An Efficient Parallel Algorithm for Numerical Solution of Low Dimension Dynamics Problems

Stepan Orlov[1], Alexey Kuzin✉ and Nikolay Shabrov[2]

Computer Technologies in Engineering dept.
Peter the Great St. Petersburg Polytechnic University
St. Petersburg, Russian Federation
[1] majorsteve@mail.ru ✉kuzin_aleksei@mail.ru [2] shabrov@rwwws.ru

Abstract. Present work is focused on speeding up computer simulations of continuously variable transmission (CVT) dynamics. A simulation is constituted by an initial value problem for ordinary differential equations (ODEs) with highly nonlinear right hand side. Despite low dimension, simulations take considerable CPU time due to internal stiffness of the ODEs, which leads to a large number of integration steps when a conventional numerical method is used. One way to speed up simulations is to parallelize the evaluation of ODE right hand side using the OpenMP technology. The other way is to apply a numerical method more suitable for stiff systems. The paper presents current results obtained by employing a combination of both approaches. Difficulties on the way towards good scalability are pointed out.

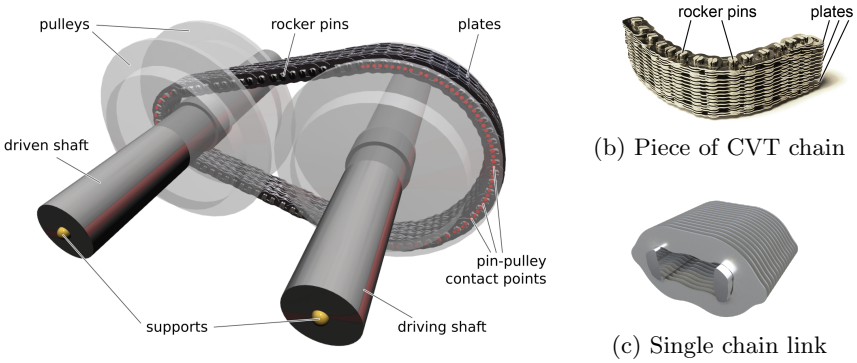
Keywords: Continuously Variable Transmission· OpenMP· Numerical Integration· Parallel Algorithm

1 CVT Model Overview

The paper considers simulations of the dynamics of continuously variable transmission (CVT) consisting of two shafts (driving and driven ones), each bearing two pulleys, and the chain consisting of *rocker pins* and *plates* (Figure 1). Pulleys have toroidal surfaces contacting with the convexo-convex end surfaces of pins. The torque is transmitted from the driving shaft to the driven one due to friction forces at pin-pulley contact points. One pulley at each shaft can move along the shaft axis, which allows to change CVT gear ratio. Rocker pins of the chain consist of two halves rolling over each other. A link of the chain is formed by 12–16 plates housing two pin halves.

Here we consider the mathematical model of CVT dynamics proposed in [1]. The model takes many details into account, which are the discrete chain structure; the extension and bending deformations of pins; the extension, bending, and torsion of plates; the elasticity of shafts, supports, and pulley-to-shaft attachments. The motion of the model is described by initial value problem for the system of ordinary differential equations (ODEs), which is obtained by means of Lagrangian mechanics:

$$A\ddot{q} = F(q, \dot{q}, t), \tag{1}$$



(a) CVT model view

(b) Piece of CVT chain

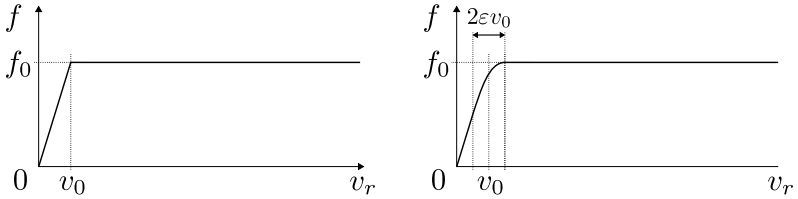
(c) Single chain link

Fig. 1. General view of CVT model; the chain; a single chain link

where q is the vector of generalized coordinates, A — positive-definite matrix of inertia, F — essentially non-linear function of q, $\dot{q}$ and t. The system can be transformed to normal form with obvious substitution: $u \equiv q$, $v \equiv \dot{q}$:

$$\dot{u} = v, \qquad A\dot{v} = F(u, v, t). \tag{2}$$

Importantly, many contact interactions in the CVT model involve friction forces. The friction law used is close to the Coulomb's one, but is regularized: the friction coefficient f is constant at relative speeds greater than the saturation speed v_0, and depends on speed linearly at speeds less than v_0. Therefore, the friction law is piecewise linear. The non-smoothness of the friction law may affect the behavior of numerical methods investigated, that's why we also consider an alternative system in which the friction law is smoothed using a parabola connecting straight parts (Figure 2).



