AFFORDABLE COMPUTATION FOR ARCHITECTURE

ANTÓNIO LEITÃO, RENATA CASTELO-BRANCO AND
INÊS CAETANO
INESC-ID/Instituto Superior Técnico, University of Lisbon
{antonio.menezes.leitao|renata.castelo.branco|ines.caetano}@ist.utl.pt

Abstract. *Current architectural requirements prioritize the need to minimize the ecological footprint. By taking advantage of computational design approaches like Algorithmic Design (AD), architects can enhance their design processes with analysis, optimization, and visualization mechanisms, which are critical to explore design solutions that meet this need. However, these mechanisms are also highly time- and resource-consuming, often implying a quality tradeoff or the acquisition of High-Performance Computing (HPC) machines. The latter are not yet affordable for most design studios but, fortunately, they can be contracted as a service. This paper evaluates the impact of computation as a service in architecture and, more specifically, the remote use of HPC for AD, with the aim of reducing the time and costs associated with computationally expensive processes. A set of experiments were made involving analysis, optimization, and rendering of a selected case study. Results indicate that HPC services are advantageous, particularly when performing embarrassingly parallelizable tasks such as rendering. However, some challenges remain, namely the required expertise.*

Keywords: *Algorithmic Design; High-Performance Computing; Design Optimization; Performance Analysis; Visualization.*

1. Introduction

Architecture must respond to the ever evolving social and environmental demands, such as the growing awareness on the industry's ecological footprint (Boeck et al., 2015; Dillen et al., 2020). By taking advantage of new digital tools and computation-based design approaches, architects have been increasingly exploring design solutions that meet these demands (Kolarevic, 2005). Algorithmic Design (AD) is one such approach (Caetano et al., 2020)

that facilitates the integration of analysis and optimization mechanisms since early design stages. This not only provides architects with a better grasp of their designs' behavior (Figliola and Battisti, 2021; Henriksson and Hult, 2015; Oxman, 2008), but also critically helps them orient the design process in a more informed manner.

Nevertheless, analysis and optimization tasks are typically highly time- and resource-consuming, often implying a quality tradeoff. A straightforward solution for this problem is the use of High-Performance Computing (HPC) resources (Isard et al., 2007; Lin et al., 2021). Core count is one of the determining factors of computer performance and, while the first Central Processing Unit (CPU) invented only had one core to run one tasks, today we expect computers to work on multiple tasks simultaneously. To do so, as well as to handle resource-intensive programs, CPUs have evolved towards multi-cores. Portable computers these days typically have CPUs with four to eight cores, which allows for the computation of around 10^{11} operations per second. High-end desktop workstations go further by combining two or four CPUs in one machine, which elevates the number of allowed operations per second to around 10^{13}. Supercomputers take this concept to the next level by offering hundreds or thousands of CPUs, reaching 10^{17} operations per second.

Unfortunately, only a small fraction of architectural studios worldwide can afford the kind of HPC described above, which limits the latter's potential benefits for nowadays architectural practice, particularly, in solving design optimization problems, which are critical to reduce the ecological impact of the industry. To promote better architecture for all, improving life quality while mitigating the industry's environmental footprint, computation must become affordable to anyone and anywhere. With this goal in mind, this research presents the results of a field report evaluating the potential performance benefits of HPC for AD.

2. Methodology

HPC is advantageous to reach architectural solutions with higher indoor environmental quality and reduced ecological footprints but the access to HPC machines is still limited. This research addresses this problem by evaluating the remote use of HPC in current AD practices using the following methodology:

1. Investigating HPC methods and their potential applications in architecture.
2. Identifying the benefits and challenges of remote HPC for architectural design.

3. Performing a set of experiments for multiple design tasks using a supercomputer to (a) validate the proposed hypothesis regarding time and cost gains, and (b) find ways to surpass the challenges encountered.
4. Analyzing the advantages obtained and challenges faced during the experiments and proposing guidelines for future use.
5. Drawing conclusions on the findings and forecasting future research paths.

In the following sections we elaborate on each of these tasks.

3. Computation in Architecture

The emergence of computation-based tools triggered new design approaches, such as AD (Caetano et al., 2020), that combine the computational power of machines with the architects' creative potential (Terzidis, 2004). Due to its algorithmic nature, AD allows automating repetitive and time-consuming design tasks, facilitates design changes, and increases design flexibility. Therefore, in addition to reducing the time and effort spent in testing new solutions, and thus increasing design space exploration, AD makes it possible to deal with higher levels of design complexity involving multiple design constraints.

Nevertheless, given the considerable computational demands of the required analysis, optimization, and image synthesis tools (Belém, 2019; D'Agostino et al., 2021; Kosicki et al., 2020), addressing the previous constraints often requires having programs running for weeks in HPC workstations, which is not compatible with most projects' deadlines. Furthermore, access to HPC is still limited due to high acquisition and maintenance costs, as well as space requirements.

A possible solution is to allow designers to benefit from HPC remotely. To that end, using AD is critical, as it allows us to algorithmically describe the different design tasks, thus facilitating their manipulation and translation into HPC machines. Our thesis is that, soon, the use of remote computing will feel as natural as other common services (Schubert et al., 2010), like water, gas, television, and internet and this work contributes to the implementation of this reality in the architectural context.

