J. Appl. Comput. Mech., 8(3) (2022) 853-864
DOI: 10.22055/JACM.2020.33053.2137

ISSN: 2383-4536
jacm.scu.ac.ir

Shahid Chamran University of Ahvaz

Journal of Applied and Computational Mechanics

Research Paper

A Comparative Study on the Efficiency of Compiled Languages and MATLAB/Simulink for Simulation of Highly Nonlinear Automotive Systems

Amir Hossein Pasdar[1], Shahram Azadi[2], Reza Kazemi[3]

[1] Infinity Industry Designers, Inc., No. 1 Modarres Street, Ferdows, 97718-15583, Iran, Email: hpasdar@gmail.com

[2] Department of Mechanical Engineering, K. N. Toosi University of Technology, No. 17 Pardis Street, Vanak Square, Tehran, 19991-43344, Iran, Email: azadi@kntu.ac.ir

[3] Department of Mechanical Engineering, K. N. Toosi University of Technology, No. 17 Pardis Street, Vanak Square, Tehran, 19991-43344, Iran, Email: kazemi@kntu.ac.ir

Received March 26 2020; Revised May 17 2020; Accepted for publication May 25 2020.
Corresponding author: A.H. Pasdar (hpasdar@gmail.com)

Abstract. In the present paper, a comparison between the simulation performance of a highly nonlinear model in MATLAB/Simulink and a compiled language has been drawn. A complete powertrain layout was formed in Simulink and the same model was developed from scratch in Fortran 2003 which led to creating a complete simulation software program named Powertrain Simulator. The results show that for a system with not many details and phase changes, both of the simulation environments offer acceptable performance. However, when the modeling layout is overly complicated, developing the model in a compiled language is a smarter choice.

Keywords: Performance comparison; Nonlinear systems; Vehicle dynamics simulation; Fortran 2003; Simulink.

1. Introduction

Languages and software programs that can be used to simulate dynamical phenomena abound. MATLAB and Simulink have been very popular choices among engineers and scientists because of their numerous libraries in different areas and their ease of use. In the field of Automotive Engineering, many libraries and toolboxes have been incorporated into Simulink which has made the software even more attractive in this area. Engineers frequently utilize MATLAB and Simulink in hardware-in-the-loop applications for faster implementation of a control program on the real physical system. Simscape Driveline has recently become a popular choice for simulation of vehicle powertrains and there have been other software tools around which were developed specifically for simulating vehicle dynamics performance, from which SIMPACK, Adams, VeDYNA, Siemens PLM Software, CarSim and ASM Vehicle Dynamics are just a few to name. However, as researchers are continuously furthering more modeling concepts and incorporating more details into their simulations, sometimes the software programs might not meet their conditions and mostly they fall back on MATLAB and especially Simulink to develop their models from scratch.

Although MATLAB and specifically Simulink are advertised for their convenience and capabilities to rapidly model any dynamical phenomenon, this convenience comes with some performance penalties. In this regard, there have been comparisons of simulation languages and environments including Simulink in various engineering fields. Giroux et al. [1] did research on the performance of Simulink for an inertial navigation system simulator. Hödlmoser and Kitzler [2] researched into a performance comparison between Simulink and other software in the implementation of fuzzy control of a two tank system. In another study, Lamberský and Vejlupek [3] evaluated the effect of Simulink optimization options on the performance of a DSPIC controller programmed with automatically generated code. Venayagamoorthy [4] compared the simulation times of a power system using Simulink and other software. Roscoe et al. [5] presented a research in which they benchmarked Simulink and studied optimization of the code for power systems control. Cansalar et al. [6] conducted related research in which they compared the simulation time of MATLAB/Simulink and LabVIEW for control applications. Zhang et al. [7] made a good comparison between the execution times of the heated hold-up tank problem in Simulink and in a developed C++ program. In another research work by Wang et al. [8], the relative performance of MATLAB and Simulink implementations of a reaction-diffusion system was compared. There has been another research concerning a comprehensive open source MR algorithm prototyping platform which was developed by Keerthi Sravan Ravi et al. [9] who included a comparison of program execution times with MATLAB. Along with the studies mentioned, there also have been attempts to make MATLAB more versatile by expanding its capabilities. Ivano Azzini et al. [10] developed open-source software that can enhance the program in multi-platform and multi-environment configurations and increase the efficiency and speedup. Luan Thanh Pham et al. [11] created a MATLAB program to improve its operability and suitability for the estimation of 3D geometry of magnetic interfaces. Benoît Barbot et al. [12] presented an implementation to integrate Simulink models into a statistical model checker for hybrid modeling and simulation.

Published online: February 27 2021

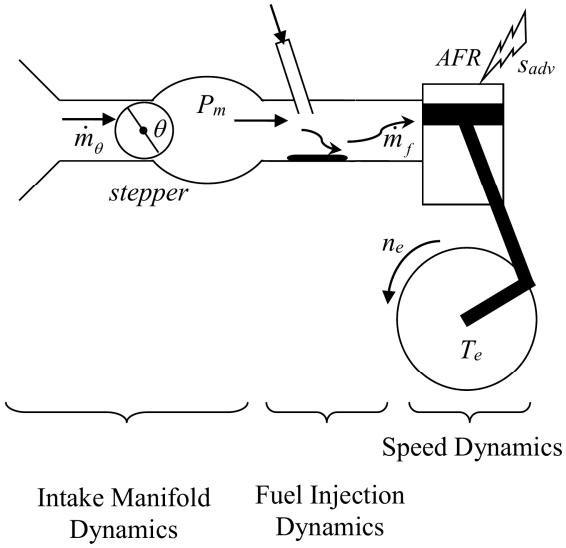

Fig. 1. The engine subsystems.

While there have been studies on the simulation performance of MATLAB/Simulink and other languages, a comparison from a performance point of view for highly nonlinear automotive systems is lacking and there is little scientific research to address this issue. This is mainly because modeling automotive systems is often an interdisciplinary approach and involves working in several subfields simultaneously which implies that in case of choosing a language with a smaller amount of abstraction between itself and the machine language, a higher amount of time will be spent on the modeling and validation part, as developing numerical codes from scratch in several fields would be inevitable. As a result of this, researchers have not investigated the potential of using a compiled language and the performance benefits it can bring about.

Although there are situations where Simulink code can be translated to various target languages [13], the current study aims at a direct comparison between the run time performance of the Simulink model and the same model in a compiled language which has been developed independently. Therefore, a comparison between Simulink, which is frequently used by researchers for simulating automotive systems, and Fortran 2003 as a representative of the compiled languages has been performed. The ease of programming in Fortran, providing legacy support for older codes and its superb adaptation to numerical calculation were the main reasons of choosing it over other similar languages like C and C++. MATLAB-like languages like Java and Python were excluded from the comparison as MATLAB is superior in integrity, support and toolboxes to them. Thus, only MATLAB/Simulink and Fortran 2003 were considered for the test.

The comparison has been done in two areas. First, it will be investigated how the two languages perform in simulating phenomena that do not involve phase changes and complicated modeling layouts. This will include testing the longitudinal performance of a vehicle with a conventional powertrain without gearshifting. Second, a thorough simulation will be done which will involve a complete drag-like test in which the car will accelerate from a low speed in the first gear until it reaches a high speed in the overdrive, and the complete procedure of the gearshiftings will be simulated. A real-world model has been utilized for both of the run modes which consists of a highly detailed representation of the conventional powertrain [14].

