A Study on Vectorisation and Paralellisation of the Monotonicity Approach

Iwona Skalna(✉) and Jerzy Duda

AGH University of Science and Technology, Krakow, Poland
skalna@agh.edu.pl, jduda@zarz.agh.edu.pl

Abstract. Solving parametric interval linear systems is one of the fundamental problems of interval computations. When the solution of a parametric linear system is a monotone function of interval parameters, then an interval hull of the parametric solution set can be computed by solving at most $2n$ real systems. If only some of the elements of the solution are monotone functions of parameters, then a good quality interval enclosure of the solution set can be obtained. The monotonicity approach, however, suffers from poor performance when dealing with large scale problems. Therefore, in this paper an attempt is made to improve the efficiency of the monotonicity approach. Techniques such as vectorisation and parallelisation are used for this purpose. The proposed approach is verified using some illustrative examples from structural mechanics.

Keywords: Parametric interval linear systems · Monotonicity approach · Vectorisation · Parallelisation

1 Introduction

Solving systems of linear equations is essential in modern engineering. Highly complex physical systems, which would require extremely complex formulae to describe, are approximated with high accuracy by a very large set of linear equations. In order to get reliable results, uncertainty, which is inevitable in real life problems, should be taken into account in any computations. Therefore, solving linear systems is one of the fundamental problems of interval computations, wherein "to solve a system" usually means to enclose a *parametric solution set* by an interval vector as tightly as possible.

The assumption that the coefficients of a linear system vary independently within given ranges is rarely satisfied in practice. That is why recently a big effort was made to develop methods that are able to solve the so-called *parametric interval linear systems*, i.e., linear systems with elements being functions of parameters that are allowed to vary within given intervals. Until now, several such methods have been developed, one can mention approximate methods described in [4,5,11,17,18].

The tightest is the resulting interval vector, the better. The narrowest possible interval enclosure is called the *interval hull solution* (or simply the hull).

R. Wyrzykowski et al. (Eds.): PPAM 2015, Part II, LNCS 9574, pp. 455–463, 2016.
DOI: 10.1007/978-3-319-32152-3_42

When the parametric solution is monotone with respect to all the parameters, the hull can be computed by solving at most $2n$ real systems. The monotonicity approach was investigated, e.g., in [6,12,15,20]. Generally, checking monotonicity is a very complex task and for large scale problems the monotonicity based methods are inefficient. Therefore, in this paper an attempt is made to reduce the computational time of the monotonicity approach. Vectorisation and parallelisation techniques are used for this purpose. It is worth to add that the monotonicity approach is extensively used in the interval global optimisation. Thus, the improvement of the efficiency of the monotonicity approach will significantly influence the efficiency of the interval global optimisation and monotonicity based methods.

The paper is organised as follows. The Sect. 2 contains preliminaries on solving parametric interval linear systems with two disjoint sets of parameters. In the Sect. 3, the monotonicity approach for computing interval hull solution is outlined. Section 4 presents general concepts of vectorisation and parallelisation. This is followed by a description of the monotonicity approach. Next, some illustrative examples of truss structures and the results of computational experiments are presented. The paper ends with concluding remarks.

2 Preliminaries

Italic font will be used for real quantities, while bold italic font will denote their interval counterparts. Let $\mathbb{IR}$ denote a set of real compact intervals $\boldsymbol{x} = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} \mid \underline{x} \leqslant x \leqslant \overline{x}\}$. For two intervals $a, b \in \mathbb{IR}$, $a \geqslant b$, $a \leqslant b$ and $a = b$ will mean that, resp., $\underline{a} \geqslant \overline{b}$, $\overline{a} \geqslant \underline{b}$, and $\underline{a} = \underline{b} \wedge \overline{a} = \overline{b}$. $\mathbb{IR}^n$ will denote interval vectors and $\mathbb{IR}^{n\times n}$ will denote square interval matrices [10]. The midpoint $\check{x} = (\underline{x}+\overline{x})/2$ and the radius $\mathrm{r}(\boldsymbol{x}) = (\overline{x} - \underline{x})/2$ are applied to interval vectors and matrices componentwise.

