DOI: 10.15439/2021F141
ISSN 2300-5963 ACSIS, Vol. 25

Practical parallelization of Gear-Nordsieck and Brayton-Gustavson-Hatchel stiff ODE solver

Marek Stabrowski
Warsaw University of Technology, Poland, e-mail: marek.2491@gmail.com

***Abstract*—The paper compares two ODE solvers using an example of a heat transfer equation. The sequential version of Brayton-Gustavson-Hatchel solver has been slightly inferior to Gear-Nordsieck solver. Algorithms profiling has led to the decision of parallelizing linear equation solving section and function evaluation. The first approach (parallelizing linear equations) improves performance of both algorithms. Second approach (parallelizing function evaluation) boosts BGH solver performance. Finally, it has been proved that wholly parallel version of BGH solver is more efficient with respect to processing time.**

***Index Terms*—differential equations, Brayton-Gustavson-Hatchel ODE solver, Gear-Nordsieck ODE solver, parallel computations**

I. Introduction

THE PROBLEM of parallelism introduction in the field of ordinary differential equations (ODE) systems is not new. During the last 50 years three main directions of parallel-techniques have been investigated [4] :
- across the method — e.g. independent stages of Runge-Kutta or extrapolation integrators evaluated in parallel;
- across the problem — e.g. waveform relaxation;
- across the time-domain - e.g. PINT, PFASST .
This paper will be devoted to parallelization across the method, applied to the field of stiff ODE solvers.

II. Basic features of Brayton-Gustavson-Hatchel ODE solver

Gear-Nordsieck method [3] is at present a classic tool for solving of stiff ordinary differential equations (ODE). Critical analysis of backward differentiation formulas (BDF) method helped to select the possible challenger - a method developed some 20 years ago by Brayton, Gustavson and Hatchel (in further course BGH method) [5]. A problem to be solved, i.e. ODE system, may be written in the form:

$$f(x, \dot{x}, t) = 0, \qquad 0 \le t \le T \tag{1}$$

where f is a vector (a set of functions). The Gear method uses Nordsieck vector components

$$(x_n, h_n, \dot{x_n}, 1/2h_n^2\ddot{x}_n, ..., (1/k!)h_n^k y_n^{(k)}) \tag{2}$$

as basic backward information. Brayton, Gustavson and Hatchel have forwarded the thesis that usage of backward information in the form

$$x_{n-j}, \qquad j = 0, 1, ...k \tag{3}$$

This work was not supported by any organization

is more efficient and leads to stable formulas, even for rapidly changing step size h.

The implementation of BGH method developed by the author [5] features variable order, operation count has been reduced and new efficient error control algorithm has been introduced. Predictor and corrector coefficients are computed through actualisation of old ones. Antisymmetry of square arrays is taken into account. Two-dimensional arrays are effectively indexed as one-dimensional. New values of step size reduction and expansion coefficients have been introduced. Asymmetric dead space in order changing section helps eliminate unnecessary order thrashing. The results of comparison of BGH algorithm [5] and open source version of Gear-Nordsieck algorithm [3], show competitiveness of BGH algorithm.

III. An example of real world ODE system – heat transfer problem

An example of heat diffusion through the wall will be used for comparison of both algorithms in sequential and parallel versions. The heat conduction equation for this case has the form

$$\frac{\partial T}{\partial t} = \frac{\lambda}{c_p \rho} \frac{\partial^2 T}{\partial x^2} \tag{4}$$

where T - temperature depends on both time and place in the wall, t - time, λ - heat transfer coefficient in the wall material, c_p - concrete heat capacity coefficient, ρ - concrete density, x - coordinate location measured across the wall.

In order to solve this parabolic partial differential equation, the derivative in space can be represented in differential form by dividing the wall thickness L into a finite number of N nodes. A system of ordinary differential equations describing temperature changes over time in individual nodes is then obtained:

$$\frac{dT_j}{dt} = \frac{\lambda}{c_p \rho} \frac{T_{j-1}^n - 2T_j^n + T_{j+1}^n}{(\Delta x)^2} \tag{5}$$

This equation describes temperature changes in nodes located inside the wall. The temperature at the edge nodes (left and right wall surfaces) can be determined from simple algebraic equations averaging the temperature inside and outside the wall.