2. Modeling of the System

The complete system is comprised of the engine, clutch, gearbox and the longitudinal vehicle model, all of which are connected through the transmitting shafts. The full details of modeling of the system have been presented in [14], and in the current paper only the computational flow of the solution is presented.

The computational model of the system has an average of 27 state-state variables. There are some others that are calculated via empirical data, or are extracted from the state values. Based on the state of engagement in the clutch and gearbox, there are 7 modes which completely change the formation of most of the equations, and can divide or merge variables. What is more, the state of the whole system changes if it is in acceleration, deceleration, upshift or a downshift. These together make this model one of the most computationally intensive problems to be solved numerically.

In the following sections, the governing equations for the corresponding subsystem to compute the derivatives of the state variables and other related parameters will be discussed.

2.1 Engine Dynamics

The complete car model utilizes a semi-empirical, mean value engine model to simulate the combustion engine part of the powertrain. The model will include the intake manifold dynamics, fuel injection dynamics and the speed dynamics as presented in Fig. 1. This model precisely relates the torque produced by the engine with the pedal inputs and the surrounding conditions. There are 2 state variables in the engine model as the engine rotational speed and the manifold pressure denoted by n_e and P_m .

To compute the derivatives of these state variables, we need to first calculate air-to-fuel ratio or *AFR* . This value is calculated based on the empirical data from the following equation:

$$\begin{aligned} \mathrm{AFR} = {} & \mathrm{p}_{00} + \mathrm{p}_{10} \times \mathrm{n}_e + \mathrm{p}_{01} \times \dot{m}_f + \mathrm{p}_{20} \times \mathrm{n}_e^{\,2} + \mathrm{p}_{11} \times \mathrm{n}_e \times \dot{m}_f + \mathrm{p}_{02} \times \dot{m}_f^{\,2} + \mathrm{p}_{30} \times \mathrm{n}_e^{\,3} + \\ & \mathrm{p}_{21} \times \mathrm{n}_e^{\,2} \times \dot{m}_f + \mathrm{p}_{12} \times \mathrm{n}_e \times \dot{m}_f^{\,2} + \mathrm{p}_{03} \times \dot{m}_f^{\,3} \end{aligned} \tag{1}$$

where p_{ij} are empirical parameters and $\dot{m}_f$ is the rate of fuel input to the engine. Calculating spark advance is done according to the following:

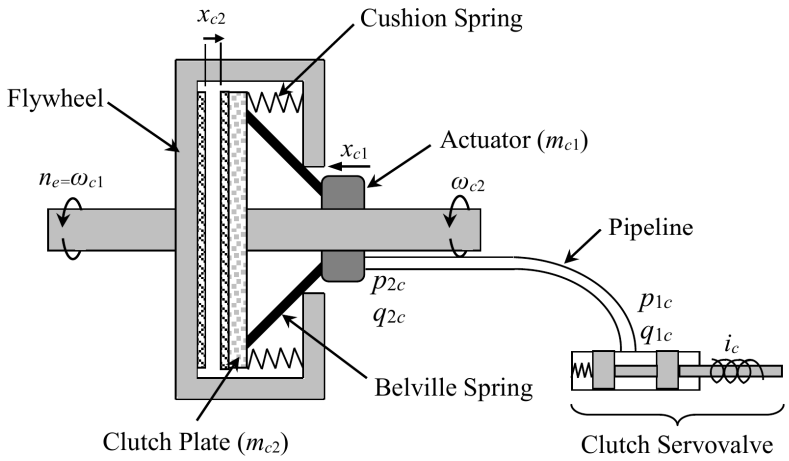

Fig. 2. The clutch system includes the servovalve, piping and the actuator along with the mechanical parts.

$$\begin{aligned} s_{adv} = {} & p_{00} + p_{10}\times n_e + p_{01}\times \dot{m}_f + p_{20}\times n_e^{\,2} + p_{11}\times n_e\times \dot{m}_f + p_{02}\times \dot{m}_f^{\,2} + p_{30}\times n_e^{\,3} + \\ & p_{21}\times n_e^{\,2}\times \dot{m}_f + p_{12}\times n_e\times \dot{m}_f^{\,2} + p_{03}\times \dot{m}_f^{\,3} + p_{40}\times n_e^{\,4} + p_{31}\times n_e^{\,3}\times \dot{m}_f + p_{22}\times n_e^{\,2}\times \dot{m}_f^{\,2} + \\ & p_{13}\times n_e\times \dot{m}_f^{\,3} + p_{50}\times n_e^{\,5} + p_{41}\times n_e^{\,4}\times \dot{m}_f + p_{32}\times n_e^{\,3}\times \dot{m}_f^{\,2} + p_{23}\times n_e^{\,2}\times \dot{m}_f^{\,3} \end{aligned} \tag{2}$$

where, like air-to-fuel ratio, p_{ij} are different empirical constants. Then, the stepper motor value will be obtained:

$$\begin{aligned} Stepper = {} & p_{00} + p_{10}\times n_e + p_{01}\times P_m + p_{20}\times n_e^{\,2} + p_{11}\times n_e\times P_m + p_{02}\times P_m^{\,2} + p_{30}\times n_e^{\,3} + \\ & p_{21}\times n_e^{\,2}\times P_m + p_{12}\times n_e\times P_m^{\,2} + p_{40}\times n_e^{\,4} + p_{31}\times n_e^{\,3}\times P_m + p_{22}\times n_e^{\,2}\times P_m^{\,2} \end{aligned} \tag{3}$$

It should be noted that the empirical coefficients i.e., p_{ij} are different for each variable.

Based on the throttle input θ, ambient pressure P_{amb} and manifold pressure, one can obtain the rate of air mass past the throttle:

$$\dot{m}_\theta = 2\sqrt{\min\left[\frac{P_m}{P_{amb}},\frac{P_{amb}}{P_m}\right]-\left(\min\left[\frac{P_m}{P_{amb}},\frac{P_{amb}}{P_m}\right]\right)^2}\times\left(C_1\times\theta^2+C_2\times\theta+C_3+C_4\times\theta^3\right)\times sign(P_m-P_{amb}) \tag{4}$$

where C_i are constants adopted from the semi-empirical model. Now, we can calculate the derivative of manifold pressure and the rate of fuel input to the engine:

$$\begin{aligned} \dot{P}_m &= K_1\times\dot{m}_\theta + K_2\times Stepper - \left(A_1 + A_2\times P_m + A_3\times n_e + A_4\times P_m^{\,2} + A_5\times P_m\times n_e + A_6\times n_e^{\,2}\right) \\ \dot{m}_f &= \left(A_1 + A_2\times P_m + A_3\times n_e + A_4\times P_m^{\,2} + A_5\times P_m\times n_e + A_6\times n_e^{\,2}\right)\times\eta_{ne}\times\eta_{pm} \end{aligned} \tag{5}$$

where K_i, A_i are empirical constants and η_{ne} and η_{pm} are the efficiencies related to the engine and manifold structures. Several other parameters are calculated in the routine which are based on the variables discussed so far and have similar equation complexity.

