TENLAB

Everything is Tensor!!!
Team Member

Xiangrong Xu
Xinchen Xie
Songqing Ye
Senhong Liu

Special thanks to Hao Zhao and Stephen Edwards
Language Intro

TENLab is an imperative, dynamically typed language inspired by Python and Matlab.

- Parallel functions to support MapReduce model in distributed system.
- Flexible tensor data type to represent all kinds of data
- Type-inference allows for concise representation of data
- Various built-in functions for matrix operations
- Automatic garbage collection (reference count analysis)
Motivations

Everything is tensor

Matlab: expensive, not lightweight enough

(tensor) Matrix multiplication -> parallel (easy parallel interface to user)

Make an easy, fast and flexible language
Tensor Layout

LLVM

{  
  i8
  i8
  i8
  i64 *
  i8 *
}

C

struct tensor {
  int8_t type,
  int8_t ndim,
  int8_t refcount
  int64_t *shape
  void *data
};

a = [[[1,2],[3,4],[5,6]]];

type: 0
ndim: 2
refcount
shape
data

Scalar like b = 2.4?
→ 0-dimensional tensor
→ ndim=0; shape=NULL;

0: int32
1: float64
2: char8
Tensor Check

Static Check — check in semantic

Inconsistent dimension

[ [1, 2.1], 3 ]

Inconsistent type

Dynamic Check — check in runtime

a + b

a.type == b.type?

a.ndim == b.ndim?

a.shape[...] == b.shape[...]?
Q: How to implement various data structures?
A: USE vartensor!

```plaintext
a = var[[1,2],3.1];
```

- **a**
  - **type:** 3
  - **ndim:** 1
  - **refcount:** 2
  - **shape:**
  - **data:**
    - `ptr to [1,2]`
    - `ptr to 3.1`
Use vartensor to implement binary tree

```
1  #[left, value, right]
2  tree = var [nil, 0, nil];
3  tree[0] = var [var [nil, 1, nil], 2, var [nil, 4, var [nil, 5, nil]]];
4  tree[2] = var [nil, 6, nil];

5  def preorder(t) {
6      if (t != nil) {
7          print(t[1]);
8          preorder(t[0]);
9          preorder(t[2]);
10     }
11     return 1;
12  }
13
14  print("Pre-Order");
15  preorder(tree);
```

Result:

```
0
 [ CPUIntType{} ]
2
 [ CPUIntType{} ]
1
 [ CPUIntType{} ]
4
 [ CPUIntType{} ]
5
 [ CPUIntType{} ]
6
 [ CPUIntType{} ]
```

Pre-Order:

```
0
2
1
4
5
6
```
User Define Parallel Environment-Syntax

```c
parallel_define <environment_name> {  
  overload <operator_name> (X, Y) {  
    map <name_of_function_one> {  
      statements;  
      return <some_variable>;  
    }  
    map <name_of_function_two> {  
      statements;  
      return <some_variable>;  
    }  
    reduce {  
      return <some_variable>;  
    }  
  }  
}
```

```c
parallel_define EnvironmentTest {  
  overload __+_ (x, y) {  
    map f1 {  
      z = x[0:2:1] + y[0:2:1];  
      return z;  
    }  
    map f2 {  
      z = x[2:4:1] + y[2:4:1];  
      return z;  
    }  
    reduce {  
      return cat(f1, f2, 0);  
    }  
  }  
}
```
User Define Parallel Environment-Syntax

```ruby
parallel_define <environment_name> { 
  overload <operator_name> (X, Y) { 
    map <name_of_function_one> { 
      statements;
      return <some_variable>;
    } 
    map <name_of_function_two> { 
      statements;
      return <some_variable>;
    } 
  }
  using <environment_name>;
  statements;
end <environment_name>;
```

```ruby
using EnvironmentTest;
a = b + c;
# end EnvironmentTest;
```
Architecture

- Source file → Scanner → Tokens → Parser → AST → Semantic Check
  - SAST
  - Code Generation
  - LLVM
  - Executable
  - Linking
  - C++ library
Built-in Functions

