# CS533
## Modeling and Performance Evaluation of Network and Computer Systems

## Types of Workloads

(Chapter 4)

---

## Types of Workloads

**benchmark** *v. trans.* To subject (a system) to a series of tests In order to obtain prearranged results not available on Competitive systems. – S. Kelly-Bootle, *The Devil's DP Dictionary*

- *Test workload* – denotes any workload used in performance study
- *Real workload* – one observed on a system while being used.
  - cannot be repeated (easily)
  - may not even exist (proposed system)
- *Synthetic workload* – similar characteristics to real workload
  - can be applied in a repeated manner
  - relatively easy to port
- *Benchmark* == Workload
  - Benchmarking is process of comparing 2+ systems with workloads

2

---

## Outline

- Introduction
- Addition instructions
- Instruction mixes
- Kernels
- Synthetic programs
- Application benchmarks

3

---

## Addition Instructions

- Early computers had CPU as most expensive component
- Most frequent operation was addition
- Computer with faster addition instruction performed better
- So, run many addition operations as test workload
- Problem
  - More instructions used
  - Some more complicated than others

4

---

## Instruction Mixes

- Number and complexity of instructions increased
- Could measure instructions individually, but used in different amounts
  - Measure relative frequencies of various instructions on real systems
  - Use as weighting factors to get avg instruction time
  - → *Instruction mixes*
    - Units are
      - Millions of Instructions Per Second (MIPS)
      - Millions of Floating-Point Ops per Sec (MFLOPS)

5

---

## Example: Gibson Instruction Mix

| | | |
|---|---|---|
| 1. | Load and Store | 13.2 |
| 2. | Fixed-Point Add/Sub | 6.1 |
| 3. | Compares | 3.8 |
| 4. | Branches | 16.6 |
| 5. | Float Add/Sub | 6.9 |
| 6. | Float Multiply | 3.8 |
| 7. | Float Divide | 1.5 |
| 8. | Fixed-Point Multiply | 0.6 |
| 9. | Fixed-Point Divide | 0.2 |
| 10. | Shifting | 4.4 |
| 11. | Logical And/Or | 1.6 |
| 12. | Instructions not using regs | 5.3 |
| 13. | Indexing | 18.0 |
| | Total | 100 |

1959, IBM 650 IBM 704

6

## Problems with Instruction Mixes

- In modern systems, instruction time variable depending upon
  - Addressing modes, cache hit rates, pipelining
  - Interference with other devices during processor-memory access
  - Distribution of zeros in multiplier
  - Times a conditional branch is taken
- Mixes do not reflect special hardware such as page table lookups
- Only represents speed of processor
  - Bottleneck may be in other parts of system

7

## Kernels

- Used set of instructions that made up a service provided by processor. A *kernel*.
  - Early on, did not consider I/O so also called a *processing kernel*
- Set of operations for problem
  - Ex: Sieve, Tree Searching, Matrix Inversion
- Some problems such as zeros and branches don't apply
- Problem
  - I/O still not considered

8

## Synthetic Programs

- Add I/O request to test load
- Add control loop so can make request as frequently as needed
- Easy to port, distribute
- Can have measurement data built in

- Still, does not necessarily make representative memory or disk accesses
- Often small, so do not exercise virtual memory

9

## Example of Synthetic Workload Generation Program

```
      DIMENSION Record(500)
!Control parameters:
      Num_Computes=500          !Repeat count for computation
      Num_Reads=35              !Number of records read
      Num_Writes=40             !Number of records written
      Num_Iterations=1000       !Repeat count for the experiment
!Open files:
      OPEN(UNIT=1,NAME='In.dat',TYPE='Old',
     1FORM='Unformatted',ACCESS='Direct')
      OPEN(UNIT=2,NAME='Out.dat',TYPE='New',
     1FORM='unformatted',ACCESS='Direct')
      CALL Get_Time(CPU1,Elapsed1)     !Record starting time

      DO 500 Iteration=1,Num_Iterations
!Perform a number of read I/Os
      DO 100 i=1,Num_Reads
      READ(1'i),Record
100   CONTINUE
!Do computation
      DO 200 j=1,Num_Computes
      DO 200 i=1,500
200   Record(i)=1 + i + i*i + i*i*i
!Perform a number of write I/Os
      DO 300 i=1,Num_Writes
      WRITE(2'i),Record
300   CONTINUE
500   CONTINUE
      CALL Get_Time(CPU2,Elapsed2)     !Get ending time
!Close files:
      CLOSE(UNIT=1)
      CLOSE(UNIT=2)
      CPU_Time=(CPU2-CPU1)/Num_Iterations
      Elapsed_Time=(Elapsed2-Elapsed1)/Num_Iterations
      TYPE *,'CPU time per iteration is ',CPU_Time
      TYPE *,'Elasped time per iteration is ',Elapsed_Time
      STOP
      END
```

