Nuno Lopes
Assistant Professor
Polytechnic Institute of Cávado and Ave
Barcelos
Algoritmi Research Centre
Guimarães
Portugal

Goran Putnik
Full Professor
University of Minho
Algoritmi Research Centre
Guimarães
Portugal

Luís Ferreira
Assistant Professor
Polytechnic Institute of Cávado and Ave
Barcelos
Algoritmi Research Centre
Guimarães
Portugal

Bruno Costa
Polytechnic Institute of Cávado and Ave
Barcelos
Portugal

Towards a High Performance Computing Scalable Implementation of Cyber Physical Systems

Cyber Physical Systems (CPS) establish an interdependency between the physical world and the cyber world. When considering a real world setup, CPS will receive a considerable input from the physical world that demands a timely response. Without the computer capacity to handle the input data, a CPS will not be able to react in a very short term to changes in the environment and provide proper output. Current CPS are unable to react in near real-time to challenges when considering a realistic problem size. Our objective is to harness the performance of High Performance Computing systems to make CPS capable of handling the necessary computation demands in order to be able to interact with the physical world. We present a case study on scheduling algorithms to demonstrate the current limitations in the computing performance and point possible solutions to achieve it.

Keywords: *Cyber Physical System (CPS), High Performance Computing (HPC), scheduling algorithm, greedy algorithms, parallel programing, real-time CPS.*

1. INTRODUCTION

The general literature considers Cyber-Physical Systems (CPS) or, better in the context of this paper, Cyber-Physical Production Systems (CPPS), as the capacity to state an efficient interaction and collaboration between physical (machines, sensors, actuators, etc.) and digital (software, control, monitoring decision systems, etc.) worlds [1,2].

The path to full digitalization with Internet of Things and Industry 4.0 force, the concept to be realigned since new processing capacities arise, new participating elements exist and new requirements of integration emerge. The common triad controller-sensor-actuator [2] will behave as a mix of computational and physical parts [3], able to be changed or redesigned, during learning processes. The productions units (machines, actuators, etc.) should continue to be reconfigured and the processes agilely rescheduled [4]. Furthermore, the software-based control units, should be prepared to be context-ware refactored, according to accuracies faults, uncertainty or other disturbances.

The shop-floor is a very dynamic environment where an optimal static configuration setup becomes obsolete in face of external disturbances, which for some cases happen permanently [5]. This feature requires an CPPS that reacts in real-time to the system changes and adapts the system configuration to pursue the best possible setup [6,7].

Scheduling algorithms are an application example of CPPS that are well explored in the literature and being known as NP-hard. NP-hard problems have the characteristic that, as the problem size grows, the computation is required to find an optimal solution that grows exponentially. As such, this complexity implies that the execution time does not drop significantly even when increasing significantly the computing resources [8].

A CPPS system should support real world setups with a large number of both jobs and machines, in order to have practical applicability. Therefore, the CPPS should be scalable on the problem size, i.e., it should remain performant by producing valid scheduling results, in a useful time interval (as close to real-time as possible), as the problem size increases.

The traditional approach to overcome this problem complexity is to use heuristics. Heuristics reduce the number of calculations needed to achieve a valid solution to the problem [8,9]. However, they do not guarantee that the optimal solution to the problem is found, but instead that a valid solution is found even if not the optimal one. By reducing the number of calculations, the time the algorithm takes to complete will also decrease.

Even when using heuristics to reduce the number of computations, the minimum number of computations necessary to handle a realistic problem size still remains large enough to prevent the system to reply in real-time. This leads to simulations with relatively small number of machines or jobs, because of the algorithm complexity.

In order to obtain performance scalability, i.e., to maintain the time necessary for the algorithm to reach a valid solution when the problem size increases, two approaches are possible: to reduce the number of computations or to increase the computing capacity per time unit. The first approach reduces the execution time of the algorithm by running fewer computations. This is the case of heuristics. Nevertheless, heuristics still

Received: November 2018, Accepted: June 2019
Correspondence to: Dr Nuno Lopes
Polytechnic Institute of Cávado and Ave.
School of Technology, Barcelos – Portugal.
Email: nlopes@ipca.pt
doi: doi:10.5937/fmet1904749L
© Faculty of Mechanical Engineering, Belgrade. All rights reserved

exhibit a significant amount of computations that limit scalability. Hence, the second approach needs to be adopted, i.e., being able to execute more computations on the same amount of time.

