Introducing Tetra: An Educational Parallel Programming System

Ian Finlayson, Jerome Mueller, Shehan Rajapakse, Daniel Easterling

May 25, 2015
Motivation

- We are in a multicore world.
- Several calls for more parallel programming, earlier in the curriculum\(^1\).
- Yet to happen in a wide-spread way.

\(^1\) “Parallel from the beginning: The Case for Multicore Programming in the Computer Science Undergraduate Curriculum”
Introductory Languages

- Shift to higher-level languages for introductory courses\(^1\).
- Python’s global interpreter lock makes parallelism impossible\(^2\).
- Java requires knowledge of libraries, inheritance and polymorphism.
- C and C++ not widely used for introductory courses.

---

1 “Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities”
2 “Understanding the Python GIL”
Parallel Debugging

- Parallel programs harder to debug.
- There is a lack of good tool support.
- Hard to “stop the world” when running parallel code.
Introducing Tetra: An Educational Parallel Programming System

Tetra

Programming Language
- High-level and similar to Python.
- Built-in support for parallelism.

IDE
- Standard IDE features for editing and running programs.
- Debugger geared especially for parallel programs.
Base Language

- Simple procedural programming language.
- Syntax based on Python.
- Statically typed with local inference.
- Requires a main function.

```python
# a simple factorial function
def fact(x int) int:
    if x == 0:
        return 1
    else:
        return x * fact(x - 1)

# the main function
def main( ):
    print("Enter n: ")
    n = read_int( )
    print(n, "! = ", fact(n))
```
Parallel Statements

- Each statement under a parallel block is executed in parallel.

- The program waits for all to finish before continuing.

```python
def main( ):
    # call two functions in parallel
    parallel:
        f1( )
        f2( )
    # now they are both done
    print("All done!")
```
The statements under a background block are executed asynchronously from the rest.

The background task is allowed to finish if the outer code returns.

```python
def main( ):
    # launch a task in the background
    background:
        f1( )
        f2( )
    # continue on
    f3( )
```
Parallel For Loops

- A parallel for loop signifies the iterations are independent of each other.
- The iterations are divided amongst a pool of threads.

```python
def main( ):
    # process values 1-100 in parallel
    parallel for i in [1 ... 100]:
        process(i)
```
Locks

- Locks provide mutual exclusion.
- No two threads enter an area with the same lock name at the same time.

```python
# globals need mutual exclusion
global value = 0

def inc( ):
    lock value_lock:
        value = value + 1

def dec( ):
    lock value_lock:
        value = value - 1
```
Sequential Sum Program in Python

```python
total = 0

def sum_range(numbers, a, b):
    i = a
    while i <= b:
        global total
        total = total + numbers[i]
        i = i + 1

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
sum_range(numbers, 0, 9)
print(total)
```
from threading import Thread, Lock

total = 0
lock = Lock()

def sum_range(numbers, a, b):
    i = a
    while i <= b:
        global total
        lock.acquire()
        total = total + numbers[i]
        lock.release()
    i = i + 1

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
t1 = Thread(target = sum_range, args = (numbers, 0, 4))
t1.start()
t2 = Thread(target = sum_range, args = (numbers, 5, 9))
t2.start()
t1.join()
t2.join()
print(total)
Sequential Sum Program in Tetra

global total = 0

def sum_range(numbers [int], a int, b int):
    i = a
    while i <= b:
        total = total + numbers[i]
        i = i + 1

def main( ):
    numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    sum_range(numbers, 0, 9)
    print(total)
global total = 0

def sum_range(numbers [int], a int, b int):
    i = a
    while i <= b:
        lock total_lock:
            total = total + numbers[i]
        i = i + 1

def main( ):
    numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    parallel:
        sum_range(numbers, 0, 4)
        sum_range(numbers, 5, 9)
    print(total)
First Class Parallelism

- Minimizes boilerplate code.
- Reduces chance of making simple mistakes.
- Requires less knowledge of the language.
Tetra code is parsed into an AST which is interpreted.

Threads are launched when parallel nodes are reached.

Goal is to achieve speedup on parallel code.

$$\text{parallel for } i \text{ in } [1 \ldots 100]: \text{process}(i)$$
Finding Primes from 1 to 1,000,000

- **one** thread: Actual Time, Ideal Time
- **two** threads: Actual Time, Ideal Time
- **four** threads: Actual Time, Ideal Time
- **eight** threads: Actual Time, Ideal Time
Speedup

10 City Traveling Salesman Problem

![Graph showing speedup for different number of threads (one, two, four, eight). The graph compares actual time (black) and ideal time (blue).]
Debugging

- Interpreter also supports dynamic debugging
- A command line debugger supports breakpoints, listing variables, inspecting values, stepping through execution, listing thread states, switching threads and stopping threads.
- Supports “stop the world” debugging.
• Integrated Development Environment being developed.
• Supports editing, saving, and running programs.
• Supports some debugging commands.
Introducing Tetra: An Educational Parallel Programming System

Ian Finlayson
Related Work

Parallel Languages

- Several languages provide first-class parallelism such as X10, Chapel, Cilk, and UPC.
- Parallel features also inspired from OpenMP.

Parallel Debuggers

- Most debuggers provide some support for debugging parallel code, but lack the ability to step through threads totally independently.
Future Work

- Improvement of the IDE.
- Additional language features.
- More robust standard library.
- A study of Tetra in a classroom setting to see if it does help students develop parallel programs.
- A native code compiler.
Most languages focused on ease of use or efficiency.

Tetra is geared towards providing parallelism in a simple way.

While there is much future work, Tetra can be used to write and debug parallel code.
Questions