Analysis of Parallel Computing Techniques on Generalized-α method

***Abstract*— This paper presents a time reduction study and comparative analysis of numerical simulations of the popular Generalized-α method (both explicit and implicit) which is used in computational structure dynamics using parallel computing tools like OpenMP, CUDA and MPI.**

I. INTRODUCTION

Computational structure dynamics problems can be solved using various iterative methods. One of the most popular techniques is the generalized-α method. This family of methods was initially developed by Chung and Hilbert for the resolution of dynamics in the context of computational mechanics of solids. Cardona and Gradin adapted the method to compute the dynamics of multibody systems. We are considering two special schemes within this family of methods, namely Central-Difference(explicit) method and Newmark-β (implicit) method.
The discussion below deals with the in numerical simulations of above methods with and without using various parallel computing techniques. Parallel computing tools like OpenMP, MPI and CUDA have been used.

II. THEORY

The semidiscrete initial value problem for linear structural dynamics problem is given by

$$\mathbf{M}\ddot{u}(t) + \mathbf{K}u(t) = \mathbf{F}(t) \tag{1}$$

where **M** and **K** are mass and stiffness matrices, **F**(t) is the vector for applied forces on masses and $u(t)$ indicates the displacement vector.

The time domain is discritized into small quanta of size Δt . At any given time instant given by 'n' the quantities acceleration a_n,velocity $v_n$and displacement u_n satisfy

$$\mathbf{M}a_n + \mathbf{K}u_n = \mathbf{F}_n \tag{2}$$

The Taylor series expansion of v_{n+1} and u_{n+1} gives

$$u_{n+1} = u_n + \Delta t v_n + \Delta t^2[(1-2\beta)a_n + 2\beta a_{n+1}]/2 \tag{3}$$

$$v_{n+1} = v_n + \Delta t[(1-\gamma)a_n + \gamma a_{n+1}] \tag{4}$$

where β and γ decides the stability and accuracy of the algorithm.
Now predictor corrector method is used for calculating u_{n+1} and v_{n+1}
Predictor Step:-

$$\tilde{u}_{n+1} = u_n + \Delta t v_n + \Delta t^2(1-2\beta)a_n/2 \tag{5}$$

$$\tilde{v}_{n+1} = v_n + \Delta t(1-\gamma)a_n \tag{6}$$

Corrector Step:-

$$u_{n+1} = \tilde{u}_{n+1} + \beta\Delta t^2 a_{n+1} \tag{7}$$

$$v_{n+1} = \tilde{v}_{n+1} + \gamma\Delta t a_{n+1} \tag{8}$$

Assuming a_n and a_{n+1} satisfy the initial value problem their value will be

$$a_n = [\mathbf{M} + \beta\Delta t^2\mathbf{K}]^{-1}[F_n - \mathbf{K}u_n] \tag{9}$$

$$a_{n+1} = [\mathbf{M} + \beta\Delta t^2\mathbf{K}]^{-1}[F_{n+1} - \mathbf{K}\tilde{u}_{n+1}] \tag{10}$$

The analysis below works on the premise of **K** being a tri-diagonal matrix (which implies that no two non-adjacent masses are connected by a spring), for whose inverse a closed form solution given by *Moawwad El-Mikkawy et al.* exists. Two special cases are considered here :-

A. Newmark-β Method

The Newmark-β Method uses β=1/4 and γ=1/2. This method is also known as the implicit method because it also uses the value of acceleration of current time step along with those of the previous time step for the calculation of u_{n+1}. This method is unconditionally stable and 2^{nd} order accurate.

B. Central Difference Method

The Central Difference Method uses β=0 and γ=1/2. This method is also known as the explicit method because it uses the value of acceleration of only the previous time step for the calculation of u_{n+1}. This method is conditionally stable and 2^{nd} order accurate.

III. PROBLEM UNDER CONSIDERATION

The problem considered for the discussion below is a system of 100 masses of $1kg$ each connected by 101 springs of stiffness kN/m to only their adjacent masses. The 50^{th} mass is given an initial impulse of 2 m/s and the displacements of the 49^{th}, 50^{th} and 51^{st} masses are considered for all subsequent comparisons. The value of Δt is taken to be 0.01 seconds for the explicit method after suitable considerations using Octave. ($\Delta t < \frac{2}{w_{max}}$). The simulations are conducted for 10 seconds , or equivalently 1000 Δt steps.

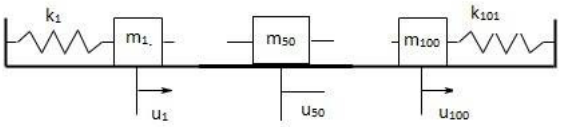

Fig. 1. Spring-Mass System

IV. Results and Data Analysis

The Following Graphs depict the Displacement of 50^{th} mass with respect to time. For Different tools of parallel computing, we can see that the graphs almost remain similar.

