2013 17th European Conference on Software Maintenance and Reengineering

Using the GPU to Green an Intensive and Massive Computation System

Giuseppe Scanniello, Ugo Erra, Giuseppe Caggianese
Dipartimento di Matematica e Informatica
Universitá della Basilicata, Italy
Email: {giuseppe.scanniello, ugo.erra, giuseppe.caggianese}@unibas.it

Carmine Gravino
Depart. of Management & Information Techonlogy
University of Salerno, Italy
Email: gravino@unisa.it

***Abstract*—In this paper, we present the early results of an ongoing project aimed at giving an existing software system a more eco-sustainable lease of life. We defined a strategy and a process for migrating a subject system that performs intensive and massive computation to a Graphics Processing Unit (GPU) based architecture. We validated our solutions on a software system for path finding robot simulations. An initial comparison on the energy consumption of the original system and the greened one has been also executed. The obtained results suggested that the application of our solution produced more eco-sustainable software.**

***Keywords*-Green Computing; Greening; GPU; Migration;**

I. Introduction

In recent years, the power consumption of servers, data centers, and electronic devices has become a major concern. For example, the power consumption of businesses in the USA doubled between 2000 and 2006 [1]. In order to tackle this problem and in the context of an increasing desire for eco-sustainable development, a new era for software development and evolution is being born. This era could be called the "green era". Its relevance is widely recognized in information technology and is demonstrated 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 the adapting, porting, migrating, and re-engineering of existing software systems [2]. 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 offer existing systems a new lease of life and satisfy the desire for energy reduction and environmental sustainability.

In this paper, we present the early results of an ongoing project aimed at developing a migration strategy and process to green subject systems, performing intensive and massive computation, to a target environment based on a Graphics Processing Unit (GPU[1]). A system for path finding robot simulations has been migrated to validate our solutions. The migrated system has been compared with the original one with respect to their energy consumption and execution time. We used an energy logger tool to get the magnitude of the alternating current needed. We considered both execution time and energy consumption because of the nature of the GPU: a possible reduction in the execution time could not lead to less energy consumption (e.g., [3]).

[1]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.

II. Migration and Sustainability

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 [4]. Migration does not mean installing a given system into an hardware operating environment that has been implemented using low-power techniques (e.g., multiple voltage planes or dynamic voltage-frequency scaling).

A. Benefits

Recent reports (e.g., [5]) show growing concern over the energy consumption of hardware/software systems and indicate how current trends could make energy a dominant factor in the Total Cost of Ownership (TCO). TCO also includes additional costs such as the cooling infrastructure and provisioning. Energy reduction represents one of the main drivers towards environmental sustainability. To a first approximation [6], both cooling and provisioning costs are proportional to the average energy consumed, therefore improvements in energy efficiency should reduce all the related costs. It is reasonable to assume that there will be an increasing interest from academia and industry in defining methods, techniques, and tools to migrate software systems that help to promote the construction of an ecologically 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, *(v)* increasing the performance of new systems, and *(vi)* promoting a new business engine for software companies.

III. Migration Strategy and Process

A. Migration Strategy

The first step to migrate a software system is to assess it from both the technical and the managerial perspectives.

1534-5351/13 $26.00 © 2013 IEEE
DOI 10.1109/CSMR.2013.55

Successively, the target environment has to be chosen and the possible problems and risks have to be assessed [2]. We instantiated this strategy as follows:

1) Assessing the Current System: We analyzed the subject software system, which has been developed at the University of Basilicata within a regional research project conducted in cooperation between the Department of Computer Science and the Department of Telecommunication Engineering.

The software system regards a massive robot simulation in the path planning of thousand of robots with respect to random obstacles in any two dimensional environments. The system enables to identify all potential obstacles in order to find a suitable path for all robots given a set of start positions. The objective of this project is 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 renewed this system because additional regional funds were received. We were asked to focus on the following goals: *(i)* increasing the parallelization, *(ii)* improving the performances, and *(iii)* reducing energy waste.

To migrate this system, we used all the available documentation and resources (e.g., use cases, source code, fault history, operational profile, and developers' knowledge). 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 components of the system.

The assessment of the system revealed that it had a good level of decomposability. For example, the presentation logic was separated from the application logic and data access logic. 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.

2) Defining the Target Environment: 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 [7]. This was the rationale for defining and using a target environment based on integrated GPUs.

GPUs are programmed through a serial program called kernel that executes in parallel a set of threads each with a private local memory. The developer organizes these threads into a hierarchy of thread blocks and grids. A thread block is a set of concurrent threads that can cooperate among themselves and have access to the shared memory. The grid is a set of thread blocks that can be executed independently. The threads have access to the same global memory. The three kinds of memory have different time access and then can be used for different purposes. The GPUs can be programmed using a language such as CUDA [8]. A CUDA program consists of several phases that are executed on either the host (i.e., the CPU) or a device (i.e., the GPU). Host code exhibits little or no data parallelism while the device code exhibits rich amount of data 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 the host's 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.