According to Putnik et al., the approach of running more computations within the same amount of time can be obtained by: 1) upgrading the capacity of existing resources or 2) incrementing the number of resources (replication) [10]. In the CPPS case, this is equivalent to upgrade existing processors, in order to increase its computing capacity, or to increase the number of computers. The first case (from Moore's Law) is nowadays no longer observable when considering the increase of processor frequency. Therefore, the only scalability path available is the increase in the number of processors (and computers).

We propose the use of High Performance Computing (HPC) to increase the computation capacity and reduce the execution time, aiming at obtaining a CPPS implementation capable of achieving a problem solution as close to real-time as possible.

The structure of this paper is as follows. Section 2 presents an overview on High Performance Computing. Section 3 presents a literature review on HPC applications for Industry 4.0. Section 4 presents an application performance assessment to improve parallelization. Section 5 shows a case study for a scheduling simulation implementation and its performance evaluation. Section 6 presents a framework for a HPC scalable implementation of a CPPS system and finally section 7 concludes the paper.

2. HIGH PERFORMANCE COMPUTING

High performance computing can be defined as the capacity to run a computer program that is efficiently executed on a parallel computer [11]. A parallel computer is a computer capable of executing multiple instructions at the same time through the support of multiple processors or cores. According to Flynn's Taxonomy, this computer falls into the Multiple-Instruction-Multiple-Data type, i.e., multiple instructions are executed concurrently (on different processors or cores) operating on different data items.

Parallel computers use a shared-memory or a distributed-memory approach for the processor-main memory communication architecture (see Figure 1). The shared-memory computer has multiple processors, or multiple cores within a single processor chip, that connect into a single shared main memory. The distributed-memory computer has multiple pairs of processor-memory, that are independent from each other, and communicate exclusively through a message passing network.

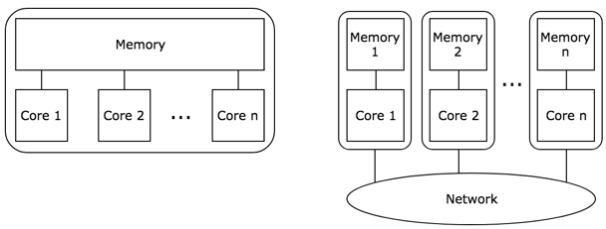

Figure 1. Shared-memory (left) versus distributed-memory (right) computers.

Shared-memory computers have multiple processors accessing a single shared main memory. Due to hardware optimizations, the traditional approach for shared-memory is to use a cache-coherent Nonuniform Memory Access (ccNUMA). In this design, the non-uniform memory access implies that memory is not uniformly accessed by all processors, but instead, each processor accesses a part of the physical memory, and an internal connection bus between processors shares the memory logical parts among each other. Hence, the access of a memory part that is locally connected into a processor will be faster than accessing a memory part that is connected into another processor. The use of multiple logical memory parts, together with cache techniques to improve performance raise a consistency requirement when two processors try to access the same memory address or position. Cache coherent techniques assure that all processors view a consistent memory data.

On the distributed memory computer, each element contains a processor and main memory pair that is independent from all others. The communication between processors is made though a communication network that exchanges (read-only) messages, i.e., the contents of messages are not altered after the message being sent even if the data is updated on the sending node's memory afterwards. Messages are explicitly sent and received through an API. Although this model differs from the shared-memory model, it is possible to use the distributed-memory model on a shared-memory computer, and vice-versa.

A common supercomputer architecture nowadays consists of clusters of commodity processors [11]. This architecture makes use of multiple core individual computers, which are alone capable of executing concurrent instructions. However, the total computing capacity of a single individual computer is not sufficient to harness the computing capacity demand for large computational applications. The solution adopted for increasing performance was to scale on the number of individual computers, interconnecting them through the use of a low latency high bandwidth network. In sum, the final architecture is composed of multi-core computers using shared-memory model that are interconnected through a message-passing distributed-memory model.

3. HPC APPLICATIONS FOR INDUSTRY4.0: LITERATURE REVIEW

The use of HPC in industry is not new. Recently the Fortissimo EU project has established a marketplace for running HPC based research projects for industry [12]. However, most of the projects and solutions presented fall into the simulation approach whose execution time falls outside the real-time time frame, being in the order of minute, hours or days. Nevertheless, the HPC solution is offered as a cloud-based service, where customers pay for the time use of the HPC infra-structure.

Bozejko presents the use of HPC through massive parallelism to solve the job-shop problem through metaheuristics [13]. The results show that the parallel algorithms exhibit a linear speedup with the number of processors (although some considerations are made). This work shows that well designed parallelization algorithms

can exhibit performance gains and justify the parallelization use. Nevertheless, the use of meta-heuristics continue to impose a high number of calculations.

