Periodica Polytechnica
Electrical Engineering and Computer Science

58(2), pp. 55-68 (2014)
DOI:10.3311/PPee.2207
Creative Commons Attribution

RESEARCH ARTICLE

Computer-Aided Geometric Sensitivity Analysis of Plane Trusses with a Square Grid Topology

Péter Dóbé / *Gábor* Domokos

RECEIVED 17 JULY 2013; ACCEPTED 10 FEBRUARY 2014

Abstract
A recent theory assigns a single scalar $0 \leq r \leq 1$ to pin-jointed plane trusses, characterising their geometric sensitivity with respect to small displacements of individual joints. Here we investigate how r is varying in $n \times m$ minimally rigid quadrangulations and find that nontrivial patterns correspond both to maximal and minimal values of r.

Keywords
truss · square grid · rigidity · geometric sensitivity · parallel algorithm · supercomputer

Péter Dóbé
Department of Control Engineering and Information Technology,
Faculty of Electrical Engineering and Informatics,
Budapest University of Technology and Economics,
Magyar tudósok krt 2., H-1117 Budapest, Hungary
e-mail: dobe@iit.bme.hu

Gábor Domokos
Department of Mechanics, Materials and Structures, Faculty of Architecture,
Budapest University of Technology and Economics,
Műegyetem rkp. 3., H-1111 Budapest, Hungary

1 Introduction

In structural engineering, a *truss* is a structure consisting of rigid bars connected by pinned joints to each other at endpoints. Trusses are broadly applied as bridges, roofs, transmission towers and other large-scale structures.
The theoretical model of a physical truss consists of one-dimensional rigid *bars* and point-like pinned *joints*. Two or more bars can be connected to each other at endpoints by joints, which allow relative rotation of bars around the axis of the joint. Forces are considered to act on joints only [18].
Realisations of trusses are in most cases *supported*, i.e. fixed to the ground or a wall. If external loads and equilibria of forces are to be taken into consideration, then the theoretical model must reflect the supports. In this case additional joints acting as unmovable support points are introduced. The nature of our current analysis makes it more convenient to have unsupported trusses as input. If necessary, however, it is easy to convert a supported truss into an unsupported one with similar properties [5].

The topology of the truss is described by the undirected *topology graph G(V, E)*. The vertex set *V* of the topology graph is in one-to-one correspondence with the set of joints in the truss. Likewise, the edge set *E* is in one-to-one correspondence with the set of bars. A given vertex is incident with a given edge if and only if the corresponding joint and bar are connected. Geometric information such as coordinates of joints and lengths of bars are excluded from the topology graph.

A truss is *rigid* if the distance of any two joints is constant. A truss is said to be *minimally rigid* if it is rigid, and by removing any bar, it becomes non-rigid. These concepts are illustrated in Figure 1. Minimally rigid trusses are often considered more desirable than non-minimally rigid ones, not only because they require less building material, but also to avoid self-stress caused by kinematic load. For checking the rigidity of a truss, the most straightforward way is the direct mechanical analysis. A statical approach - which is applicable to supported trusses - writes constraints on external forces in equilibrium [18], while a kinematical one writes constraints on the first-order derivatives of the position vectors (i.e. the speed vectors) of joints [21].

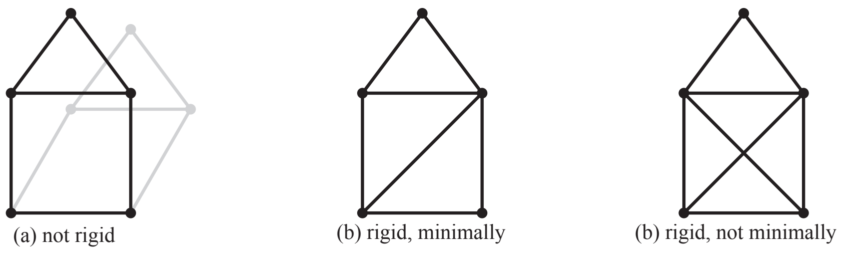

Fig. 1. Simple examples for a non-rigid, a minimally rigid and a non-minimally rigid truss. For the non-rigid truss, a deformed version is shown in grey.

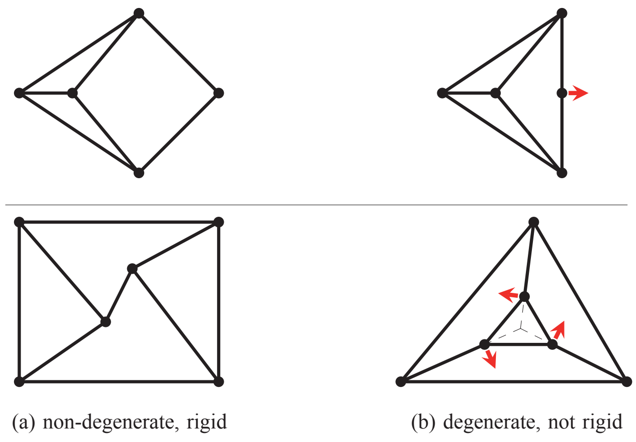

Fig. 2. Pairs of trusses having the same topology but differing in rigidity. In the non-rigid cases, arrows represent possible infinitesimal motions of joints.

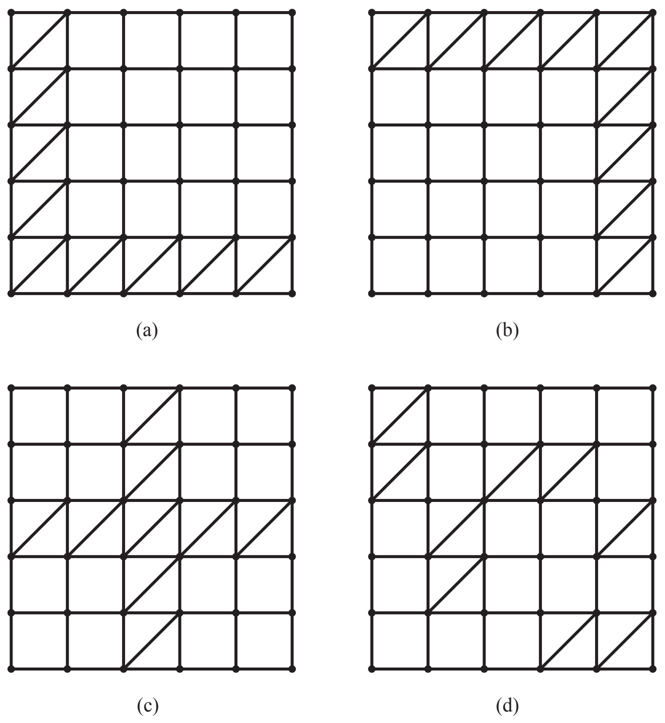

Fig. 3. Four examples out of the 1 087 992 minimally generically rigid 5 × 5 grid truss topologies

An important feature of rigidity and minimal rigidity is that in almost all cases they can be checked by observing only the topology graph G, without regard to geometric data. Topology graphs of rigid trusses are called *generically rigid* graphs, and those of minimally rigid trusses are called *minimally generically rigid* graphs [21]. Based on a theorem by Laman [15], several polynomial-time algorithms have been designed for checking the two-dimensional minimal generic rigidity of a topology graph [16,10]. A more recent one is capable of checking non-minimal generic rigidity as well [3]. These algorithms are suitable for determining the rigidity or minimal rigidity of all plane trusses with the exception of those with a degenerate geometry. Examples for such degenerate geometries can be seen in Figure 2b.

A concept recently introduced is *geometric sensitivity* [20]. A truss is said to be geometrically sensitive if a small relocation of a joint with no external forces applied causes a change in the internal forces in a large part of the truss. The *influenced zone* of an unloaded joint is the subset of bars in which the internal forces change when the given joint is slightly moved. The geometric sensitivity of a truss can be measured by the average size of its influenced zones. Bars of an influenced zone along with joints connected to their ends constitute a rigid truss in its own right [20].

