I. Troch, F. Breitenecker, eds.   ISBN 978-3-901608-35-3

DESIGN AND PERFORMANCE SIMULATION OF THE MATHEMATICAL MODELS IN CLUSTER SYSTEMS– *FINAL PAPER* FOR SHORT PAPER CONTRIBUTIONS

P. Kvasnica[1], I. Kvasnica[2],
[1] Kvasnica Alexander Dubček University of Trenčín; [2] Department of Environment Trenčín, Slovak Republic

Corresponding Author P. Kvasnica, Alexander Dubček University of Trenčín, Inst. of Informatic, Mechatronic Faculty, Študentská 2, 911 50 Trenčín Slovak Republic; `kvasnica@tnuni.sk`

Abstract. Simulation must be used to study dynamic systems. There are many methods of modeling systems which do not involve simulation but which involve the solution of a closed-form system (system of linear equations). The power of simulation is that – even for easily solvable linear systems – a uniform model execution technique can be used to solve a large variety of systems. Another important aspect of the simulation technique is that one builds a simulation model to replicate the actual system. When one uses the closed-form approach, the model is sometimes twisted to suit the closed-form nature of the solution method rather than to accurately represent the physical system. A harmonious compromise is to tackle system modeling with a hybrid approach using both closed-form methods and simulation. For example, computational models of this type and resolution are currently employed for configuration analysis, aerodynamic interference analysis and aerodynamic data generation. This evolutionary procedure is often very effective.

1 Preface

Simulation is often essential in the following cases: 1) the model is very complex with many variables and interacting components; 2) the underlying variables relationships are nonlinear; 3) the model contains random variates; 4) the model output is to be visual as in a 3D computer animation [1].

The necessity of high fidelity modeling in the design process is often conflicting with the requirement of short turn around time. A balance between modeling requirement and turn around time limitation needs to be established at every phase in the design cycle. A way to reduce the turn around time is to use powerful parallel computer systems. Traditionally, supercomputer resources have been equivalent with large cost and therefore not widespread in industry. This started to change in the late 1990s when PC–cluster with Linux, so called Beowulf system. When designing a PC–cluster for a specific application several design choices have to be made. The selection of computer node configuration and interconnecting network are major factors influencing the parallel performance [5].

The design task is to achieve an optimal integration of all components into an efficient, robust and reliable simulator with high performance that can be manufactured with low technical and economical risk at an affordable cost over the whole lifetime of the product. The product design process is in general divided into three phases which tend to overlap in a staggered fashion.

In the **conceptual design** phase the simulator is defined at a system level. Many variants are studied, and the design selected is the one that bests fulfils the specifications of the market or a customer.

In the **preliminary design** phase the tentatively selected concept is refined until feasibility is established, i.e. extensive array of design sensitivities are generated, design margins, etc.

In the **detailed design** phase in which details of the product are elaborated, optimizations are made and data sets are generated.

2 Computer simulation

Computer simulation is the discipline of designing a model of an actual or theoretical physical system, executing the model on a digital computer, and analyzing the execution output. The use of simulation is an activity that is as natural as a child who *role plays*. To understand reality and all of its complexity, we must build artificial objects and dynamically act out roles with them. Computer simulation is the electronic equivalent of this type of role playing and it serves to drive synthetic environments and virtual worlds. Within the overall task of simulation, there are three primary sub-fields: model design, model execution and model analysis, in Fig. 1.

Models can take many forms including declarative, functional, constraint, spatial or multimodel. The next task, once a model has been developed, is to execute the model on a computer. You need to create a computer program which steps through time while updating the state and event variables in your mathematical model. You can, for instance, *leap* through time using *event scheduling* or you can employ small time increments using *time slicing*. You can also execute (i.e., simulate) the program on a massively parallel computer. This is called *parallel and distributed simulation* [1]. For many large-scale models, this is the only feasible way of getting answers back in a reasonable amount of time.