A more recent work of Dabah et al. also propose the use of HPC with multiple cores and general purpose graphical processing units (GPGPUS) to improve the execution time of a job-shop scheduling problem using optimization algorithms [12]. Their best results exhibit a speedup of 160x compared to the sequential execution time, for a simulation problem size of 100 jobs and 20 machines that takes 418 seconds. While the speedup is considerable, again, the global execution time is in the order of 418 secs.

Kan et al. proposed the use of parallel computing for improving the performance of a monitoring system that reacts to changes in the machine behaviour [14]. The parallelism used 12 nodes, with a speedup of 3x the time of the sequential counterpart, for a 180.000 machine network simulation.

Zhong and Xu propose a job-shop scheduling model that captures the real-time feedback from the environment to improve the decision making [7]. The simulation considers a setup of 48 jobs and the algorithm applied does not produce an optimum solution. This work presents the case for using real-time feedback, a feature of a CPPS, to improve the production scheduling output.

In summary, the works that make use of HPC target scheduling optimization algorithms that are computing intensive and take too much time (in the order of hundreds of seconds) to reach a solution for a reasonable problem size. In turn, other works that do not use HPC rely on simple heuristics to provide some kind of feedback in order to improve the existing scheduling of a very small problem size. The problem sizes considered in the previous works are not representative of a real setup scenario in the context of Industry 4.0, where we expect to have thousands if not millions of machines and the same amount of jobs respectively.

4. APPLICATION PERFORMANCE ASSESSMENT: TOWARDS PARALLELIZATION

To improve the overall efficiency of an application, the goal is to reduce its execution time while maintaining its correction. The correction of an application can be defined as the ability to produce a correct output result for any input, regardless of the implementation details. Starting from a sequential implementation, it is expected that a parallel version will be able to produce a correct output but using less execution time. Another aspect of the execution is determinism. In determinism, it is expected for the application to produce always the same output from the same input. Sometimes, parallel algorithm implementations do not assure determinism. Nevertheless, this property can be skipped if all the possible output results are considered correct.

4.1 Application Parallelization

According to the Foster's Methodology [15], there are four steps that can be taken to design a parallel application: Partitioning, Communication, Aggregation and Mapping. Partitioning is about dividing the computation algorithm in smaller computation tasks so that it is possible to identify parallelization opportunities, i.e., more than one task than can be executed at the same time. Communication deals with the coordination of task executions and auxiliary data that needs to be exchanged among the previous tasks in order to conclude the computation. Aggregation is an optimization step where some tasks are grouped into a single composite task due to data dependencies, or computation load to improve performance. Mapping is the final step where tasks are assigned to logical processors in order to load balance resource usage and reduce overall communication costs.

When applying this methodology, one has to identify what is the computational problem to be analysed, and if a previous sequential implementation exists. When starting from an already existing sequential application, the parallelization approach must first identify the critical hotspots that are present in the sequential implementation. Critical hotspots are pieces of code that take a considerable amount of time to execute, which makes them good candidates to be parallelized and hence, to have its execution time reduced. By targeting specific parts of the sequential implementation, one considers the computational problem to be only specific pieces of the sequential implementation, the hotspots, and not necessarily the whole sequential application.

4.2 Application Performance Assessment

To assess the hotspots of an application, an application profiler is necessary. The purpose of this tool is to run the application with additional monitoring to trace its internal behaviour so that an execution count and elapsed time spent at each internal code piece (functions) can be collected and analysed further with a global function call graph.

Multiple profiler tools exist that are capable of collecting such information, both commercial and open-source based [16-18]. It is possible to trace the execution time of the inner functions and have a global overview where computing time is spent. Some tools perform a statistical approximation for the execution time, therefore an error exists in the trace result. Nevertheless, these results allow the identification of pieces of code (functions) that for a significative large execution profile are hotspots.

5. CASE STUDY: SCHEDULING SIMULATION ALGORITHM

A scheduling simulation algorithm was selected as a case study for the implementation of a Cyber Physical Production System with the objective of producing a correct output for a real size domain problem input. The scheduling problem has as its inputs a set of jobs and a set of machines. Each job requires a small set of operations which may run on some specific machines. This scheduling problem is generically named Job Shop.

5.1 Case Study Implementation Details

The algorithm implemented allocates job operations into machines using a variant of a "greedy" algorithm which

selects the best machine based only on local knowledge at the moment, that may not necessarily be the best possible machine among all. This approach skips the use of any classical optimization solution on the output schedule [5]. Although not optimal, the algorithm result is close enough to be considered a good solution, while taking much less time to compute.