Similarly to rigidity, influenced zones in a non-degenerate truss are a function of the topology graph only: finding them can be reduced to finding minimally generically rigid subgraphs of the topology graph [20]. This has led to algorithms for finding the influenced zones by topology, including the first polynomial-time algorithm for minimally rigid trusses [5] and a more efficient algorithm applicable to non-minimally rigid trusses as well [13]. Section 5 discusses geometric sensitivity analysis in detail.

Our main target of examination is a specific class of unsupported plane trusses: those having a square grid topology with added diagonals in some of the squares (see Figure 3 and Figure 4 for examples). The rigidity and geometric sensitivity of such trusses depend on the number and location of the diagonal bars. We wish to count and select all minimally rigid cases and find the connection between their diagonal bar locations and the geometric sensitivity. The main steps of our analysis are detailed in Section 2.

It is important to note that regarding rigidity, a regular square grid truss (see Figures 4a, 4b and 4c) may have different properties than a grid having the same topology but consisting of non-regular quadrilaterals (see Figures 4d, 4e and 4f). The explanation for this is that concerning geometry, irregular grids are generic, while regular ones are degenerate. Figures 4c and 4f show an example for two trusses with the same topology but different geometries, where only the irregular one is rigid.

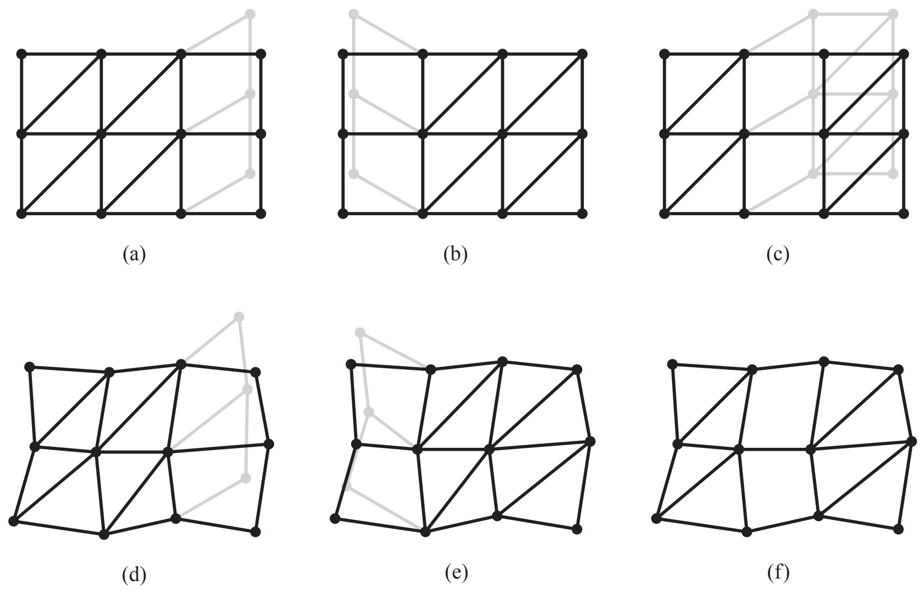

Fig. 4. 2 × 3 degenerate (top) and generic (bottom) quadrilateral grid trusses with matching topologies. For non-rigid trusses, a deformed version is shown in grey. The twelve other possible diagonal bar placements yield rigid trusses in both the degenerate and the generic case.

Degenerate trusses are non-existent in reality, as actual constructed trusses always have small errors, making their geometries generic. As a mathematical model however, degenerate cases are easier to handle. Consequently, the rigidity analysis of degenerate grid trusses is a highly researched topic: results include a method for checking rigidity specialised for such trusses [4]. Based on this, we can even count and directly enumerate all minimally rigid cases, as explained in Section 3.

In this paper, we primarily study generic grid trusses, which are more interesting from an engineering point of view. Being a more difficult problem, the rigidity analysis of such trusses boasts fewer scientific results. No specialised methods for selecting rigid cases are known, so we must resort to numerical solutions, as explained in Section 4. It is clear though that the number of topologies of rigid generic grids is not less than the number of topologies of rigid degenerate ones, since if a degenerate truss is rigid, then all generic trusses having the same topology are rigid as well.

To calculate the geometric sensitivity of generic grids, we use a general-purpose topology-based method, which assumes genericity of the input truss. The geometric sensitivity of degenerate grids is out of the scope of this paper, but it is known that a degenerate truss is never more sensitive than a generic truss with the same topology [20].

Due to the large number of trusses to be processed, the analysis is highly compute-intensive. This computational requirement will be served by a high performance computing system, detailed in Section 8.

2 Goal and steps of examination

We intend to analyse minimal rigidity and sensitivity properties of trusses constrained to a two-dimensional plane, having a square grid topology, with diagonal bars placed in some of the squares. Only the topology of the truss is at our disposal as input data, no geometric information is used. Examples for such topologies can be seen in Figure 3.

The first stage of our examination is to find the diagonal bar combinations that yield minimally rigid trusses for different grid sizes. A direct enumeration of all such cases would be the most suitable solution to this. If that is not possible, enumeration of a larger superset of cases and filtering out the minimally rigid cases is also an option. Sections 3 and 4 discuss how to do it for degenerate and generic grids, respectively. For the latter, less researched type of grids, the number of minimally rigid cases of a given size is an additional result of interest.

The second stage is to calculate the geometric sensitivity of each minimally rigid non-degenerate case, observing how sensitivity depends on the locations of diagonal bars. Two specific outcomes of this sensitivity analysis are most interesting for us. First, we wish to find the configurations of diagonal bars resulting in the least and most sensitive grids. Second, we would like to obtain distribution plots of the geometric sensitivity values of all trusses analysed.

We take into account only trusses with diagonals slanted in a single direction, for example where each one is aligned along a "southwest-northeast'' line, as illustrated in Figure 3. However, it can be easily seen that changing the orientation of the diagonals does not affect rigidity, and the difference in sensitivity becomes infinitely small as the size of the grid approaches infinity, so this is not a significant limitation.

The topology of the grid truss can be described by the height and width of the grid along with the identifiers of squares containing a diagonal bar. This form of input leads to redundant cases as some of the topologies are isomorphic (e.g. Figures 3a and 3b, or Figures 4a and 4b). Nevertheless, such isomorphy classes within the whole input domain contain no more than four topology graphs, regardless of the size of the grid. Such a small level of redundancy does not necessitate an algorithm or any other means for filtering the isomorphic topologies.

Due to the lack of known analytical solutions for the problem, we follow a numerical approach, checking rigidity and calculating geometric sensitivity with software implementations of efficient algorithms. As the number of rigid square grids of a given size is an exponential function of the height and width of the grid, our analysis is limited to small trusses. Even for small trusses the number of cases is too large to be handled on a single processor core, therefore we also aim to parallelise the problem formally and use a distributed system in order to gain speed. Distributed high performance computing (HPC) infrastructures that can suit our goals include a computing cluster or supercomputer with a job scheduler (such as Condor [19]) installed, or a computing grid [9] with a modern middleware (such as ARC [2]) deployed. In Section 6, we give an overview of parallelisation possibilities, and in Section 7 we formulate performance metrics for the chosen approach.

3 Counting and enumerating minimally rigid degenerate grids

Regular square grids have degenerate geometries, so the minimal generic rigidity of their topology graphs does not necessarily mean that such trusses are rigid (see Figure 4c for a counterexample). Therefore the algorithms for checking minimal generic rigidity cannot be applied to these cases.

Instead, we can use a result by Bolker and Crapo [4] to test the minimal rigidity of degenerate grids by topology. For this, we construct an auxiliary bipartite graph $H(A, B, F)$. The vertices in set A and set B correspond to rows and columns of the grid respectively. We place an edge $(i, j) \in F$ for $i \in A, j \in B$ if and only if there is a diagonal bar in the intersection of the row matching i and the column matching j. According to observations by Bolker and Crapo, the square grid truss is rigid if and only if $H(A, B, F)$ is connected, and minimally rigid if and only if $H(A, B, F)$ is a tree. This can be easily checked for example with a breadth-first search. Figure 5 demonstrates that using this technique we can make sure that a square grid truss with the topology shown in Figure 3d is in fact minimally rigid.

