

# Comparison of the Execution Times of Ada, C and Java

Marcus Weiskirchner  
*EADS Deutschland GmbH*  
*Military Aircraft*  
*81663 Muenchen, Germany*  
*marcus.weiskirchner@m.eads.net*

September 25, 2003

## Abstract

This paper presents a comparison of the execution times in Ada, C and Java. Two Benchmarks were used for this comparison, Whetstone [CW76] and Dhrystone [Wei88]. These benchmarks were compiled with Ada-Multi [GHS03] and Ada-GNAT [GNA01], with two Java Real-Time Virtual Machines Perc VM [New02] and Jamaica VM [SW01], with the gcc C-compiler as a reference program and interpreted with the Sun VM [LY99]. Furthermore, different operating systems have been used, like VxWorks as a real-time operating system, Windows, Linux and Unix Sun Solaris.

## 1 Introduction

This report should give a comparison of the execution times in Ada, C and Java, using two Benchmarks, Whetstone and Dhrystone. Section 2 gives a short description of both benchmarks, describing what tests are made and how the specific measurements were made. Section 3 tries to interpret the benchmarks execution times on a VxWorks system with all the existing compilers for VxWorks in Java, Ada and C. Some differences in the compiled programs are also mentioned. Section 4 lists all used operating systems and how the execution times of the benchmarks and the compilers change, depending on the underlying operation system. Not every compiler was available for every operating system. Section 5 includes some further information, like a comparison of the filesizes of different compiled programs, ahead-of-time (AOT) versus just-in-time (JIT), the reproducibility of the benchmarks and some unexpected results of the test. All diagrams of the benchmark's execution times are included to appendix A to D. Appendix E lists the used

compiler options and appendix F to H contains the source codes of both benchmarks in all three languages.

## 2 Used Benchmarks

Two benchmarks have been used for measuring the execution time. Dhrystone [Wei88] as a fixed point benchmark and Whetstone [CW76] as a floating point benchmark. It was important to have some benchmarks available in Ada as well as in Java. The benchmarks in C are a kind of reference.

### 2.1 Whetstone

There are two versions of the Whetstone benchmark [CW76], which had been tested. One with a 64-bit calculation and the other with 32-bit calculation. All test results are included to the appendix. Most tests are faster in the 32-bit calculation, that's why the comparison of the compilers isn't different from the 64-bit test.

The benchmark has 8 different single tests. These tests are executed in a loop with varied loop cycles. The following table gives a short description of the single tests and the weight of each test:

- Array elements (floating point), 12
- Array as parameter (floating point), 14
- Conditional jumps (if then else), 345
- Integer arithmetic (fixed point), 210
- Trigonometric functions (sin, cos, etc.), 32
- Procedure calls (floating point), 899
- Array references (assigns), 616
- Standard functions (exp, sqrt, etc.), 93

The whole test was executed with an iteration of 100 in each single test and a loop cycle of 100 over all single tests. This full test was run ten times, incrementing the loop cycle by ten (means 100 cycles first time, 110 second, 120 third, etc.). The result of the test is the average execution time for one single cycle and the average over all ten tests.

### 2.2 Dhrystone

The Dhrystone benchmarks [Wei88] is used to model what was viewed as a "typical" application mix of mathematical and other operations. Integer performance predominated, with little or no floating-point calculations, and

applications could be contained inside small memory subsystems. Most operations include invoking other functions and procedures, doing some calculations and returning the result to the invoking procedure. In Ada and C pointers are used to return results. Because Java doesn't use pointers that way an object was created to return the result to the invoking methods. The first version of the Java program was creating a new object every time the method was called. Thus the comparison of the languages wasn't 100% fair and that's why Java took a lot more time for executing this benchmark. Therefore in the modified version (or better: the more compareable version) of the benchmark, a global object (reference) was used to return the result to the invoking method. Now the test isn't measuring the time it takes to create an object, but the time for executing the methods. Because of this small conditioning the Java benchmark is up to four times faster.

One run of the benchmark includes 100.000 iterations of the main loop. The result of the benchmark is the average time needed for one run.

### **3 Interpretation of the Test Results**

The more interesting testing was made on the real-time operating system VxWorks. Thus these test results are taken for the further discussion. Otherwise the results of the other operating systems and the different hardware is similar to the VxWorks system. All other test results are included in appendix A to D.

#### **3.1 Whetstone on VxWorks**

Figure 1 shows the result of the Whetstone benchmark on a VxWorks system, version 5.4 on a VME board with a PPC 400 MHz and a 750 kernel. To have a fair comparison, some details have to be considered. Including or excluding **runtime-checks** has a big influence in optimising speed. C does not have any checks during runtime. Java is always performing runtime-checks and in Ada runtime-checks can be switched on or off by a compiler switch.

In figure 1 both Ada compiler use runtime-checks. Therefore we have a more or less direct comparison of the execution time of the Whetstone benchmark in Ada and in Java. All programs are optimized for speed, or better to say, the used compiler options are optimized for speed. The result of the benchmark was a bit suprising. I did not expect a faster execution time in Java than in Ada, compiled with Ada-Multi. But at a second try I used the Ada-GNAT compiler, which seems to compile the program better, means the GNAT compiled Whetstone runs faster than the Java program, compiled with the Jamaica VM.

The last bar in the diagram shows the execution time of the C-compiled program. This is the fastest version. On the other hand there must be noted, that the C-version doesn't make any runtime-checks. Therefore there is an-

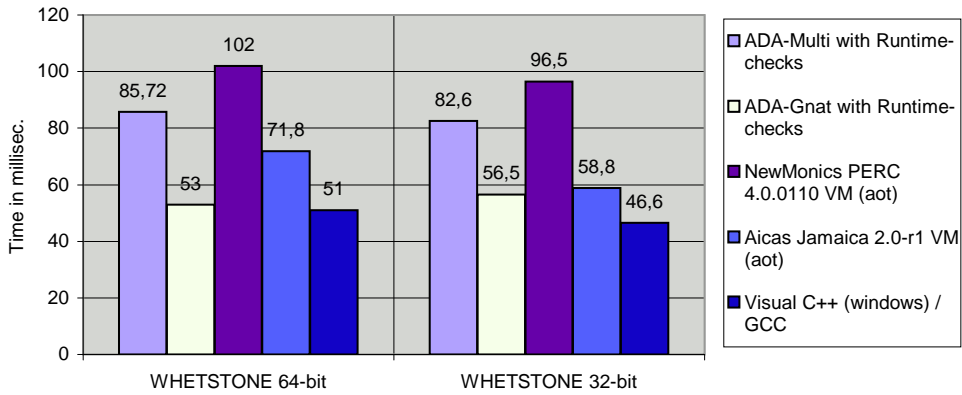


Figure 1: Whetstone 64/32-bit with Runtime Checks

other diagram comparing the Ada compiled programs with the C-version. Here the timings of the Ada compiled programs are closer to the C-version (see fig. 5).

### 3.2 Dhrystone on VxWorks

Figure 2 shows the result of the Dhrystone benchmark on a VxWorks system, version 5.4 on a VME board with a PPC 400 MHz and a 750 kernel. As mentioned above in the Whetstone benchmark, including or excluding **runtime-checks** has a big influence in optimising the speed.

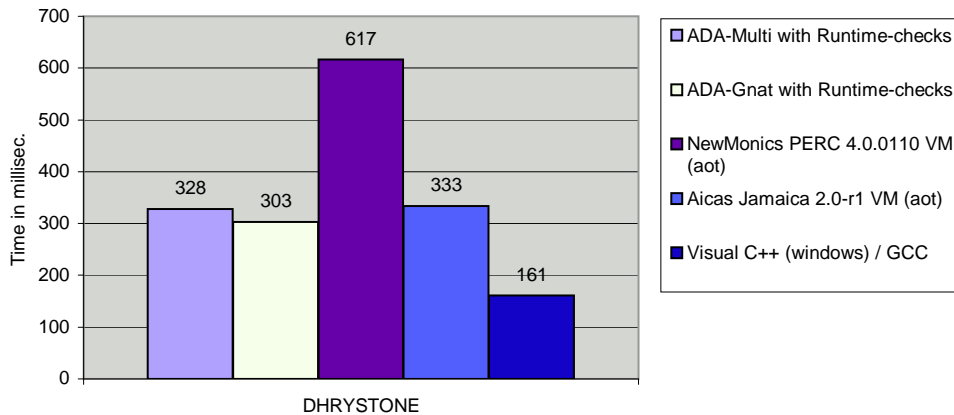


Figure 2: Dhrystone with Runtime Checks

In figure 2 again two Ada compilers (Ada-Multi and Ada-GNAT) and

two Java Real-Time Virtual Machines are compared. This time the Ada programs are executed faster or equal than the Java programs, depending on the used compiler. Again, all benchmarks without runtime-checks make the Ada programs almost two times faster.

## 4 Differences in Operating Systems

### 4.1 Windows

In figure 3 both Ada programs, the Perc VM, the Sun VM and the C program are compared. Again to be correct the C program is a bit out of the comparison, making no runtime-checks. But also the Sun VM is out of place, because it can not guarantee any time-constraints (no real-time garbage collection). These results should just be a reference to the results.

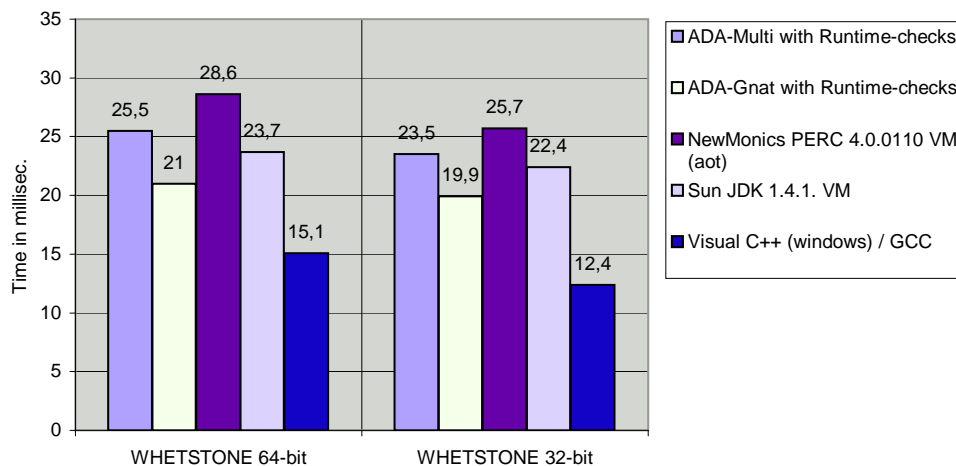


Figure 3: Whetstone 64/32-bit with Runtime Checks

But still, if you compare the Ada programs with the Perc VM, the Java program is almost as fast as the Ada programs. The difference between Ada to C gets smaller without runtime-checks (see Fig. 13, p. 13)

The Dhrystone benchmark has different results than the Whetstone. Here the Ada-Multi program needs half the time as Ada GNAT. The reference programs with C and Sun VM have their own competition and are a lot faster than the others. The interpreted (JIT) Java program needs almost the same time as the C program. The RT Perc VM executes the benchmark a little bit slower. It also seems, that the JIT compilation works perfect for this example. The full Dhrystone benchmarks was run 10 times (started by the main-method), the first run was the slowest, and all other runs were faster.

An explanation for this effect can be, that reused parts of the program don't have to be recompiled and therefore are executed faster than the first run.

## 4.2 Sun Solaris

Appendix C.1, p. 17, show the results of Ada programs with/without runtime-checks and of the JIT interpretation of the Sun VM. As a conclusion it can be said, except of Ada-Multi, all other execution times have more or less the same execution time (between 32 ms and 42 ms).

The Dhrystone benchmark has a similar view. Just the C program needs half the execution time as the GNAT (without runtime-checks).

By the look at the diagrams always have in mind that there are a lot of differences in the programs itself. Here the Ada programs with and without the runtime-checks are in the same diagram, as well as the C program (no runtime-checks) and the Java program, which is a JIT compilation, means just-in-time compilation with no timing guarantees.

## 4.3 Linux

The appendix C.2, p. 18 shows the diagrams of Whetstone and Dhrystone on a Linux operating system. Four different programs compiled with Perc, Jamaica, Sun VM and C are compared. The two RT Virtual Machines have almost the same execution time. The C Programm is the fastest, but has still a similar execution time. The Dhrystone shows a very clear picture, RT Virtual Machines take a lot more time to execute the benchmark as Sun VM and the C program, which are two respectively almost five times faster.

# 5 Additional Information

## 5.1 Filesize

Sometimes the file size can also be an interesting issue. Therefore appendix D.1, p. 19 shows a bar diagram of the different languages and the file sizes. For VxWorks Ada must include the Ada-Runtime Object adalib.o. Java programs have a similar problem and have to include the Java Virtual Machine. But as it looks in the size comparison, Perc VM includes a lot more libraries than Jamaica or doesn't make a perfect optimization of code. But no other language can beat the C compilation, which size is just 8 kb.

On a Windows System the file sizes are similar and again the Perc VM needs a lot of disc space. There is no Version of Jamaica available for Windows.

## 5.2 AOT versus JIT

The Perc VM includes ready compiled Virtual Machines for every platform. Thus it is possible to let run Java byte code directly with this JIT VM. To optimize the bytecode before starting the program, an optimization tool is included to get an "accelerated" bytecode. The figure D.2 compares the execution time of a AOT compiled program and the JIT version of the benchmarks on different platforms. With the two benchmarks the AOT version and the JIT version of the program have almost the same execution time.

## 5.3 Reproducibility

During the benchmarks it was conspicuous, that the execution times of some single test were not exactly the same time as in the test before. As a first assumption I disabled the instruction and data cache to get rid of cache effects. To be sure that the telnet daemon is not working during the Benchmark, I used the serial connection for communication. All these test were made on the VxWorks system to have no operating system side effects. But the runtime effects were still the same.

The conclusion of these tests are: Ada and C always produce the same execution times on every single test, only a tolerance of one millisecond on some tests. Jamaica produced a less reproduceable benchmark view. There were much more differences in the single tests, but still less than 0,5% tolerance overall. The Perc VM has differences in the single tests up to 1,5% tolerance overall.

Some speculations of Aicas, why there is a small tolerance:

- The synchronisation thread is on. Should be off by default in VxWorks. Manually set off achieved no difference.
- The benchmark itself runs within different threads with same priority. Different priorities of threads could help and should be mapped to different VxWorks priorities. This is the default for VxWorks.

## 5.4 Curiosities

The Whetstone benchmark compiled with the **Ada-GNAT** for VxWorks executes the 64-bit program faster than the 32-bit version, what is an unexpected result.

Comparing the Dhrystone benchmark compiled with **Ada-GNAT** and **Ada-Multi**, the Ada-Multi version is two times faster, but only on the Windows operating system.

## 6 Conclusion and Future Work

There are a lot of dependencies influencing the execution times of the benchmarks. The benchmarks itself have totally different results in execution times. Not every compiler is available for every operating system. Every compiled program has a different execution time, depending on the used compiler. The language is not the main reason, it is more the used compiler. For the Whetstone benchmark it definitely depends on the used compiler, most of the platforms provide a similar result. The Dhrystone benchmark has different execution times on different platforms. The first version of Dhrystone was probably translated to Java by a automatic tool. Therefore on every method invocation a new object was created, unlike the original C or Ada code. This artificial object creation was removed from the Java code for the comparison of the three languages.

The runtime-checks are also a big issue, Java always uses runtime checks, C never and Ada by compiler switch. Therefore two kinds of diagrams are appended. Java is always compared with the programs using runtime checks (except C). Some tests with the Sun VM were also made, although this is a non real-time VM having no RT garbage collection.

