Symbiosis

Research Article | Journal of Computer Science Applications and Information Technology | Open Access

Received: April 15, 2017
Accepted: April 25, 2017
Published: May 02, 2017

LIM: Language-Integrated Mathematical Computation

Xiaokui XIE*, Bernard J Lewis and Hongshi HE

Hunan Agricultural University, Changsha 410128, China

***Corresponding author:** Xiaokui XIE, Hunan Agricultural University, Changsha 410128, China, E-mail: xiexiaokui@qq.com

1 Abstract

Theory, experiment, and scientific computation comprise three main tools for scientific research and technology development. With respect to computation, traditional scientific languages such as IDL (Interactive Data Language), APL, Fortran and MatLab were widely used, but they were also complex or error-prone when writing large programs because they lack some attributes of modern object-oriented languages. Here Language-Integrated Mathematic (LIM) computation was designed as a high-level, object-oriented extension for modern general-purpose programming languages. Matrix data structure is the base of LIM with a simpler and lightweight syntax like dynamic and special mathematic languages, such as matrix generator, slice, composition and deconstruction, while "function inference" is the core of LIM for automatic algorithm generation which is both dynamic and type safe. LIM provides streamlined mathematical and scientific computational facilities while allowing developers to use existing, familiar tools and techniques, enabling scientists and engineers to develop data- intensive and computational-intensive programs in a natural, user-friendly way handling big data era.

2 Keywords:

Scientific computation; Mathematics; Programming language; LIM;

3 Introduction

Scientific computation is an integral part of modern scientific research and technology development. Much software is developed in programming languages employing mathematical computation to solve scientific problems [3-5]. While traditional mathematical languages such as APL, Fortran and MatLab (Mathworks, Inc.) are used extensively in mathematical and scientific computation, they lack some important attributes of modern languages, such as generics (type parameter) and type safety (strong typing). For that reason it is often difficult to write, reuse and debug large-scale programs written in these languages [8]. Modern languages such as C# (Microsoft corporation), Java (Oracle corporation) and C++ have been quite successful in business and general software applications, but extra work was required with minimal benefit and became cumbersome in applications requiring large scientific computations.

A considerable amount of research has been devoted to mathematical computation [2,7,8]. Some new special languages such as IDL (Interactive Data Language for ENVI), open source R, Microsoft f# and Oracle Fortress have been created; and some libraries such as JScience for Java (http://www.jscience.org/), Mathdotnet for .NET (https://www.mathdotnet.com/) and NumPy (http://www.numpy.org/) for Python have been established. The software industry now has reached a stable point in the evolution of object-oriented (OO) programming technologies. It continues to be a big challenge in scientific and engineering programming to reduce the complexity of mathematical computation using OO technology. Rather than design an entirely new programming language or add libraries to meet the requirements of a particular area, a more general approach is taken here to create Language-Integrated Mathematic (LIM) by adding general-purpose mathematical facilities to the OO Framework.

LIM adopts a formula translation syntax which is quite familiar to scientists and engineers. It bridges the worlds of OO and scientific computation, representing the logical prescription for bringing type-safe scientific computation to modern OO language, thus conferring "first-class citizenship" on scientific computation.

4 How LIM Works

To see how LIM works, we begin with a square root operation on a matrix. Below is the LIM code for C#.

If one were to compile and run this program, the output would appear as follows:

1.0, 2.0,4.0,5.0,7.0,8.0

We can dissect the second statement in Figure 1 into the following three statements in Figure 2:

The expression uses three operators - ':'; '[]'; and 'Sqrt'. The first is a matrix generator; the second is an element/elements getter or setter by index/indexes (matrix slice); and the third is a mathematical function operator to find the square root of every el-

```
using System;
using System.Lim;
using static System.Math;
class Program
{
   public static void Main(string[] args)
   {
      // nums is of type double [,] by type inference
       var nums ={{ 1, 4, 9 },{16,25,36}, {49,64,81}};
      // root is of double [,] by type inference
      var root = Sqrt(nums[ : , 0:2]);
      Console.WriteLine(root.ToString());
   }
}
```

