2. ParallelDemo

February 16, 2014

Part I

Sample Codes

NOTE:

Each process has an associated identifier. The process providing the interactive julia prompt always has an id equal to 1, as would the julia process running the driver script in the example above. The processors used by default for parallel operations are referred to as workers. When there is only one process, process 1 is considered a worker. Otherwise, workers are considered to be all processes other than process 1.

The base Julia installation has in-built support for two types of clusters:

1. A local cluster specified with the -p option as shown above. 2. And a cluster spanning machines using the –machinefile option. This uses ssh to start the worker processes on the specified machines.

Functions addprocs, rmprocs, workers and others, are available as a programmatic means of adding, removing and querying the processes in a cluster.

```
In [1]: workers()
```

```
Out [1]: 1-element Array{Int64,1}:
          1
```

```
In [2]: procs()
```

```
Out [2]: 1-element Array{Int64,1}:
          1
```

```
In [3]: addprocs(8)
```

```
Out [3]: 8-element Array{Any,1}:
          2
          3
          4
          5
          6
          7
          8
          9
```

```
In [4]: r = remotecall(3, rand, 2, 2)
```

```
Out [4]: RemoteRef(3,1,4)
```

```
In [5]:  b=fetch(r)
Out [5]: 2x2 Array{Float64,2}:
          0.525251  0.775677
          0.440102  0.352892

In [6]:  s = @spawnat 2 rand(2,2)
Out [6]: RemoteRef(2,1,6)

In [7]:  fetch(s)
Out [7]: 2x2 Array{Float64,2}:
          0.243035  0.296231
          0.405337  0.618076

In [8]:  remotecall_fetch(2, rand, 2,2)
Out [8]: 2x2 Array{Float64,2}:
          0.668564   0.0206293
          0.0210516  0.774269

In [9]:  remotecall_fetch(2, getindex,r,1,1)
Out [9]: 0.5252510559469807

In [10]: r = @spawn rand(2,2)
Out [10]: RemoteRef(3,1,10)

In [11]: fetch(r)
Out [11]: 2x2 Array{Float64,2}:
          0.164833  0.311987
          0.604117  0.707194

In [12]: s = @spawn 1+fetch(r)
Out [12]: RemoteRef(3,1,12)

In [13]: fetch(s)
Out [13]: 2x2 Array{Float64,2}:
           1.16483  1.31199
           1.60412  1.70719

In [14]: a=RemoteRef[]
         for i in 1:10
             push!(a,@spawn sin(i))
         end

In [15]: @parallel (+) for i in a
           fetch(i)
         end
Out [15]: 1.4111883712180104

In [16]: sum(map(x->fetch(x),a))
Out [16]: 1.4111883712180104

In [21]: @everywhere function producer()
           for i in 1:3
             produce(i)
           end
         end
         task=Task(producer)
         consume(task)
Out [21]: 1

In [18]: consume(task)
Out [18]: 2
```

In [24]:
```
n=10;
@everywhere function ssin()
    for i in 1:n
        produce(sin(i))
    end
end
p=Task(ssin)
reduce(+,map(x->consume(p),1:n))
```

```
1.4111883712180104
```

Out [24]:

In [26]:
```
s=sin(1)
@time for i in 2:10000000
  s+=sin(i)
end
s
```

```
elapsed time: 2.701724951 seconds (319999968 bytes allocated)
1.9558914085412367
```

Out [26]:

In [27]:
```
@time @parallel (+) for i=1:10000000
  sin(i)
end
```

```
elapsed time: 0.476710806 seconds (517064 bytes allocated)
1.9558914085412054
```

Out [27]:

In [28]:
```
@time reduce(+,(map(i->sin(i),1:10000000)))
```

```
elapsed time: 3.724218083 seconds (879985408 bytes allocated)
1.9558914085412236
```

Out [28]:

In [29]:
```
@time reduce(+,[sin(i) for i in 1:10000000])
```

```
elapsed time: 2.61912771 seconds (400000032 bytes allocated)
1.9558914085412236
```

Out [29]:

In [30]:
```
@time sum( map(1:10000000) do x
          sin(x);
      end
)
```

```
elapsed time: 3.178096776 seconds (559984848 bytes allocated)
1.9558914085412236
```

Out [30]:

Part II

Data Movement

method 1 A = rand(1000,1000) Bref = @spawn A^2...$fetch(Bref)$

method 2 Bref = @spawn rand(1000,1000)^2...$fetch(Bref)$

Part III

Parallel Maps/Loops

In [47]:
```
@everywhere function count_heads(n)
    c::Int = 0
    for i=1:n
        c += randbool()
    end
    c
```

```
end
```

In [48]:
```
@time count_heads(200000000)/200000000
```

```
elapsed time: 0.688498406 seconds (159000 bytes allocated)
0.50000936
```

Out [48]:
```
#require("count_heads")
```

In [49]:
```
@time begin
a = @spawn count_heads(100000000)
b = @spawn count_heads(100000000)
(fetch(a)+fetch(b))/200000000
end
```

```
elapsed time: 0.395470371 seconds (21752 bytes allocated)
0.50006082
```

Out [49]:
```
@time begin
nheads = @parallel (+) for i=1:200000000
    int(randbool())
end
nheads/200000000
end
```

In [50]:
```
elapsed time: 0.168353892 seconds (526264 bytes allocated)
0.49999534
```

Out [50]:
```
@time begin
    a=0
    for i=1:200000000
      a+=int(randbool())
    end
    a/200000000
end
```

In [51]:
```
elapsed time: 8.610191092 seconds (3199988192 bytes allocated)
0.50003185
```

Out [51]:
```
@time sum(map(x->int(randbool()),1:200000000))/200000000
```

In [54]:
```
elapsed time: 22.731768799 seconds (7999985040 bytes allocated)
0.49999467
```

Out [54]:
```
b = zeros(100000);
@parallel for i=1:100000
  b[i] = i
end
```

In [55]:
```
b[1:10]  // assignment is local to the process in @parallel for
```

In [56]:
```
10-element Array{Float64,1}:
```

Out [56]:
```
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
```