Code Optimization

Orion Sky Lawlor

olawlor@uiuc.edu

2003/9/17
Roadmap

- Introduction
- gprof
- Timer calls
- Understanding Performance
Introduction

- **Scientific Performance Method:**
  - **Measure (don’t assume!):**
    - Find the bottlenecks in the code
      - They aren’t where you expect!
    - Fix the worst problems first
    - Consider stopping-- is it good enough?
  - **Fix**
    - Improve algorithms first
    - Improve implementations second
  - **Repeat (indefinitely)**
gprof—UNIX performance tool

- Compile and link with "-pg" flag
  - Adds instrumentation to code
- Run serial program normally
  - Instrumentation runs automatically
- Run gprof to analyze trace
  - `gprof pgm gmon.out`
- Shows a function-level view of execution time
- Heisenberg measurement error!
Timer Calls

- CPU time (virtual time)
  - Time spent running your code
  - CmiCpuTimer() – 10ms resolution

- Wall clock time (real time)
  - Includes OS interference, network delays, context-switching overhead
  - CmiWallTimer() – to 1ns resolution
  - This is what really counts
How to call the timer: one time

- What’s wrong with this?
  - double s=CmiWallTimer();
  - foo();
  - double e=CmiWallTimer()-s;
  - CkPrintf(“foo took %fs\n”,e);

- If CmiWallTimer takes 100ns, and foo takes 50ns, this may print 150ns!
  - Only a problem for very fast functions (or slow timers!)
How to call the timer: repeat

- Repetition can decrease apparent timer overhead and increase resolution:
  - const int n=1000;
  - double s=CmiWallTimer();
  - for (i=0;i<n;i++) foo();
  - double e=(CmiWallTimer()-s)/n;
  - CkPrintf(“foo took %fs\n”,e);

- Problem: what if foo’s performance is cache-sensitive?
Understanding Performance: CPU

GHz 1ns

- Integer Arithmetic
- Floating Point
- Branches
- Cache

10ns

- / or %
- (int)x
- / Subroutine

100ns

- Bitshifts and masks
- *(int *)&x

“inline”

MHz 1us

- Cache-friendly algorithms

10us

100us

KHz 1ms
Understanding Performance: OS

- Timer
- sin, tan, ...
- Malloc
- Syscall
- Tables, identities
- Reuse buffers
- Avoid
- Disk
- Timeslice
Understanding Performance: Net

GHz 1ns

10ns

100ns

MHz 1us

10us

100us

KHz 1ms

Message

Barrier

Rethink

Probe

RDMA
Conclusions

- Performance is the whole point of parallel programming
  - painful but necessary
- Asymptotics matter—find $O(n)$
- Constants matter—count mallocs
- Scientific Performance Method:
  - Measure
  - Fix
  - Repeat