Figure 1 Lim program written in C# to calculate the square root of a matrix

```
// generate one dimension array with value {0,1} of type int[]
int[] indice0 = 0:nums.Size[0]; // {0, 1, 2}
int[] indice1 = 0:2; // {0, 1}
// extract elements with the specified row and column indices
double[,] subNums = nums[indice0,indice1]; //{{1,4},{16,25},{49,64}}
// call Sqrt operate on the extracted elements and
// return a matrix of the same dimension
double[,] root = Sqrt(subNums); // {{1.0, 2.0},{4.0, 5.0},{7.0, 8.0}}
```

Figure 2 Dissected program for the second statement of the program depicted in Figure1

ement in a matrix. Note that there is no "***double[] Sqrt(double[] source)***" function in the source code or elsewhere.

Thus the data flow of above program can be demonstrated by the following figure 3.

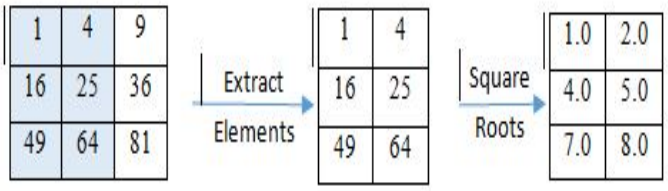

Figure 3 Data flow of program depicted in Figure1

Matrix is the most important data structure in LIM. Generics technology is used to define arbitrary data types. This can maximize code reusability, type safety and performance and allow LIM to integrate with a wide variety of databases seamlessly and speedily. Most matrix operators such as arithmetical and Boolean calculations in MatLab and Fortran are implemented here on the matrix data type using OO with other important concepts such as matrix composite, matrix properties and matrix deconstruction, making it possible for developers to work with LIM in a way with which they are already comfortable. The following C# code demos the concepts of matrix composite, matrix properties and matrix deconstruction mentioned above. (Figure 4)

"***double[] Sqrt(double[] source)***" was implicitly created by a mechanism called "**function inference**" **(FI)**. LIM can automatically generate program and implicitly process many loop and selection algorithms, greatly reducing the number of programming statements.

```
var A = {1, 2}; // A is type of int[];
var B = {3, 4}; // B is type of int[];
// matrix composite
var C = {A, B}; // C is type of int[,] with value of { {1,2}, {3,4}};
// matrix properties
var D = C.Size; // D is type of int[] with value of {2, 2};
// matrix deconstruction
var {A1, B1} = C; // A1 equals to A, B1 equals to B
var {{a,b},{c, d}} = C; // variables will get corresponding value a=1; b=2; c=3; d=4;
```

Figure 4 Demos of matrix composite, matrix properties and matrix deconstruction

5 Function Inference

For a function call, if the user does not explicitly provide a function that exactly matches the input parameters, then the compiler will try to create an overloaded function according to the inference rules by properly calling visible functions which is called **base function**.

Function inference can generate most loop and selection algorithms dynamically for a matrix with the provided base functions. It is particularly suitable for mathematical, scientific and engineering computation, allowing scientists and engineers develop mathematical programs rapidly, directing more energy toward substantive, as opposed to computational, intricacies of scientific problems.

Two kinds of Function inferences will be discussed here:

5.1 Elemental Function Inference

Elemental functions are unary functions. The provided base function is a scalar quantity version which takes one atomic type (structure, class, etc.) as an input parameter. The return type can be any type or be void. Some examples of elemental operators are conversion operators and trigonometric functions, such as Sin, Cos, Tan et, al.

The automatically-generated elemental function inference works by calling the provided base function once for each member of the underlying array and returns the calculated values as an array with the same dimension as the input parameters.

