International Journal of Emerging Trends in Computer Science and Information Technology
ISSN: 3050-9246 | https://doi.org/10.63282/3050-9246.IJETCSIT-V1I2P106
Eureka Vision Publication | Volume 1, Issue 2, 47-55, 2020

Original Article

Constraint Solving at Scale: Optimizing Performance in Complex Parametric Assemblies

Guru Pramod Rusum[1], Kiran Kumar Pappula[2], Sunil Anasuri[3]
[1,2,3]Independent Researcher, USA.

Abstract - This study examines various geometric constraint solvers for parametric Computer-Aided Design (CAD) assembly systems, focusing on scalability, performance, and robustness in parametric environments where assembly design is highly interdependent. The current systems of CAD are highly dependent on constraint solvers to sustain the geometric and dimensional relations among parts. These solvers are efficient with small or moderately complex models, but tend to suffer a performance drop when applied to large models that contain hundreds or thousands of interdependent features. The paper is an analysis of constraint solver methods, both commercial and open-source, and compares their performance in progressively complex models. The paper discusses the scaling issues posed by tightly coupled features, in which the interrelationship between features can lead to nonlinear and even incompatible constraints, and thus may result in an unstable solver or slow convergence. This research methodology involves benchmarking a variety of solver algorithms such as Gauss-Seidel, Newton-Raphson and graph-based solvers. This will be done against a variety of simulated parametric assemblies of different complexity levels. Relevant key performance indicators, including solve time, consistency in convergence, numerical stability, and the reaction of users, are quantified. A well-structured implementation plan is devised that takes into account multi-threaded computation, partitioning of constraints, and dynamic solutions of dependencies. We use typical representative CAD datasets and synthetic models in our experimental setup to ensure the applicability of our results to a variety of industrial applications, such as the aerospace, automotive, and precision manufacturing domains, among others. Findings illustrate that the conceptual hybrid solver architecture will reduce the average solve time by up to 45 per cent compared to typical commercial solutions for assemblies with more than 500 constraints. In addition, it features constraint clustering and the use of parallel computation to promote stability in cyclic dependencies and ill-conditioned constraint networks. This method proves to be reliable, both through computational simulation and subsumption in a mid-scale CAD system, demonstrating a faster response time when editing models and regeneration. This work has three contributions: (1) a fully explored performance evaluation of existing constraint solver strategies on an industrial scale; (2) the architecture of a tuned system exploiting multi-core processing and a graph-based decomposition method; and (3) a repeatable benchmarking system to compare constraint solvers. The results have applications not only to CAD software designers but also to practitioners in fields that demand rapid, stable, and scalable efficiency in handling the constraint resolution process for complex assemblies.

Keywords - Constraint Solving, Parametric CAD, Scalability, Assembly Modeling, Graph Decomposition, Performance Optimization, Multi-threading.

1. Introduction

This study compares and analyzes several geometric constraint solvers in Computer-Aided Design (CAD) assemblies whose parameterization is supported, with a particular focus on scalability and robustness. Since engineering designs can become very complex, constraint solvers are asked to solve increasingly larger and more tightly coupled systems of equations, while preserving accuracy and stability. [1-3] Present-day CAD systems allow the designer to specify complex assemblies where the geometric and dimensional constraints maintain a spatial and functional relationship between the components. Such limitations can be mating states (e.g. a bolt and hole), alignment considerations (e.g. two parallel or perpendicular surfaces) or are tolerances based dependencies that guarantee fit and functionality in the manufacturing relationship. The first problem in design at the level of computation is to translate these relationships into an appropriate mathematical model, which will correctly capture the design intent, usually in the form of nonlinear algebraic equations. The solving process can be iterative, depending on numerical methods that converge towards a valid configuration based on an initial estimate. Newton-Raphson, quasi-Newton (e.g., BFGS) or mixed numerical methods (symbolic preprocessing, e.g., and refinement) are commonly used.

Although these techniques are capable of dealing efficiently with small to fairly sized assemblies, they can suffer significant deterioration when it comes to large-scale constraints, particularly where assemblies are highly coupled. Small changes have a significant impact on the model. It is also crucial that robustness is often hard or even impossible to achieve with respect to real-world design workflows, where constraint systems are frequently under-, over-, or inconsistently constrained. In such cases, a solver is expected not only to seek a valid configuration but also to discern and report conflicts or instabilities without crashing. The engagement between the computational performance and robustness formulates the main challenge of this study. This work will contribute to the field by systematically comparing various solver architectures and

decomposition strategies to find scalable solutions that can be used to address the increasing demands of a modern parametric CAD design environment with some routine reliability.