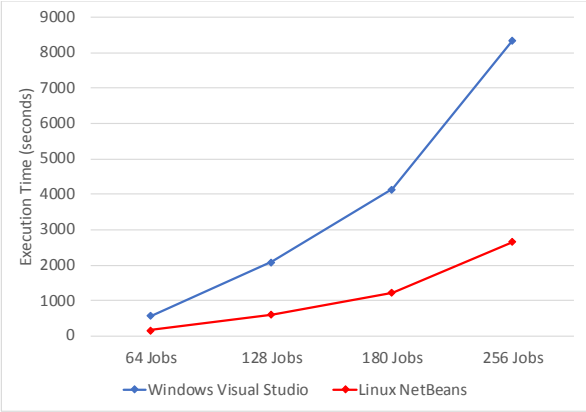

Figure 2. Execution time for a scheduling simulation of 64 Machines with different job sizes (64, 128, 180 and 256).

The algorithm implementation was developed using the NetBeans IDE and compiled using the sequential C++ programming language. Two auxiliary libraries were used to support file and directory management (MinGW) and the JSON file format used to store simulation data into persistent files (RapidJSON). The original implementation runs on Windows OS, but a later implementation was ported to both Windows and Linux. The porting did not changed any algorithm specific code, but only platform dependent code, i.e., the algorithm code (C++) remains identical for both platforms. The application uses a command line textual interface, which has a very low impact on computing resource usage during execution. The porting of the application for a second operating system was motivated for the analysis of the OS impact in the performance of the application, which in theory should be null.

The problem size used in the simulation ranged from a reference data set of 8 machines x 8 jobs (with 7 operations/job) up to 64 machines x 256 jobs, which were obtained by scaling the original data set. The Figure 2 shows the execution time for a scheduling simulation of 64 machines with multiple job sizes (64, 128, 180 and 256) for the two platforms on a machine with an Intel Core i7 CPU up to 3,5 GHz, 32 GBytes of RAM, and storage of 512 GBytes SSD and 1 TByte HDD.

The results show that the execution time grows exponentially with the job size, showing that the algorithm complexity is not negligible, although based on a heuristic and hence theoretically with low complexity. Additionally, the results also show a considerable difference between operating system platforms, which we believe to be related with compiler optimizations and library implementations, attending that the algorithm in itself was not modified in its source code form.

5.2 Hotspot Analysis

The simulation application was profiled using the Intel Vtune Amplifier v2019 in order to trace potential hotspots [16]. This simulation used a problem size of 64 machines with 64 jobs on a Linux OS with the debug option activated, which took 879 secs to execute. The execution time of the application in debug mode within the profiler has a considerable overhead when compared to the release mode, which would be 170 seconds. However, the use of the debug mode is a requirement to run the profiler and trace the execution time spent within the application. The results are shown in Figures 3 and 4.

