Using Python for an Early Introduction to Concepts of Parallelism, Distributed Systems, and Parallel Image Processing

Steven Bogaerts
Assistant Professor of Computer Science
Wittenberg University
Springfield, OH

Joshua Stough
Assistant Professor of Computer Science
Washington and Lee University
Lexington, VA

http://cs.wlu.edu/~stough/SC13

Python Installation

- Easy!
  - www.python.org, “Download”, “Python 2.7.3”
- 2.x or 3.x?
  - 3.x has some changes to the base language (not backwards compatible)
    - Better handling of unicode
    - Exception chaining
    - …
  - Many third-party libraries still support only 2.x
  - Most current Linux distributions and Macs us 2.x as default
- So we’ll stick with 2.x here

Why Python?

- Simple syntax (as we'll demonstrate)
  - No variable declaration
  - Variables can hold any type
  - Automatic garbage collection
  - No explicit memory management
- Allows consideration of interesting problems sooner
- Students definitely need to learn the concepts Python brushes over…
  - …but not necessarily in the first course or two
  - What is the meaning of each `const`?

```
const string & foo(const int * const p) const;
```

Python Crash Course

- Reasons:
  - So you can follow the rest of our presentation
  - Demonstrate the kinds of concepts you can consider early on with Python in CS1
- See pythonCrashCourse.py

Parallelism in Python

- Our purpose: For learning, not for all-out speed
- Options
  - pprocess
  - Celery
  - MPI4Py
  - Parallel Python
  - Multiprocessing module

Our Choice: Multiprocessing Module

- Comparatively simple
- Good documentation
- Comes with Python 2.6+
- Does not work in IDLE
  - Edit with any editor, then run at terminal
  - Might need to set `PYTHONPATH` environment variable to your Python installation's `Lib` directory
    - Could use a batch file:

```
SET PYTHONPATH="C:\Program Files\Python\2.7.3\Lib"
"C:\Program Files\Python\2.7.3\python.exe"
```

    - Then use Python `import` command to load a file
- So how do we teach parallelism with the multiprocessing module?

One Effective Way to Teach Parallelism in CS1

Using the Python Multiprocessing Module

Application of this Approach

- First attempt: Fall 2009
  - Tried parallelism too early in the semester!
    (about 1/3 of the way through CS1)
  - Introduction of some concepts needed better organization
- Fall 2010, Fall 2011, Spring 2013
  - Concepts introduced much later
    (about 3/4 of the way through CS1)
  - Now a smooth integration with the rest of the course
- Students having this CS1 experience (and related experiences in CS2, etc.) have shown strong understanding of parallelism before beginning our Sequential and Parallel Algorithms course

How do you fit it in?

- Yes, it is a new topic, and yes, a little something might need to be cut
- We ended up shifting concepts that are also covered in other courses
  - Our CS2 covers writing classes in great detail, so much less is now in CS1
- But parallelism also serves as a great complement to the rest of CS1 (and other courses, in different ways)
  - A great *medium* to study and review core CS1 topics

This Presentation is about Python, but…

- We do some non-Python introduction first:
  - The world is “obviously” parallel.
  - Big-picture descriptions of some applications.
  - Physical activities
    - Low-level: binary adder
    - Higher-level: card sorting
  - Terminology, history
  - Communication
    - Shared memory vs. message passing

Teaching Parallelism in CS1 with Python

- All materials on website, students follow along on own computer
- Big picture on slides
  - Overview at the start
  - "Cheat sheet" when done
- Heavily-commented code illustrates details
  - Some completed examples
  - Some exercises
  - Pause after each section for students to fill in "Key Ideas" sections

Parallel Programming Mechanisms

- Process
  - A running program
    - Keeps track of current instruction and data
  - Single-core processor: only one process actually runs at a time
    - Many processes "active" at once – OS goes from one to another via a *context switch*
- Threads
  - A process can contain multiple threads – things that can/should happen at the same time
  - Multi-core processor: multiple threads of a given process can run at the same time

Programming Background

- Tuples
  - Comma required for length 1
  - Comma optional for length >1
- Keyword arguments
  - For example: `func(y = 14, x = 27)`
- `from random import randint`
  `randint(low, high)`
  - Includes low and high!
- `from time import time, sleep`
  - `time.time()` for current time in seconds
    - Call a second time and subtract for elapsed time
  - `time.sleep(seconds)` to sleep for that amount of time

Spawning Processes

- `from multiprocessing import *`
- Create and start a process:
  - `procVar = Process(target = funcNoParen, args = tupleOfArgs)`
  - `procVar.start()`
- Get process info:
  - `current_process().pid`
  - `current_process().name`
    - Gives name specified by the "name=___" argument in process creation

Locks

- Only one process can acquire a given lock at a time
  - Any other process that tries will sleep until lock is released
- Use to control access to stdout and other shared resources
- `lockVar = Lock()`
  - Pass `lockVar` to all processes that need it
- `lockVar.acquire()`
- `lockVar.release()`

Communication

- `queueVar = Queue()`
  - Pass `queueVar` to all processes that need it
  - `queueVar.put(dataToSend)`
  - `dataToReceive = queueVar.get()`
    - Process will sleep until there's something to get
  - The first data `put` into the queue is the first data `get`-ed out of the queue

- `procVar.join()`
  - Makes current process sleep until the procVar process completes

Sleeping

- When would a process sleep?
  - Calls the `time.sleep` function
  - Waiting for a process to finish (`procVar.join()`)
  - Waiting to acquire a lock (`lockVar.acquire()`)
  - Waiting for something to be put in the queue (`queueVar.get()`)

An Additional (and Somewhat) Effective Way to Teach Parallelism in CS1

Using the Python Multiprocessing Module

Using Playing Cards

- First day: sort a deck of cards, and show me how
  - In pairs, precise, simple steps
    - If you can't describe what you are doing as a process, you don't know what you're doing. (W.E. Deming)
  - Introduces:
    - variable assignment ('take that card…'), conditionals, expressions (comparison), loops, (potentially) functional abstraction (find min)
- Much later, during search/sorting/complexity
- Now they're ready, know O(N^2) sorting

Parallelism using Playing Cards

- Whenever there is a hard job to be done I assign it to a lazy man; he is sure to find an easy way of doing it. (W. Chrysler)







Pool/map, Process/Pipe

- Pool/map: easy, great for data parallelism
  - parallel[Hello|SumPrimes|MontePi|Integration|MergesortPool].py
  - `from multiprocessing import Pool`
  - `mypool = Pool(processes=N)`
  - `mypool.map(myfunc, args)`
  - `args` is list of arguments to evaluate with `myfunc`
  - `myfunc` can accept only one argument (using wrapping)
- Process/Pipe: data/task parallelism
  - parallel[Quicksort|Mergesort].py
  - `parentConn, childConn = Pipe()`
    - duplex (both can send and receive)

Links:

- Obviously: http://docs.python.org/library/multiprocessing.html
- Our code: http://cs.wlu.edu/~stough/SC13/
- CS1 quotes: http://www.cs.cmu.edu/~pattis/quotations.html
- Jokes: http://www.phy.ilstu.edu/~rfm/107f07/epmjokes.html
- Distributed computing using multiprocessing: http://eli.thegreenplace.net/2012/01/24/distributed-computing-in-python-with-multiprocessing/
- Various options for PDC in Python:
  http://wiki.python.org/moin/ParallelProcessing
  http://wiki.python.org/moin/DistributedProgramming
  http://code.google.com/p/distributed-python-for-scripting/