The burned fuel can be computed by:

$$\dot{m}_{fb} = K_3\times\dot{m}_f\times n_e \tag{6}$$

where K_3 is another calculated parameter. Finally, the generated torque by the engine can be computed by:

$$\begin{aligned} T_e = {} & D_1 + D_2\times\dot{m}_f + D_3\times AFR + D_4\times AFR^2 + D_5\times s_{adv} + D_6\times s_{adv}^{\,2} + D_7\times n_e + D_8\times n_e^{\,2} + \\ & D_9\times n_e\times s_{adv} + D_{10}\times s_{adv}\times\dot{m}_f + D_{11}\times s_{adv}^{\,2}\times\dot{m}_f + B_1 + B_2\times\dot{m}_f + B_3\times\dot{m}_f^{\,2} + B_4\times\dot{m}_f^{\,3} \end{aligned} \tag{7}$$

where D_i and B_i are constants derived from experiments.

The derivative equation of the other state variable of the engine i.e., the engine speed will be discussed in the next sections where the coupling of the engine with the clutch is discussed.

This set of equations presents a system of differential algebraic equations. In fact, every subsystem of the model is a set of DAE and no analytical solution is present.

2.2 Clutch Dynamics

The clutch part of the model introduces the most nonlinearity to the model. It includes the subsystems of the servovalve, pipeline, the actuator and the torque-transmitting characteristics. Figure 2 shows the schematic drawing of the clutch system.

One of the main characteristics of a clutch is how the force is transmitted through the Belville springs. They introduce nonlinearity and hysteresis based on the direction of motion. The computation of the generated force is done through two polynomials whose coefficients denoted by k_i are obtained through empirical data, and the force is calculated based on the pressure of the actuator, position and velocity of the release bearing to check which polynomial should be used. The general representation of this force is as follows:

$$f_{xv}(x_{c1}, v_{c1}, p_{2c}) = \sum_i k_i x_{c1}^{\,i-1} \tag{8}$$

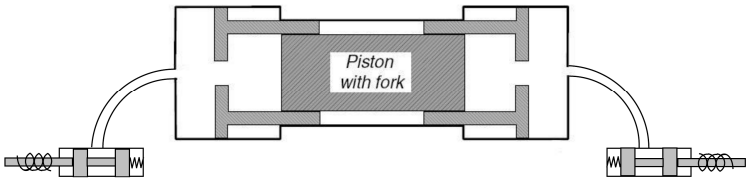

Fig. 3. An automated gearbox which has two distinct electrohydraulic actuators for moving the fork from one side to the other.

The release bearing movement and velocity are denoted by x_{c1} and v_{c1}. The downstream oil pressure which is applied to the bearing is p_{2c} and the downstream flow is represented by q_{2c}. In the same way, the equivalent force imposed on the clutch plate is calculated by:

$$f_{cu}(x_{c2}) = \sum_{i} k_i x_{c2}^{\,i-1} \tag{9}$$

which is a function of clutch plate position x_{c2}. Considering above, based on whether clutch actuation is needed for gearshifting or not, the following set of state equations depict the clutch system and its hydraulic actuator:

$$\begin{aligned} \dot{x}_{c1} &= v_{c1} \\ \dot{v}_{c1} &= \frac{-b_{c1} \times v_{c1} - f_{xv} + A_{ac} \times p_{2c}}{m_{c1}} \\ \dot{x}_{c2} &= v_{c2} \\ \dot{v}_{c2} &= \frac{-b_{c2} \times v_{c2} + c_r \times f_{xv} - f_{cu}}{m_{c2}} \end{aligned} \tag{10}$$

where b_{c1} and b_{c2} are the damping coefficients of the release bearing and the clutch plate respectively, and their masses are denoted by m_{c1} and m_{c2}. A_{ac} is the cross-sectional area of the actuator where pressure p_{2c} is present, and c_r is the coupling ratio of the Bellville springs. The governing equations of the pipeline system are represented by:

$$\begin{aligned} \dot{q}_{2c} &= K_1 \times p_{1c} + K_2 \times q_{2c} + K_3 \times p_{2c} + K_4 \times q_{1c} \\ \dot{p}_{1c} &= K_5 \times (q_{1c} - q_{2c}) \\ \dot{p}_{2c} &= \frac{K_6 (q_{2c} - A_{ac} \times v_{c1})}{K_7 + A_{ac} \times x_{c1}} \end{aligned} \tag{11}$$

where $K_i (i = 1,7)$ are parameters calculated from the physical characteristics of the pipe and the oil used, and q_{1c} is the upstream (servovalve) flow. In the same way, p_{1c} denotes the upstream (servovalve) pressure in the piping. The upstream flow is calculated by:

$$q_{1c} = K_8 \times sign(|p_{1c} - p_0|) \times \sqrt{|p_{1c} - p_0|} \times \sum_{j} A_{cs}(j)\, x_{cs}^{\,j-1} \tag{12}$$

where K_8 is another calculated parameter, p_0 is the reservoir pressure of the clutch servovalve, x_{cs} is the clutch servovalve plunger position and A_{cs} is a polynomial representing the cross-sectional orifice area of the servovalve based on the plunger movement. The calculation of upstream flow is done based on the position of its plunger as the filling and dumping orifice areas are different, and there is a dead band as well.

The clutch servovalve movements are modeled by:

$$\begin{aligned} \dot{x}_{cs} &= v_{cs} \\ \dot{v}_{cs} &= \frac{-F_{0cs} - k_{cs} \times x_{cs} - b_{cs} \times v_{cs} + k_{fcs} \times i_{cs} \times (1 - e^{-t/\tau_{cs}}) + C \times |p_{1c} - p_0| \times \sum_{j} A_{cs}(j)\, x_{cs}^{\,j-1}}{m_{cs}} \end{aligned} \tag{13}$$

where m_{cs}, v_{cs} denote the clutch servovalve mass and velocity, k_{cs}, b_{cs} are the servovalve plunger spring stiffness and damping coefficients, k_{fcs}, τ_{cs} are electromagnetic constants, i_{cs} is the input current to the servovalve and C is a calculated parameter.

The governing equations for the rotational movement of the clutch system will be discussed later along with the connecting shafts modeling.

2.2 Gearbox Dynamics

The gearbox is a typical multi-speed transmission which incorporates synchronizers. The gearbox itself acts as a separate subsystem when the whole powertrain is working. Whenever a gearshift is intended, it changes the gear ratio by moving the synchronizers. The joint with the other subsystems is the synchronizer dynamics which relate the torque flowing through the system to the force acting on the synchronizers for changing the gears. Whenever the torque to the force ratio crosses specific values, the gearbox does the action of disengagement from the current gear or synchronization to the next intended gear.

In a manual transmission, the mechanical force which is applied by the driver's hand moves the synchronizers, while in a fully automated transmission the force is applied by an electrohydraulic actuator similar to the clutch one. As presented in Fig. 3, in a fully automated gearbox two distinct electrohydraulic actuators move the fork between the two adjacent gears. The governing equations for the gearbox actuators are similar to the clutch ones, and the actuators only work to move the fork from one end to the other and vice versa. Because of this, substituting the generated force with a mean equivalent value can also be used to produce accurate results as the intention of the system is to move the gearbox collar to engage and disengage gears without feedback.