The heat diffusion through the wall is now described by the N-2 system of first order ordinary differential equations (5) and two algebraic equations. This problem can be easily scaled, i.e. the number of differential equations (5) can be changed.

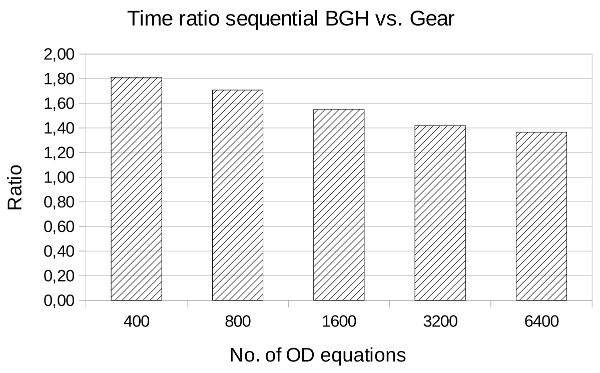

Fig. 1. Execution time ratio of BGH algorithm vs. Gear algorithm - sequential versions

IV. Comparison of Brayton-Gustavson-Hatchel and Gear-Nordsieck sequential ODE solvers

The implementation of Gear-Nordsieck algorithm redesigned by J. P. Moreau [3] has been selected for the comparisons in current research. The tests reported here have been performed on the computer with SkyLake processor. It features four physical cores (threads) and the additional four virtual cores/threads (hyperthreading). The source code of both algorithms has been compiled with C/C++ compiler version 8.3.1 and subsequently has been run on the Skylake desktop (4 physical cores) with Linux Fedora 31 operating system.

Temperature distribution in the concrete wall (thickness = 0.1 m) have been computed for the time points 2, 4, 6 sec. Efficiency of sequential BGH algorithm, in the sense of execution time, is only slightly inferior to Gear algorithm (fig. 1). It can be observed that the advantage of Gear algorithm diminishes with increased number of ordinary differential equations. For 400 equations Gear algorithm outperforms BGH algorithm by the factor of almost 2.0 but for 6400 equations this factor falls to 1.2.

V. Profiling of BGH and Gear ODE solvers

It is advisable, before any form of software tuning, to locate critical sections, functions and subroutines, consuming meaningful execution time. Profiling of both algorithms (for 800 equations) implementations has been carried out with the aid of valgrind/callgrind tool. Two following tables (table I and table II) present representative sample data of sections/subroutines call count and approximate percent share of execution times.

TABLE I
Callgrind profiling of serial BGH algorithm

	approx. %% time	no. of calls
BGHstiff	1.17	8000
solve	95.36	812
decomp	2.52	46
HeatTransfer	0.67	74414
interp	0.11	8000
predictor	0.06	1208

TABLE II
Callgrind profiling of serial Gear-Nordsieck algorithm

	approx. %% time	no. of calls
gear4	6.95	4
awp	5.85	16
engl45	49.1	483351
dgl14	28.95	213
gausol	3.14	8000
gaudec	2.88	71

Subroutines performing linear equations LU decomposition and solving `gausol` and `gaudec` are counterparts of `decomp` and `solve`. Computation of the function to be integrated is located in subroutines `engl45`, `dgl14` vs. `HeatTransfer`. It is apparent that the function evaluation (formula (5) for BGH solver) is one candidate for parallelization (see section 6).

Another parallelization candidate is linear equation solving routine `gausol` and `solve`. Both these routines consume more execution time than LU decomposition routines. However, such conclusion and approach is superficial and naive. It has been proved elsewhere [6] that the decomposition routines `gaudec` and `decomp` are more promising with respect to parallelization (see section 7).

