M^2

Shelley Zhong
Tengyu Zhou
Christine Pape
Jeffrey Monahan
Montana St. Pierre

Motivation

- Matrix workloads increasingly common and complex
- Existing languages sacrifice performance, readability, or both
- Target matrix based problems with lightweight, familiar, and easy to learn syntax

Overview

PROJECT TIMELINE	
SEP 8 – SEP 27	Language proposal – ideas for language, heavily focused on matrix specifics
SEP 28 – OCT 16	LRM – nailed down details, fleshed out the rest of the elements to form a complete language
OCT 17 – NOV 8	Hello World – Get the bare minimum working
NOV 8 - DEC 1	Team Milestone 1 – get matrix fully working as a datatype
DEC 1 – DEC 8	Team Milestone 2 - work out semantic issues and make final edits to our language specification
DEC8 - END	Final Presentation/Delivery – implement last functions and ensure the project is thoroughly tested

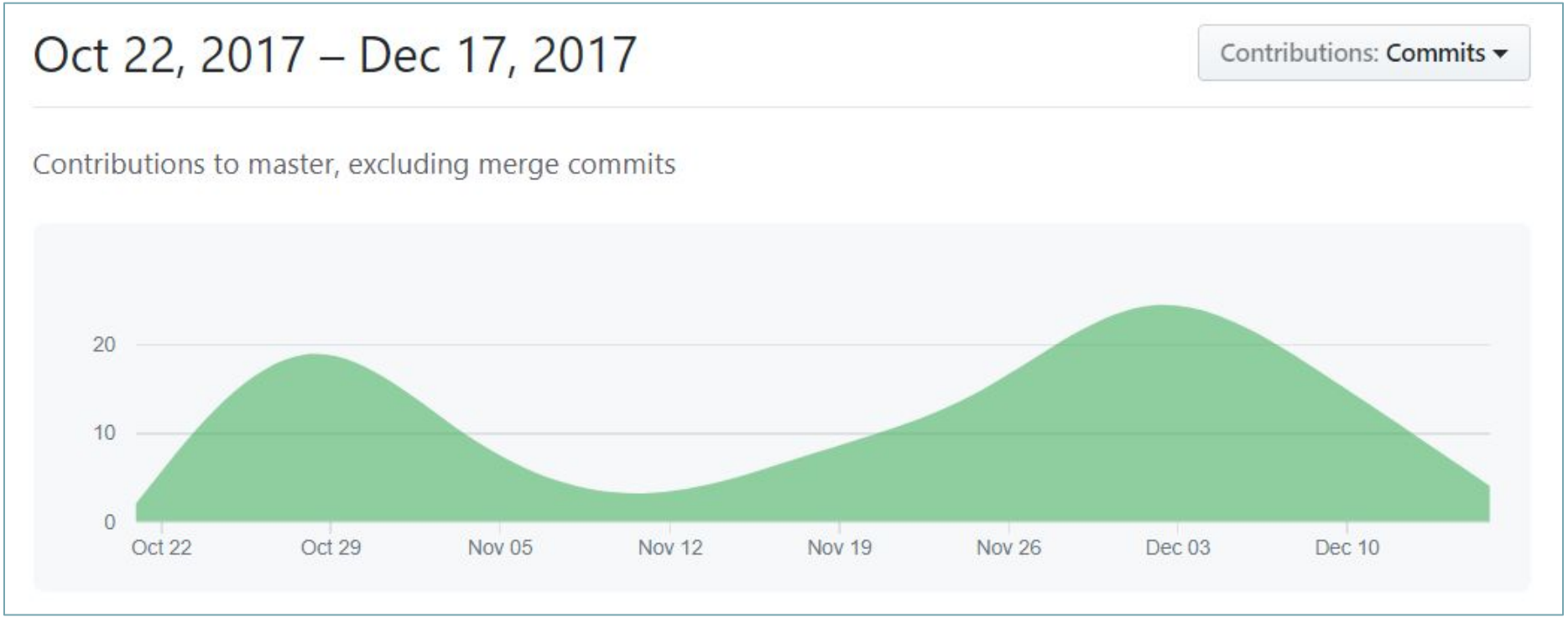
Project specifications

- Language: OCaml
- Code generation: LLVM
- Version Control: Git

Team Dynamics

- Communication: Facebook Messenger
- Fixed weekly group meetings
- Remote collaboration through google drive
- Additional meetings as necessary

Project Log


Oct 22, 2017 – Dec 17, 2017
Contributions: Commits
Contributions to master, excluding merge commits
20
10
0
Oct 22
Oct 29
Nov 05
Nov 12
Nov 19
Nov 26
Dec 03
Dec 10

Syntax - Types and Operators

Basic Syntax:

Variable Declaration: We have six types within our language - void, boolean, int, float, String, and matrix. To declare and instantiate a variable, one must first declare the variable name and its type. Once the name and type are declared, one may then assign the variable,

```
int i;
i = 1;          ← end of lines are declared by semicolons
```

Pointers? Easy, we don't have them.