This connection between minimally rigid cases and bipartite graphs forming trees leads to further information. For example, since the number of spanning trees of the complete bipartite graph $K_{n,m}$ is $n^{m-1} m^{n-1}$, this is also the number of minimally rigid degenerate $n \times m$ square grids.

Hurlbert provides an encoding of spanning trees in a complete k-partite graph [12], which is similar to Prüfer's encoding of spanning trees in the complete graph [17]. This encoding is a one-to-one mapping between the spanning trees and tuples of integers. The mapping can be efficiently performed in both directions, and each tuple can be easily converted to a unique integer. Utilising this encoding for $k = 2$ we can map the set of

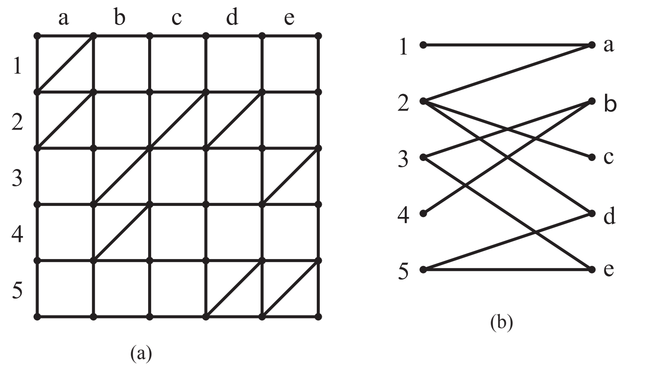

Fig. 5. A minimally rigid square grid (a) and its corresponding auxiliary bipartite graph (b)

all $H(A, B, F)$, $|A| = n$, $|B| = m$ bipartite trees - and thus the set of all $n \times m$ minimally rigid square grid trusses - to an interval of integers. This allows us to enumerate all minimally rigid cases, and also to partition the input domain of the sensitivity analysis to disjoint subdomains, which is invaluable aid in distributed execution, as explained in Section 6.

As mentioned in the Introduction, instead of degenerate square grids we focus on grids consisting of irregular quadrilaterals, which have non-degenerate, generic geometries. For enumerating all rigid cases among them, the method presented above would provide incomplete results: although all generic grids having the enumerated topologies are rigid, cases such as Figure 4f are excluded. Consequently, we need different techniques, detailed in the next section.

4 Filtering minimally rigid generic grids

A quadrilateral grid truss where joints are randomly placed on the plane with algebraically independent position vectors has a non-degenerate geometry. Therefore, in order to test whether such a truss is minimally rigid, we can check the minimal generic rigidity of its topology graph with one of the polynomial-time generic rigidity checking algorithms.

However, there is presently no known way to enumerate all minimally generically rigid grid topologies, or even to calculate their number with a closed-form expression. Nor is there a known algorithm for checking minimal generic rigidity specialised for square grid topologies that is more efficient than the general purpose ones. The following was the only specific connection we found:

Statement 1. *An $n \times m$ square grid topology graph with d diagonal edges is minimally generically rigid if and only if*

1. *$d = n + m - 1$, and*
2. *for all $n' \times m'$ rectangular areas of the grid where d' is the number of diagonal edges, $d' \leq n' + m' - 1$.*

This can be proven using Laman's theorem [15] and Henneberg's "vertex addition" step [11]. Since even these two rules were insufficient to establish a way for enumeration, the only possibility is to filter the minimally generically rigid graphs out of a larger set of cases. To do this, we traverse all potentially minimally generically rigid cases, test the rigidity of each with a fast algorithm, and only if the test is passed do we proceed with the sensitivity analysis.

The total number of possible diagonal placements in an $n \times m$ grid is 2^{nm}, as for each square we can decide whether to place a diagonal bar or not. On the other hand, by taking advantage of Rule 1 in Statement 1, the input domain of the rigidity test can be reduced to $\binom{nm}{n+m-1}$ cases, which are all possible placements of $n + m - 1$ diagonals in the grid. To enumerate all such combinations of diagonals, we can use the *combinatorial number system* [14], which assigns a unique integer between 0 and $\binom{nm}{n+m-1} - 1$, called *rank*, to each combination. From the rank, the truss topology can be easily recreated. This mapping to a contiguous interval of integers also helps in the distributed execution of the rigidity testing.

5 Sensitivity analysis

To get a picture on the geometric sensitivity of a truss confirmed to be rigid, the influenced zones of all joints have to be determined. For this, we can take advantage of the following connection [20]:

Theorem 1. *For non-degenerate geometries, the bars in the influenced zone of a selected joint correspond to the edges in the* rigid core *of the vertex in the topology graph matching the joint.*

The rigid core of a vertex $v \in V$ in a minimally generically rigid topology graph *G(V, E)* is defined as

- the empty graph or the graph $(\{v\}, \emptyset)$ if $d(v) < 3$ (the *trivial case*);
- the smallest minimally generically rigid induced subgraph of *G(V, E)* that contains v and all its neighbour nodes if $d(v) \geq 3$.

where $d(v)$ denotes the degree i.e. the number of neighbour nodes of v. It should be noted that we have provided two alternative definitions for the trivial case. According to the original definition formulated in [20], the trivial rigid core is the empty graph. However, during our calculations, we included the single node v in the rigid cores of 2-degree nodes. In our current case, the choice of definition does not affect the sensitivity measurement significantly, since a quadrilateral grid never has more than four trivial rigid cores.

A naive way to find a rigid core of a vertex with at least 3 neighbours is to exhaustively seek through all induced subgraphs containing v and its neighbours in increasing order of the number of additional nodes, check its minimal generic rigidity, and stop at the first minimally generically rigid subgraph. This naive algorithm can not be used in practice, as it has exponential time requirement. An earlier result of ours is a proof that the problem of finding the rigid core of a node in a minimally generically rigid graph has polynomial time complexity [5]. Our proof provides a polynomial time algorithm, which is based on the minimisation of an appropriate submodular target function similar to that used in rigidity checking [10]. A more efficient, simple algorithm [13] applies the arc reorientation algorithm [3] to find cycles in the rigidity matroid of *G(V, E)*. With a different definition of rigid cores for non-minimally generically rigid graphs, Theorem 1 holds for non-minimally rigid trusses. Taking advantage of this, the matroid-based method can be extended to enable finding rigid cores in non-minimally rigid trusses as well [13].

As a scalar measure of how sensitive a truss is, we can use the *joint sensitivity index*: the average ratio of the number of joints in the influenced zone to the total number of joints [20]:

$$r = \frac{1}{|J|} \sum_{j \in J} \frac{|Z(j)|}{|J|} \tag{1}$$

where J is the set of all joints and $Z(j)$ is the set of joints in the influenced zone of j. For an $n \times m$ grid truss, we get

$$r = \frac{\sum_{j \in J} |Z(j)|}{(n+1)^2 (m+1)^2} \tag{2}$$

The index equals or is close to 1 for "very sensitive" trusses where the influenced zones of most or all joints equal or almost equal the whole truss. To obtain this index, we need to calculate the sizes of the influenced zones of all joints. According to the discussion above, for typical geometries this can be done by applying the most practical rigid core finding algorithm $|V|$ times.

6 Parallelisation approaches

In case of computing problems involving heavy calculation or processing large data, speedup can be gained by using parallel computing techniques. Our analysis of trusses, being computationally intensive, can also benefit from parallel execution.

To design parallel applications, one can choose among several parallel machine models. One such model is the *parallel random-access machine* (PRAM) [7], which assumes an unlimited number of processors and unlimited shared memory with uniform access time. A more realistic model, the *multicomputer* [8] consists of a finite number of autonomous von Neumann machines interconnected by a network. Exchange of data is done by sending messages instead of using a shared memory. Communication is considered expensive, and data access times are taken into account in performance analysis. This model is closer to most actual general-purpose supercomputers, such as those used for our research, described in Section 8.

The three problems mentioned in Section 2 - counting rigid cases, finding the least/most sensitive grids and getting the distributions of sensitivities - are closely related. Each involves the rigidity check of a huge number of trusses, and the two sensitivity-related problems also require calculating the geometric sensitivities of the rigid ones. Therefore, to develop a parallel application, similar considerations are needed for all three problems.

