MODELING, IDENTIFICATION AND CONTROL, 1986, VOL. 6, NO. 4, 249–255
doi:10.4173/mic.1985.4.5

Perspectives in simulation hardware and software architecture†

W. O. GRIERSON‡

Keywords: *ADSIM, SYSTEM 100, SYSTEM 10, hybrid computers, time-critical simulation, real-time simulation.*

Historically, analog and hybrid computer systems have provided effective real-time solutions for the simulation of large dynamic systems. In the mid 1970s, ADI concluded that these systems were no longer adequate to meet the demands of larger, more complex models and the demand for greater simulation accuracy. The decision was to design an all-digital system to satisfy these growing requirements (see Gilbert and Howe, (1978)). This all-digital approach was called the SYSTEM 10. The SYSTEM 10 has been effective in solving time-critical simulation problems and in replacing the previous approach of utilizing hybrid computers. Recent advances in 100 K emitter coupled logic (ECL) now make it possible to support a new generation of equipment that expands modeling capabilities to serve simulation needs. The hardware and software concepts of this system, called the SYSTEM 100, are the subject of this paper.

The author received a Bachelors' degree in Physics from the University of Michigan and a Masters' degree in Operations Research from Wayne State University. During 20 years in applied simulation he directed large real-time simulation studies and was responsible for the purchase of simulation equipment. He is currently responsible for the international operations of Applied Dynamics International.

1. Introduction

The task of solving time-critical simulation problems is generally associated with ordinary differential equations whose right-hand-side components include many scalar computations. Because of the nature of these computations, they can not be 'vectorized' and accommodated effectively within a general-purpose digital computer. The solution time on general-purpose mainframes is typically much slower than real time on all but the simplest problems. The problem is further complicated when constraints imposed by hardware-in-the-loop timing requirements are added to the simulation scenario. An added dimension is the engineer's need to 'interact' with the model and experiment with different concepts in a timely manner. Further, the engineer needs to document his work so that he may communicate the results more effectively (see Grierson *et al.* (1980)). The SYSTEM 100 provides an effective approach to satisfying these requirements.

2. The SYSTEM 100 design concepts

The details of hardware architecture provide some indication of potential problem execution speed. However, the true performance of any equipment can only

Received 15 June 1985.

† This paper was presented at the International Seminar on Modern Methods in Dynamic Simulation of Industrial Processes, Trondheim, Norway, May 1985.

‡ Applied Dynamics International, 3800 Stone School Rd., Ann Arbor, Michigan 48104, U.S.A.

be measured by executing the type of problem for which it was designed. Generally, the resulting throughput speed is a measure of the software effectiveness in utilizing the available hardware resources. When ADI began the SYSTEM 100 project, they established parallel efforts for hardware and software design. The resulting system is an optimized trade-off among many factors. Before reviewing these results, it is important to list some of the considerations:

Design objectives

- Expansion of current modeling capabilities
- Greater than 32-bit floating-point arithmetic
- Provision for future expansions
- One model for design and hardware testing
- High efficiency for long vector and scalar computations
- Parallel hardware and software design

Design constraints

- Costs competitive with market traditions
- Combatibility with industrial requirements
- Responsiveness to 'Host' computer changes
- 'True' high-level simulation software
- Support for program documentation
- Support for hardware-in-the-loop (HIL) testing
- Support for an 'experimental' operations environment
- Real-time code execution
- Realistic hardware/software servicing.

