Developing a Multi-threaded Algebraic Application using Mathematical Pseudo Language for Efficient Computing

Anurag Tiwari¹, Akul Gupta² and Dr. Vivek Kumar Singh³
¹Department of Computer Science Babu Banarasi Das University Lucknow, India
E-mail- anuragrtiwari@gmail.com
²P.G Student, Department of Computer Science BBD University, Lucknow, India
E-mail – akul_animesh@yahoo.com
³Asst. Prof & Head, Department of Information Technology BBDNITM, Lucknow, India
E-mail-viveksinghbbd@gmail.com

Abstract—Solving mathematical expression on pen and paper has been an area of concern for the past few decades due to their inability to yield correct, exact and quick results most importantly when dealing in large expression with added complexity. Hence digital media was chosen to be an option for solving such issues. Henceforth we came up with an idea of a complete computer algebra system which one can think of. It has powerful parser working at the backend, with introduced parallelism that solves large and complex expressions with a lot of added variations quickly. This is accomplished using the defined grammar for parsing and MPL algorithms implemented within the code that has the capability to deal with a variety of challenges put forward by the user.

Index Terms— Structural Parser, Symbolic Computation, Multi-threading, Computer Algebra System; Mathematical Pseudo Language.

I. INTRODUCTION

Java programming language was first developed for developing web applications by Sun Microsystems. It provides interpreter for byte code (Java Virtual Machine), a library, an object hierarchy, abstract window tool kit and swing components along with the compiler to build graphical user interface. It also provides programming conventions for event driven inter component operations. It is a cross-platform language so that the applications can be run in any platform or any operating system. This programming language has been evolved as a modern programming language in the field of high performance computing because of its appealing features. It provides some in-built classes for developing applications in multi-core cluster system as well as in distributed systems [1, 2, and 3]. Thread class is used to create multiple threads inside a single process. In addition to that java.util.concurrent package provides a way for efficient utilization of threads. Using thread and concurrency package parallelism can be achieved at the loop level [4, 5] (i.e. the loop iterations can be executed concurrently). For deploying an application in multiprocessor system multi-threaded programming model has been used. Libraries like OpenMP [6], JOMP,
to deal with dynamic, irregular OpenMP applications, we claim that the key step is to transmit information extracted by the compiler to the underlying thread scheduler in a continuous way. Indeed, only a tight integration of application-provided meta-data and architecture description can let the underlying runtime system take appropriate decisions during the whole application run time [7]. Support this concept. OpenMP is developed in C language and it is compatible with C and C++. JOMP is developed in Java language. MPL algorithms are generic, meaning that they are not tied to particular sequence class implementations, and can operate on a wide range of arguments as long as they satisfy the documented requirements. However we can use Java’s inbuilt thread class and concurrency package to develop multiprocessor applications. In section II the Symbolic Computer Algebra System and Mathematical Pseudo Language (MPL), section III is about symbolic computation in java is discussed. Section IV describes multi-threading concepts. The algorithm of Symbolic Computation is discussed in section V. comparison of java to traditional language is valued in section VI. How multi threading is used for the Algebra System is explained in section VII. In section VIII we cover the working mechanism and in section IX we discussed the whole conclusion about our system.

II. SYMBOLIC COMPUTER ALGEBRA SYSTEM

Symbolic computation is a Computer Algebra System [8] which takes the mathematical expression using alphabets, numeric characters and symbolic notations and parses the expression using a recursive parser and finally gives the output. Symbolic computation software is efficient for certain tasks, but worse than the human mind for others. Solving a system of linear equations symbolically on a computer is slow, taking into account the additional time it takes to input the data, to carry out automatic simplification of the resulting expression, and to read the output (not to mention the time it takes to figure out how to use the software). With the current state of the software, it is usually much faster to do it by hand. If we cannot manage by hand, neither will the computer. Simplifying expressions is also something humans seem to be far better in than computers. Especially when the same calculation is carried out by hand over and over again, the human brain recognizes simplifications that elude current symbolic algebra packages. A Mathematical Pseudo Language is used to interpret the symbolic expressions and convert it into a string which is actually given to the parser [9].

