APPLICABILITY OF ADA TASKING FOR AVIONICS EXECUTIVES

Roger E. Kontak, Maj, USAF

Electronic Systems Division (AFSC) Airlift and Weather Systems Directorate (ESD/AVMM) 2
Hanscom Air Force Base, MA

Abstract

This study evaluated Ada tasking performance and its suitability for avionics schedulers known as executives by comparing variations of Ada executives written by the author with the existing Digital Avionics Information System written in JOVIAL. The system overhead of each model was evaluated while running a series of representative application tasks. The study found that Ada tasking had considerably more overhead than its JOVIAL counterpart in order to maintain precise cyclical timing. Another outcome was that several Ada compilers were unable to produce code which could be run on the MIL-STD-1750A computer. This points to the present immaturity of Ada compilers targeted toward embedded aircraft computers and adds support for the need to revise standards and develop compilers as necessary to provide an efficient Run Time System for Ada executives.

Introduction

The Ada programming language was developed for the Department of Defense (DoD) in response to the perceived software crisis in the 1970's. A large portion of software costs related to this crisis are incurred for embedded computer systems. "By definition, an embedded computer system is one that forms a part of a larger system whose purpose is not primarily computational, such as a weapons system or a process controller" (Booch, 1983:13). For example, an aircraft embedded computer system may perform functions such as flight control, autopilot operation, weapons delivery applications, and similar routines. Embedded systems have particular programming requirements including:

- Parallel processing
- Real-time control
- Exception handling
- Unique input/output (I/O) control

(Booch, 1983:13)

Ada was designed primarily to reduce embedded system software development costs (ARINC, 1986:1). Although real-time control is only one requirement for embedded systems, these systems are often referred to as real-time systems. A real-time system is one which must respond to externally generated input stimuli within a specified period of time. In other words, it has processing deadlines. "If a real-time system performs the correct function, but delivers the result too late then it has failed to satisfy its requirements" (Auernheimer and Kemmerer, 1986:879).

The scheduler for real-time avionics systems that is responsible for delivering results on time is known as the executive. An executive which handles tasks at specific intervals is known as a cyclical executive. The executive provides an operating system for handling concurrent (parallel) processes which run on the embedded avionics computer. For Air Force embedded avionics systems, this executive has traditionally been written in the JOVIAL programming language. A JOVIAL executive, which usually contains assembly language subroutines, is created for each embedded system. This method, however, leads to program code segments that are not portable and may have software design inefficiencies.

Ada has a special construct, known as tasking, for writing executives. Tasks are entities that execute in parallel. Each task is considered to be executed by a logical processor of its own when run on a single processor system (Booch, 1983:9-1).

These facilities are quite unlike the services provided by a typical run-time executive or operating system. Real-time systems will be designed as a set of cooperating concurrent processes (Ada tasks) using the Ada tasking model (Burger and Nielsen, 1987:49).

Executives written using Ada tasking promote the Ada goals of reliability and maintainability (Booch, 1983:47).

The Problem

Tasking is an important feature of Ada for

1 Ada is a registered trademark of the United States Government (Ada Joint Program Office)

2 This study is condensed from a masters thesis prepared for the Air Force Institute of Technology, Wright-Patterson Air Force Base, OH

U.S. Government work not protected by U.S. copyright.
measurements of overhead for individual Ada. JOVIAL J73/I is the previous standard for Air Force avionics. Unfortunately, little is known about Ada tasking which could be rewritten using Ada tasking. To promote these goals, however, tasking must also operate efficiently without excessive overhead.

Scope

This study was designed to quantify Ada tasking performance in real-time avionics executives. The study was directed toward evaluating executives run on the MIL-STD-1750A computer, a single processor embedded computer system used in Air Force avionics systems. The benchmarks developed for this computer should help programmers determine if Ada tasking can be used in place of JOVIAL in the avionics executive.

Experimental Design

General Approach

The main goal of this study was to investigate the overhead associated with tasking for avionics executives. This goal was pursued by comparing two executives: one using Ada tasking and the other using task scheduling written with JOVIAL J73/I.

