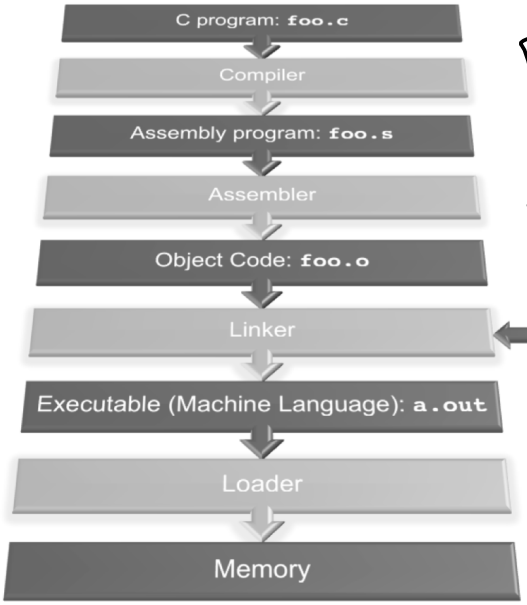
Jon Dis 7 Notes

CS 61C Fall 2018

CALL, WSC, MapReduce, Spark

Discussion 7: October 8, 2018

1 Compile, Assemble, Link, Load, and Go!



1.4 ← relative addr - within this program

abs addr, w/ other files too

1.1 What is the Stored Program concept and what does it enable us to do?

instructions are just data too
→ program can manipulate another program

1.2 How many passes through the code does the Assembler have to make? Why?

2: find labels
conv. instruction & forward references

1.3 What are the different parts of the object files output by the Assembler?

See solns - Header, Text, Data, Relocation Table, Symbol Table, Debug Info

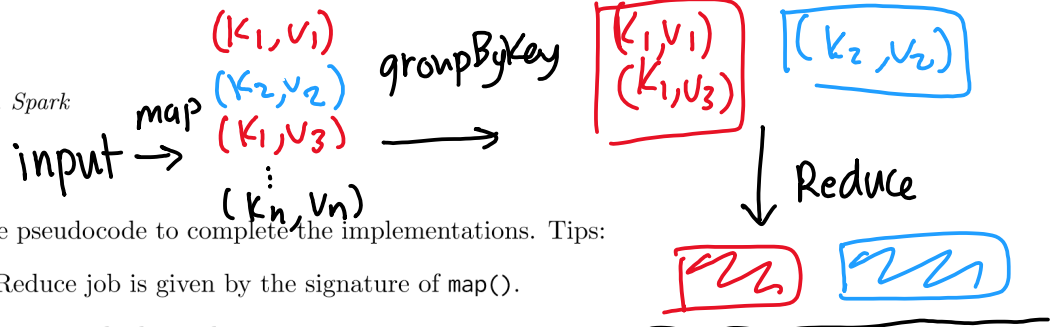
1.4 Which step in CALL resolves relative addressing? Absolute addressing?

Assembler Linker

1.5 What does RISC stand for? How is this related to pseudoinstructions?

Reduced Instruction Set Computing
- small set of basic instructions
pseudo inst are more complex, but get converted to basic inst.

2 MapReduce



For each problem below, write pseudocode to complete the implementations. Tips:

- The input to each MapReduce job is given by the signature of `map()`.
- `emit(key k, value v)` outputs the key-value pair `(k, v)`.
- `for var in list` can be used to iterate through `Iterables` or you can call the `hasNext()` and `next()` functions.
- Usable data types: **`int`**, **`float`**, `String`. You may also use lists and custom data types composed of the aforementioned types.
- `intersection(list1, list2)` returns a list of the intersection of `list1`, `list2`.

map: apply operation to all input (k,v)

reduce: combine vals of a key to produce output

2.1 Given a set of coins and each coin's owner, compute the number of coins of each denomination that a person has.

Declare any custom data types here:

```
CoinPair:
    String person
    String coinType
```

input of one usually matches output of step before

```
1 map(String Person, String coinType):
    Key = (Person, coinType)
    emit (key, 1)
```

matches

CoinPair — #of CoinPairs

```
1 reduce(CoinPair key, Iterable<Int> values):
    total = 0
    for count in values:
        total += count
    emit (key, total)
```

now a iterable of values (grouped by key) [val1, val2...]

2.2 Using the output of the first MapReduce, compute each person's amount of money. `valueOfCoin(String coinType)` returns a float corresponding to the dollar value of the coin.

matches

```
1 map(CoinPair Key, int total):
    emit( Key[0],
          valueOfCoin( Key[1] ) * total)
```

Can use key.person, but you usually see indices

```
1 reduce(String key, Iterable<float> values):
    total = 0
    for amt in values:
        total += amt
    emit (key, total)
```

person name — total money

3 Spark

Resilient Distributed Datasets (RDD) are the primary abstraction of a distributed collection of items

Transforms $RDD \rightarrow RDD$

`map`(f) Return a new dataset formed by calling f on each source element.

`flatMap`(f) Similar to map, but each input item can be mapped to 0 or more output items (so f should return a sequence rather than a single item).

`reduceByKey`(f) When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function f, which must be of type $(V, V) \to V$.

Actions $RDD \to Value$

`reduce`(f) Aggregate the elements of the dataset *regardless of keys* using a function f.