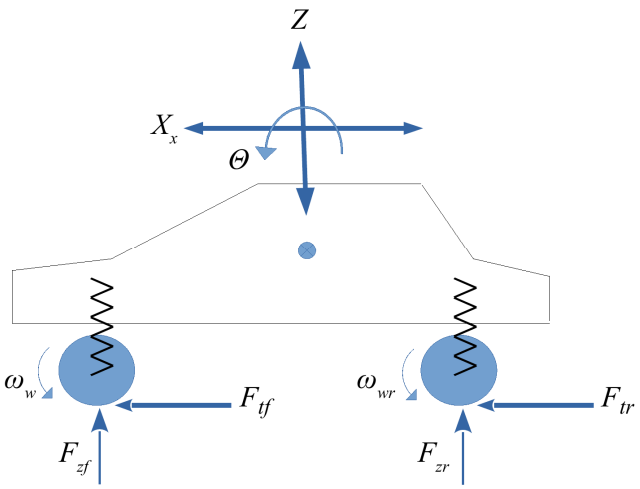

Fig. 4. The longitudinal model of the vehicle.

2.2 Transmission Shafts Dynamics

The whole powertrain and the car model are connected via the transmitting shafts. The modeling includes the gearbox input and output shafts, the longitudinal model of the tire and the interaction with the road. The existing elasticity and damping in the shafts have been taken into account and an advanced model of the tire suited for longitudinal performance has been utilized to correctly simulate the slip phenomenon. Modeling of both the driving wheel (front wheel) and the driven one (rear wheel) has been included.

If we consider n_e, ω_{c2} the angular velocities of the engine and the clutch, and in the same way ω_{g1}, ω_{g2} denote the input and output gearbox shaft speeds, we can construct the driving equations of the connecting shafts along with the engine speed as follows:

$$\begin{aligned}
\dot{\theta}_{cg} &= \omega_{c2} - \omega_{g1} \\
\dot{\theta}_{gw} &= \omega_{g2} - \omega_{w} \\
\dot{n}_e &= \frac{T_e - T_c}{J_{c1}} \\
\dot{\omega}_{c2} &= \frac{T_c - k_{cg} \times \theta_{cg} - b_{cg} \times (\omega_{c2} - \omega_{g1})}{J_{c2}} \\
\dot{\omega}_{g1} &= \frac{k_{cg} \times \theta_{cg} + b_{cg} \times (\omega_{c2} - \omega_{g1}) - b_g \times \omega_{g1} - k_{df} \times f_{df}}{J_{g1}} \\
\dot{\omega}_{g2} &= \frac{b_g \times \omega_{g1} + k_{df} \times f_{df} - k_{gw} \times \theta_{gw} - b_{gw} \times (\omega_{g2} - \omega_{w})}{J_{g2}}
\end{aligned} \tag{14}$$

where θ_{cg}, θ_{gw} are the angular torsion in the primary and secondary gearbox shafts, ω_w is the wheel speed, J_{c1} is the equivalent clutch inertia on the release bearing side, J_{c2} is the same inertia but on the clutch plate side, k_{cg}, b_{cg} are the rotational stiffness and damping in the primary gearbox shaft, k_{gw}, b_{gw} are the same parameters for the secondary shaft, J_{g1}, J_{g2} are the primary and secondary equivalent inertias, b_g is the rotational damping coefficient in the primary shaft support, f_{df} is the forced applied to the gearbox fork for engagement and disengagement, and k_{df} is the relating parameter to the torque flow. The clutch torque is calculated through the following equation based on the angular velocity difference between the clutch and the engine denoted by ω_{rel} and its state of engagement:

$$T_c(\omega_{rel}, n_c) = 2 \times r_{cm} \times n_c \times \sum_j (r_{cm}\omega_{rel})^{j-1} \times \mu_c(j) \tag{15}$$

where n_c is the clamp load in the clutch, r_{cm} is the mean radius of the clutch plate and μ_c is a polynomial defining the friction coefficients based on the relative angular velocity and the stiction or slip mode of the system.

2.2 Vehicle Governing Equations

The half vehicle model consists of pitch motion, bounce motion and the longitudinal movement of the vehicle. Furthermore, the vertical movement of the front and rear axles has been modeled. The selected degrees of freedom which are shown in Fig. 4 are enough to simulate the longitudinal performance of the vehicle.

The governing equations are computationally calculated by:

$$
\begin{aligned}
&\dot{\omega}_w = \frac{f_{dr} \times \left(k_{gw} \times \theta_{gw} + b_{gw} \times (\omega_{g2} - \omega_w)\right) - T_b - r_w \times F_{tf}}{I_w} \\
&\dot{X}_x = V_x \\
&\dot{V}_x = \frac{2F_t}{M_v} \\
&\dot{Z} = Z_v \\
&\dot{\Theta} = \Theta_v \\
&\dot{Z}_v = 2 \times \frac{k_{rs} \times (-Z + l_r \times \Theta) + c_{rs} \times (-Z_v + l_r \times \Theta_v) - k_{fs} \times (Z + l_f \times \Theta) - c_{fs} \times (Z_v + l_f \times \Theta_v)}{M_v} \\
&\dot{\Theta}_v = 2 \times \frac{-k_{fs} \times l_f \times (Z + l_f \times \Theta) - c_{fs} \times l_f \times (Z_v + l_f \times \Theta_v) - k_{rs} \times l_r \times (-Z + l_r \times \Theta) - c_{rs} \times l_r \times (-Z_v + l_r \times \Theta_v) + F_t \times h_g}{I_{yy}} \\
&\dot{\omega}_{wr} = \frac{-T_b - r_w \times F_{tr}}{I_w}
\end{aligned}
\tag{16}
$$

where ω_w, ω_{wr} are the rotational speeds of the driving and driven wheels (here the front and rear wheels), f_{dr} is the final drive ratio, T_b is the brake torque applied to the wheel, r_w denotes the wheel radius, I_w is the wheel inertia, X_x, V_x are the distance travelled and the speed of vehicle, M_v is total mass of the vehicle, Z, Θ represent bounce and pitch motions and Z_v, Θ_v denote their rate, k_{fs}, k_{rs} are the front and rear suspension stiffness, and in the same way c_{fs}, c_{rs} denote the front and rear suspension damping, l_f, l_r show the distance of vehicle center of gravity to the front and rear axles, and h_g is the height of center of gravity. Finally, I_{yy} is the total rotational inertia of the vehicle at the center of gravity.

The traction forces on the front and rear wheels i.e., F_{tr} and F_{tr} , are calculated by:

$$
\begin{aligned}
&F_{zr} = \frac{M_v \times g \times l_f \times \cos(\varphi)}{2(l_f + l_r)} + k_{rs} \times (-Z + l_r \times \Theta) + c_{rs} \times (-Z_v + l_r \times \Theta_v) \\
&F_{zf} = \frac{M_v \times g \times l_r \times \cos(\varphi)}{2(l_f + l_r)} - k_{fs} \times (Z + l_f \times \Theta) - c_{fs} \times (Z_v + l_f \times \Theta_v) \\
&\kappa_f = \frac{r_w \times \omega_w - V_x}{V_x} \\
&\kappa_r = \frac{r_w \times \omega_{wr} - V_x}{V_x} \\
&F_{tf} = F_{zf} \times D \times \sin\left(C \times \tan^{-1}\left(B \times \kappa_f - E \times \left(B \times \kappa_f - \tan^{-1}\left(B \times \kappa_f\right)\right)\right)\right) \\
&F_{tr} = F_{zr} \times D \times \sin\left(C \times \tan^{-1}\left(B \times \kappa_r - E \times \left(B \times \kappa_r - \tan^{-1}\left(B \times \kappa_r\right)\right)\right)\right)
\end{aligned}
\tag{17}
$$

In this set of equations g is the gravitational constant, φ is the road angle, κ_f, κ_r are the longitudinal slip values for the front and rear wheels, and the set of D, B, C are parameters that define the type of road the vehicle moves on. Having calculated the front and rear traction forces, one can obtain the total traction that drives the vehicle forward and is computed by:

$$
F_t = F_{tf} + F_{tr} - K_{rol} \times (F_{zf} + F_{zr}) - \frac{M_v}{2} \times g \times \sin(\varphi) - C_{drag} \times A_{vfr} \times V_x^{\,2} \tag{18}
$$

where K_{rol} is a coefficient related to rolling resistance, C_{drag} is the aerodynamic drag constant and A_{vfr} will be the frontal vehicle area.