When considering parallelising the problem of analysing the rigidity or sensitivity of several trusses, one immediately sees that different trusses can be processed independently. Such *embarrassingly parallel* problems [8] are frequently encountered in science and engineering. For these problems, the *parameter study* (or *parameter sweep*) approach can give a simple parallelised solution.

A parameter study algorithm on a multicomputer can be described as follows. An *input task* partitions the input domain of the problem into subdomains. There are several *worker tasks*, each of which requests the parameters of a subdomain from the input task via message passing. A worker task processes a subdomain, and sends the result to an *output task*, then it requests the parameters of another subdomain to process. The output task is responsible for aggregating the subresults received from the workers into the final result of the problem. Each of these tasks can be assigned (*mapped*) to a separate abstract machine in the multicomputer for maximum performance if possible.

In our case, the input domain is the set of topologies of quadrilateral grid trusses to be examined. In Section 4 we explained how we map this set to an interval of integers, which is straightforward to partition into non-overlapping subintervals. The operation carried out by the worker tasks includes the rigidity analysis of the trusses corresponding to each integer in the subinterval. For the two more complicated problems, the sensitivity index of each rigid truss needs to be calculated, too. Depending on the problem, the output of the workers is the number of rigid cases within the subdomain, the partial top/bottom lists or the partial distributions. After all workers are done, the final aggregating step is performed. When counting rigid cases or calculating the distributions of sensitivity indexes, aggregating means simply adding up the counts for the subintervals. When seeking the most and least sensitive trusses, overall top and bottom lists can be assembled by ranking the sensitivity values in the lists for the subintervals.

In a parameter study application, the worker tasks can run simultaneously and independently, with no communication or synchronisation required. Communication is only needed between the input task and worker tasks, and between worker tasks and the output task. These applications are therefore well suited for widely used general-purpose HPC systems such as computing clusters, computing grids or supercomputers. In these infrastructures, a *job scheduler* (or *batch system*) is responsible for the coordination of computation *jobs*. Software development tools such as Saleve [6] exist to aid the creation of parameter study applications and their deployment in various HPC systems.

In our parallel application design, the role of the input task is played by the combination of a simple program and the job scheduler. The program splits the interval representing the whole input domain into subintervals. Jobs performing the actual analysis of trusses are enqueued in the scheduler, each one getting the start and end of a subinterval as input. The worker tasks are represented by the processors of the distributed system that are available for executing the analysis of a subdomain. When a processor is available, the scheduler launches a new job doing the truss analysis. This is how the communication between the input task and the workers is implemented. The output of the worker tasks is written to files on a shared file system. A simple aggregating program running offline (i.e. without involvement of the scheduler) acts as the output task, reading and processing the contents of these files. Communication is done by disk operations here.

There can be alternative approaches to consider in addition to our chosen design. It should be noted for example that when analysing the sensitivity of a truss, the influenced zones can also be determined independently without communication. If our input domain is the set of topology nodes instead of the set of topologies, then we can make a much finer-grained partitioning for sensitivity-related problems, having more subdomains than trusses. However, this level of granularity is not reasonable here for several reasons. We deal with a very large number of relatively small trusses, so the high demand for computation power is due to the quantity of trusses. The analysis of a small truss takes only a short time even on a single medium-performance CPU core, so there is no use decomposing it into subtasks. Refining the granularity of a parameter study beyond a certain point reduces efficiency, as we will show in the next section. Also, a domain partitioning like that would require more complicated input and output tasks for our problems. Such fine-grained decomposition would be justified in other problems of research, for example when analysing the sensitivity of a few very complex trusses having several thousand nodes.

There are other ways to parallelise the analysis of a truss in a non parameter study manner. For example, both the rigidity checking and the rigid core finding algorithms include a breadth-first search step [3], which can be replaced by a parallel search algorithm [8]. We discarded this idea as it would have led to an unacceptable amount of communication and unnecessarily fine granularity.

The input task can be parallelised as well. The sequential algorithm for splitting the interval $\{0, \ldots, N-1\}$ representing the input domain into J subintervals requires a division and $J-1$ other operations (additions or multiplications) in order to calculate interval boundaries. In theory, this could be accelerated with a divide and conquer strategy. One can show using a PRAM model of $J-1$ processors that the interval splitting can be done in just two steps: first we calculate $l = \left\lceil \frac{N}{J} \right\rceil$, then each processor with identifier i (where $i \in \{1, \ldots, J-1\}$) calculates il.

Similarly, we can provide a theoretical acceleration of the aggregation stage as well. If the aggregation means summing, then a PRAM with a sufficient number of processors can do this in $O(\log J)$ time [7]. If the aggregation means merging top or bottom lists into a single list of k values, then a naive PRAM algorithm can perform this in $O(k)$ time by finding a minimum or maximum k times.

In reality, there is no speedup due to the communication overhead brought in. This is not a problem, as the splitting and the aggregation take only a very short time on a single core even in case of several thousand subdomains. For the same reason, we chose not to overlap the operation of the output task with the worker tasks, and instead start the aggregation in an offline, separate step only when all the subresults are produced.

7 A performance model for parameter study

In this section we provide a performance analysis for our parameter study algorithm. The goal is to tune parameters of the parallelisation for optimal performance. What we can decide in our parameter study design is how to split the input domain. To make splitting simple, we choose equal-sized subdomains. Specifying the size of a subdomain is then equivalent with specifying the number of subdomains (J). An important question is how the choice of J affects the speedup of parallel execution.

We make further simplifications in the performance model. Execution times of the domain splitting and aggregation are safe to omit, as they are negligible compared to the time required for the actual truss analysis. We also use average estimates for some varying parameters.

The *execution time* or *wall-clock time* of our parallel application is the time elapsed between the submission of jobs into the scheduler and the completion of the last job. Without taking into account the time required by the domain splitting and aggregation, the execution time can be calculated as the total time all processors spend communicating, computing and doing nothing during this period divided by the number of processors [8]:

$$T_P = \frac{1}{P}(T_{\text{comm}} + T_{\text{comp}} + T_{\text{idle}}) \tag{3}$$

Here P is the number of momentarily available processor cores in the distributed system that are utilisable for worker tasks. As mentioned before, the analysis of a single truss is done sequentially, so our application cannot benefit from having more processors than the number of trusses, N. Thus our calculations apply only for $P \leq N$, which is the case in practice.

It is also important that the exact value of P is not known in advance, because it depends on other scientific applications occupying resources of the system. It even changes dynamically during execution. We can only make a rough prediction of an average value taking this into account.

Communication time T_{comm} means most importantly the scheduler overhead of launching a new job. To a lesser degree, it also includes disk I/O operations on the shared file system. The average communication time per job, here denoted by t_s, is considered independent of other parameters. For J jobs, the total time spent communicating is:

$$T_{\text{comm}} = Jt_s \tag{4}$$

The computation time T_{comp} is the time spent analysing trusses. To check the rigidity of trusses of the same size, approximately the same amount of time is required for each one. This also means close to equal run times for all jobs. When sensitivity analysis is involved as well, the required time has a high variation: it is very small if the truss is not rigid.

We denote the average computation time per truss with t_c. This can be measured by choosing a small sample subinterval from the input interval, measuring the run time of the analysis on this subinterval, and dividing it by the sample size. For sensitivity-related problems, the sample should be chosen from around the middle of the input interval. Otherwise, most or all of the trusses in the sample may be non-rigid, resulting in a misleadingly small estimate for t_c.

If the whole input set contains N trusses, then the total computation time is:

$$T_{\text{comp}} = Nt_c \tag{5}$$

Idle time T_{idle} is the period during which a processor is free, but does not get a subdomain to process. In our parameter study application, execution of a new job can start as soon as another one is complete, as long as there are unprocessed subdomains. If $J \geq P$, then processors become idle only when running out of subdomains.

To estimate idle time for $P < J \leq N$, we consider a worst case scenario, where all worker tasks complete work on their subdomains simultaneously, and there is only one subdomain left. If this happens, then a single processor computes on the remaining subdomain, while the other $P - 1$ processors are idle. The length of a subinterval is approximately $\frac{N}{J}$, so the estimated worst case total idle time is:

$$T_{\text{idle}} = (P-1)\frac{N}{J}t_c \tag{6}$$

This worst case cannot be prevented, because as stated, P changes dynamically. Also, despite our best efforts to create jobs with equal run time, there will inevitably be variances. In addition, t_s may not be constant as assumed. This makes job completion times impossible to predict.