Figure 1 gives probably the best comparison of this paper, comparing two Ada compiled programs, two Java compiled programs (with real-time Java VMs) and a C compiled program. All tests were made with the RT operating system VxWorks on a Power PC. The benchmark itself makes a lot of calculations with floating points, no pointers are used.

As a conclusion, the difference in execution time between realtime Java and Ada is less than expected. Differences between the Ada runtime environments seem to have a more significant impact on execution speed than choosing between Ada and Java as long as runtime-checks are used. As demonstrated by the machine-translated Java benchmark program, it is important to avoid object creation in methods that are executed frequently inside loops.

A more detailed test on tasking should be made in future. Hartstone would be a good standardized benchmark for testing task switches and task behavior, but is presently not publicly available for Java. A short benchmark sending and receiving messages between real-time tasks using a message queue can also help to have a better view of task behavior and timing constraints of Java and Ada.

## A Benchmarks on VxWorks Systems

### A.1 VxWorks 5.5 with a PPC603 350MHz

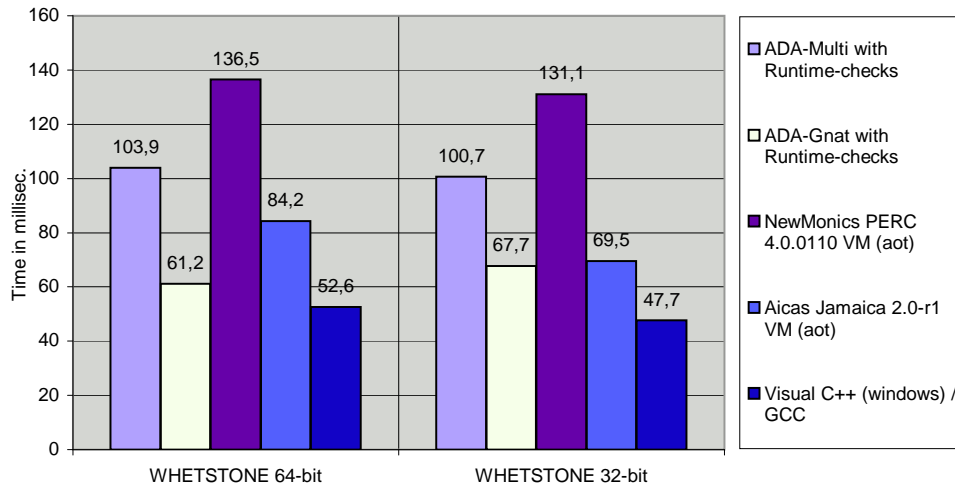


Figure 4: Whetstone 64/32-bit with Runtime Checks

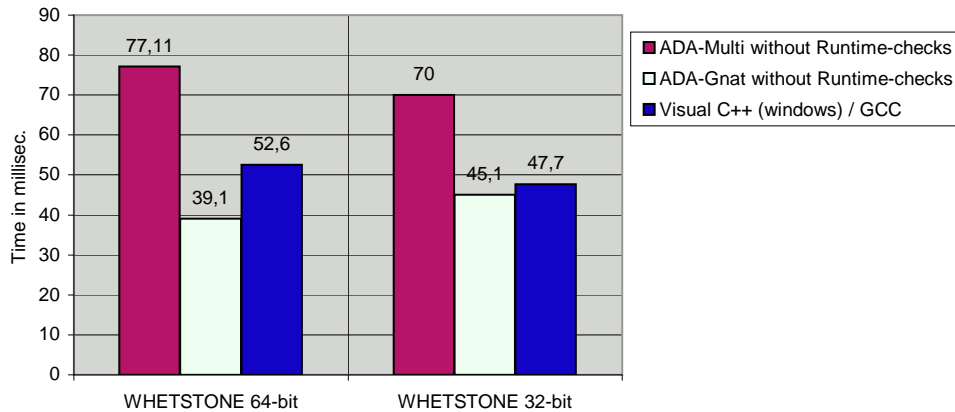


Figure 5: Whetstone 64/32-bit without Runtime Checks

A.1 VxWorks 5.5 with a PPC603 350MHz Benchmarks on VxWorks Systems

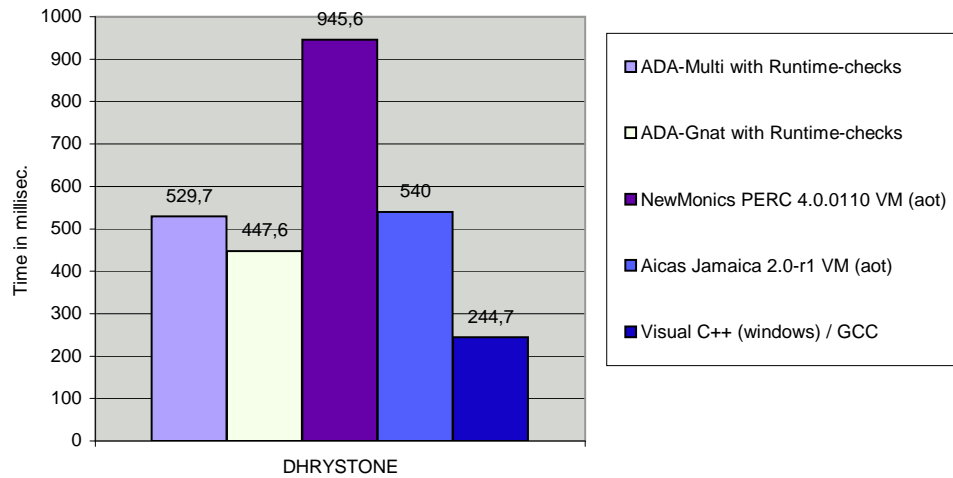


Figure 6: Dhrystone with Runtime Checks

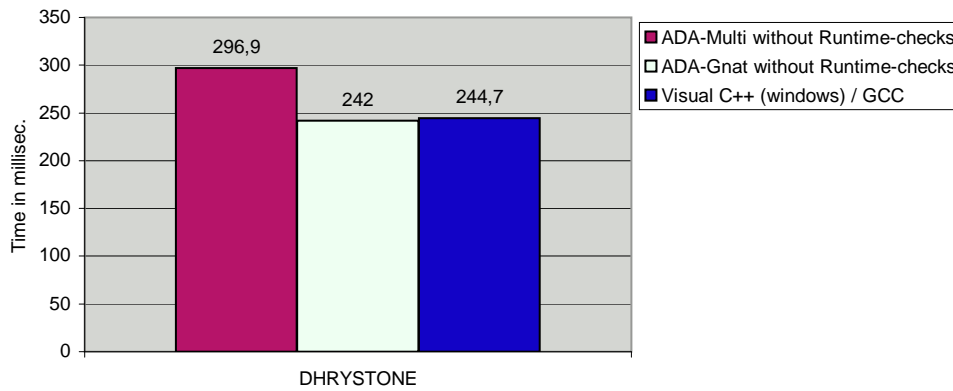


Figure 7: Dhrystone without Runtime Checks

**A.2 VxWorks 5.4 with a PPC750 400MHz**

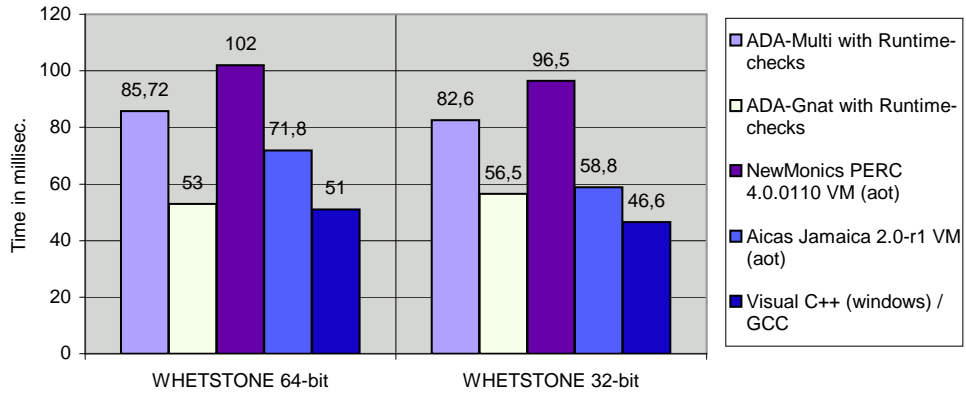


Figure 8: Whetstone 64/32-bit with Runtime Checks

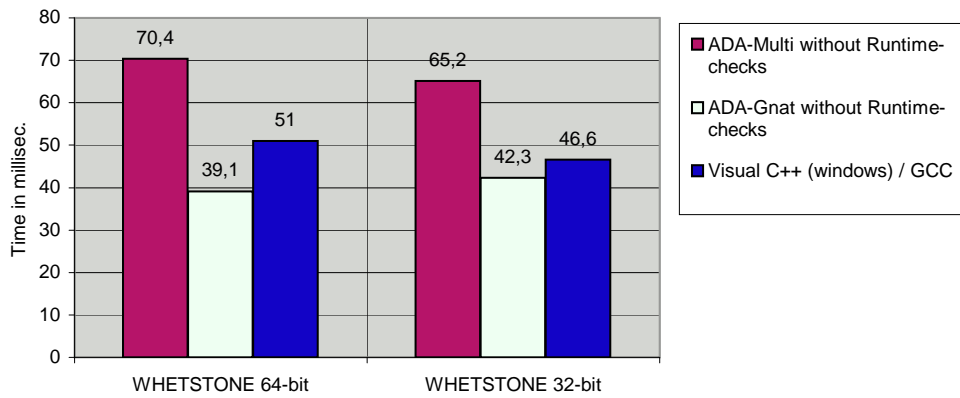


Figure 9: Whetstone 64/32-bit without Runtime Checks

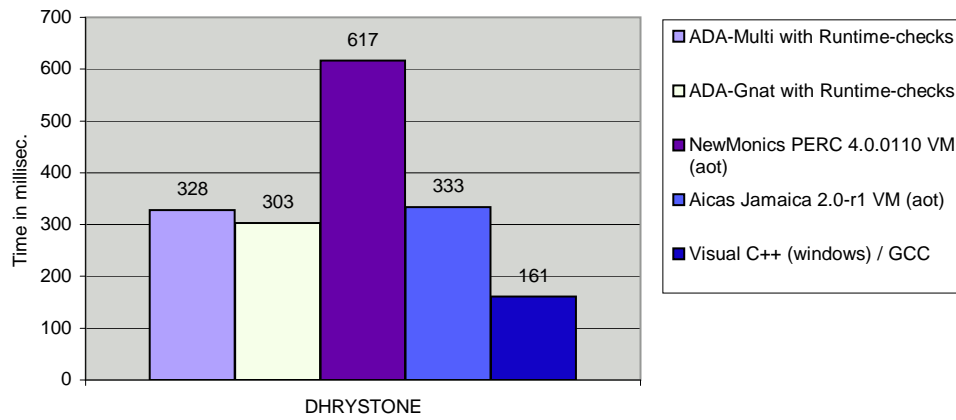


Figure 10: Dhrystone with Runtime Checks

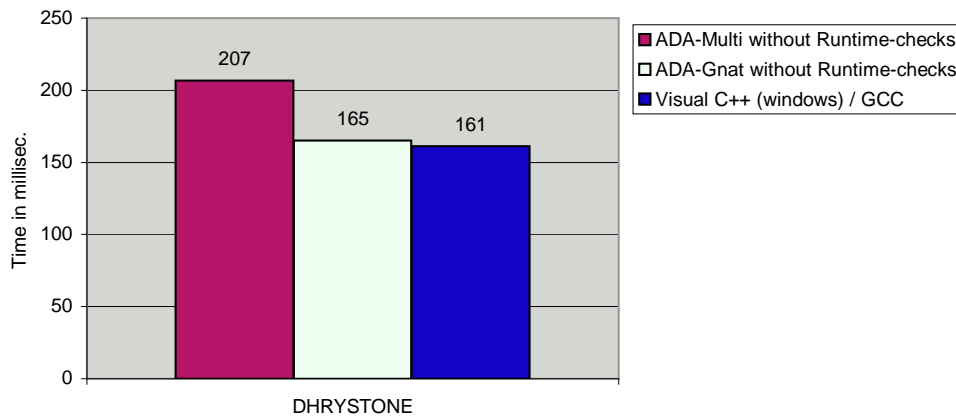


Figure 11: Dhrystone without Runtime Checks

## B Benchmarks on Windows Systems

### B.1 Windows XP with AMD Athlon 2000+

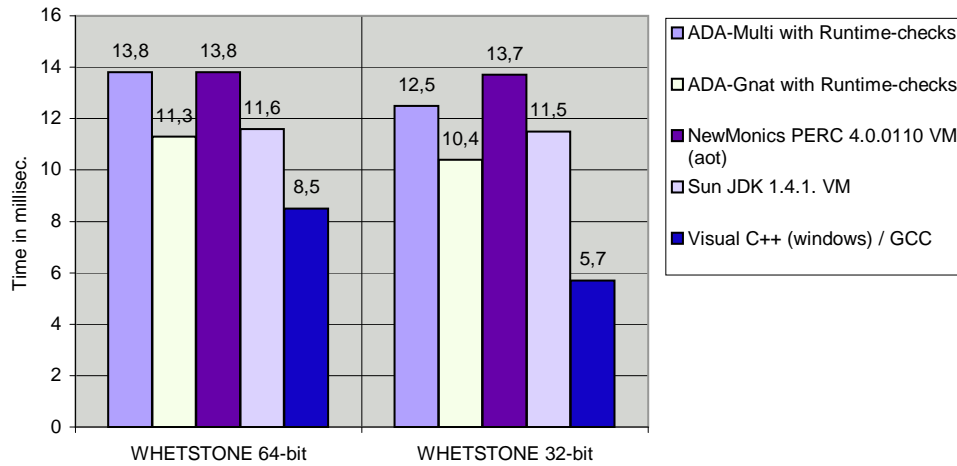


Figure 12: Whetstone 64/32-bit with Runtime Checks

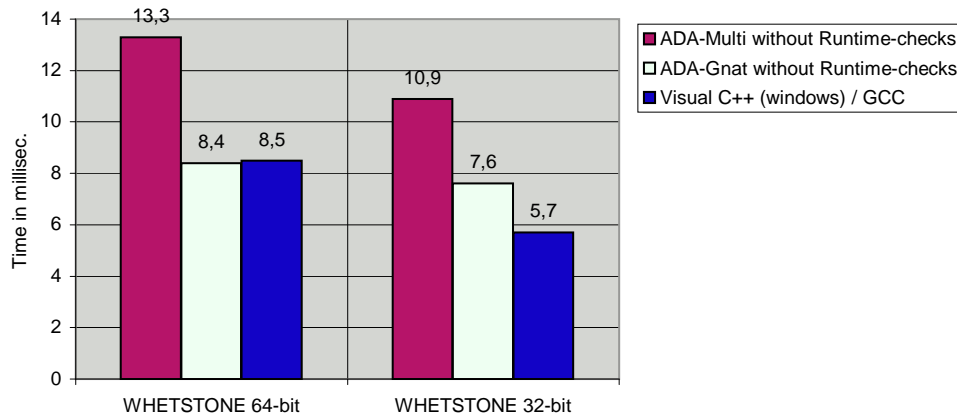


Figure 13: Whetstone 64/32-bit without Runtime Checks

B.1 Windows XP with AMD Athlon 2000+ Benchmarks on Windows Systems

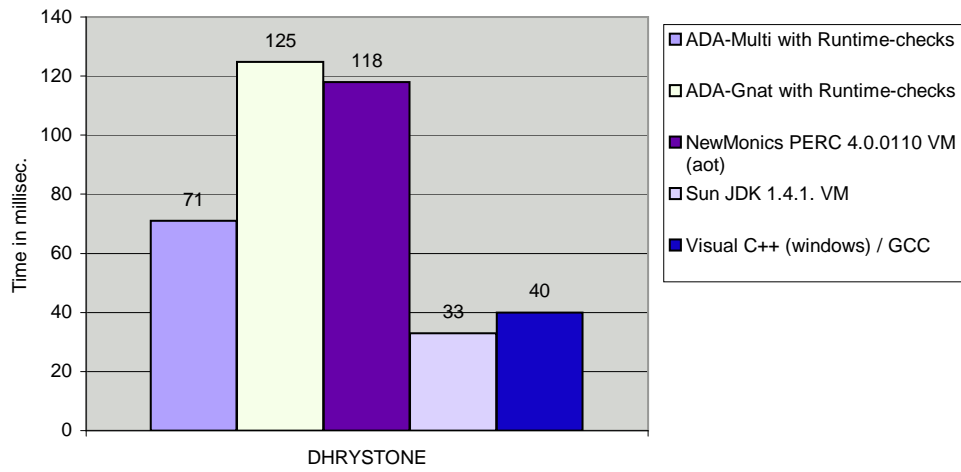


Figure 14: Dhrystone with Runtime Checks

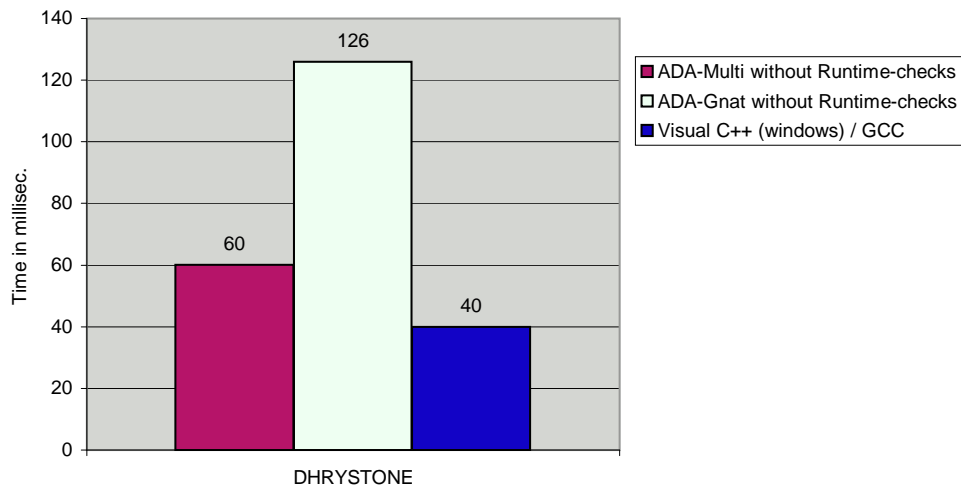


Figure 15: Dhrystone without Runtime Checks

**B.2 Windows 2000 with Intel Pentium III 933MHz**

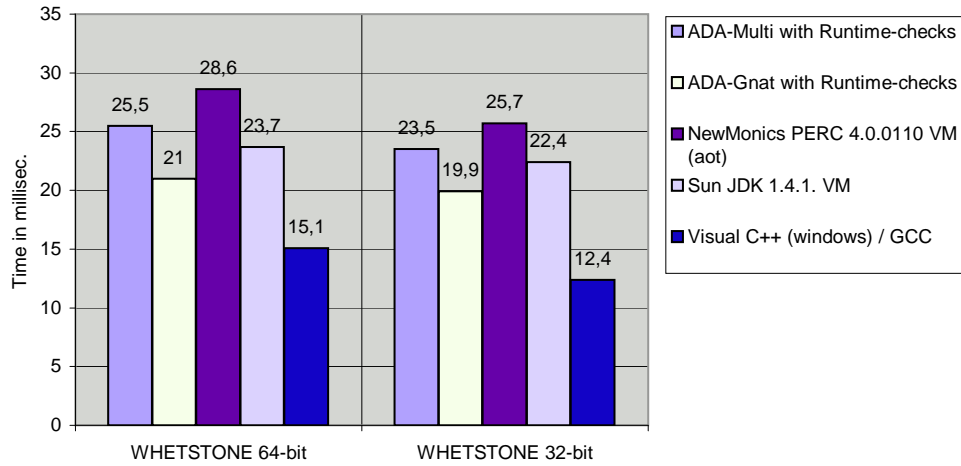


Figure 16: Whetstone 64/32-bit with Runtime Checks

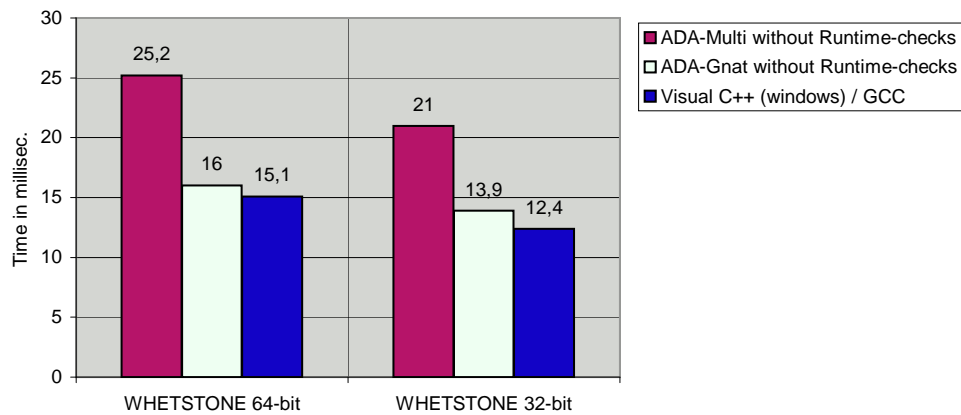


Figure 17: Whetstone 64/32-bit without Runtime Checks

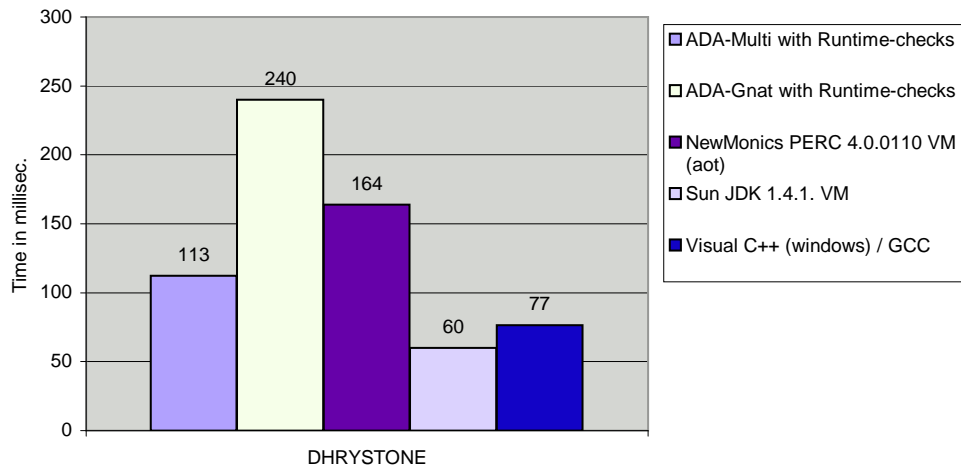


Figure 18: Dhrystone with Runtime Checks

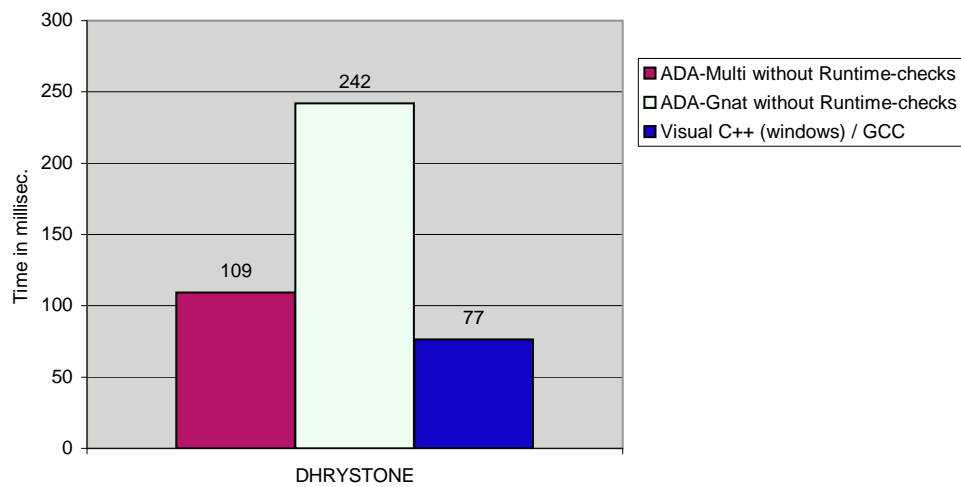


Figure 19: Dhrystone without Runtime Checks

## C Benchmarks on Unix and Linux System

### C.1 Sun Solaris with SPARC Processor

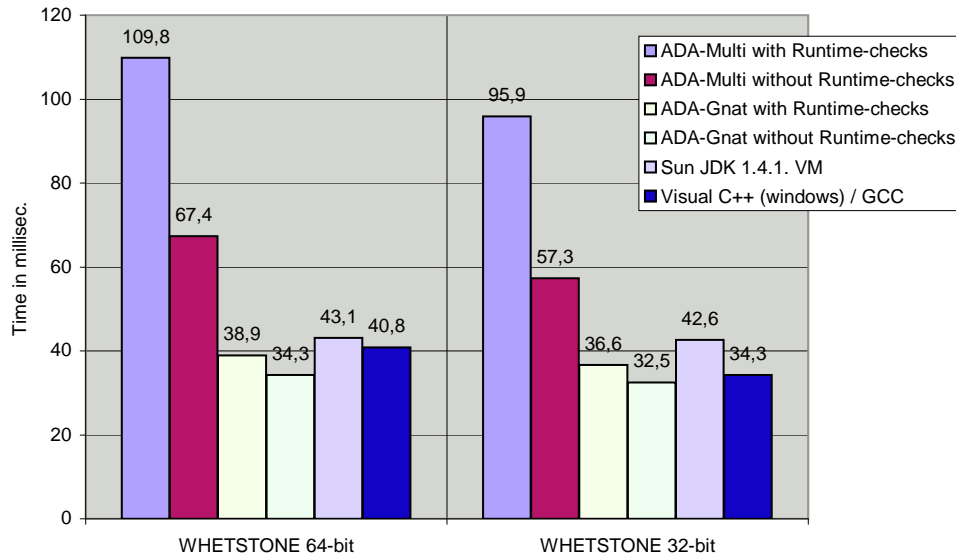


Figure 20: Whetstone 64/32-bit

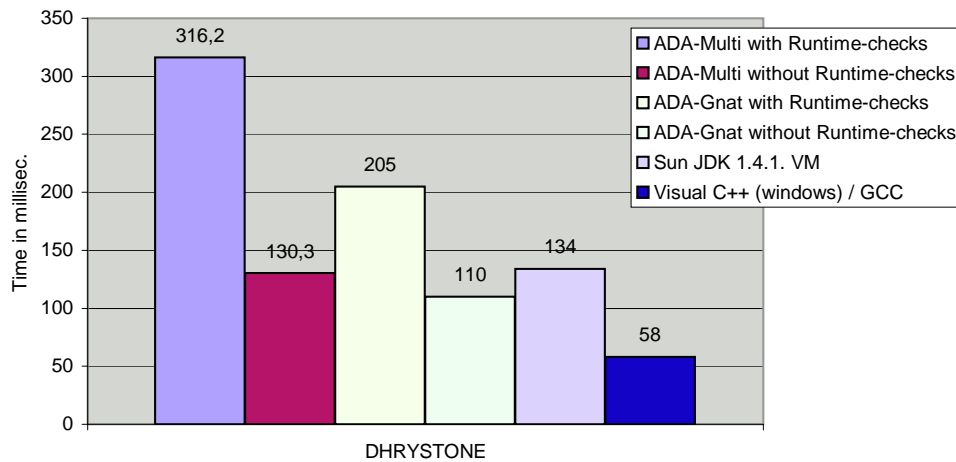


Figure 21: Dhrystone

**C.2 Linux SuSe 8.0 with AMD Athlon 2000+**

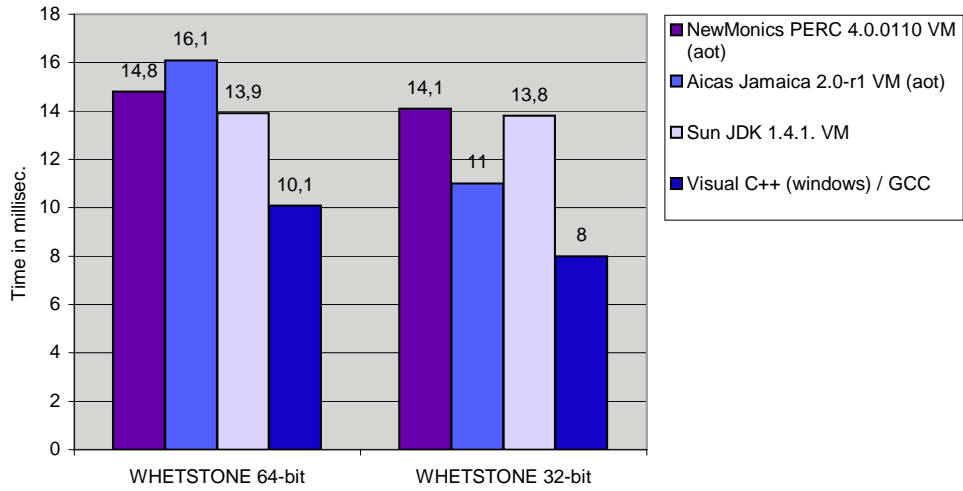


Figure 22: Whetstone 64/32-bit

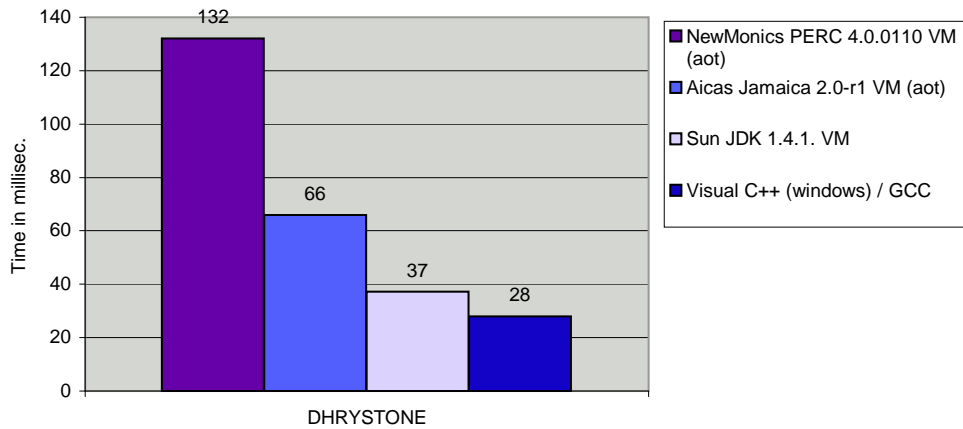


Figure 23: Dhrystone

## D Additional Information

### D.1 Comparison of the File Size

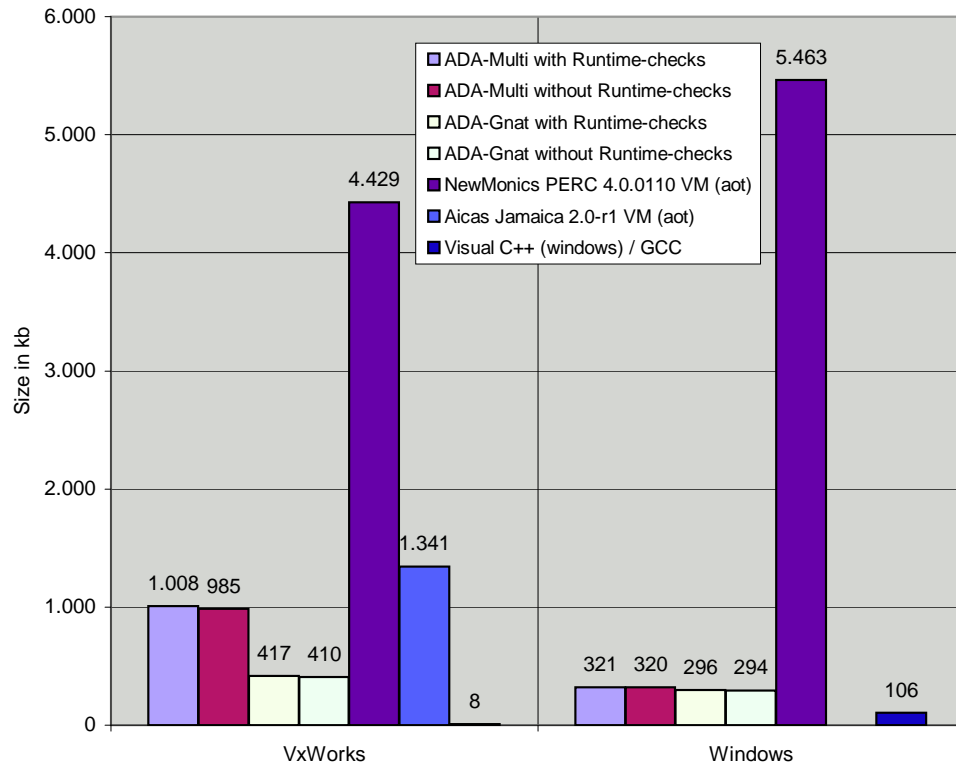


Figure 24: Different Filesizes

## D.2 Comparison PERC AOT vs. JIT

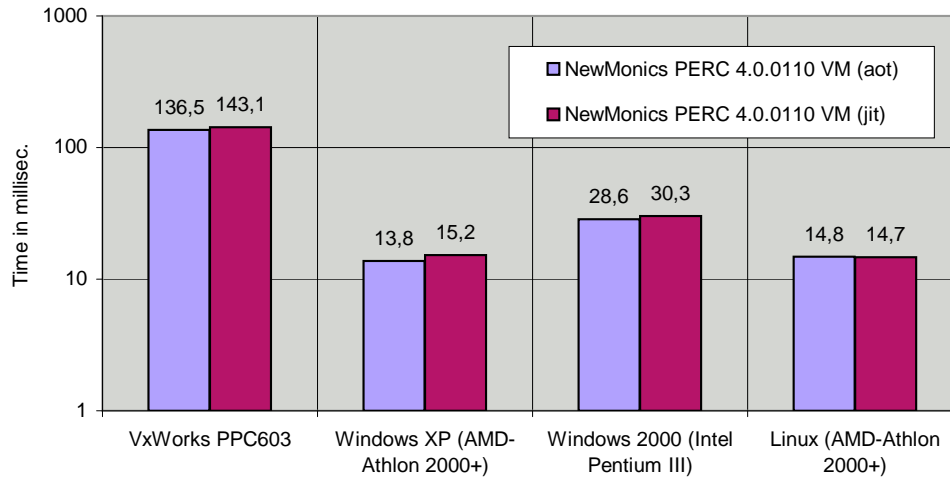


Figure 25: AOT versus JIT

## E Compiler Options

### E.1 NewMonics Perc VM

```
romize
-r debug
-target PPC -os VxWorks
-pni-classes pni_classes
-aot
-hardware-floating-point
-lazy-resolve
whetstone/whet -o image.o
ldppc -o pvm.o -r hooks.o main.o pvmStart.o image.o
```

### E.2 Aicas Jamaica VM

```
jamaica
-compile
-smart
-timeSlice 0
-target vxworks-powerpc
-heapSize 20m -stackSize 150k -numThreads 15 -numDynamicTypes 0
whetstone.whet
```

### E.3 Ada-Multi

In the GUI some options are made different from the default:

- File Options: Optimization: Optimize for Speed
- Language Options: Ada: Runtime checks on/off

### E.4 Ada-GNAT

```
powerpc-wrs-vxworks-gnatmake
-O3
-gnatn
-v
-mlongcall
```

and `-gnatp` without `runtime-checks`

### E.5 C / Visual C++

```
powerpc-wrs-vxworks-gcc
-O2 -nostdinc -nostdlib -ffreestanding
-Wall -U_BIG_ENDIAN -DCPU=PPC603 -mlongcall
-o c/whetstone.o -c c/whetstone.c -DNDEBUG