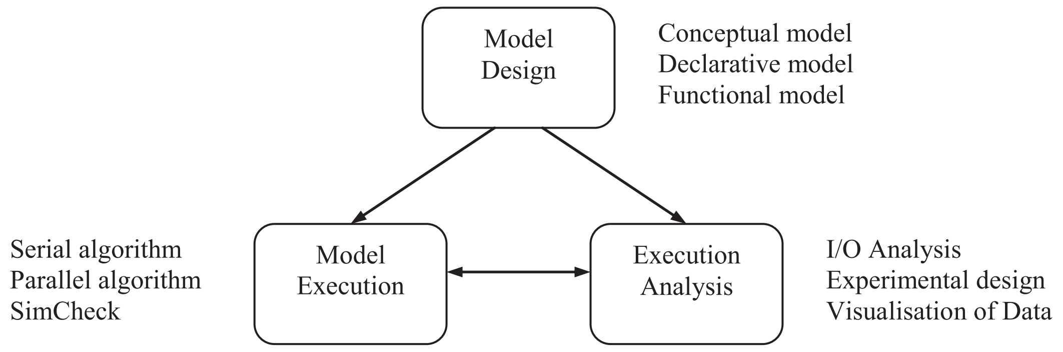

Figure 1: Three Sub-Fields of Computer Simulation.

Simulation of a system can be done at many different levels of fidelity. The physics-based models and output, another may think of more abstract models which yield higher-level. Models are designed to provide answers at a given abstraction level – the more detailed the model, the more detailed the output.

3 Performance and validity of simulation

Complexity is a measure of the resources required to analyze, simulate, and develop the model. Typically, the performance of a simulator will increase as the complexity of a model decreases. Fox example, the speed of simulator (measured, say, in the number of model transition executions it can do in a given time) will usually increase as either the number of components or the number of states per component diminish. We can similarly talk of the performance of model development or the model exploration process in terms of how it takes to develop the model or how many alternatives can be examined in a given time, respectively.

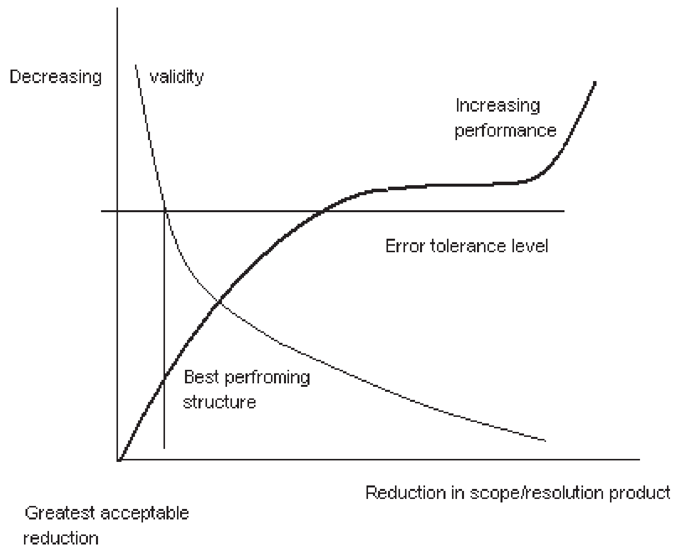

Figure 2: Performance increasing while validity decreases as the resolution product trade-off

axis x means reduction in resolution and axis y means decreasing of resolution.

So on the hand, performance can be expected to improve with reduced complexity. On the other hand, reducing the resolution simulator often entails a loss of validity of the model. Of course, has to be measured within an experimental frame of interest. This performance trade-off is illustrated in Fig. 2, where, we see, conceptually depicted, performance increasing while validity decreases as the resolution product is decreased. In such a situation, we can set a level of error tolerance that determines the level of error we can live with its. The complexity reduction or performance improvement at the tolerance level is the best we can achieve. The purposes for which the model is the being constructed or deployed should determine such threshold limits [6].

However, it may well be that if we require a lower level of resolution in the model output, then the error may be below tolerance, or even vanish, for the same level of detail included in the model. Indeed, the complexity in the model may be much lower at the threshold of acceptability for the low-resolution frame than for its high-resolution counterpart. Correspondingly, the fastest simulation might be much faster in low-resolution frame than in the high-resolution one.