4. Computation as a Service

Providing HPC as a service presents itself as a possible solution to eradicate the current inequality in access to computation resources worldwide and make it affordable to a wider audience. Distributed HPC allows users to run

programs on a grid of remote machines from the comfort of their homes, and it is already being used for rendering and gaming on the cloud (Armbrust et al., 2009). Blender, Autodesk, and Google Stadia, for instance, allow users to remotely use computer farms for specific tasks.

Nevertheless, many of the available HPC resources run on operating systems that are quite different from those typically used by architects, such as Windows or MacOS. As such, HPC often requires converting design data and processes to match the specificities of its environment. Moreover, HPC processes tend to be script-based, not providing immediate feedback nor supporting user interaction via Graphical User Interface (GUI). This means that the description of the converted processes must be, first, entirely algorithmic; second, carefully planned to avoid mid-process errors; and third, adapted to the computing environment used (e.g., distributed computing or grid computing). Contrastingly, traditional architectural processes have a visual-based nature, thus largely deviating from the language understood by HPC machines.

Unlike traditional architectural processes, AD already relies on algorithmic descriptions and thus it is a step closer to supporting architectural design in HPC environments. However, it still requires the adaptation of the algorithms since, in most cases, to benefit from a supercomputer, one must parallelize the work, distributing computational tasks through the available computing nodes. Typical AD processes are composed of a plethora of tasks related to design exploration, analysis, optimization, and visualization, and these tasks can range from embarrassingly parallelizable to entirely sequential. Therefore, different parallelization solutions must be considered for each case (Pereira, 2022).

5. Parallel Computing

Parallel computing is a type of computation that benefits from multiple processors to solve a problem, performing many calculations simultaneously (Quinn, 1994). Parallelization, in turn, is the act of processing data in parallel, instead of serially, therefore allowing several problems to be solved independently and simultaneously. Two main parallelization options currently exist: (1) distributing the algorithmic instructions through the available hardware by using multi-threading and multi-processing or (2) using distributed (or cloud) computing strategies.

Multi-threading is the simplest parallelization option, and it supports running multiple tasks on multiple executing threads simultaneously on a single multi-core machine, i.e., a machine that has more than one processing unit, providing easy access to shared memory. Multi-processing also involves a single machine, but each task is implemented by an isolated process and,

thus, does not typically share memory with others. Therefore, exchanging data between processes is not as efficient. In both cases, scalability remains an issue since we are limited to the computing power of a single machine.

Distributed computing involves running multiple processes on different machines. Since it uses a network of machines, this strategy offers greater scalability. However, as the machines are physically separated from each other, exchanging data involves a slow communication process.

To run processes efficiently with parallel computing, data exchange should be minimized, which suggests design tasks that are entirely independent from one another. Figure 1 illustrates several embarrassingly parallelizable cases where no dependencies between separate processes exist.

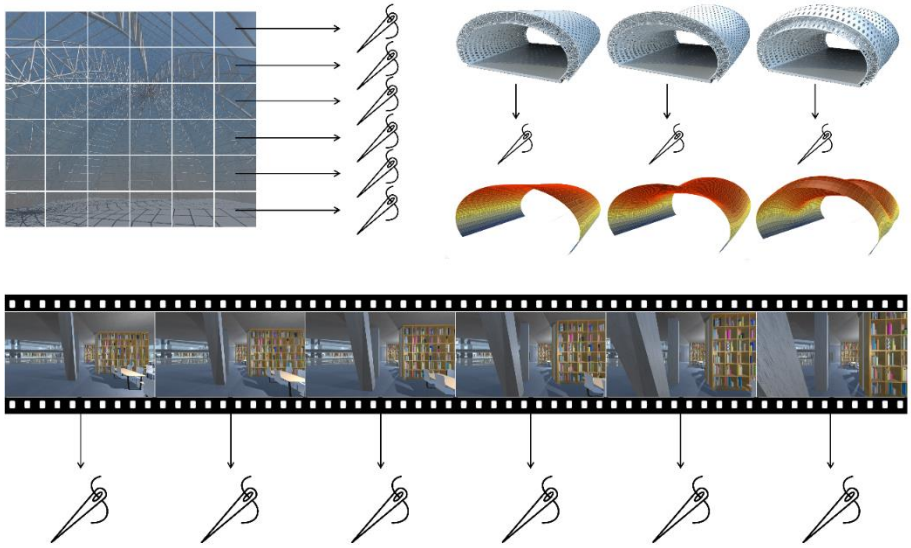

Figure 1. Embarrassingly parallelizable cases: each pixel of a render image (top left); each frame of a render sequence (bottom); and each simulation in a sensitivity analysis (top right).

Cloud computing is a type of distributed computing that delivers computational services remotely through the internet (Armbrust et al., 2009). It is thus a potential solution to the existing inequality of access to HPC, allowing users to benefit from the computation power of a grid of machines anywhere on Earth with internet access. By only requiring the use of personal computers as terminals from where instructions are launched and results collected, cloud computing constitutes an affordable option that has the potential to approximate the status of other daily life services, such as water, gas, television, and internet. It is also a promising solution to enable and democratize the use of analysis, optimization, and image synthesis processes, towards a more socially and environmentally conscious architecture.

6. Challenges of HPC for AD

HPC currently has two main challenges: (1) the need to send the instructions in a batch-processing style and (2) the need to carefully plan the distribution of the tasks to perform. This section elaborates on these two issues, proposing guidelines to overcome them.