3. Numerical Simulation

In order to correctly compare the two simulation environments, first, the complete dynamical model of the powertrain and the vehicle was designed in Simulink as it had already been done in [14]. Simce the equations of the system change whenever there is a gearshifting or mode of movement, the system in the different states was considered and Stateflow was utilized to model the combinatorial and sequential decision logic. The model was simplified as much as possible to better its performance [15], [16] and accelerator mode was chosen in all of the simulations. Moreover, the software was instructed to compile the code for the specific hardware used. To enhance the figure of the comparison, the simulation was performed on an older Intel® *Core™ i3-380M* Processor (3M Cache, 2.53 GHz). MATLAB version 2014a was chosen to do the task and Gentoo Linux 64bit was used as the underlying operating system for running the programs. To minimize any discrepancy between the results of the two programs, the solver ode45 was used in Simulink and a value of 1.e-5 was set as the maximum relative error while the maximum stepsize was defined as 0.1. Furthermore, zero crossing detection was disabled throughout the program. The solver chosen had no difficulty simulating the system and while producing accurate results, it was the fastest possible solver, too.

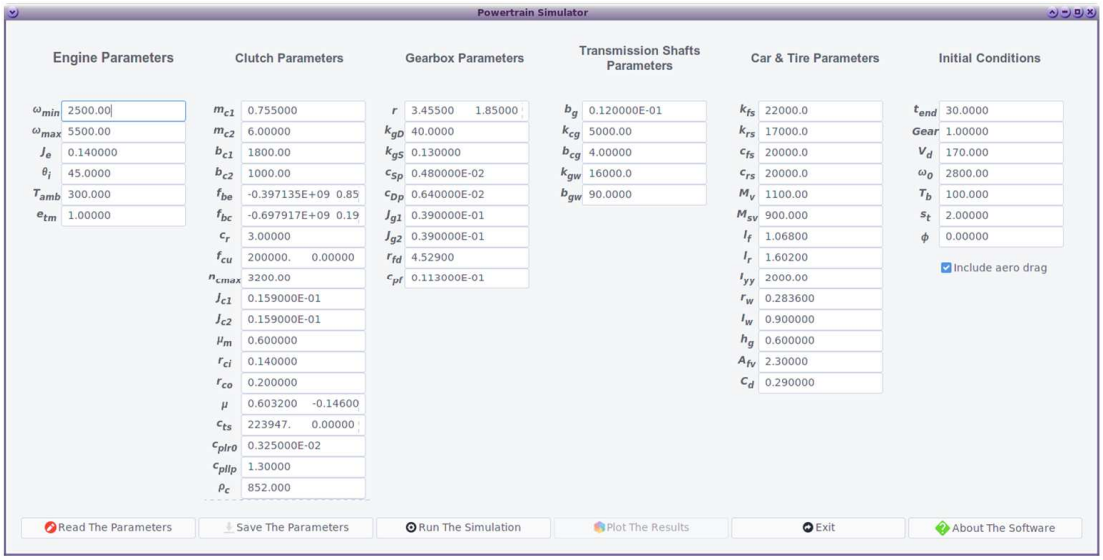

Fig. 5. Powertrain Simulator software written in Fortran 2003 and GTK.

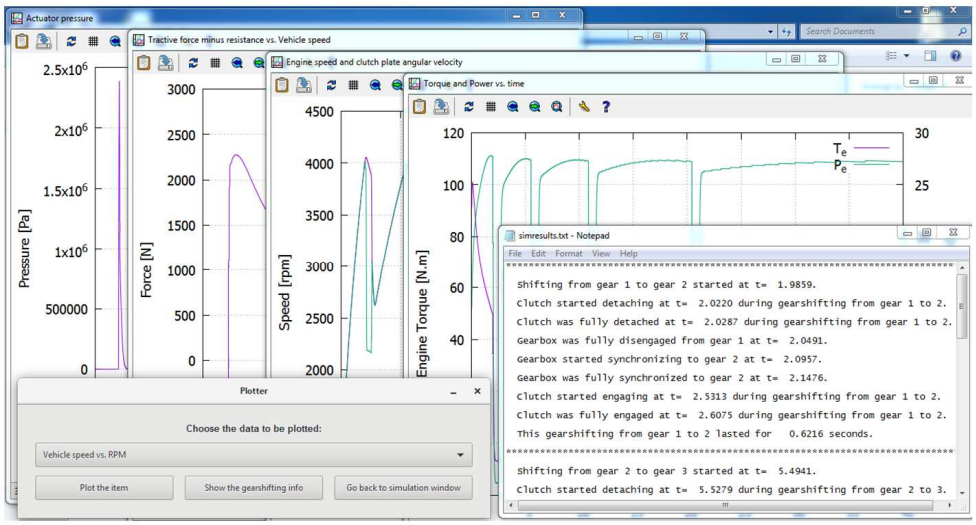

Fig. 6. The capability of Powertrain Simulator in plotting and extracting gearshifting details.

On the other hand, a software program which was named Powertrain Simulator was developed in Fortran 2003 from scratch that implemented the same model but in Fortran language. The equations were formed in the state-space representation and a fifth-order Runge–Kutta method using the Cash-Karp implementation as the adaptive stepsize algorithm [17] was utilized to solve the set of the ordinary differential equations (ODE). Considering the high level of details and phase changes that create a dynamic set of equations which need to be checked in a real-time manner, no pre-written codes or numerical library were used in the program as the modeling and the simulation type stipulated that the solver needs to be developed from scratch. The high-precision nature of modeling creates many variables in huge arrays and because of that, modern features of Fortran 2003 had to be deployed to store the variables and the intermediate results in linked lists and return the results of subroutines in allocatable arrays and derived types, thereby making it possible to code more efficiently and less complicatedly. These features are not present in an existing numerical library where the programmer is able to change the ODE system on-the-fly or allocate memory just as needed, since the existing solutios do not support advanced features of Fortran 2003.

In order to visualize the results, a plotter was also built into the program which was based on Gnuplot v5 [18] to produce high quality figures as part of the software core. Moreover, using the high level interoperability of Fortran 2003 with C, the program was wrapped in a Graphical User Interface (GUI) built in GTK3 toolkit using GTK-Fortran software [19] and all of the needed event signals and handlers were added to the GUI layer for easy interaction with the core program. An illustration of the program in GUI has been presented in Fig. 5 and Fig. 6. As well as being capable of plotting 40 variables and results in different conditions, the software core can give exact details and timing of each gearshifting performed (see Fig. 6). The results of the Fortran program was checked for verification against those of the Simulink which had already been validated in [14].

The comparison of the simulation environments was done in two run modes. The first run included simulation of the vehicle when the system does not alter its continuity i.e., there will be no clutch actuations or gearbox handling. The simulation starts from a specific speed and continues in the same gear until a certain vehicle speed is reached.