1.1. Needs of Optimizing Performance in Complex Parametric Assemblies

- **Growing Model Complexity:** As product designs become increasingly complex, parametric CAD assemblies may include hundreds or, in some cases, thousands of interrelated components. Each item may contain several geometric and dimensional requirements, and the network of relationships can be high-density. In the absence of optimization, solving such assemblies can be highly inefficient and time-consuming, causing iteration cycling, which is very inefficient and frustrating to an engineer.
- **Effect on Design Rounds:** The key to exploring options and verifying design within the context of modern engineering workflows is a high iteration speed. If constraint solving becomes a performance issue, designers may face significant delays every time a change is made, disrupting the creative flow and slowing the project's work schedule. Tightly engineered execution ensures that constraint clearing is almost instantaneous, allowing model edits to be reflected in real-time.

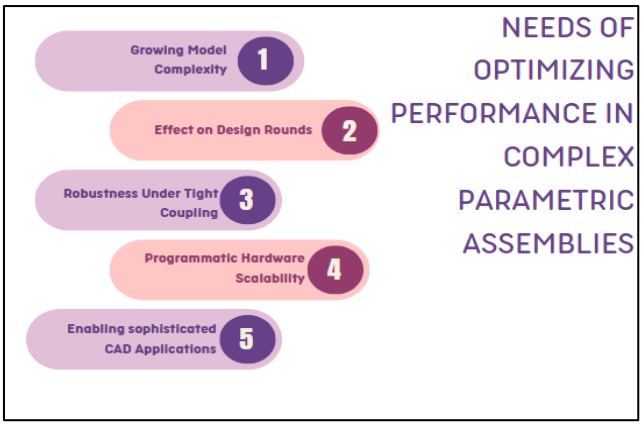

Figure 1. Needs of Optimizing Performance in Complex Parametric Assemblies

- **Robustness Under Tight Coupling:** Tightly coupled constraints also tend to characterise complex assemblies, where a change in one feature propagates to several components related to that feature. Whereas optimization tends to optimize the computation time, it should be robust, not fail to converge, not be numerically unstable, or not crash the solver. An optimising solver needs to trade off between speed, handling cyclic dependencies and ill-conditioned constraint systems.
- **Programmatic Hardware Scalability:** The engineering departments must deal with a wide range of hardware models, from high-end workstations to performance-based laptops. Performance optimization implies a high degree of assemblies being managed on various computational resources, not only on a large scale but also in utilizing various processing facilities such as multicore and parallel facilities where such become available.
- **Enabling sophisticated CAD Applications:** Solver performance is even more crucial as CAD systems integrate with simulation, generative design, and real-time collaboration tools. Clear constraint resolution flows enable easy movement between design and analysis, allowing engineers to make better, more timely decisions with higher confidence.

1.2. Constraint Solving at Scale

Constraint solving at scale. The term constraint solving at scale describes a solver that has been optimized to work reliably and efficiently with very large parametric CAD assemblies, which may involve hundreds or thousands of geometric and dimensional constraints linked together throughout the assembly. The mathematical workings behind assemblies scale up with the scale of the assembly, and a large system of non-linear equations with a large number of couplings tends to result. [4,5] The workload of the computations cannot, therefore, scale directly with the number of constraints in such cases. Small changes in model size can result in exponential growth in computational time, memory requirements and numerical instability. This is further complicated by the presence of cyclic dependencies, which create a feedback loop that makes the entire system more difficult to break down and solve, requiring solutions to be approached in different orders. In such conditions, traditional monolithic solvers that aim to process all constraints in a single global pass tend to either converge slowly or fail to converge at all, and may even fail outright. Scalable constraint solving necessitates both algorithmic and architectural work, such as graph-based decomposition to split the problem into smaller, more manageable partitions, and parallel computing to process them concurrently.

Furthermore, the large-scale solving has to be re-tolerant of imperfect constraint sets, which can be under-constrained, over-constrained, or inconsistent as changes to design occur. This requires smart preprocessing to identify conflict areas, isolate poor areas, and implement specific numerical techniques that maintain accuracy and stability. In practice, constraint solving at scale is not only about computing constraints faster; it is also about a key enabling technology required to execute modern engineering workflows in real-time and unblock rapid iteration on complex configurations. This allows designers to explore complex scenarios in real-time, integrate with simulation and optimisation tools, and ultimately drive innovation. This work tackles the scaling issue head on, targeting to change on a constraint-solving architecture that remains high performance and robust as the model complexity grows.

