Early Adopter - Knox College

4 year liberal arts college

3 faculty, approximately 25 majors (sophomores−seniors)

Incorporation of many parallel topics in required core courses, normally taken during students' sophomore and junior years. Some topics deferred until specialized upper-level elective.

For more details:

`http://cs.knox.edu/parallel.html`

{dbunde, jdooley, jspacco}@knox.edu

Our major

Intro courses: (required)

Intro, Data structures, Discrete math

Core courses: (required)

Information management

Computer organization & assembly language

Operating systems & networking

Algorithm design & analysis

Software development & professional practice

Electives: (choose 3)

Parallel Prog., HCI, Networking, AI, Crypto, Software Eng., Databases, Automata & Prog. lang., OS, Graphics, etc

Support courses: (choose 2)

other math, symbolic logic, cognitive psych

(adoption-related courses in **bold**)

Computer org. & assembly lang.

(Systems; taught W '11; 7 students)

Binary numbers: unsigned ints, signed ints, floating point, arithmetic ($\approx$ 3.5 70-minute meetings)

Assembly language (MIPS): conditionals, loops, memory layout, functions & recursion, polling & interrupts, RISC/CISC ($\approx$ 15.5 meetings)

Architecture: circuit design, data paths (including pipelining, hazards, and idea of out-of-order), caching, virtual memory ($\approx$ 10 meetings)

Parallel systems: Flynn's classification, multiprocessor topologies, GPUs & CUDA, Roadrunner as example of large system ($\approx$ 4 meetings)

Computer Org.: Multiprocessor topologies

Derive diameter ($\approx$ latency), bisection bandwidth, and # links ($) for path, star, clique, 2D mesh/torus, hypercube, balanced tree

Shows both pure math $\left(\binom{n}{2}$ links in clique$\right)$ and engineering tradeoffs

Exam question:

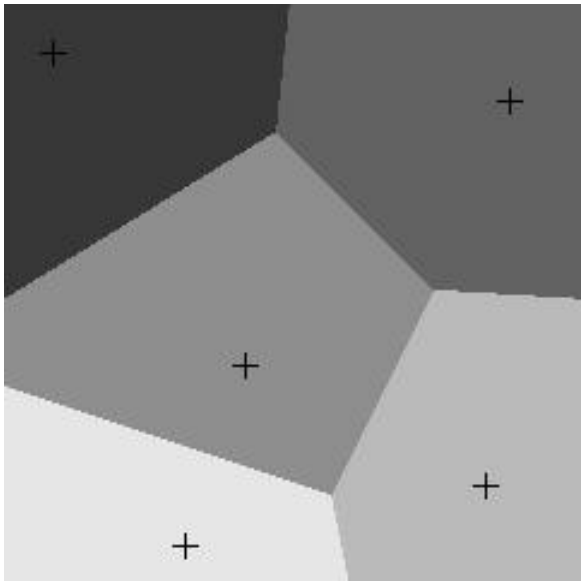
- What are the diameter and bisection bandwidth of 3D mesh with n nodes?
- What significance do these two quantities have for system performance?

Hard question (?!); most had difficulty expressing 1st part.
Nearly all got gist of 2nd, but some students not specific enough

Computer Org.: CUDA assignment

Students given discrete Voronoi code

(written by [Christian Trefftz and Greg Wolffe])



Each thread does a pixel, finding closest "seed" and setting color (actually #)

Task: Draw rectangle given positions of opposite corners (other figs possible)

Embarassingly parallel; the "Hello World" of CUDA programming

Illustrates SIMD, heterogeneity, explicit memory management

Final had toy image-creation problem; most students succeeded

Computer Org.: Discussion of Roadrunner

Read and discussed SC'08 paper introducing Roadrunner (6,120 nodes, each with attached Cell processor)

Emphasis on programming models for heterogeneous system

Introduction to high-perf. systems (#7 on Top 500) and scientific applications

Reading this paper showed students benchmarks, performance measures, pipelining, and scaling in the "real world"

Computer Org.: Student comments

CUDA:

- Some students wanted "horse race" to see that it was faster
  - (might require problem w/ more computation)
- Some uncomfortable w/o knowing C

Paper discussion:

- Exposed students to new ideas (made it challenging)
- Some previous students have liked this aspect of course
- Will need to find fresh paper (or guest lecture) periodically

Several students said parallel material wasn't well integrated

- May work better with more on intraprocessor parallelism
- (Intended to do more, but ran out of preparation time)

Comments about Architecture topics

Message passing (topologies, diameter, latency, bandwidth)

- In curriculum: C/K, 0.9–3.5 hours
- I spent $\approx$ 1.75 hours w/ goal of C
- Felt about right, but they struggled w/ exam question

Data and control hazards

- In curriculum: N
- I spent $\approx$ 1.5 hours w/ goal of K/C
- Some coverage seems necessary w/ pipelining

Spec mark

- In curriculum: K, 0.5–1 hours
- Had planned to add this, but didn't find materials in time
- (Appears in Null & Lobur, but not Patterson & Hennessey)
- Refer to specific source material to facilitate adoption

Plan for operating sys. & networking

(Systems; in progress; 16 students)

"Systems programming" with OS and networking topics

Learn C: Pointers, strings, manual memory management (≈ 7 70-minute meetings)

OS: Process life cycle and scheduling, threads, synchronization, deadlock, file systems (≈ 11 meetings)

Networking: Protocol stacks and reference models, socket programming, IP, TCP, UDP, checksums and error correction, HTTP, DNS, SMTP, MIME, RPC (≈ 11 meetings)

Parallel programming: Speedup, thread pools, shared memory libraries, loop parallelism (≈ 6 meetings)

Integration plan

After teaching C and sockets, look at "database" server

- Begin w/ threads as concept to manage complexity
- Motivate synchronization w/ conflicting accesses
- Cover classic synchronization problems and deadlock

Use hand-managed threads to achieve parallel speedup

Introduce parallel library (probably OpenMP, but considering Cilk)

Do simple message-passing application with sockets and then introduce RPC

Algorithm design and analysis (DS/A; planned W '12)

Plan to discuss parallel aspects of divide & conquer / recursion

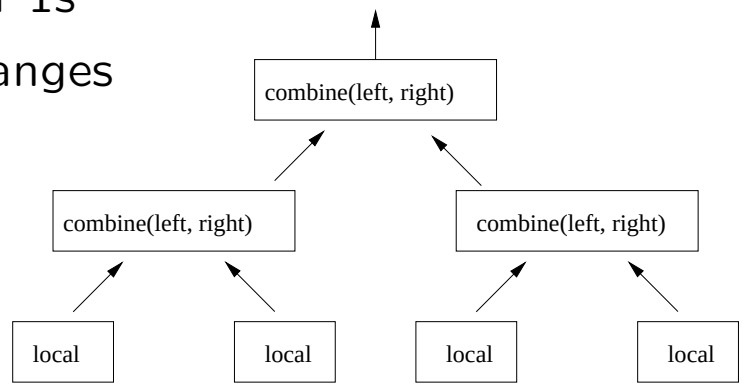
Use reduction and scan as examples for dynamic programming

Reduce ex.: find size of array's longest run of 1s

Procs arranged in tree w/ responsibility for ranges

Report lengths to parent:

- longest run starting at beginning of range,
- longest run overall,
- longest run ending at end of range



Software develop. & prof. practice (planned W '12)

Plan to add parallel "patterns" to design part of course:

fork/join, master/worker, thread pools, scans, reductions

Tools to address correctness and performance issues

Profilers, race condition detection

Required moving user interface design material to another course

More advanced/specific material for upper-level electives

AI:

Parallel search (beam search, genetic algorithms)

Automata theory & Prog. lang.:

Parallel languages (functional, CSP-based, etc)

Operating systems:

Thread management, deeper look at concurrency, transaction support

Networking & distributed systems:

Distributed file systems, MapReduce/Hadoop, Grid, Cloud

Deviation: emphasis on non-scientific examples

Curriculum has 5 hours "specialized computation" on matrix ops
(matrix product, transposition, convolution, linear systems)

These are well-studied examples w/ interesting communication patterns and important applications

But

1. Less motivating to my students
   (Liberal arts students w/o heavy science emphasis)

2. Don't seem representative of non-scientific code
   (I don't implement these now; why will I need to in parallel?)

Don't have replacement to suggest (yet), but like images & text processing