Definition 1. *A parametric linear system*

$$A(p)x = b(p) \tag{1}$$

is a linear system with elements that are real valued functions of a K-dimensional vector of parameters $p = (p_1, \ldots, p_K) \in \Re^K$, i.e., for each $i, j = 1, \ldots, n$,

$$\begin{aligned} A_{ij} &: \Re^K \ni (p_1, \ldots, p_K) \to A_{ij}(p_1, \ldots, p_K) \in \Re, \\ b_i &: \Re^K \ni (p_1, \ldots, p_K) \to b_i(p_1, \ldots, p_K) \in \Re. \end{aligned} \tag{2}$$

Functions describing the elements of a parametric linear system can be generally divided into affine-linear and nonlinear. However, nonlinear dependencies can be easily reduced to affine-linear using affine arithmetic [1]. Therefore, the following consideration are limited to the affine-linear case.

Remark: Obviously, the transformation from nonlinear to affine-linear dependencies causes some loss of information [1], nevertheless, the approach based on affine arithmetic is worth considering as it is simple and quite efficient.

Definition 2. *A parametric interval linear system with affine dependencies is given by*

$$A(p)x = b(p), \tag{3}$$

where $A(p) = A^{(0)} + \sum_{k=1}^{K} A^{(k)} p_k$, $b(p) = b^{(0)} + \sum_{k=1}^{K} b^{(k)} p_k$, $A^{(i)} \in \Re^{n\times n}$, *and* $b^{(i)} \in \Re^n$ $(i = 1, \ldots, K)$.

If the involved parameters are subject to uncertainty, which means that they allowed to vary within given intervals (the interval-based model of uncertainty is adopted in this paper), then a *parametric interval linear system* is obtained.

Definition 3. *A parametric interval linear system is an infinite set (family) of parametric real linear systems*

$$\{A(p)x = b(p) \mid p \in \boldsymbol{p}\}. \tag{4}$$

The family (4) is usually written in a compact form as

$$A(\boldsymbol{p})x(\boldsymbol{p}) = b(\boldsymbol{p}). \tag{5}$$

Definition 4. *A parametric (united) solution set of the system (5) is a set of solutions to all systems from the family (4), i.e.,*

$$S(\boldsymbol{p}) = \{\, x \mid \exists p \in \boldsymbol{p},\ \ A(p)x = b(p)\}. \tag{6}$$

In order that the solution set be bounded, the parametric matrix $A(\boldsymbol{p})$ must be regular, i.e., $A(p)$ must be non-singular for each $p \in \boldsymbol{p}$.

In general case, the problem of computing the solution set (6) as well as its hull are NP-hard. Therefore, usually an outer interval enclosure, i.e., the vector $\boldsymbol{x}^{\text{out}} \supset S(\boldsymbol{p})$, is computed instead. However, when the solution of a parametric system is monotone with respect to all parameters, then the hull of the solution set can be computed with polynomial cost in n and K. If the solution is monotone with respect to some of the parameters, then a good quality outer solution can be computed with polynomial cost in n and K.

3 Monotonicity Approach

For the sake of completeness of the paper, a brief reminder of the monotonicity approach is presented below.

Let $E^K = \left\{e \in \mathbb{R}^K \mid e_k \in \{-1, 0, 1\},\ k = 1, \ldots, K\right\}$. For $\boldsymbol{p} \in \mathbb{IR}^K$, $e \in E^K$, $p_k^e = \underline{p}_k$ if $e_k = -1$, $p_k^e = \check{p}_k$ if $e_k = 0$, and $p_k^e = \overline{p}_k$ if $e_k = 1$.

Theorem 1. *Let* $A(\boldsymbol{p})$ *be regular and let the functions* $x_i(p) = \left\{A^{-1}(p) \cdot b(p)\right\}_i$ *be monotone on an interval box* $\boldsymbol{p} \in \mathbb{IR}^K$, *with respect to each parameter* p_k $(k = 1, \ldots, K)$. *Then, for each* $i = 1, \ldots, n$,

$$\{\square S(\boldsymbol{p})\}_i = \left[\left\{A\left(p^{-e^i}\right)^{-1} b\left(p^{-e^i}\right)\right\}_i, \left\{A\left(p^{e^i}\right)^{-1} b\left(p^{e^i}\right)\right\}_i\right], \tag{7}$$

where $e_k^i = \operatorname{sign}\frac{\partial x_i}{\partial p_k}$, $k = 1, \ldots, K$.