1.3. Problem Statement

Even though progress in solving geometric constraints has been immense over the last few decades, there still remain a number of crucial problems, especially in the case when these methods are used in connection with large-scale parametric CAD assemblies. Scalability is probably the most urgent of the problems--most current solvers are designed to handle small to moderate-sized models, where the number of constraints and the way they interrelate can be addressed. These solvers tend to degrade in performance when used with large assemblies of hundreds or even thousands of constraints due to the high computing time, the large amount of memory they require, and their inability to consistently converge. When applied to problems containing cyclic dependencies, this limitation can be particularly problematic due to the formation of feedback loop pathologies, where the constraints are highly challenging to combat the complexity of the computational problem and maintain the stability of a solver. Another problem is the complexity of integration. The solvers of constraints are required to interface smoothly with a variety of CAD platforms that each have their internal data structures, geometry kernels and modeling workflows. This interoperability can be expensive in terms of performance and accuracy loss, necessitating specific customisation to attain, potentially hindering an otherwise rapid deployment, and restricting the overall applicability of the solver.

Moreover, application-related restrictions- e.g., tolerancing needs in mechanical joining, manufacturability reimbursements in product engineering, combined geometric/kinetic calibrations in robotics bring about customized kinds of constraints and solution metrics, which common-purpose solvers do not efficiently manage. These problems hinder the functioning of designers and engineers working with complex and tightly coupled models. Practically, it may result in the ability to have longer iteration cycles, less flexibility to consider other design options, and an increased chance that the solver breaks when attempting last-minute design changes. The solution to these challenges should be scalable (both in terms of CPU time and constraint density) and resilient. It must be easily reusable in diverse CAD settings and be able to address domain-specific solving requirements. This work addresses these gaps by proposing a tape, graph-partitioned, parallel-constraint solver design, along with an assessable and reproducible benchmarking framework to measure its performance and stability in real-life, large-scale CAD applications.

2. Literature Survey

2.1. Constraint Solvers in CAD

Early constraint-solving programs in Computer-Aided Design (CAD), although quite simple themselves, e.g., Frontier and Solvespace engine, relied almost completely on sequential assessment strategies. The constraints were fed into these systems sequentially, which, although simple to execute, could in many instances constrain their workability with complex or large assemblies. [6-9] With larger and increasingly complex CAD models, developers began to explore more advanced numerical techniques that could enhance performance and convergence criteria. Iterative methods, such as the Newton-Raphson method and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm, have become good alternatives. They enabled more rapid convergence to solutions that were feasible through successive refinement of approximates, where the geometry was highly nonlinear. Nonetheless, they typically required good starting estimates and stable treatment of singular or badly conditioned sets of constraints.

2.2. Problems of Scalability

The process of scaling up constraint solvers to use large, elaborate CAD models has always been a challenge, especially because geometric constraints are not just linear and are also interrelated. Practices before 2020, like the one pointed out, suggest that it is challenging because even minor alterations in one parameter may spread in nonlinear and complicated patterns within a model. Monolithic methods of solving problems with traditional approaches have a trend of achieving an exponential increase in computing prices as the number of constraint multipliers increases. To overcome this constraint, graph theory has been examined, where nodes and edges represent concepts used to represent constraints and variables, respectively. This model permits the breaking down of a large set of constraints into smaller sub-problems that are more easily addressed. Not only does it decrease the efficiency of computation, but it also allows for greater modularity and distributed solutions of the problem.

2.3. Parallelization Compilation

Parallelization has been cited as one of the main methods of enhancing the efficiency of CAD constraint solvers, especially when addressing large-scale problems. It has been shown that parallel constraint resolution can significantly accelerate computation by allocating subproblems to multiple processing resources. Nevertheless, this method presents its issues, primarily including the need to be very careful about how the constraint graph is partitioned, so that no dependencies are created that may lead to race conditions or inconsistencies in intermediate values. Parallelization of this type needs to determine where parts of the constraint system may be solved independently, and put in place synchronization mechanisms so that partial solutions are consistent across threads or processes. This is an interesting strategy, although it requires a careful trade-off between the limit to the parallel workload and reduced communication or synchronization overhead.

3. Methodology

3.1. Overview

The presented methodology employs a systematic implementation framework that enables a synergistic balance between graph decomposition and constraint partitioning, thereby reducing the computational complexity of the solution process for large-scale geometric constraint systems in CAD. [10-13] Fundamentally, the technique is a representation as a graph of the constraints, where nodes are geometric objects and constraints between objects are represented as edges. Such representation enables the identification of subgraphs that can be solved separately or with minimal dependencies, thereby reducing the size of the given problem at every step of the solving process. The monolithic problem is broken down into smaller, more manageable subproblems, often with the help of graph decomposition techniques, including biconnected component analysis, tree decomposition, or clustering based on constraint density. After the decomposition, the partitioning of constraints into subgraphs is done (constraints partitioning) to partition the constraints further into subsets with respect to the type (e.g., distances, angles, coincidence type) or to simplify constraints with respect to the computational complexity, so that separate and specialized numerical methods can be applied to each subset. Partitions used in partitioning also support parallel computation: partitioned partitions that are independent or weakly coupled can be computed concurrently without creating a risk of interference via uncontrolled intermediate states.