The rationale for comparing different executive models was twofold. First, empirical measurements of overhead for individual Ada tasking features alone have already been made in studies by Burger and Nielscn (1987) and Clapp et al. (1986). Second, the suitability of Ada tasking for avionics systems must be evaluated with respect to overall system performance. Since JOVIAL J73/I is the previous standard for Air Force avionics software, comparing an Ada version with a JOVIAL implementation of the same system yields an analysis of how well Ada tasking compares with the previous standard.

The first step of performing an experiment to compare Ada and JOVIAL was to determine requirements for the experimental design. Rather than attempting to formulate these requirements, a search was made for existing JOVIAL executives which could be rewritten using Ada tasking. To be useful, the chosen JOVIAL executive had to be representative of avionics systems. The only executive found to be available for this study was a subset of the DAIS executive. Fortunately, the DAIS is highly representative of avionics executives and variations of the DAIS are actually in use in many avionics applications (King, 1987). The DAIS is an ideal executive to study since it continues the research done by Scarpelli (1980) and also expands work done by the Aeronautical Systems Division (ASD) System Engineering and Avionics Facility (SEAFAC) which used the DAIS for a KC-135 executive (King, 1987).

DAIS is a system architecture which can be configured for various avionic applications and missions using core elements or building blocks. The purpose of the DAIS concept is to reduce the proliferation and nonstandardization of aircraft avionics, and permit the Air Force to assume initiative in the specification of standard avionic systems and interfaces for future Air Force system acquisitions. (DAIS, 1977:5)

To provide flexibility for various avionics configurations, the DAIS is driven by tables which contain lists of all application tasks and specific task requirements. The tables to run the DAIS were built using a scaled down, yet representative, series of application tasks for a typical avionics executive. The representative tasks were provided by SEAFAC based upon their experience with avionics executives. The task specifications included the phase, frequency, and priority required for task execution. A more detailed explanation of these specifications and the detailed design of the experiment is provided in the next section. Since the tests were designed to study the task scheduling overhead of each model, the function of each application task was immaterial to the experimental design. Thus, each application task contained the same body. The task bodies were designed to work within the Ada executive as well as the DAIS. Rather than use null task bodies, the task bodies were designed to prevent task elimination through Ada compiler optimization by using global variables which are referenced both inside and outside the task bodies.

After running the DAIS on a MIL-STD-1750A computer, an Ada tasking executive was designed and written to schedule identical application tasks. This executive was originally debugged, run, and tested on a DEC VAX-11/782. After program errors were eliminated, the actual tests and comparisons of both version were performed on the 1750A computer. Since the 1750A has no operating system to assist with performance measurement, running the executives on the 1750A presented more difficulty in obtaining measurements. The measurements obtained, however, were more accurate because there was no interference from the operating system nor multiple users to affect the experimental outcome.

Measures and Factors

The primary performance concern of an avionics executive is run-time processing efficiency (Dewar, 1987). To measure overall run-time efficiency, the run-time overhead must be measured. Ideally, the run-time overhead should leave enough CPU time available to give application tasks primary access to the processor. To measure run-time overhead,
idle CPU time was recorded and compared for each model. Since each model performed identical application tasks, the difference in idle CPU time indicated the model's relative run-time efficiency. Several tests were subsequently developed to determine whether the Ada tasking version significantly affected task execution.

To make this determination, all factors which could affect task execution were considered and included in the experiment as necessary. The factors which could affect this performance are illustrated in Figure 1.

Since each model performed identical application tasks, the difference in idle CPU time indicated the model's relative run-time efficiency. Several tests were subsequently developed to determine whether the Ada tasking version significantly affected task execution.

To make this determination, all factors which could affect task execution were considered and included in the experiment as necessary. The factors which could affect this performance are illustrated in Figure 1.

Figure 1. Experimental Factors

Statistical Tests

The factors under investigation were analyzed in accordance with procedures for factorial designs at two levels. These tests looked at two levels for each factor under consideration. For instance, when considering the workload factor, high workload and low workload comprised the two levels. All combinations of factors must be tested unless a fractional factorial design is used. In this study, all combinations were tested except those cases which would not run due to compiler limitations.