Now consider the family of parametric linear equations (4) and assume that $A_{ij}(p)$ and $b_i(p)$ $(i, j = 1, \ldots, n)$ are continuously differentiable in $\boldsymbol{p}$. If x is a solution to the system $A(p)x = b(p)$, then $x = A(p)^{-1}b(p)$, which means that x is a function of p. Thus, the global monotonicity properties of the solution with respect to each parameter p_k can be verified by checking the sign of derivatives $\frac{\partial x}{\partial p_k}(p)$ on the domain $\boldsymbol{p}$. The differentiation of the Eq. (1) with respect to p_k $(k = 1, \ldots, K)$ yields

$$\left\{ A(p)\frac{\partial x}{\partial p_k}(p) = \frac{\partial b}{\partial p_k}(p) - \frac{\partial A(p)}{\partial p_k}x(p) \;\middle|\; p \in \boldsymbol{p} \right\}. \tag{8}$$

Since $A_{ij}(p)$, $b_j(p)$ are affine linear functions of p, thus $\frac{\partial A_{ij}}{\partial p_k}$, $\frac{\partial b_i}{\partial p_k}$ are constant on $\boldsymbol{p}$. Hence, the approximation of $\frac{\partial x}{\partial p_k}(p)$ can be obtained by solving the following K parametric linear systems

$$A(\boldsymbol{p})\frac{\partial x}{\partial p_k} = b'(\boldsymbol{x}^*) \tag{9}$$

where $b'(\boldsymbol{x}^*) = \{b^{(k)} - A^{(k)}x^* \mid x^* \in \boldsymbol{x}^*\}$ and $\boldsymbol{x}^*$ is some initial solution to the system (5).

For a fixed i $(1 \leqslant i \leqslant n)$, let $\boldsymbol{D}_{ki}$ denotes the interval estimate of $\{\frac{\partial x_i}{\partial p_k}(p) \mid p \in \boldsymbol{p}\}$ obtained by solving the Eq. (9). Now assume that each $\boldsymbol{D}_{ki}$ $(k = 1, \ldots, K)$ has a constant sign or equals 0. Then, in order to calculate the hull of $\{S(\boldsymbol{p})\}_i$, the elements of the vector e^i must be determined as follows: $e^i_k = 1$ if $\boldsymbol{D}_{ki} \geqslant 0$, $e^i_k = 0$ if $\boldsymbol{D}_{ki} \equiv 0$, and $e^i_k = -1$ if $\boldsymbol{D}_{ki} \leqslant 0$. If the sign of some of the partial derivatives was not determined definitely, then a new vector of parameters is constructed by substituting the respective endpoints for interval parameters. The process of determining the sign of derivatives restarts and continues until no further improvement is obtained. The algorithm of the method is presented below. Parts of code in Algorithm 1 which are candidates for parallelisation and vectorisation are indicated by comments.

4 Parallelisation and Vectorisation

Parallelisation is the process of converting sequential code into a multi-threaded one in order to use available processors simultaneously. The parallelisation process often also includes *vectorisation*, because contemporary central processor units are able to perform operations on multiple data in a single instruction. This ability is called SIMD (single instruction multiple data). It allows to convert an algorithm from a scalar implementation, in which a single instruction can deal with one pair of operands at a time, to a vector process, where a single instruction can refer to a vector of operands (series of adjacent values). Vectorisation can be carried out either automatically by contemporary C++ compilers or forced by a programmer usually by using an appropriate pragma. Vectorisation not always brings performance improvement due to additional data movement and pipeline synchronisation. Thus, the vectorisation can be profitable for loops that

Algorithm 1. Monotonicity approach

$\boldsymbol{x}_0 \supseteq \square\{x \mid A(p)x = b(p) \text{ for some } p \in \boldsymbol{p}\}$
// potential candidate for parallelisation
for $k = 1$ to K **do**
  // potential candidate for vectorisation
  $\boldsymbol{D}_k \supseteq \square\{y \mid \exists p \in \boldsymbol{p}\ A(p)y = \partial b/\partial p_k - \partial A(p)/\partial p_k \boldsymbol{x}_0\}$
end for
// potential candidate for parallelisation
for $i = 1$ to n **do**
  for $k = 1$ to K **do**
    // Assign a value to e_k^i based on $\boldsymbol{D}_{ki}$
  end for
  // potential candidates for vectorisation
  $\boldsymbol{x}_i^{\min} \supseteq \square\{x \mid A(p^{-e})x = b(p^{-e})\}_i$
  $\boldsymbol{x}_i^{\max} \supseteq \square\{x \mid A(p^{e})x = b(p^{e})\}_i$
  $\boldsymbol{x}_i^{\text{out}} = [\underline{\boldsymbol{x}}_i^{\min}, \overline{\boldsymbol{x}}_i^{\max}]$
end for

run for a suitable number of iterations. Such loops however, must meet certain constraints including continuous access of memory, no data dependency and only single exit from the loop.

