Short modules for introducing parallel concepts

David Bunde
Knox College

Work partially supported by NSF DUE-1044299. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.
Why introduce parallelism?

- It’s here
- CC 2013 says you should
- Students want to see it
Module-based approach

• It’s hard to revise curriculum or entire course, but relatively easy to carve out a couple of days
• Modules are self-contained 2-3 day units that fit within existing courses
• Include course materials and background support
The modules

• Mandelbrot set with OpenMP

• Short exercises with CUDA

• Chapel in Algorithms

Materials available:
http://faculty.knox.edu/dbunde/teaching/CCSC-MW13
Note on “tutorial”
My context

• Dept has 3 FTEs, 27 majors (soph-senior)
• Trimester calendar
  – Students take 3 classes a term, we teach 2
  – Cover ~1 semester of material into 10 weeks
  – 70-minute periods; MWF lecture, Th lab
• Classes with 10-20
• Mac labs, Linux servers
Module 1
Mandelbrot set with OpenMP
Overview

• Built around program that generates Mandelbrot set as .bmp file

• OpenMP
  – threading library built into most C compilers

• Used several ways as part of discussion of threads and concurrency in OS course
Setting all the pixels

for (int i = 0; i < numCols; i++) {
  for (int j = 0; j < numRows; j++) {
    x = ((double)i / numCols -0.5) * 2;
    y = ((double)j / numRows -0.5) * 2;
    color = mandelbrot(x,y);

    pixels[i][j].rgbtBlue = pixels[i][j].rgbtGreen =
    pixels[i][j].rgbtRed = color;
  }
}
}
OpenMP

• Old standard (1\textsuperscript{st} in 1997), but still widely used
• Implemented as pragmas in C and Fortran
• Widely supported (gcc, Visual Studio, Intel, ...)
  – requires –fopenmp flag in gcc
Parallel for loop

#pragma omp parallel for
for(int i=1; i<=100; i++) ...

Prior code

Iterations 1–25

Iterations 26–50

Iterations 51–75

Iterations 76–100

Subsequent code
Applying parallel for

```c
#pragma omp parallel for
for (int i = 0; i < numCols; i++) {
    for (int j = 0; j < numRows; j++) {
        x = ((double)i / numCols -0.5) * 2;
        y = ((double)j / numRows -0.5) * 2;
        color = mandelbrot(x,y);

        pixels[i][j].rgbtBlue = pixels[i][j].rgbtGreen =
        pixels[i][j].rgbtRed = color;
    }
}
```
Resulting output (closeup)
Privatizing local variables

#pragma omp parallel for private(x,y,color)
for (int i = 0; i < numCols; i++) {
    for (int j = 0; j < numRows; j++) {
        x = ((double)i / numCols -0.5) * 2;
        y = ((double)j / numRows -0.5) * 2;
        color = mandelbrot(x,y);

        pixels[i][j].rgbtBlue = pixels[i][j].rgbtGreen =
        pixels[i][j].rgbtRed = color;
    }
}

How well does it parallelize?

Original (serial) running time: 2.39 seconds
Parallel running time: 1.43 seconds

Speedup = \( \frac{\text{Serial time}}{\text{Parallel time}} \) = 1.67

(On my Macbook Pro, with Intel Core i5 processor)
Parallelizing inner loop

#include <omp.h>

#pragma omp parallel for private(x,y,color)
for (int i = 0; i < numCols; i++) {
    for (int j = 0; j < numRows; j++) {
        ...
    }
}

#pragma omp parallel for private(x,y,color)
for (int i = 0; i < numCols; i++) {
    for (int j = 0; j < numRows; j++) {
        ...
    }
}
Parallelizing inner loop

```c
#pragma omp parallel for private(x,y,color)
for (int i = 0; i < numCols; i++) {
    for (int j = 0; j < numRows; j++) {
        ...
    }
}
```

```c
for (int i = 0; i < numCols; i++) {
    #pragma omp parallel for private(x,y,color)
    for (int j = 0; j < numRows; j++) {
        ...
    }
}
```

