Lab Project #4: Cachelab or Malloclab

Hugh C. Lauer

Department of Computer Science

(Slides shamelessly adapted from Bryant & O’Hallaron)

Lab Project #4

- Do *either*
  - Cachelab — a cache simulator and optimized array transpose program

    OR
  - Malloclab — implementation of a high-performance malloc() package

- **Due Sunday, May 4, 2014, 11:59 PM**
  - No Grace Day available!
    - Due to grading deadlines

Cachelab

- **Two parts:–**

- **Cache simulator**
  - Interprets memory access traces from **`valgrind`**
  - 200-300 lines of *C* code

- **Array transpose function**
  - Minimize number of cache misses & evictions
  - Performance counts!
  - **`valgrind`** traces performance on cache simulator

Cache simulator

- **Four arguments:**
  - s:– 2^s is number of sets
  - E:– associativity; 2^E is the number of lines per set
  - b:– 2^b is number of bytes per cache line

  - t:– tracefile (output from **`valgrind`**) of memory references

Cache simulator (continued)

- **Read sequence of memory traces from trace file**
- **Ignore instruction fetches!**
- **Pass all data accesses (load, store, modify) thru your simulated cache**
  - Record hits, misses, evictions
- **Print summary of all cache activity …**
  - … using provided **`PrintSummary()`** function

Follow Programming Rules in project description

Matrix transpose

- **Write a *fast* matrix transpose function …**
- **… that is *cache aware***
- **Test cases:–**
  - 32 × 32
  - 64 × 64
  - 61 × 67
- **Score based on number of *misses!***
  - Fewer misses is better
  - Too many misses ⇒ zero points for that case!

Use autograder to check your score!

Questions?

See project description on course web-site for details, handout

Malloclab

- Implement your own version of `malloc()`, `realloc()`, and `free()`
- Test against traces of memory allocation calls
- Optimize for
  - Spatial efficiency
  - Throughput

Competing demands!

Getting Malloclab

- Download
  - `malloclab-handout.tar`
  - `Malloclab-traces.zip`
- Links in project description document

OR

- Go to course web site, navigate to Projects, select from *Handouts* column

Study

- **Kernighan & Ritchie, §8.7**
  - A good, simple workhorse version of **`malloc()`** and **`free()`**
- **Bryant & O'Hallaron, §9.9**
  - Lots of techniques about fast, efficient implementations of **`malloc()`** and **`free()`**
- **No lectures on this subject this term!**

Two-person Teams

- **You may *optionally* work in teams of two**
- **Register your team with cs2011-staff@cs.wpi.edu so that we may enter it into Turnin**
- **Remember to register early!**
  - We may not be watching e-mail when project is nearly due!

Working on Malloclab

- **Modify *only* mm.c**

- **Implement in mm.c**
  - **`mm-malloc()`**
  - **`mm-free()`**
  - **`mm-realloc()`**

- **Test using `./mdriver` and traces**
  - Correctness
  - Space utilization
  - Throughput

Working on Malloclab (continued)

- Compare your performance with `libc` version of `malloc()`
- Evaluation by autograder
- Autograder parameters set to award ½ of performance points to K&R algorithm

Submission

- **Rename `mm.c` → `userID-mm.c`**
- **Submit to Turnin**
  - Project *Malloclab*
- **Include README file describing**
  - Principal data structures
  - Algorithms for **`malloc()`**, **`free()`**, etc.
  - How you coalesce adjacent free nodes
    - If you do!
  - Other issues

Questions?