The number of the linear equations routines calls does not depend on the number of nodes, i.e. on the number of differential equations. However, the dimension of the system rises with the square of the nodes number. It is quite reasonable to expect, that parallelization will be more efficient in the case of larger, fine-grained systems. Different results may be expected in the case of parallelization and fine-tuning of function evaluation. In the case of 400 differential equations BGH algorithm performs about 22% of function evaluations with respect to Gear algorithm. For 6400 ordinary differential equations this ratio falls to 1%. It may be expected that BGH algorithm will be more efficient in the case of more complex ODE formulas.

VI. Parallelization of function computation in Gear and BGH ODE solvers

At first the results of parallel computation of ODE function (5) will be presented. Parallelization will be implemented in both cases with the aid of POSIX pthreads library [1, 2]. Computation is performed in N nodes across the wall. Quite naturally, this set of N computations may be divided into the segments assigned to individual cores through creation of appropriate threads.

Parallelization has limited influence on computation efficiency (timing) for 400 equations (fig.2). The situation is better for 800 equations, as forking of 4 threads speeds-up the processing by the factor of 3.5, reaching 6 for 8 threads. For 1600 equations, the speed-up factor reaches the value of 4 for 4 threads and 7 to 8 for 8 threads. Limited speed-up

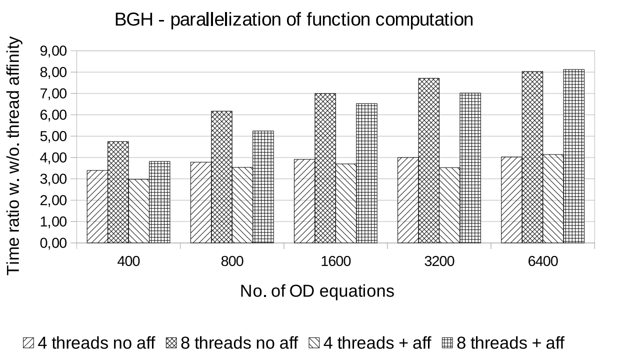

Fig. 2. The effect of function (5) parallelization in BGH algorithm; 4 and 8 threads without and with threads affinity

for lower equations count results from overhead of threads forking. In another series of experiments, an attempt to limit the overhead of threads forking has been implemented. It has been achieved through fixed bounding of individual threads with specific processor cores (affinity mechanism). For lower equation count, introduction of threads affinity has adverse influence on computation efficiency (fig. 2). The setting of threads affinity incurs higher overhead of thread forking. For higher equation count (e.g. at 1600 equations and above) this additional overhead is relatively small, compared with real number crunching inside individual threads. Summing it up - there has been no execution speed-up due to threads affinity setting.

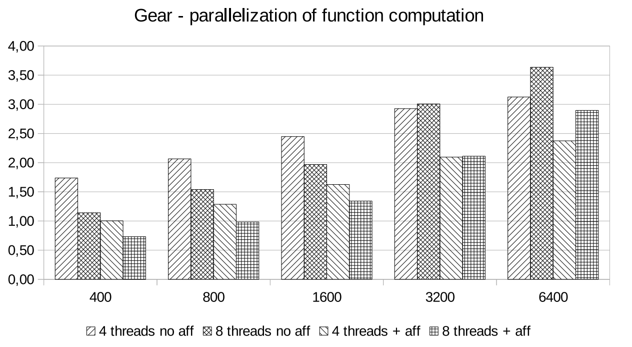

Fig. 3. The effect of function (5) parallelization in Gear algorithm; 4 and 8 threads without and with threads affinity

Similar experiments with parallelization of function evaluation in Gear-Nordsieck algorithm, presented in fig. 3, are disappointing. In the case of basic 4-thread parallelization, the speed-up ranges from 2 (800 equations) up to 3. Hyperthreading improves slightly the results for higher number of equations. Affinity setting degrades the performance. Comparison of BGH and Gear-Nordsieck solvers with parallelized function evaluation seems to confirm the preliminary profiling research. Parallelizing of this code section improves the performance of BGH solver almost by the factor equal to the number of processor cores. In the case of Gear-Nordsieck solver, the improvement is markedly lower, moreover for basic (without affinity) parallelization only.