There are some CAS tools which are already developed. They are MAPLE [10], MATHEMATICA [11], MATLAB, MuPAD etc. Mathematical pseudo language (MPL) is an algorithmic language that is used to describe the concepts and algorithms of computer algebra. MPL algorithms are readily expressed in the programming languages of MAPLE, MATHEMATICA, and MuPAD. In this Computer Algebra System the MPL algorithm is also implemented. MPL mathematical expressions are constructed with the following symbols and operators.

- Integers and Fractions that utilize rational number arithmetic.
- Identifiers that are used in expressions like summation, integration etc.
- The algebraic operators +, -, *, /, ^ (Power), and! (Factorial)
- Function forms that are used for mathematical functions sin(x), indefint(x^2+sin(x))dx, log[b](expr), permut[n, r] etc.

III. SYMBOLIC COMPUTATION IN JAVA

A. Objects as functions

The view of mathematical objects as functions is powerful. The difficulty is to transmit the data across the components and that can be achieved by using different programming language [11].

B. Storage management

Languages like Lisp, Java and Maple provides automatic garbage collection where C and C++ require explicit garbage collection. The most challenging thing for component technology is to combine the different memory models in a single computation.

C. Exceptions

The most familiar way of handling exceptions is try/throw/catch mechanism in Object oriented programming languages. Exception handling must be included between the components.
D. Legacy code

It is always recommended to start the computation from the bottom layer because all the functionalities can be custom made. However, such design leads to monolithic systems which must keep pace with new algorithmic advances. There remains the legacy code, often written in a different programming language.

E. Algorithmic shortcuts

One of the important paradigms in Symbolic Computation is the evaluation technique. Some libraries can be used for the same. Sometimes the libraries can be modified also. Building Symbolic Computation is better in Java as compared to other programming language like C, C++ etc.

F. Generic Programming

The main purpose of generic programming is the ability to write algorithms with variable types. The template feature of C++ is designed for the first instantiation method. Dynamic evaluation is used in Maple’s domain package. C++ has virtual class member functions for this purpose, which has been adopted by abstract method is Java.

Originally Java was thought as architecture independent programming language. With the byte code, virtual machine, and the automatic garbage collection its execution resembles that of Maple, Mathematica and Lisp.

G. Plug and Play

Java is a downward compatible language that means any program written in C or in other programming languages can be invoked by a Java program via its native method mechanism. The java runtime environment decides if a native code library is available and chooses between a Java and a more efficient native implementation [12].

There is an alternate way facilitated by Java's networking capability. The java and non-java components can communicate via sockets or pipeline.

H. Memory Management

Java provides dynamic memory management facility. Automatic garbage collection provides a convenience to the programmer, but it has some condition. The virtual machine must determine if the object is still in use. C++ can provide garbage collection through constructors and destructors but it need some library package to be imported. However the efficient memory management is required for Symbolic Computation as it deals with large data structure.

Java 1.6 allows the programmer some control over memory allocation through what are called weak references. For example, references to newly made objects can be placed into a table that acts as an object cache. If an object is found in this table, it needs not be constructed but can be shared. Maple implements the object cache technique for sharing one and the same sub expression. A reference in the table should be weak, so that if it remains the only reference to an object its storage is garbage collected and the reference in the table set to null.

I. Multi threading

A program can be divided into executable parts that are called as Threads [13]. Each thread defines its separate path of execution. Implementation of threads and processes differs from one operating system to another but one or more threads can be contained inside a single process. So thread is also known as a light weight process. In a process multiple threads can be created which run simultaneously. So multi threading is known as a special form of multi tasking.

It also provides the abstract model for concurrent execution. With the help of multi threading multiple parts of a program can be run simultaneously in each processor in order to optimize the execution time and efficiency. So it is always beneficial to use multi threading concept to develop application for multiprocessor system.

