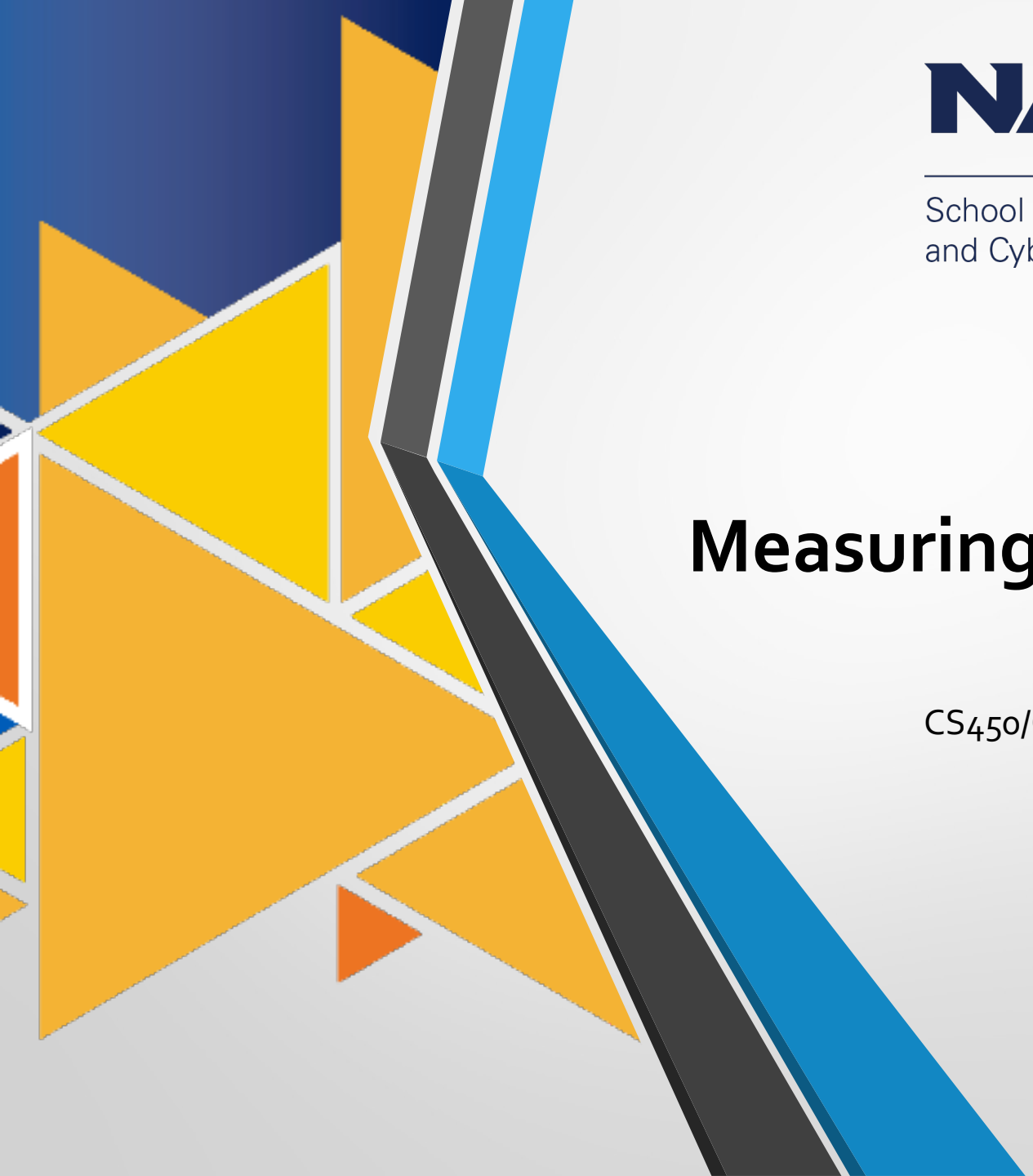

NORTHERN ARIZONA UNIVERSITY

School of Informatics, Computing, and Cyber Systems

Measuring the program performance

CS450/CS550: Parallel Programming

Week 6

Computer performance

- Computer performance is the amount of useful work accomplished by a computer system
- Computer performance is estimated in terms of accuracy, efficiency and speed of executing computer program instructions
- Performance is often measured by an elapsed time (running time, wall-clock time, response time, latency, execution time, etc.); the response time is usually stated together with the processor used for running a program (e.g., "The program takes 2.5 seconds on an Intel i9 2.8GHz")
- The processors' performance may be measured by the clock rate in GHz or the Millions of Instructions Per Second (MIPS); however, not all clock cycles are equal across processor families
- The generic rate used to measure the computer performance is the number of Floating-Point Operations Per Second (FLOPS)