powerpc-wrs-vxworks-ld
-r -nostdlib -u main -o c/whetstone c/whetstone.o
```

Or for the Linux system:

```
gcc
-Wall -o c/whetstone c/whetstone.c
-I/usr/include -L/usr/lib -lm -O2
```

And for the Windows System with Visual C++ the "Optimization for Speed" was chosen.

## F Java Program Code

### F.1 Whetstone 64-bit

```

package whetstone;

public class whet
{
    static long begin_time,
    end_time,
    total_time;

    /*
     * Whetstone benchmark in C. This program is a translation of the
     * original Algol version in "A Synthetic Benchmark" by H.J. Curnow
     * and B.A. Wichman in Computer Journal, Vol 19 #1, February 1976.
     *
     * Convert into java and automatic iterations by Tarquin Mills from C, by anon from Unix Hensa
     *
     * Used to test compiler optimization and floating point performance.
     *
     * Compile by: cc -O -s -o whet whet.c
     * or:         cc -O -DPOUT -s -o whet whet.c
     * if output is desired.
     */

    static int ITERATIONS; /* ITERATIONS/10 = Millions Whetstone instructions */
    static int numberOfCycles;
    static int cycleNo;

    static double x1, x2, x3, x4, x, y, z[]=new double[1], t, t1, t2;
    static double e1[]= new double[4];
    static int i, j, k, l, n1, n2, n3, n4, n6, n7, n8, n9, n10, n11;

    public static double mainCalc()
    { /* initialize constants */

        /*System.out.println("Start");*/
        t = 0.499975;
        t1 = 0.50025;
        t2 = 2.0;

        /* set values of module weights */

        n1 = 0 * ITERATIONS;
        n2 = 12 * ITERATIONS;
        n3 = 14 * ITERATIONS;
        n4 = 345 * ITERATIONS;
        n6 = 210 * ITERATIONS;
        n7 = 32 * ITERATIONS;
        n8 = 899 * ITERATIONS;
        n9 = 616 * ITERATIONS;
        n10 = 0 * ITERATIONS;
        n11 = 93 * ITERATIONS;

        begin_time = System.currentTimeMillis();

        for (cycleNo=1; cycleNo <= numberOfCycles; cycleNo++) {

            /* MODULE 1: simple identifiers */

            x1 = 1.0;
            x2 = x3 = x4 = -1.0;

            for(i = 1; i <= n1; i += 1) {
                x1 = ( x1 + x2 + x3 - x4 ) * t;
                x2 = ( x1 + x2 - x3 + x4 ) * t; // correction: x2 = ( x1 + x2 - x3 - x4 ) * t;
                x3 = ( x1 - x2 + x3 + x4 ) * t; // correction: x3 = ( x1 - x2 + x3 + x4 ) * t;
                x4 = (-x1 + x2 + x3 + x4 ) * t;
            }

            // if (cycleNo==numberOfCycles) pout(n1, n1, n1, x1, x2, x3, x4);

```

```

/* MODULE 2: array elements */

e1[0] = 1.0;
e1[1] = e1[2] = e1[3] = -1.0;

for (i = 1; i <= n2; i +=1) {
e1[0] = ( e1[0] + e1[1] + e1[2] - e1[3] ) * t;
e1[1] = ( e1[0] + e1[1] - e1[2] + e1[3] ) * t;
e1[2] = ( e1[0] - e1[1] + e1[2] + e1[3] ) * t;
e1[3] = (-e1[0] + e1[1] + e1[2] + e1[3] ) * t;
}

// if (cycleNo==numberOfCycles) pout(n2, n3, n2, e1[0], e1[1], e1[2], e1[3]);

/* MODULE 3: array as parameter */

for (i = 1; i <= n3; i += 1)
pa(e1);

// if (cycleNo==numberOfCycles) pout(n3, n2, n2, e1[0], e1[1], e1[2], e1[3]);

/* MODULE 4: conditional jumps */

j = 1;
for (i = 1; i <= n4; i += 1) {
if (j == 1)
j = 2;
else
j = 3;

if (j > 2)
j = 0;
else
j = 1;

if (j < 1 )
j = 1;
else
j = 0;
}

// if (cycleNo==numberOfCycles) pout(n4, j, j, x1, x2, x3, x4);

/* MODULE 5: omitted */

/* MODULE 6: integer arithmetic */

j = 1;
k = 2;
l = 3;

for (i = 1; i <= n6; i += 1) {
j = j * (k - j) * (1 -k);
k = 1 * k - (1 - j) * k;
l = (1 - k) * (k + j);

e1[l - 2] = j + k + 1; /* C arrays are zero based */
e1[k - 2] = j * k * l;
}

// if (cycleNo==numberOfCycles) pout(n6, j, k, e1[0], e1[1], e1[2], e1[3]);

/* MODULE 7: trig. functions */
x = y = 0.5;

for(i = 1; i <= n7; i +=1) {
x = t * Math.atan(t2*Math.sin(x)*Math.cos(x)/(Math.cos(x+y)+Math.cos(x-y)-1.0));
y = t * Math.atan(t2*Math.sin(y)*Math.cos(y)/(Math.cos(x+y)+Math.cos(x-y)-1.0));
}

// if (cycleNo==numberOfCycles) pout(n7, j, k, x, x, y, y);

```

```

/* MODULE 8: procedure calls */

x = y = z[0] = 1.0;

for (i = 1; i <= n8; i +=1)
    p3(x, y, z);

// if (cycleNo==numberOfCycles) pout(n8, j, k, x, y, z[0], z[0]);

/* MODULE9: array references */

j = 0;
k = 1;
l = 2;

e1[0] = 1.0;
e1[1] = 2.0;
e1[2] = 3.0;

for(i = 1; i <= n9; i++)
    p0();

// if (cycleNo==numberOfCycles) pout(n9, j, k, e1[0], e1[1], e1[2], e1[3]);

/* MODULE10: integer arithmetic */

j = 2;
k = 3;

for(i = 1; i <= n10; i +=1) {
    j = j + k;
    k = j + k;
    j = k - j;
    k = k - j - j;
}

// if (cycleNo==numberOfCycles) pout(n10, j, k, x1, x2, x3, x4);

/* MODULE11: standard functions */

x = 0.75;
for(i = 1; i <= n11; i +=1)