4 Simulation of mathematical model

This concept allows to hide implementation details parts of models (sub-models). It is then possible to easily change the underlying implementation of sub-models without changing the other parts of the model provided that the interface of the sub model remains intact.

The mathematical models increase of angle trajectory of aircraft and angle of climb are defined :

$$\Delta\theta(s) = -W_\theta^{\delta_T}(s)\Delta\delta_T(s) - W_\theta^{\delta_B}(s)\Delta\delta_B(s), \tag{1}$$

$$\Delta\vartheta(s) = -W_\vartheta^{\delta_T}(s)\Delta\delta_T(s) - W_\vartheta^{\delta_B}(s)\Delta\delta_B(s), \tag{2}$$

where

s – Laplace operator in differential equations, $W_\theta^{\delta T}(s)$ – transfer function increment angle of trajectory of aircraft per fuel supply, $W_\theta^{\delta B}(s)$ – transfer function increment angle of trajectory of aircraft per elevator, $W_\upsilon^{\delta T}(s)$ – transfer function increment angle attack of aircraft per fuel supply, $W_\theta^{\delta B}(s)$ – transfer function increment angle attack of aircraft per elevator in system equations [2]. Transfer function Δ(s) of system is next :

$$\Delta(s) = \begin{pmatrix} s + a_x^V & a_x^\alpha & a_x^\theta & 0 \\ a_y^V & a_y^\alpha & \left(s + a_y^\theta\right) & 0 \\ a_{mz}^V & \left(a_{mz}^{\dot\alpha} s + a_{mz}^\alpha\right) & 0 & \left(s^2 - a_{mz}^{\omega z} s\right) \\ 0 & 1 & 1 & -1 \end{pmatrix} = 0. \tag{3}$$

Between mathematical model in equation (1) and last system valid

$$W_\theta^{\delta_T}(s) = -a_\theta^{\delta T} \frac{\Delta_{13}(s)}{\Delta(s)}, \tag{4}$$

where

s – Laplace operator in differential equations, $a_\theta^{\delta T}$ – aerodynamic derivation, Δ(s) – determinant of a transfer function, $\Delta_{13}(s)$ – minor of the determinant a given element a_{13} (1-th row and 3-th column) in system equations.

Between mathematical model in equation (2) and last system valid

$$W_\theta^{\delta_B}(s) = -a_y^{\delta B} \frac{\Delta_{23}(s)}{\Delta(s)} - a_{mz}^{\delta B} \frac{\Delta_{33}(s)}{\Delta(s)}, \tag{5}$$

where

s – Laplace operator in differential equations, $a_y^{\delta B}$, $a_{mz}^{\delta B}$ – aerodynamic derivations, Δ(s) – determinant of a transfer function, $\Delta_{23}(s)$ – minor of the determinant a given element a_{23} (2-th row and 3-th column), $\Delta_{33}(s)$ – minor of the determinant a given element a_{33} (3-th row and 3-th column), in system equations (3).

The most used parallel systems are based on either shared memory, distributed memory or hybrid distributed-shared memory systems. The authors published their experiences with distributed architecture (see [3]) where divide the modeling problem among many computing nodes. Due to the fact that there is no global memory, it is necessary to move data from one local memory to another by means of message passing.

The message passing system is replaced by OpenMP standard that supports multi-platform shared-memory parallel programming in C/C++. OpenMP parallelism uses the fork-join programming model, where some threads is created by a fork operation and is joined at the end, as shown in Figure 3. Multiple threads within the same process can be assigned to different processor or core. Threads running simultaneously on multiple processors or cores may work concurrently to execute a parallel program. If multiple threads collaborate to execute a program, they will share the resources, including the address space of the corresponding process.

Programmatically, OpenMP is based on compiler directives imbedded in source code which make a simple way to maintain a single code for the sequential and parallel version of the application due to the existence of conditional compilation.