The `time` command in Unix

- The `time` command determines the duration of execution of a particular command
- The `time` command can also display the system resource usage of the process:
  - Real-time (real): the real-life time to run from start to finish (includes any time taken by other processes and the time spent waiting for them to be complete)
  - User time (user): the amount of CPU time spent in user mode during the process (other processes and blocked time are not included)
  - System time (sys): the total CPU time spent in kernel mode during the process (other processes and time spent blocked by other processes are not counted)

The `time` function in C

- The C library function `time` returns the time since the Epoch (00:00:00 UTC, January 1, 1970), measured in seconds
- Function syntax:
```
time_t time(time_t *t);
```
- If `t` is not `NULL`, the return value is also stored in variable `t`:
```
time_t t;
time(&t);
```
- To run the `time` function, the `time.h` header should be included

The `gettimeofday` function in C

- The `gettimeofday` function allows getting the time and a time zone
- Function syntax:

```
int gettimeofday(
struct timeval *tv, struct timezone *tz);
```

- The `tv` argument is a following structure:

```
struct timeval {
  time_t tv_sec; /* seconds */
  suseconds_t tv_usec; /* microseconds */};
```

- The use of the `timezone` structure is obsolete; the `tz` argument should normally be specified as `NULL`
- To run the `gettimeofday` function, the `sys/time.h` header should be included

Estimating the number of measurements

The necessary sample size for a random variable with a normal distribution:

$$n_p = \left\lceil \frac{u_\alpha^2 \cdot \sigma^2}{\varepsilon^2} \right\rceil$$

- α is a significance level ($\alpha = 0.05$)
- u_α is the value of a random variable with standard normal distribution ($N: 0, 1$), such that $\mathrm{Prob}\{-u_\alpha < U < u_\alpha\} = 1 - \alpha$
- ε is a permissible maximum error of the estimated mean value: $\varepsilon = \mu \cdot \alpha$
- μ and σ are the parameters of the normally distributed variable (expected value and standard deviation)

Measuring performance rate

- How to measure a generic performance rate:
  - Time a section of code
  - Count how many operations are in the section of code
  - Compute the rate as the number of operations divided by the measured time
- Example:

```
time_start()
for (int i = 0; i < 1000000; i++)
    x = y * z + a
time_stop()
```

  - Number of operations = $2 * 10^6$ (1 mln additions, 1 mln multiplications)
  - Performance rate = ($2 * 10^6$ / time) FLOPS

Profiler

- A profiler is a tool that monitors the execution of a program and reports the amount of time spent in different functions
- Profiling cycle:
  - Compile the code with the profiler
  - Run the code
  - Identify the most expensive function
  - Optimize that function (call it less often, make it faster, …)
  - Repeat to find and optimize other expensive functions
- Most languages have profilers
- UNIX has a profiler for C/C++ called `gprof`

The `gprof` tool

- `gprof` is a performance analysis tool
- `gprof` was originally written by a group led by Susan Graham at the University of California Berkeley at the beginning of 1980's
- The first results on using `gprof` were published in 1982: https://docs-archive.freebsd.org/44doc/psd/18.gprof/paper.pdf
- Another implementation was written as part of the GNU project in 1988 by Jay Fenlason
- The manual on GNU `gprof` is available at https://ftp.gnu.org/old-gnu/Manuals/gprof-2.9.1/html_mono/gprof.html

Profiling with `gprof`

- Compile and link the program with profiling enabled:

```
gcc -pg test_profile.c -o test_profile
```

- Execute the program to generate a profile data file:

```
./test_profile
```

- Run `gprof` to analyze the profile data (eventually – redirect results to a file):

```
gprof test_profile [> profile_output.txt]
```

Forms of output in `gprof`

- **Flat profile**: shows how much time your program spent in each function, and how many times that function was called
- **Call graph**: shows, for each function, which functions called it, which other functions it called, and how many times; there is also an estimate of how much time was spent in the subroutines of each function
- **Annotated source listing**: a copy of the program's source code, labeled with the number of times each line of the program was executed

Call graph

- A call graph (a call multigraph) is a control-flow graph, which represents calling relationships between subroutines in a computer program
- Each node in a call graph represents a procedure and each edge (f, g) indicates that procedure f calls procedure g
- A cycle in a call graph indicates recursive procedure call

Reading the call graph in `gprof`

- The dashed lines divide the call graph table into entries, one for each function
- In each entry, the primary line is the one that starts with an index number in square brackets. The end of this line shows which function the entry is for
- The preceding lines in the entry describe the callers of the entry function (parents) and the following lines describe its subroutines (children)
- The entries are sorted by time spent in the function and its subroutines

Call graph for `test_profile.c`