CS 744: SPLIT ANNOTATIONS

Shivaram Venkataraman
Fall 2020
Course Project Checkins – due tomorrow!
In-class project presentations
  Dec 8\textsuperscript{th} and Dec 10\textsuperscript{th}
  Sign up sheet on Piazza
NEW HARDWARE AND DATA MODELS
Multi-core machines

Multiple functions and libraries

// inputs are double arrays with `len` elems
vdLog1p(len, d1, d1); // d1 = log(d1)
vdAdd(len, d1, tmp, d1); // d1 = d1 + tmp
// d1 = d1 / vol_sqrt
vdDiv(len, d1, vol_sqrt, d1);
COMPILER-BASED APPROACHES

Replace every library call to emit intermediate representation (IR)

Compile all the IR together

Lots of code change required!
GOALS

Provide data movement optimizations across libraries

Require minimal or no changes to existing libraries

Leverage existing hand-tuned code for speedups
d1 = price * strike

\[ d1 = \text{np.log2}(d1) + \text{strike} \]
@splittable(
    size: SizeSplit(size), a: ArraySplit(size),
    mut out: ArraySplit(size))
void vdLog1p(long size, double*a, double*out)

Split types: N⟨V0...Vn⟩ e.g.: ArraySplit(10, 2) for 10 element array, 2 pieces
Split annotation:
  Name and split type to each argument and return value
IMPLEMENTING SPLIT API

NameConstructor(A0,...An) => Parameters

Split(D arg, int start, int end, Parameters) => D

Merge(Vector<D>, Parameters) => D

@splittable(m:MatrixSplit(m, axis), axis:_)
    -> ReduceSplit(axis)
vector sumReduceToVector(matrix m, int axis);
User Application

\[
y = \text{lib}.f();
\]
\[
z = \text{lib}.g(y);
\]

Annotations (§3)

Wrapped Library

Mozart Client Library (§4)

- Construct task graph
- Decide when to execute graph

Mozart Runtime (§5)

- Split and pipeline data, execute functions in parallel

T1  T2  T3
Writing Annotations: Function decorators

```python
@sa((DataFrameSplit(), DataFrameSplit()), {}, DataFrameSplit())
def divide(series, value):

Capturing the graph
  Wraps original Python function and registers in graph
  Returns a Future object

Evaluation Points
  Lazily evaluate by overriding __getattr__
```
MOZART RUNTIME

Take dataflow graph $\rightarrow$ execution plan

Series of **stages** each stage split, pipeline and merge

Choosing a batch size

Set number of elements per batch using L2 cache size
SUMMARY

Applications compose data processing libraries
Data movement is bottleneck on multi-core machines

Key idea: Split and pipeline data across functions

Split Annotations to reduce programmer effort
Mozart: Client library and runtime for lazy evaluation
DISCUSSION

https://forms.gle/F2LJ21qFkBGWyyypB7
How does the dataflow graph that is executed by Mozart compare to dataflow graphs we have seen in other systems like Spark/PyTorch etc.
(a) Relative Intensity

(b) Speedup over no SAs
NEXT STEPS

Next class: TPU
Project check-ins on HotCRP!