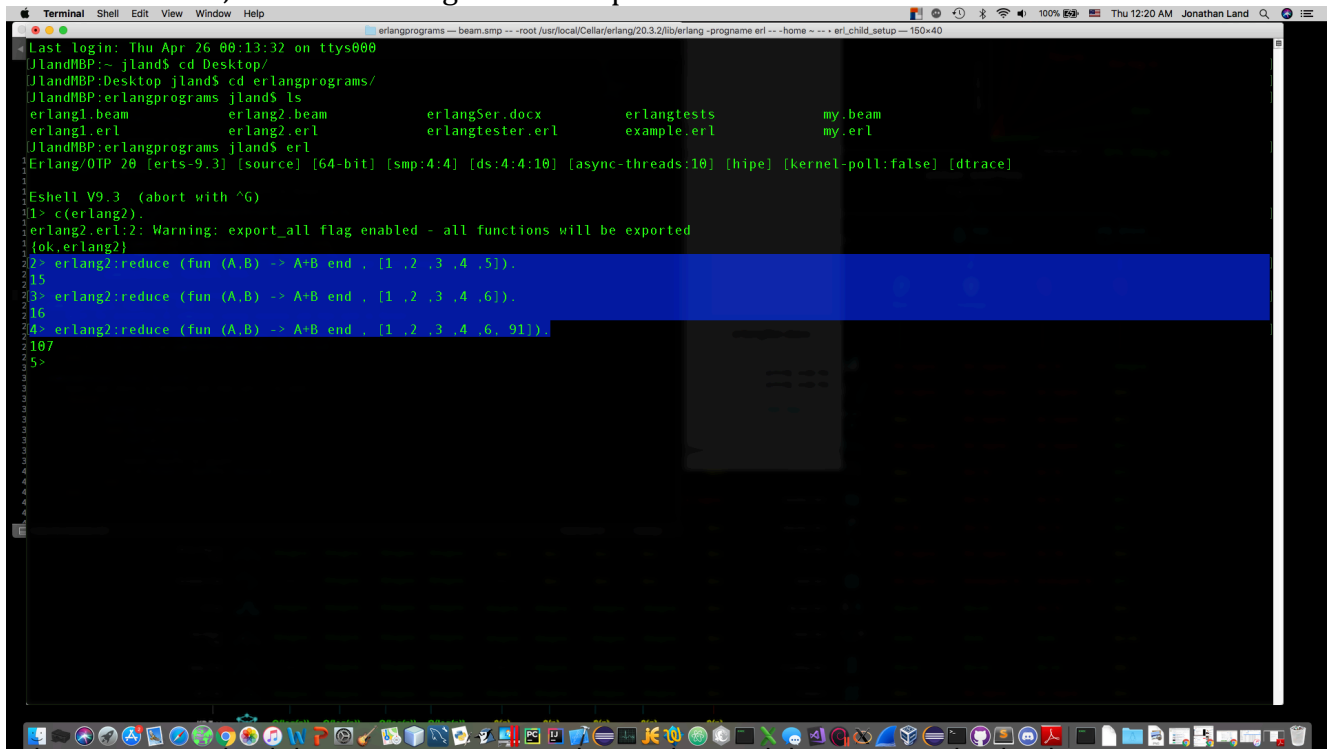


# Erlang Parallelization

## Jonathan Land

Screenshots: associative and commutative (odd and even sized lists).