Time: 1.43 sec

Time: 1.35 sec
Inside mandelbrot function

double mandelbrot(double x, double y) {
    int maxIteration = 1000; int iteration = 0;

    double re = 0, im = 0;
    while((re*re + im*im <= 4) && (iteration < maxIteration)) {
        double temp = re*re - im*im + x;
        im = 2*re*im + y;
        re = temp;
        iteration++;
    }

    if(iteration != maxIteration) return 255; else return 0;
}
Inside mandelbrot function

double mandelbrot(double x, double y) {
  int maxIteration = 1000; int iteration = 0;

double re = 0, im = 0;
while((re*re + im*im <= 4) && (iteration < maxIteration)) {
  double temp = re*re - im*im + x;
  im = 2*re*im + y;
  re = temp;
  iteration++;
}
if(iteration != maxIteration) return 255; else return 0;
}
Swapping loop order

#pragma omp parallel for private(x,y,color)
for (int j = 0; j < numRows; j++) {
    for (int i = 0; i < numCols; i++) {
        Time: 1.35 sec
Dynamic scheduling

#include <omp.h>

#pragma omp parallel for schedule(dynamic)
for (int i = 0; i < numCols; i++) {
    for (int j = 0; j < numRows; j++) {
        ...
    }
}

Time: 0.98 sec
Summary of versions

- Serial version 2.39 sec
- Incorrect parallel version (race)
- Parallel outer loop 1.43 sec
- Parallel inner loop 1.35 sec
- Swap loop order 1.26 sec
- Dynamic scheduling 0.98 sec
Alternative: Pthread library

- Can do (most of) lesson using POSIX-standard threads (pthreads)

- Not easy to do dynamic scheduling

```c
void* func(void* arg) {
    ...
}
```
Classroom hints

• Can’t have too many students sharing same machine

• Go over concepts before and/or after showing code
How I’ve used it

• Previous lecture introducing threads
• Lab using pthreads (Mandelbrot or other example)
• Lecture on lab and using Mandelbrot (OpenMP) to illustrate concepts
  – Definite improvement over doing same material with Pthreads in lecture
OpenMP or Pthreads first?

• OpenMP first
  – Give high-level concepts before lots of syntax
  – Want to spend most of time on concepts so do it first

• Pthreads first
  – Demonstrate execution model before showing “magic”
  – Could use other examples for simplicity
“TODO” list

• Which order for Pthreads vs. OpenMP?
  – Join my experiment!
• More colorful versions of Mandelbrot
• Interactive image generation
• Other examples

Please share!
Module 2
Short exercises with CUDA

Part of Bunde, Karavanic, Mache, Mitchell, “Adding GPU computing to Computer Organization courses”, EduPar 2013
What is CUDA?

• “Compute Unified Device Architecture”

• NVIDIA’s architecture and language for general-purpose programming on graphics cards

• Really a library and extension of C (and other languages)
Why CUDA?

• Easy to get the hardware
  – My laptop came with a 48-core card
  – Department has 448-core card (< $600)
  – NVIDIA willing to donate equipment

• Exciting for students
  – They have cards and want to use them
  – Easy to see performance benefits
Game of Life (GoL)

- Simulation with cells updating in lock step
- Each turn, count living neighbors
- Cell alive next turn if
  - alive this time and have 2 living neighbors, or
  - have 3 living neighbors
Module constraints

• Brief time: Course has lots of other goals
  – One 70-minute lab and parts of 2 lectures

• Relatively inexperienced students
  – Some just out of CS 2
  – Many didn’t know C or Unix programming
Unit goals

- Idea of parallelism
- Benefits and costs of system heterogeneity
- Data movement and NUMA
- Generally, the effect of architecture on program performance
Approach taken

• Introductory lecture
  – GPUs: massively parallel, outside CPU, kernels, SIMD
• Lab illustrating features of CUDA architecture
  – Data transfer time
  – Thread divergence
  – Memory types (next time)
• “Lessons learned” lecture
  – Reiterate architecture
  – Demonstrate speedup with Game of Life
  – Talk about use in Top 500 systems
CUDA programming model

- Device has many cores, organized into groups
- 32-thread warps execute the same instruction
Data transfer

//allocate memory on the device:
cudaMalloc((void**) &a_dev, N*sizeof(int));
...

//transfer array a to GPU
cudaMemcpy(a_dev, a, N*sizeof(int), cudaMemcpyHostToDevice);
...

