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[1] programming style (an example is Hadoop[2]) 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 what language syntax sugare and compiler passes are needed to produce efficient parallel code.

Language Details

Our language/library will be inspired by array and data flow programming languages such as Fortran[3], APL[4], and LINQ[5] where one expresses computation based on operations on vectors and arrays. This defines two vectors of size 100

```
n :: Integer = 1000;
as :: []Real = rand.Real(n);
bs :: []Real = rand.Real(n);
```

We can then add the two vectors using an overloaded plus operator

```
res :: []Real = as + bs;
```

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;
}
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++, t1);
res = count / n;
```

We will extend our APL/LINQ-like language by borrow the `DataFrame` idea for the R[6] programming language — this an array of structures type data structure. We plan on expressing all commonly used analytics operations such as sort, mean, max, min, histogram, variance, etc... in this framework.

Compiler Pass

The most important factor in distributed computing is how to manage memory transfer. If a node computes a chunk of data and it is used in subsequent instructions, then it should reuse the output rather than send and request the data again. There are two approaches to facilitate this. The first is a runtime approach: Hadoop, for example, dispatches tasks to maximize reuse of local data. This done via the Hadoop scheduler which has a mapping between nodes and data state.

The second is a compiler transformation. This is mainly done via loop fusion. If for example, one writes a program `map(f, map(g, lst))` then a compiler pass can transform this into `map(f g, lst)`. A simple peephole optimizer can scan for this instruction pattern and perform this transofmration. A generalization of this technique for other list primitives is found in the the Haskell vector library. Using a concept called Stream Fusion[7], Haskell fuses most function loops to remove unecessary temporaries and list traversals. In this project, we will look at how Haskell performs this transformation and how applicable it is in a non-shared memory model.

Deliverable

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Proceedings of the 6th conference on symposium on opearting systems design & implementation - volume 6*, Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10.

[2] P. Zikopoulos, C. Eaton, and others, *Understanding big data: Analytics for enterprise class hadoop and streaming data*, McGraw-Hill Osborne Media, 2011.

[3] J.C. Adams, W.S. Brainerd, and C.H. Coldberg, *Programmer's guide to fortran 90*, Intertext Publications, 1990.

[4] R.P. Polivka and S. Pakin, *APL: The language and its usage*, Prentice Hall Professional Technical Reference, 1975.

[5] E. Meijer, B. Beckman, and G. Bierman, "Linq: reconciling object, relations and xml in the. net framework," *Proceedings of the 2006 aCM sIGMOD international conference on management of data*, ACM, 2006, pp. 706–706.

[6] R.C. Team and others, "R: A language and environment for statistical computing," *R foundation for Statistical Computing*, 2005.

[7] D. Coutts, R. Leshchinskiy, and D. Stewart, "Stream fusion: From lists to streams to nothing at all," *ACM sIGPLAN notices*, ACM, 2007, pp. 315–326.