For example, the prototype of the provided base function to return the sine value of a specified number is "double Sin (double d)" and the following statement calculates the sine of a matrix by function inference: (Figure 5)

```
using System;
using System.Lim;
using static System.Math;
var angles={{0, PI/6},{PI/2, PI}};
var sine = Sin(angles );
// return an matrix {{0.0, 0.5},{1.0, 0.0}}
// sine is type of double[,] and is calculated by auto-generated algorhtim with function inference
```

Figure 5 Demos of elemental function inference

Note that there is only a static function double Sin(double a) in System.Math class. There is no function like ***double[] Sin(double[] A)*** anywhere. But the compiler for LIM can generate appropriate overloaded functions (auto-generated algorithm)

Citation: Xiaokui X, Bernard JL, Hongshi HE (2017) LIM: Language-Integrated Mathematical Computation. J Comp Sci Appl Inform Technol. 2(2):1-4

for the matrix parameter by FI.

Elemental function inference process generally behaves like this:

(1) LIM will try to find a visible function with the same function name which accepts scale parameter of the same type as the matrix element. The found function is regarded as the base function.

(2) LIM will auto-generate the proper function prototype in addition to dynamically generating proper algorithm. In the above example, the generated function is *double[] Sin(double[] A)*.

(3) The dynamically generated function will be called.

Without LIM, many repeated loop and selection statements must be written by hand, making programs complex, error-prone and cumbersome. Other mathematical computation languages and dynamic languages can also handle matrix parameters directly, but they lost type information, thus no interpreter's or compiler's error will occur until running.

5.2 Aggregate Function Inference

An aggregate function receives a sequence of values as a whole (e.g., vector, array) for input parameters and returns the aggregated value. There are two kinds of aggregate function inferences: for single matrix parameter and for multiply matrix parameters.

The automatic-generated aggregate function inference works by signaling (calling) the provided operator once for each member of the underlying array and returns the result as a scalar quantity or array according to the input parameters. Important examples of aggregate operators include statistical operators such as Max, Min, and Product and so on.

For example, the following program uses Sum to accumulate the sum of total values over a numeric matrix, and then sum the matrix and itself Figure 6:

```
public static int Sum(int a, int b) => a + b; // base function
var nums ={{1,2, 3}, {4,5,6}, {7,8,9}};
// single parameter aggregate functioin inference
// first get value of {12,15,18} and then return 45
var sum = Sum(Sum(nums)); // equals to int[] t = Sum(numes); int sum = Sum(t);
// multiple parameter aggregate function inference
var sum3 = Sum(nums, nums, nums); // sum3 = {{3,6,9},{12,15,18},{21,24,27}};
```

Figure 6 Demos of aggregate function inference

The process of the above program can be shown as following Figure 7:

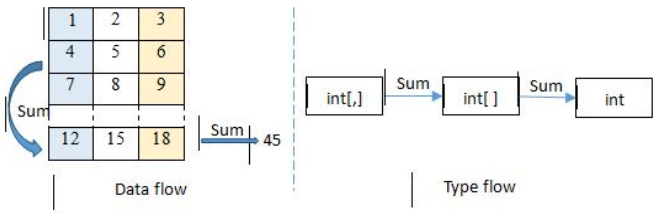

Figure 7 Data flow and type flow by aggregate function inference

Aggregate function inference follows the following steps:

(1) If the parameter is type of one dimension matrix, LIM will try to find a visible function with the same function name which accepts **two scale parameters** of the same type as the matrix element. The found function is regarded as base function.

For multiple parameter function inference, if the parameter is n (n>1) dimension matrix, LIM will try to search a visible function with the same function name which accepts n-1 dimension matrix parameter with the same element type. If no result got, compiler will try to search function taking in n-2 dimension matrix parameter, until a base function is found.

