Uebung: Popular Benchmarks

Sieve of Erathostenes:

- Find Primes
- Depends on:
  - Memory Systems Performance
  - Implementation Array/Bitset
  - Working Set in Memory

Ackermann Function:

- Silly recursive Funktion
- Inverse of x=ln*(y) (log star), a product of infinite log shaving by theoreticians

```
A(1,j)=2^j
A(i,1)=A(i-1,2)
A(i,j)=A(i-1,A(i,j-1))
```

see also:
Cormen, Leiserson, Rivest: Algorithms

4.6 POPULAR BENCHMARKS

In trade presses, the term *benchmark* is almost always used synonymously with workload. Kernels, synthetic programs, and application-level workloads, for example, are all called benchmarks. Although the instruction mixes are a type of workload, they have not been called benchmarks. Some authors have attempted to restrict the term benchmark to refer only to the set of programs taken from real workloads. This distinction, however, has mostly been ignored in the literature. Thus, the process of performance comparison for two or more systems by measurements is called **benchmarking**, and the workloads used in the measurements are called **benchmarks**. Some of the well-known benchmarks are described next.

4.6.1 Sieve

The sieve kernel has been used to compare microprocessors, personal computers, and high-level languages. It is based on Eratosthenes' sieve algorithm and is used to find all prime numbers below a given number n. The algorithm, in its manual form, consists of first writing down all integers from 1 to n and then striking out all multiples of k for $k = 2,3,\ldots,\sqrt{n}$. For example, to find all prime numbers from 1 to 20, the steps are as follows:

1. Write down all numbers from 1 to 20. Mark all as prime:

   [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
   [12] [13] [14] [15] [16] [17] [18] [19] [20]

2. Remove all multiples of 2 from the list of primes:

   [1] [2] [3] 4 [5] 6 [7] 8 [9] 10 [11] 12 [13] 14 [15] 16 [17] 18 [19] 20

3. The next integer in the sequence is 3. Remove all multiples of 3:

   [1] [2] [3] 4 [5] 6 [7] 8 9 10 [11] 12 [13] 14 15 16 [17] 18 [19] 20

4. The next integer in the sequence is 5, which is greater than the square root of 20. Hence, the remaining sequence consists of the desired prime numbers.

A Pascal program to implement the sieve kernel is given in Figure 4.2.

4.6.2 Ackermann's Function

This kernel has been used to assess the efficiency of the procedure-calling mechanism in ALGOL-like languages. The function has two parameters and is defined recursively. The function Ackermann(3,n) is evaluated for values of n from 1 to 6. The average execution time per call, the number of instructions executed per call, and the amount of stack space required for each call are used to compare various systems.

A listing of the benchmark program in SIMULA is shown in Figure 4.3. The value of the function Ackermann(3,n) is $2^{n+3} - 3$. This knowledge is used

in the code to verify the implementation of the benchmark. The number of recursive calls in evaluating Ackermann(3,n) has been shown by Wichmann (1976) to be

$$(512 \times 4^{n-1} - 15 \times 2^{n+3} + 9n + 37)/3$$

This expression is used to compute the execution time per call. For Ackermann(3,n), the maximum depth of the procedure calls is $2^{n+3} - 4$. Hence, the amount of stack space required doubles when n is increased by 1.

```
PROGRAM Prime (OUTPUT);
CONST
  MaxNum = 8191; (* Lists all primes up to MaxNum *)
  NumIterations = 10; (* Repeats procedure NumIterations times *)
VAR
  IsPrime : ARRAY [1..MaxNum] OF BOOLEAN;
  i,k,Iteration : INTEGER; (* Loop indexes *)
  NumPrimes : INTEGER; (* Number of primes found *)
BEGIN
  WRITELN('Using Eratosthenes Sieve to find primes up to ', MaxNum);
  WRITELN('Repeating it ',NumIterations,' times.');
  FOR Iteration := 1 TO NumIterations DO
    BEGIN      (* Initialize all numbers to be prime *)
      FOR i := 1 TO MaxNum DO
        IsPrime[i] := TRUE;
      i := 2;
      WHILE i*i <= MaxNum DO
        BEGIN
          IF IsPrime[i] THEN
            BEGIN (* Mark all multiples of i to be nonprime *)
              k := i + i;
              WHILE k <= MaxNum DO
                BEGIN
                  IsPrime[k] := FALSE;
                  k := k + i;
                END; (* of WHILE  k *)
            END; (* of If IsPrime *)
          i := i + 1;
        END; (* of WHILE i*i *)
    NumPrimes := 0;
    FOR i := 1 TO MaxNum DO (* Count the number of primes *)
      IF IsPrime[i] THEN NumPrimes := NumPrimes + 1;
      WRITELN(NumPrimes, ' primes');
    END; (* of FOR Iterations *)
  (* The following can be added during debugging to list primes. *)
  (* FOR i := 1 TO MaxNum DO IF IsPrime[i] THEN WRITELN(i); *)
END.
```

FIGURE 4.2 Pascal program to implement sieve workload.


```
BEGIN
  INTEGER n;              !Loop index;
  INTEGER j;              !Function value;
  INTEGER num_calls;      !Number of recursive calls;
  INTEGER k;              !Contains 2**(n+3);
  INTEGER k1;             !Contains 4**(n-1);
  REAL t1,t2;             !CPU time values;

  INTEGER PROCEDURE Ackermann(m,n); VALUE m,n; INTEGER m,n;
    Ackermann := IF m=0 THEN n+1
                 ELSE IF n=0 THEN Ackermann(m-1,1)
                      ELSE Ackermann(m-1,Ackermann(m,n-1));
!Main Program;
  k := 16; K1 := 1;     !Initialize k and k1 for n=1;
  FOR n := 1 STEP 1 UNTIL 6 DO
    BEGIN
      t1 := CPUTIME;            !Beginning CPU time;
      j := Ackermann(3,n);      !Compute the function;
      t2 := CPUTIME;            !Ending CPU time;
      IF j <> k-3 THEN OUTTEXT("Wrong Value");
      OUTTEXT("Net CPU Time for Ackermann(3,");
      OUTINT(n,1); OUTTEXT(") is");
      OUTREAL(t2-t1,7,15); OUTIMAGE;
      Num_calls := (512*k1-15*k+9*n+37)/3;
      OUTTEXT("CPU Time per call:");
      OUTREAL((t2-t1)/num_calls,7,15);
      OUTIMAGE;
      k1 := 4*k1;       !Update k1 for the next n;
      k := 2*k;         !Update k for the next n;
    END
END
```

FIGURE 4.3 SIMULA program to implement Ackermann's function.

Assignment 2 / Uebung 2

Chapter 3 of the textbooks describes three methodologies of evaluation....

Problem 1

3.1 What methodology would you choose?

- **a.** To select a personal computer for yourself
- **b.** To select 1000 workstations for your company
- **c.** To compare two spread sheet packages
- **d.** To compare two data-flow architectures, if the answer was required:
  - i. Yesterday
  - ii. Next quarter
  - iii. Next year

Problem 2

3.2 Make a complete list of metrics to compare

- **a.** Two personal computers
- **b.** Two database systems
- **c.** Two disk drives
- **d.** Two window systems

Problem 3

4.2 Implement the Sieve workload in a language of your choice, run it on systems available to you, and report the results.