MatCab Language Design Proposal

Tianchen Yu (ty2275), Cheng Xiang (cx2142), Yu Qiao (yq2145) and Ran Yu (ry2239)

Introduction

MatCab, a programming language which simplifies and accelerates matrix manipulations that we propose, is a C-style, in terms of syntax, language that particularly aims at easier and faster matrix computing for programmers. Matrix computing is one of the most common linear algebra operations used in scientific computation. Operations like image processing and data mining both require considerable amount of highly efficient matrix calculations. Our MatCab language, which is the combination of matrix and cab, provides an easy way to apply basic arithmetical operation to matrices and its C-like syntax can help C programmers get rid of the burden of hundreds of lines of c-code for matrix computing, making the computation simple, compact and easy to read.

We introduce new operators dedicatedly designed for matrices, thus making MatCab simpler in syntax to perform calculations. At the low level of MatCab, we introduce parallel computing techniques to speed up the process. . When performing linear algebra computation, MatCab will allocate some or all of the tasks to GPU, alleviating CPU burdens and making the best of GPU capacity. To be precise, MatCab takes advantage of CUDA language and GPU accelerations, thus making it faster in scientific computation

Key Features

MatCab is a strong typed programming language. Our target users are fluent C language programmers with high demand of fast and simple matrix calculations. To simplify the syntax, MatCab supports specific operators used for matrix computing.

Language Specification

Variable declaration:

    type: <variable name>;

Data types:

    int, float, char, string, array, mat, vector

(in that mat is the data type of a matrix)

```
mat a=[1,2; 2,3; 2,4]//Declare and initialize a 3 X 2 matrix with initial value
mat b=zeros(3,4)  //This will initialize a 3 X 4 matrix with all elements set to zeros
```

Assignment and function definition:

=

Functions:

fun <function_name> (type <argument_name>, ...) {<body>; return(optional);}

Control flow:

if-else:

```
if (<expr>),
        {<body>;}
else
        {<body>;}
```

Loops: while and for

```
while(<expr>)
        {<body>;}

for (<expr1>,<expr2>,<expr3>)
        {<body>;}
```

another way to write for loops:

```
for(<variable_name> in <variable_name>)
        {<body>;}
```

Boolean operators:

and, or, !

Arithmetic operators:

+ - * / = != < > <= >= %

Matrix operators:

+ - * .*(cross product) ‘(transpose operator) “(inverse operator)

[] return the row vector

[][] return the element

```
//IN BNF
primary_exp:
        id
        | string
        | const

exp:
        assignment_exp
        | exp ‘,’ assignment_exp

assignment_exp:
        id assignment_operator compute_exp
        | conditional_exp

assignment_operator:
        ‘=’ | ‘+=’ | ‘-=’

compute_exp:
        id
        | id compute_operator compute_exp

compute_operator:
        ‘+’|’-’|’*’|’/’|’*.’
```

(where ‘*.’ is the operator designed for matrix multiplication. More operators are to be introduced along with the progress of our project)

```
conditional_exp:
        primary_exp compare_operator primary_exp

compare_operator:
        ‘>’ | ‘<’ | ‘<=’ | ‘>=’

stat:
        exp_stat
        | selection_stat
        | iteration_stat

exp_stat
        exp ‘;’
        |‘;’
```

```
iteration_stat:
        'while' '(' exp ')' stat
        | 'do' stat 'while' '(' exp ')' ';'
        | 'for' '(' exp ';' exp ';' exp ')' stat

selection_stat:
        'if' '(' exp ')' stat
        | 'if' '(' exp ')' stat 'else' stat
        | 'switch' '(' exp ')' stat

const:
        int_const
        | char_const
        | float_const
```

Sample Codes

```
#define THRESHOLD 100

int main()
{
        mat a=[1,2; 2,3; 2,4];
        mat b=[1,2; 2,3; 2,4];
}

int matrixMultiplication(&matA, &matB, &result)
{
        int ma, na, mb, nb;
        dim(&matA, &ma, &na);        //dim() is a standard library function that finds
the dimensions of a given matrix
        dim(&matB, &mb, &nb);
        if (na != mb)
                return -1;

        //To load the work to GPU is somewhat time consuming, comparing to its
actual processing time. So if the dimensions of the 2 matrices do not exceed the
threshold, the multiplication will be calculated on CPU, otherwise the work will be
assigned to GPU
        if (ma > THRESHOLD  || na > THRESHOLD || nb > THRESHOLD)
                *result = matA *. matB;
        else
                *result = matA * matB;
        return 0;
}
```