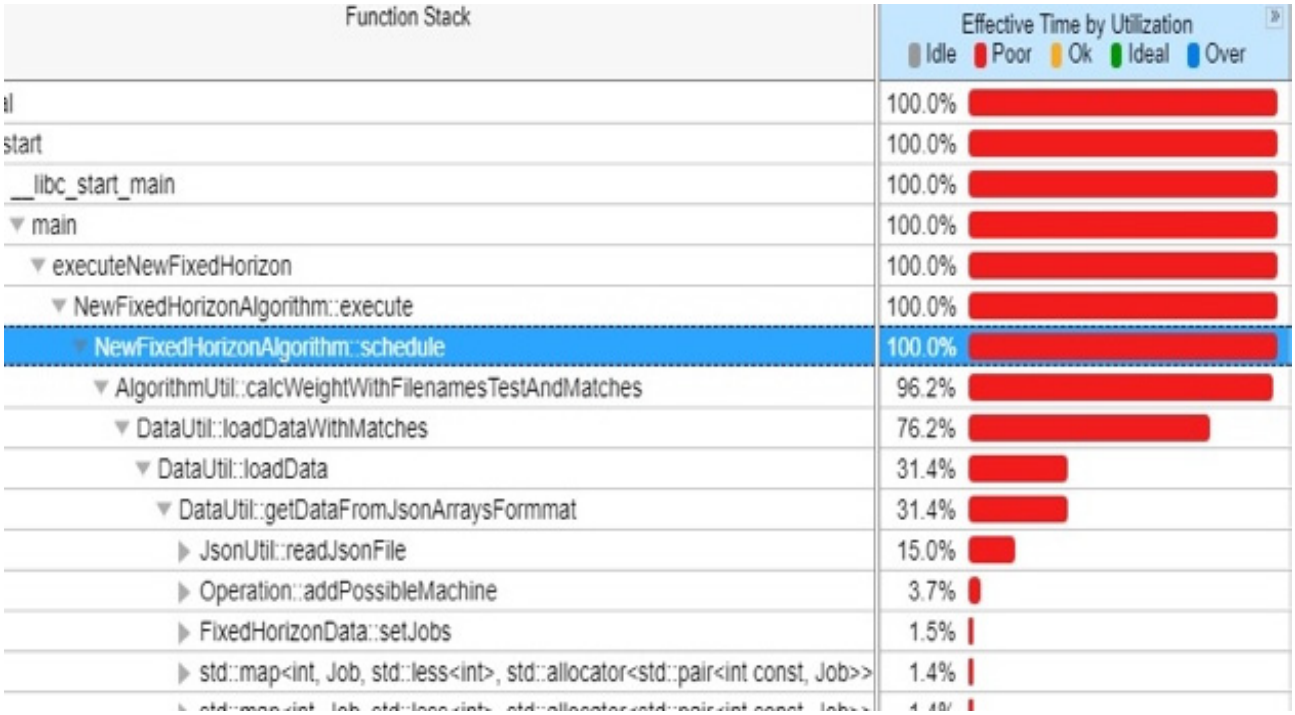

Figure 3. Top-Down Call Graph with relative execution time.

Figure 4. Bottom-Up Call Graph with absolute execution time

The Figure 3 shows the Top-Down Call Graph of the application where the first column shows the function stack and the second column shows the accumulated percentage of the global execution time spent on the function. The `main()` function, which encompasses all the application functionality, accounts for 100% of the execution time. As expected the application is running the schedule algorithm at 100% of the total time, which means that any other auxiliary code accounts for a very small percentage of time. Within the scheduling, we observe that 76% of the total time accounts for "load DataWithMatches" that is where data is loaded from auxiliary files from disk. Within this, 15% of time is spent in reading data from files using the JSON library and an additional 16.4% is spent in auxiliary data processing from the JSON library into internal data structures. Overall, 31.4% of the total time is spent in loading and processing data, which is a considerable amount of time devoted to loading data rather than computing the scheduling.

The Figure 4 shows the Bottom-Up Call Graph with the absolute execution time spent within each function. The total time of the application was around 800 secs. The two functions that take the most execution time are the constructor and destructor of the Map auxiliary data structure and the new object constructor.

5.3 Results Analysis

The first hotspot found is the significative time spent in creating and destroying auxiliary data structures (map) while calculating the schedule. Overall, the application spends about half of its global execution time within these auxiliary functions (the sum of the map constructor, destructor, new object operator and the new allocator function account for more than 400 secs). The map data structure is an auxiliary data structure needed for the calculation of the output result, and presents a parallelization opportunity.

The scheduling algorithm in itself, after the auxiliary data structures are built in place, does not account for any significative computing load. This is due to the "greedy" heuristic nature of the algorithm, with very low complexity on the problem size.

The hotspot analysis also revealed that the auxiliary JSON library took 15% of the total application time, which is not efficient at handling the amount of data that is temporarily stored in disk for supporting the planning of the scheduling. This was an unexpected hotspot, considering that this application is cpu intensive, making use of the cpu for creating a schedule as its only major operation. The disk access is made through a single bus, hence it is not easily parallelizable as parallel requests to the disk would have to pass through the same single hardware path. The handling of temporary data has to be designed in a different way, either through a different file format other than JSON that optimizes the disk access and processing (both tasks account for 30% of the execution time), or trying to avoid storing data in disk and keep all data in main memory if possible (depends on the amount of RAM available).

5.4 Identifying Requirements for HPC

The initial results for the simulator reveal that taking 2653 secs (about 44 minutes) for producing a valid scheduling solution for a problem size of about 64 machines and 256 jobs is not feasible to reach a near real-time execution. We expect that with the use of a parallel algorithm implementation deployed on an HPC computer that the execution time will decrease to a near real-time range.

