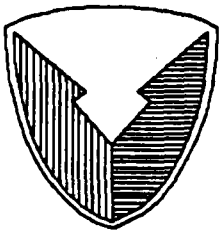

CECOM

CENTER FOR SOFTWARE ENGINEERING

ADVANCED SOFTWARE TECHNOLOGY

Subject: **Final Report - Documentation for Real-Time Performance Benchmarks for Ada**

CIN: **C02 092LY 0002 00**

24 MARCH 1989

Documentation for

Real-time Performance Benchmarks

For Ada

Contract Number: DAAA21-85-C-0238

Prepared For:

U.S. Army, CECOM
Advanced Software Technology
AMSEL-RD-SE-AST-SS-R
Ft. Monmouth, NJ 07703-5000

Prepared By:

Arvind Goel
TAMSCO
145 Wyckoff Road
Eatontown, NJ 07724

October, 1988.

This file contains notes pertaining to the real-time benchmarks that have been developed by the Center for Software Engineering, US Army, Ft. Monmouth, NJ This file should be read before the benchmarks can be executed. This file supplements the discussion of the benchmarks in the report titled "Real-time Performance Benchmarks for Ada". Since the benchmarks developed were run on the Verdix Ada Compiler System (hosted on the SUN 3/60 and targeted to the MC68020 bare target), these files are tailored to run on the Verdix compiler. But they can be easily changed to run on other compiler implementations.

1. Directory Structure

At the top level of the directory structure is the directory **bench**. There are 5 directories under **bench**:

1. **micro**: The directory micro contains the benchmarks that measure the performance of Ada features that are important for real-time embedded applications;
2. **rts**: The directory rts contains benchmarks that measure runtime implementation dependencies;
3. **paradigms**: The directory paradigms contains benchmarks that implement macro constructs and real-time programming paradigms;
4. **documentation**: The directory documentation contains the relevant documentation that describes the benchmarks; AND
5. **etc**: The directory etc contains files needed for execution under the Verdix Ada compiler system targeted to the MC68020 and hosted on the SUN 3/60 workstation.

2. Micro

The directory micro contains the directories that contain the micro benchmarks. Under each directory, there are three files with the suffixes: _compile, _loader, and _run. The file with the suffix _compile lists the compilation order for the tests in that directory. The file with the suffix _loader creates the object files for the tests in that directory. The file with the suffix _run runs the object files created in that directory.

The directories under **micro** are as follows:

- **lv** lv contains loop verification benchmarks. This benchmark is executed to verify that textually similar loops should take equal amount of time to execute.

- **t**: t contains tasking activation/termination benchmarks. These benchmarks measure tasking activation/termination timings under various conditions.
- **r**: r contains tasking synchronization benchmarks. These benchmarks determine the time required to perform rendezvous under various loads and conditions.
- **ex**: ex contains exception handling related benchmarks. Exception handling and propagation timings are measured by these benchmarks.
- **chap13**: chap13 contains Chapter 13 related benchmarks. These benchmarks may not compile for some Ada compiler systems.
- **pragma**: pragma contains benchmarks related to pragmas.
- **dd**: dd contains benchmarks that measure dynamic allocation time in declarative regions.
- **dn**: dn contains dynamic allocation with new operator benchmarks (file dn_compile contains the list of benchmarks that need to be compiled for these cases). The directory dn also contains benchmarks that determine allocation time without memory being freed by UNCHECKED_DEALLOCATION (file dn_compile1 contains the list of files that need to be compiled for these benchmarks). The directory **new** under dn contains tests that determine the affect of additional tasks on time for dynamic allocation (file dn_compile contains files for 5 tasks and dn_compile1 contains for 10 tasks).
- **co**: co contains Clock function calling overhead and resolution benchmarks.
- **io**: io contains input/output benchmarks.
- **tm**: tm contains mathematical benchmarks.
- **d**: d contains regular case do nothing subprogram overhead benchmarks.
- **i**: i contains inline subprogram overhead benchmarks.
- **p**: p contains cross package subprogram overhead benchmarks.
- **g**: g contains generic subprogram overhead benchmarks.
- **c**: c contains generic cross package subprogram overhead benchmarks.