    x = Math.sqrt( Math.exp( Math.log(x) / t1));

// if (cycleNo==numberOfCycles) pout(n11, j, k, x, x, x, x);

} /* for */

end_time = System.currentTimeMillis();

System.out.println(" (time for " +numberOfCycles+ " cycles): "
    +(end_time - begin_time)+ " millisec.");

return (end_time - begin_time);

} /* Main */

public static void pa(double e[])
{
    int j;

    j = 0;
    do {
        e[0] = ( e[0] + e[1] + e[2] - e[3] ) * t;
        e[1] = ( e[0] + e[1] - e[2] + e[3] ) * t;
        e[2] = ( e[0] - e[1] + e[2] + e[3] ) * t;
        e[3] = ( -e[0] + e[1] + e[2] + e[3] ) / t2;
        j += 1;
    } while (j < 6);
}

```

```

}

public static void p3(double x,double y,double z[])
{
x = t * (x + y);
y = t * (x + y);
z[0] = (x + y) /t2;
}

public static void p0()
{
el[j] = el[k];
el[k] = el[l];
el[l] = el[j];
}

// public static void pout(int n, int j, int k, double x1, double x2, double x3, double x4)
// {
// System.out.println(n+"\t"+j+"\t"+k+"\t"+x1+"\t"+x2+"\t"+x3+"\t"+x4);
// }

public static void main(String[] args)
{
System.out.println("WHETSTONE - using 64-bit floating-point data");

ITERATIONS = 100; /* ITERATIONS/10 = Millions Whetstone instructions */
numberOfCycles = 100;
int numberOfRuns = 10;

float elapsedTime = 0;
float meanTime = 0;
float rating = 0;
float meanRating = 0;
int intrating = 0;

for (int runNumber=1; runNumber <= numberOfRuns; runNumber++) {

System.out.print(runNumber+ ". Test");

// Call the Whetstone benchmark procedure
// compute elapsed time
elapsedTime = (float)(mainCalc()/1000);

// sum time in milliseconds per cycle
meanTime = meanTime + (elapsedTime * 1000 / numberOfCycles);

// Calculate the Whetstone rating based on the time for
// the numbers of cycles just executed
rating = (1000 * numberOfCycles) / elapsedTime;

// Sum Whetstone rating
meanRating = meanRating + rating;
intrating = (int)rating;

// Reset no_of_cycles for the next run using ten cycles more
numberOfCycles += 10;
}

meanTime = meanTime/numberOfRuns;
meanRating = meanRating/numberOfRuns;
intrating = (int)meanRating;

System.out.println("Number of Runs " + numberOfRuns);
System.out.println("Average time per cycle " + meanTime + " millisec.");
System.out.println("Average Whetstone Rating " + intrating + " KWIPS");

}

} /* benchmarks */

```

## F.2 Dhrystone

```

package dhrystone;

public class dhry extends GlobalVariables {

    static long numberOfRuns = 10;
    static long numberOfLoops = 100000;

    public static double execute() {

        int        Int_Loc_1,
                  Int_Loc_2,
                  Int_Loc_3;
        int[]      Int_Loc_3_Ref = new int[1];
        int[]      Int_Loc_1_Ref = new int[1];
        char       Char_Index;
        int[]      Enum_Loc = new int[1];
        String     String_Loc_1,
                  String_Loc_2;

        long       begin_time,
                  end_time,
                  total_time;

        int        Run_Index,
                  Meas_Index;

        Next_Record_Glob      = Second_Record;
        Record_Glob           = First_Record;

        Record_Glob.Record_Comp = Next_Record_Glob;
        Record_Glob.Discr       = Ident_1;
        Record_Glob.Enum_Comp   = Ident_3;
        Record_Glob.Int_Comp    = 40;
        Record_Glob.String_Comp = "DHRYSTONE PROGRAM, SOME STRING";
        String_Loc_1            = "DHRYSTONE PROGRAM, 1'ST STRING";

begin_time = System.currentTimeMillis();

        for (Run_Index = 1; Run_Index <= numberOfLoops; ++Run_Index) {

            Proc_5();
            Proc_4();

            Int_Loc_1 = 2;
            Int_Loc_2 = 3;

            String_Loc_2 = "DHRYSTONE PROGRAM, 2'ND STRING";

            Enum_Loc[0] = Ident_2;
            Bool_Glob = !Func_2(String_Loc_1, String_Loc_2);

            while (Int_Loc_1 < Int_Loc_2) {
                Int_Loc_3_Ref[0] = 5 * Int_Loc_1 - Int_Loc_2;
                Proc_7(Int_Loc_1, Int_Loc_2, Int_Loc_3_Ref);
                Int_Loc_1 += 1;
            }

            Int_Loc_3 = Int_Loc_3_Ref[0];
            Proc_8(Array_Glob_1, Array_Glob_2, Int_Loc_1, Int_Loc_3);
            Proc_1(Record_Glob);

            for (Char_Index = 'A'; Char_Index <= Char_Glob_2; ++Char_Index) {
                if (Enum_Loc[0] == Func_1(Char_Index, 'C')) {
                    Proc_6(Ident_1, Enum_Loc);
                }
            }

            Int_Loc_3 = Int_Loc_2 * Int_Loc_1;
            Int_Loc_2 = Int_Loc_3 / Int_Loc_1;
            Int_Loc_2 = 7 * (Int_Loc_3 - Int_Loc_2) - Int_Loc_1;

            Int_Loc_1_Ref[0] = Int_Loc_1;
            Proc_2(Int_Loc_1_Ref);

