Lab 5: Data-Driven Tasks

Instructor: Vivek Sarkar

1 Update your HJ/DrHJ Installation

The performance measurements for today's lab should be done on Sugar, and we've already updated the HJ installation there. (See Lab 4 handout on instructions to access the HJ installation in the COMP 322 userid on Sugar.)

However, if you're also working with a local installation, please update it from the HJ download page, `https://wiki.rice.edu/confluence/display/PARPROG/HJDownload`, to make sure that you have the latest updates and bug fixes.

2 Matrix Expression Language

We have provided a sequential program, `MatrixEval.hj`, to evaluate matrix expressions consisting of the following terms and operators:

- The only leaf terms supported are identifiers which can be of two forms:

  Identity Matrix: An identifier of the form $m\langle num1\rangle$ represents a square identity matrix of size $\langle num1\rangle \times \langle num1\rangle$. For example, *m100* represents the 100×100 identity matrix. (The expression language has no variable declarations, so there's no significance to the name m other than the fact that it denotes a matrix.)

  Random Matrix: An identifier of the form $m\langle num1\rangle x\langle num2\rangle s\langle seed\rangle$ represents a random matrix of size $\langle num1\rangle \times \langle num2\rangle$, for which the elements are generated using `java.util.Random` starting with an integer (long) *seed*, and calling `nextInt()` to generate successive elements of the matrix. For example, *m100x200s5* represents the 100×200 random matrix generated using 5 as the initial seed.

- The + operator represents matrix addition. An exception is thrown if the matrices don't have the same dimension sizes i.e., if they are not conformable. Otherwise, the matrix sum is returned.
- The − operator represents matrix subtraction. An exception is thrown if the matrices don't have the same dimension sizes i.e., if they are not conformable. Otherwise, the matrix difference is returned.
- The ∗ operator represents matrix multiplication. An exception is thrown if the number of columns in the first matrix operand does not equal the number of rows in the second matrix operand i.e., if they are not compatible for matrix multiplication. Otherwise, the matrix product is returned.
- Usual precedence and evaluation rules apply for the above operators, and parentheses can also be used.

As an example, "*m3 + m3 * m3*", will be evaluated as follows:

$$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} + \begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} \times \begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} = \begin{bmatrix}2&0&0\\0&2&0\\0&0&2\end{bmatrix}$$

3 Recap of Data-Driven Tasks

Data-driven tasks were covered in lecture 8. To use this feature, be sure to include the following import statement at the start of your program: "`import hj.lang.DataDrivenFuture;`"

This extension is enabled by adding an `await` clause to the async statement as follows:

$$\texttt{async await } (DDF_a, DDF_b, \cdots) \ \langle \ \texttt{statement} \ \rangle$$

Each of $DDF_a, DDF_b, \cdots$ is an instance of the standard `DataDrivenFuture` class in HJ. A DDF acts as a container for a single-assignment value, like regular `future` objects. However, unlike `future` objects, DDF's can be used in an `await` clause, and any async task can be a potential producer for a DDF (though only one task can be the actual producer at runtime because of the single-assignment property).

The example HJ code fragment in Figure 1 shows five logically parallel tasks and how they are synchronized through DDFs. Initially, two DDFs are created as containers for data items `left` and `right`. Then a `finish` is created with five `async` tasks. The tasks, `leftReader` and `rightReader`, include `left` or `right` in their `await` clauses respectively. The fifth task, `bothReader`, includes both both `left` and `right` in its `await` clause. Regardless of the underlying scheduler, the first two `async`s are guaranteed to execute before the fifth `async`.

```
DataDrivenFuture left = new DataDrivenFuture();
DataDrivenFuture right = new DataDrivenFuture();
finish { // begin parallel region
    async left.put(leftBuilder()); // Task_1
    async right.put(rightBuilder()); // Task_2
    async await (left) leftReader(left.get()); // Task_3
    async await (right) rightReader(right.get());// Task_4
    async await (left, right) bothReader(left.get(), right.get());  //Task_5
}  // end parallel region
```

Figure 1: Example Habanero Java code fragment with Data-Driven Futures.

Another example of DDFs can be found in `DDFEx.hj` on the course web site as an example program for Lab 5. The "`async await`" data-driven-task construct is a lower-overhead optimized variant of the standard future task construct because the get() operations are guaranteed to be non-blocking when accessed in a corresponding "`async await`" task.

4 Parallelizing MatrixEval using Data-Driven Tasks

The code in `MatrixEval.hj` parses the input expression, and then calls the `eval()` methods to evaluate the expression. The major potential for parallelism is in the `eval()` method in class `Binary`, shown in Listing 1. Given the semantics of expression evaluation, the calls to `lft.eval()` and `rgt.eval()` can execute in parallel.

Your assignment today is to use the `async await` feature in HJ to parallelize the evaluation of these two calls using *data-driven tasks (DDTs)* and *data-driven futures (DDFs)* (Lecture 8). HJ's `DataDrivenFuture` class now accepts type parameters, so you can use the `DataDrivenFuture<MatrixEval.Matrix>` type for DDFs in this assignment.

WARNINGS:

1. *You may need to modify method call interfaces e.g., adding a DDF parameter to eval(), to complete this assignment.*

2. *Be sure to add "break;" statements in "switch" statements if needed.*

You should run your program on SUGAR, to evaluate the parallelization. As before, you can compile the program as follows, after repeating the setup from Lab 4:

```
hjc MatrixEval.hj
```

To run the program, use the following command on a compute node (obtained using the "qsub -I ..." command discussed in Lab 4):

```
hj -places 1:8 MatrixEval test.txt
```

where `test1.txt` is a text file containing the input expression. What speedups do you see with parallelization?

You're welcome to test your code with other input expressions, both for correctness (with small matrices) and for performance (with larger matrices). There is a `PrintMatrix()` method included that you may choose to use when debugging your code with small inputs such as `test0.txt`.

```
1         public MatrixEval.Matrix eval() {
2             switch (opr) {
3             case Lexical.plus:
4                 return MatrixEval.matrixAdd(lft.eval(), rgt.eval());
5             case Lexical.minus:
6                 return MatrixEval.matrixMinus(lft.eval(), rgt.eval());
7             case Lexical.times:
8                 return MatrixEval.matrixMultiply(lft.eval(), rgt.eval());
9             default:
10                error("Unhandled binary operator");
11            }
12            return null;
13        }
```

Listing 1: Sequential implementation of eval() method in class Binary