original nonsmooth friction law

smoothed friction law, $\varepsilon = \frac{1}{2}$

Fig. 2. Friction laws used in numerical experiments

Each step of numerical integration of the system (1) includes the evaluation of the right hand side F based on the current state vector $q, \dot{q}$, and the evaluation of accelerations by solving the system of linear algebraic equations $A\ddot{q} = F$. These steps are repeated 4 times per integration step when the RK4 integration

method is used. Once the integration step is completed, the state vector at time instant $t + h$ is known. Then a *phase transition* procedure is run to detect pin — pulley contact state transition within the step. If transitions are found, the step is truncated, and the state is interpolated at the time of first transition.

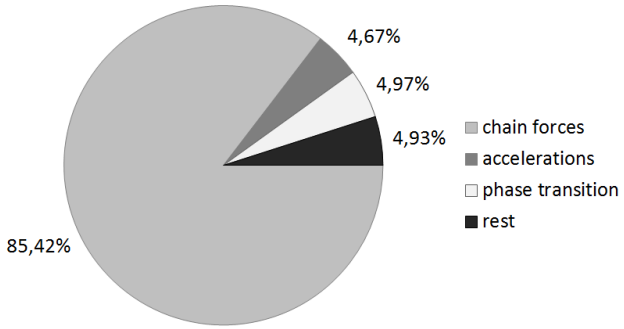


Fig. 3. CPU time consumption in CVT simulation. Sequential code

Parallelism can be introduced into the procedure in a natural way by evaluating the ODE right hand side in parallel. Presently the following parts of the integration step are parallelized: chain forces calculation, which takes almost all of the right hand side evaluation time, the reverse pass of Cholesky procedure of accelerations calculation, and the phase transition. Time costs of these separate operations in percents to overall simulation time obtained in sequential mode are shown in Figure 3. The largest part of simulation time is spent on chain forces calculation, which is both the forces between pins and plates and the contact forces between pin tips and pulleys. Altogether with Cholesky reverse pass and phase transition it takes almost 95% of overall time. Therefore, the program contains about 5% of sequential code.

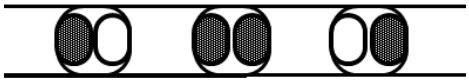


Fig. 4. Pin halves affected due to pin force application

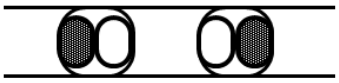


Fig. 5. Pin halves affected due to link force application

An application of parallel technologies for the model of bearings, which is an example of periodic mechanical structure is presented in the work [2]. Also there are a lot of works dedicated to the parallel-in-time approach [3–5], which

is to be perspective for the problems of small dimension. Therefore the approach can be considered as a field for future work for the problem considered while the present article shows the boundaries of speedup achieved using fine grained parallelism across the single integration step.

The chain forces that reside in the right hand side of (1) are generalized forces corresponding to the generalized coordinates of the pins. These forces emerge from interactions of two kinds. First of them are interactions between pin halves, and between pin halves and link plates; other forces, further referred to as *contact forces*, are result of contact between pin tips and pulleys. The forces of pins and links interactions also can be split into two groups: so called *pin forces* and *link forces*. Pin forces arise between halves of the same pin, these are, for example, contact interactions between pin halves. These interactions, when applied, give a contribution into generalized forces of both contacted halves and the pin halves of neighboring pins that belong to the same link with contacted ones (see Figure 4). Link forces arise from interactions between pins that belong to the same link therefore these are outer halves of adjoining pins (see Figure 5). These are, for example, the forces acting due to the link plates deformation. These interactions, when applied, give contribution to generalized forces of pin halves of the link only (Figure 5).