All code which will be executed is stated within the main() function. main() must have the return type int and the last line of the function should be "return 0;". The main() function may be declared at any position in a source file.

Operators: + "addition", - "subtraction", * "multiplication", / "division, == "comparison", < "less than", > "greater than", <= "less than or equal", >= "greater than or equal", != "not equal"

No explicit arrays; though, our matrix implementation is very similar.

Syntax - Control Flow

- if (condition) {} else {}
- for (init-expr; condition; itr-expr)
- while (condition) {}

Example for loop:
```
int i;
for (i = 0; i < 100; ++i) {}
```
← variable id and type must be stated before assignment

Example while loop:
```
int i;
i = 0;
while (i < 100) {
    ++i;
}
```
← Incrementation operator, same as writing i = i + 1;. Decrementing is the same process (a.k.a --i is the same as writing i = i - 1;)

Interesting aside: A for loop that only has a conditional statement operates exactly like a while loop.

Syntax - Functions

Function Declaration:

Functions are defined in the following format - return type, function name, opening parenthesis, parameters, closing parenthesis and, lastly, the function body.

Function example:

```
void foo() {
        int i;
        for (i = 0; i < 10; ++i) {
                printInt(i);
                }
        }

int main() {
        foo();
        return 0;
}
```

← variable type/id must be declared before assignment
← highlight denotes the scope of variable “i”
← example of one of our print functions, the others are printStr(“”), which is similar to println() in java, printFloat(), and printBool()

← example of “return 0” at the end of main() indicating that the that the program executed successfully.

Syntax - Matrices

Matrices:

Defined just like any other variable, in that they type and id must be declared before assignment; however, along with the type and id, the dimensions of the matrix must be declared as well. The order in this declaration is as follows:

```
matrix matrix_type [r][c] var_id;      ← r = number of rows, c = number of columns
var_id = [[11, 12, 13..., 1c];...[r1, r2, r3..., rc]];
```

Example:

```
matrix int [2][2] m;
m = [[1, 0]; [0, 1]];
```

Matrix Operators: + “matrix addition”, - “matrix subtraction, * “matrix multiplication”. The other operators (e.g. less than, greater than, comparison) do not apply to matrices.

Syntax - Matrices (cont.)

Vectors and Built-in Matrix Functions:

In M^2, vectors and matrices are tied into one type. To define a vector, one need only instantiate a 1-dimensional matrix.

Example:

```
matrix int [3][1] vector;
vector = [[1]; [0]; [0]];
```

M^2 has five built-in matrix functions. These are functions to calculate the number of rows, the number of columns, transpose, trace, and submatrix of a

Example:

m

```
atrix int [3][3] m = [[1, 2, 3]; [4, 5, 6]; [7, 8, 9]];
m:rows;                                          ← return type int
        m:cols;                                  ← return type int
        m:transpose;                             ← return type matrix
        m:trace;                                 ← return type int/float
m:subMatrix(index1, index2, index3, index4);     ← return type matrix
```

Syntax - Matrix Usage

Simple Matrix Program for Inverse:

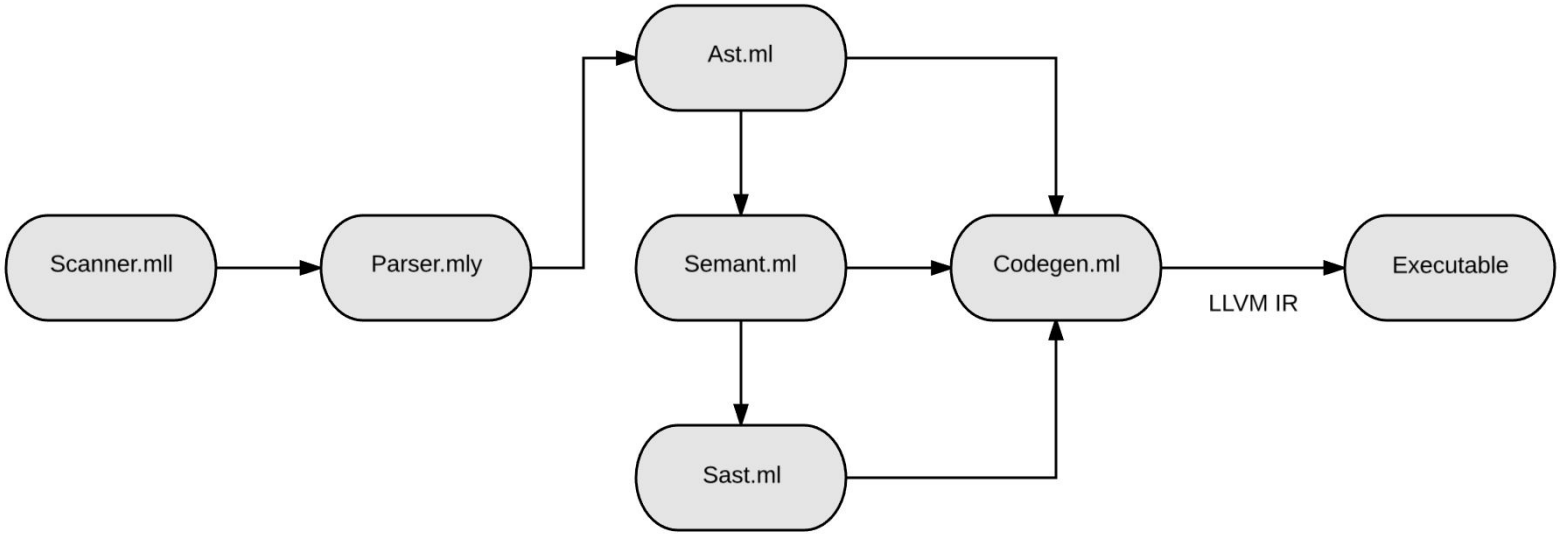
Accessing individual elements of a matrix is similar to accessing an array in C (index starts at zero and goes until row_num - 1 and column_num - 1). The first index is the row index and the second is the column index (i.e. m[0][1] accesses the element of the matrix belonging to the first row and column).

Main comes first in this example but, again, order does not matter.

Example program, which was written by Siyü for testing.

```
int main(){
        matrix float [2][2] m;
        matrix float [2][2] inverse;
        float det;

        m = [[1.0,2.0];
                [3.0,4.0]];

        det = m[0][0] * m[1][1] - m[1][0] * m[0][1];

        if(det != 0.0){
                inverse[0][0] = m[1][1] / det;
            inverse[0][1] = - m[0][1] /det;
            inverse[1][0] = - m[1][0] /det;
            inverse[1][1] = m[0][0] / det ;
            printStr("The inverse of matrix");
                myprint(m);
                printStr("is");
                myprint(inverse);
        }
        else{
                printStr("Cannot find inverse");
        }

        return 0;
}

void myprint(matrix float [2][2] M){
  int i;
  int j;
  for(i = 0; i < M:rows; ++i){
    for(j = 0; j < M:cols; ++j){
      printFloat(M[i][j]);
    }
    printStr("");
  }
}
```

Architecture



Testing

- Unit test (control flow, operators, expression, etc.)
- Integration test
- Fail test: fail-[test_name].m2 and fail-[test_name].err
- Pass test: test-[test_name].m2 and fail-[test_name].out
- 41 fail tests and 57 pass tests

```
int main(){
    int i;
    i = 2;
    for(;i<3;){
        i=i+1;
        printInt(i);
    }
    return 0;
}
```

test-for.m2

```
int main() {
  matrix float [2][2] m;
  m = [[2.0,1.0];
       [0.0,3.0]];

  printStr("Matrix m is");
  myprint(m);

  printStr("m + m is");
  myprint(m + m);

  printStr("m - m is");
  myprint(m - m);

  printStr("m * m is");
  myprint(m * m);

  return 0;
}

void myprint(matrix float [2][2] M){
  int i;
  int j;
  for(i = 0; i < M:rows; ++i){
    for(j = 0; j < M:cols; ++j){
      printFloat(M[i][j]);
    }
    printStr("");
  }
}
```

test-floatMatrixBinop1.m2

Testing

- Automation: testall.sh
- Run ./testall.sh
- Output .ll and .out files

```
-n fail-duplicateGlobal...
OK
-n fail-equality...
OK
-n fail-functionArgument...
OK
-n fail-logical...
OK
-n fail-matrixAccess...
OK
-n fail-matrixBinop...
OK
-n fail-matrixBinop2...
OK
-n fail-matrixBinop3...
OK
-n fail-matrixBinop4...
OK
```

```
-n test-var1...
OK
-n test-var2...
OK
-n test-while1...
OK
-n test-while2...
OK
-n fail-assign1...
OK
-n fail-assign2...
OK
-n fail-assign3...
OK
-n fail-expr1...
OK
-n fail-expr2...
OK
```

Lessons Learned

Tengyu: Ocaml is so hard to write. 1000 lines of code seems to be easier than 100 lines. And prepare yourself everyday so that you can handle all the sudden changes!