VII. Parallelization of linear equation solving in Gear and BGH ODE solvers

The second area of the BGH and Gear algorithms, potentially amenable to parallelization and speed-up, is the linear equation solving section. Parallel linear equation solvers have been designed and tested very extensively [6, 2]. The efficiency of parallelization depends, among others, on the sparsity of the coefficient matrix. In general, parallel speed-up is larger in the case of rather dense matrices and falls down for the sparse ones.

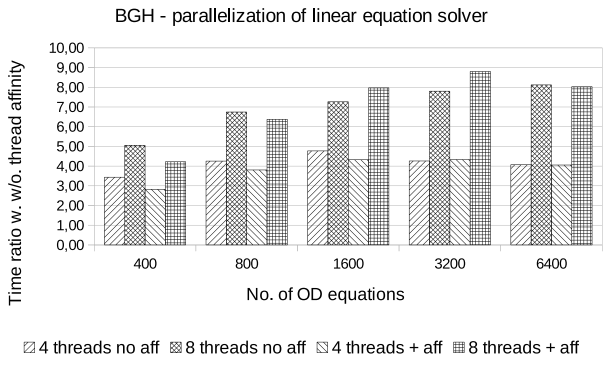

Fig. 4. The effect of linear equation solver parallelization in BGH algorithm; 4 and 8 threads without and with threads affinity

Straightforward parallelization (i.e. without thread affinity) results in speed-up proportional to the number of threads (fig. 4). The results for lowest equation count are slightly inferior, as the parallelization gains are offset by the overhead of threads forking. Similarly, as in the case of function evaluation, introduction of thread affinity does not improve efficiency.

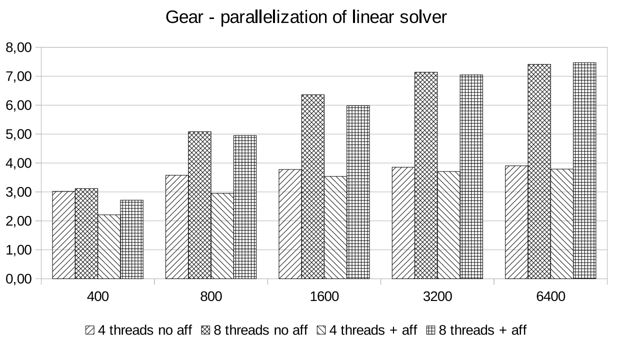

Fig. 5. The effect of linear equation solver parallelization in Gear algorithm; 4 and 8 threads without and with threads affinity

Parallelization of linear equation solver in Gear-Nordsieck algorithm leads to similar results. For higher equations/nodes number, the speed-up (fig. 5) is almost proportional to the number of forked threads. The improvement for lower equation count (800 and below) is lower than in the case of BGH solver. Also, the influence of affinity setting is negligible.

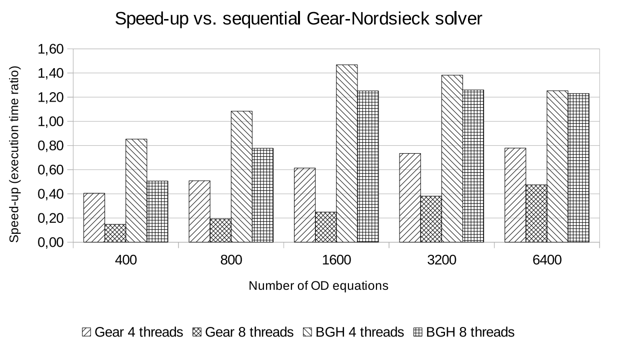

Fig. 7. Cumulative speed-up of multi-threaded solvers with linear equations and function computation vs. sequential Gear-Nordsieck solver