Multithreaded programs can be run efficiently in multiprocessor architecture where each thread can be assigned to individual processors. This reduces the execution time and improves the efficiency of the process.

For this paper multi-threaded model is used with sixteen threads which will be executed in a multiprocessor system having sixteen processors. An application can be run using one thread or multiple threads. For that the applications first interact with the operating system and then threads are executed in processors. One thread can be run in one processor at a time. If an application contains more than one thread then more processors are involved in execution of that application. In Figure 1 it has been shown that application first interacts with operating system then operating system decides which thread will be executed in which processor.
This section describes the use of Java to experiment with the software component paradigm in building Problem Solving Environments (PSEs) that use symbolic computation. Java Beans are made from ordinary Java code by employing additional coding conventions when building software components.

An application is built by plugging together and configuring instances of Beans. The application builder can view their task as configuration and wiring -plugging together Beans and entering initial values for configuration (data) fields of each Bean.

Java Studio [14] is an example of a component composition system for Java Beans. It presents a visual programming tool to application development using component configuration and wiring. We found Java's multithreading features a great aid in building software components, making it easier to handle simultaneous or interrupting incoming events. Java Studio also capitalizes on strengths of Java in the following ways:

- It builds graphical user interface icons automatically for each Bean as it is imported, using reaction to determine the input and output ports as well as the number, name, and type of configuration fields for the component.
- Java Studio can provide a number of graphical user interface-building widgets as ‘giveaways’ because the Java library already provides the base functionality.
- To assist rapid prototyping, components can be loaded and executed on the fly.

Our Bean building efforts have focused on providing a Bean form for the Matlab and Maple engines found in TechTalk. We called this ‘MathServerBean’.

V. ALGORITHM

Symbolic computation takes the input in two ways. First is from the user directly and takes the special symbols i.e. (Summation, Integration, Theta, exponential etc) from the symbol panel. Three basic swing components are used to represent the input.

- JTextField
- JLabel
- JPanel

The Symbolic Computation package is divided into five basic modules.

- Add Symbol
- Listener
- Extract Input
- Parser
- Extract Output

A. Add Symbol

The special symbols as mentioned earlier are added to input in a standard mathematical expression. And the special symbols are grouped with related inputs. i.e. (The integration symbol is grouped with its upper limit and lower limit).

Add Symbol Algorithm

- Take the coordinate pointer to add symbols
- Set the dimension of symbols
- Add the symbols to input panel
Update the panel
Increment the coordinate pointer

B. Listener
Listener is used to handle the events generated either by Key Board or Symbol Palette. Two types of Listeners are there.
- Symbol Listener
- Constant Listener
Symbol Listener handles the events generated by Symbol Palette. And Constant Listener handles the events generated by Key Board. Both the listener uses the Java APIs.

C. Parser
A parser takes the input expression and tokenizes the input into different segments, evaluates them finally gives the result. Here the structural parser scans the mathematical expression represented in symbolic notations and converted it into a string which can be accepted by the parser. Then the expression is tokenized and each token is passed to the respective parser module. Each parser module expects a double value. If the input is of type double then it evaluates the corresponding value and returns that value. If it finds an expression then it calls the main parser recursively and if it gets the double value then evaluates it otherwise it sends back the string as constant expression.

VI. A Comparison Of Java To Traditional Languages
We now discuss how well Java meets the requirements for building symbolic computation components in comparison with C++, Lisp, Smalltalk, Maple, and Mathematica. We will also take into account the Al/Alidor [15] compiler.