For the total execution time, we therefore get

$$T_P = \frac{1}{P}\left(Jt_s + Nt_c + (P-1)\frac{N}{J}t_c\right) \tag{7}$$

The *efficiency* of a parallel algorithm is defined as the fraction of time that processors spend doing useful work [8]. If T_1 is the execution time of a sequential application of the same purpose, then the efficiency of our parameter study application is:

$$E = \frac{T_1}{PT_P} = \frac{Nt_\mathrm{c}}{Jt_\mathrm{s} + Nt_\mathrm{c} + (P-1)\frac{N}{J}t_\mathrm{c}} \tag{8}$$

The *speedup* describes how many times faster the parallel application is compared to a sequential implementation [8]:

$$S = \frac{T_1}{T_P} = PE \tag{9}$$

A truss is analysed with our application in a sequential manner, therefore according to Amdahl's law [1], its speedup for N trusses is not greater than N.

Speedup increases with the number of processor cores, up to the point where $P = N$. The maximal speedup could be achieved with N jobs on N processors, each job analysing one truss:

$$S_\mathrm{max} = \frac{T_1}{T_N} = \frac{Nt_\mathrm{c}}{t_\mathrm{s} + t'_c} = \eta N \tag{10}$$

where t'_c is the longest duration of the analysis of a single truss, which is significantly greater than the average duration t_c, because non-rigid trusses take only a short time to process. Due to this fact and the scheduler overhead, $\eta < 1$ i.e. the limit of the speedup is somewhat less than the problem size N.

Having a limited number $P \ll N$ of processor cores, we aim for maximum efficiency, which leads to the maximum speedup for the given P value. An optimal number of jobs maximises E:

$$J_\mathrm{opt} = \arg\max_J E = \arg\min_J T_P = \sqrt{\frac{(P-1)Nt_\mathrm{c}}{t_\mathrm{s}}} \tag{11}$$

If we provide sufficiently accurate estimates for P, t_c and t_s, then with the correct number of jobs we can attain an efficiency close to the theoretical maximum, which is:

$$E_\mathrm{max} = \frac{Nt_\mathrm{c}}{Nt_\mathrm{c} + 2\sqrt{(P-1)Nt_\mathrm{c}t_\mathrm{s}}} \tag{12}$$

A parallel algorithm is *scalable* if its efficiency does not deteriorate much as the size of the input domain and/or the computing infrastructure increases. Scalability-wise our parameter study performs well: the maximum efficiency actually increases with increasing N. With constant N, the decline in efficiency is roughly proportional to the square root of the number of available processors.

So far we have considered the effects the domain splitting and aggregation have on the execution time as insignificant. If we wish to take these sequential stages into account as well, then Formula 7 for the execution time of the parallel algorithm should be modified as follows:

$$T_P^* = T_\mathrm{split} + T_\mathrm{aggr} + \frac{1}{P}\left(Jt_\mathrm{s} + Nt_\mathrm{c} + (P-1)\frac{N}{J}t_\mathrm{c}\right) \tag{13}$$

As stated in Section 6, the essence of splitting the input domain is $J-1$ operations, and in case of counting rigid trusses, the aggregation means adding up J numbers. Therefore, we may use $T_\mathrm{split} \approx c_1 J$ and $T_\mathrm{aggr} \approx c_2 J$ as estimates. Assembling top and bottom lists from partial lists of k elements each has time requirement $O(Jk\log k)$. However, we have set k to a small constant, 20, so we may estimate the time needed by the aggregation as $T_\mathrm{aggr} \approx c'_2 J$.

When calculating distributions of sensitivity indexes, it might be desirable to have more detailed distribution data for larger trusses. This also requires more values to add up in the aggregation stage as we increase the height n and width m of the grid. However, even if we decide to produce the most detailed distributions possible (as explained in the next section), the number of additions is only $O(Jn^2m^2)$. At the same time, the problem size N increases exponentially with n and m. Therefore, the estimated time required to aggregate partial distributions is $T_\mathrm{aggr} \approx c''_2 J \log^2 N$.

Accordingly, in order to consider the effects of splitting and aggregation when determining the optimal number of jobs, a modified version of Formula 11 should be used:

$$J^*_\mathrm{opt} = \sqrt{\frac{(P-1)Nt_\mathrm{c}}{t_\mathrm{s} + P\alpha}} \tag{14}$$

where $\alpha = c_1 + c_2$ for counting rigid cases, $\alpha = c_1 + c'_2$ for listing the least/most sensitive trusses, and $\alpha = c_1 + c''_2 \log^2 N$ for the sensitivity distribution problem. The maximal efficiency will then be

$$E^*_\mathrm{max} = \frac{Nt_\mathrm{c}}{Nt_\mathrm{c} + 2\sqrt{(P-1)Nt_\mathrm{c}(t_\mathrm{s} + P\alpha)}} \tag{15}$$

In practical cases where $J \ll N$, the influence of the domain splitting and aggregation stages on the time requirement or efficiency is not noticeable, so it is safe to use Formula 11 instead, in order to keep the model simple. These two stages are only significant when calculating the theoretical maximum of the speedup, where $J = P = N$. Formula 10 shall here be modified as

$$S^*_\mathrm{max} = \frac{Nt_\mathrm{c}}{N\alpha + t_\mathrm{s} + t'_c} = \frac{t_\mathrm{c}}{\alpha}\left(1 - \frac{t_\mathrm{s} + t'_c}{t_\mathrm{s} + t'_c + N\alpha}\right) \tag{16}$$

This means that in case of the rigid counting and listing problems, maximal speedup has an upper bound independent of N:

$$S^*_\mathrm{max} < \lim_{N\to\infty} S^*_\mathrm{max} = \frac{t_\mathrm{c}}{\alpha} \tag{17}$$

It should also be noted that for the distribution problem, α is a function of J, thus also of N. As a consequence, it can be seen from Formula 16 that with increasing N, the speedup will actually decrease beyond a certain point.

8 Execution on supercomputers

To process all potentially minimally rigid grids of size $n \times m$ for small n and m values, we had two supercomputers at our disposal. We began with using *Hercules*, the supercomputer at

Miskolc University, and later tasks were carried out on *Superman*, the supercomputer of Budapest University of Technology and Economics. These two computers have similar configurations: each consists of 30 nodes, with 12 CPU cores per node. A fast interconnect network makes communication possible among nodes. Each node has access to a distributed file system for convenient data sharing. The Condor scheduler [19] is installed, providing fair share of computing resources among various research projects.

We implemented the parameter study design presented in Section 6, and deployed it on the supercomputers. The truss analysis relies almost exclusively on integer arithmetic, so the CPU performance was the most important parameter of the system. Other factors such as communication bandwidth, storage space and disk access speed are less relevant for us. For trivially small trusses where n and m are not over 4, we used a single-core desktop machine instead of the supercomputers, without splitting the input domain.

First, we wished to count the minimally rigid $n \times m$ generic grid trusses. For this, a fast C++ program was developed, which got the start and end of an integer interval as command line arguments. This interval specified a set of $n \times m$ grids, each having $n + m - 1$ diagonals. The program performed Tibor Jordán's rigidity matroid algorithm [3] for checking rigidity. The number of minimally rigid cases in the input set was written to the standard output.

Second, we wanted to see the most and least sensitive grid trusses. A modified version of the program mentioned above, after verifying rigidity, calculated the rigid cores and joint sensitivity index as well, using an algorithm also based on the rigidity matroid [13]. It produced as output the list of the twenty least and most sensitive grids in the input set.

After this, we were interested in the distribution of sensitivity index values. We can observe from Formula 2 in Section 5 that the joint sensitivity index r cannot take more than $(n + 1)^2(m + 1)^2$ possible values. We are discussing small grids, therefore this is not a considerably large number, allowing us to count trusses having index r for each occurring r value. We can make a distribution plot from the result, which is detailed and illustrative at the same time, without the need of creating a histogram with manually adjusted histogram bins. We once again modified our program to perform this counting on a subdomain.