6.1. BATCH PROCESSING

HPC currently entails batch processing, which means computing tasks are submitted to the remote computing service, returning the results only after completion. A similar scenario occurs in AD, which involves, first, planning a sequence of instruction for the computer to perform; then, describing them in a program; and, lastly, forcing their execution by running the program.

During the execution of simple design-related instructions, model re-generation is usually fast enough to give a false sense of interactivity, i.e., allowing us to visualize each program change reflected on the 3D model almost immediately. As such, the execution method lying underneath often goes unnoticed. However, the same is not true for the type of processes that benefit from HPC. As the execution time is typically longer, the interactivity illusion often fails, offering little to no visual feedback on the course of the process. This means that mistakes are only discovered at the end of the process, or when an error breaks it midway. Given that these processes may last for days or even weeks, it is critical to minimize the chances of errors.

To that end, users should carefully plan parallelization jobs before sending them to HPC services, performing sanity checks, which involve testing and validating every component and stage separately, and making limited runs at a smaller scale. In most cases, the time invested in these validations pays off; their relevance proportionally growing with the size of the computation task.

Another challenge of batch processing lies in the architects' typical lack of experience with textual programming. Due to the smoother learning curve and typical interactivity, visual programming offers a more democratized access to computational methods at small scales, allowing architects with little to no programming experience to rapidly achieve interesting results. However, it tends not to scale to large construction projects without the aid of textual scripting (Janssen, 2014; Leitão et al., 2012; Ma et al., 2021), therefore not supporting the processes that potentially benefit from HPC. To that regard, textual programming is the *de facto* tool for large-scale development.

6.2. ALLOCATING JOBS

Parallelized approaches require job allocation, that is, deciding how to map tasks to the HPC hardware. While some tools, such as POVRay and

Accelerad, already know how to handle job allocation, in other cases, this falls under the programmer's responsibility. In that case, it typically requires executing commands at the level of the operating system, a task that typically lies outside of the architect's comfort zone.

Additionally, when the task involves the modeling tools architects typically use, such as AutoCAD or Rhinoceros, the situation becomes worse because they are usually not compatible with HPC operating systems. A possible solution is to create virtual machines running the needed operating system but there are other problems. As these tools are mainly conceived for single-user/single-threaded use, they can hardly steer parallel processes by themselves. There are workarounds for this, such as having several instances of the tools running concurrently, but other challenges may arise with this solution, such as concurrent file writing or license limitations. While the first can, once more, be solved with virtual machines, the latter has no solution besides buying more licenses.

Following the same logic, when multithreaded tools, such as POVRay, are not using all the available hardware resources, we can also force the execution of multiples instances to increase the parallelization. In all these cases, however, the programmer must plan the division of labor among the tool's instances and must ensure the parallelization does not surpass the available resources.

7. Evaluation

This section investigates the potential of using HPC resources to perform different AD tasks through practical experiments using the Khepri AD tool (Sammer et al., 2019) due to its portability between different design, analysis, and optimization tools. For that, we modeled a structural case study in Khepri and selected, among the supported tools, those that already support batch processing, namely Frame3DD and POVRay. From the experiments developed, we present three relevant ones encompassing analysis and rendering (section 7.4) and optimization (section 7.5). In the following sections, we elaborate on (1) the adopted AD workflow for remote HPC, (2) how we overcame the challenges of HPC (particularly those regarding batch processing and job allocation), and (3) the time gains in each case.

7.1. CASE STUDY

The performed experiment addresses the design space exploration of a simple truss structure inspired by Gaudi's catenary curves (Figure 2 left), whose legs can be interconnected using different truss schemes (Figure 2 right). The truss is made of Bamboo and is placed on a slab with a randomized outline, which

means it does not have an axis of symmetry and, therefore, presents an interesting resistance test case. In this case study, we were interested in simulating and optimizing its structural performance, as well as in developing render images of possible design variations.

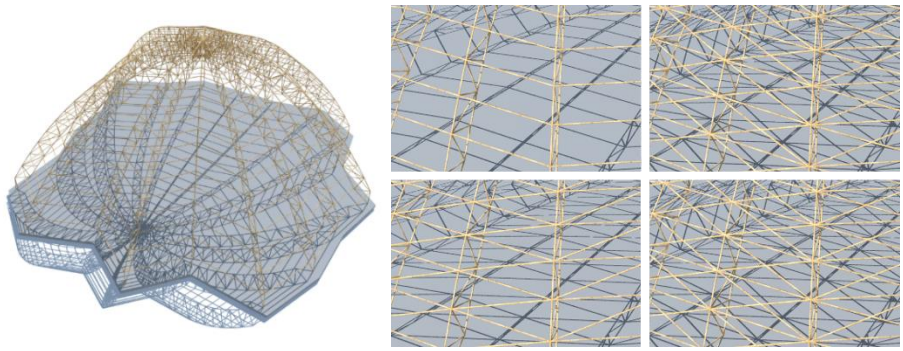

Figure 2. Gaudi-inspired truss structure with randomized outline (left) and detail of different truss schemes (right).

7.2. HARDWARE CONDITIONS

The evaluation was conducted on a supercomputer containing four computing nodes, each providing 96 AMD EPYC 7552 cores, running at 2.2 GHz and accessing 512 GB of RAM. In total, the partition allowed 384 simultaneous execution threads, using 2 TB of memory. Although these capabilities were constrained by the supercomputer's topology and the available resources at each moment, they still represent a significant amount of computing power when compared to current commodity hardware, which typically supports only 8 execution threads using 16 GB of RAM.