```

```

        Int_Loc_1 = Int_Loc_1_Ref[0];
    }

    end_time = System.currentTimeMillis();
    total_time = end_time - begin_time;

    System.out.println(" (time for " + numberOfLoops+ "): " +total_time+ " millisec.");

    return (total_time);
}

static void Proc_1(Record_Type Pointer_Par_Val) {

    Record_Type Next_Record = Pointer_Par_Val.Record_Comp;

    Pointer_Par_Val.Record_Comp = Record_Glob;

    Pointer_Par_Val.Int_Comp = 5;

    Next_Record.Int_Comp = Pointer_Par_Val.Int_Comp;
    Next_Record.Record_Comp = Pointer_Par_Val.Record_Comp;
    Proc_3(Next_Record.Record_Comp);

    if (Next_Record.Discr == Ident_1) {
        Next_Record.Int_Comp = 6;
        Int_Ref[0] = Next_Record.Enum_Comp;
        Proc_6(Pointer_Par_Val.Enum_Comp, Int_Ref);
        Next_Record.Enum_Comp = Int_Ref[0];
        Next_Record.Record_Comp = Record_Glob.Record_Comp;
        Int_Ref[0] = Next_Record.Int_Comp;
        Proc_7(Next_Record.Int_Comp, 10, Int_Ref);
        Next_Record.Int_Comp = Int_Ref[0];
    } else
        Pointer_Par_Val = Pointer_Par_Val.Record_Comp;
}

static void Proc_2(int Int_Par_Ref[]) {

    int Int_Loc;
    int Enum_Loc;

    Int_Loc = Int_Par_Ref[0] + 10;
    Enum_Loc = 0;

    do
        if (Char_Glob_1 == 'A') {
            Int_Loc -= 1;
            Int_Par_Ref[0] = Int_Loc - Int_Glob;
            Enum_Loc = Ident_1;
        }
    while (Enum_Loc != Ident_1);
}

static void Proc_3(Record_Type Pointer_Par_Ref) {

    if (Record_Glob != null)
        Pointer_Par_Ref = Record_Glob.Record_Comp;
    else
        Int_Glob = 100;

    Int_Comp_Ref[0] = Record_Glob.Int_Comp;
    Proc_7(10, Int_Glob, Int_Comp_Ref);
    Record_Glob.Int_Comp = Int_Comp_Ref[0];
}

static void Proc_4() {

    boolean Bool_Loc;

    Bool_Loc = Char_Glob_1 == 'A';
    Bool_Loc = Bool_Loc || Bool_Glob;
}

```

```

        Char_Glob_2 = 'B';
    }
    static void Proc_5() {
        Char_Glob_1 = 'A';
        Bool_Glob = false;
    }
    static void Proc_6(int Enum_Par_Val, int Enum_Par_Ref[]) {
        Enum_Par_Ref[0] = Enum_Par_Val;
        if (!Func_3(Enum_Par_Val))
            Enum_Par_Ref[0] = Ident_4;
        switch (Enum_Par_Val) {
        case Ident_1:
            Enum_Par_Ref[0] = Ident_1;
            break;
        case Ident_2:
            if (Int_Glob > 100)
                Enum_Par_Ref[0] = Ident_1;
            else
                Enum_Par_Ref[0] = Ident_4;
            break;
        case Ident_3:
            Enum_Par_Ref[0] = Ident_2;
            break;
        case Ident_4:
            break;
        case Ident_5:
            Enum_Par_Ref[0] = Ident_3;
            break;
        }
    }
    static void Proc_7(int Int_Par_Val1, int Int_Par_Val2, int Int_Par_Ref[]) {
        int Int_Loc;
        Int_Loc      = Int_Par_Val1 + 2;
        Int_Par_Ref[0] = Int_Par_Val2 + Int_Loc;
    }
    static void Proc_8(int[] Array_Par_1_Ref, int[][] Array_Par_2_Ref, int Int_Par_Val_1, int Int_Par_Val_2) {
        int Int_Index,
            Int_Loc;
        Int_Loc = Int_Par_Val_1 + 5;
        Array_Par_1_Ref[Int_Loc] = Int_Par_Val_2;
        Array_Par_1_Ref[Int_Loc+1] = Array_Par_1_Ref[Int_Loc];
        Array_Par_1_Ref[Int_Loc+30] = Int_Loc;
        for (Int_Index = Int_Loc; Int_Index <= Int_Loc+1; ++Int_Index)
            Array_Par_2_Ref[Int_Loc][Int_Index] = Int_Loc;
        Array_Par_2_Ref[Int_Loc][Int_Loc-1] += 1;
        Array_Par_2_Ref[Int_Loc+20][Int_Loc] = Array_Par_1_Ref[Int_Loc];
        Int_Glob = 5;
    }
    static int Func_1(char Char_Par_1_Val, char Char_Par_2_Val) {
        char Char_Loc_1,
            Char_Loc_2;

```

```

Char_Loc_1 = Char_Par_1_Val;
Char_Loc_2 = Char_Loc_1;
if (Char_Loc_2 != Char_Par_2_Val)
return Ident_1;
else
return Ident_2;
}

static boolean Func_2(String String_Par_1_Ref, String String_Par_2_Ref) {

    int Int_Loc;
    char Char_Loc = '\0';

    Int_Loc = 2;
    while (Int_Loc <= 2)
        if (Func_1(String_Par_1_Ref.charAt(Int_Loc), String_Par_2_Ref.charAt(Int_Loc + 1)) == Ident_1) {
            Char_Loc = 'A';
            Int_Loc += 1;
        }
    if (Char_Loc >= 'W' && Char_Loc < 'Z')
        Int_Loc = 7;
    if (Char_Loc == 'X')
        return true;
    else {
        if (String_Par_1_Ref.compareTo(String_Par_2_Ref) > 0) {
            Int_Loc += 7;
            return true;
        } else
            return false;
    }
}

static boolean Func_3(int Enum_Par_Val) {

    int Enum_Loc;

    Enum_Loc = Enum_Par_Val;
    if (Enum_Loc == Ident_3)
        return true;
    else
        return false;
}

    public static void main(String argv[]) {
//Msg.out = System.err;

    int i;
    double mean_time = 0;

    for (i=1; i<= numberOfRuns; i++) {
System.out.print(i+ ". Test");
mean_time = mean_time + execute();
    }

    System.out.println("\nAverage Time over " +numberOfRuns+ " runs: "
        +(mean_time / numberOfRuns)+ " millisec.");
}

}

package dhrystone;
public class DhrystoneConstants {

    public static final int Ident_1    = 0;
    public static final int Ident_2    = 1;
    public static final int Ident_3    = 2;
    public static final int Ident_4    = 3;
    public static final int Ident_5    = 4;
}

package dhrystone;

```

```

public class GlobalVariables extends DhrystoneConstants {

    static Record_Type      Record_Glob,
                           Next_Record_Glob;
    static int              Int_Glob;
    static boolean          Bool_Glob;
    static char             Char_Glob_1,
                           Char_Glob_2;
    static int[]            Array_Glob_1 = new int[128];
    static int[][]          Array_Glob_2 = new int[128][128];
    static Record_Type      First_Record = new Record_Type(),
                           Second_Record = new Record_Type();

    static int[] Int_Comp_Ref = new int[1];
    static int[] Int_Ref = new int[1];
}

package dhrystone;
public class Record_Type {

    Record_Type Record_Comp;
    int         Discr;
    int         Enum_Comp;
    int         Int_Comp;
    String      String_Comp;
    int         Enum_Comp_2;
    String      String_Comp_2;
    char        Char_Comp_1;
    char        Char_Comp_2;
}

```

## G C Program Code

### G.1 Whetstone 64/32-bit

```

/*
 * C Converted Whetstone Double Precision Benchmark
 * Version 1.2 22 March 1998
 *
 * (c) Copyright 1998 Painter Engineering, Inc.
 * All Rights Reserved.
 *
 * Permission is granted to use, duplicate, and
 * publish this text and program as long as it
 * includes this entire comment block and limited
 * rights reference.
 *
 * Converted by Rich Painter, Painter Engineering, Inc. based on the
 * www.netlib.org benchmark/whetstoned version obtained 16 March 1998.
 *
 * A novel approach was used here to keep the look and feel of the
 * FORTRAN version. Altering the FORTRAN-based array indices,
 * starting at element 1, to start at element 0 for C, would require
 * numerous changes, including decrementing the variable indices by 1.
 * Instead, the array E1[] was declared 1 element larger in C. This
 * allows the FORTRAN index range to function without any literal or
 * variable indices changes. The array element E1[0] is simply never
 * used and does not alter the benchmark results.
 *
 * The major FORTRAN comment blocks were retained to minimize
 * differences between versions. Modules N5 and N12, like in the
 * FORTRAN version, have been eliminated here.
 *
 * An optional command-line argument has been provided [-c] to
 * offer continuous repetition of the entire benchmark.
 * An optional argument for setting an alternate LOOP count is also
 * provided. Define PRINTOUT to cause the POUT() function to print
 * outputs at various stages. Final timing measurements should be
 * made with the PRINTOUT undefined.
 *
 * Questions and comments may be directed to the author at
 * r.painter@ieee.org
 */

```

```

*/
/*
C*****
C   Benchmark #2 -- Double Precision Whetstone (A001)
C
C   o This is a REAL*8 version of
C the Whetstone benchmark program.
C
C   o DO-loop semantics are ANSI-66 compatible.
C
C   o Final measurements are to be made with all
C WRITE statements and FORMAT sttements removed.
C
C*****
*/

/* standard C library headers required */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#ifdef VXX
    #include <tickLib.h>
#include <sysLib.h>
#define timeTicks tickGet
#define ticksPerSec sysClkRateGet()
#else
#define timeTicks clock
#define ticksPerSec CLOCKS_PER_SEC
#endif

#ifdef SP
    #define SPDP float
    #define Precision "32-bit"
#else
    #define SPDP double
    #define Precision "64-bit"
#endif

/* the following is optional depending on the timing function used */
#include <time.h>

/* map the FORTRAN math functions, etc. to the C versions */
#define DSIN sin
#define DCOS cos
#define DATAN atan
#define DLOG log
#define DEXP exp
#define DSQRT sqrt
#define IF if

/* function prototypes */
void POUT(long N, long J, long K, double X1, double X2, double X3, double X4);
double whets(int ITERATIONS, int no_of_cycles);
void PA(SPDP E[]);
void P0(void);
void P3(SPDP X, SPDP Y, SPDP *Z);

/*
COMMON T,T1,T2,E1(4),J,K,L
*/

int    J, K, L, N1, N2, N3, N4, N6, N7, N8, N9, N10, N11;
SPDP  X1,X2,X3,X4,X,Y,Z,T,T1,T2;
SPDP  E1[5];

int
main(int argc, char *argv[])
{
/*
C-----
C   Performance in Whetstone KIP's per second is given by

```

```

C
C (100*LOOP*II)/TIME
C
C   where TIME is in seconds.
C-----
*/

int no_of_runs    = 10;
int no_of_cycles  = 100;
int ITERATIONS    = 100;
float mean_time   = 0.0;
float mean_rating = 0.0;

float elapsed_time;
float rating;
int   int_rating;
double time_rec;
int   run_no;

printf("WHETSTONE - using %s floating point data. (C-Code)\n", Precision);

for (run_no = 1; run_no <= no_of_runs; run_no++) {

printf("%d. Test",run_no);

// Call the Whetstone benchmark procedure
time_rec = (whets(ITERATIONS, no_of_cycles) / 1000);

// compute elapsed time
elapsed_time = (float)time_rec;

// sum time in milliseconds per cycle
mean_time    = mean_time + (elapsed_time * 1000.0f) / no_of_cycles;

// Calculate the Whetstone rating based on the time for
// the numbers of cycles just executed
rating       = (1000.0f * no_of_cycles) / elapsed_time;

// Sum Whetstone rating
mean_rating  = mean_rating + rating;
int_rating   = (int)rating;

// Reset no_of_cycles for the next run using ten cycles more
no_of_cycles += 10;
}

// Compute average time in milliseconds per cycle and write
mean_time = mean_time / no_of_runs;

// Calculate average Whetstone rating and write
mean_rating = mean_rating / no_of_runs;
int_rating  = (int)mean_rating;

// OUTPUT THE RESULTS FOR MARKS
printf("\nNumber of Runs          %d\n",no_of_runs);
printf("Average time per cycle      %f millisec.\n",mean_time);
printf("Average Whetstone Rating     %d KWIPS\n",int_rating);
printf("\n");

return(0);
}

double
whets(int ITERATIONS, int no_of_cycles) {

long I;
int cycle_no;

long startsec, finisec;

T = 0.499975;
T1 = 0.50025;

```

```

T2 = 2.0;

N1 = 0;
N2 = 12 * ITERATIONS;
N3 = 14 * ITERATIONS;
N4 = 345 * ITERATIONS;
N6 = 210 * ITERATIONS;
N7 = 32 * ITERATIONS;
N8 = 899 * ITERATIONS;
N9 = 616 * ITERATIONS;
N10 = 0;
N11 = 93 * ITERATIONS;

/*
C
C Start benchmark timing at this point.
C
*/
startsec = timeTicks();

#ifdef PRINTOUT
printf("\n");
#endif

for (cycle_no = 1; cycle_no <= no_of_cycles; cycle_no++) {

/*
C
C Module 1: Simple identifiers
C
*/
X1 = 1.0;
X2 = -1.0;
X3 = -1.0;
X4 = -1.0;

for (I = 1; I <= N1; I++) {
    X1 = (X1 + X2 + X3 - X4) * T;
    X2 = (X1 + X2 - X3 + X4) * T;
    X3 = (X1 - X2 + X3 + X4) * T;
    X4 = (-X1 + X2 + X3 + X4) * T;
}

#ifdef PRINTOUT
IF (cycle_no==no_of_cycles)POUT(N1,N1,N1,X1,X2,X3,X4);
#endif

/*
C
C Module 2: Array elements
C
*/
E1[1] = 1.0;
E1[2] = -1.0;
E1[3] = -1.0;
E1[4] = -1.0;

for (I = 1; I <= N2; I++) {
    E1[1] = ( E1[1] + E1[2] + E1[3] - E1[4]) * T;
    E1[2] = ( E1[1] + E1[2] - E1[3] + E1[4]) * T;
    E1[3] = ( E1[1] - E1[2] + E1[3] + E1[4]) * T;
    E1[4] = (-E1[1] + E1[2] + E1[3] + E1[4]) * T;
}

#ifdef PRINTOUT
IF (cycle_no==no_of_cycles)POUT(N2,N3,N2,E1[1],E1[2],E1[3],E1[4]);
#endif

/*
C
C Module 3: Array as parameter
C
*/
for (I = 1; I <= N3; I++)