A possible solution to overcome the hotspot found in the management of the auxiliary data structures, necessary for the algorithm, is to redesign the object allocation strategy, in order to incorporate a parallel approach through the Foster methodology. The first step is to partition the creation of the data structures through multiple smaller tasks so that parallelization is possible. In this case, multiple threads of execution create local data structures that have to be integrated into a single global data structure (on the shared-memory model). Since each object constructor is not computing intensive by itself, the hotspot is created by calling the constructor multiple times. This execution time can therefore be split through multiple execution processors and hence decrease the execution time spent within these functions. The communication step consists in the integration of the memory references of multiple local auxiliary data structures (variable references) into the global data structure. The aggregation and mapping steps might group some constructors into the same logical group for performance since there will be more auxiliary objects than processors.

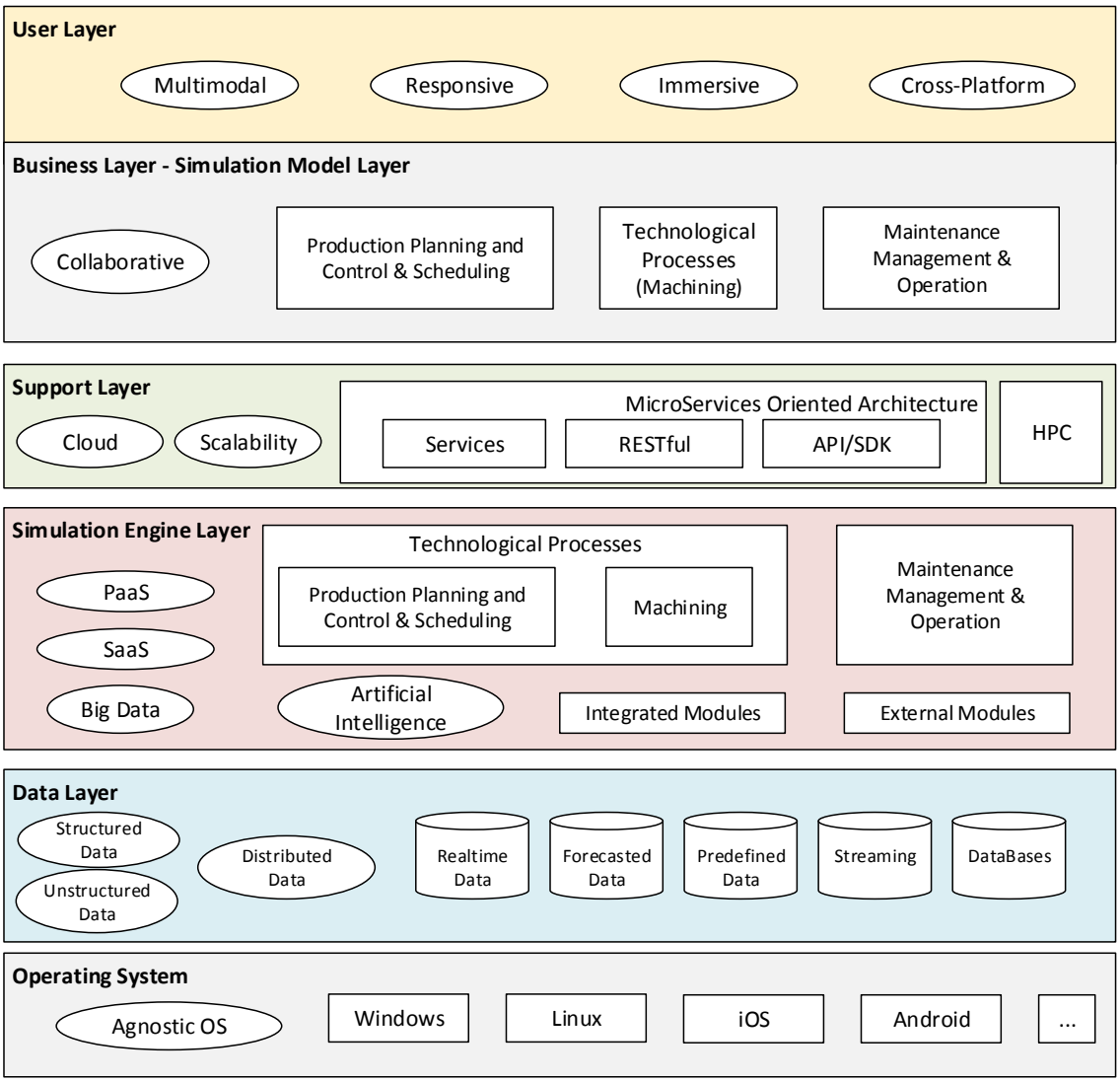

Figure 5. CPPS supporting platform stack of technologies and methods, HPC is present in the Support Layer.