Table I. Cyclical Tasks

<table>
<thead>
<tr>
<th>Period</th>
<th>Tasks</th>
<th>Priority (1-low)</th>
<th>Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>A2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>A4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>A8</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>B8</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>16</td>
<td>A16</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>B16</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>32</td>
<td>A32</td>
<td>6</td>
<td>31</td>
</tr>
<tr>
<td>64</td>
<td>A64</td>
<td>7</td>
<td>63</td>
</tr>
<tr>
<td>128</td>
<td>A128</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

These tasks were chosen to realistically model the cyclical tasks found in a typical avionics executive (King, 1987). The phase of a task, which indicates the initial minor cycle for a given task, prevented having too many tasks attempting to run in the same minor cycle. The phase distributed the demand on processing resources evenly over all minor cycles.

Detailed Design

Although the Ada tasking executives and the JOVIAL DAIS executive were designed to handle tasks similarly, the Ada models used the Ada Run Time System (RTS) to handle task interaction while the DAIS contained its own task scheduling functions. A comparison of the models reveals a great degree of conceptual similarity. The implementation differences account for different amounts of overhead for each model. All models are based on a real-time system in which tasks are coordinated with the passage of time. The minimum time granularity in which task activation can be specified to occur is known as a minor cycle. A major frame, the longest period of time which may be specified for a synchronous action to occur, was defined as 128 minor cycles for each model. Initially, both versions were run with a four second major frame duration yielding minor cycles of 1/32 of a second. This time is typical of avionics executives and was subsequently reduced in each model to study the effects of increased workload.

All models were tested with the same series of application tasks so that comparing executives would show differences in overhead. The cyclical tasks used for these tests are shown in Table I.
Asynchronous tasks were subsequently added to each model by calling a linear congruential pseudo-random number generator during each minor cycle. Random numbers were used to represent random events which trigger asynchronous tasks. Depending on the random number encountered, each minor cycle could contain zero to nine asynchronous tasks. The correspondence of random numbers with a particular number of asynchronous tasks was chosen to realistically model the asynchronous task workload in a typical avionics executive (King, 1987). As shown by the figures displayed in Table 11, there is a high probability of few asynchronous tasks in a given minor cycle and a low probability of all nine asynchronous tasks occurring in one minor cycle.

Table I. Asynchronous Task Activation

<table>
<thead>
<tr>
<th>Random Number Generated</th>
<th>Number of Asynchronous Tasks called</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>2 .. 3</td>
<td>8</td>
</tr>
<tr>
<td>4 .. 6</td>
<td>7</td>
</tr>
<tr>
<td>7 .. 10</td>
<td>6</td>
</tr>
<tr>
<td>11 .. 17</td>
<td>5</td>
</tr>
<tr>
<td>18 .. 26</td>
<td>4</td>
</tr>
<tr>
<td>27 .. 37</td>
<td>3</td>
</tr>
<tr>
<td>38 .. 49</td>
<td>2</td>
</tr>
<tr>
<td>50 .. 65</td>
<td>1</td>
</tr>
<tr>
<td>66 .. 100</td>
<td>0</td>
</tr>
</tbody>
</table>

Workload, the last factor common to all models, was studied by changing the time between minor cycles. A shorter time between minor cycles represents a higher workload per period of time. The remaining factors, Ada compiler selection and system timing method, were compared only with each Ada version since these factors do not apply to the DAIS model. Having explained design details common to both models, the DAIS and Ada models will now be described separately in greater detail.

DAIS Model

The DAIS uses events for task scheduling purposes. Events refer to occurrences such as the start of a new minor cycle or actuation of a switch which could trigger task activation. Random events such as actuation of a switch are simulated by random numbers in this experiment. System timing to establish the interval between minor cycles is performed by the master executive which sets a hardware clock to interrupt whenever one minor cycle has elapsed (ASD, 1982:47). The DAIS controls task states by referencing the table of tasks and scheduling tasks with respect to events and priority.