```

```

PA(E1);

#ifdef PRINTOUT
IF (cycle_no==no_of_cycles)POUT(N3,N2,N2,E1[1],E1[2],E1[3],E1[4]);
#endif

/*
C
C Module 4: Conditional jumps
C
*/
J = 1;
for (I = 1; I <= N4; I++) {
if (J == 1)
J = 2;
else
J = 3;

if (J > 2)
J = 0;
else
J = 1;

if (J < 1)
J = 1;
else
J = 0;
}

#ifdef PRINTOUT
IF (cycle_no==no_of_cycles)POUT(N4,J,J,X1,X2,X3,X4);
#endif

/*
C
C Module 5: Omitted
C Module 6: Integer arithmetic
C
*/

J = 1;
K = 2;
L = 3;

for (I = 1; I <= N6; I++) {
J = J * (K-J) * (L-K);
K = L * K - (L-J) * K;
L = (L-K) * (K+J);
E1[L-1] = J + K + L;
E1[K-1] = J * K * L;
}

#ifdef PRINTOUT
IF (cycle_no==no_of_cycles)POUT(N6,J,K,E1[1],E1[2],E1[3],E1[4]);
#endif

/*
C
C Module 7: Trigonometric functions
C
*/
X = 0.5;
Y = 0.5;

for (I = 1; I <= N7; I++) {
X = T * DATAN(T2*DSIN(X)*DCOS(X)/(DCOS(X+Y)+DCOS(X-Y)-1.0));
Y = T * DATAN(T2*DSIN(Y)*DCOS(Y)/(DCOS(X+Y)+DCOS(X-Y)-1.0));
}

#ifdef PRINTOUT
IF (cycle_no==no_of_cycles)POUT(N7,J,K,X,X,Y,Y);
#endif

/*
C
C Module 8: Procedure calls

```

```

C
*/
X = 1.0;
Y = 1.0;
Z = 1.0;

for (I = 1; I <= N8; I++)
P3(X,Y,&Z);

#ifdef PRINTOUT
IF (cycle_no==no_of_cycles)POUT(N8,J,K,X,Y,Z,Z);
#endif

/*
C
C Module 9: Array references
C
*/
J = 1;
K = 2;
L = 3;
E1[1] = 1.0;
E1[2] = 2.0;
E1[3] = 3.0;

for (I = 1; I <= N9; I++)
P0();

#ifdef PRINTOUT
IF (cycle_no==no_of_cycles)POUT(N9,J,K,E1[1],E1[2],E1[3],E1[4]);
#endif

/*
C
C Module 10: Integer arithmetic
C
*/
J = 2;
K = 3;

for (I = 1; I <= N10; I++) {
    J = J + K;
    K = J + K;
    J = K - J;
    K = K - J - J;
}

#ifdef PRINTOUT
IF (cycle_no==no_of_cycles)POUT(N10,J,K,X1,X2,X3,X4);
#endif

/*
C
C Module 11: Standard functions
C
*/
X = 0.75;

for (I = 1; I <= N11; I++)
X = DSQRT(DEXP(DLOG(X)/T1));

#ifdef PRINTOUT
IF (cycle_no==no_of_cycles)POUT(N11,J,K,X,X,X,X);
#endif

/*
C
C     THIS IS THE END OF THE MAJOR LOOP.
C
*/

} /* for: end of major loop */

/*
C
C     Stop benchmark timing at this point.

```

```

C
*/
finisec = timeTicks();

printf (" (time for %d cycles): %ld millisec.\n", no_of_cycles, ((finisec-startsec)*1000/ticksPerSec));

return ((finisec-startsec)*1000/ticksPerSec);
}

void
PA(SPDP E[])
{
J = 0;

L10:
E[1] = ( E[1] + E[2] + E[3] - E[4]) * T;
E[2] = ( E[1] + E[2] - E[3] + E[4]) * T;
E[3] = ( E[1] - E[2] + E[3] + E[4]) * T;
E[4] = (-E[1] + E[2] + E[3] + E[4]) / T2;
J += 1;

if (J < 6)
goto L10;
}

void
P0(void)
{
E1[J] = E1[K];
E1[K] = E1[L];
E1[L] = E1[J];
}

void
P3(SPDP X, SPDP Y, SPDP *Z)
{
/* SPDP X1, Y1;
X1 = X;
Y1 = Y;
X1 = T * (X1 + Y1);
Y1 = T * (X1 + Y1);
*Z = (X1 + Y1) / T2; */

X = T * (X + Y);
Y = T * (X + Y);
*Z = (X + Y) / T2;
}

#ifdef PRINTOUT
void
POUT(long N, long J, long K, double X1, double X2, double X3, double X4)
{
printf("%7ld %7ld %7ld %12.4e %12.4e %12.4e %12.4e\n",
N, J, K, X1, X2, X3, X4);
}
#endif
#endif

```

## G.2 Dhrystone

```

/* From: gemini@homxb.UUCP (Rick Richardson)
* Newsgroups: net.sources
* Subject: Dhrystone Version 1.1 Source
* Message-ID: <1369@homxb.UUCP>
* Date: 1 Apr 86 01:58:37 GMT
* Date-Received: 2 Apr 86 22:32:20 GMT
* Organization: PC Research, Inc.
*/

/* EVERYBODY: Please read "APOLOGY" below. -rick 01/06/85
* See introduction in net.arch, or net.micro
*
* "DHRYSTONE" Benchmark Program

```

```

*
* Version:      C/1.1, 12/01/84
*
* Date:        PROGRAM updated 01/06/86, RESULTS updated 03/31/86
*              Timer routines added 26 Sep 1992.
*
* Author:      Reinhold P. Weicker, CACM Vol 27, No 10, 10/84,pg.1013
*              Translated from ADA by Rick Richardson
*              Every method to preserve ADA-likeness has been used,
*              at the expense of C-ness.
*
* Compile:     (1) Registers, NO optimization, 'UNIX' timer option.
*              cc -DUNIX -DROPT dhry11.c -o dhryr
*
*              (2) Registers, optimization, 'UNIX' timer option.
*              cc -DUNIX -DROPT -O dhry11.c -o dhryro
*
*              (3) NO registers, NO optimization, 'UNIX' timer option.
*              cc -DUNIX dhry11.c -o dhry
*
*              (4) NO registers, optimization, 'UNIX' timer option.
*              cc -DUNIX -O dhry11.c -o dhryo
*
* The following program contains statements of a high-level programming
* language (C) in a distribution considered representative:
*
* assignments          53%
* control statements   32%
* procedure, function calls  15%
*
* 100 statements are dynamically executed. The program is balanced with
* respect to the three aspects:
*   - statement type
*   - operand type (for simple data types)
*   - operand access
*     operand global, local, parameter, or constant.
*
* The combination of these three aspects is balanced only approximately.
*
* The program does not compute anything meaningful, but it is
* syntactically and semantically correct.
*
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define TRUE          1
#define FALSE         0

#ifdef VXW
#include <tickLib.h>
#include <sysLib.h>
#define timeTicks tickGet
#define ticksPerSec sysClkRateGet()
#else
#define timeTicks clock
#define ticksPerSec CLOCKS_PER_SEC
#endif

typedef enum          {Ident1, Ident2, Ident3, Ident4, Ident5} Enumeration;
typedef int           OneToThirty;
typedef int           OneToFifty;
typedef char          CapitalLetter;
typedef char          String30[31];
typedef int           Array1Dim[51];
typedef int           Array2Dim[51][51];

const int             number_of_runs = 10;
const long            number_of_loops = 100000;

```

```

struct Record
{
    struct Record      *PtrComp;
    Enumeration        Discr;
    Enumeration        EnumComp;
    OneToFifty         IntComp;
    String30           StringComp;
};

typedef struct Record RecordType;
typedef RecordType *  RecordPtr;
typedef int           boolean;

double dhry, mips;

double Proc0();
void Proc1(RecordPtr PtrParIn);
void Proc2(OneToFifty *IntParIO);
void Proc3(RecordPtr *PtrParOut);
void Proc4();
void Proc5();
void Proc6(Enumeration EnumParIn,
           Enumeration *EnumParOut);
void Proc7(OneToFifty IntParI1,
           OneToFifty IntParI2,
           OneToFifty *IntParOut);
void Proc8(Array1Dim Array1Par,
           Array2Dim Array2Par,
           int IntParI1,
           int IntParI2);

Enumeration Func1(CapitalLetter CharPar1, CapitalLetter CharPar2);
boolean Func2(String30 StrParI1, String30 StrParI2);
boolean Func3(Enumeration EnumParIn);

/*
 * Package 1
 */
int IntGlob;
boolean BoolGlob;
char Char1Glob;
char Char2Glob;
Array1Dim Array1Glob;
Array2Dim Array2Glob;
RecordPtr PtrGlob;
RecordPtr PtrGlobNext;

double Proc0()
{
    OneToFifty IntLoc1;
    OneToFifty IntLoc2;
    OneToFifty IntLoc3;
    char CharIndex;
    Enumeration EnumLoc;
    String30 String1Loc;
    String30 String2Loc;

    long starttime;
    long endtime;
    long i;

    PtrGlobNext = (RecordPtr) malloc(sizeof(RecordType));
    PtrGlob = (RecordPtr) malloc(sizeof(RecordType));
    PtrGlob->PtrComp = PtrGlobNext;
    PtrGlob->Discr = Ident1;
    PtrGlob->EnumComp = Ident3;
    PtrGlob->IntComp = 40;
    strcpy(PtrGlob->StringComp, "DHRYSTONE PROGRAM, SOME STRING");
    strcpy(String1Loc, "DHRYSTONE PROGRAM, 1'ST STRING"); /*GOOF*/

```

```

/*****
-- Start Timer --
*****/

starttime = timeTicks();

for (i = 0; i < number_of_loops; ++i)
{
Proc5();
Proc4();
IntLoc1 = 2;
IntLoc2 = 3;
strcpy(String2Loc, "DHRYSTONE PROGRAM, 2'ND STRING");
EnumLoc = Ident2;
BoolGlob = ! Func2(String1Loc, String2Loc);

while (IntLoc1 < IntLoc2)
{
IntLoc3 = 5 * IntLoc1 - IntLoc2;
Proc7(IntLoc1, IntLoc2, &IntLoc3);
++IntLoc1;
}
Proc8(Array1Glob, Array2Glob, IntLoc1, IntLoc3);
Proc1(PtrGlb);

for (CharIndex = 'A'; CharIndex <= Char2Glob; ++CharIndex)
if (EnumLoc == Func1(CharIndex, 'C'))
Proc6(Ident1, &EnumLoc);
IntLoc3 = IntLoc2 * IntLoc1;
IntLoc2 = IntLoc3 / IntLoc1;
IntLoc2 = 7 * (IntLoc3 - IntLoc2) - IntLoc1;
Proc2(&IntLoc1);
}

/*****
-- Stop Timer --
*****/

endtime = timeTicks();

printf(" (time for %ld number of loops): %f\n", number_of_loops, (float)((endtime-starttime)*1000/ticksPerSec));
return ((endtime-starttime)*1000/ticksPerSec);
}

void Proc1(RecordPtr PtrParIn)
{
#define NextRecord    (*(PtrParIn->PtrComp))

//structassign(NextRecord, *PtrGlb);
NextRecord = *PtrGlb;

PtrParIn->IntComp = 5;
NextRecord.IntComp = PtrParIn->IntComp;
NextRecord.PtrComp = PtrParIn->PtrComp;
Proc3(&NextRecord.PtrComp);

if (NextRecord.Discr == Ident1)
{
NextRecord.IntComp = 6;
Proc6(PtrParIn->EnumComp, &NextRecord.EnumComp);
NextRecord.PtrComp = PtrGlb->PtrComp;
Proc7(NextRecord.IntComp, 10, &NextRecord.IntComp);
}
else
//structassign(*PtrParIn, NextRecord);
*PtrParIn = NextRecord;

#undef NextRecord
}

void Proc2(OneToFifty *IntParIO)

```

```

{
    OneToFifty      IntLoc;
    Enumeration      EnumLoc = 0;

    IntLoc = *IntParIO + 10;
    for(;;)
    {
        if (Char1Glob == 'A')
        {
            --IntLoc;
            *IntParIO = IntLoc - IntGlob;
            EnumLoc = Ident1;
        }
        if (EnumLoc == Ident1)
            break;
    }

    void Proc3(RecordPtr *PtrParOut)
    {
        if (PtrGlb != NULL)
            *PtrParOut = PtrGlb->PtrComp;
        else
            IntGlob = 100;
        Proc7(10, IntGlob, &PtrGlb->IntComp);
    }

    void Proc4()
    {
        boolean      BoolLoc;

        BoolLoc = Char1Glob == 'A';
        BoolLoc |= BoolGlob;
        Char2Glob = 'B';
    }

    void Proc5()
    {
        Char1Glob = 'A';
        BoolGlob = FALSE;
    }

    void Proc6(Enumeration EnumParIn, Enumeration *EnumParOut)
    {
        *EnumParOut = EnumParIn;
        if (! Func3(EnumParIn) )
            *EnumParOut = Ident4;
        switch (EnumParIn)
        {
            case Ident1: *EnumParOut = Ident1; break;
            case Ident2: if (IntGlob > 100) *EnumParOut = Ident1;
                        else *EnumParOut = Ident4;
                        break;
            case Ident3: *EnumParOut = Ident2; break;
            case Ident4: break;
            case Ident5: *EnumParOut = Ident3;
        }
    }

    void Proc7(OneToFifty IntParI1,
               OneToFifty IntParI2,
               OneToFifty *IntParOut)
    {
        OneToFifty IntLoc;

        IntLoc = IntParI1 + 2;
        *IntParOut = IntParI2 + IntLoc;
    }

```

```

}

void Proc8(Array1Dim Array1Par,
           Array2Dim Array2Par,
           int      IntParI1,
           int      IntParI2)
{
    OneToFifty IntLoc;
    OneToFifty IntIndex;

    IntLoc = IntParI1 + 5;
    Array1Par[IntLoc] = IntParI2;
    Array1Par[IntLoc+1] = Array1Par[IntLoc];
    Array1Par[IntLoc+30] = IntLoc;
    for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
        Array2Par[IntLoc][IntIndex] = IntLoc;
    ++Array2Par[IntLoc][IntLoc-1];
    Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];
    IntGlob = 5;
}

Enumeration Func1(CapitalLetter CharPar1, CapitalLetter CharPar2)
{
    CapitalLetter CharLoc1;
    CapitalLetter CharLoc2;

    CharLoc1 = CharPar1;
    CharLoc2 = CharLoc1;
    if (CharLoc2 != CharPar2)
        return (Ident1);
    else
        return (Ident2);
}

boolean Func2(String30 StrParI1, String30 StrParI2)
{
    OneToThirty IntLoc;
    CapitalLetter CharLoc = 0;

    IntLoc = 1;
    while (IntLoc <= 1)
        if (Func1(StrParI1[IntLoc], StrParI2[IntLoc+1]) == Ident1)
            {
                CharLoc = 'A';
                ++IntLoc;
            }
        if (CharLoc >= 'W' && CharLoc <= 'Z')
            IntLoc = 7;
        if (CharLoc == 'X')
            return(TRUE);
        else
            {
                if (strcmp(StrParI1, StrParI2) > 0)
                    {
                        IntLoc += 7;
                        return (TRUE);
                    }
                else
                    return (FALSE);
            }
}

boolean Func3( Enumeration EnumParIn)
{
    Enumeration EnumLoc;

    EnumLoc = EnumParIn;
    if (EnumLoc == Ident3) return (TRUE);
}