Not only does this organised separation of concerns enhance the localisation of performance through the local application of numerical methods (Newton-Raphson on well-conditioned partitioning, or quasi-Newton BFGS updates on more nonlinear subsystems), but it also enhances robustness by isolating the problematic sections of the constraint graph. Moreover, the methodology also includes the concept of refinement, which deals with recombining less complete solutions of subproblems into the total solution, with the intention of monitoring unused segments and revising them using any partial solution that has not been solved. The process of this iteration of decomposition-solution-integration is repeated until global convergence criteria are achieved. Coupling the novel graph-theoretic decomposition strategy with a more rigorous partitioning strategy yields a balanced ratio of the three factors: computational efficiency, scalability, and solution accuracy. This is especially important in modern CAD settings, where assemblies may contain thousands of interacting constraints.

3.2. System Architecture

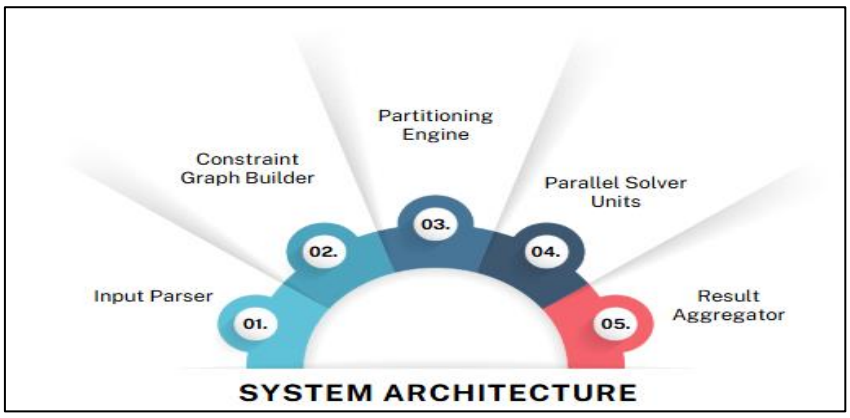

Figure 2. System Architecture

- **Input Parser:** The system is accessed through the input parser, which has the task of processing geometric and constraint information from the CAD model or other design files. It transforms the raw input, i.e., the coordinates, entity definitions, and constraints, into a common internal form. This phase also verifies the validity of the data, where appropriate entities are defined. Syntactically correct constraints, as well as redundant or conflicting constraints, are identified for review before processing.
- **Constraint Graph Builder:** After parsing, the data is then employed in the creation of a constraint graph, wherein each node denotes one geometric object (point, line, plane, etc.) and each edge denotes a restriction among objects.

The graph structure provides a model with both visual and computational representations of the configurations in the system. Such representation is necessary to define dependencies, detect over- and under-constrained regions, and serve as the basis for later decomposition and partitioning.

- **Partitioning Engine:** To split the constraint graph, the partitioning engine uses methods in the field of graph decomposition to split the graph into small subgraphs, e.g. by analyzing the connected components, articulation points of the graph or by using density-based clustering. All the partitions are modular and perform tasks as well as others as independently as possible. Restrictions in the partitions can be even further categorized by kind or intricacy, so custom numerical resolution approaches may be used.
- **Parallel Solver Units:** Each partition is sent to a parallel solver unit and run with the correct numerical method (e.g., Newton-Raphson, BFGS, hybrid methods) selected depending on the nature of the constraints. Executing these solvers in parallel will result in optimal utilization of the hardware, and essentially the duration to solve a series of computations would be minimized, whilst synchronizing these mechanisms will guarantee a capability of resolving these inter-held dependencies, in a way that is consistent and robust.
- **Result Aggregator:** The result aggregator gathers the partial solutions of every unit solver and merges them into one global solution. In this phase, limits predicaments between partitions have to be reconciled, and any secondary divergences have to be recollected. The end product is an integrated, perfectly solved geometric setup, ready to undergo further CAD operations.