Due to this specificity of interactions structure, the model of chain forces calculation chosen in the application seems to be natural. Let us consider chain consisting of N pins and N links. It can be split to M continuous parts by pin numbers n_i, $i \in [0; M]$, $(n_0 = n_M)$. These pins and links are distributed between M threads so each i-th thread takes $[n_i, n_{i+1})$ links and $[n_i + 1, n_{i+1})$ pins. Therefore M pins stay unassigned. Each thread calculates generalized forces in assigned pins independently; there is no synchronization needed. The calculation ends with a barrier, and then each i-th thread calculates pin forces in the unassigned pin n_{i+1}. Figure 6 demonstrates this procedure for the case of $M = 2$ and $N = 8$. The threads 0 and 1 take pins $1-3$ and $5-7$ respectively. Pin forces for unassigned pins 0 and 4 are evaluated after the barrier by threads 1 and 0 respectively. It is worth noticing that a portion of generalized forces is applied to pins 0 and 4 before the barrier. The threads add these generalized forces in parallel to the same pins but into the different halves.

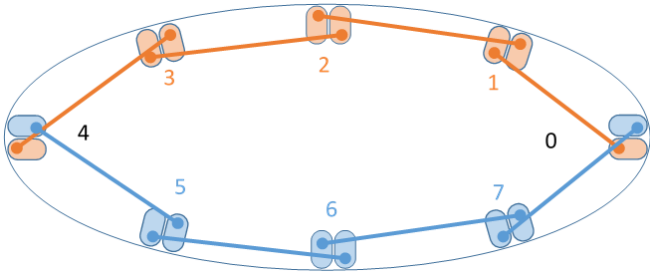


Fig. 6. An example of pins distribution between threads

Contact forces should be computed in a different way. These are link forces according to the schema of assignment, but they cannot be included into the procedure described because of calculation imbalance they produce. Therefore, pin halves in contact with pulleys are distributed uniformly between threads.

2 Results of Parallelization

Here the results of dynamics simulation of CVT with chain containing 84 pins are described. The simulations are performed on two nodes of "Polytechnic RSK Tornado" of Supercomputer Center Polytechnic of SPbPU, the parameters of the nodes are listed in Table 1.

Table 1. Parameters of computer used in simulations

Cores per socket	14
NUMA nodes	2
CPUs	Intel Xeon E5-2697 v3 2.60GHz
Linux	CentOS Linux release 7.0.1406 (Core)
C++ compiler	Intel 2017.5.239

The CPUs affinity has been set up, so when $M \leq 14$ only one NUMA node is in use and when $M > 14$ the additional cores have been requested on the second node (M is the number of threads).

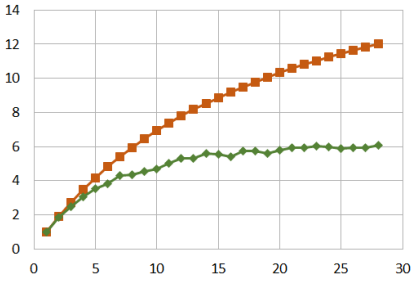


Fig. 7. Simulation speedup when acceleration calculation and phase transition are evaluated in parallel.

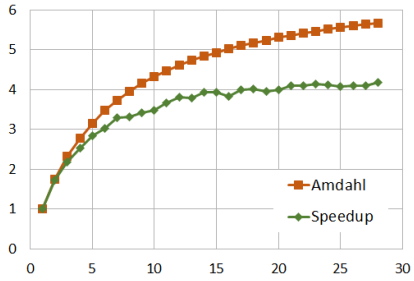


Fig. 8. Simulation speedup when acceleration calculation and phase transition are not evaluated in parallel.

The speedup of the whole simulation as a function of thread count is presented in Figure 7. Also the chart contains ideal curve that corresponds to Amdahl's law with the fraction of serial work equal to 5%, which is the case presented in Figure 3. The speedup is far from ideal mostly because of the bad scalability of the calculation of accelerations and phase transitions. One can see that

analogous simulation with these parts evaluated sequentially has better correspondence to the ideal curve (Figure 8), besides it loses in absolute speedup to the first simulation. The fraction of sequential code in this case, according to the Figure 3, is about 14%, and Amdahl's curve lays significantly lower.