```

```

return (FALSE);
}

int main()
{
int i;
double mean_time = 0;

for (i=1; i<= number_of_runs; i++) {
printf("%i. Test",i);
mean_time = mean_time + Proc0();
}

printf("\nAverage Time over %i runs: %f millisec.\n",number_of_runs, (mean_time / number_of_runs));

return(0);
}

```

## H Ada Program Code

### H.1 Whetstone 64-bit

```

-----
-- =                               M A R K S
-- =                               MBB Ada Real-time benchmark Suite
-- =                               -----
-- = Test Identifier                : WHETSTONE_64
-- =
-- = Revision History               :
-- =                               -----
-- = Date          Version         Who          Changes made
-- =
-- = 15-DEC-1988   [1.0]           M. Riedmeier   Created.
-- = 22-Jul-2003   [2.0]           A. Rosskopf    Adapted to Ada95
-- =
-- = Test Objective                 :
-- =                               -----
-- =
-- = Ada version of the Whetstone Benchmark Program using 64-bit
-- = floating-point data and Ada95 predefined elementary functions.
-- = (This MARKS benchmark has been derived from PIWG benchmark A000093.)
-- =
-- = For a description of the ALGOL60 version of the benchmark, see:
-- = "Timing Studies using a Synthetic Whetstone Benchmark",
-- = by Sam Harbaugh and John A. Forakis,
-- = Computer Journal, February 1976, pages 43-49
-- =
-- = Authors Disclaimer:
-- = "The Whetstone measure deals only with the most basic scientific/
-- = computational aspects of the languages and computers and no general
-- = conclusions should be drawn from this work. Application specific
-- = benchmarks should be written and run by anyone needing to draw
-- = conclusions regarding suitability of languages, compilers and
-- = hardware. This data is reported to stimulate interest and work in
-- = run time benchmarking and in no way is meant to influence anyone's
-- = choice of languages or software in any situation."
-- =
-----

with Ada.Text_IO ; use Ada.Text_IO ;           --MARKS_Ada95
with Time_Keeping; use Time_Keeping;          --MARKS
with Marks_Types; use Marks_Types;           --MARKS

with Ada.Numerics.Generic_Elementary_Functions ; --MARKS_Ada95

```

```

-----
procedure WHETSTONE_64 is
-----

-- Important Note: Ada runtime checks will have a large effect on runtime !

--pragma SUPPRESS(ACCESS_CHECK);
--pragma SUPPRESS(DISCRIMINANT_CHECK);
--pragma SUPPRESS(DISCRIMINANT_CHECK);
--pragma SUPPRESS(INDEX_CHECK);
--pragma SUPPRESS(LENGTH_CHECK);
--pragma SUPPRESS(RANGE_CHECK);
--pragma SUPPRESS(DIVISION_CHECK);
--pragma SUPPRESS(OVERFLOW_CHECK);
--pragma SUPPRESS(STORAGE_CHECK);
--pragma SUPPRESS(ELABORATION_CHECK);

-- Elementary floating-point functions (Ada 95) -----

package EF_64 is
    new Ada.Numerics.Generic_Elementary_Functions (Marks_Types.Float_64) ;
use EF_64 ;

-- package MyFloatIO is
--new Float_IO (Marks_Types.FLOAT_32) ;
-- use MyFloatIO;

function SIN (F : FLOAT_64) return FLOAT_64 renames EF_64.Sin ;
function COS (F : FLOAT_64) return FLOAT_64 renames EF_64.Cos ;
-- function ATAN (F : FLOAT_64) return FLOAT_64 renames EF_64.Arctan;
function SQRT (F : FLOAT_64) return FLOAT_64 renames EF_64.Sqrt;
function LOG (F : FLOAT_64) return FLOAT_64 renames EF_64.Log;
function EXP (F : FLOAT_64) return FLOAT_64 renames EF_64.Exp;

function ATAN (F : FLOAT_64) return FLOAT_64 is
begin
    return EF_64.Arctan (Y => F, X => 1.0) ;
end ATAN ;

-- procedure pout (n,j,k: in INT_32; x1,x2,x3,x4 : in FLOAT_64) is
-- begin
--put_line (INT_32'Image(n) & " " & INT_32'Image(j) & " " & INT_32'Image(k) & " " &
-- FLOAT_64'Image(x1) & " " & FLOAT_64'Image(x2) & " " &
-- FLOAT_64'Image(x3) & " " & FLOAT_64'Image(x4));
-- end pout;

-----
procedure WHETSTONE(I, NO_OF_CYCLES : in INT_32;
--o START_TIME, STOP_TIME : out FLOAT) is
Time_Rec : IN OUT T_Time_Rec ) IS --MARKS
-----

-- Calling procedure provides the loop count weight factor, I, and
-- the encompassing loop count, NO_OF_CYCLES.

type VECTOR is array (INT_32 range <>) of FLOAT_64;
X1,X2,X3,X4,X,Y,Z,T,T1,T2 : FLOAT_64;
E1 : VECTOR(1..4);
J,K,L,N1,N2,N3,N4,N5,N6,N7,N8,N9,N10,N11 : INT_32;

procedure PA(E: in out VECTOR) is
-- tests computations with an array as a parameter
J : INT_32;
-- T,T2 : FLOAT are global variables
begin
    J:=0;
    <<LAB>>
    E(1) := (E(1) + E(2) + E(3) - E(4)) * T;
    E(2) := (E(1) + E(2) - E(3) + E(4)) * T;
    E(3) := (E(1) - E(2) + E(3) + E(4)) * T;
    E(4) := (-E(1) + E(2) + E(3) + E(4)) / T2;
    J := J + 1;
    if J < 6 then
        goto LAB;
    end if;
end if;

```

```

end PA;

procedure P0 is
-- tests computations with no parameters
-- T1,T2 : FLOAT are global
-- E1 : VECTOR(1..4) is global
-- J,K,L : INTEGER are global
begin
  E1(J) := E1(K);
  E1(K) := E1(L);
  E1(L) := E1(J);
end P0;

procedure P3(X,Y: in FLOAT_64; Z : out FLOAT_64) is
-- tests computations with simple identifiers as parameters
-- T,T2 : FLOAT are global
xx, yy: float_64;

begin
  XX := T * (X + Y);
  YY := T * (X + Y);
  Z := (XX + YY) / T2;
end P3;

begin
-- Set constants
T := 0.499975;
T1 := 0.50025;
T2 := 2.0;
-- Compute the execution frequency for the benchmark modules
N1 := 0;      --Module 1 not executed
N2 := 12 * I;
N3 := 14 * I;
N4 := 345*I;
N5 := 0;      -- Module 5 not executed
N6 := 210*I;
N7 := 32*I;
N8 := 899*I;
N9 := 616*I;
N10:= 0;     -- Module 10 not executed
N11:= 93*I;

Start_Timing (Time_Rec, No_of_Cycles) ; --MARKS

CYCLE_LOOP:
for CYCLE_NO in 1..NO_OF_CYCLES loop
-- Module 1 : computations with simple identifiers
  X1 := 1.0;
  X2 := -1.0;
  X3 := -1.0;
  X4 := -1.0;
  for I in 1..N1 loop
    X1 := (X1 + X2 + X3 - X4) * T;
    X2 := (X1 + X2 - X3 + X4) * T;
    X3 := (X1 - X2 + X3 + X4) * T;
    X4 := (-X1 + X2 + X3 + X4) * T;
  end loop;
-- end Module 1

--if (cycle_no = no_of_cycles) then pout (n1,n1,n1,x1,x2,x3,x4); end if;

-- Module 2: computations with array elements
  E1(1) := 1.0;
  E1(2) := -1.0;
  E1(3) := -1.0;
  E1(4) := -1.0;
  for I in 1..N2 loop
    E1(1) := (E1(1) + E1(2) + E1(3) - E1(4)) * T;
    E1(2) := (E1(1) + E1(2) - E1(3) + E1(4)) * T;
    E1(3) := (E1(1) - E1(2) + E1(3) + E1(4)) * T;
    E1(4) := (-E1(1) + E1(2) + E1(3) + E1(4)) * T;
  end loop;
-- end Module 2

```

```

--if (cycle_no = no_of_cycles) then pout (n2,n3,n2,E1(1),E1(2),E1(3),E1(4)); end if;

    -- Module 3 : passing an array as a parameter
    for I in 1..N3 loop
        PA(E1);
    end loop;
-- end Module 3

--if (cycle_no = no_of_cycles) then pout (n3,n2,n2,E1(1),E1(2),E1(3),E1(4)); end if;

    -- Module 4 : performing conditional jumps
    J := 1;
    for I in 1..N4 loop
        if J=1 then
            J := 2;
        else
            J := 3;
        end if;
        if J>2 then
            J := 0;
        else
            J := 1;
        end if;
        if J<1 then
            J := 1;
        else
            J := 0;
        end if;
    end loop;
--end Module 4

--if (cycle_no = no_of_cycles) then pout (n4,j,j,x1,x2,x3,x4); end if;

    -- Module 5 : omitted

    -- Module 6 : performing integer arithmetic
    J := 1;
    K := 2;
    L := 3;
    for I in 1..N6 loop
        J := J * (K-J) * (L-K);
        K := L*K - (L-J) * K;
        L := (L-K) * (K+J);
        E1(L-1) := FLOAT_64(J+K+L);
        E1(K-1) := FLOAT_64(J*K*L);
    end loop;
-- end Module 6

--if (cycle_no = no_of_cycles) then pout (n6,j,k,E1(1),E1(2),E1(3),E1(4)); end if;

    -- Module 7 : performing computations using trigonometric
    -- functions
    X := 0.5;
    Y := 0.5;
    for I in 1..N7 loop
        X := T*ATAN(T2*SIN(X)*COS(X)/(COS(X+Y)+COS(X-Y)-1.0));
        Y := T*ATAN(T2*SIN(Y)*COS(Y)/(COS(X+Y)+COS(X-Y)-1.0));
    end loop;
-- end Module 7

--if (cycle_no = no_of_cycles) then pout (n7,j,k,x,x,y,y); end if;

    -- Module 8 : procedure calls with simple identifiers as
    -- parameters
    X := 1.0;
    Y := 1.0;
    Z := 1.0;
    for I in 1..N8 loop
        P3(X,Y,Z);
    end loop;
-- end Module 8

--if (cycle_no = no_of_cycles) then pout (n8,j,k,x,y,z,z); end if;

    -- Module 9 : array reference and procedure calls with no

```

```

--          parameters
J := 1;
K := 2;
L := 3;
E1(1) := 1.0;
E1(2) := 2.0;
E1(3) := 3.0;
for I in 1..N9 loop
  P0;
end loop;
-- end Module 9

--if (cycle_no = no_of_cycles) then put (n9,j,k,E1(1),E1(2),E1(3),E1(4)); end if;

-- Module 10 : integer arithmetic
J := 2;
K := 3;
for I in 1..N10 loop
  J := J + K;
  K := K + J;
  J := K - J;
  K := K - J - J;
end loop;
-- end Module 10

--if (cycle_no = no_of_cycles) then put (n10,j,k,x1,x2,x3,x4); end if;

-- Module 11 : performing computations using standard
-- mathematical functions
X := 0.75;
for I in 1..N11 loop
  X := SQRT(EXP(LOG(X)/T1));
end loop;
-- end Module 11

--if (cycle_no = no_of_cycles) then put (n11,j,k,x,x,x,x); end if;

end loop CYCLE_LOOP;

Stop_Timing (Time_Rec) ;          --MARKS

Put (" (time for " & INT_32'IMAGE(NO_OF_CYCLES) & " cycles):" ) ;
Show_Timing (Time_Rec) ;
new_line ;

end WHETSTONE;

-----
procedure COMPUTE_WHETSTONE_KIPS is
-----
-- Variables used to control execution of benchmark and to
-- compute the Whetstone rating :

NO_OF_RUNS   : INT_32; -- Number of times the benchmark is executed
NO_OF_CYCLES : INT_32; -- Number of times the group of benchmark
-- modules is executed

I           : INT_32;

Time_Rec    : T_Time_Rec ; --MARKS

ELAPSED_TIME : FLOAT_32;
-- Time measured between START_TIMING and STOP_TIMING
MEAN_TIME    : FLOAT_32; -- Average time per cycle
RATING       : FLOAT_32; -- Thousands of Whetstone instructions per sec
MEAN_RATING  : FLOAT_32; -- Average Whetstone rating
INT_RATING   : INT_32; -- Integer value of KWIPS

begin -----

Put_Line ("WHETSTONE - using 64-bit floating-point data. (Ada-Code)");

MEAN_TIME    := 0.0;
MEAN_RATING  := 0.0;
NO_OF_CYCLES := 100;
NO_OF_RUNS   := 10;

```

```

I := 100;

RUN_LOOP:
for RUN_NO in 1..NO_OF_RUNS loop
put(INT_32'IMAGE(RUN_NO) & ". Test");
-- Call the Whetstone benchmark procedure
WHESTSTONE(I,NO_OF_CYCLES,Time_Rec) ; --MARKS

-- Compute elapsed time
ELAPSED_TIME := Float_32(Time_Rec.Measured_Dur) ; --MARKS

-- Sum time in milliseconds per cycle
MEAN_TIME := MEAN_TIME + (ELAPSED_TIME*1_000.0)
              /FLOAT_32(NO_OF_CYCLES);

-- Calculate the Whetstone rating based on the time for
-- the number of cycles just executed
RATING := (1_000.0 * FLOAT_32(NO_OF_CYCLES))/ELAPSED_TIME;

-- Sum Whetstone rating
MEAN_RATING := MEAN_RATING + RATING;
INT_RATING := INT_32(RATING);

-- Reset NO_OF_CYCLES for next run using ten cycles more
NO_OF_CYCLES := NO_OF_CYCLES + 10;

end loop RUN_LOOP;

-- Compute average time in milliseconds per cycle and write
MEAN_TIME := MEAN_TIME/FLOAT_32(NO_OF_RUNS);

-- Calculate average Whetstone rating and write
MEAN_RATING := MEAN_RATING/FLOAT_32(NO_OF_RUNS);
INT_RATING := INT_32(MEAN_RATING);

-- OUTPUT THE RESULTS FOR MARKS
Put_Line ("Number of Runs           "
          & INT_32'IMAGE(NO_OF_RUNS)) ;

Put_Line ("Average time per cycle      "
          & duration'IMAGE(duration(MEAN_TIME)) & " millisec.");

Put_Line ("Average Whetstone Rating      "
          & INT_32'IMAGE(INT_RATING) & " KWIPS");

end COMPUTE_WHESTSTONE_KIPS;

begin
COMPUTE_WHESTSTONE_KIPS;
end WHESTSTONE_64 ;

-----
--=
--= MODULE NAME :   time_keeping
--=
--=
--= VERSION : 1.3
--= AUTHOR  : Alfred Roskopf
--=
--=
--=
--= DESCRIPTION :
--=
--= This package provides as standard way of time keeping for benchmark
--= tests. It offers two procedures to record a start time and a stop
--= time in a user provided "time record".
--=
--= The package uses calendar to read the times and to calculate the
--= corresponding elapsed time.
--=
--=
--= Example :
--=
--= test1_rec : t_time_rec
--=
-----