VIII. Comparison of the solvers with parallel function evaluation

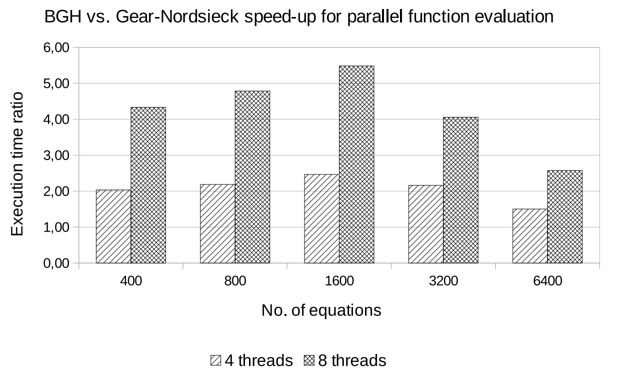

Fig. 6. Comparison of BGH and Gear-Nordsieck solvers with parallel function evaluation; 4 and 8 threads

It has been proved that parallelization of linear equation solving improves the performance of both solvers in almost equal degree with small advantage of BGH solver. Parallelization of function evaluation favors BGH solver. Direct comparison of such parallel versions of both solvers is presented in fig. 6. It follows that BGH solver is faster by the factor of 2 for 4-thread version, reaching the speed-up of 4 to 5 for 8-thread (hyperthreading) version. This advantage is a bit lower for higher equation count.

IX. Cumulative comparison of the solvers

In previous sections, two partial parallelization modifications of both solvers have been presented and investigated. However, the end user is rather interested in the final cumulative effect of these modifications. In order to perform such comparison, a basic sequential Gear-Nordsieck solver has been selected as the reference. Both solvers have been parallelized in a cumulative way, i.e. through parallel linear equation and function computation. First, it can be observed (fig. 7) that hyperthreading leads to inferior performance, as compared with 4-thread version. Next, parallel versions of Gear-Nordsieck solver are significantly slower than the sequential version. Third observation reveals good parallelization potential of BGH solver. Parallel version of BGH solver outperforms the fastest version of Gear-Nordsieck solver by 20-40% for higher equation count.

X. Conclusions

Comparison of basic sequential version of Gear-Nordsieck ODE solver and Brayton-Gustavson-Hatchel solver has shown that both solvers are almost equally efficient with regard to execution time. The performance of both solvers improves with rising number of differential equations. Two most promising sections of both solvers have been parallelized. The improvement of execution time has been observed in the case of linear equation solving parallelization. The speed-up has been proportional to the number of forked threads, at least for higher equation number. Parallelization of function evaluation has led to similar improvement only in the case of BGH solver. The speed-up in the case of Gear solver has been significantly lower than the threads number with limitation to most basic parallelization (without hyperthreading). These results conform to introductory profiling analysis. For both solvers, introduction of thread affinity in both parallelization cases, i.e. equation solving and function computation, has adverse influence or no influence on execution timing.

References

[1] J. Bylina. “A Framework for Generating and Evaluating Parallelized Code”. In: *Proceedings of the 2017 Federated Conference on Computer Science and Information Systems*. Vol. 11. 2017, pp. 493–496. DOI: 10.15439/2017F230.

[2] S. Fialko and V. Karpilovskyi. “Multithreaded Parallelization of the Finite Element Method Algorithms for Solving Physically Nonlinear Problems”. In: *Proceedings of the 2018 Federated Conference on Computer Science and Information Systems*. Vol. 15. 2018, pp. 311–318. DOI: 10.15439/2018F40.

[3] J. P. Moreau. *Website dedicated to numerical analysis*. 2021. URL: http://jean-pierre.moreau.pagesperso-orange.fr.

[4] S. I. Solodushkin and I. F. Iumanova. “Parallel Numerical Methods for Ordinary Differential Equations: a Survey”. In: *CEUR Workshop Proceedings*. Vol. 1729. 2016, pp. 1–10. URL: http://ceur-ws.org/Vol-1729/paper-01.

[5] M. Stabrowski. “Efficient Algorithm for Solving of Stiff Ordinary Differential Equations”. In: *Simulation Practice and Theory* 5 (1997), pp. 333–344. URL: https://www.sciencedirect.com/journal/simulation-modelling-practice-and-theory.

[6] M. Stabrowski. “Parallel Real-world LU Decomposition: Gauss vs Crout Algorithm”. In: *Open Computer Science* (2018), pp. 210–217. URL: https://www.degruyter.com/view/j/comp.