CS533 Project Proposal: Map Reduction on Hetrogenous Systems

Abdul Dakkak, Carl Pearson, Liwen Chang

With the advent of personal digital devices, big data has become an issue faced not only by large companies but by regular users. Processing this data using traditional languages is both inefficient and sometimes not feasible. By their nature, for many tasks, computing on big data is a highly parallel and scalable and a multitude of solutions have been proposed to make writing programs to process big data more manageable. One of the most successful, in terms of deployment, is the Map-Reduce programming style (an example is Hadoop) which all companies employ in some way or form. The problem with this programming style is that many programming patterns cannot be easily mapped into it.

In this project we will look at what is needed from a compiler and architecture point of view to make computing with large data both efficient and practical. We will do so by having two new instructions in the compiler IR: `map` and `reduce`. These will map computation across cluster nodes, CPU cores, and, if time permits, GPU cores efficiently. Multiple components will be examined: good work distribution, efficient use of the CPU cache, a way to hide network traffic latency by interleaving computation and data transfer, and efficient use of the cache.

We will then develop a language and library that performs analytics on big data by expressing the computation in terms of these `map` and `reduce` instructions. We will also examine Our language/library will be inspired by data flow programming languages such as APL and frameworks like LINQ where one expresses computation based on operations on vectors and arrays. We will extend our APL/LINQ-like language by borrow the `DataFrame` idea for the R programming language — this an array of structures type data structure. To give you a taste of the language, in this program we approximate `pi` using Monte Carlo integration

```
def f(a : Real, b : Real) : Bool {
  return a*a + b*b < 1;
}

n : Integer = 1000;
as : []Real = rand.Real(n);
bs : []Real = rand.Real(n);
res : Integer = zip(as, bs).count(f) / n;
```

this would be translated into the map/reduce operations of

```
t1 = map(f, zip(as, bs));
count = 0;
reduce(\x : if (x) count += 1, t1);
res = count / n;
```

We plan on expressing all commonly used analytics operations such as sort, mean, max, min, histogram, variance, etc... in this framework.