//transfer array res back from GPU:
cudaMemcpy(res, res_dev, N*sizeof(int), cudaMemcpyDeviceToHost);

direction indicator
Invoking the kernel

```c
int threads = 512; // # threads per block
int blocks = (N+threads−1)/threads; // # blocks (N/threads rounded up)
kernel<<blocks,threads>>>(res_dev, a_dev, b_dev);
```

- Blocks are an organizational unit for threads
- Performance is very dependent on #blocks and #threads
- One rule: #threads should be multiple of 32
__global__ void kernel(int* res, int* a, int* b) {
    //function that runs on GPU to do the addition
    //sets res[i] = a[i] + b[i]; each thread is responsible for one value of i
    int thread_id = threadIdx.x + blockIdx.x*blockDim.x;
    if(thread_id < N) {
        res[thread_id] = a[thread_id] + b[thread_id];
    }
}

since #threads potentially > array size
Lab activity 1: Data transfer time

- Students compare running time of:
  - working CUDA program to add pair of vectors
  - program with data transfer, but no arithmetic
  - program that does arithmetic and only 1 direction of data transfer
Lab activity 1: Data transfer time

• Students compare running time of
  – working CUDA program to add pair of vectors
  – program with data transfer, but no arithmetic
  – program that does arithmetic and only 1 direction of data transfer

• Observe that data transfer is bulk of the time
Lab activity 2: Thread divergence

- Compare two apparently equivalent kernels:

```c
__global__ void kernel_1(int *a) {
    int tid = threadIdx.x;
    int cell = tid % 32;
    a[cell]++;
}
```

```c
__global__ void kernel_2(int *a) {
    int cell = threadIdx.x % 32;
    switch(cell) {
    case 0: a[0]++; break;
    case 1: a[1]++; break;
    ...  //continues to case 7
    default: a[cell]++;
    }
}
```
Lab activity 2: Thread divergence

• Compare two apparently equivalent kernels:

```c
__global__ void kernel_1(int *a) {
    int tid = threadIdx.x;
    int cell = tid % 32;
    a[cell]++;
}
```

```c
__global__ void kernel_2(int *a) {
    int cell = threadIdx.x % 32;
    switch(cell) {
    case 0: a[0]++; break;
    case 1: a[1]++; break;
    ...  //continues to case 7
    default: a[cell]++;
    }
}
```

• Observe vastly different running times
  – Threads in a warp devote time to 1 instruction per clock cycle *even if not all run it* (others nop)
Lab activity 3: Memory types

Based on Chap 6 of [Sanders and Kandrot, “CUDA by example”, 2011]

• “Ray tracing” that tests intersections with array of objects in the same order

• Speeds up with switch to constant memory
  – values are transmitted to entire half warp
  – allows caching
Lab activity 3: Memory types

Based on Chap 6 of [Sanders and Kandrot, “CUDA by example”, 2011]

- “Ray tracing” that tests intersections with array of objects in the same order
- Speeds up with switch to constant memory
  - values are transmitted to entire half warp
  - allows caching
- Performance is worse if threads access objects in different orders
Survey results: Good news

• Asked to describe CPU/GPU interaction:
  – 9 of 11 mention both data movement and invoking kernel
  – Another just mentions invoking the kernel
Survey results: Good news

• Asked to describe CPU/GPU interaction:
  – 9 of 11 mention both data movement and invoking kernel
  – Another just mentions invoking the kernel

• Asked to explain experiment illustrating data movement cost:
  – 9 of 12 say comparing computation and communication cost
  – 2 more talk about comparing different operations
Survey results: Not so good news

• Asked to explain experiment illustrating thread divergence:
  – 2 of 9 were correct
  – 2 more seemed to understand, but misused terminology
  – 3 more remembered performance effect, but said nothing about the cause
Conclusions

• Unit was mostly successful, but thread divergence is a harder concept

• Students interested in CUDA and about half the class requested more of it

• Bottom line: A brief introduction is possible even to students with limited background
Classroom hints

• Need graphics card on local machine (at least for GoL)

• For my unit, show GoL before doing the lab
Alternate models

• Lewis and Clark, Portland State
  – Lecture introducing CUDA
  – Lab/HW using it to speed up Game of Life

• Daniel Ernst
  – Longer unit with both OpenMP and CUDA
  – General emphasis on tuning data layout and access pattern
“TODO” list

• New example for types of memory
• Explain thread divergence better
• Middle ground: adding programming to mine or conceptual material to L&C version
• Porting code to other base languages (Java)
• Other programming example (?)