2.1. The SYSTEM 100 hardware architecture

The SYSTEM 100 uses a multi-processor architecture as shown in the figure above. There are currently five processors in the system that have been uniquely designed to support the computations required by real-time dynamic simulation. An ALU and MUL are the arithmetic and multiplier processors, respectively, in the system. The STO is the storage processor that supports variable data and function table data storage for the model. The COM is the communications processor. This processor stores the application program and provides data communication via a dual ported memory to 'other' digital systems. The COM processor is the primary device for controlling the operations of the other processors. The supervisor processor (SUP) supports the downloading of ADSIM instructions from the host computer prior to run time. The SYSTEM 100 uses a host computer for program preparation, program debugging, program storage, running diagnostics, etc. While the host is required for all off-line or non-runtime tasks, it is not generally required during run-time to support the computations. Each of these processors may operate independently and have their own program memory, program counter, and instruction decoder. The processors are interconnected by a bus system called the PLUSBUS (see Fadden, (1983)).

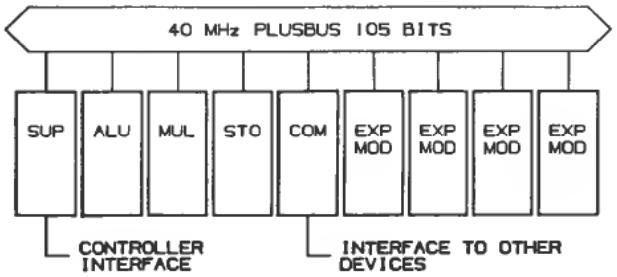

Figure 1

Fundamental properties of the SYSTEM 100 architecture include

80 MHz clock

40 MHz PLUSBUS, 105 bits wide, (65 bits data, 40 bits address and control)

64-bit instruction word length in each processor

65-bit floating-point add each 100 nanoseconds (1 sign bit, 12 exponent bits, and 52 significand bits)

53-bit floating-point multiply each 100 nanoseconds (1 sign bit, 12 exponent bits, and 40 significand bits)

32K words of COM processor memory for user programs

16K words of STO processor memory for user data

External word format matching capability for 16, 32, 48, and 64-bit fixed and floating-point representations.

The parallel execution of one floating-point add and one floating-point multiply results in a design speed for the SYSTEM 100 of 20 million floating-point operations per second (20 MFLOPS). Figure 1 also shows spaces for 4 additional processors on the PLUSBUS. One future consideration is to add an additional ALU or MUL processor. This would allow maximum resource utilization when processing algorithms where two adds and one multiply, or, one add and two multiplies are required, respectively. The addition of an ALU or MUL would increase the design speed to 30 MFLOPs. Other considerations for future expansions include a large data memory processor and an input-output processor.

2.2. The SYSTEM 100 software architecture

The software for the SYSTEM 100 is called ADSIM, an acronym for ADvanced SIMulation language. ADSIM has been designed to support the computational requirements of simulation as well as manage the resources of the hardware effectively. To illustrate this effectiveness, consider the computation for sine (x). The execution time for sine (x) is 1·1 microseconds. The computation uses a 53-bit format with an error of less than one-half of one LSB (for the 40-bit significand). Within the SYSTEM 100, the sine (x) function requires one instruction in COM processor. This is the extent of the user code requirement for this computation. The algorithm to support this computation is called a 'kernel' and is distributed across the ALU, MUL, and STO processors. A kernel is the fundamental code, written by ADI, that supports the computational requirements of the user code (currently there are approximately 1000 kernels written for ADSIM). By distributing the kernels

across the processors, ADI is able to 'phase' the operations for these computations. This means that while one kernel is finishing its processing, the COM processor may begin to work on the next kernel as required by the user code. In general, it can be expected that upwards of 50 per cent of the design speed of the SYSTEM 100 may be realized in actual simulation problems. An accurate estimate of problem execution speed is:

.7*(Number integrations) + 1.*(Number math functions) + ···

0.1*(number adds & multiplies) + 12.0 overhead = ···

Approximate frame time in microseconds

As mentioned above, the hardware and software designs of the SYSTEM 100 have been conducted in parallel and some trade-offs have been made. The sine function is an example. Initially, the hardware design did not accommodate the determination of the quadrant in which the computation was made. A change in hardware saved five instructions (500 nanoseconds) that had previously been reserved for determining the quadrant location. This 'compromise' not only saved five instructions but also allowed 'other' algorithms to be started in a more effective manner.