To change the accuracy of the results desired, one can change the parameters in the spec of the package LOOP_CONTROL. The actual number of iterations performed will be the product of INNER_ITERATIONS and OUTER_ITERATIONS.

Before running the tests you should determine the clock resolution by compiling and running the tests under the co directory. Under each directory, the LOOP_CONTROL package which has parameters for the number of iterations the test will run, they are currently set for a ten millisecond clock. If the second difference tests indicate a clock resolution other than ten milliseconds, the INNER_ITERATIONS and OUTER_ITERATIONS parameters should be scaled appropriately. The lower the clock resolution, the higher the required number of

iterations.

2.1 Subprogram Overhead

For subprogram overhead, other characters in the benchmark names are as follows:

Other characters (for subprogram overhead):
i - integer
e - enumeration
a - array
r - record
n - none
u - unconstrained
t - trivial
c - call

A number indicates size or number of parameters. The call files contain drivers for each individual test. All files of one directory should be compiled using the compile file (e.g. d_compile for the d directory) to determine compilation order. The compile files use the compile command for the compiler you are running. In the case of the generic subprogram tests, the compilation order may need to be changed. Your compiler will complain if this is the case. To fix this, compile all files with over and triv in their names (triv first, over second) before compiling files with _c.a in their names. After compiling all files for a directory, each call unit (e.g. the unit in d_n_c.a) should be linked into an individual test. Thus, there is one test for each call file.

The values of INNER_ITERATIONS, OUTER_ITERATIONS, and NUMBER_OF_TESTS should be adjusted as needed for the compiler used. These value appear in the loop_spec.a package in each directory. They should NOT be changed to constants, this is an optimizer protection. In the delay test, the values HOW_FAR and MULT_OF_D_SMALL must also be adjusted for a particular test. These values occur in the same package and are commented there.

2.2 Dynamic Allocation

The file names for the dynamic allocation tests follows the format of xx_ddttnnU.a where

xx is either "dn" or "dd" for declarative region or new allocation
dd is the dimensionality of the object being declared. this is omitted for scalar objects.
tt is the type declared in the test:
in - integer
en - enumeration
rc - record
st - string
ar - statically bounded array
d - dynamically bounded array
nn is the size of the object declared, for arrays this is the total number of elements in the array.
U indicates the program unit in the file
S - Specification for packages
B - Body of first package
B2- Body of second package
omitted - body of main

3. rts

The directory rts under bench contains the benchmarks that determine runtime implementation dependencies. The following directories exist under rts:

- **t**: t contains benchmarks that determine tasking related runtime implementation dependencies.
- **r**: r contains benchmarks that determine rendezvous related runtime implementation dependencies.
- **mm**: mm contains benchmarks that determine memory management related runtime implementation dependencies.
- **ex**: ex contains benchmarks that determine exception related runtime implementation dependencies.
- **dt**: dt contains scheduling and delay statement benchmarks.
- **io**: io contains Input/Output benchmarks.

4. paradigms

The directory paradigms contains the benchmarks that determine the performance of macro constructs and real-time paradigms.

5. Verdix Ada Compiler System

To run the benchmarks under the Verdix Ada compiler system targeted to the MC68020 and hosted on SUN 3/60, the following steps are needed.

1. Do a.mklib on the directory where the benchmarks are located (e.g. t).
2. cd t and copy the files bench/etc/ada.lib and bench/etc/debug.inp to t.
3. Edit the file ada.lib to change the path name of the link modules.
4. The file link.dat under bench/etc specifies the sizes of the code, data, and bss sections.
5. Execute the file t_compile, and t_loader and the output should be files in the format a.out.
6. Download the a.out to the MC68020 and execute the file using the a.db command.

NOTE

The benchmarks are contained on a 1/4" cartridge tape in tar format. If interested in obtaining a copy please contact Mary Bender, 201-544-2105 for details.