Please share!
Module 3a
Chapel in Algorithms

(Based on experiences of Kyle Burke and our joint tutorial at SC Ed Program, 2012)
What is Chapel?

- Parallel programming language developed with programmer productivity in mind
- Originally Cray’s project under DARPA’s High Productivity Computing Systems program
- Suitable for shared- or distributed memory systems
- Installs easily on Linux and Mac OS; use Cygwin to install on Windows
Why Chapel?

• Flexible syntax; only need to teach features that you need

• Provides high-level operations

• Designed with parallelism in mind
Flexible syntax

• Supports scripting-like programs:
  writeln(“Hello World!”);

• Also provides objects and modules
Provides high-level operations

• Reductions and scans (more later)
• Function promotion:
  \[ B = f(A); \quad //\text{applies } f \text{ elementwise for any function } f \]
• Includes built-in operators:
  \[ C = A + 1; \]
  \[ D = A + B; \]
  \[ E = A \times B; \]
  ...

Designed with parallelism in mind

• Operations on previous slides parallelized automatically

• Create asynchronous task w/ single keyword

• Built-in synchronization for tasks and variables
“Hello World” in Chapel

- Create file hello.chpl containing
  ```chapel
  writeln(“Hello World!”);
  ```
- Compile with
  ```bash
  chpl –o hello hello.chpl
  ```
- Run with
  ```bash
  ./hello
  ```
Variables and Constants

• Variable declaration format:
  
  [config] var/const identifier : type;

var x : int;
const pi : real = 3.14;
config const numSides : int = 4;
Serial Control Structures

• if statements, while loops, and do-while loops are all pretty standard

• Difference: Statement bodies must either use braces or an extra keyword:

  if(x == 5) then y = 3; else y = 1;
  while(x < 5) do x++;
Example: Reading until eof

```plaintext
var x : int;
while stdin.read(x) {
    writeln(“Read value “, x);
}
```
Procedures/Functions

```
proc addOne(in val : int, inout val2 : int) : int {
    val2 = val + 1;
    return val + 1;
}
```
Arrays

• Indices determined by a range:
  
  ```
  var A : [1..5] int; //declares A as array of 5 ints
  var B : [-3..3] int; //has indices -3 thru 3
  var C : [1..10, 1..10] int; //multi-dimensional array
  ```

• Accessing individual cells:
  
  ```
  ```

• Arrays have runtime bounds checking
For Loops

• Ranges also used in for loops:
  
  ```
  for i in 1..10 do statement;
  for i in 1..10 {
    loop body
  }
  ```

• Can also use array or anything iterable
Parallel Loops

• Two kinds of parallel loops:
  - forall i in 1..10 do statement;  //omit do w/ braces
  - coforall i in 1..10 do statement;

• forall creates 1 task per processing unit

• coforall creates 1 per loop iteration
  - Used when each iteration requires lots of work and/or they must be done in parallel
Asynchronous Tasks

• Easy asynchronous task creation:
  begin statement;

• Easy fork-join parallelism:
  cobegin {
    statement1;
    statement2;
    ...
  }  //creates task per statement and waits here
Sync blocks

- sync blocks wait for tasks created inside it
- These are equivalent:

```plaintext
cobegin {
    begin statement1;
    begin statement2;
    ...
}
csync {
    begin statement1;
    begin statement2;
    ...
}
```
Sync variables

• sync variables have value and empty/full state
  – store \( \leq 1 \) value and block operations can’t proceed

• Can be used as lock:
  
  ```javascript
  var lock : sync int;
  lock = 1; //acquires lock
  ...
  var temp = lock; //releases the lock
  ```
Analysis of Algorithms

• Chapel material
  – Assign basic tutorial
  – Teach forall & cobegin (also algorithmic notation)

• Projects
  – Partition integers
  – BubbleSort
  – MergeSort
  – Nearest Neighbors
Algorithms Project: List Partition

• Partition a list to two equal-summing halves.
• Brute-force algorithm (don't know P vs NP yet)
• Questions:
  – What are longest lists you can test?
  – What about in parallel?
• Trick: enumerate possibilities and use forall
Algorithms Project: BubbleSort

• Instead of left-to-right, test all pairs in two steps!