For a given n and m, instances of these programs were launched as jobs, each getting a subinterval of $\left\{0,\ldots,\binom{nm}{n+m-1}-1\right\}$ to process. A simple auxiliary program performed the interval splitting, and generated a *submit description file* for the Condor scheduler.

To determine the optimal number of jobs, we did not use Formula 11, as we did not have appropriate estimates for the parameters within yet. Instead, we heuristically tuned the number of jobs so that each one took a couple of hours to finish, resulting in a few thousand jobs.

After all jobs finished, a separate program aggregated the subresults to obtain the final results. This program read in the output files of all jobs from the shared file system, and performed the summing or list merging, depending on the problem type.

9 Results

Using the supercomputer Hercules, first we determined the number of topologies corresponding to minimally rigid generic grid trusses or various $n \times m$ grid sizes. Table 1 summarises these results in increasing order of the number of diagonals (denoted by d). In the table, rows for grids with equal height and width are highlighted. Results above the horizontal line were obtained with a single-core computer, before using supercomputers.

Column 4 shows the total number of topologies checked for minimal generic rigidity. This is the number of all possible placements of $d = n + m - 1$ diagonals. Column 5 shows the number of cases found to be minimally generically rigid. For this value, no formula is currently known, but it obviously cannot be less than the number of topologies for minimally rigid degenerate square grids (shown in Column 6), which can be calculated as described in Section 3.

We can see in Row 2 that among 2×3 grids, there are exactly two topologies for non-rigid generic grids, and three topologies (which include the former two) for non-rigid degenerate grids. These are represented by the generic grid trusses in Figures 4d and 4e, and by the degenerate grid trusses in Figures 4a, 4b and 4c.

The calculations on the supercomputer proceeded up to $d = 13$, sweeping through several hundred billion cases within a few days. Afterwards, we launched further tasks on Hercules to find the twenty least and most sensitive grid trusses among the minimally rigid ones, for $n \times n$ grid sizes up to $n = 6$. We found that in the least sensitive grids, the diagonals were placed in row i, column j so that either $i \approx \left\lfloor \frac{n}{2} \right\rfloor + 1$ or $j \approx \left\lfloor \frac{n}{2} \right\rfloor + 1$. This means that the diagonals formed a "+"-like shape in the middle of the truss. For example, the topology of the least sensitive 5×5 grid, having joint sensitivity index 0.280864, can be seen in Figure 3c. Among the most sensitive ones, there appeared to be a large number of cases (although no more than twenty were recorded) all having the same joint sensitivity index, many of them having no particular pattern in the distribution of diagonals.

These results have been summarised in a downloadable appendix[1]. In the appendix, topologies of the twenty most and least sensitive trusses are shown for different grid sizes, grouped by joint sensitivity index value. For improved visibility, diagonal bars are represented by squares filled with red. The rank of the diagonal combination is given below each diagram.

1 http://sirkan.iit.bme.hu/~dobe/truss/sens-appendix.pdf

Tab. 1. Number of topologies of minimally rigid $n \times m$ grid trusses with d diagonals in case of generic and degenerate geometry

d	n	m	**Total** $\binom{nm}{d}$	**Rigid generic** ?	**Rigid degenerate** $n^{m-1}m^{n-1}$
3	2	2	4	4	4
4	2	3	15	13	12
5	2	4	56	40	32
5	3	3	126	98	81
6	2	5	210	121	80
6	3	4	924	623	432
7	2	6	792	364	192
7	3	5	6 435	3 685	2 025
7	4	4	11 440	7 116	4 096
8	2	7	3 003	1 093	448
8	3	6	43 758	21 048	8 748
8	4	5	125 970	70 584	32 000
9	2	8	11 440	3 280	1 024
9	3	7	293 930	118 053	35 721
9	4	6	1 307 504	649 776	221 184
9	5	5	2 042 975	1 087 992	390 625
10	2	9	43 758	9 841	2 304
10	3	8	1 961 256	655 625	139 968
10	4	7	13 123 110	5 729 556	1 404 928
10	5	6	30 045 015	14 888 525	4 050 000
11	2	10	167 960	29 524	5 120
11	3	9	13 037 895	3 621 234	531 441
11	4	8	129 024 480	49 211 996	8 388 608
11	5	7	417 225 900	189 681 461	37 515 625
11	6	6	600 805 296	288 906 180	60 466 176
12	2	11	646 646	88 573	11 264
12	3	10	86 493 225	19 939 899	1 968 300
12	4	9	1 251 677 700	415 712 461	47 775 744
12	5	8	5 586 853 480	2 308 746 354	320 000 000
12	6	7	11 058 116 888	5 063 020 974	784 147 392
13	2	12	2 496 144	265 720	24 576
13	3	11	573 166 440	109 606 097	7 144 929
13	4	10	12 033 222 880	3 473 854 720	262 144 000
13	5	9	73 006 209 045	27 259 652 307	2 562 890 625
13	6	8	192 928 249 296	83 016 541 716	9 172 942 848
13	7	7	262 596 783 764	118 247 295 490	13 841 287 201

An upper bound on the joint sensitivity index can be calculated based on the maximal possible sizes of influenced zones. Joints in the four corners of the grid always have small influenced zones. Other joints can have very large influenced zones, but they exclude one or both of the top-left and bottom-right corner joints. Our experiments show that the joint sensitivity indexes of the most sensitive grid trusses reach this theoretical maximum.

Due to the abundance of such maximal sensitivity trusses, one can experimentally find combinations of diagonals corresponding to some of these cases for larger grids as well. 7×7 examples, having joint sensitivity index 0.911621, can be seen in Figure 6. Some of these, such as 6a, 6b and 6c, show a pattern in the placement of the diagonals, while others, such as 6d appear to be random.

We used the supercomputer Superman to get the distributions of joint sensitivity index values of $n \times n$ grids. The complete analysis could be done within reasonable time only for sizes up to 6×6. However, we created distribution plots for larger grids based on a small subset of the input domain, with the assumption that the characteristic shape of the plot is similar to that

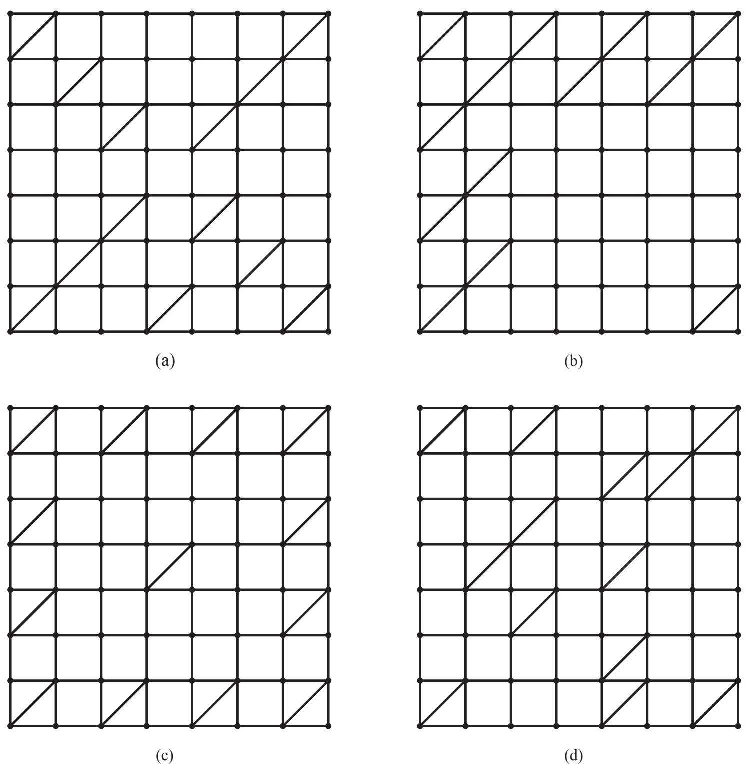

Fig. 6. Four examples of maximal sensitivity 7 × 7 grid truss topologies

Tab. 2. Run time statistics of the most demanding task types.