The second mode includes a complete drag run. The vehicle starts from a low speed and accelerates toward a high speed. Whenever the engine crosses a specific RPM, an upshift will be triggered and the system will go through all of the procedures and phase changes that will ensue. The simulation will be completely integrated i.e., the subsystems will run with full interaction with each other and unlike most of the other simulations carried out in this area where gearshifting becomes out of scope for whole test runs, it will not be apart from the main run. The concept of this simulation and the involved phase changes have been depicted in Fig. 7.

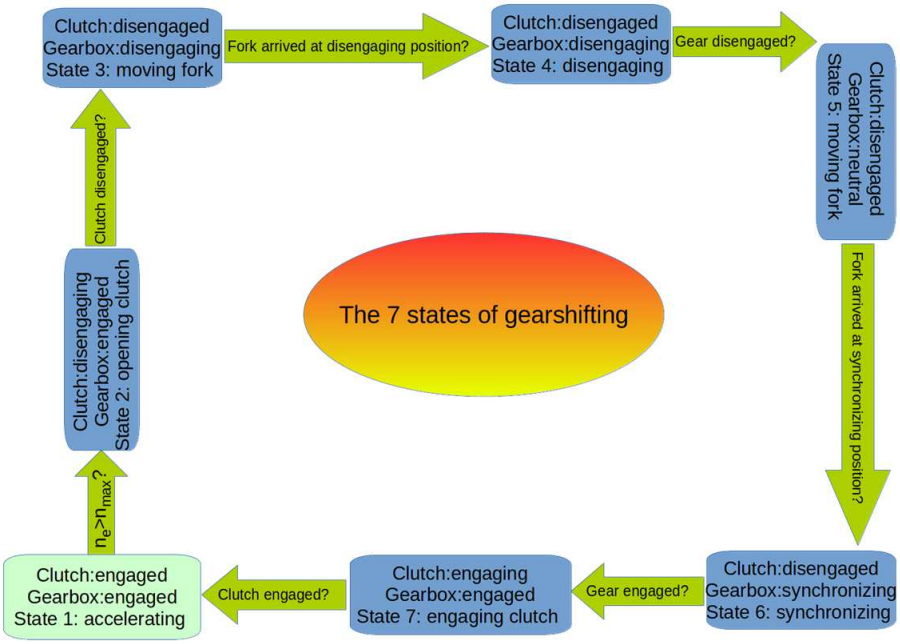

Fig. 7. The gearshifting procedure and the involved state changes.

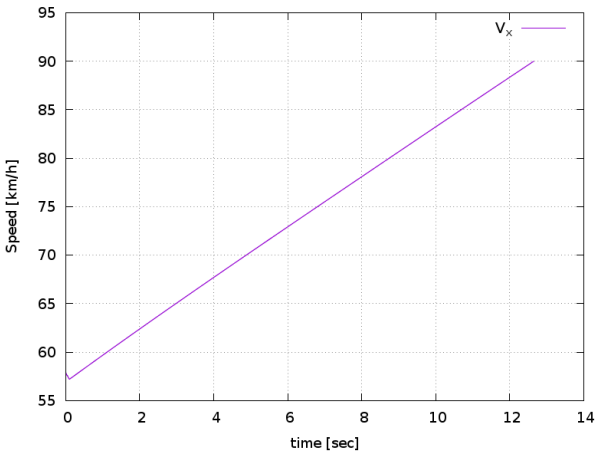

Fig. 8. The simulated vehicle speed for the first run mode.

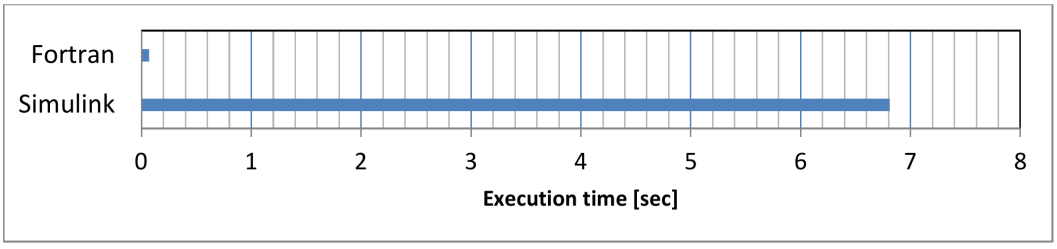

Fig. 9. The run time performance of the first test mode.

4. Results and Discussion

4.1 Execution Time

Simulink Profile Report was used to correctly measure the time spent running and the Fortran program utilized a special subroutine to measure the real time spent for the simulation. To achieve better accuracy in estimation of simulation time, the tests were performed 5 times in each environment and the average time was then calculated. It should be noted that because the model was parameterized in both programs, there was no need to compile the Simulink or the Fortran code for every run; thus, the compilation time has been omitted from the time calculations.

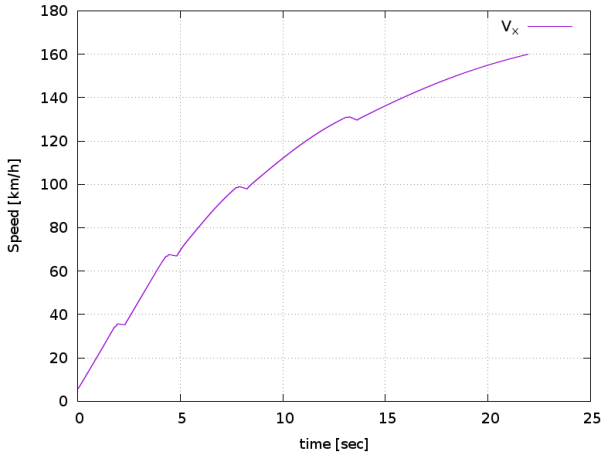

Fig. 10. The simulated vehicle speed for the second run.

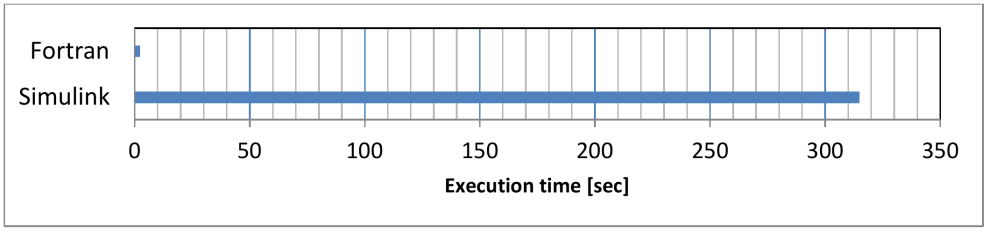

Fig. 11. The execution times for the second run mode.

First, the comparison was done for the first simulation mode. In this mode, the car started with the 4th gear with an engine speed set to 2500 rpm and by setting the throttle to 60 degrees, car accelerated to reach 90 km/h on a road with a grade of 10 degrees. Figure 8 shows the simulated vehicle speed for this run. In this simulation, the Fortran program produced around 14000 nodes for the solution of each state variable.

Figure 9 illustrates the time spent in Simulink and in Fortran program for the first test mode. The Fortran program finished the simulation in just 0.07 seconds while it took Simulink 6.81 seconds to complete the simulation. It shows that both of the simulation programs had little difficulty to finish the run in short time, however, relatively speaking, the Fortran program was very much more efficient and finished much sooner.