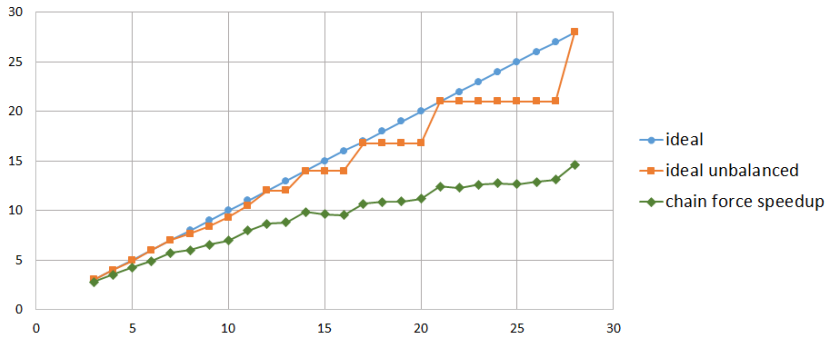


Fig. 9. Chain forces calculation speedup

The dependency of chain forces evaluation speedup on thread count is presented in Figure 9. The chart contains *experimental* curve, ideal speedup (*ideal*), and ideal speedup taking into account thread non-ideal balancing (*ideal unbalanced*). The last curve is defined as follows. According to work distribution schema described above, each i-th thread takes contiguous part of the chain consisting of N_i pins. In the ideal case all values N_i are equal to each other for $i \in [0, M)$, but it obviously takes place only when the total pin count $N = 84$ is divisible by M. Otherwise N_i can differ from each other by one. For example, when $M = 11$, 7 threads process 8 pins each and 4 threads — only 7 pins each. In this case the overall time is determined by the "slowest" thread that, obviously, processes 8 pins (it is supposed that all pins are processed in the same time and this takes place in the simulation). The curve *ideal unbalanced* is calculated as a function of M in the following way:

$$f(M) = \frac{N}{N_{i,max}(M)}, \tag{3}$$

where $N_{i,max}(M) = \max_{i \in [0,M)} (N_i)$ — the largest pin count per thread for the case of M threads. This is a piecewise line with horizontal segments. The segments consist of points where, despite the growth of thread count, $N_{i,max}$ remains the same. For example, when $M = 14$ all threads receive the same count of pins $N_i = 6$ and when $M = 15$, 9 threads receive 6 pins each and 6 threads — 5 pins each, so there is no speedup in this transition. One can notice that the bends of experimental curve repeat those ones of *ideal unbalanced*. Therefore, the bends of experimental curve take place because of the irregularity of $N_{i,max}$ decrements as the thread count grows. The jumps of speedup take place when

N is divisible by M: $M = 12, 14, 21, 28$. These are the cases of ideal balancing when all threads take the same count of pins. Analogous behavior takes place when $M = 17$. This is an "almost ideally" balanced case: 16 threads receive 5 pins each and one — 4 pins each.

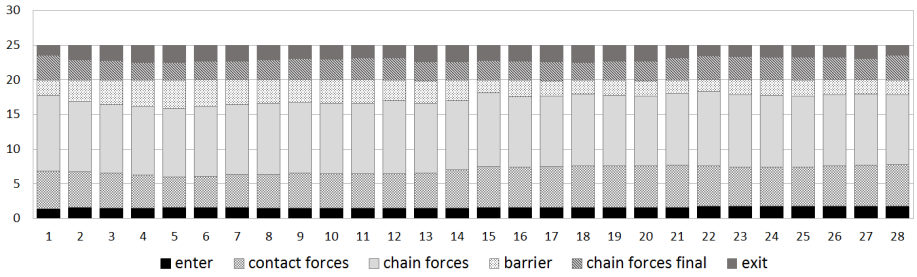


Fig. 10. Chain forces calculation time per thread

One can see that the experimental curve resides significantly lower than ideal ones, which can be explained by two factors. Firstly, there is an imbalance of loading between threads while contact forces are calculated; secondly, there is significant overhead time at parallel sections opening/closing. Both effects can be seen in Figure 10. This diagram shows overall execution time of parallel code for each thread that evaluates chain forces for $M = 28$. Each bar represents overall time of one thread and is split into the stages. The stages *enter* and *exit* represent opening and closing of OpenMP section, *contact forces* is the stage of contact forces calculation, *chain forces* is the stage of pin/link forces calculation before the barrier, *barrier* is the stage that represents OpenMP barrier neccesary before the rest M pin forces evaluation. And finally *chain forces final* is the stage of pin forces calculation of the rest M boundary pins.

One can notice that times of contact forces evaluation differ from one thread to another, while the times of pin/link forces evaluation are almost the same. This imbalance is noticeable, but one can see that more significant time is spent in sections *enter*, *exit* and *barrier*. This overhead cannot be explained only by load imbalance. Firstly, hotspot detection with `Intel VTune Amplifier` shows significant overhead time at OpenMP section opening/closing. Secondly, we have performed the same simulation with the application built with GCC compiler (GCC 5.4.0 and 7.2.0 have been used with the same results). This simulation has demonstrated much longer time (almost 1.5 times longer) of execution of sections *enter*, *exit*, and *barrier*, while durations of other sections have no significant changes. This fact also proves indirectly that the durations of sections *enter*, *exit* and *barrier* are determined in general by time spent in OpenMP code and not by unbalance of thread load.