The newest Intel and AMD processors implements Advanced Vector Extensions (AVX) instruction set that operates on 256 bit SIMD registers. For double precision floating point numbers this allows to perform basic mathematical operations on 4 numbers at once. Example of addition with SIMD is shown in Fig. 1. In the experiments presented in this paper, the newest Intel C++ compiler (16.0) is used. This compiler efficiently analyse the code and indicates which loops are worth to be vectorised. It can also be forced to vectorise other loops by using `#pragma simd`. Both these mechanisms are used in the experiments to improve the efficiency of the monotonicity approach.

While vectorisation plays only a supporting role in parallelisation process, the main benefit can be achieved by transforming the code so that it is able to utilise many threads simultaneously. This is realised in either task parallelism model (the so called fork-join parallelism) or single program multiple data (SPMD) model. For a single multi-core processor the first model is usually implemented as parallel constructs nest in a straightforward manner [21].

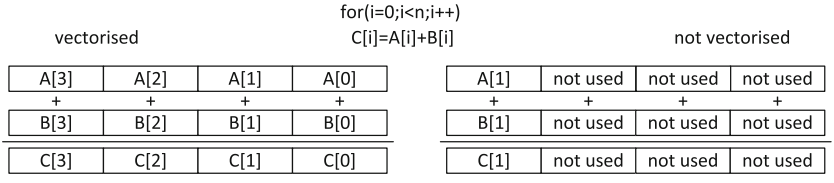

Fig. 1. Loops vectorisation

Similarly to the vectorisation, parallelisation can be done automatically by a C++ compiler or can be guided by a user. When using an Intel compiler three methods can be used: Threading Building Blocks (TBB) or Cilk Plus (originally developed in MIT) and auto parallelisation with OpenMP. The two first methods use work-stealing strategy, in which each processor maintains its own local queue and when the local queue is empty, the worker randomly steals work from victim worker queues, while in OpenMP a master thread forks a specified number of slave threads and divides a task among them. In our experiments all three types of parallelisation are used.

5 Numerical Examples

To check the performance of the monotonicity approach some illustrative examples of structural mechanical systems are considered. The obtained results are compared with the results given by a method (it should be added that there are several such methods [2,4,12,17], however a great majority of them yields very similar results, especially for the problem considered here) for computing outer interval solution of parametric interval linear systems.

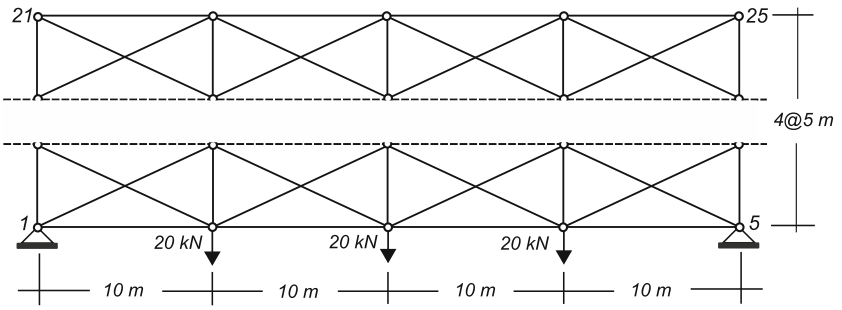

Fig. 2. Example 1: 5-bay 4-floor plane truss structure

Example 1 (*5-bay 4-floor plane truss structure*). For the plane truss structure shown in Fig. 2 the displacements of the nodes are computed. The truss is subjected to downward forces $P_2 = P_3 = P_4 = 20$[kN] as depicted in the figure; Young's modulus $Y = 2.0 \times 10^{11}$[Pa], cross-section area $C = 0.0001$[m^2], length of horizontal bars is $L = 10$[m], and the lenght of vertical bars is $H = 5$[m]. The truss is fully supported at the nodes 1 and 5. This gives 72 interval parameters. Table 1 shows the relative times for various combinations of vectorisation and parallelisation. The baseline has been set to the variant of the program that used no vectorisation and no parallelisation. Tests have been run on the machine with Intel Xeon 1220v2 CPU with 4 cores and no hyper-threading ability.