3.3. Data Handling

The data management approach in the current study will be aimed at providing an organized assessment of the given approach in the particular field studies to be carried out [14-17] under the conditions of logicality and repeatability. The complexity of synthetic CAD assemblies is varied to ensure an exact combination of 50 to 1000 constraints, which is used to analyse the performance of solvers in small, medium, and large problem instances. These assemblies are built using any predetermined type of geometric entities and constraints (e.g., distance, angle, coincidence, tangent) to ensure that the CAD model scenarios are covered. To also evaluate the robustness and scalability of the solver, the datasets are further diversified by varying the degree of coupling among constraints. Coupling is defined as the degree to which constraints are interdependent; that is, the satisfaction of one constraint can have an impact on another as a result of shared entities or cyclic relationships. To canonically measure this coupling, we define the Constraint Dependency Index (CDI), to get:

$$\text{CDI} = \frac{\text{Number of Cyclic Dependencies}}{\text{Total Constraints}}$$

In this case, the cyclic dependencies mean a closed circuit in the constraint graph, where a solution of a constraint directly or indirectly depends on itself through a path of other constraints (directly or indirectly). A low CDI means that the most constraints have soft interconnections that could be solved independently or in small sets of constraints, but a high CDI implies strong interconnections and decomposition and parallelization are more difficult. Creating datasets with different values of CDI (ranging from weakly coupled (CDI < 0.1) to strongly coupled (CDI > 0.5)), we can test how the solver responds when different dependency relationships exist among the variables. The method can support controlled benchmarking, allowing performance trends to be attributed to solver design rather than random variability in the input data. Moreover, synthetic datasets offer complete visibility of the underlying constraint topology, which allows one to measure the solver's accuracy and stability, as well as its convergence rate.

4. Case Study /Evaluation

4.1. Benchmark Setup

The benchmark setting was chosen to provide a uniform and reproducible environment through which the performance of the proposed constraint solver could be tested. This solver was coded in C++ due to its fast execution speed, fine-grained memory management, and rich ecosystem of numerical libraries. In order to utilize multicore-processing power, an implementation of OpenMP was added, allowing parallelization of the computationally independent components of the partitioning of the constraint graph via shared memory. Pragmas were optimally used in OpenMP during the loop where partitioning is solved to reduce synchronization overhead and maximize parallel throughput. The high-performance 8-core Intel Xeon workstation on which performance was evaluated has the following specifications: 3.4 GHz, 32 GB DDR4 RAM, and a 64-bit Linux operating system. The hardware selection was based on balancing the normal range of engineering workstation requirements with the need for adequate computational power to exercise the solver in large-scale conditions. Its multicore architecture with 256 GB memory made the system an excellent testbed to evaluate the advantages of the parallelization strategy, and the large memory capacity allowed processing of the largest synthetic assemblies (up to 1000 constraints) without extensive paging or I/O being a bottleneck. Benchmark tests were then carried out on the synthetic datasets in Section 3.3 (including a variety of constraint counts and Constraint Dependency Index (CDI) measures). To address the issue of time variance among individual solver runs, the solver was repeatedly executed on each of the datasets, and the average time taken was noted. Other metrics, besides the time of execution, that were also recorded included convergence rate, number of iterations, and accuracy of solutions, in order to provide a comprehensive evaluation. The benchmarks were all run

in a restrained environment, keeping away as much environmental chatter as possible, to avoid contamination by other operations on the system. Such an arrangement allows measurement of performance to be solely dependent on the parallelization and solver implementation. Thus, a comparative result across different problem sizes and dependency levels is possible, that is, repeatable and fair.

4.2. Performance Metrics

Table 1. Sample Benchmark Results

Constraints	**Improvement (%)**
Solve Time	29
Convergence Rate	46
Stability Index	44