```

```

--=                                     =
--=         i := 1                       =
--=                                     =
--=         start_timing (test1_rec, no_iterations)
--=                                     =
--=         while i <= no_iterations loop
--=             -- feature to be tested
--=             increment (i)
--=         end loop
--=                                     =
--=         stop_timing (test1_rec)
--=                                     =
=====
with Ada.Text_IO; use Ada.Text_IO;

-----
package body time_keeping is
-----

-----
procedure start_timing (time_rec      : in out t_time_rec ;
                       no_iterations : in   INT_32 := - 1) is
-----
begin
    time_rec.no_iterations := no_iterations ;
    time_rec.start_time    := clock ;
end start_timing ;

-----
procedure stop_timing (time_rec      : in out t_time_rec) is
-----
begin
    time_rec . stop_time := clock ;
    time_rec . measured_dur := time_rec . stop_time -
                               time_rec . start_time ;
    time_rec . single_micro := int_32(1_000_000.0 *
                                     float(time_rec . measured_dur) /
                                     float(time_rec . no_iterations) ) ;
end stop_timing ;

-----
procedure increment (i : in out INT_32) is
-----
begin
    i := i + 1;
end;

-----
procedure show_timing (time_rec      : in out t_time_rec) is
-----
begin
    --put      ( Int_32'Image (Time_Rec.Single_micro) ) ;
    put      ( duration'Image (Time_Rec.measured_dur*1000) & " millisec." ) ;

    end show_timing ;

end time_keeping ;

```

## H.2 Dhrystone

```

/* From: gemini@homxb.UUCP (Rick Richardson)
* Newsgroups: net.sources
* Subject: Dhrystone Version 1.1 Source
* Message-ID: <1369@homxb.UUCP>
* Date: 1 Apr 86 01:58:37 GMT
* Date-Received: 2 Apr 86 22:32:20 GMT
* Organization: PC Research, Inc.
*/

/* EVERYBODY:      Please read "APOLOGY" below. -rick 01/06/85

```

```

*           See introduction in net.arch, or net.micro
*
* "DHRYSTONE" Benchmark Program
*
* Version:      C/1.1, 12/01/84
*
* Date:        PROGRAM updated 01/06/86, RESULTS updated 03/31/86
*              Timer routines added 26 Sep 1992.
*
* Author:      Reinhold P. Weicker, CACM Vol 27, No 10, 10/84,pg.1013
*              Translated from ADA by Rick Richardson
*              Every method to preserve ADA-likeness has been used,
*              at the expense of C-ness.
*
* Compile:     (1) Registers, NO optimization, 'UNIX' timer option.
*              cc -DUNIX -DROPT dhry11.c -o dhryr
*
*              (2) Registers, optimization, 'UNIX' timer option.
*              cc -DUNIX -DROPT -O dhry11.c -o dhryro
*
*              (3) NO registers, NO optimization, 'UNIX' timer option.
*              cc -DUNIX dhry11.c -o dhry
*
*              (4) NO registers, optimization, 'UNIX' timer option.
*              cc -DUNIX -O dhry11.c -o dhryo
*
* The following program contains statements of a high-level programming
* language (C) in a distribution considered representative:
*
* assignments          53%
* control statements   32%
* procedure, function calls  15%
*
* 100 statements are dynamically executed. The program is balanced with
* respect to the three aspects:
*   - statement type
*   - operand type (for simple data types)
*   - operand access
*   operand global, local, parameter, or constant.
*
* The combination of these three aspects is balanced only approximately.
*
* The program does not compute anything meaningful, but it is
* syntactically and semantically correct.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define TRUE          1
#define FALSE         0

#ifdef VXX
    #include <tickLib.h>
#include <sysLib.h>
#define timeTicks tickGet
#define ticksPerSec sysClkRateGet()
#else
#define timeTicks clock
#define ticksPerSec CLOCKS_PER_SEC
#endif

typedef enum    {Ident1, Ident2, Ident3, Ident4, Ident5} Enumeration;
typedef int    OneToThirty;
typedef int    OneToFifty;
typedef char   CapitalLetter;
typedef char   String30[31];
typedef int    Array1Dim[51];
typedef int    Array2Dim[51][51];

const int     number_of_runs = 10;

```

```

const long number_of_loops = 100000;

struct Record
{
struct Record      *PtrComp;
Enumeration        Discr;
Enumeration        EnumComp;
OneToFifty         IntComp;
String30           StringComp;
};

typedef struct Record RecordType;
typedef RecordType * RecordPtr;
typedef int         boolean;

double dhry, mips;

double Proc0();
void Proc1(RecordPtr PtrParIn);
void Proc2(OneToFifty *IntParIO);
void Proc3(RecordPtr *PtrParOut);
void Proc4();
void Proc5();
void Proc6(Enumeration EnumParIn,
Enumeration *EnumParOut);
void Proc7(OneToFifty IntParI1,
OneToFifty IntParI2,
OneToFifty *IntParOut);
void Proc8(Array1Dim Array1Par,
Array2Dim Array2Par,
int IntParI1,
int IntParI2);

Enumeration Func1(CapitalLetter CharPar1, CapitalLetter CharPar2);
boolean Func2(String30 StrParI1, String30 StrParI2);
boolean Func3(Enumeration EnumParIn);

/*
 * Package 1
 */
int IntGlob;
boolean BoolGlob;
char Char1Glob;
char Char2Glob;
Array1Dim Array1Glob;
Array2Dim Array2Glob;
RecordPtr PtrGlob;
RecordPtr PtrGlobNext;

double Proc0()
{
OneToFifty IntLoc1;
OneToFifty IntLoc2;
OneToFifty IntLoc3;
char CharIndex;
Enumeration EnumLoc;
String30 String1Loc;
String30 String2Loc;

long starttime;
long endtime;
long i;

PtrGlobNext = (RecordPtr) malloc(sizeof(RecordType));
PtrGlob = (RecordPtr) malloc(sizeof(RecordType));
PtrGlob->PtrComp = PtrGlobNext;
PtrGlob->Discr = Ident1;
PtrGlob->EnumComp = Ident3;
PtrGlob->IntComp = 40;
strcpy(PtrGlob->StringComp, "DHRYSTONE PROGRAM, SOME STRING");

```

```

strcpy(String1Loc, "DHRYSTONE PROGRAM, 1'ST STRING"); /*GOOF*/

/*****
-- Start Timer --
*****/

starttime = timeTicks();

for (i = 0; i < number_of_loops; ++i)
{
  Proc5();
  Proc4();
  IntLoc1 = 2;
  IntLoc2 = 3;
  strcpy(String2Loc, "DHRYSTONE PROGRAM, 2'ND STRING");
  EnumLoc = Ident2;
  BoolGlob = ! Func2(String1Loc, String2Loc);

  while (IntLoc1 < IntLoc2)
  {
    IntLoc3 = 5 * IntLoc1 - IntLoc2;
    Proc7(IntLoc1, IntLoc2, &IntLoc3);
    ++IntLoc1;
  }
  Proc8(Array1Glob, Array2Glob, IntLoc1, IntLoc3);
  Proc1(PtrGlb);

  for (CharIndex = 'A'; CharIndex <= Char2Glob; ++CharIndex)
  if (EnumLoc == Func1(CharIndex, 'C'))
  Proc6(Ident1, &EnumLoc);
  IntLoc3 = IntLoc2 * IntLoc1;
  IntLoc2 = IntLoc3 / IntLoc1;
  IntLoc2 = 7 * (IntLoc3 - IntLoc2) - IntLoc1;
  Proc2(&IntLoc1);
}

/*****
-- Stop Timer --
*****/

endtime = timeTicks();

printf(" (time for %ld number of loops): %f\n", number_of_loops, (float)((endtime-starttime)*1000/ticksPerSec));
return ((endtime-starttime)*1000/ticksPerSec);
}

void Proc1(RecordPtr PtrParIn)
{
#define NextRecord      (*(PtrParIn->PtrComp))

//structassign(NextRecord, *PtrGlb);
NextRecord = *PtrGlb;

PtrParIn->IntComp = 5;
NextRecord.IntComp = PtrParIn->IntComp;
NextRecord.PtrComp = PtrParIn->PtrComp;
Proc3(&NextRecord.PtrComp);

if (NextRecord.Discr == Ident1)
{
  NextRecord.IntComp = 6;
  Proc6(PtrParIn->EnumComp, &NextRecord.EnumComp);
  NextRecord.PtrComp = PtrGlb->PtrComp;
  Proc7(NextRecord.IntComp, 10, &NextRecord.IntComp);
}
else
//structassign(*PtrParIn, NextRecord);
*PtrParIn = NextRecord;

#undef NextRecord
}

```

```
void Proc2(OneToFifty *IntParIO)
{
  OneToFifty      IntLoc;
  Enumeration      EnumLoc = 0;

  IntLoc = *IntParIO + 10;
  for(;;)
  {
    if (Char1Glob == 'A')
    {
      --IntLoc;
      *IntParIO = IntLoc - IntGlob;
      EnumLoc = Ident1;
    }
    if (EnumLoc == Ident1)
      break;
  }
}

void Proc3(RecordPtr *PtrParOut)
{
  if (PtrGlob != NULL)
    *PtrParOut = PtrGlob->PtrComp;
  else
    IntGlob = 100;
  Proc7(10, IntGlob, &PtrGlob->IntComp);
}

void Proc4()
{
  boolean      BoolLoc;

  BoolLoc = Char1Glob == 'A';
  BoolLoc |= BoolGlob;
  Char2Glob = 'B';
}

void Proc5()
{
  Char1Glob = 'A';
  BoolGlob = FALSE;
}

void Proc6(Enumeration EnumParIn, Enumeration *EnumParOut)
{
  *EnumParOut = EnumParIn;
  if (! Func3(EnumParIn) )
    *EnumParOut = Ident4;
  switch (EnumParIn)
  {
    case Ident1: *EnumParOut = Ident1; break;
    case Ident2: if (IntGlob > 100) *EnumParOut = Ident1;
                 else *EnumParOut = Ident4;
                 break;
    case Ident3: *EnumParOut = Ident2; break;
    case Ident4: break;
    case Ident5: *EnumParOut = Ident3;
  }
}

void Proc7(OneToFifty IntParI1,
           OneToFifty IntParI2,
           OneToFifty *IntParOut)
{
  OneToFifty IntLoc;
```

```

IntLoc = IntParI1 + 2;
*IntParOut = IntParI2 + IntLoc;
}

void Proc8(Array1Dim Array1Par,
          Array2Dim Array2Par,
          int      IntParI1,
          int      IntParI2)

{
  OneToFifty  IntLoc;
  OneToFifty  IntIndex;

  IntLoc = IntParI1 + 5;
  Array1Par[IntLoc] = IntParI2;
  Array1Par[IntLoc+1] = Array1Par[IntLoc];
  Array1Par[IntLoc+30] = IntLoc;
  for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
  Array2Par[IntLoc][IntIndex] = IntLoc;
  ++Array2Par[IntLoc][IntLoc-1];
  Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];
  IntGlob = 5;
}

Enumeration Func1(CapitalLetter CharPar1, CapitalLetter CharPar2)

{
  CapitalLetter  CharLoc1;
  CapitalLetter  CharLoc2;

  CharLoc1 = CharPar1;
  CharLoc2 = CharLoc1;
  if (CharLoc2 != CharPar2)
  return (Ident1);
  else
  return (Ident2);
}

boolean Func2(String30 StrParI1, String30 StrParI2)

{
  OneToThirty    IntLoc;
  CapitalLetter  CharLoc = 0;

  IntLoc = 1;
  while (IntLoc <= 1)
  if (Func1(StrParI1[IntLoc], StrParI2[IntLoc+1]) == Ident1)
  {
    CharLoc = 'A';
    ++IntLoc;
  }
  if (CharLoc >= 'W' && CharLoc <= 'Z')
  IntLoc = 7;
  if (CharLoc == 'X')
  return(TRUE);
  else
  {
    if (strcmp(StrParI1, StrParI2) > 0)
    {
      IntLoc += 7;
      return (TRUE);
    }
    else
    return (FALSE);
  }
}

boolean Func3( Enumeration EnumParIn)

{
  Enumeration EnumLoc;

```

```
EnumLoc = EnumParIn;
if (EnumLoc == Ident3) return (TRUE);
return (FALSE);
}

int main()
{
  int i;
  double mean_time = 0;

  for (i=1; i<= number_of_runs; i++) {
    printf("%i. Test",i);
    mean_time = mean_time + Proc0();
  }

  printf("\nAverage Time over %i runs: %f millisec.\n",number_of_runs, (mean_time / number_of_runs));

  return(0);
}
```

## References

- [CW76] H. J. Curnow and B. A. Wichmann. Whetstone benchmark: A synthetic benchmark. *The Computer Journal, Volume 19, No.1 (Feb. 1976)*, 1976.
- [GHS03] Green Hills Software Inc. GHS. Adamulti - ada 95 integrated development environment. <http://www.ghs.com>, 2003.
- [GNA01] Ada Core Technologies GNAT. Gnat reference manual. <http://gcc.gnu.org>, 2001.
- [LY99] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification*. Addison Wesley, second edition, 1999. <http://java.sun.com>.
- [New02] Inc. NewMonics. Perc virtual machine 4.0 – user manual. <http://www.newmonics.com>, 2002.
- [SW01] Fridtjof Siebert and Andy Walter. Jamaica virtual machine – user documentation. <http://www.aicas.com>, 2001.
- [Wei88] Dr. Reinhold P. Weicker. Dhrystone benchmark: Rationale for version 2 and measurement rules. *SIGPLAN Notices 23,8 (Aug. 1988)*, [49-62], 1988.

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Used Benchmarks</b>	<b>2</b>
2.1	Whetstone . . . . .	2
2.2	Dhrystone . . . . .	2
<b>3</b>	<b>Interpretation of the Test Results</b>	<b>3</b>
3.1	Whetstone on VxWorks . . . . .	3
3.2	Dhrystone on VxWorks . . . . .	4
<b>4</b>	<b>Differences in Operating Systems</b>	<b>5</b>
4.1	Windows . . . . .	5
4.2	Sun Solaris . . . . .	6
4.3	Linux . . . . .	6
<b>5</b>	<b>Additional Information</b>	<b>6</b>
5.1	Filesize . . . . .	6
5.2	AOT versus JIT . . . . .	7
5.3	Reproducibility . . . . .	7
5.4	Curiosities . . . . .	7

<b>6</b>	<b>Conclusion and Future Work</b>	<b>8</b>
<b>A</b>	<b>Benchmarks on VxWorks Systems</b>	<b>9</b>
A.1	VxWorks 5.5 with a PPC603 350MHz . . . . .	9
A.2	VxWorks 5.4 with a PPC750 400MHz . . . . .	11
<b>B</b>	<b>Benchmarks on Windows Systems</b>	<b>13</b>
B.1	Windows XP with AMD Athlon 2000+ . . . . .	13
B.2	Windows 2000 with PIII 933MHz . . . . .	15
<b>C</b>	<b>Benchmarks on Unix and Linux System</b>	<b>17</b>
C.1	Sun Solaris with SPARC Processor . . . . .	17
C.2	Linux with Athlon 2000+ . . . . .	18
<b>D</b>	<b>Additional Information</b>	<b>19</b>
D.1	Comparison of the File Size . . . . .	19
D.2	Comparison PERC AOT vs. JIT . . . . .	20
<b>E</b>	<b>Compiler Options</b>	<b>20</b>
E.1	NewMonics Perc VM . . . . .	20
E.2	Aicas Jamaica VM . . . . .	20
E.3	Ada-Multi . . . . .	21
E.4	Ada-GNAT . . . . .	21
E.5	C / Visual C++ . . . . .	21
<b>F</b>	<b>Java Program Code</b>	<b>22</b>
F.1	Whetstone 64-bit . . . . .	22
F.2	Dhrystone . . . . .	26
<b>G</b>	<b>C Program Code</b>	<b>30</b>
G.1	Whetstone 64/32-bit . . . . .	30
G.2	Dhrystone . . . . .	36
<b>H</b>	<b>Ada Program Code</b>	<b>42</b>
H.1	Whetstone 64-bit . . . . .	42
H.2	Dhrystone . . . . .	48