Large overhead time is expectable with the schema of parallelism selected, when OpenMP sections are short and start and exit at each integration step.

Probably it is worth defining one common OpenMP section that would contain the whole ODE time integration procedure, but this is not straightforward and requires significant reorganization of integration scheme and can be a challenge for the future work.

3 Investigation of Numerical Methods

3.1 Previous Work

For a long time production versions of CVT simulation software have been using the classical explicit Runge–Kutta fourth order scheme (RK4) [6] to solve the initial value problem for equations (2). Explicit schemes impose limitations on step size h due to the stability requirement: the value $h\lambda$, where λ is an eigenvalue of ODE right hand side Jacobian matrix, must belong to the stability region of the method. That is the reason for long simulation times in the case of CVT dynamics equations. Previous work [7] presents the results of attempts to apply several other numerical integration methods, including explicit schemes (classical DOPRI45, DOPRI56, DOPRI78 methods), Gragg–Bulirsch–Stoer method (GBS), and Richardson's extrapolation applied to explicit Euler method; semi-implicit W-methods of Rosenbrock [8] type (SW24 [9] and Richardson's extrapolation applied to a W-method of first order); the trapezoidal rule implicit method (TRPZ). Among those methods, only the trapezoidal rule method allows to obtain sufficiently accurate numerical solution at steps much larger than the step chosen for RK4 (in the numerical example, step size for RK4 needs to be at most $5 \cdot 10^{-8}$ second, while the trapezoidal rule has allowed us to obtain numerical solution of the same accuracy at step $2 \cdot 10^{-6}$ second). Although the stability of implicit methods do not impose limitation on step size, such a limitation appears nevertheless: the nonlinear equation system that needs to be solved at each step has to be solved using a Newton-like iterative method; in our case, iterations only converge when the step size is not too large.

The trapezoidal rule has proved to be able to provide accurate numerical solution at time steps 40 times greater than steps we have to use with the RK4 scheme. However, the practical usefulness of the method is doubtful. In our numerical tests we found that the Newton's iterations converge at 5–10 steps only when the Jacobian matrix is fully recomputed at each iteration. Attempts to apply Broyden updates would break the sparsity of the Jacobian; attempts to limit Broyden's updates to current Jacobian sparsity stencil seem to work, but still recomputing the Jacobian may be needed more than once per time step, and the number of iterations increases up to tens. Jacobian updates proposed in [10] didn't work at all in our tests with CVT equations, although they worked well for other much simpler equation examples. Finally, not updating the Jacobian at all during the time step may increase the number of Newton's iterations up to hundreds, and the iterations may even fail to converge at all. Therefore, one practical approach to speedup simulations using the Newton's method is to compute the Jacobian matrix in parallel, probably employing the approach presented in [11] in order to reduce the number of ODE right hand side evaluations,

and to use a parallel LU solver for linear systems at Newton's iterations. This approach hasn't been tried yet.

3.2 Eigenvalues of ODE Right Hand Side Jacobian Matrix

We continued our search for a numerical method that is more suitable for the numerical integration of CVT dynamics equations. To make the search more purposeful, we have investigated eigenvalues of ODE right hand side Jacobian matrix. Namely, the full eigenvalue problem was numerically solved at some typical state in a stationary regime of CVT operation. The eigenvalues found split into two groups: firstly, there are complex eigenvalues corresponding to damped oscillations; secondly, there are real negative eigenvalues corresponding to non-oscillatory dissipation processes. Absolute values of eigenvalue imaginary parts have maximum at approx. 10^6 s^{-1} , while absolute values of real parts — at approx. 10^8 s^{-1}. Figure 11 shows the computed eigenvalues of the Jacobian matrix in the linear scale (left) and the logarithmic scale (right). Notice that in the latter case, the scale is logarithmic everywhere except the vicinity of zero — the region limited by dashed lines; in that region, the scale is linear.