For the second run, Fig. 10 shows the simulated vehicle speed where the drag test was performed on a flat road. While the engine speed was at 900 rpm and the first gear was engaged, the acceleration commenced by opening the throttle valve to 75 degrees and the car kept moving until it reached 160 km/h when the simulation ended. Whenever the engine speed crossed 5500 rpm, an upshift was initiated which underwent the procedures of releasing the accelerator pedal, signaling the clutch to open, commanding the gearbox to disengage from the current gear and synchronize to the next gear and, finally, engage the clutch as depicted in Fig. 7. As it can be seen in Fig. 10, 4 gearshiftings occurred in this simulation. The Fortran program, Powertrain Simulator, produced a grid of solution with around 450000 points for each variable in this simulation.

The execution times of the programs for the second run have been presented in Fig. 11. The Simulink program finished the run in 315 seconds while it took Powertrain Simulator only 2.36 seconds to finish the whole simulation. Comparing the run times between Simulink and Fortran, the big advantage of using a compiled language becomes crystal clear.

The speed increase of more than 133-fold in switching from Simulink to Fortran for the second run is significant and this speed increase results from several factors. The first reason is the way Simulink implements the conditions through Stateflow. In the Fortran program, the phase changes are modeled in simple If-Else blocks while in Simulink they are implemented via Stateflow machines where the introduction of Stateflow charts has caused some overhead in the final code execution. Another factor is that the model in the second run includes differential algebraic equations and algebraic loops which make the solution more intensive by mandating the programmer to use memory blocks in specific subsystems of the Simulink model. The Fortran program does this automatically through the default “Do loop,” thereby providing a smoother solution flow. Moreover, inclusion of more MATLAB functions in the second run led to lowering the Simulink performance as the numerous inevitable external calls to MATLAB functions disrupted the computational flow and slowed down the Simulink program.

4.2 Work Hour Performance

In order to consider the cost of developing the programs, the authors recorded the total number of hours needed to build and develop each of the simulation programs. For Simulink, the time spent in hours to develop the model from scratch including writing the needed MATLAB functions, drawing Stateflow charts and layout designing with any debugging done was recorded. For the Fortran program, the same procedure was followed to identify the time needed to code the main program and its functions and subroutines plus the ODE solver and any effort made to debug the program. It should be mentioned that the work hours only include the core software and the time for developing the GUI has been excluded from the calculations. The core software includes all the capabilities for solving, extracting data and plotting. Furthermore, it already provides an interactive text mode interface by which it reads and writes the inputs and outputs and creates plots. In fact, everything that the GUI can achieve was already implemented in the core program and as an add-on module it only serves as an ornamental layer.

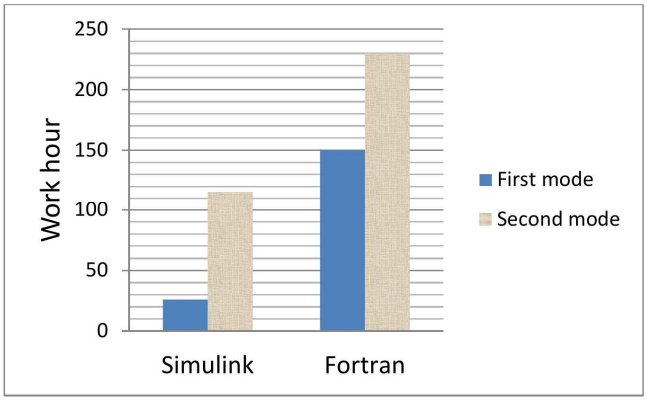

Fig. 12. The work hours spent developing the programs for the first and the second tests.

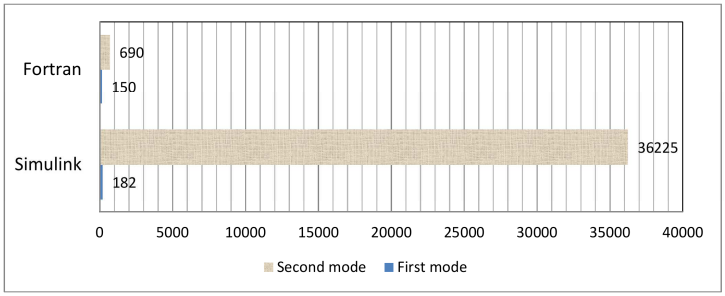

Fig. 13. The work hour performance values in [hour.sec] for the run modes.

To measure the actual productivity of each program, the concept of work hour performance has been introduced here. The work hour performance for the development of a program is defined by multiplying the work hours by the execution time in seconds. Therefore, the productivity of a program can be estimated by how many work-hour seconds it takes to complete. 1 work-hour second corresponds to developing a program within one hour which takes 1 second to run for one time of execution. In this regard, the execution times have been rounded down to the nearest integer more than or equal to the real time. Following this approach is because the human's perception of 1 millisecond and 1 second in execution time does not differ vastly, so the base unit of time should not be lower than a second. Therefore, in this way, the productivity of each program for each run mode was obtained. It should be noted that the more efficient the program is, the lower the value of work hour seconds would be which indicates that the productivity of the program would be higher.

Figure 12 shows the estimated time spent in creating the simulation programs in hours for the two run modes. For the first run, the Fortran program took the authors roughly 150 hours while the Simulink one only took 26 hours. For the second run mode there was a 4-fold increase in the time for completing the Simulink program which went up from 26 hours to 115 hours. For the Fortran code, completing the program for the second run mode took the authors almost 230 hours.

Figure 13 shows the work hour performance of developing the programs. For the first run, the work hour performance for the Fortran program is 150 work-hour seconds while that of the Simulink becomes 182. It should be noted that because of rounding down the execution times for both of the programs, a value of 1 second was considered for the Fortran program and the run time of the Simulink program was changed to 7 seconds.

In the second test mode, following the rule for calculating the work hour performance, the value for the Fortran program is 690 work-hour seconds, while that of the Simulink program turns out to be much higher at 36225 work-hour seconds.

From the results of the first test mode it can be concluded that for designing a system which does not involve phase changes and complicated layouts, the productivity values of both of Simulink and the compiled language are comparable, with the Fortran one being relatively higher. In the Fortran program, its run-time performance has been overshadowed by its development costs as one has to create many codes from scratch for the basics and the initial amount of coding does not depend on the complexity of the model. On the other hand, in the Simulink model rapid development of the first test model has compensated for its lower performance.

For the second run mode, the productivity of the Fortran program is much higher than the Simulink counterpart. It was predicted that due to much higher performance of the Fortran code, its work hour performance value would be lower i.e., it would achieve higher productivity. However, the increased work hours spent on the Simulink program for the second mode has contributed to widening the gap, too.

There are factors that have caused completing the Simulink program for the second run to take longer. The main reason is the increased complexity of the program layout which led to spending more time sketching the design as there were numerous subsystems with initial conditions acquired from the others and more enabled and triggered subsystems to maintain. It should be noted that the increased time spent on developing the Fortran program was mainly because of debugging it, and amending the code for the second run itself took less time than Simulink. Moreover, Simulink stipulates a new subsystem for every phase and the conflict between different subsystems must be avoided by enabling the desired one and disabling the others. In contrast, Fortran can solve the problem while the number of equations is fixed to the maximum number of stat-space variables and whenever there is a phase change, the variables can be merged and the set of ODEs continues to be solved. Another reason is introducing conditional branch execution through Stateflow diagrams. A simple if-else-if block in Fortran is translated to a Stateflow diagram, the layout and the trigger signals of which have to be sketched and taken care of. Therefore, the final complexity of the code in Simulink has become higher which led to spending more time on the modeling.