3) Identifying Problems and Risks: In our case, the most relevant risks are concerned to the effective improvement in terms of energy efficiency and computational performances. For example, it could be possible that the migrated system does not meet the expected migration goals. Other issues could be related to the limited understanding of the system, impact analysis, testing, and identification of the target environment. The complexity of the GPU architecture represents another risk. In this context, coding needs in-depth knowledge of resources available in terms of thread and memory hierarchy.

B. Migration Process

Our migration process is based on the following phases:

1) *Reverse Engineering*. In this phase suitable representations of a subject system are produced. This phase has also the effect of increasing the comprehension of the system and source code, in particular.
2) *Reengineering*. Components that perform computationally intensive tasks are identified and reengineered.
3) *Integration*. The newly developed components are wrapped and integrated within the migrated system.
4) *Testing*. The migrated system is tested.

These steps represent the baseline for a migration process (e.g., [2]), while their instantiation is new here. In the following subsections, we discuss the steps 2 and 3.

1) Reengineering: The component that performed intensive computation was the one implemented the A* search algorithm [9]. The input of this algorithm is: a set of robots' starting positions, a goal position, and moving costs. 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 computational 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 parallelize the execution using 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.

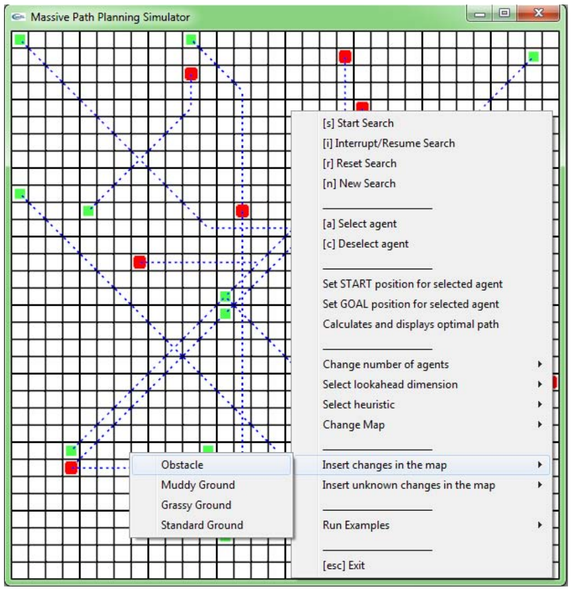
2) Integration: In the GPU parallel pathfinding, a robot is a CUDA thread. In our case, the kernels are four. One kernel is in a charge of executing the A* algorithm. Other two kernels are for the initializations needed for the algorithm execution. The latter kernel concerns the output to be visualized in the graphical user interface.

Figure 1 shows an excerpt of the source code for invoking the implementation for GPU of the A* algorithm. The statements from 2 to 5 are in charge of allocating the memory for the CPU, while those from from 8 to 15 are for the allocation of 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 is in charge of moving the output of that algorithm from the GPU to the CPU.

The input is a grid map that represents the search space and includes the start and goal positions of each robot in the simulation. 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 the lines the paths. The graphical user interface is the same for both the original and migrated systems.

```
 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);
```

Figure 1. Source code to invoke the greened component.



Massive Path Planning Simulator
[s] Start Search
[i] Interrupt/Resume Search
[r] Reset Search
[n] New Search
[a] Select agent
[c] Deselect agent
Set START position for selected agent
Set GOAL position for selected agent
Calculates and displays optimal path
Change number of agents
Select lookahead dimension
Select heuristic
Change Map
Insert changes in the map
Insert unknown changes in the map
Run Examples
[esc] Exit
Obstacle
Muddy Ground
Grassy Ground
Standard Ground

Figure 2. A screenshot of the system.