Computer used	Task purpose	Grid size	#jobs	Average CPU time per job	Total CPU time	Wall clock time
Hercules	rigid count	7 × 7	4377	05:08:16	937d 00:42:15	3d 08:00:00
Hercules	top/bottom sensitivities	6 × 6	301	03:09:33	39d 14:58:42	05:40:19
Superman	*r* distribution (complete)	6 × 6	301	1d 00:31:13	307d 12:37:15	3d 06:31:27
Superman	*r* distribution (partial)	9 × 9	12845	01:20:56	722d 01:18:16	4d 21:27:54

of the whole domain. Here we picked every kth subdomain to process, where $k = 500$ for 7 × 7, $k = 250000$ for 8 × 8, and $k = 500000000$ for 9 × 9 grids. The distribution plots can be seen in Figure 7.

Experiences regarding run times of the hardest tasks are summarised in Table 2. Here, "Wall clock time" is the time elapsed between submitting the jobs and the completion of the last job. From the column labelled "Total CPU time", it is clear that our analysis would have taken several years using a single processor core dedicated exclusively to these tasks.

Compared to other types of tasks, counting minimally generically rigid truss topologies is relatively fast, so we could examine several different grid sizes, up to 7 × 7. Row 1 of Table 2 shows the time requirement for the latter.

Not surprisingly, making the top/bottom sensitivity lists is much slower, since the sensitivity analysis involves finding the rigid cores of all $(n + 1)(m + 1)$ nodes in the truss, each having a time requirement roughly equivalent to that of the rigidity check. Therefore, the size limit of our examinations was 6 × 6 here, for which the time requirement can be seen in Row 2 of Table 2.

It is more interesting that compared to making the lists, getting the joint sensitivity index distribution for 6 × 6 grids took considerably more time (see Row 3 of Table 2). In both tasks, calculating the sensitivity value is expected to be the major part of the computation. The difference might be caused either by unoptimised code for gathering distribution data, or by different build environments on the two supercomputers.

We generated distribution data for 9 × 9 grids as well. The duration of this task was similar (see Row 4 of Table 2). However, as noted before, in this case we took into account only 1/500000000 part of the complete domain.

One may also notice that wall-clock time values relative to total CPU times are somewhat higher on Superman than on Hercules. The reason for this is that Superman is used by many

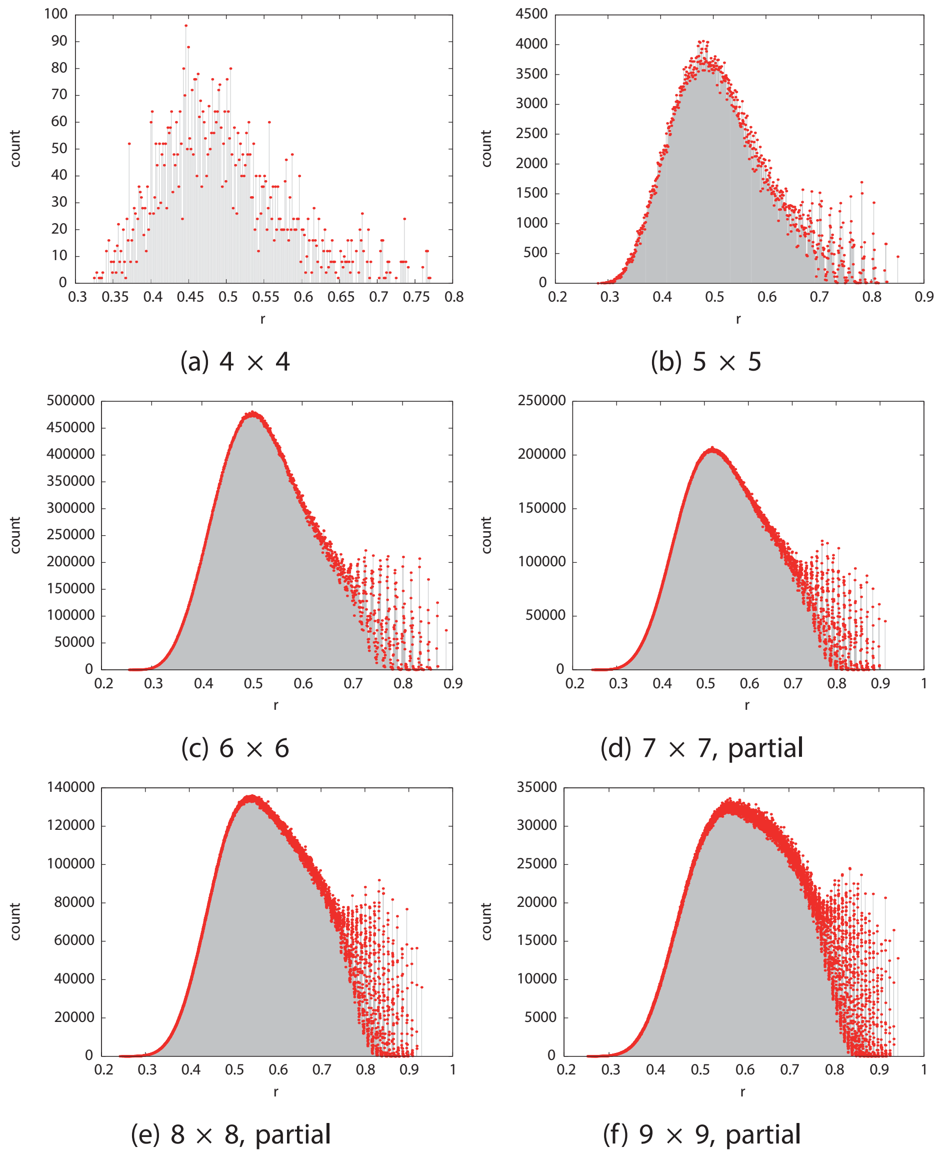

Fig. 7. Distributions of joint sensitivity indexes of grid trusses. Only plots a, b and c represent the complete domain.

other research projects, thus it often has higher load. Also, our partial distribution task was launched in two stages, with two days difference.

10 Validation of the performance model

After getting the desired results for our problems, we conducted simple experiments to measure parameters of the performance model presented in Section 7, and to validate this model. We took advantage of a period of time when a constant, larger number of processors of the Hercules supercomputer was available exclusively for us to use.

For the experiment, we selected the problem of top/bottom sensitivity lists for 6 × 6 grids - a problem of medium complexity (see Row 2 of Table 2) - to execute with various numbers of jobs. Initially we chose $J = 312$, the previously expected number of free processors as the number of jobs, and doubled it for each subsequent execution, up to $J = 4992$. For each execution, the number of processors running our jobs was $P = 299$.

As we had expected, on this scale of the number of jobs the domain splitting time T_{split} and the aggregation time T_{aggr} turned out to be very small, which even made them impossible to measure accurately. Therefore we excluded the time requirement of these stages from our experiments, and used the simpler model instead.

In Section 7 we suggested a sampling approach to measure t_c, the average computation time per job. As we already had had knowledge of the total CPU time $T_1 = Nt_c = 3423522$s (see Row2, Column 6 of Table 2), we instead simply divided this value by $N = 600805296$ (see Column 4 of the row corresponding to $n = 6$, $m = 6$ in Table 1), to get $t_c = 0.005698$s.

We measured t_s, the scheduler overhead time per job separately for each execution. Average and maximum values were gathered after the executions, although the information available from logs only allowed us to make accurate measurements for the averages. Interestingly, contrary to our expectations, the overhead seemed non-constant: it increased with J, as seen in Figure 8. A possible explanation for this is that the

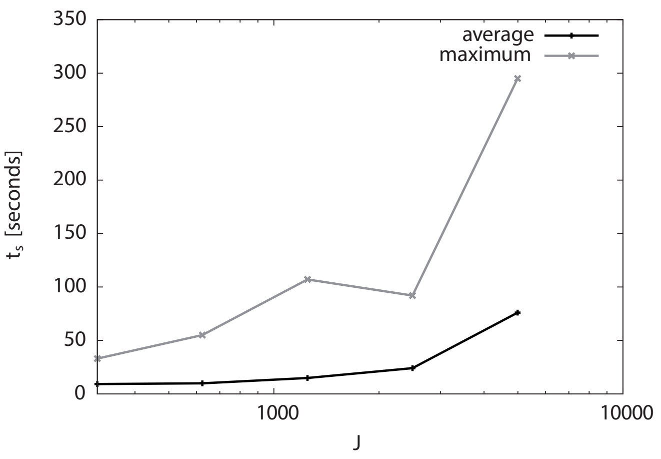

Fig. 8. Measured scheduler overhead times for executions of the same input with different numbers of jobs.