NOTE: As will be seen in the code, I struggled with the syntax for if statement (or whatever else was needed) to just return one element in the list if there was one was left. I know it is one line or letter of code, but I could not get it to compile.



```
Terminal Shell Edit View Window Help
erlangprograms -- beam.smp -- -root /usr/local/Cellar/erlang/20.3.2/lib/erlang -progname erl -- -home ~ -- -erl_child_setup -- 150x40

Last login: Thu Apr 26 00:13:32 on ttys000
jlandMBP:~ jland$ cd Desktop/
jlandMBP:Desktop jland$ cd erlangprograms/
jlandMBP:erlangprograms jland$ ls
erlang1.beam      erlang2.beam      erlangSer.docx    erlangtests       my.beam
erlang1.erl       erlang2.erl       erlangtester.erl  example.erl       my.erl
jlandMBP:erlangprograms jland$ erl
Erlang/OTP 20 [erts-9.3] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-threads:10] [hipe] [kernel-poll:false] [dtrace]

1 Eshell V9.3 (abort with ^G)
1 1> c(erlang2).
1 erlang2.erl:2: Warning: export_all flag enabled - all functions will be exported
1 {ok,erlang2}
1 2> erlang2:reduce (fun (A,B) -> A+B end , {1 ,2 ,3 ,4 ,5}).
2 15
2 3> erlang2:reduce (fun (A,B) -> A+B end , {1 ,2 ,3 ,4 ,6}).
2 16
2 4> erlang2:reduce (fun (A,B) -> A+B end , {1 ,2 ,3 ,4 ,6, 91}).
2 107
2 5>
```

```
Terminal Shell Edit View Window Help
erlangprograms -- beam.smp -- -root./usr/local/Cellar/erlang/20.3.2/lib/erlang -progname erl -- -home ~ - - - erl_child_setup -- 150x40

Last login: Thu Apr 26 00:13:32 on ttys000
jlandMBP:~ jland$ cd Desktop/
jlandMBP:Desktop jland$ cd erlangprograms/
jlandMBP:erlangprograms jland$ ls
erlang1.beam      erlang2.beam      erlangSer.docx    erlangtests       my.beam
erlang1.erl       erlang2.erl       erlangtester.erl  example.erl       my.erl
jlandMBP:erlangprograms jland$ erl
Erlang/OTP 20 [erts-9.3] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-threads:10] [hipe] [kernel-poll:false] [dtrace]

Eshell V9.3 (abort with ^G)
1> c(erlang2).
erlang2.erl:2: Warning: export_all flag enabled - all functions will be exported
1 {ok,erlang2}
2> erlang2:reduce (fun (A,B) -> A+B end , [ 1 ,2 ,3 ,4 ,5]).
15
3> erlang2:reduce (fun (A,B) -> A+B end , [ 1 ,2 ,3 ,4 ,6]).
16
4> erlang2:reduce (fun (A,B) -> A+B end , [ 1 ,2 ,3 ,4 ,6, 91]).
107
5>
```

```
Terminal Shell Edit View Window Help
erlangprograms -- beam.smp -- -root./usr/local/Cellar/erlang/20.3.2/lib/erlang -progname erl -- -home ~ - - - erl_child_setup -- 150x40

Last login: Thu Apr 26 00:13:32 on ttys000
jlandMBP:~ jland$ cd Desktop/
jlandMBP:Desktop jland$ cd erlangprograms/
jlandMBP:erlangprograms jland$ ls
erlang1.beam      erlang2.beam      erlangSer.docx    erlangtests       my.beam
erlang1.erl       erlang2.erl       erlangtester.erl  example.erl       my.erl
jlandMBP:erlangprograms jland$ erl
Erlang/OTP 20 [erts-9.3] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-threads:10] [hipe] [kernel-poll:false] [dtrace]

Eshell V9.3 (abort with ^G)
1> c(erlang2).
erlang2.erl:2: Warning: export_all flag enabled - all functions will be exported
1 {ok,erlang2}
2> erlang2:reduce (fun (A,B) -> A+B end , [ 1 ,2 ,3 ,4 ,5]).
15
3> erlang2:reduce (fun (A,B) -> A+B end , [ 1 ,2 ,3 ,4 ,6]).
16
4> erlang2:reduce (fun (A,B) -> A+B end , [ 1 ,2 ,3 ,4 ,6, 91]).
107
5> erlang2:reduce (fun (A,B) -> A++B end , ["A", "B", "C", "D", "E "]).
"E DCBA"
6> erlang2:reduce (fun (A,B) -> A++B end , ["A", "B", "C", "D", "E"]).
"EDCBA"
7> erlang2:reduce (fun (A,B) -> A++B end , ["A", "B", "B", "B", "A", "D", ]).
* 1: syntax error before: ']'
7> erlang2:reduce (fun (A,B) -> A++B end , ["A", "B", "B", "B", "A", "G", "H", "I"]).
"IHGABBBBA"
8>
```