Rich built-in functions to make programming in TENLab friendly

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>print(x)</td>
<td></td>
</tr>
<tr>
<td>int_of(x)</td>
<td></td>
</tr>
<tr>
<td>cat(x,y,</td>
<td></td>
</tr>
<tr>
<td>axis)</td>
<td>float_of(x)</td>
</tr>
<tr>
<td>shape(x)</td>
<td>floor(x)</td>
</tr>
<tr>
<td>ones(x)</td>
<td>ceil(x)</td>
</tr>
<tr>
<td>zeros(x)</td>
<td>round(x)</td>
</tr>
<tr>
<td>rand(x)</td>
<td>abs(x)</td>
</tr>
<tr>
<td>sum(x)</td>
<td></td>
</tr>
</tbody>
</table>

# Random Init
w = rand([8192]);
b = 0.;

# Mean Square Error Loss function
def loss() {
    y_test = x_train * w + b;
    error = sum((y_test - y_train).^2 .* 0.5);
    return error / float_of(num_of_dimensions);
}
Test Suite

Divide into 5 categories and each one is responsible for a particular feature (tensor-test, built-in-test, stmt-test, pe-test, and fails)

Automated test script to compare sample program output with *.out file.
  test-*: Success tests print output to .out files and compare this to expected output
  fail-*: Fail tests print error messages to .err files and compared to expected error messages

Generally create one new test for each new feature or commit

About 72 tests and 4 demos in the final repository
Future work

More tensor features: gradient, loaded for GPU for faster computation, ...

More complicated GC

More functionalities: File I/O, import, more built-in functions...

More parallel options: richer operators, parallel stdlib, parallelize user-defined functions, ...
DEMO TIME!
Demo 1: Quick Sort
Demo 1: Topological Sort
Demo 2: Linear Regression
Demo 3: Parallel Functions
Thank You

Willing to answer questions!!!
User Define Parallel Environment-impl

#define i8 @etADDf1(i8* %0, i8* %1) {
  entry:
    (... function)
    ret i8* %z3
}

#define i8 @etADDf2(i8* %0, i8* %1) {
  entry:
    (... function)
    ret i8* %z3
}

#define i8 @etADDreduce(i8** %result) {
  entry:
    (... function)
    ret i8* %tmp0p
}

@etADDmaps = global [2 x i8* ([i8*, i8*])]* zeroinitializer
@endADDreduce.1 = global i8* ([i8**])* null

define i8 @main() {
  entry:
    store i8* ([i8*, i8*])* @etADDf1, i8* ([i8*, i8*])* getelementptr inbounds
      (i8*, i8**)* @etADDmaps, i32 0, i32 0)
    store i8* ([i8*, i8*])@etADDf2, i8* ([i8*, i8*])* getelementptr inbounds
      (i8*, i8**)* @etADDmaps, i32 0, i32 1)
    store i8* ([i8**])@etADDreduce, i8* ([i8**])@etADDreduce.1
    ret i8 0
}

declare i8* @add(i8*, i8*)

(... declaration of all built in functions)

declare i8* @pe_calc(i8* ([i8*, i8*])**, i32, i8* ([i8**])*, i8*, i8*)
typedef void* (*pf)(void*, void*);
typedef void* (*rd)(void**);

typedef void* (*rd)(void**);

extern "C" void* pe_calc(pf* mapfunctions, int num, rd reduce, void* a, void* b) 
{
    int i;
    future<void*> pres[num];
    void *res[num];

    for (i = 0; i < num; i++) {
        pres[i] = async((*mapfunctions[i]), a, b);
    }

    for (i = 0; i < num; i++) {
        res[i] = pres[i].get();
    }

    return (*reduce)(res);
}
Tensors

Stack
- Local Variable
- Local Variable
- Global Variable

Heap
- Tensor
- Tensor
- Tensor
<table>
<thead>
<tr>
<th>Name</th>
<th>Roles</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xiangrong Xu</td>
<td>Built-in Programmer</td>
<td>Tensor operations, Built-in functions(from parser to codegen), Relevant Testing</td>
</tr>
<tr>
<td>Xinchen Xie</td>
<td>Project Manager</td>
<td>Tensor/Vartensor-related syntax, semant &amp; codegen, Tensor/Vartensor runtime checking, Tensor/Vartensor Indexing</td>
</tr>
<tr>
<td>Songqing Ye</td>
<td>Elite Programmer</td>
<td>Parallel Environment(from parser to codegen); Test for PE and STMT</td>
</tr>
<tr>
<td>Senhong Liu</td>
<td>First-class Designer</td>
<td>Architecture Design. Basic expression and statement implementation. LLvm translation.</td>
</tr>
</tbody>
</table>