7.3. BATCH PROCESSING

There are large differences between the hardware of the supercomputer and that of a typical laptop, but the differences in their software are even bigger. As the supercomputer uses the Linux-based CentOS 7 operating system, which mostly operates in batch mode, the scripts sent to it must carefully describe the intended executions and the resources needed. Moreover, it does not provide immediate feedback, nor does it support any program requiring either user interaction or a GUI.

To help deal with these challenges, we used the job scheduling system of the popular open-source cluster management tool Slurm. Since not all software available for Linux can directly run on a supercomputer, we also installed the exact same operating system on a local virtual machine. This

allowed us to more easily recompile the software and, only after successfully testing them on our own virtual machine, move it to the supercomputer.

The AD tool used in the case study, Khepri, is based on the Julia programming language, which supports multi-threading and distributed computing, provided by the Distributed standard library as well as external packages, such as MPI.jl and DistributedArrays.jl, Khepri, however, is not thread-safe, meaning that it is not prepared for parallel execution. Hence, we were particularly interested in testing its distributed computing capabilities.

7.4. USING MULTITHREADED SOFTWARE

Design space exploration is one of the simplest applications of HPC in AD. In this case, the idea was to study the impact of the design parameters in the performance of the above-mentioned truss structure.

Our first experiment tested a vertical load of increasing magnitude (from 0 to 100N) applied to all non-supported truss nodes. For each load case, the structure was analyzed using Frame3DD and the computed truss node displacements were used to show the shape of the truss under load in the render produced by POVRay (Figure 3).

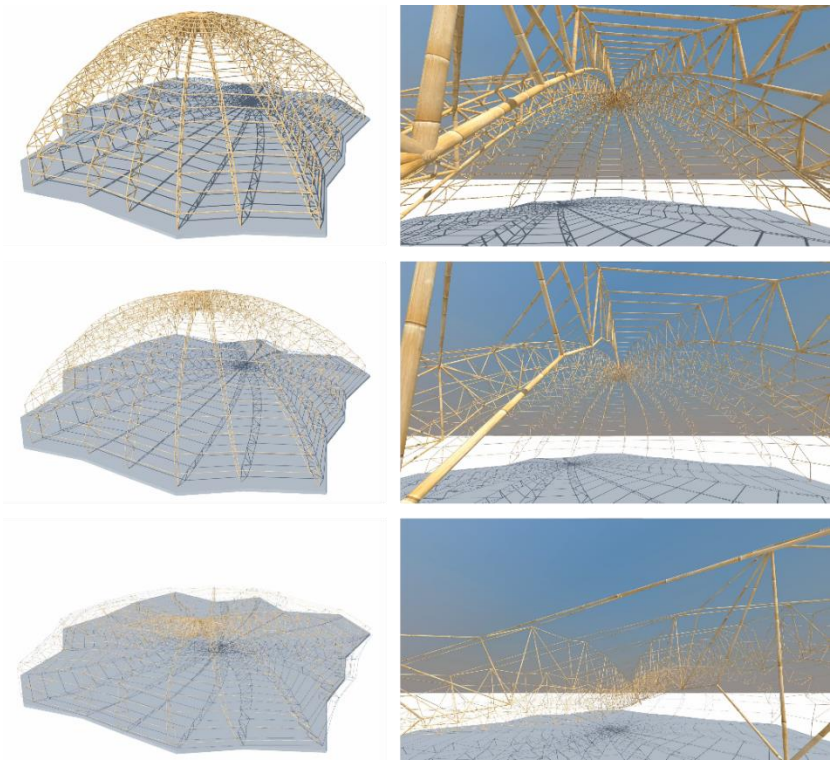

Figure 3. Renders of the truss under different loads.

Each structural analysis was entirely sequential, so we could not benefit from multiple threads on a single evaluation. However, the most time-consuming task in this process was not the analysis itself, but rather the rendering of the result afterwards - a task that is highly parallelized. For rendering in POVRay we took full advantage of the 96 CPUs available on each node. The parallelization process is schematized in Figure 4. The evaluation of 200 different load cases together with the Full HD rendering of the results took 1h46m to complete. In a typical PC, the same experiment would have taken approximately 14 hours.

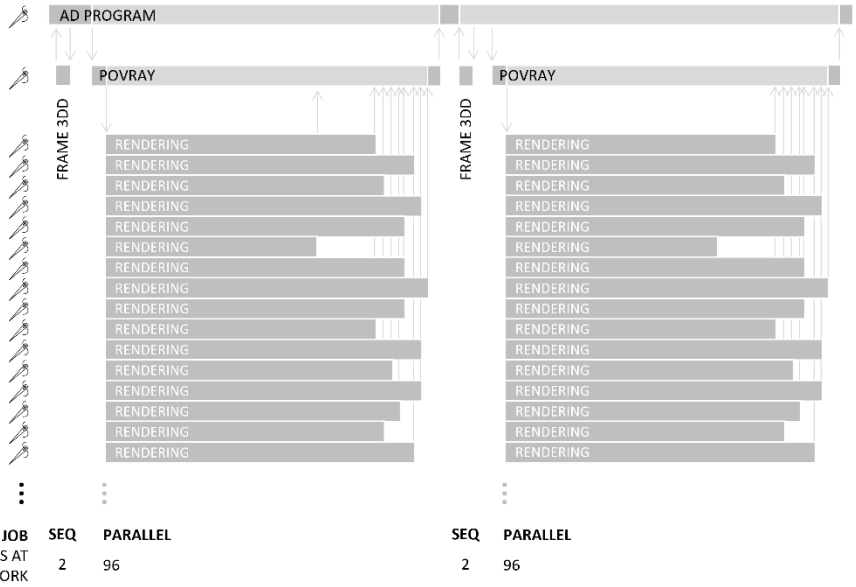

Figure 4. The AD program is sequentially launching analysis processes in Frame3DD and rendering processes in POVRay. POVRay, as a multithreaded tool, automatically distributes the task among the available cores.