3. ADSIM examples

The following examples illustrate simulation programming in ADSIM, the high-level language for the SYSTEM 100 computer (see Fadden, (1984)).

Example 1 is shown in Fig. 2. These statements are the total requirements in ADSIM to represent van der Pol's equation. The lines beginning with '!' are comments and are ignored by the ADSIM compiler. The 'DYNAMIC continuous' statement begins that portion of the code representing state variable computation. The ''' symbol denotes the differential operator d( )/d*t*. Differential equations are entered in first order form. Since van der Pol's equation has two state variables, it

```
Title   van der Pol's Equation, a simple example in ADSIM
!
!  The model   Xddt + MU(1 − X * X)Xdt + X = 0
!
DYNAMIC continuous
            X' =   Y
            Y' = − MU * (1 − X * X) * Y − X
END DYNAMIC
!
!  Coefficient and initial condition values
!  and run specifications.
!
DATA          MU = 1, X@ = 1, Y@ = 0
RUNSPECS      endtime = 15, speedup = 1000
!
!  Finish with execution control for continuous simulation.
!
EXECUTE continuous
```

Figure 2. Example 1.

requires two lines of code to represent the model. (Note: because ADSIM is non-procedural, the equations may be entered in any order. The ADSIM compiler will automatically sort the equations to ensure proper computational order. A 'NOSORT' option is also available.) The 'END DYNAMIC' statement indicates to the compiler the completion of state variable computations for that portion of the code. Repeated declarations of dynamic blocks are permissible to define the model completely. Data for the model is entered as shown. The '@' symbol designates initial conditions. The RUNSPECS command specifies the run duration to be 15 seconds and requests the simulation to be executed at 1000 times faster than real time.

Example 2 is shown in Fig. 3. In this example, the 'method' command is used within the dynamic block to specify the integration method to be employed. The RK4 designates a Runge-Kutta, fourth-order algorithm (the default algorithm is a second-order Adams Bashforth method). The PARAMETERS command specifies the values that may be changed at run-time. The parameters K, L, and M are shown grouped under the name '#PA'. Similarly, the initial conditions are grouped by the name '#IC'. These groups enable the engineer to display related groups of data rather than reviewing the entire problem list. The use of brackets, i.e. [...], designate computations that are performed once, prior to the simulation execution. When computations are enclosed in parentheses, i.e. (...), they are performed on each pass through the instructions. The reader will also notice that comments may be included on the same line as the simulation code by using the '!' symbol. This problem includes no specification for problem speed-up relative to real time. When this option is deleted, the problem execution will automatically be set to real time.

Example 3 is shown in Fig. 4. The previous examples demonstrate problem control by the user. The user, with this feature, may interactively command and control all problem execution from the terminal. This example illustrates how the user may control problem execution automatically according to pre-defined criteria.

```
TITLE   Simple Nonlinear Pendulum
!
!  The model   thddt + (K/M) * thdt − (g/L) * sin (th) = 0
!
DYNAMIC continuous
METHOD RK4
              thd' = − [K/M] * thd − [g/L] * sin (th)
               th' =    thd
END DYNAMIC
!
!  Parameters, data, and the endtime for
!  the simulation run.
!
PARAMETERS     g = 32.2                ! Units on g are ft/sec-sec.
PARAMETERS     #PA  K,  L,  M
GROUP          #IC  thd@,  th@
DATA           #IC    0,  1.2
RUNSPECS       endtime = 5.0           ! Time of simulation in seconds
!
EXECUTE continuous
```

Figure 3. Example 2.

```
TITLE:  Solution Set Using Initial and Terminal Regions
!
!
REGION initial
      A = 0          ! Initialize the set of runs.
100   A = A + B      ! Prepare the run.
END REGION
!
!
DYNAMIC continuous
        x' = y              ! Put in the model and the "end-run"
        y' = -Ay - x        ! conditions.
  end_run = (x * x + y * y) .LT. C

END DYNAMIC
!
!
REGION terminal
         IF A .LT. D THEN   ! Is another run required ? If yes,
           GOTO 100         ! "go to" it, otherwise, complete
         ENDIF              ! the set of runs.
END REGION
!
!  Enter data and run specifications.
!  No run is longer than endtime.
!
DATA  B = .05, C = .02, D = 2, x@ = 1
RUNSPECS   speedup = 1000, endtime = 20
!
EXECUTE continuous
```

Figure 4. Example 3.