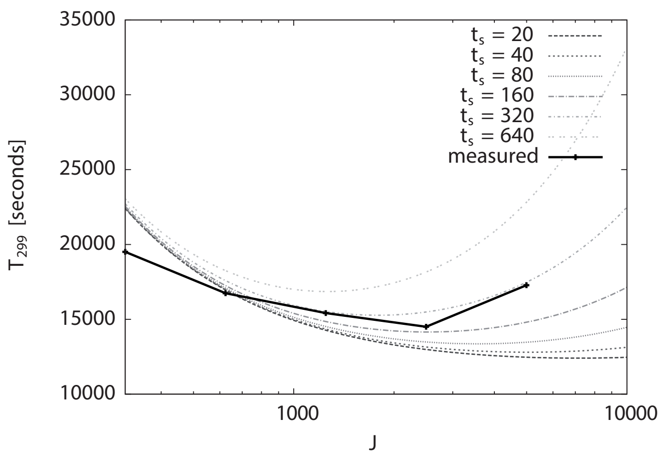

Fig. 9. Execution time on 299 cores, as a function of the number of jobs.The graph shows calculations using Formula 7 for various scheduler overhead times, and the measured data.

matchmaking mechanism of the scheduler slows down in case of a large amount of jobs waiting in the queue.

To check the validity of the performance model, we compared values calculated using Formula 7 as a function of J with actual measured wall-clock times. The measured times are represented by the solid black line in Figure 9. Based on this, it can be concluded that the wall-clock time has in fact a minimum around $J = 2496$. To calculate theoretical wall-clock times using the model, we had all necessary parameters at our disposal except t_s, which is non-constant according to our experience. We chose to compare the measurements with calculations using several different t_s parameter values, shown as grey dashed curves in Figure 9. The minima of these curves can be obtained using Formula 11.

It should not be forgotten that we built a very simple model, with heavy simplifications. For example, the worst case approach for calculating the idle time is an overestimation, which is likely the cause of the calculated values being significantly greater than measured times at smaller J values. Additionally, the wall-clock time in practice increases more steeply than the calculated curves, due to t_s increasing with the number of jobs. We can also observe that the measurements are closer to those calculations that correspond to somewhat higher t_s values than what we had experienced in practice. Using a maximum instead of an average for t_s may result in a better fit.

11 Summary

Using algorithms based on combinatorics, graph theory and matroid theory, and taking advantage of the power of a HPC infrastructure, we analysed rigidity and sensitivity attributes of plane trusses having a square grid topology with diagonal bars. The problem we faced had no analytic solution, and the input domain grew exponentially large as the grid increased in size. In spite of that, a supercomputer proved powerful enough for us to examine a large number of simple yet non-trivial cases, allowing us to get an overview of their properties. We designed a parallel algorithm and proved its scalability in order to take advantage of the power of a supercomputer. A model has been set up for the performance of this parallel algorithm, and tested with experiments.

One of our achievements is the identification of diagonal bar location patterns of the minimal and maximal sensitivity grid trusses. Although our method was a numerical analysis of small grids, we were able to provide plausible arguments to those for arbitrary grid sizes.

We have also shed light on an unexpected phenomenon of the distributions of joint sensitivity indexes: they appear to converge, at least partially, to a continuous distribution. This phenomenon is clearly visible even though we did not perform any summing or averaging over sections of the index.

Acknowledgement

The work reported in the paper has been developed in the framework of the project “Talent care and cultivation in the scientific workshops of BME” project. This project is supported by the grant TÁMOP-4.2.2.B-10/1--2010-0009.

References

1 **Amdahl Gene M.**, *Validity of the single processor approach to achieving large scale computing capabilities*. AFIPS '67 (Spring), Proceedings of the April 18-20, 1967, spring joint computer conference. New York. pp. 483-485, (1967). DOI: 10.1145/1465482.1465560

2 **Appleton O. et al.**, *The next-generation ARC middleware.* Annales des Telecommunications/Annals of Telecommunications, 65 (11-12), pp. 771-776, (2010).

3 **Berg A. R., Jordán T.**, *Algorithms for Graph Rigidity and Scene Analysis.* Lecture Notes in Computer Science, pp. 78-79, (2003). DOI: 10.1007/978-3-540-39658-1_10

4 **Bolker E. D., Crapo H.**, *How to brace a one-story building.* Environment and Planning B: Planning and Design, 4 (2), pp. 125-152, (1977). DOI: 10.1068/b040125

5 **Dóbé P., Domokos G.**, *Combinatorial Measurement of the Geometric Sensitivity of Plane Trusses.* The 7th Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications. Kyoto. pp. 29-36, (2011).

6 **Dóbé P., Kápolnai R., Szeberényi I.**, *Saleve: toolkit for developing parallel Grid applications*. Híradástechnika, LXIII (1), pp. 60-64 (2008).

7 **Eppstein D., Galil Z.**, *Parallel Algorithmic Techniques for Combinatorial Computation*. Annual Review Computer Science, 3, pp. 233-283, (1988).

8 **Foster I.**, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Boston: Addison-Wesley, (1995).

9 **Foster I., Kesselman C.**, *The Grid 2: Blueprint for a New Computing Infrastructure*. Elsevier Science, (2003).

10 **Graver J. E., Servatius H., Servatius B.**, *Combinatorial Rigidity*. Graduate Studies in Mathematics. American Mathematical Society, (1993).

11 **Henneberg L.**, *Die graphische Statik der starren Systeme*. B. G. Teubner, (1911).

12 **Hurlbert G. H.**, *On encodings of spanning trees.* Discrete Applied Mathematics, 155 (18), pp. 2594-2600, (2007). DOI: 10.1016/j.dam.2007.07.014

13 **Jordán T., Domokos G., Tóth K.**, *Geometric Sensitivity of Rigid Graphs*. In 7th Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications. Kyoto, June 2011. pp. 133-142, (2011).

14 **Knuth D. E.**, *The Art of Computer Programming: Generating All Combinations and Partitions*. Art of Computer Programming Series, Volume 4, (2005). ISBN 0201853949

15 **Laman G.**, *On graphs and rigidity of plane skeletal structures.* Journal of Engineering Mathematics, 4 (4), pp. 331-340, (1970). DOI: 10.1007/bf01534980

16 **Lovász L., Yemini Y.**, *On Generic Rigidity in the Plane*. SIAM Journal on Algebraic Discrete Methods, 3 (1), pp. 91-98, (1982). DOI: 10.1137/0603009

17 **Prüfer H.**, *Neuer Beweis eines Satzes über Permutationen*. Archiv für Mathematik und Physik, 27, pp. 742-744, (1918).

18 **Szabó J., Roller B.**, *Rúdszerkezetek elmélete és számítása*. Budapest: Műszaki Könyvkiadó, (1971).

19 **Thain D., Tannenbaum T., Livny M.**, *Distributed computing in practice: the Condor experience*. Concurrency and Computation: Practice and Experience, 17 (2-4), pp. 323-356, (2005). DOI: 10.1002/cpe.938

20 **Tóth K., Domokos G., Gáspár Zs.**, *Statikailag határozott rácsos tartók geometriai érzékenysége*. Építés- Építészettudomány, 37 (3-4), pp. 225-240, (2009). DOI: 10.1556/eptud.37.2009.3-4.2

21 **Whiteley W.**, *Rigidity and scene analysis*. In Goodmanand, J. E., O'Rourke, J. (eds). Handbook of discrete and computational geometry. USA: CRC Press, Inc., BocaRaton, FL, pp. 893-916, (1997).