The screenshot shows a Sublime Text editor with an Erlang file named `erlang2.erl`. The code implements a parallel sorting algorithm using a `reduce` function. The algorithm splits a list into two halves, sorts them recursively, and then merges the results. The code is as follows:

```

1 -module(erlang2).
2 -compile(export_all).
3 -export([reduce/2]).
4
5 reduce(Func, List) ->
6   reduce(root, Func, List).
7
8 %When done send results to Parent
9 reduce(Parent, _, [A]) ->
10   %send to parent
11   Parent ! { self(), A };
12
13 %get contents of list, apply function and store in Parent
14 reduce(Parent, Func, List) ->
15   { Left, Right } = lists:split(trunc(length(List)/2), List),
16   Me = self(),
17   %io:format("Splitting in two~n"),
18   Pl = spawn(fun() -> reduce(Me, Func, Left) end),
19   Pr = spawn(fun() -> reduce(Me, Func, Right) end),
20   %merge results in parent and call Func on final left and right halves
21   combineParent, Func, Pl, Pr).
22
23 %merge p1 and p2 and combine in parent
24 combineParent, Func, [Pl, Pr] ->
25   %wait for processes to complete (using receive) and then send to Parent
26   receive
27     { Pl, Sorted } -> combineParent, Func, Pr, Sorted;
28     { Pr, Sorted } -> combineParent, Func, Pl, Sorted
29   end.
30
31 combineParent, Func, P, List ->
32   %wait and store in results and then call ! to send
33   receive
34     { P, Sorted } ->
35       Results = Func(Sorted, List),
36       case Parent of
37         root ->
38           Results;
39         _ ->
40           %send results to parent
41           Parent ! { self(), Results }
42       end
43   end.
44
45

```

On the right side of the screenshot, there is a diagram titled "ARRAY SORTING Algorithms". It shows a tree structure of sorting algorithms categorized by Time Complexity and Space Complexity. The algorithms listed are: Best, Average, Worst, and Worst. The complexities are as follows:

Algorithm	Time Complexity	Space Complexity
Best	$O(n \log n)$	$O(n \log n)$
Average	$O(n \log n)$	$O(n \log n)$
Worst	$O(n^2)$	$O(n \log n)$
Worst	$O(n \log n)$	$O(n)$

The diagram also includes a graph showing the growth of operations for different algorithms:  $O(n!)$ ,  $O(2^n)$ ,  $O(n^2)$ ,  $O(n \log n)$ , and  $O(n)$ .

## Erlang Code

```

-module(erlang2).
-compile(export_all).
-export([reduce/2]).

```

```

reduce(Func, List) ->
  reduce(root, Func, List).

```

```

%When done send results to Parent
reduce(Parent, _, [A]) ->
  %send to parent
  Parent ! { self(), A };

```

```

%I tried these to take care of one el in list, but it didn't work
%length ([]) ->
% Parent !      {self(), A};

```

```

%get contents of list, apply function and store in Parent
reduce(Parent, Func, List) ->
  { Left, Right } = lists:split(trunc(length(List)/2), List),
  Me = self(),
  %io:format("Splitting in two~n"),
  Pl = spawn(fun() -> reduce(Me, Func, Left) end),
  Pr = spawn(fun() -> reduce(Me, Func, Right) end),
  %merge results in parent and call Func on final left and right halves

```

```
combine(Parent, Func,[Pl, Pr]).
```

```
%merge pl and pr and combine in parent
```

```
combine(Parent, Func, [Pl, Pr]) ->
```

```
    %wait for processes to complete (using receive) and then send to Parent
```

```
    receive
```

```
        { Pl, Sorted } -> combine(Parent, Func, Pr, Sorted);
```

```
        { Pr, Sorted } -> combine(Parent, Func, Pl, Sorted)
```

```
    end.
```

```
combine(Parent, Func, P, List) ->
```

```
    %wait and store in results and then call ! to send
```

```
    receive
```

```
        { P, Sorted } ->
```

```
            Results = Func(Sorted, List),
```

```
            case Parent of
```

```
                root ->
```

```
                    Results;
```

```
                %send results to parent
```

```
                _ -> Parent ! {self(), Results}
```

```
            end
```

```
    end.
```