• Two nested forall loops (in sequence) inside a for loop
Algorithms Project: MergeSort

- Parallel divide-and-conquer: use cobegin
- Elegant division: split the Domain
- Speedup not as noticeable
- Example of expensive parallel overhead
Algorithms Project: Nearest Neighbors

• Find closest pair of (2-D) points.
• Two algorithms:
  – Brute Force
    • (use a forall like bubbleSort)
  – Divide-and-Conquer
    • (use cobegin)
    • A bit tricky
• Value of parallelism: much easier to program the brute-force method
Algorithms Takeaway

• Learning curve of Chapel is so low, students can start using parallelism very quickly
Module 3b
Reductions

(Reduction framework from Lin and Snyder, Principles of parallel programming, 2009.)
Summing values in an array

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>1</th>
<th>4</th>
<th>3</th>
<th>1</th>
<th>3</th>
<th>0</th>
<th>2</th>
</tr>
</thead>
</table>
Summing values in an array
Summing values in an array

\[
\begin{array}{cccc}
2 & 1 & 4 & 3 \\
3 & 7 & 10 & 4 \\
& 6 & 2 & \\
\end{array}
\]
Summing values in an array

2 1 4 3 1 3 0 2
Summing values in an array

2 1 4 3 1 3 0 2

3 7 10 6 16
Finding max of an array
Finding the maximum index
Finding the maximum index
Parts of a reduction

- Tally: Intermediate state of computation
- Combine: Combine 2 tallies
- Reduce-gen: Generate result from tally
Parts of a reduction

- **Tally**: Intermediate state of computation
  \[(value, \text{index})\]
- **Combine**: Combine 2 tallies
  take whichever pair has larger value
- **Reduce-gen**: Generate result from tally
  return the index

Init: Create "empty" tally
Accumulate: Add 1 value to tally
Two issues

• Need to convert initial values into tallies
• May want separate operation for values local to a single processor
Two issues

- Need to convert initial values into tallies
- May want separate operation for values local to a single processor

"Empty" tally ➔ Tally of these values
Parts of a reduction

• Tally: Intermediate state of computation

• Combine: Combine 2 tallies

• Reduce-gen: Generate result from tally

• Init: Create “empty” tally

• Accumulate: Add 1 value to tally
Parallel reduction framework

Tally: Intermediate state of computation
i = Init: Create "empty" tally
a = Accumulate: Add 1 value to tally
c = Combine: Combine 2 tallies
rg = Reduce-gen: Generate result from tally
Defining reductions

• Tally: Intermediate state of computation

• Combine: Combine 2 tallies

• Reduce-gen: Generate result from tally

• Init: Create “empty” tally

• Accumulate: Add 1 value to tally

Sample problems: +
Defining reductions

• Tally: Intermediate state of computation
• Combine: Combine 2 tallies
• Reduce-gen: Generate result from tally
• Init: Create “empty” tally
• Accumulate: Add 1 value to tally

Sample problems: +, histogram
Defining reductions

- Tally: Intermediate state of computation
- Combine: Combine 2 tallies
- Reduce-gen: Generate result from tally
- Init: Create “empty” tally
- Accumulate: Add 1 value to tally

Sample problems: +, histogram, max
Defining reductions

- Tally: Intermediate state of computation
- Combine: Combine 2 tallies
- Reduce-gen: Generate result from tally
- Init: Create “empty” tally
- Accumulate: Add 1 value to tally

Sample problems: +, histogram, max, 2nd largest
Defining reductions

• Tally: Intermediate state of computation

• Combine: Combine 2 tallies

• Reduce-gen: Generate result from tally

• Init: Create “empty” tally

• Accumulate: Add 1 value to tally

Sample problems: +, histogram, max, 2\textsuperscript{nd} largest, length of longest run
Can go beyond these...

- `indexOf` (find index of first occurrence)

- sequence alignment  [Srinivas Aluru]

- n-body problem  [Srinivas Aluru]
Relationship to dynamic programming

• Challenges in dynamic programming:
  – What are the table entries?
  – How to compute a table entry from previous entries?