The considerably slower run-time execution of the Simulink program in the second mode argues its unsuitability to be used in further simulations or optimization. If any optimization that involves repetition of the whole program (e.g., implementing Genetic Algorithm) is intended, developing the model using a compiled language will be the only logical approach.

5. Conclusion

Simulink is undoubtedly a great tool for simulating the behavior of dynamical systems especially if the model can exploit its component libraries. Moreover, it can be used for developing models from scratch where the existing modules and components cannot be utilized due to increased complexity of the model and inclusion of more details. In this regard, one should consider the performance of the final program and the work hours spent developing the model. For simulating systems that do not include phase changes and do not need complicated layouts for sketching, Simulink offers good run-time speed and convenience that is unmatched. The time spent modeling these types of systems is less than that of developing the same model in a compiled language. Furthermore, the Simulink program offers an acceptable work hour performance value comparable to that of the compiled language. Conversely, as more details are incorporated into the model, not only the time spent on developing the model itself is nonlinearly increased, but also the final program would suffer from performance slowdowns. This is when a compiled language, especially Fortran as the front-runner in scientific computing, can vastly improve the performance. When it comes to simulating real-world models which are highly detailed and include phase changes in their states, Fortran is able to increase the performance to an unprecedented level. The time spent modeling the system in the compiled language does not considerably differ from the time spent developing a model in Simulink and, furthermore, the final work hour performance value of the Fortran program becomes much lower than the Simulink program which argues the superior productivity of a compiled language for highly nonlinear dynamical systems. Thus, it is reasonable that a compiled language should be considered for modeling and simulating these system types. Especially, if the program has to be repeated for further investigation e.g., designing an optimized controller, a compiled language should be the primary modeling environment as the Simulink program would lead to diminishing results.

Author Contributions

A.H. Pasdar planned the scheme, initiated the project, did the mathematical modeling and developed all of the programs; S. Azadi reviewed the modeling and validation of the approach; R. Kazemi provided guidance on how the comparison should be done. All authors discussed the results, reviewed and approved the final version of the manuscript.

Acknowledgments

Not applicable.

Conflict of Interest

The authors declared no potential conflicts of interest with respect to the research, authorship and publication of this article.

Funding

The authors received no financial support for the research, authorship and publication of this article.

Data Availability Statements

The datasets generated and/or analyzed during the current study are available from the corresponding author on reasonable request.

References

[1] Giroux R., Landry R.J., Leach B., Gourdeau R. Validation and Performance Evaluation of a Simulink Inertial Navigation System Simulator, *Canadian Aeronautics and Space Journal*, 49(4), 2003, 149-61.
[2] Hödelmoser S., Kitzler F. Comparison of Matlab, Simulink and Anylogic Approach to Argesim Benchmark C9 'Fuzzy Control of a Two Tank System', *Simulation Notes Europe*, 23(3-4), 2013, 195-200.
[3] Lamberský V., Vejlupek J. Benchmarking the Performance of a Dspic Controller Programed with Automatically Generated Code, *Technical Computing Prague*, 2011.
[4] Venayagamoorthy G.K., Comparison of Power System Simulation Studies on Different Platforms-Rscad, Pscad/Emtdc, and Simulink Simpowersystems, *International Conference on Power System Operations and Planning (ICPSOP)*, 2005.
[5] Roscoe A., Blair S., Burt G.M., Benchmarking and Optimisation of Simulink Code Using Real-Time Workshop and Embedded Coder for Inverter and Microgrid Control Applications, *2009 44th International Universities Power Engineering Conference (UPEC)*: IEEE, 2009.
[6] Cansalar C.A., Maviş E., Kasnakoğlu C., Simulation Time Analysis of Matlab/Simulink and Labview for Control Applications, *2015 IEEE International Conference on Industrial Technology (ICIT)*: IEEE, 2015.
[7] Zhang H., Saporta B., Dufoura F., Deleuzed G. Dynamic Reliability by Using Simulink and Stateflow, *Chemical Engineering Transactions*, 33, 2013, 529-534.
[8] Wang K., Steyn-Ross M.L., Steyn-Ross D.A., Wilson M.T., Sleigh J.W., Shiraishi Y. Simulations of Pattern Dynamics for Reaction-Diffusion Systems Via Simulink, *BMC Systems Biology*, 8(1), 2014, 45.
[9] Ravi K.S., Potdar S., Poojar P., Reddy A.K., Kroboth S., Nielsen J.-F., et al. Pulseq-Graphical Programming Interface: Open Source Visual Environment for Prototyping Pulse Sequences and Integrated Magnetic Resonance Imaging Algorithm Development, *Magnetic Resonance Imaging*, 52, 2018, 9-15.
[10] Azzini I., Muresano R., Ratto M. Dragonfly: A Multi-Platform Parallel Toolbox for Matlab/Octave, *Computer Languages, Systems & Structures*, 52, 2018, 21-42.
[11] Pham L.T., Oksum E., Gómez-Ortiz D., Do T.D. Magb_Inv: A High Performance Matlab Program for Estimating the Magnetic Basement Relief by Inverting Magnetic Anomalies, *Computers & Geosciences*, 134, 2020, 104347.
[12] Barbot B., Bérard B., Duplouy Y., Haddad S., Integrating Simulink Models into the Model Checker Cosmos, *Application and Theory of Petri Nets and Concurrency*, Cham: Springer International Publishing, 2018.
[13] Casoria S., Mahseredjian J., Roussel R., Beaudry J., Sybille G. A Portable and Unified Approach to Control System Simulation, *Pulse*, 4(F5Y), 2001, F6Y.
[14] Pasdar A.H., Azadi S., Kazemi R. High-Precision, Real-World Modeling of a Semi-Automatic Powertrain, *Simulation*, 90(9), 2014, 1041-58.
[15] Popinchalk S. Improving Simulation Performance in Simulink, *The MathWorks, Inc*, 2012.

[16] Board M.A.A. Control Algorithm Modeling Guidelines Using Matlab, Simulink and Stateflow, Version 2.0, *The MathWorks Automotive Advisory Board August* 31, 2012.
[17] Lipovetsky S. Numerical Recipes: The Art of Scientific Computing, *Technometrics*, 51(4), 2009, 481.
[18] Phillips L. *Gnuplot Cookbook*, Packt Publishing Ltd, 2012.
[19] Magnin V., Tappin J., Hunger J., De Lisle J. Gtk-Fortran: A Gtk+ Binding to Build Graphical User Interfaces in Fortran, *Open Journals*, 4(34), 2019, 1109.

ORCID iD

Amir Hossein Pasdar https://orcid.org/0000-0002-2472-149X
Shahram Azadi https://orcid.org/0000-0002-5817-9838
Reza Kazemi https://orcid.org/0000-0003-1441-4602

How to cite this article: Pasdar A.H., Azadi S., Kazemi R. A Comparative Study on the Efficiency of Compiled Languages and MATLAB/Simulink for Simulation of Highly Nonlinear Automotive Systems, *J. Appl. Comput. Mech.*, 8(3), 2022, 853–864. https://doi.org/10.22055/JACM.2020.33053.2137