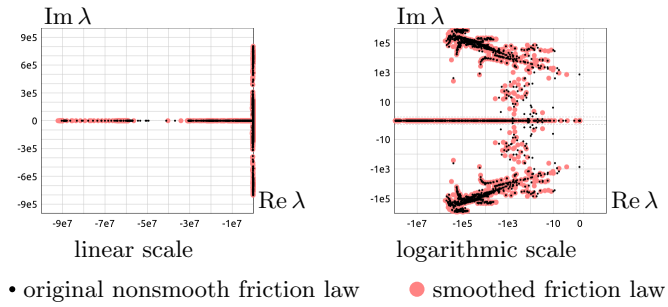


Fig. 11. Eigenvalues of ODE right hand side Jacobian matrix

Further we proved that the largest negative eigenvalues are caused by friction forces acting between pin halves, and the operating points in the friction law are at the linear part ($v < v_0$). The idea of the proof is to change (e.g., increase 10 times) the saturation speed v_0 in the friction law, then recompute the Jacobian matrix and find its eigenvalues. The results of evaluation are shown in Figure 12. In case (a), we increased the saturation speed v_0 10 times for the friction law between pin halves; in case (b), we also increased it 10 times for friction between pins and plates; in case (c), in addition, we increased it 10 times for friction between pins and pulleys.

Therefore, largest negative eigenvalues correspond to the friction between pin halves. This fact allows us to conclude that such eigenvalues are proportional to the tension force in the most loaded straight part of the chain, and, consequently, vary from one CVT operation regime to another.

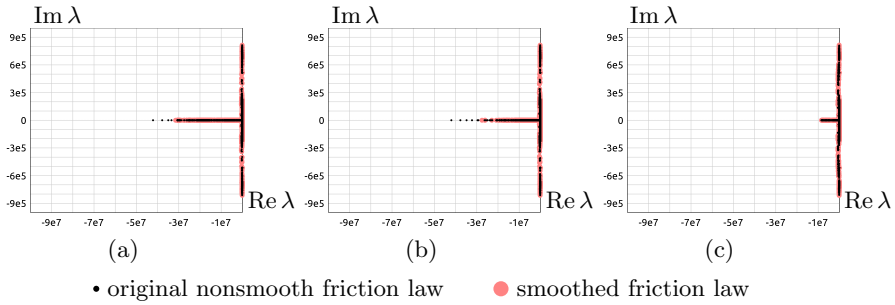


Fig. 12. Eigenvalues of ODE right hand side Jacobian matrix for modified friction law

The behavior and suitability of a numerical integration method strongly depend on the eigenvalues of the Jacobian matrix of the ODE right hand side. The product of a characteristic time span length H and the eigenvalue λ_* with maximum absolute value are of particular interest. The value H is the time between two neighboring states that we wish to have in the numerical solution, that is, the "output step". The product $H|\lambda_*|$ can be used to estimate how much steps a numerical method will take to reach time point $t+H$ starting from time point t. We can state that in our case the output step H is in the range 10^{-5}–10^{-4} second, depending on further processing of numerical solution.

The results of the Jacobian matrix investigation indicate that the ODE system we are dealing with is mildly stiff, because the product $H|\lambda_*|$ is in the range 10^3–10^4.

3.3 Applying Stabilized Explicit Method

Having investigated the stiffness properties of the ODE system, we decided to consider so called stabilized explicit, or Chebyshev–Runge–Kutta [12, ch. IV.2], numerical integration methods. Among those we picked the one called DUMKA3 [13]. The choice of that particular method was based on public availability of solver implementation code in C programming language.

A stabilized explicit scheme has stability region, defined by the condition $|R_s(h\lambda)| \leq 1$, extended into real negative direction of complex plane $h\lambda$. The function R_s is a polynomial of degree s, called the *stability polynomial*.

The DUMKA3 solver actually implements a family of s-stage Runge–Kutta schemes, each of which realizes stability polynomial of degree s varying from 3 up to 324. The solver implements automatic step size control, based on step local error estimation, and polynomial degree control, based on the estimation of ODE right hand side Jacobian matrix spectral radius. Preliminary tests have shown that solver performance with both control options enabled is far from optimal in our system. Besides, a numerical estimation of Jacobian spectral radius do not work well due to discontinuities of the ODE right hand side in the case of nonsmooth friction law, resulting sometimes in too large values. We disabled both control options. The motivation was to obtain best performance at fixed step for each fixed polynomial degree. Giving each polynomial an index k from 0

to 13 (DUMKA3 implements 14 polynomials), we denote corresponding schemes by suffixes -Pk. In particular, we tested degrees $s = 21$ ($k = 4$), $s = 27$ ($k = 5$), $s = 36$ ($k = 6$), $s = 48$ ($k = 7$), $s = 63$ ($k = 8$), $s = 81$ ($k = 9$). Polynomials $k \leq 4$ and $k \geq 9$ have shown poor performance: in the first case the stability region is too small in the real negative direction, and in the second case it is too small in the imaginary direction.