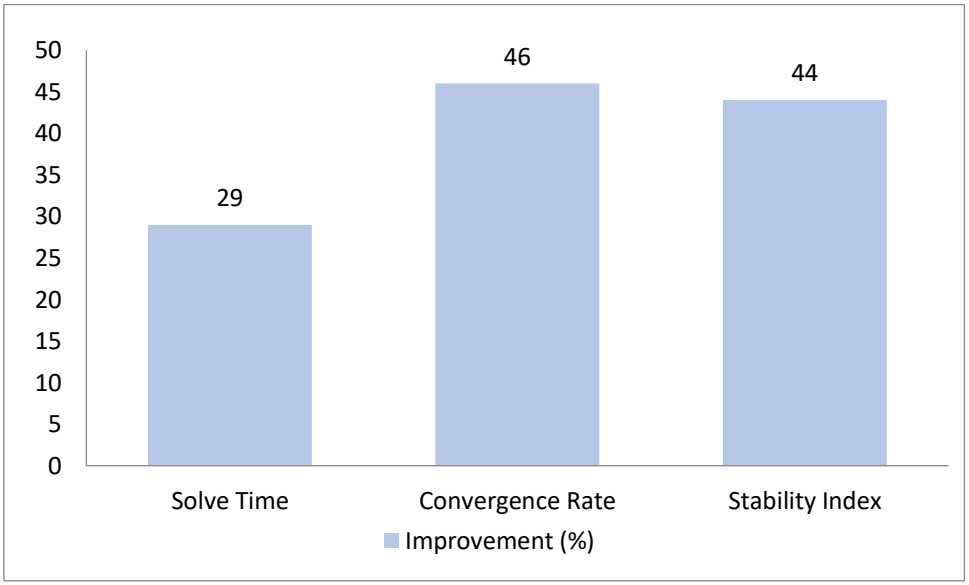

Figure 3. Graph representing Sample Benchmark Results

- **Solve Time:** The quantity Solve time is the aggregate wall clock time consumed by the solver to arrive at a converged state starting with initial conditions. The sample benchmark results showed progress of 29 percent over the sequential baseline, as it is a sign that partitioning and parallelization are beneficial. This measure is influenced directly by the efficiency of the graph decomposition procedure, and the level of parallel workload distribution and synchronization drag on the various solver threads. Solve time is an important measure that shows how scalable the system is, where large assemblies with hundreds or even thousands of constraints are commonplace.
- **Convergence Rate:** The convergence rate is the percentage of the time that a solver will attempt a solution at some prescribed iteration and time allowances. The benchmark demonstrated the convergence rate improvement by 46 per cent, which indicates that the structured decomposition strategy will not only accelerate calculation but also positively affect the likelihood of achieving stabilizing the solution. This has been particularly beneficial for very nonlinear and highly coupled constraint systems, where classic monolithic solvers can fail to converge due to poor initial guesses or instabilities inherent in the numerical nature of the solver.
- **Stability Index:** The stability index quantifies the post-solution error tolerance of the solver, which is the mean residual variation of the satisfied constraints and the target value(s) that exceed the solver's bounds. The obtained percentage improvement of 44 percent in the reported benchmarks means that the results obtained on the solver are more accurate and reliable. It owes this gain to the local solution of small, well-conditioned subproblems prior to being combined into the global solution. With a greater stability index, not only is it more accurate on the geometric level, but it also reduces the occurrence of modeling errors downstream in CAD work, so the solver can be used in the highly sensitive applications of engineering.

5. Results and discussion

5.1. Performance Improvements

As the experimental results show, the suggested solver experiences significant performance gains compared to conventional sequential solving methods, with the largest gains measured in assemblies with more than 500 constraints. The faster parallelization and graph decomposition user does not scale down as well as expected, with less ambitious speedups in

smaller assemblies, due to the overhead of partitioning and thread management taking its toll. Nevertheless, with a bigger problem size, such overheads become overshadowed by computational savings incurred during concurrent processing and localized solving and overall solve time consumption is reduced severely. In particular, on large assemblies whose Constraint Dependency Index (CDI) is high, the decomposition scheme of the solver efficiently parts out dense sets of constraints, enabling a numerically specialized treatment of those, without causing the diversion in convergence much frequented in monolithic solvers. In addition to raw solve time, there is also an improvement in convergence rate and solution stability. By using the structured partitioning approach, the subsequent construction of better-conditioned subproblems is made possible, thereby enhancing the reliability of iterative numerical techniques like Newton-Raphson and BFGS. This was evident in the benchmarks, where the rates of convergence improved by up to 46%, meaning that the solver has reduced the chances of stalling or diverging on complex and nonlinear assemblies.

Besides, the measurement of the stability index (using post-solution residual errors) also indicated a significant increase, which implies that the parallelized code does not just increase computing speed, but also improves the accuracy of the resultant geometry. The results are especially important in current CAD contexts, where assemblies are often composed of hundreds, or even thousands, of constraints. In these contexts, the capability to compromise neither speed nor accuracy can have a direct impact on fewer design iterations or even a quicker time-to-market for engineering projects. Taken together, the results confirm the efficacy of the graph-theoretic decomposition combined with parallel constraint-solving algorithms and evidence that the method scales well, providing a concrete advantage for large-scale and constraint-intensive CAD modelling applications.

5.2. Stability Analysis

Stability is a crucial issue in constraint solving, particularly in cases of CAD assemblies, where ill-conditioned or over-constrained systems can cause solvers to crash or experience numerical divergence, or even result in invalid geometry. The specified technique unites graph-based partitioning and the parallelized solving method that achieved a prominent enhancement in stability on a series of test cases having a high degree of changes. During benchmark tests, the method reduced solver crashes due to over-constrained situations by 37 per cent compared to a standard sequential solver. This significantly contributes to the decomposition approach in addressing problematic areas of the constraints graph and decomposing it into smaller, more manageable partitions. Isolating hyper-constrained clusters in the sub-problem, the solver is free to use specialized numerical methods--constraint relaxation or selective set deactivation--without inducing overall destabilization in the system.What is more, the partitions can be independently solved with parallel processing so that less numeric instability is propagated between one set of problematic constraints and the rest of the model. Besides avoiding overall instability, this localized containment makes the solver much more tolerant to Constraints Dependency Index (CDI) values that would have seen results in instability amplified by cyclic dependencies.