• Challenges in reduction framework:
  – What is the tally?
  – How to compute a new tallies from previous ones?
Reductions in Chapel

• Express reduction operation in single line:
  
  ```chapel
  var s = + reduce A;  //A is array, s gets sum
  ```

• Supports +, *, ^ (xor), &&, ||, max, min, ...

• minloc and maxloc return a tuple with value and its index:
  
  ```chapel
  var (val, loc) = minloc reduce A;
  ```
Reduction example

• Can also use reduce on function plus a range
• Ex: Approximate $\pi/2$ using $\int_{-1}^{1} \sqrt{1-x^2} \, dx$ :

```plaintext
config const numRect = 10000000;
const width = 2.0 / numRect;       //rectangle width
const baseX = -1 - width/2;
const halfPI = + reduce [i in 1..numRect]
                (width * sqrt(1.0 – (baseX + i*width)**2));
```
Defining a custom reduction

• Create object to represent intermediate state

• Must support
  – accumulate: adds a single element to the state
  – combine: adds another intermediate state
  – generate: converts state object into final output
Classes in Chapel

class Circle {
    var radius : real;
    proc area() : real {
        return 3.14 * radius * radius;
    }
}

var c1, c2 : Circle; //creates 2 Circle references
var c1 = new Circle(10); /* uses system-supplied constructor
                         to create a Circle object
                         and makes c1 refer to it */

var c2 = c1; //makes c2 refer to the same object
delete c1; //memory must be manually freed
Inheritance

class Circle : Shape {    //Circle inherits from Shape
    ...
}

var s : Shape;

s = new Circle(10.0);  //automatic cast to base class

var area = s.area();    /* call recipient determined
                       by object’s dynamic type */
Example “custom” reduction

class MyMin : ReduceScanOp { //finds min element (equiv. to built-in “min”)
  type eltType; //type of elements
  var soFar : eltType = max(eltType); //minimum so far

  proc accumulate(val : eltType) {
    if(val < soFar) { soFar = val; }
  }

  proc combine(other : MyMin) {
    if(other.soFar < soFar) { soFar = other.soFar; }
  }

  proc generate() { return soFar; }
}
And that’s not all... (scans)

- Instead of just getting overall value, also compute value for every prefix

<table>
<thead>
<tr>
<th>A</th>
<th>2</th>
<th>1</th>
<th>4</th>
<th>3</th>
<th>1</th>
<th>3</th>
<th>0</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>10</td>
<td>11</td>
<td>14</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>
And that’s not all... (scans)

- Instead of just getting overall value, also compute value for every prefix

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>1</th>
<th>4</th>
<th>3</th>
<th>1</th>
<th>3</th>
<th>0</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>10</td>
<td>11</td>
<td>14</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

- Useful answering queries like
  “What is the sum of elements 2 thru 7?”
  \[= \text{sum}[7] - \text{sum}[1]\]
Computing the scan in parallel

Upward pass to compute reduction
Computing the scan in parallel

Upward pass to compute reduction
Downward pass to also compute scan
Downward pass with function labels

\[ i = \text{init} \]
\[ a = \text{accumulate} \]
Many options for module 3

• Using Chapel for ease of parallelization

• Reductions on paper (defining and/or using)

• Also implementing reductions in Chapel

Side question: Where to put it?
Caveats

• Still in development
  – Reductions serialized on multicore (as of 1.6)
  – Error messages thin
  – New versions every 6 months – some big changes
  – Not many libraries

• No development environment
  – Command-line compilation in Linux
“TODO” list

• Notes, slides, assignments, etc
• Evidence on tie to dynamic programming
• Sample adoption strategies
• More applications of reductions and scans

Please share!
Other resources

• CS in Parallel
  http://csinparallel.org

• Dan Grossman’s CS 2 notes
  http://homes.cs.washington.edu/~djg/teachingMaterials/spac/

• NSF/IEEE-TCPP Curriculum Initiative
  http://www.cs.gsu.edu/~tcpp/curriculum/
Thanks for your time

dbunde@knox.edu
http://faculty.knox.edu/dbunde/teaching/CCSC-MW13