6. FRAMEWORK PROPOSAL FOR HPC SCALABLE IMPLEMENTATION FOR CPPS

To have a Cyber Physical Production System be responsive in almost real-time, and considering the need for running scheduling simulations, a proposal to integrate HPC into the CPPS framework is presented next. The previous experiences with "traditional" approach to simulations, based on common PC technology, meaning, single or dual processor with at most 16 cores, shows that for large problems, the simulations lasted too long, from several hours (for smaller cases) to several days (for larger cases). The simulation times experienced, even if much lesser than few hours, are, in principle, not acceptable for simulations in CPS in which it is supposed to have the capacity of response in, virtually, real-time. That is, the capacity to provide operation of the system and associated decision making, which includes simulation as one of the decision making instruments, to be realised on the "flow". The inclusion of HPC technology emerges as necessary within a CPPS framework.

The case study presented identifies which of the algorithm components that can benefit from the parallel programing approach, open the possiblity for the algorithm to run on large HPC computers.

We propose the following components, technologies and methods for a CPPS framework in Figure 5. The framework follows the N-Tier pattern architecture with 6 main tiers of (1) User Layer, (2) Business Layer (Simulation Model Layer), (3) Support Layer, (4) Simulation Engine Layer, (5) Data Layer, and (6) Operating System. Each "Layer" comprises two different types of components, represented graphically by two different graphical symbols: elliptical, and rectangular, which corresponds to the layer's characteristics or particularities and to the layer's components, respectively:

1) *User Layer*: represents applications and support for all interfaces, views, presentations and communications for users; Immersive environments of Mixed Reality will offer more real predictive and simulations experiences.
2) *Business Layer - Simulation Model Layer:* represents applications and support for all main CPPS phases, including Machining and Production Control, as well as Management, all in co-designing collaborative and creative environment;
3) *Support Layer:* represents cloud applications and support for all cross-platform applications; Sustains the

required scalability and **High Performance Computing** for the required real-time feature in CPPS.

4) *Simulation Engine Layer:* represents applications and support for all simulation modules for Technological Processes, Management and Maintenance Control;
5) *Data Layer:* represents applications and support for all applications for data repository and management, including real-time data and streaming data coming from external systems;
6) *Operating System:* represents the portability and scalability of operating system that supports all the components.

7. CONCLUSION

Cyber Physical Production Systems (CPPS) in the context of Industry 4.0 will have to process a tremendous amount of data in a continuous stream for dealing with permanent environment disturbances that create an impact on the scheduling decisions already made. Hence CPPS are required to respond in (near) real-time to these changes by reconfiguring decisions. We present the use case of a scheduling algorithm that must face continuous rescheduling for a realistic problem size in face of external disturbances.

Although the real-time requirement of CPPS is already identified in the literature, all CPPS examples present a relatively small problem size that ranges up to 50 machines. This problem size is small when compared to the expected size of future production systems, in the context of I4.0, where the number of machines and number of jobs might scale up to the thousands, theoretically up to millions. The HPC literature surveyed considers 20 machines and 100 jobs, which again is less than what to be expected from a real world Industry 4.0 setup.

We identify some scalability limitations on our case study that must be overcome to obtain a responsive CPPS, capable of adapting to continuous disturbances by presenting a rescheduling at each change. We propose to use HPC technology to support the scalable implementation of a scheduling algorithm, so that a rescheduling is obtained in a very short period of time. We propose to use a greedy algorithm that very quickly points a good (but not necessarily optimal) solution. The HPC back-end will facilitate the production of a reschedule in a timely manner for realistic problem sizes, larger than the ones observed in the current literature. Although possible, the use of cloud based solutions to support the virtualization of the infrastructure is not in itself a sufficient solution since the scalability of the system depends on the efficient use of multiple machines, which can only be obtained through HPC technologies, provided either as a standalone or cloud service solutions. We study the weak points in our current implementation and point out some possible improvements to reach an algorithm that scales out on an HPC infrastructure.

ACKNOWLEDGMENT

This work has been supported by FCT – Fundação para a Ciência e Tecnologia, Portugal, within the Project Scope: UID/CEC/00319/2019.

REFERENCES

[1] Monostori, Kádár, Bauernhansl, Kondoh, Kumara, Reinhart, Sauer, Schuh, Sihn and Ueda, "Cyber-physical systems in manufacturing," *CIRP Annals - Manufacturing Technology,* vol. 65, pp. 621-641, 2016.

[2] Tomiyama and Moyen, Resilient architecture for cyber-physical production systems, *CIRP Annals - Manufacturing Technology,* vol. 67, 161-164, 2018.