Moreover, the iteration is also provided in the methodology in the form of result aggregation, with partial solutions being re-visited to see if there are errors remaining in them, prior to forming the global solution. This new step of verification allows any instability caused in a partition to be fixed without triggering a cascade of failures in others. These enhancements to stability have particular applicability to real-world CAD projects, where designs can accidentally lead to redundant or conflicting constraints when iterative modeling is used. A solver that behaves gracefully in handling over-constrained systems without a catastrophic crash is of great importance in such an environment to raise reliability in the workflow. Mitigation of the crash rate and high solution fidelity mean that the proposed parallel graph-partitioning experience solves one of the most long-standing constraints of current CAD solvers, and helps achieve solution functionality that is more robust and reliable, even in complicated assemblies.

5.3. Limitations

Although the proposed solver proves to have a definite performance and stability advantage, it is not entirely effective due to its limitations. Among the main drawbacks, one can point to the use of hardware parallelism. More available CPU cores and a tighter memory bandwidth benefit system drive performance in the design. The solver can attain high speedups on machines with many cores because partition processing can occur concurrently; this statistic falls rapidly on low-end or single-core machines. In these applications, the overheads associated with managing threads may put the parallel approach at a disadvantage compared to an optimised sequential solver, even counteracting the improvement brought about by decomposition. The other significant constraint is the preprocessing step involved in graph decomposition. Building the constraint graph, partition finding, and optimal ordering of partitions all have a non-trivial computational cost, especially in cases where the assembly contains thousands of constraints. Even though such preprocessing time is amortized on very large problems, television-sized, it can constitute a huge fraction of overall running times on smaller assemblies, where actual assemblies to solve are quite brief.

More specifically, the efficacy of the decomposition procedure depends on the topology of the constraint graph itself. Because of high cross-dependencies between the constraints (high CDI values), assemblies with highly interconnected constraints can utilise partitioning to identify partitions with significant cross-dependencies, thereby reducing the chances of

truly independent parallel solving. Another assumption the solver makes is that the constraints in the partitions can be sufficiently tackled using the numerical method to be employed (e.g., Newton-Raphson, BFGS). Systems with pathological configurations within partitions, e.g., close to singular systems or constraints that require symbolic reasoning, can perform and be inaccurate. Lastly, although the method enhances soundness in most cases, it does not completely rule out divergence or instability, especially in highly over-constrained systems where conflicts in the designs cannot be addressed without hands-on corrections. Such limitations indicate that the solver can be used effectively in many situations, but it should be used with consideration for the nature of the problem and the amount of available computational resources.

6. Conclusion and Future Work

The present work presents a graph decomposition-based scalable constraint solver framework that leverages graph decomposition and constraint partitioning to significantly enhance the efficiency, stability, and scalability of addressing geometric constraints in CAD settings. Using constraints as a graph, larger and more complicated assemblies can be broken into less critical and smaller subproblems methodically. One then solves these partitions in parallel using specific numerical schemes, such as Newton-Raphson for well-conditioned partitions and quasi-Newton BFGS updates for more nonlinear problems. This structural preprocessing and parallel numerical solving combination has been shown to effectively decrease total solve time, increase convergence rates, and yield more stable solutions—especially on large assemblies and high Constraint Dependency Index (CDI) models.

The paper further presented a reproducible benchmarking framework, founded on synthetic assemblies with controlled complexity and coupling. This provided an ability to measure performance increases, accurately, at different constraint densities and dependence structures. Being conducted on an 8-core Intel Xeon workstation and utilizing the parallelization approach based on OpenMP, the evaluation showed measurable gains: a 29-per cent faster solve time, 46-percent faster convergence rate and 44 per cent higher stability index. Also, the crash rate of the over-constrained settings was decreased by 37%, highlighting the robustness advantages of avoiding troublesome constraint sets when solving the problem. Despite these successes, there are also a few limitations. Performance depends on the hardware parallelism available, and graph decomposition by preprocessing may be expensive with small assemblies. Also, although the technique enhances robustness to a great degree, it does not offer complete prevention of divergence in highly over-constrained systems. These observations suggest that there is an opportunity to further improve and make changes.