In the next test, we measured the scalability of POVRay, by rendering image sequences of the 3D structure in two sizes (1024x768 and 1920x1024) and with different materials (Figure 5) while increasing the number of CPUs on each test. Figure 6 shows the time spent for each case and for different numbers of threads. To eliminate possible fluctuations in the load of the computing node, we present the average of three repeated tests.

Once more comparing to commodity hardware, while in our experiments the time per image when using 96 cores was 30.5 and 110.3 seconds for the small and large resolution image, respectively (Figure 6 top), in a typical PC we would likely be limited to eight threads. Still using a supercomputer, the time per image using eight threads was 214 and 857 seconds for the small and

large resolution image, respectively, which represents an 8X slowdown. On a desktop, depending on the hardware, this slowdown can be even bigger.

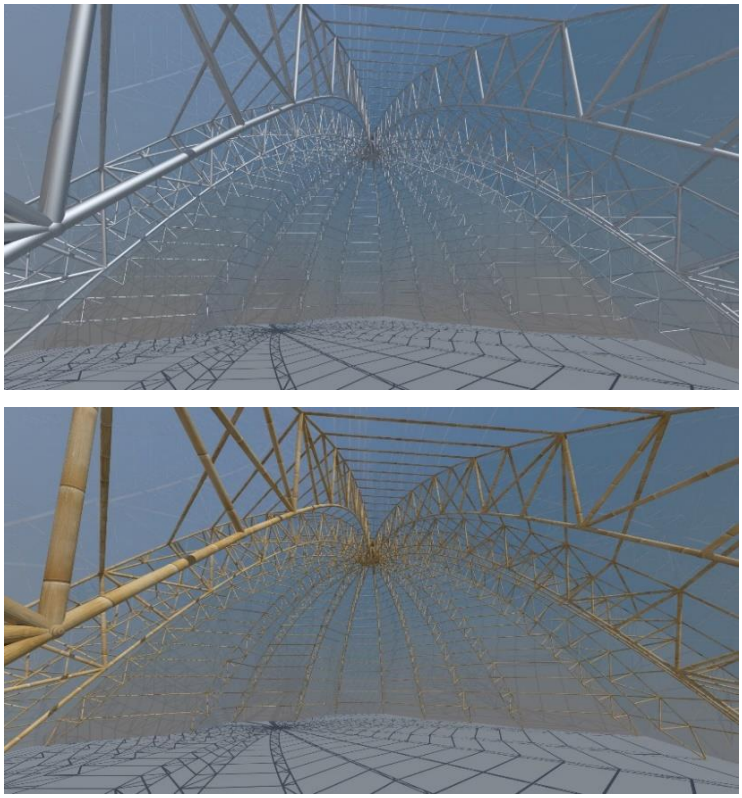

Figure 5. Renders of the truss with different materials: streel and glass (top), and bamboo and glass (bottom).

Now focusing on HPC, in general, there are relevant speedups up to the upper limit of threads. Although it pales in comparison to the initial gains, from 80 threads to 96 threads there is still a significant reduction in the high-resolution image. Based on this analysis, we can determine the number of threads we should use. As is visible in Figure 6 (bottom), for the rendering task with the smaller resolution, it only paid off to use up to 80 threads, obtaining an almost 40X speedup when compared to using just one thread. After that, the gains were marginal. In the case of the larger resolution one, despite the fluctuations, not only were we able to reach a speedup of almost 65X, but the trend line also evidenced the potential for achieving even bigger speedups. In fact, POVRay can take advantage of 512 threads, which means we were still a long way from the limit.