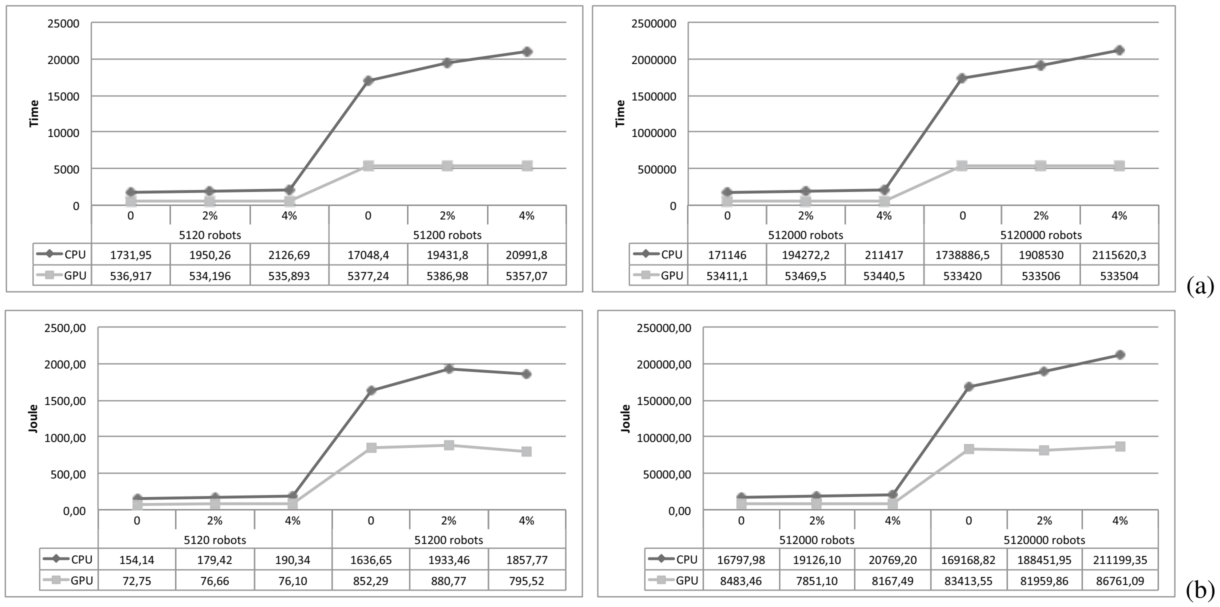
IV. Assessment

A. Definition and Context

The main goal of this investigation is quantitatively evaluating the benefit deriving from the application of the proposed migration strategy in terms of execution time and reduction of the energy consumption. We considered both execution time and energy reduction because they could be not directly related [3]. The comparison was performed over 12 different input configurations on the same grid map of 256×256 tiles. Each configuration was constituted of a fixed number of robots and of a fixed number of obstacles to be avoided by each robot. The number of robots were: 5120, 51200, 512000, and 5120000. On the other hand, the number of obstacles was 0, 2%, and 4% of the number of tiles.

B. Data Collection and Analysis

The 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 migrated systems on each input configuration chosen, we instrumented the source code of both these



	5120 robots			51200 robots		
	0	2%	4%	0	2%	4%
CPU	1731,95	1950,26	2126,69	17048,4	19431,8	20991,8
GPU	536,917	534,196	535,893	5377,24	5386,98	5357,07

	512000 robots			5120000 robots		
	0	2%	4%	0	2%	4%
CPU	171146	194272,2	211417	1738886,5	1908530	2115620,3
GPU	53411,1	53469,5	53440,5	533420	533506	533504

	5120 robots			51200 robots		
	0	2%	4%	0	2%	4%
CPU	154,14	179,42	190,34	1636,65	1933,46	1857,77
GPU	72,75	76,66	76,10	852,29	880,77	795,52

	512000 robots			5120000 robots		
	0	2%	4%	0	2%	4%
CPU	16797,98	19126,10	20769,20	169168,82	188451,95	211199,35
GPU	8483,46	7851,10	8167,49	83413,55	81959,86	86761,09

Figure 3. Graphics for (a) execution time (in seconds) and (b) energy consumption (in Joule)

systems. The computation time was measured in seconds. To measure the total energy consumption in Joule, we used the EcoDHOME MCEE USB energy logger. We employed an energy logger because there are not accepted metrics to estimate the consumption of GPU based implementations.

C. Results

Figure 3 shows the results achieved in terms of execution time and energy consumption for all the input configurations. The execution time of the greened system is always lower (see Figure 3.a) than the original one. Moreover, the differences in the execution time is more evident when the number of robots increases. It is also clear that the number of obstacles affects the execution time only in the case of the original system, whatever is the number of robots.

As for energy consumption, the migrated system needed less Joule than the original one (see Figure 3.b). This is true for all the input configurations. As for the execution time, the number of obstacles slightly affect the energy consumption of the greened system. Differently, the energy consumption of the original system is affected by the number of obstacles.

V. Conclusion

In this paper, we have presented a strategy and a process to migrate and existing system to a target environment based on the GPU. We validated our solutions on a system for massive robot simulations. The migrated system has been compared with the original one through a preliminary quantitative evaluation. The results indicated that the energy consumption of the greened system is lower and then more eco-sustainable.

References

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

[2] 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.

[3] 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.

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

[5] J. G. Koomey, "Estimating total power consumption by servers in the u.s. and the world," 2007.

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

[7] M. Rofouei, T. Stathopoulos, S. Ryffel, W. Kaiser, and M. Sarrafzadeh, "Energy-aware high performance computing with graphic processing units," in *HotPower*. USENIX Association, 2008, pp. 11–11.

[8] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," *Queue*, vol. 6, no. 2, pp. 40–53, 2008.

[9] 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.