Jeff: Sounds very cheesy, but ALWAYS tell the truth. You have to overcome problems/hurdles as a group, and it won't work if you're dishonest with each other. That means that you can't always spare people's feelings and that you certainly can't let things just sit on the backburner.

Shelley: Teammates can be a great source of inspiration! This is a group project, so talk to each other, work collaboratively and have fun. Be prepared to suffer and learn Ocaml and llvm :)

Lessons Learned (cont.)

Christine: It’s very easy to start off strong in the beginning and then fall off towards the middle of the semester and not even realize it. Be sure to evaluate your progress regularly as a team to try and stay on track!

Montana: Always contribute ideas even if they aren’t entirely formed as they can inspire valuable discussion and contemplation. Additionally, don’t be demure and fear testing a new idea, because a good team is always there to support you.

Demo!

Solve System of Linear Equations

- x + y + z = 3
- 2x + 3y + z = 6
- x + y + 5z = 7

```
[[1  1  1];          [[x];          [[3];
[2  3  1];     *     [y];     =     [6];
[1  1  5]]           [z]]           [7]]
```

Solve System of Linear Equations

```
int main() {
  matrix float [3][3] m;
  matrix float [3][1] m2;
  matrix float [3][3] tr;
  matrix float [3][3] adj;
  matrix float [3][3] inverse;
  matrix float [3][1] result;
  float det;
  int i;
  int j;

  m = [[1.0, 1.0, 1.0];
       [2.0, 3.0, 1.0];
       [1.0, 1.0, 5.0]];

  m2 = [[3.0];
        [6.0];
        [7.0]];

  tr = m:transpose;

  det = m[0][0] * (m[1][1] * m[2][2] - m[1][2] * m[2][1])
    - m[0][1] * (m[1][0] * m[2][2] - m[1][2] * m[2][0])
    + m[0][2] * (m[1][0] * m[2][1] - m[1][1] * m[2][0]);
```

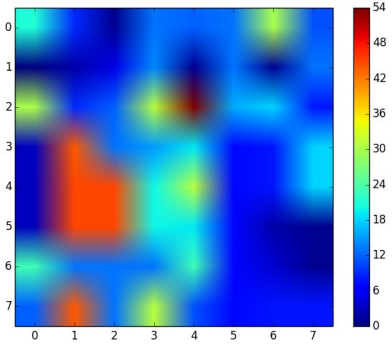
```
  if (det != 0.0){
    adj[0][0] = tr[1][1]*tr[2][2] - tr[1][2]*tr[2][1];
    adj[0][1] = tr[1][0]*tr[2][2] - tr[1][2]*tr[2][0];
    adj[0][2] = tr[1][0]*tr[2][1] - tr[1][1]*tr[2][0];
    adj[1][0] = tr[0][1]*tr[2][2] - tr[0][2]*tr[2][1];
    adj[1][1] = tr[0][0]*tr[2][2] - tr[0][2]*tr[2][0];
    adj[1][2] = tr[0][0]*tr[2][1] - tr[0][1]*tr[2][0];
    adj[2][0] = tr[0][1]*tr[1][2] - tr[0][2]*tr[1][1];
    adj[2][1] = tr[0][0]*tr[1][2] - tr[0][2]*tr[1][0];
    adj[2][2] = tr[0][0]*tr[1][1] - tr[0][1]*tr[1][0];


    adj[0][1] = adj[0][1] * (0.0-1.0);
    adj[1][0] = adj[1][0] * (0.0-1.0);
    adj[1][2] = adj[1][2] * (0.0-1.0);
    adj[2][1] = adj[2][1] * (0.0-1.0);

    for(i = 0; i < inverse:rows; ++i){
      for (j = 0; j < inverse:cols; ++j){
        inverse[i][j] = adj[i][j] / det;
      }
    }

    result = inverse * m2;
```