Call `sc.parallelize(data)` to parallelize a Python collection, `data`.

flow
map → reduceByKey
→ reduce
map → reduceByKey
→ map → reduce
⋮
etc.

3.1 Given a set of coins and each coin's owner, compute the number of coins of each denomination that a person has. Then, using the output of the first result, compute each person's amount of money. Assume `valueOfCoin(coinType)` is defined and returns the dollar value of the coin.

The type of `coinPairs` is a list of (person, coinType) pairs.

```
1 coinData = sc.parallelize(coinPairs)
```

```
out1 = coinData.map(lambda (k1,k2): ((k1,k2),1))
       .reduceByKey(lambda v1,v2: v1+v2)
out2 = out1.map(lambda (k,v): (k[0], v*valueOfCoin(k[1])))
       .reduceByKey(lambda v1,v2: v1+v2)
```

4 Amdahl's Law

In the programs we write, there are sections of code that are naturally able to be sped up. However, there are likely sections that just can't be optimized any further to maintain correctness. In the end, the overall program speedup is the number that matters, and we can determine this using Amdahl's Law:

$$\text{True Speedup} = \frac{1}{S + \frac{1-S}{P}}$$

where S is the Non-sped-up part and P is the speedup factor.

Notice
reduceByKey takes in values of (k,v) pair

4.1 You write code that will search for the phrases "Hello Sean", "Hello Jon", "Hello Dan", "Hello Man", "Bora is the Best!" in text files. With some analysis, you determine you can speed up 40% of the execution by a factor of 2 when parallelizing your code. What is the true speedup?

$$\frac{1}{0.6 + \frac{0.4}{2}} = \frac{1}{0.8} = 1.25$$

4.2 You are going to run your project 1 feature analyzer on a set of 100,000 images using a WSC of more than 55,000 servers. You notice that 99% of the execution of your project code can be parallelized on these servers. What is the speedup?

$$\frac{1}{0.01 + \frac{0.99}{55000}} \approx \frac{1}{0.01} = 100$$

wait for Availability, failure, etc. to get full picture

5 Warehouse-Scale Computing

Sources speculate Google has over 1 million servers. Assume each of the 1 million servers draw an average of 200W, the PUE is 1.5, and that Google pays an average of 6 cents per kilowatt-hour for datacenter electricity.

5.1 Estimate Google's annual power bill for its datacenters.

unit conversion

1.5×10^6 servers · 0.2kW/server · \$0.06/kW-hr · 8760 hrs/year = \$ 157.68 Mil/year

PUE — IT power

5.2 Google reduced the PUE of a 50,000-machine datacenter from 1.5 to 1.25 without decreasing the power supplied to the servers. What's the cost savings per year?

(1.5 − 1.25) · 50000 · 0.2 kW/server · \$0.06/kW-hr · 8760hrs/year

old PUE — new PUE — IT power (only for 50K servers we saved energy on)

6 MapReduce/Spark Practice: Optimize Your GPA

6.1 Given the student's name and course taken, output their name and total GPA.

Declare any custom data types here: CourseData

```
CourseData:
    int courseID
    float studentGrade // a number from 0-4
```

```
1 map(String student, CourseData value):
```

emit(student, value.studentGrade)

```
1 reduce(String key, Iterable<float> grades):
```

totPoints = 0
totClasses = 0
for g in grades:
    totPoints += g
    totClasses++
emit(key, totPoints/totClasses)

6.2 Solve the problem above using Spark.

The type of `students` is a list of (studentName, courseData) pairs.

```
1 studentsData = sc.parallelize(students)
2 out = studentsData.map(lambda (k, v): (k, (v.studentGrade, 1)))
```

1 ← # of classes for this 1 grade

.reduceByKey(sRed)
.map(sMap)

Tip
Always figure out what your inputs/outputs are

pair is (name, (points, classes))

let's use funcs — not lambdas

```
def sRed(v1, v2):
    return (v1[0]+v2[0], v1[1]+v2[1])
```

```
def sMap(pair):
    return (pair[0], pair[1][0]/pair[1][1])
```

7 MapReduce/Spark Practice: Optimize the Friend Zone

Hard Problem!

7.1 Given a person's unique int ID and a list of the IDs of their friends, compute the list of mutual friends between each pair of friends in a social network.

Declare any custom data types here:

```
FriendPair:
    int friendOne
    int friendTwo
```

```
1  map(_________________, _________________):
```

```
1  reduce(_________________, _________________):
```

~ See solutions for MapReduce part ~

7.2 Solve the problem above using Spark.

The type of persons is a list of (personID, list(friendID) pairs.

```
1  def genFriendPairAndValue(pID, fIDs):
2      return [((pID, fID), fIDs) if pID < fID else (fID, pID) for fID in fIDs]
3
4  def intersection(l1, l2):
5      return [x for x in b1 if x in b2]
6
7  personsData = sc.parallelize(persons)
```

genFPAV

inter

usually a sign is if your helper func returns a list

use flatMap since one input can produce more than 1 (K,V) pair

```
out = personData.flatMap(lambda (k,v) : genFPAV(k,v))
```

can also just do flatMap(genFPAV)

```
.reduceByKey(lambda v1,v2 : inter(v1,v2))
```

can also do reduceByKey(inter)