The times presented in the table show the benefits that can be achieved by the vectorisation of the monotonicity algorithm, which are 10 % on average.

Table 1. Comparison of the computation times for Example 1

	Bridge_1	Bridge_4	Bridge_5	Bridge_6
no vectorisation; no parallelisation	76.09	76.38	77.06	77.27
forced vectorisation; no parallelisation	98.04	97.4	98.17	98.04
auto vectorisation; no parallelisation	69.61	69.79	70.09	70.18
auto vectorisation; auto parallelisation	69.14	71.76	21.26	21.67
no vectorisation; Cilk parallelisation	22.68	22.71	22.80	23.17
auto vectorisation; Cilk parallelisation	20.72	20.59	19.23	19.36
auto vectorisation; TBB parallelisation	18.68	19.93	19.03	19.06

Automatic parallelisation does not improve the processing times and in one case it consumes even more time. This is due to the fact, that compiler cannot be sure that the processing data are fully independent. When Cilk and TBB methods have been used, the improvement is significant and when combined with vectorisation they can improve the processing times up to four times.

Example 2 (Baltimore bridge built in 1870). Consider the plane truss structure shown in Fig. 3 subjected to downward forces of $P_1 = 80[kN]$ at node 11, $P_2 = 120[kN]$ at node 12 and P_1 at node 15; Young's modulus $Y = 2.1 \times 10^{11}$ $[Pa]$, cross-section area $C = 0.004[m^2]$, and length $L = 1[m]$. Assume that the stiffness of 23 bars is uncertain by $\pm 5\,\%$. This gives 23 interval parameters.

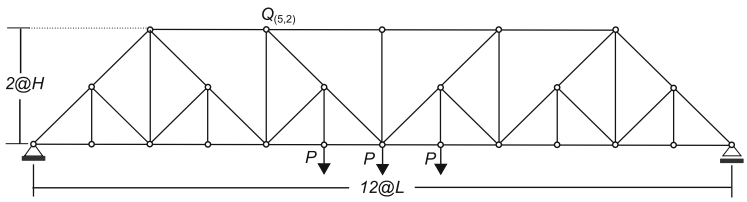

Fig. 3. Example 2: Baltimore bridge

The comparison of the performance of different variants of code vectorisation and paralellisation is presented in Table 2. Again the tests have been run on Intel Xeon 1220v3 processor with 4 cores. For the more complex problem the benefit in processing times is higher and equals up to 15 %. When we combine either Cilk or TBB parallelisation method with vectorisation the overall improvement reaches more than four times. The difference between both two fork-joint models is unconvincing, so each of them can be successfully applied. The overall computational experiments prove, that the more complex problem is the more is the benefit from using parallelisation and vectorisation.

Table 2. Comparison of the computation times for Example 2

	Bridge_1	Bridge_4	Bridge_5	Bridge_6
no vectorisation; no parallelisation	150.08	150.29	149.52	149.86
forced vectorisation; no parallelisation	203.61	204.03	203.25	203.39
auto vectorisation; no parallelisation	131.73	131.32	130.59	130.55
auto vectorisation; auto parallelisation	131.21	130.11	129.61	129.56
no vectorisation; Cilk parallelisation	41.67	41.57	41.31	40.85
auto vectorisation; Cilk parallelisation	36.73	35.89	35.75	36.06
auto vectorisation; TBB parallelisation	36.09	35.57	35.39	35.76

6 Conclusions

Checking the sign of the derivatives is a clue to test the global monotonicity of the solution of parametric linear systems. The global monotonicity enables calculating the interval hull solution easily by solving at most $2n$ real systems. The main deficiency of the monotonicity approach is its poor performance. As shown by the performed experiments, the performance of the monotonicity approach can be improved by using techniques such as vectorisation and parallelisation, which are available for contemporary C++ and Fortran compilers like Visual C++, Gnu cpp or Intel compiler in Parallel Studio XE.

The presented methodology can be applied to any problem which requires solving linear systems with input data dependent on uncertain parameters. The monotonicity approach is also a crucial acceleration techniques for interval global optimisation applied for the problem of solving parametric interval linear systems. The improved version of monotonicity approach can significantly decrease the computational time of the interval global optimisation.