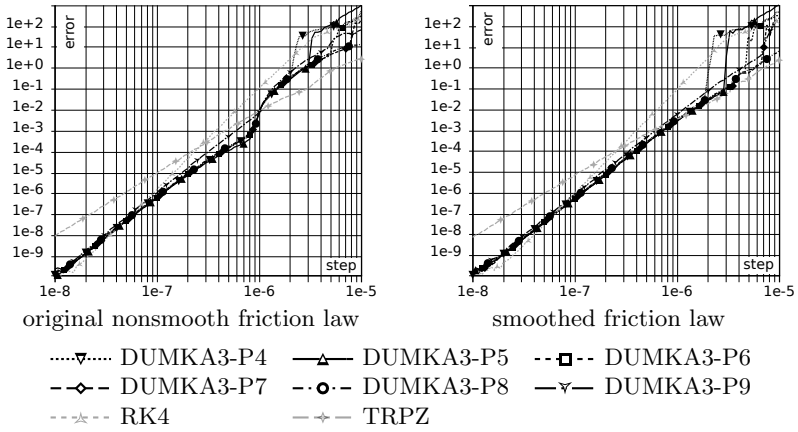


Fig. 13. Dependency of step local error norm on step size

Figure 13 shows the dependency of step local error norm on the step size. DUMKA3 results are compared against RK4 and the trapezoidal rule; the latter one is known [7] to give sufficiently accurate solution at $h = 2 \cdot 10^{-6}$ s. It can be shown that the slope for each curve at steps below 10^{-6} corresponds to the order of the scheme (3 for DUMKA3, 4 for RK4, and 2 for TRPZ). Notice also that at steps above 10^{-6} local error for all DUMKA schemes rises sharply at some point; additional error jumps can be see at step 10^{-6} for nonsmooth friction law. At large step sizes, local error for DUMKA schemes is approximately the same as for the trapezoidal rule, in the case of smoothed friction law; for nonsmooth fricion law, the trapezoidal rule gives smaller error.

Figure 14 shows the sample curve (pin axial force when it enters the driving pulley set) computed numerically. It follows from the figure that schemes DUMKA3-P5 – DUMKA3-P8 give sufficiently accurate solution at step $h = 2 \cdot 10^{-6}$, and at step $h = 4 \cdot 10^{-6}$ only the scheme DUMKA3-P8 does so.

Comparing simulation times, we can conclude that the DUMKA3 solver can perform simulations several times faster than RK4, which is summarized in table 2. Notice that the value n_{RHS} in the table is the total number of ODE right hand side evaluation in the test simulation (the same as used to obtain the sample curve).

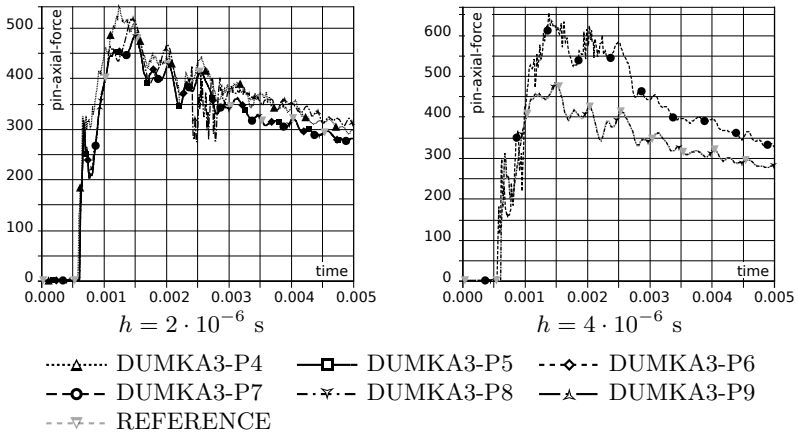


Fig. 14. Sample curve obtained with different polynomial degrees

Table 2. Performance of DUMKA3 schemes compared against RK4

scheme	h, s	n_{RHS}	speedup against RK4
RK4	$5 \cdot 10^{-8}$	400824	1
DUMKA-P5	$2 \cdot 10^{-6}$	75817	5.9
DUMKA-P7	$3 \cdot 10^{-6}$	95425	4.7
DUMKA-P8	$4 \cdot 10^{-6}$	104821	4.4

4 Conclusions and Future Work

The paper describes our activities in speeding up of simulation of nonlinear problem of dynamics modeling of CVT.