This is accomplished via the initial and terminal 'REGION' control blocks. Within 'REGION initial', the run is initialized and provision is made for executing the next run. The 'REGION terminal' counts the number of runs. If less than 20 runs have been executed, A will be incremented by B and a new run will be executed. The session is terminated when either 20 runs are executed or the end condition, (x*x + y*y).LT.C, is detected.

4. Problem interaction at run-time

An added requirement of simulation systems is that they support user interaction at run time. This is required because dynamic simulation is generally used as a design tool. This means that the engineer needs the ability at run-time to change parameters, initial conditions, integration step size, integration algorithms, run duration, etc. In general, this requirement means that the engineer must have a convenient means to 'experiment' with the model and be able to change anything except the fundamental equations. This should be accomplished without recompiling the program and as quickly as possible. ADSIM supports these requirements via the interact file. The example in Fig. 5 illustrates some of the options available to the engineer at run time. The variable names and data refer to the non-linear pendulum problem above.

```
...
AD100 > METHODS ALL                          !  display integration methods
TH                    AB2
THD                   AB2
AD100 > METHODS ALL RK4                      !  change them to RK4
AD100 > GO                                   !  run simulation using RK4
...
...
AD100 > DATA #IC                             !  display values in group #IC
TH                    0.
THD                   1.2
AD100 > DATA THD@ 1.0                        !  change THD i.c. to 1.0
AD100 > GO                                   !  run with new initial condition
...
...
AD100 > RUNSPECS STEPTIME                    !  display integration stepsize
STEPTIME          5.000E-5                   !  stepsize is 50 microseconds
AD100 > RUNSPECS STEPTIME .000100            !  change to 100 microseconds
AD100 > GO                                   !  run with new stepsize
...
```

Figure 5. Example 4.

The prompt from the interact file is 'AD100>'. The 'METHODS ALL' command requests all methods in use to be displayed. Currently, ADSIM supports the following methods.: AB1 ... AB4, AM1 ... AM4, RK2, RK4, RTRK2, RTRK4 and NOP. The NOP is a 'no-op' command used primarily during problem check out to open loops and isolate various portions of the model. Each of these commands are executed quickly without recompiling the ADSIM program. The alternative to a prompting scenario operation is to automate the sequencing as illustrated in Example 3 in Fig. 4.

5. Conclusion and summary

The computer hardware and software to service time-critical simulation requirements continue to evolve to serve expanding problem needs. Future systems will continue to make use of new digital technologies as they become available. One of the more challenging areas in computation is to optimize software operations so they may effectively utilize available digital resources.

References

Fadden, E. J. (1983). *The SYSTEM 10 Plus: Advancing Scientific Computation* (Applied Dynamics International, Ann Arbor).

Fadden, E. J. (1984) *The SYSTEM 10 Plus: Broader Horizons* (Applied Dynamics International, Ann Arbor).

Gilbert, E. O. and Howe, R. M. (1978). Design Considerations in a Multiprocessor Computer, *AFIPS Conference Proceedings, Vol. 47* (AFIPS Press, Arlington), pp. 385–393.

Grierson, W. O., Lipski, D. B. and Tiffany, N. O. (1980). Simulation Tools: Where Can We Go?, *Proceedings of the Summer Computer Simulation Conference.* (AFIPS Press, Arlington). pp. 3–6.