[3] Putnik, G. D., Ferreira, L., Lopes, N., Putnik, Z.: What is Cyber-Physical System: Definitions and Models Spectrum, in *FME Transactions*. Vol. 47 No. 4, pp. 663-674, 2019.

[4] Petrovic, Milica, Miljkovic, Zoran, and Babic, Bojan: Integration of process planning, scheduling, and mobile robot navigation based on triz and multi-agent methodology, in *FME Transactions*, vol. 41, no. 2, pp. 120-129, 2013.

[5] C. Alves, Modelling and Evaluation of "Fixed Horizon", "Rolling Horizon" and "Real Time Management" Production Scheduling Paradigms in Ubiquitous Production Networks under Conditions of Dynamic Environments for Economic and Environmental Sustainability, PhD Thesis, University of Minho, 2017.

[6] Mourtzis, Vlachou, Milas and Xanthopoulos, "A cloud-based cyber-physical system for adaptive shop-floor scheduling and condition-based maintenance," *Journal of Manufacturing Systems,* vol. 47, pp. 179-198, 2018.

[7] Zhong and Xu, "A Job-Shop Scheduling Model with Real-time Feedback for Physical Internet-based Manufacturing Shopfloor," in *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*, Beijing, China, 2015.

[8] Garey and Johnson, Computers and Intractability, W. H. Freeman, 1979.

[9] Sathish, Jayaprakash and Saravanan: Multi period disassembly-to-order of end of life product based on scheduling to maximize the profit in reverse logistic operation, in *FME Transactions*, vol. 45, no. 1, pp. 172-180, 2017.

[10] G. Putnik, A. Sluga, H. ElMaraghy, R. Teti, Y. Koren, T. Tolio and B. Hon, "Scalability in manufacturing systems design and operation: State-of-the-art and future developments roadmap," *CIRP Annals - Manufacturing Technology,* vol. 62, pp. 751-774, 2013.

[11] G. Hager and G. Wellein, Introduction to High Performance Computing for Scientists and Engineers, CRC Press, 2011.

[12] A. Dabah, A. Bendjoudi, A. AitZai, D. El-Baz and N. N. Taboudjemat, "Hybrid multi-core CPU and GPU-based B&B approaches for the blocking job shop scheduling problem," *Journal of Parallel and Distributed Computing,* vol. 117, pp. 73-86, 2018.

[13] W. Bożejko, "Solving the flow shop problem by parallel programming," *Journal of Parallel and Distributed Computing,* vol. 69, pp

[14] C. Kan, H. Yang and S. Kumara, " Parallel computing and network analytics for fast Industrial Internet-of-Things (IIoT) machine information processing and condition monitoring," *Journal of Manufacturing Systems,* vol. 46, pp. 282-293, 2018.. 470-481, 2009

[15] P. Pacheco, An Introduction to Parallel Programming, Morgan Kaufmann, 2011.

[16] Intel, "Intel Vtune Amplifier," 2019. [Online]. Available: https://software.intel.com/en-us/vtune.

[17] GNU Software Foundation, "Gprof Documentation," 2019. [Online]. Available: https://sourceware.org/binutils/docs/gprof/.

[18] Valgrind Developers, "Valgrind - Callgrind," 2019. [Online]. Available: http://valgrind.org/info/tools.html#callgrind.

ПРЕМА СКАЛАБИЛНОЈ ИМПЛЕМЕНТАЦИЈИ САЈБЕР-ФИЗИЧКИХ СИСТЕМА (СФС) НА ОСНОВУ РАЧУНАРСКИХ СИСТЕМА ВИСОКИХ ПЕРФОРМАНСИ

Н. Лопеш, Г.Д. Путник, Л. Фереира, Б. Кошта

Сајбер-Физичких Система (СФС) успоставља међузависност физичког света и сајбер света. При разматрању старног света, СФС ће добити значајне улазне податке из физичког света који захтева правовремени одговор. Без компјутерских капацитета за обраду улазних података, СФС неће бити у могућности да реагује у кратком року на промене у окружењу и обезбеди одговарајуће излазне податке. Садашњи СФС нису у стању да реагује у реалном времену на изазове приликом разматрања проблема реалних обима. Циљ је да искористимо перформансе рачунарских система високих перформанси како бисмо учинили СФС способним да одговори потребним рачунарским захтевима како би били у могућности да оперишу у интеракцију са физичким светом. Представљена је студија случаја о алгоритмима програмирања производње како би се показала тренутна ограничења у перформансама рачунања и указали на могућа решења да би се циљеви постигли.