The procedure of ODEs integration over time can be parallelized in natural way when the right hand side is being calculated in parallel at each step. It can be done in this problem because of regular structure of the chain. Therefore the chain forces calculation can be distributed between threads by dividing the chain into continuous segments. This calculation procedure requires only one explicit barrier.

Another stages that can be calculated in parallel are accelerations evaluation and phase transition.

This approach leads to the problem of rather high overhead because of small parallel sections lengths. It can be overcome in the approach with one common parallel section that spans the whole integration procedure. We consider it as a perspective for future work.

The attempt to use stabilized explicit Runge–Kutta solver, DUMKA3, has proven to be successful in terms of performance. However, original polynomial order control implemented in the solver doesn't work: it tends to increase polynomial order, but in that case Jacobian eigenvalues with maximum imaginary parts fall outside the stability region, since it becomes a bit narrower in the imag-

inary direction. This motivates us to consider other stabilized explicit solvers, first of all RKC and SERK2 [14], and probably others, constructed according to the approach presented in [15], because there is a way to construct a method with stability region that best fits the spectrum of ODE right hand side Jacobian. Among all numerical integration methods tested so far for our problem, DUMKA3 has shown the best performance, for the case of sequential code. In the same time, estimations show that the trapezoidal rule method considered in [7] has the potential to gain the speedup of about 100 due to the parallelization of Jacobian calculation, which, however, does not reduce computational costs because the number of CPU cores required for that is about 600. The combination of the ODE right hand side parallelization and the DUMKA3 solver promises to achieve speedups of about 30.

References

1. Shabrov, N., Ispolov, Yu., Orlov, S.: Simulations of continuously variable transmission dynamics. ZAMM 94 (11), pp. 917–922. WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim (2014).
2. Nordling, P., Fritzson, P.: Solving ordinary differential equations on parallel computers – applied to dynamic rolling bearings simulation. In: Dongarra J., Waśniewski J. (eds) Parallel Scientific Computing. PARA 1994. Lecture Notes in Computer Science, vol 879. Springer, Berlin, Heidelberg, (1994).
3. Ruprecht, D., Krause, R.: Explicit parallel-in-time integration of a linear acoustic-advection system. Computers & Fluids, 59, pp. 72–83 (2012).
4. Jun Liu, Yao-Lin Jiang: A parareal waveform relaxation algorithm for semi-linear parabolic partial differential equations. Journal of Computational and Applied Mathematics, 236 (17), pp 4245–4263 (2012).
5. Kreienbuehl, A., Benedusi, B., Ruprecht, D., Krause, R.: Time parallel gravitational collapse simulation. Communications in Applied Mathematics and Computational Science. 12 (1), pp. 109–128, (2015).
6. Hairer, E., Nørsett, S.P., Wanner, G.: Solving Ordinary Differential Equations I (2Nd Revised. Ed.): Nonstiff Problems. Springer-Verlag New York, Inc. (1993).
7. Orlov, S., Kuzin, A., Shabrov, N.: Two approaches to speeding up dynamics simulation for a low dimension mechanical system. Communications in Computer and Information Science, 793, pp. 95–107 (2017).
8. Rosenbrock, H.H.: Some general implicit processes for the numerical solution of differential equations. Comput J 5, pp. 329–330 (1963).
9. Steihaug, T., Wolfbrandt, A.: An attempt to avoid exact Jacobian and nonlinear equations in the numerical solution of stiff differential equations. Math. Comp., 33 pp. 521–534 (1979).
10. Hart, W.E., Soesianto, F.: On the solution of highly structured nonlinear equations. Journal of Computational and Applied Mathematics 40 (3), pp. 285–296 (1992).
11. Ypma, T.J.: Efficient estimation of sparse Jacobian matrices by differences. Journal of Computational and Applied Mathematics 18 (1), 17–28 (1987).
12. Hairer, E., Wanner, G.: Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems. Springer-Verlag Berlin Heidelberg (1996).
13. Medovikov, A.A.: High order explicit methods for parabolic equations. BIT Numerical Mathematics 38 (2), pp. 372–390, (1998).

14. Martín-Vaquero, J., Janssen, B.: Second-order stabilized explicit Runge–Kutta methods for stiff problems. Computer Physics Communications 180 (10), 1802–1810 (2009).
15. Torrilhon, M., Jeltsch, R.: Essentially optimal explicit Runge–Kutta methods with application to hyperbolic–parabolic equations. Numerische Mathematik 106 (2), pp. 303–334 (2007).