Parallelism in Python

Steven Bogaerts
Computer Science
DePauw University
Who Am I?

• Working in parallelism in various courses since about 2008
  – Interdisciplinary team-taught non-major course
  – CS0 (Scratch)
  – CS1 (Python, Java)
  – CS2 (Python, Java)
  – Algorithms (C++)

• Goals:
  – Two-way exchange of ideas about teaching parallelism in Python in intro courses.
  – Time for you to adapt materials while we’re all here to discuss them together.
Credit

• Joshua Stough, Bucknell College
  – Some of the Python materials presented here

• Dan Grossman, University of Washington
  – Divide-and-conquer parallelism (in Java)
Who Are You?

- Name
- Institution
- Past experience in parallelism or teaching parallelism?
- Goals for our time in the Python sessions
Outline for the Python Sessions

• Exploring the multiprocessing module...
  – Primarily message-passing
    Just a little shared memory
• ...but the biggest challenge isn’t technical, it’s pedagogical.
  – Hands-on, but not overwhelming
  – Quick, but not rushed
  – Appetizing, but not satiating

• Overview of lots of level-appropriate applications

• Time to work
  – More (smaller group? individual?) discussions
  – Practice the technical concepts
  – Adapt materials into something perfect for you
Know Your Students

• Meet students where they are - one size does not fit all!

• Lots of time here to explore what most interests you, practice, adapt for your students
Python Installation

• [www.python.org](http://www.python.org), “Downloads”, “Python 3.5.1”

• 2.x or 3.x?
  – 3.x has some changes to the base language (not backwards compatible)
    • Better handling of unicode
    • Exception chaining
    • ...
  – A few third-party libraries still support only 2.x
    • [https://python3wos.appspot.com/](https://python3wos.appspot.com/)

• We’ll go with 3.x here
Python 3.x

• Full details of what’s new:  
  https://docs.python.org/3/whatsnew/3.0.html
• What affected me:
  – print(“hello”) instead of print “hello”
  – cmp() no longer exists
  – 1/2 gives float. 1//2 gives int.
  – != instead of <>
  – raw_input(...) (2.x) is now just input(...) (3.x)
    input(...) (2.x) is now eval(input(...)) (3.x)
  – Classes must explicitly extend object (or something else)
• Auto conversion tool!
  https://docs.python.org/3/library/2to3.html#to3-reference
  – Windows:
    pythonpath\Python.exe pythonpath\Tools\scripts\2to3.py -w filename.py
  – Helpful, but doesn’t catch more subtle changes
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

• Just in case it’s helpful some day…
• Lists, loops, accumulation, file I/O, dictionaries, simple classes, comparators and the built-in sort operation
Parallelism in Python

• Options
  – pprocess
  – Celery
  – MPI4Py
  – Parallel Python
  – Multiprocessing module

• My purpose: For learning, not for all-out speed
My 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
    - See [Python command line with path.bat](https://example.com)
- So how do I teach parallelism with the multiprocessing module?
Multiprocessing Module: Non-Programming Prerequisites

- High-level overview of parallelism concepts (1-3 hours of class time)
  - Non-technical examples of tasks being done in parallel
  - Non-technical overview of process coordination through message-passing

- Examples:
  - 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

- Later at this workshop, we can discuss these in more detail if you want
Multiprocessing Module: Python Prerequisites

• Bare minimum:
  – Printing
  – Variables
  – Tuples
  – Writing and calling functions
  – …
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 the Sequential and Parallel Algorithms course
General Approach: Analogy to OOP

• Different paradigm from imperative programming
• Decades ago:
  – How can OOP be integrated into introductory programming courses? Is this wise, or even possible?
• Answer:
  – Not just in a high-level OOP course
  – CS1: objects, basic class construction
  – Data structures: abstract data types via classes
  – Graphics: abstraction through classes
  – ...
  – Some extra time to learn mechanics, but OO is simply the medium now
    • Most key concepts of imperative programming have not been sacrificed!
Parallelism as a Medium

• Yes, it is a new topic, and 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
Teaching Parallelism in CS1 with Python

• All materials on website, students follow along on own computer
• Big picture on slides
  – Overview at the start
  – Quick reference when done
• Heavily-commented code illustrates details
  – Some completed examples
  – Some exercises
  – Pause after each section for students to fill in “Key Ideas” sections
Spawning Processes