Buckholz, 1969

10

## Application Workloads

- For special-purpose system, may be able to run representative applications as measure of performance
  - Ex: airline reservation
  - Ex: banking
- Make use of entire system (I/O, etc).
- Issues may be
  - input parameters
  - multiuser
- Only applicable when specific applications are targeted

11

## Popular Benchmarks: Sieve (1 of 2)

- Sieve of Eratosthenes (finds primes)
- Write down all numbers 1 to *n*
- Strike out multiples of k for k = 2, 3, 5 … sqrt(*n*)
  - In steps of remaining numbers

12

## Popular Benchmarks: Sieve (2 of 2)

1. Write down all numbers from 1 to 20. Mark all as prime:
   $\boxed{1}\ \boxed{2}\ \boxed{3}\ \boxed{4}\ \boxed{5}\ \boxed{6}\ \boxed{7}\ \boxed{8}\ \boxed{9}\ \boxed{10}\ \boxed{11}\ \boxed{12}\ \boxed{13}$
   $\boxed{14}\ \boxed{15}\ \boxed{16}\ \boxed{17}\ \boxed{18}\ \boxed{19}\ \boxed{20}$
2. Remove all multiples of 2 from the list of primes:
   $\boxed{1}\ \boxed{2}\ \boxed{3}\ 4\ \boxed{5}\ 6\ \boxed{7}\ 8\ \boxed{9}\ 10\ \boxed{11}\ 12\ \boxed{13}\ 14$
   $\boxed{15}\ 16\ \boxed{17}\ 18\ \boxed{19}\ 20$
3. The next integer in the sequence is 3. Remove all multiples of 3:
   $\boxed{1}\ \boxed{2}\ \boxed{3}\ 4\ \boxed{5}\ 6\ \boxed{7}\ 8\ 9\ 10\ \boxed{11}\ 12\ \boxed{13}\ 14\ 15$
   $16\ \boxed{17}\ 18\ \boxed{19}\ 20$
4. $5 \geq \sqrt{20} \Rightarrow$ Stop



## Popular Benchmarks: Ackermann's Function (1 of 2)

- Assess efficiency of procedure calling mechanisms
- Ackermann's Function has two parameters, is recursive
  - Benchmark is to call Ackerman(3,*n*) for values of $n = 1$ to 6
- Return value is $2^{n+3}$-3, can be used to verify implementation
- Number of calls:

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

  - Can be used to compute time per call
- Depth is $2^{n+3} - 4$, stack space doubles $n$++



## Popular Benchmarks: Ackermann's Function (2 of 2)

(Simula)

```
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
```



## Popular Benchmarks: Whetstone

- Set of 11 modules designed to match observed frequencies in ALGOL programs
  - Array addressing, arithmetic, subroutine calls, parameter passing
  - Ported to Fortran, most popular in C, …
- Many variations of Whetstone, so take care when comparing results
- Problems – specific kernel
  - only valid for small, scientific (floating) apps that fit in cache
  - Does not exercise I/O



## Popular Benchmarks: LINPACK

- Programs that solve dense systems of linear equations
  - Many float adds and multiplies
  - Core is Basic Linear Algebra Subprograms (BLAS), called repeatedly
- Usually, solve 100x100 system of equations
- Represents mechanical engineering applications on workstations
  - Drafting to finite element analysis
  - High computation speed and good graphics processing



## Popular Benchmarks: Dhrystone

- Pun on Whetstone
- Intent to represent systems programming environments
- Most common was in C, but many versions
- Low nesting depth and instructions in each call
- Large amount of time copying strings
- Mostly integer performance with no float operations

## Popular Benchmarks: Lawrence Livermore Loops

- 24 vectorizable, scientific tests
- Floating point operations
  - Physics and chemistry apps have found 40-60% floating point operations
- Relevant for: fluid dynamics, airplane design, weather modeling

19

## Popular Benchmarks: Debit-Credit

- Was Defacto Standard for Transaction Processing Systems
- Retail bank wanted 1000 branches, 10k tellers, 10000k accounts online with peak load of 100 TPS
- Performance in TPS where 95% of all transactions with 1 second or less of response time (arrival of last bit, sending of first bit)
- Now, Transaction Processing Council (TPC) has made more precise benchmarks
  - TPC-A, TPC-B, TCP-C

20

## Popular Benchmarks: SPEC

- Systems Performance Evaluation Cooperative (SPEC) (http://www.spec.org)
  - Non-profit, leading computer vendors
  - Suite of benchmarks
- CPU2000: CPUINT and CPUFP
  - Making CPU2004
- Graphics
- Systems and Applications:
  - Web, Java Client-Server, Network Files System, Mail
- Results database
- Performance compared to baseline machine

21