Ada Tasking Model

The Ada tasking model uses the Ada RTS to control task state in a manner similar to the DAIS executive. System timing in an Ada executive is not handled automatically by the RTS but must instead be established by the program designer. The delay construct can be used to establish the desired interval between minor cycles. Another alternative is to set a hardware clock, as in the DAIS, and tie the clock interrupt to the start of a new minor cycle. Both methods of system timing were evaluated in this experiment to determine the significance of tasking methodology on system overhead as will be explained in the next sections.

Ada with Delay Statements.

Cyclical tasking is difficult to implement in Ada without using interrupts because the RTS provides no automatic means of system timing. Nevertheless, it can be done with a minor cycle task calling a scheduler task at the beginning of each minor cycle. The scheduler task in turn calls each task to be run during the minor cycle. The program structure is shown in Figure 2.

Asynchronous tasks were subsequently added to this model by calling the random number generator from within the Scheduler task. The workload was changed by changing the minor cycle duration.

```
    task Minor_Cycle
      loop 1..Number_of_Major_Frames
        loop 1..128
          start Scheduler
          delay Minor_Cycle_Duration
          end loop
        end loop
      end Minor_Cycle

    task Scheduler
      loop
        accept start do
          If proper period, phase and conditions then
            start appropriate application task
        end loop
      end Scheduler

    task cyclical_task
      loop
        accept start do
          counter := counter + 1
          if counter > 5000 then
            counter := 0
          end if
        end loop
      end cyclical_task
```

Figure 2. Ada Program Structure

Ada with Interrupts.

In an attempt to obtain more accurate system timing, a similar tasking model was developed with minor cycle timing handled by an interrupt tied to
a hardware clock. The interrupt was used to call the minor cycle task rather than using a delay to maintain system timing.

The clock is set and the interrupt vector is handled through 1750A assembly language programs. These programs are combined with the Ada programs and called through Ada's pragma interface. A separate assembly routine is needed for each minor cycle duration desired.

This model was also tested with the same asynchronous tasks and by varying the workload. The results of all test combinations are found in the results section.

Overhead Measurement and Statistical Analysis

Overhead Measurement.

Determining how to measure executive overhead was the most pervasive problem of this study. Executive overhead could not be isolated for measurement with calls to a system timer. The only method found to measure overhead was to measure idle CPU time. Idle CPU time was tracked by a low priority task which incremented a value whenever higher priority tasks were not ready to run. The count was performed through an assembly routine which incremented a register every time called. Overflows were detected and saved in a second register. At the end of each major frame, these registers were read and reset back to zero. The CPU idle time was then analyzed by comparing the number of increments.

In the Ada models, the assembly routine was called by a single task of low priority which executed in an infinite loop. Since the DAIS would not allow a task to run in two consecutive minor cycles, two separate low priority tasks were established. One task called the assembly routine in phase zero and the other called it in phase one. Both tasks had a period of two so they would execute every other minor cycle.

Statistical Analysis.

The first factors tested were those pertinent to the Ada models alone. Unfortunately, only one compiler was able to successfully compile and run the programs, so the compiler effect could not be evaluated. The effect of the system timing method was analyzed, however, and used to determine the significance of this factor. Once this factor was evaluated, all combinations of remaining factors were compared between the DAIS, Ada with delay statements, and Ada with interrupts.

The number of experimental repititions was determined statistically and five runs were performed. The output of these runs was analyzed with Analysis of Variance procedures to determine the significance of each factor.

Experimental Results

All factors were evaluated with the exception of the comparison of the compiler effect on Ada executives. All four of the Ada compilers tested had "bugs." As a result of these problems, only one compiler successfully provided executable code. This section begins by describing the problems encountered with each compiler and then proceeds to actual test results and statistical interpretation of these results.

Compiler Problems

The compilers used will not be identified by name, but will instead be referred to as A, B, C, and D to avoid mentioning proprietary information. Compiler A was able to run all tests successfully. The other compilers were unable to execute the tests for various reasons. Compiler B compiled and linked successfully, but produced a tasking error during run-time. Compiler C compiled successfully, but encountered an internal linking error. Finally, Compiler D encountered an internal compiler error and could not compile the Task Scheduler package. All compiler problems were submitted to the compiler vendors for debugging if the vendors had a current contract with the Air Force for that compiler.