Planet Simulation Based on Matrix

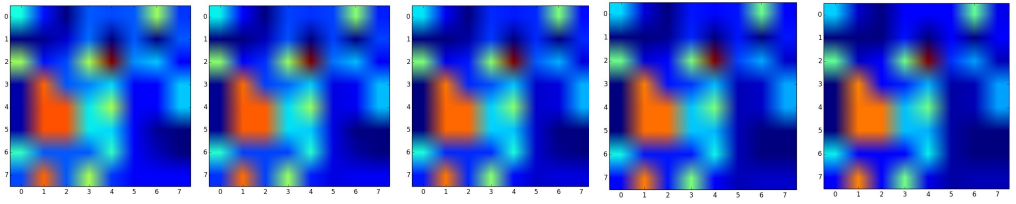
We can develop algorithms to simulate real world phenomenon using our language! Supposing we try to simulate mass transfer of planets. Heavier planet tends to attracts more mass.

```
int main() {
    matrix int [8][8] m;
    int i;
    int j;
    int k;
    int sum;
    int sumM;
    int new;
    m = [[21,9,1,13,12,13,30,11];
        [0,2,5,14,1,13,1,13];
        [30,9,12,31,54,16,18,8];
        [3,44,13,15,19,7,8,18];
        [3,45,45,20,31,7,8,18];
        [3,45,45,20,19,7,2,1];
        [23,13,13,13,23,7,4,1];
        [12,44,13,31,11,7,8,8]
        ];
    myprint(m);
    for(i = 0; i < 6; ++i) {
        sum = 0;
        sumM = 0;
        for(j = 0; j < 8; ++j) {
            for(k = 0; k < 8; ++k) {
                if(m[j][k] > 0) {
                    m[j][k] = m[j][k] - 1;
                    ++sum;
                }
                sumM = sumM + m[j][k];

            }
        }
        for(j = 0; j < 8; ++j) {
            for(k = 0; k < 8; ++k) {
                new = sum * m[j][k] / sumM;
                m[j][k] = m[j][k] + new;
            }
        }
        myprint(m);
        printStr("");

    }
    return 0;
}
```

Planet Simulation Based on Matrix



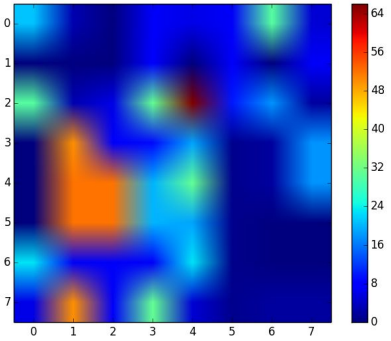
[[21,9,1,13,12,13,30,11],
[0,2,5,14,1,13,1,13],
[30,9,12,31,54,16,18,8],
[3,44,13,15,19,7,8,18],
[3,45,45,20,31,7,8,18],
[3,45,45,20,19,7,2,1],
[23,13,13,13,23,7,4,1],
[12,44,13,31,11,7,8,8]]



[[21,8,0,12,11,12,30,10],
[0,1,4,13,0,12,0,12],
[30,8,11,31,56,15,18,7],
[2,45,12,14,19,6,7,18],
[2,46,46,20,31,6,7,18],
[2,46,46,20,19,6,1,0],
[23,12,12,12,23,6,3,0],
[11,45,12,31,10,6,7,7]]

[[21,6,0,10,9,10,30,8],
[0,0,2,11,0,10,0,10],
[30,6,9,31,60,13,18,5],
[0,47,10,12,19,4,5,18],
[0,48,48,20,31,4,5,18],
[0,48,48,20,19,4,0,0],
[23,10,10,10,23,4,1,0],
[9,47,10,31,8,4,5,5]]

[[21,4,0,8,7,8,30,6],
[0,0,0,9,0,8,0,8],
[30,4,7,31,64,11,18,3],
[0,49,8,10,19,2,3,18],
[0,50,50,20,31,2,3,18],
[0,50,50,20,19,2,0,0],
[23,8,8,8,23,2,0,0],
[7,49,8,31,6,2,3,3]]



[[21,3,0,7,6,7,30,5],
[0,0,0,8,0,7,0,7],
[30,3,6,31,66,10,18,2],
[0,50,7,9,19,1,2,18],
[0,52,52,20,31,1,2,18],
[0,52,52,20,19,1,0,0],
[23,7,7,7,23,1,0,0],
[6,50,7,31,5,1,2,2]]

Planet Simulation Based on Matrix