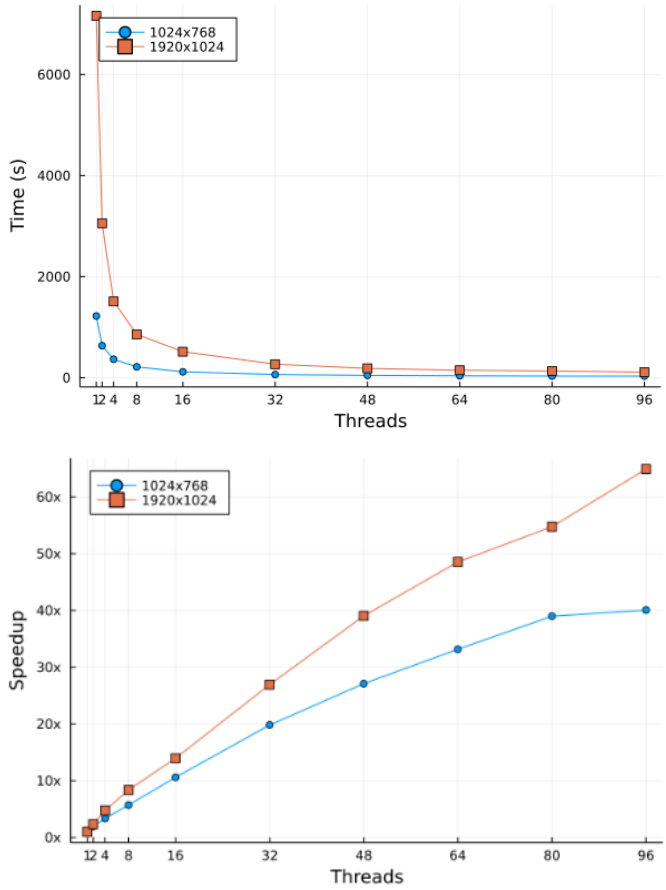

Figure 6. On top, the trend lines for the mean time spent in rendering for different numbers of processes (threads) for the two image sizes (1024x768 and 1920x1024). On the bottom, the speedup obtained from the same data set.

7.5. LAUNCHING CONCURRENT PROCESSES

To measure the potential gains that parallelization could provide to optimization problems, we evaluated a non-parallelizable objective function: the optimization of the truss' structural performance, measured by the maximum displacement of the nodes. To that end, we selected the variable to optimize during the experiment – a vector containing the X and Y coordinates of the truss' center node, where all the arches join - and kept all remaining design variables unchanged, including the truss' height.

To evaluate the objective function, we used the structural analysis tool Frame3DD. To optimize this function, we used two different parallelized optimization strategies from the BlackBoxOptim library: Exponential Natural

Evolution Strategy (xNES) and Separable Natural Evolution Strategy (sNES). BlackBoxOptim supports multi-threading and multi-processing, allowing the optimization algorithm to evaluate many candidate solutions at the same time. Since Khepri is not yet thread-safe, we opted for multiple independent processes.

To evaluate the scalability of the optimization process, we performed several experiences with a varying number of working cores. We followed the BlackBoxOptim guidelines, setting up a master process responsible for running the optimization and worker processes responsible for evaluating candidate solutions (Figure 7).

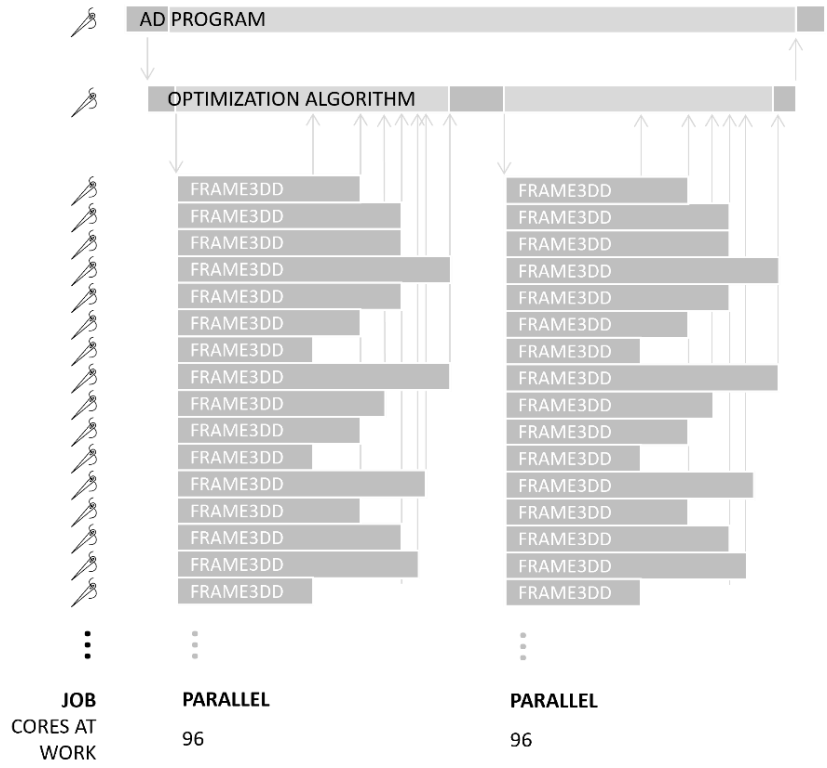

Figure 7. The optimization algorithm functioning as the master process, launching parallel Frame3DD processes to evaluate candidate solution batches.

For reproducibility purposes, we fixed the seed of the master process' random number generator, allowing us to repeat the experiments with a different number of workers while ensuring the same solution is reached after the same number of steps. We performed three independent runs for each test to smooth out the noise, set an initial population size of 500, and allowed the

optimizations to do a maximum of 5000 objective function evaluations. Figure 8 presents the mean time spent in the optimization with different numbers of processes for both algorithms.

