COMS 4115 Programming Languages and Translator

Ryan Decosmo (rd2680)
Olesya Medvedeva (oam2113)
Jyhyun Song (js4390)
Charis Lam (cl3257)

Language Proposal: MiniMap (MML)

Purpose:

MiniMap would be a functional language that aims to efficiently process medium to large text files by using the many processors on a local graphics card to parallel process the input information. There has already been some work done using OCaml to interface with graphics cards, but we are currently unsure if we will be implementing this at the OCaml level, as there are resources using LLVM (links below) that would also allow us to take advantage of the GPU. This would give us the option to add an additional step to our architecture when interfacing with the GPU.

https://developer.nvidia.com/cuda-llvm-compiler
https://llvm.org/docs/AMDGPUUsage.html

Intended Users:

Researchers and students who need to do some medium-large text processing quickly, but do not have access to a hadoop cluster.

Features:

Front end: users would work with an interface of Map, Reduce, and File objects, however they choose.

Back end: we will reduce and translate the MiniMap objects first into OCaml and then into code for the graphics card. Alternatively we may translate from OCaml to LLVM and then use Nvidia's LLVM CUDA compiler or the LLVM GPU API defined in the links above.

Structurally with MiniMap, we are thinking of borrowing ideas and syntax from Scala, specifically the "everything is an object" approach and the ability to use tuples. We think this could be an effective approach to parallelizing code.

Hopes and Dreams That Will Most Likely Get Crushed Along With The Rest Of The Proposal (aka. Goals):

Short term: to be able to write a program with MiniMap that processes text in a quantifiably faster time than programs written in other languages that process text.

Long term: (if this proposal is acceptable) we hope to construct MiniMap in such a way that eventually it could become open sourced and built upon by other developers. Currently our plan is to focus on text files, but this could potentially be extended to other file types and processing options.

Example Case:

A researcher is working with a large text or CSV file that they need to process on their local machine. However, unless they have access to a hadoop cluster and can deploy a MapReduce job, this can slow them down. On the other hand, if they do write a MapReduce and choose to deploy on a service like AWS, it could easily become expensive-- especially if it needs to be done repeatedly. With MiniMap, we hope to allow them to manipulate their data for free and with some reasonable speed (AWS would still be a quicker, though more expensive).

Sample Program:

```
object CountWordsExample {

//entry point into the program
  def main(args: Array[String]) {
        val processedOutPut = Reduce(Map(generateFile("/path/to/my/File.txt")))
        processedOutPut.save("/path/to/my/Save.txt");
}

//define how to split text file into chunks
—   def generateFile( file:File , delimiter:String ) : FileChunk = {
    parallelize(file.split(delimiter))    //parallelize the text file based on a delimited ex: \n,\r, # of Bytes
  }

//define what operations to perform on a chunk
—   def Map(chunk:FileChunk) : MiniMap(String,Int) = {
    val chunks = chunk.split("\\s")   //split on the regex for space
    for(chunk<-chunks){
        emit(miniMap(chunk,1)  //output a miniMap to be collected in their reducer
    }
  }

//define how to collect those miniMaps together
—   def Reduce( miniMaps:MiniMap(String, Int)) : Int= {
    var sum:Int = 0
    for(map<-miniMaps){
        sum+=map(1)
    }
    return sum
  }
```