• Simplified overview
  – Running program, with current instruction and data maintained
  – Single-core processor: one process at a time
    • Concurrency – context switches and the illusion of parallelism
    • View running processes to illustrate
  – Multi-core processor: literally execute multiple processes at once

• Helpful study advice: people believe that personal “multitasking” is parallelism, but it is really concurrency...
Programming Background

- See spawningProcesses.py
- 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()`)
Highlights of applications.py

• A collection of slightly more complex applications of multiprocessing
  – Still level appropriate (CS2, maybe CS1)
  – Probably too many to do all

• I’ll briefly present highlights and possible uses

• Then we’ll have time for whatever – individual adaptation, smaller group discussions, etc.
Adaptations: Big-Picture

• Broadly speaking, each example could be used in many ways
  – Students write code like this from scratch
  – Students write some code, given starter code, in or out of class
  – In-class discussion
Searching and Timing

• A purely sequential example, but gives students practice working with timing and experimentation
• Compares linear and binary search, on lists of three data types: Student, string, int
• Students can form hypotheses about efficiency, run quick experiments, report on results, and draw conclusions
Timing Message Passing

• Sender sends a message to receiver repeatedly (producer / consumer)

• Ideas to explore:
  – Effect of message size or number of messages
  – Adding more producers and/or consumers
  – Means of creating strings:
    https://waymoot.org/home/python_string
Quadratic Formula

- Familiar to students
- One child process computes “+”, the other “-”
- Reinforces basic syntax, but could also ask students to argue whether this is a worthwhile application of parallelism.
Adding a List of Numbers

• Split the list into two pieces for two children. Each sums the numbers, reports to parent.

• Possible adaptations:
  – More than two children
  – Timing, probably change to shared memory Array
Finding the Nearest Point

• Given a list of (x,y) coordinates of houses, and the coordinates of a railroad station – which house is closest?

• Possible adaptations:
  – Each child looks at part of the list
  – Working with two vs. an arbitrary number of children
  – Copying of the list to each child: inefficient!
    • Use two Arrays instead – for x and y coordinates, respectively
Finding the Largest Key

• Splitting up a list, each child looks for a needed key in its part
  – Have to manage an offset for each child, based on which sublist it has
• Note alternative approach in splitting list
  – Put “extras” in the last process
Pattern Matching (DNA?)

• Split the text into overlapping segments for the children, returning a list of all occurrences of the pattern.
K-Medoid Clustering

• Given a collection of (x,y) points, return a clustering of those points into groups by distance
• Uses John Zelle’s graphics module (from his Python textbook, Python Programming, An Introduction to Computer Science)
A Pipeline for Floating-Point Addition

• A three-step pipeline ("assembly line") for adding a series of pairs of floating-point numbers
• Won’t be faster than just using Python’s addition, of course, but illustrates the idea
Please spend a moment to think about what direction you want the “hands on” work to go next.

• Discuss:
  – Concerns / questions you have
  – Challenges you’ve faced or expect to face
  – What worked, what didn’t
  – “CS Unplugged”-style parallelism exercises
  – Other courses?

• Develop:
  – Get more practice in parallelism in Python by working on the exercises in application.py
  – Work on some other application that would work well for your students and your interests
  – Adaptations of these materials to fit your voice, what your students will expect, etc.

Please write down your thoughts / hopes / plans. In a few minutes, we’ll collect ideas and decide where to go from there (all together, in smaller groups, or separately for a time).
Analogies I’ve Used

• Effectiveness and overhead: Shovel driveway vs. dig a hole with a shovel vs. run a company
• Clock speed: making PB&J
• Interprocess communication: counting people in the building (info at start vs. text vs. Google Doc)
  – Low-level: binary adder
  – Higher-level: card sorting
• Locks: the conch in Lord of the Flies
• Join: children smelling wild flowers in a park
Divide-and-Conquer Parallelism

- Code: splitting an array into 4 pieces
- Analogy: Cooks peeling a huge pile of potatoes
- Head cook has four helper cooks
- Split potatoes into 4 smaller piles
- Suppose you manage 5 cooks – nothing for fifth cook!
- Suppose you manage 3 cooks – whichever cook finishes gets the fourth pile
- What if one pile has potatoes with more bad spots?
- What if some cooks have other things they need to do too?
- Alternative: take one reasonable size sub-pile at a time, keep going until done.