Zelkowitz, "The empirical investigation of perspective-based reading," *Empirical Software Engineering*, vol. 1, no. 2, pp. 133–164, 1996.

[64] B. Kitchenham, H. Al-Khilidar, M. Babar, M. Berry, K. Cox, J. Keung, F. Kurniawati, M. Staples, H. Zhang, and L. Zhu, "Evaluating guidelines for reporting empirical

software engineering studies," *Empirical Software Engineering*, vol. 13, no. 1, pp. 97–121, 2008.

[65] E. Rogers, *Diffusion of Innovations, 5th Edition*. Free Press, 2003. [Online]. Available: http://books.google.it/books?id=9U1K5LjUOwEC