Even if all four compilers did work, only two would have handled the interrupt version because only all Ada Language Reference Manual chapter 13 features are implemented by each compiler. The compiler specifications with respect to each pertinent feature are shown in Table III.

Test Results

The results are reported for each combination of factors in Table IV. Due to the small amount of variance between runs, the five initial runs were sufficient to obtain an answer within 10% of the mean for all test combinations. Therefore, no more runs were necessary.

The results indicate that the Ada model with interrupts has fewer increments than the DAIS or the Ada model with delays. Since the number of increments was shown to correspond with idle CPU time, a model with more idle CPU time is more efficient in terms of executive overhead than its counterpart. The results can therefore be interpreted to show that the Ada model with interrupts was less efficient than the DAIS or the Ada model with delays. There is a significant difference between the Ada version with delays and the DAIS, but this difference is compounded by the interaction of both workload and task mixture. The results are sufficient to provide an equation which can predict idle CPU time, but do not indicate that Ada tasking with delays is better or worse than the DAIS in all cases.

The results from Table IV show that the DAIS has more idle CPU time under low workload.
Table III. Compiler Features

<table>
<thead>
<tr>
<th>Used Pragma Address</th>
<th>Address Clause for Interrupt Handling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler Successfully Interface Clause</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>Yes</td>
</tr>
<tr>
<td>B</td>
<td>No</td>
</tr>
<tr>
<td>C</td>
<td>No</td>
</tr>
<tr>
<td>D</td>
<td>No</td>
</tr>
</tbody>
</table>

Table IV. Idle CPU Time for all Combinations of Factors

<table>
<thead>
<tr>
<th>Ada with Delays</th>
<th>Ada with Interrupts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Number</td>
<td>Cyclic Mixture Low High Low High</td>
</tr>
<tr>
<td>1</td>
<td>238603 138497 215931 112478</td>
</tr>
<tr>
<td>2</td>
<td>238634 138528 217247 114586</td>
</tr>
<tr>
<td>3</td>
<td>238700 138560 216593 113972</td>
</tr>
<tr>
<td>4</td>
<td>238815 138355 215496 112474</td>
</tr>
<tr>
<td>5</td>
<td>238915 138653 213989 113501</td>
</tr>
<tr>
<td>Mean</td>
<td>238733 138518 216251 113556</td>
</tr>
<tr>
<td>Std Dev</td>
<td>130 108 680 790</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DAIS Run Number</th>
<th>Cyclic Mixture Low High Low High</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>248880 120550 242689 114328</td>
</tr>
<tr>
<td>2</td>
<td>248864 120560 242702 114530</td>
</tr>
<tr>
<td>3</td>
<td>248693 120545 242564 114628</td>
</tr>
<tr>
<td>4</td>
<td>248691 120546 242434 114297</td>
</tr>
<tr>
<td>5</td>
<td>248673 120550 242408 114291</td>
</tr>
<tr>
<td>Mean</td>
<td>248684 120550 242519 114374</td>
</tr>
<tr>
<td>Std Dev</td>
<td>8 6 118 103</td>
</tr>
</tbody>
</table>

Table III. Compiler Features

conditions than Ada tasking with delays regardless of the application task mixture. Ada tasking with delays, on the other hand, performs better than DAIS under high workload with cyclical tasking. Neither version shows a significant advantage under high workload conditions with a mixture of cyclical and asynchronous tasks.

The Interrupt driven model displays approximately 30 to 50 percent more processing overhead than the Ada model with delays for all combinations of factors. For example, with cyclical tasks and low workload, Ada with interrupts has 180,600 increments per major frame and Ada with delays has 238,733. This shows the interrupt model having 24 percent greater overhead than the delay model under these conditions.