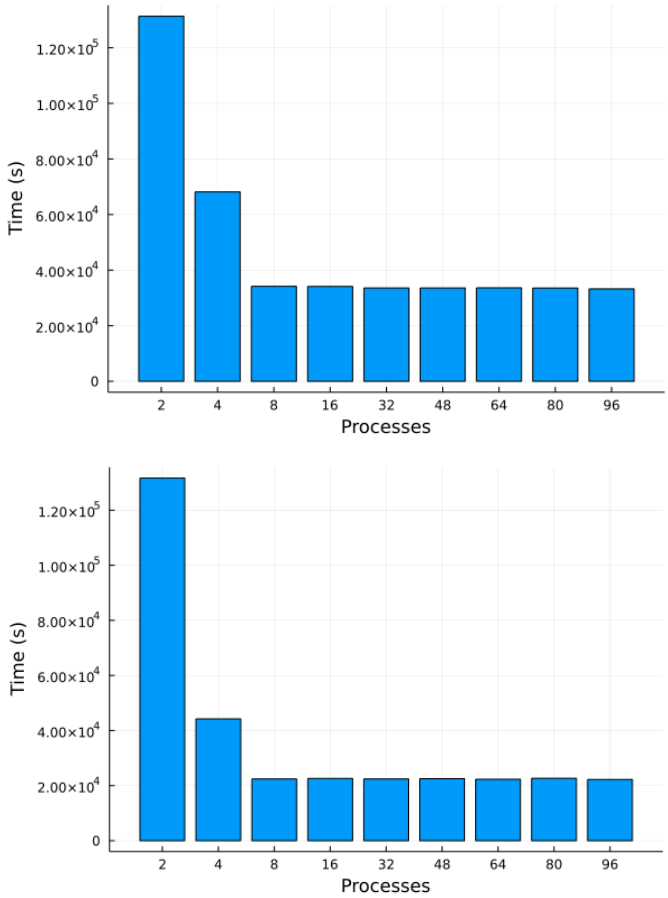

Figure 8. Time spent in the optimization process of the truss structure for different numbers of processes using xNES (top) and sNES (bottom).

Results show that the optimization clearly benefited from the parallel evaluation of candidate solutions but only up to eight concurrent processes. It seems that the BlackBoxOptim library is not yet fully capable of exploring a large number of computing resources. Unlike the previous experiments, the gains of the supercomputer do not surpass those of a normal laptop.

8. Discussion

This section discusses the results of the evaluation, reflects upon the way the two main HPC challenges were handled, the lessons learnt in the process, and the time and cost gains of the experience.

Regarding the first topic, results show that accessing HPC resources is advantageous for architectural design practice, particularly when performing embarrassingly parallel tasks such as rendering. In these cases, we concluded that HPC provides large computational gains (the greater the number of processes, the greater the speedups achieved), and even greater gains are expected with higher numbers of threads than those tested in these experiments. In design optimization tasks, however, the speedups obtained were not as impressive. The results lead to the conclusion that for the specific optimization algorithms and for the problem addressed in this research there is no need to use more than eight processes.

There is, however, a silver lining: if it does not compensate to launch more than eight processes for a given optimization algorithm, we can use the remaining computing resources to evaluate other algorithms. This is particularly beneficial, for instance, when addressing the No Free Lunch theorem (Pereira and Leitão, 2020), which states that no optimization algorithm is better than all others in all cases. The consequence is that multiple algorithms need to be tested and HPC allows these tests to be done simultaneously, thus taking no longer than running the slowest of them.

The two main challenges of HPC, batch processing and job allocation, were surpassed in these experiments with a high dose of manual labor. Some of the parallelization solutions presented were achieved only after several trial-and-error loops, leaving us with a list of lessons for future use:

1. HPC resources provide little to no compatibility with the tools architects typically use: tools requiring GUIs can hardly run in HPC environments and even batch-oriented tools frequently need to be adapted. Furthermore, lack of administrative privileges impedes software installations on HPC environments. Workarounds need to be found.
2. Not all software or design tasks can benefit from HPC resources. The structural analysis tool used (Frame3DD) is one example that could not benefit from multiple CPUs because the software was not parallelized. A specific parallelization strategy must then be devised for each case.
3. Even in the processes that directly benefit from parallelization, the performance improvements achieved are variable. For instance, while the rendering tasks considerably benefited from supercomputing resources, the optimization tasks using parallelized algorithms only

benefited up to a point. These limits should be tested and known before launching large processes since failing to consider them may constitute a waste of resources.

4. It only pays off to parallelize if the time it takes to start the parallel tasks is significantly smaller than the time needed to complete those tasks. Otherwise, instead of speeding up the computation, we might end up slowing it down.

Finally, we circle back to the goal of this paper - affordable computation for architecture - by discussing the cost of these experiments. Despite the huge computational power of the supercomputer we used in this evaluation, its operational costs are rated at 0,01€ core*hour. This means that even when using all available resources (4 nodes with 96 cores each), we pay less than 4€ per hour, which is an enormous cost reduction when compared to the acquisition and running costs (electricity and maintenance) of a personal workstation or, in any case, the sort of workstation required to handle the demanding computations presented in reasonable time (Isard et al., 2007).

9. Conclusion

Algorithmic Design (AD) allows architects to enhance their design processes by facilitating the integration of analysis and optimization since early design stages. However, these tasks are typically highly time- and resource-consuming, which makes them difficult to apply on the typical hardware available to architects. High-Performance Computing (HPC) is a tempting solution to these problems.

In this paper, we presented a field report evaluating the potential benefits of remote HPC for AD workflows. The work outlines the two main issues associated with HPC for AD, batch processing and job allocation, and describes how we overcame them in the process of parallelizing algorithmic design, analysis, optimization, and visualization.

Our results show that remote HPC can considerably reduce the time and costs associated with computationally expensive processes, making AD approaches accessible to users with limited resources. However, some challenges remain as expertise is required to surpass the issues associated with HPC. Future research paths should focus on parallelization strategies that can facilitate the planning, testing, and launching of processes for architects.

