Parallel Erlang

Mark Greenstreet

CpSc 418 – Sept. 12, 2013

Outline:

- Processes
- Count 3s and other Trees
- Parallel Programming Abstractions

Objectives

- Introduce Erlang's features for concurrency and parallelism
  - Spawning processes.
  - Sending and receiving messages.
- Count 3s as a simple parallel program
  - The parallel version
  - Refining the parallel version
  - Applying that structure to other problems
- Parallel Programming Abstractions
  - Reduce
  - The `workers` and `wtree` modules

Processes – Overview

- The built-in function spawn creates a new process.
- Each process has a process-id, pid.
  - The built-in function self() returns the pid of the calling process.
  - `spawn` returns the pid of the process that it creates.
  - The simplest form is `spawn(Fun)`.
    - A new process is created.
    - The function `Fun` is invoked with no arguments in that process.
- Sending a message.
  - `Pid ! Message`
    sends `Message` to the process with pid `Pid`.
  - Message is any Erlang term (i.e. an arbitrary expression).
- Receiving messages:
  See next slide.

Receiving Messages (short version)

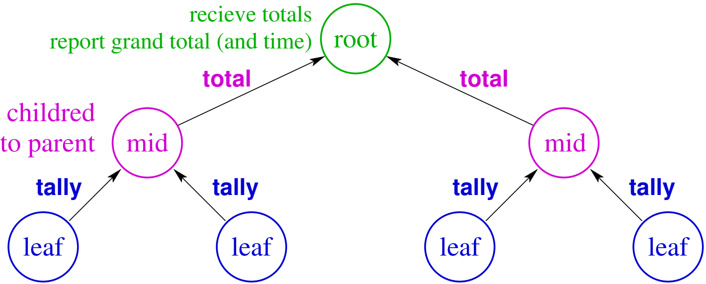
```
receive
    Pattern1 -> Expr1;
    Pattern2 -> Expr2;
    ...
    PatternN -> ExprN
end
```

- If there is a pending message for this process that matches one of the patterns,
  - The message is delivered, and the value of the `receive` expression is the value of the corresponding *Expr*.
  - Otherwise, the process blocks until such a message is received.

A simple example

```
1> MyPid = self().
<0.152.0>
2> spawn(fun() -> MyPid ! "hello world" end).
<0.164.0>
3> receive Msg1 -> Msg1 end.
"hello, world"
```

Count 3s with a Tree

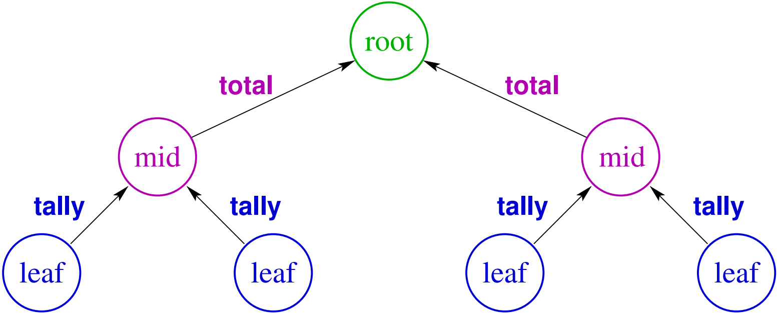

recieve totals
report grand total (and time)
root
total
total
receive tallys from childred
add and send total to parent
mid
mid
tally
tally
tally
tally
leaf
leaf
leaf
leaf

create list of N/P elements
count 3s in the list
send total to parent

Let’s try it

This space left blank for your notes. I’ll show the erlang separately.

A better timing measurement



```
leaf:
  create list of N/P elements
  send 'ready' to parent
  wait for 'go'
  count 3s in the list
  send total to parent
```

```
mid:
  wait for readies:
    send ready to parent
  wait for go:
    send gos to children
  wait for tallies
    send total to parent
```

```
root:
  wait for readies
  start timer
  send gos
  wait for totals
  compute grand total
  end timer
  report results
```

Let's try it

This space left blank for your notes. I'll show the erlang separately.

The reduce pattern

Counting 3s is fun, but it won't pay the rent.

- What if I wanted to know the sum of the elements in a big list, distributed across a tree of processes?
- What if I wanted to know the maximum of the elements in a big list, distributed across a tree of processes?
- What if I wanted to know the third largest of the elements in a big list, distributed across a tree of processes?
- What if I wanted to know the longest run of that atom a `cow` in a big list distributed across a tree of processes?

What's the pattern?

This space left blank for your notes. I'll put something into the final version of the slides.

What should a worker process look like?

This space left blank for your notes. I'll put something into the final version of the slides.

Keeping data between requests

This space left blank for your notes. I'll put something into the final version of the slides.

APIs

- workers pools

  http://www.ugrad.cs.ubc.ca/~cs418/resources/erl/doc/workers.html
- worker trees

  http://www.ugrad.cs.ubc.ca/~cs418/resources/erl/doc/workers.html

Count 3s using `wtree`

This space left blank for your notes. I'll show the erlang separately.

Summary

To be added in the final version.

Preview

September 17: Performance Loss

Reading: Lin & Snyder, chapter 3, pp. 61–68

September 19: Performance Measurement

Homework: Homework 2 goes out – parallel programming with Erlang
Reading: Lin & Snyder, chapter 3, pp. 68–77
Homework: Homework 1 deadline for early-bird bonus

September 24: Matrix Multiplication

Reading: Lin & Snyder, chapter 3, pp. 77–85
Homework: Homework 1 due

September 26: Superscalars and compilers

Reading: The MIPS R10000 Superscalar Microprocessor (Yeager)

October 1: Shared Memory Multiprocessors

Reading: Lin & Snyder, chapter 2, pp. 30–43.
Homework: Homework 3 goes out

October 3: Message Passing Multiprocessors

October 8: Models of Parallel Computation

Reading: Lin & Snyder, chapter 3, pp. 43–59.

Review Questions

To be added in the final version.