References

1. de Figueiredo, L.H., Stolfi, J.: Self-validated numerical methods and applications. In: Brazilian Mathematics Colloquium Monographs, IMPA/CNPq, Rio de Janeiro, Brazil (1997)
2. Hladík, M.: Enclosures for the solution set of parametric interval linear systems. Int. J. Appl. Math. Comput. Sci. **22**(3), 561–574 (2012)
3. Jansson, C.: Interval linear systems with symmetric matrices, skew-symmetric matrices and dependencies in the right hand side. Computing **46**(3), 265–274 (1991)
4. Kolev, L.: A method for outer interval solution of linear parametric systems. Reliable Comput. **10**, 227–239 (2004)
5. Kolev, L.: Outer solution of linear systems whose elements are affine functions of interval parameters. Reliable Comput. **6**, 493–501 (2002)
6. Kolev, L.: Solving linear systems whose elements are non-linear functions of intervals. Numer. Algorithms **37**, 213–224 (2004)

7. Muhanna, R.L., Erdolen, A.: Geometric uncertainty in truss systems: an interval approach. In: Rafi, L., Muhanna, R.L.M., (eds.): Proceedings of the NSF Workshop on Reliable Engineering Computing: Modeling Errors and Uncertainty in Engineering Computations, Savannah, Georgia, USA, 22–24 February 2006, pp. 239–247 (2006)
8. Muhanna, R., Kreinovich, V., Solin, P., Cheesa, J., Araiza, R., Xiang, G.: Interval finite element method: new directions. In: Rafi, L., Muhannah, R.L.M., (eds.) Proceedings of the NSF Workshop on Reliable Engineering Computing (REC), Svannah, Georgia USA, 22–24 February 2006, pp. 229–244 (2006)
9. Neumaier, A.: Worst case bounds in the presence of correlated uncertainty. In: Rafi, L., Muhannah, R.L.M., (eds.): Proceedings of the NSFWorkshop on Reliable Engineering Computing (REC), Savannah, Georgia USA, 22–24 February 2006, pp. 113–114 (2006)
10. Neumaier, A.: Interval Methods for Systems of Equations. Encyclopedia of Mathematics and Its Applications. Cambridge University Press, Cambridge (1990)
11. Popova, E.D.: On the solution of parametrised linear systems. In: Kraemer, W., J.W.v.G., (eds.) Scientific Computing, Validated Numerics, Interval Methods, Kluwer Academic Publishers, pp. 127–138 (2001)
12. Popova, E., Lankov, R., Bonev, Z.: Bounding the response of mechanical structures with uncertainties in all the parameters. In: Rafi, L., Muhannah, R.L.M., (eds.) Proceedings of the NSF Workshop on Reliable Engineering Computing (REC), Svannah, Georgia USA, 22–24 February 2006, pp. 245–265 (2006)
13. Pownuk, A.: Calculations of displacement in elastic and elastic-plastic structures with interval parameters. In: 33rd Solid Mechanics Conference, Zakopane, Poland, pp. 160–161, September 2000
14. Rao, S., Berke, L.: Analysis of uncertain structural systems using interval analysis. AIAA J. **35**(4), 727–735 (1997)
15. Rohn, J.: A method for handling dependent data in interval linear systems. Technical report 911, Academy of Sciences of the Czech Republic, Czech Republic (2004)
16. Rump, S.: Verification methods for dense and sparse systems of equations. In: Herzberger, J. (ed.) Topics in Validated Computations. Studies in Computational Mathematics, vol. 5, pp. 63–135. Elsevier, Amsterdam (1994)
17. Skalna, I.: A method for outer interval solution of parametrized systems of linear interval equations. Reliable Comput. **12**(2), 107–120 (2006)
18. Skalna, I.: Methods for solving systems of linear equations of structure mechanics with interval parameters. Comput. Assist. Mech. Eng. Sci. **10**(3), 281–293 (2003)
19. Skalna, I.: Evolutionary optimization method for approximating the solution set hull of parametric linear systems. In: Boyanov, T., Dimova, S., Georgiev, K., Nikolov, G. (eds.) NMA 2006. LNCS, vol. 4310, pp. 361–368. Springer, Heidelberg (2007)
20. Skalna, I.: On checking the monotonicity of parametric interval solution of linear structural systems. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2007. LNCS, vol. 4967, pp. 1400–1409. Springer, Heidelberg (2008)
21. Cytron, R., Lipkis, J., Schonberg, E.: A compiler-assisted approach to SPMD execution. In: Proceedings of the 1990 ACM/IEEE Conference on Supercomputing (Supercomputing 1990), CA, USA, pp. 398–406. IEEE Computer Society Press, Los Alamitos (1990)