(2) LIM will auto-generate the proper function prototype in addition to dynamically generating proper algorithm. In the above example, two functions will be created, first is double Sum(double[] A) and the other double[] Sum(double[,] A).

(3) The dynamically generated function will be called.

In short, inferred functions are early binding and statically checked, so they are type-safe. FI enables compiler to optimize large-scale data processing such as parallel and optimal algorithm, so it provides rapid and secure development method for OO and functional programming (FP).

6 Conclusions

LIM = matrix data structure + function inference, matrix is the base and FI is the core. Matrix is simple, lightweight data structure which makes LIM much like traditional mathematical and dynamic programming language, such as IDL, APL, Fortran and Python. LIM adopts an easy-to-read style particularly suited to rapid development of scientific and technological computation. Function inference is utilized for automatic program generation, enabling a compiler to produce faster and more efficient code. LIM enables higher levels of application performance, robustness and reliability by maintaining a single and easy-to-manage programming model. When available, parallel technologies can be employed to utilize multiple processors or cores for execution in the new age of concurrency and multi-core without any changes in the LIM programming model and source code. To date, harnessing the power of the GPU require programmers use of a GPU-specific programming model like CUDA, OpenCL, or OpenACC [6], and programming and optimizing on current many-core accelerated HPC systems is very challenging [1]. Compiler for LIM can optimize for big data processing on GPU, or even on supercomputer automatically in the future. Beside, LIM is type safe extension for general purpose language which means that is can handle user interface, database, web, cloud computation very easily. Thus, performance loss and type mismatch considerably greatly in mixed programming does not exist in LIM.

Base on LIM architecture, many professional toolboxes can be developed easily and elegantly. LIM enables scientists and engineers to write computationally intensive programs in a more natural, user-friendly way than do traditional programming languages such as c, c++, Fortran and MatLab. In short, LIM adds a dimension of enjoyment and interest to the computational process.

Citation: Xiaokui X, Bernard JL, Hongshi HE (2017) LIM: Language-Integrated Mathematical Computation. J Comp Sci Appl Inform Technol. 2(2):1-4

7 References

1. Xu C, Deng X, Zhang L, Fang J, Wang G, Jiang Y, et al. Collaborating CPU and GPU for large-scale high-order CFD simulations with complex grids on the TianHe-1A supercomputer. Journal of Computational Physics, 2014;278:275-297.

2. Chudoba R, Sadilek V, Rypl R, Vorechovsky M. Using Python for scientific computing: Efficient and flexible evaluation of the statistical characteristics of functions with multivariate random inputs. Computer Physics Communications. 2013;184(2):414-427.

3. Katz RF, Knepley MG, Smith B, Spiegelman M, Coon ET. Numerical simulation of geodynamic processes with the portable extensible toolkit for scientific computation. Physics of the Earth & Planetary Interiors. 2007;163(1-4):52-68.

4. Mattick JS, Gagen MJ. Mathematics/computation: accelerating networks. Science. 2005;307(5711):856-858.

5. Noack, M. RSYST: from nuclear reactor calculations towards a highly sophisticated scientific. Fuel & Energy Abstracts. 1996;37:233.

6. Basu P, Williams S, Straalen BV, Oliker L, Colellaa P, Hall M. Compiler-Based Code Generation and Autotuning for Geometric Multigrid on GPU-Accelerated Supercomputers. Parallel Computing. 2017.

7. Esterie P, Falcou J, Gaunard M, LaprestÃľ JT, Lacassagne L. The numerical template toolbox: A modern C++ design for scientific computing. Journal of Parallel & Distributed Computing. 2014;74(12):3240-3253.

8. Waldrop MM. Frustrated with Fortran? Bored by Basic? Try OOP!. Science. 1993;261(5123):849-850.

Citation: Xiaokui X, Bernard JL, Hongshi HE (2017) LIM: Language-Integrated Mathematical Computation. J Comp Sci Appl Inform Technol. 2(2):1-4