The results were then analyzed with Analysis of Variance (ANOVA) procedures to determine which factors significantly influenced executive overhead. The null hypothesis for each test was that the factor was not a significant influence; the alternate hypothesis was that the factor provided a significant influence. Initially, the Ada with delay and Ada with interrupt versions were compared with one another. Each factor under consideration was found to have a significant influence. Since the Ada version with delay statements performed better than the interrupt version, the DAIS was compared with the delay version. Once again, each factor under consideration was found to have a significant influence.

Test results given in number of increments per major frame.

Heading codes: Cyclic/Mixture indicates purely cyclic tasks or mixture of cyclic and asynchronous tasks. High/Low indicates high or low workload.

744
Conclusions

The Ada executive models developed and analyzed in this study show that Ada tasking does not possess inherently high overhead when compared with the DAIS. On the contrary, executives can be developed, using Ada tasking with delay statements, which have comparable overhead to the DAIS. These executives, however, are prone to timing problems which may negate Ada tasking utility.

The conclusions drawn are limited in the sense that they are based on a comparison of the DAIS and an Ada tasking executive rather than absolute overhead requirements for any avionics executive. Nevertheless, since the DAIS is representative of avionics executives and the application tasks were chosen to model typical avionics systems, the applicability of Ada tasking for avionics executives can be inferred from the results herein. The final analysis of Ada tasking can only be made by using tasking in a complete executive which is ultimately flown and flight tested. The results in this study indicate there is a significant amount of risk and problems that may be encountered if Ada tasking is mandated for such a program.

Other limitations of this study include the fact that various architectures including multiprocessor implementations were not evaluated. In addition, several other Ada compilers targeted toward the 1750A exist, but were not available for use. Likewise, various JOVIAL executives exist, but were not available for experimentation due to proprietary reasons. These limitations must be borne in mind when considering the conclusions reached in this study. The results indicate that Ada tasking with delays has nearly the same overhead as the DAIS and is therefore worthwhile for avionics applications. The problem of cumulative drift, however, confounds this conclusion. Ada delay statements allow cumulative drift because delays are by definition a minimum time interval. Ada is therefore able to perform better under high workloads by postponing the delay expiration.

Although unproven, it is doubtful that the Ada tasking version with delays would be able to handle avionics tasks effectively. Timing problems would most likely have to be handled. Handling these problems implies creating more system overhead as in the interrupt driven model.

While providing more accurate timing, the interrupt driven model is likely to cause excessive demands on the processor. Given the size and weight constraints of embedded avionics systems, increasing processor capacity to accommodate Ada tasking is not a viable alternative.

The DAIS also maintains precise timing by associating the minor cycle duration directly with a system timer. This is nearly analogous to the Ada executive with interrupts. The DAIS therefore limits drift and associated timing problems.

Compiler development and enhancement should make Ada tasking more efficient. Ada compilers targeted toward the 1750A are relatively immature at this time.

Recommendations

Compiler evolution alone may not be sufficient to make Ada tasking effective for avionics systems. Tasking itself should be redesigned to efficiently handle cyclical tasks. A software interrupt is necessary to provide precise timing for cyclical tasks, but this interrupt need not be tied to a task entry. Rendezvous with an interrupt, as seen in the results, creates a high degree of system overhead.

A table driven approach, such as the DAIS, is one means of providing efficient tasking. Another approach is to develop a pragma "cyclical executive" as proposed by Phillips and Stevenson (1984). This pragma would associate an interrupt with a task entry for each desired frequency. Although this approach promises more accurate system timing, the affect on system overhead could be detrimental as in the Ada tasking with interrupt model developed in this study.

A combination of the above proposals shows great promise for providing accurate timing while maintaining low overhead. The pragma "cyclical executive" can be used to provide system timing, and cyclical tasks can be table driven as in the DAIS. In this manner, the task scheduler need not rendezvous with each application task to be run. Instead, the application task can be flagged to run without incurring the overhead of an additional rendezvous.

Future efforts should be made to provide precise cyclical tasking without excessive overhead. Specifically, the overhead associated with table driven cyclical tasks should be investigated. The overhead measurement techniques used in this study can be applied to measure idle CPU time and thus compare system overhead of future developments.
References