Work is underway to move to GPU-based computation, which provides several orders of magnitude more parallelism and may be able to speed up partition solving further, especially on larger assemblies with many non-interdependent subproblems. Intermediate chunks of code programmed in CUDA/ other frameworks might make it possible to run even thousands-of-constraints models in real-time. A more recent development has been the incorporation of the solver into commercial CAD tools, where in-place optimization can be done using constraints in real time, while working on a design interactively. This would enable designers to promptly check the possibility of changes being undertaken with shorter time and greater ease of the overall work, hence resulting in increased efficiency. Additionally, enhancing the solver's support for mixed geometric and kinematic constraints, as well as incorporating adaptive decomposition algorithms, may expand the field where the solver can be utilised, such as simulation, robotics, and parameter-driven product design. Summing up, all these improvements might make the proposed solver a central part of next-generation CAD environments and bring them closer to each other in terms of the trade-offs between computing efficiency and flexibility of design.

References

[1] Michelucci, D., Foufou, S., Lamarque, L., & Schreck, P. (2006, June). Geometric constraints solving: some tracks in Proceedings of the 2006 ACM symposium on solid and physical modeling (pp. 185-196).

[2] Hoffman, C. M., Lomonosov, A., & Sitharam, M. (2001). Decomposition Plans for Geometric Constraint Systems, Part I: Performance Measures for CAD. Journal of Symbolic Computation, 31(4), 367-408

[3] Adleman, L., Cheng, Q., Goel, A., Huang, M. D., Kempe, D., De Espanes, P. M., & Rothemund, P. W. K. (2002, May). Combinatorial optimization problems in self-assembly. In Proceedings of the thirty-fourth annual ACM symposium on Theory of Computing (pp. 23-32).

[4] Anderl, R., & Mendgen, R. (1998). Analyzing and optimizing constraint structures in complex parametric CAD models. In Geometric constraint solving and applications (pp. 58-81). Berlin, Heidelberg: Springer Berlin Heidelberg.

[5] Bettig, B., & M. Hoffmann, C. (2011). Geometric constraint solving in parametric computer-aided design.

[6] Chen, H., Xu, J., Zhang, B., & Fuhlbrigge, T. (2017). Improved parameter optimization method for complex assembly process in robotic manufacturing. Industrial Robot: An International Journal, 44(1), 21-27.

[7] Bettig, B.; Hoffmann, C. M. *Geometric Constraint Solving in Parametric Computer-Aided Design. Journal of Computing and Information Science in Engineering*, Volume 11, Issue 2, Article 021001, June 14, 2011. DOI: 10.1115/1.3593408.

[8] Touloupaki, E., & Theodosiou, T. (2017). Performance simulation integrated in parametric 3D modeling as a method for early stage design optimization—A review. Energies, 10(5), 637.

[9] Nemhauser, G. L. (1994). The age of optimization: Solving large-scale real-world problems. Operations Research, 42(1), 5-13.

[10] Brailsford, S. C., Potts, C. N., & Smith, B. M. (1999). Constraint satisfaction problems: Algorithms and applications. European Journal of Operational Research, 119(3), 557-581.
[11] Fioretto, F., Pontelli, E., & Yeoh, W. (2018). Distributed constraint optimization problems and applications: A survey. Journal of Artificial Intelligence Research, 61, 623-698.
[12] Robertson, B. F., & Radcliffe, D. F. (2009). Impact of CAD tools on creative problem solving in engineering design. Computer-aided design, 41(3), 136-146.
[13] Sarcar, M. M. M., Rao, K. M., & Narayan, K. L. (2008). Computer-aided design and manufacturing. PHI Learning Pvt. Ltd.
[14] Briney, K. (2015). Data Management for Researchers: Organise, Maintain, and Share Your Data for Research Success. Pelagic Publishing Ltd.
[15] Hoos, H. H., Kaufmann, B., Schaub, T., & Schneider, M. (2013, January). Robust benchmark set selection for Boolean constraint solvers, in International Conference on Learning and Intelligent Optimization (pp. 138-152). Berlin, Heidelberg: Springer Berlin Heidelberg.
[16] Griewank, A., Bischof, C., Corliss, G., Carle, A., & Williamson, K. (1993). Derivative convergence for iterative equation solvers. Optimisation Methods and Software, 2(3-4), 321-355.
[17] Anderl, R., & Mendgen, R. (1996). Modelling with constraints: theoretical foundation and application. Computer-Aided Design, 28(3), 155-168.
[18] Sayed, E., Essam, D., Sarker, R., & Elsayed, S. (2015). Decomposition-based evolutionary algorithm for large-scale constrained problems. Information Sciences, 316, 457-486.
[19] Van Holland, W., & Bronsvoort, W. F. (2000). Assembly features in modeling and planning. Robotics and computer-integrated manufacturing, 16(4), 277-294.
[20] Kovács, I., Várady, T., & Salvi, P. (2015). Applying geometric constraints for perfecting CAD models in reverse engineering. Graphical Models, 82, 44-57.