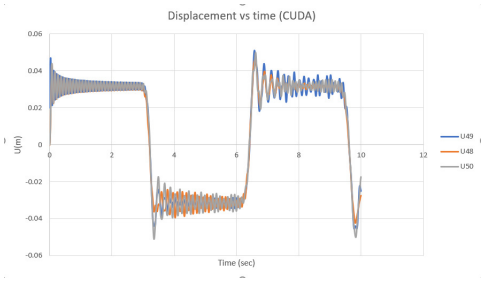

Fig. 2. CUDA

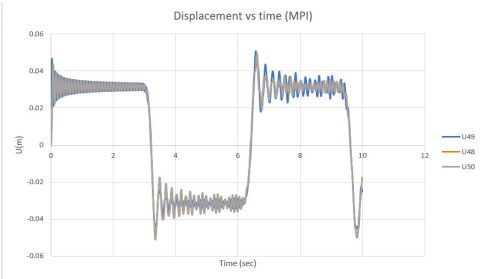

Fig. 3. MPI

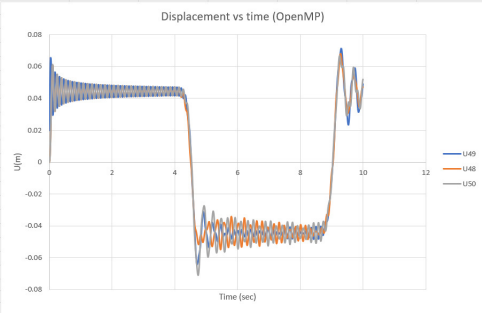

Fig. 4. OpenMP

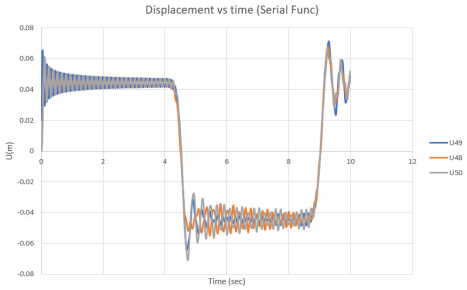

Fig. 5. Serial with function

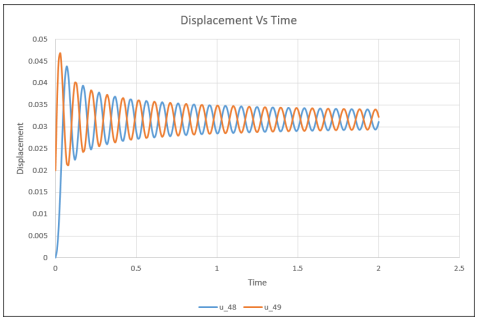

Fig. 6. Displacement 49^{th} and 50^{th} Mass over time

The following Graph depicts the displacement of 49^{th} and 50^{th} Mass over time. We can observe that the displacements of adjacent masses is opposite to each other.
The following Table depicts the average time taken by different codes to execute.

Method	Average Time Elapsed (in seconds)
Explicit-Serial(w/o) function	0.0169284
Explicit-Serial with function	0.2039362
Explicit-MPI	0.02653145
Explicit-OpenMP(w/o) function	0.0371246
Explicit-OpenMP with function	0.662123
Explicit-CUDA	0.1339292
Implicit-Serial	3.871794
Implicit-OpenMP	45.53616

Average-Time Study for different codes

V. CONCLUSIONS

The displacements for the serial code and the OpenMP code were found to be in good agreement with each other. The displacements for the MPI code and the CUDA code were found to be in good agreement with each other. The graphs are of a similar nature with slight differences in both phase as well as magnitude.

The deviation between these sets of values can be attributed to the propagated error while conducting a floating point operation. A simple analysis on Octave revealed this error to be 0.02 % per flop which percolated through thousands of such operations to cause a visible deviation.

The time-study reveals MPI to perform better than CUDA and OpenMP in case of the explicit method. This can be because CUDA threads are not suited for heavy computations.

Also, large amounts of data transfer between the host and kernel bogs CUDA down.

In case of OpenMP, the overhead of calling functions makes the code slower than it would have been without functions. This can be improved by using OpenMP in a explicit code that updates ith values rather than the current one that calls functions for each Matrix Vector Operations.

The much more general implicit method takes a lot more time to solve since the overhead associated with calling and executing the inverse function is quite large. OpenMP slows the already not-so parallel code even further.

It must be noted that background processes also contribute to the errors in the above time-values.

REFERENCES

[1] MoawwadEl-Mikkaw, Abdelrahman Karawia, Inversion of general tridiagonal matrices (Applied Mathematics Letters)

[2] Newmark, N. M. (1959) A method of computation for structural dynamics. Journal of Engineering Mechanics, ASCE, 85 (EM3) 67-94.