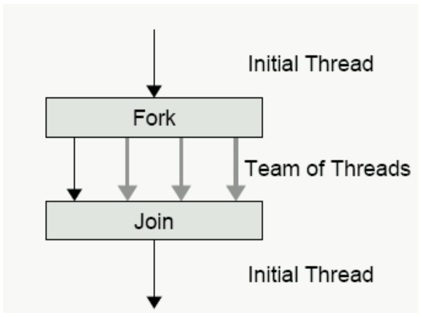

Figure 3: OpenMP: Fork-Join programming model.

5 Performance and measurement

The performance evaluation is performed with the application code using inviscid flow modeling. Computational models of this type and resolution are currently employed for configuration analysis, aerodynamic interference analysis and aerodynamic data generation.

The parallel performance is analyzed in terms of computational performance and parallel efficiency on four processors cluster 'EvaOne'. The performance is measured on the entire code (excluding I/O), rather than detailed instrumentation of selected code sections. This is done by collecting the number of floating point operations through hardware counters and measuring the wall clock time for the parallel runs.

For the type of algorithm used in this study, the load balancing and the communication overhead items are the most crucial to deal with to obtain good parallel performance. In this example the mean transfer time for messages is affected by latency already at two Intel Core Extreme QX6800 2.93GHz, 1066HHz, 8MB L2 quad core processors. Comparing the performance on clusters with different processors clearly verifies this. Clusters with low latency networks deliver good parallel efficiency also for a large number of processors.

In comparison, clusters with Gigabit Ethernet show a decrease in parallel efficiency already after 2 processors. With 2 processors on Intel© Core2 Quad QX6800 a total performance of 187,5 GFlops is achieved and using 2 processors on Intel© Core2 Quad QX6700 a performance of 170,2 GFlops is reached [4]. In comparison the theoretical processor speed only differs by 10 % and the remaining difference is mainly due to larger 2nd level cache on the Intel Core Extreme QX6800 system. Using a system that delivers more than 170 GFlops in application performance clearly affects the turn around time.

6 Conclusion

When the problem is parallelized over more processors two parts will influence the performance results more than the other. Firstly the computation to communication ratio will decrease as the partitioning introduces new internal boundaries between domains. Both the total amount of data communicated as well as the number of messages increase. The communication pattern becomes more fragmented and the mean message size decreases. Secondly, when more processors are added the total amount of fast cache memory also increases. This means that a larger part of the total problem will reside in the cache with a subsequent performance gain. This is called cache effect and can result in a super linear speedup, i.e. higher speedup numbers than number of processors.

In conclusion, using this approach a manufacturer of industrial automated systems does not need the plant (physical part of the machine). The design and simulation network mathematic model was make on cluster system. In consequence, this approach allows to reduce the times of flight simulator.

7 References

[1] Fishwick, P.: *What is simulation? Why do simulation?*. [online]. 19. October 1995. [cit. 2008-09-09]. On internet: <http://www.cis.ufl.edu/~fishwick/introsim/paper.html>.

[2] Krasovskij, A., A.: *Sistemy avtomaticeskogo upravlenja poletom i ich analiticeskoje konstruirovanie*. Nauka, Moskva 1980, pp. 589.

[3] Kvasnica, P., Kvasnica, I.: *Simulation of Flying Parallel Computing in Computer*. In: International Review of Aerospace Engineering (IREASE), 2008, in printing.

[4] Processors – Intel @ microprocessor export compliance metrics [online]. 19. November 2008. [cit. 2008-12-11]. On internet: <http://www.intel/com/support/processors/sb/cs-023143.htm>.

[5] Sillén, M.: *Application of Parallel Computers to Enhance the Flow Modelling Capability in Aircraft Design*. In: Department of Mechanical Engineering, Linköpings universitet, Sweden ,2006.

[6] Zeigler, B. P., Praehofer, H, Kim, T., G.: *Theory of Modeling an Simulation*. Second Edition, Academic Press, San Diego, USA, 2000.