Acknowledgements

This work was supported by national funds through *Fundação para a Ciência e a Tecnologia* (FCT) (references UIDB/ 50021/2020, PTDC/ART-DAQ/31061/2017) and PhD grants under contract of FCT (grant numbers SFRH/BD/128628/ 2017, DFA/BD/4682/2020).

References

ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A.D., KATZ, R.H., KONWINSKI, A., LEE, G., PATTERSON, D.A., RABKIN, A., STOICA, I. AND ZAHARIA, M., 2009. Above the Clouds: A Berkeley View of Cloud Computing.

BELÉM, C.G., 2019. *Optimization of Time-Consuming Objective Functions: Derivative-free approaches and their application in architecture*. Instituto Superior Técnico, University of Lisbon.

BOECK, L., VERBEKE, S., AUDENAERT, A. AND MESMAEKER, L., 2015. Improving the Energy Performance of Residential Buildings: A literature review. *Renewable and Sustainable Energy Reviews*, 52, 960-975.

CAETANO, I., SANTOS, L., AND LEITÃO, A., 2020. Computational design in architecture: Defining parametric, generative, and algorithmic design. *Frontiers of Architectural Research*, 9, 287–300.

D'AGOSTINO, D., D'AGOSTINO, P., MINELLI, F. AND MINICHIELLO, F., 2021. Proposal of a new automated workflow for the computational performance-driven design optimization of building energy need and construction cost. *Energy and Buildings*, 239.

DILLEN, W., LOMBAERT, G., MERTENS, R., BEURDEN, H. Van, JASPAERT, D. AND SCHEVENELS, M., 2020. Optimization in a realistic structural engineering context: Redesign of the Market Hall in Ghent. *Engineering Structures*, 228.

FIGLIOLA, A. AND BATTISTI, A., 2021. Feedback on the Design Processes for the Materialization of Informed Architectures. *In:* A. FIGLIOLA AND A. BATTISTI, ed. *Post-Industrial Robotics: Exploring Informed Architecture*. Springer Singapore: Singapore, 155-173.

HENRIKSSON, V. AND HULT, M., 2015. *Rationalizing Freeform Architecture: Surface discretization and multi-objective optimization*. Chalmers University of Technology.

ISARD, M., BUDIU, M., YU, Y., BIRRELL, A. AND FETTERLY, D., 2007. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. *SIGOPS Oper. Syst. Rev*, 41, 59–72.

JANSSEN, P., 2014. Visual Dataflow Modelling: Some thoughts on complexity. *In: Fusion: Proceedings of the 32nd eCAADe Conference*. Newcastle upon Tyne, UK, 305-314.

KOLAREVIC, B., 2005. Towards the performative in Architecture. *In:* B. KOLAREVIC AND A.M. MALKAWI, ed. *Performative Architecture: Beyond Instrumentality*. Spon Press: London, 203-214.

KOSICKI, M., TSILIAKOS, M. AND TSIGKARI, M., 2020. HYDRA Distributed Multi-Objective Optimization for Designers. *In: Impact: Design with all Senses*. Springer International Publishing, Cham, 106–118.

LEITÃO, A., SANTOS, L. AND LOPES, J., 2012. Programming Languages for Generative Design: A Comparative Study. *International Journal of Architectural Computing*, 10(1), 139-162.

LIN, B., CHEN, H., YU, Q., ZHOU, X., LV, S., HE, Q. AND LI, Z., 2021. MOOSAS – A systematic solution for multiple objective building performance optimization in the early design stage. *Building and Environment*, 200.

MA, W., WANG, X., WANG, J., XIANG, X. AND SUN, J., 2021. Generative Design in Building Information Modelling (BIM): Approaches and Requirements. *Sensors*, 21.

OXMAN, R., 2008. Performance-based Design: Current Practices and Research Issues. *International Journal of Architectural Computing*, 6(1), 1-17.

PEREIRA, I., 2022. *Reconstructing Architectural Optimization Workflows*. Instituto Superior Técnico, University of Lisbon.

PEREIRA, I. AND LEITÃO, A., 2020. More is more: The no free lunch theorem in architecture. *In: Imaginable Futures: Design Thinking, and the Scientific Method: Proceedings of the International Conference of Architectural Science Association*. Auckland, New Zealand, 765-774.

QUINN, M.J., 1994. *Parallel Computing: Theory and Practice*, 2nd edition. ed. McGraw-Hill, Inc., USA.

SAMMER, M., LEITÃO, A. AND CAETANO, I., 2019. From Visual Input to Visual Output in Textual Programming. *In:* M. HAEUSLER, M. SCHNABEL AND T. FUKUDA, ed. *Intelligent & Informed: Proceedings of the 24th International CAADRIA Conference*. Wellington, New Zealand, 645-654.

SCHUBERT, L., JEFFERY, K., NEIDECKER-LUTZ, B., BAROT, P., BEHR, F., BOSCH, P. AND BRANDIC, I., 2010. *The Future of Cloud Computing - Opportunities for European Cloud Computing Beyond 2010*.

TERZIDIS, K., 2004. Algorithmic Design: A Paradigm Shift in Architecture? *In: Architecture in the Network Society: 22nd eCAADe Conference Proceedings*. Warsaw, Poland, 201-207.