A. Generic Programming
A main goal of generic programming is the ability to write algorithms with variable types: for example, a Gröbner basis procedure is parameterized with the coefficient field and can be instantiated for a multitude of different fields. In [18] we distinguish three implementations of genericity:
1. The field operations that define the common interface (addition, division, equality tests, etc.) are inline-compiled into the procedure for a selected field;
2. They are retrieved via a function pointer in the stub for the field object (an abstract base class);
3. They are linked into the procedure at load time. The template feature of C++ is designed for the first instantiation method. A large example is C++'s standard template library and we have used this approach in FoxBox. Dynamic evaluation is used in the Scratchpad II/Axiom system and in Maple's Domain package. C++ has virtual class member functions for these purposes, which have been adopted by Java under the name of abstract methods. A main point in is that in C++ one can write a generic program so that the choice of the three instantiation methods for type parameters is left to the users.

Originally Java was thought as a computer architecture independent programming system. It adopted Smalltalk's approach of an interpreted intermediate instruction set, namely, Java's byte code and virtual machine. With automatic garbage collection its execution mechanism resembles that of Maple, Mathematica, and Lisp. However, even the use of C++'s virtual member functions is discouraged for reason of efficiency. Our experience with Maple and Lisp certainly warns us that Java byte code may suffer even worse inefficiencies. It is expected, nevertheless, that Java programs will be compiled into native machine instructions. Compilers may, for reason of improving the efficiency, dynamically adapt the produced code to the present run-time environment.

B. Parallelism
Threads With multi-CPU PCs becoming common place, multi-threading becomes an increasingly important tool for expressing fine-grained parallelism. Three features characterize an efficient multi-threaded development environment:
1. Thread control
2. Memory locking
3. Thread-safe libraries
Java provides built-in support for creating and starting multiple threads of execution. C or C++ require additional libraries, like the Posix threads library; Kuchlin has endowed Saclib with multi-threading in this
manner. Aldor cannot take advantage of these libraries for reasons discussed below. Computer algebra systems like Maple or Mathematica do not provide any tools for thread creation, although an experimental version of Maple uses threads in order to parallelize selected internal operations. Memory locking is vital in a multi-threaded environment in order to prevent memory accesses of independent threads from interfering with each other. The Posix threads library provides semaphores and memory locks as library calls. Java integrates semaphores into the language itself, yielding the potential of enhanced efficiency through data-flow analysis and compiler optimization.

VII. SYMBOLIC COMPUTATION ALGORITHM

A. Extract Input

Extract Input scans the input panel and stores the components in an array of components. It scans component and extracts the information and stores in a string. The string is prepared based upon the algorithmic language (Mathematical Pseudo Language). Symbolic Input and Expression String equation are as follow.

Algorithm is as follows

1. Scan the components
2. Set the input string as "+null+
3. If it is text field then add the content of that
text field into input string
4. If it is label then update the input by adding the Label value
5. If it is panel then check for special symbol
6. If it contains \( \sum \)
   Add sum[delim, upper_limit, lower_limit] to input
   - If it contains \( \prod \)
     Add permu[n,r] to input
   - If it contains \( \sum \)
     Add combi[n,r] to input
   - If it contains \( \int \)\( u \)
     Add inte[upper_limit, lower_limit] to input string
   - If it contains \( \sqrt{\text{n}} \)
     Add root[n] to input string
8. If it contains \( \log_{bx} \)
   Add \( \log_{b}(x) \) to input string
9. String To Symbol

StringToSymbol class is used to convert a string which is in MPL (pseudo language) to Symbolic Expression. It takes two objects as input to the method. First one is the string and second one is the panel where the symbolic expression will be added.

Algorithm is as follows

1. Extract the input from String
2. Tokenize the input string and store in a array input[ ]
3. For i -> 1 to length of component array
4. If name is sum
   call AddSummation.add( )
If name is prod
   call AddProd.add()
If name is indefint
   call AddIndefIntIntegration.add()
If name is definit
   Call AddDefiniteIntegration.add()
If name is sroot
   call AddSRoot.add()
If name is nroot
   call AddNRoot.add()
If name is log1
   call AddLog.add()
If name is log b

**C. String To Error**

The output returned from the parser may contain errors. The errors are represented by error codes. StringToError class is used to extract the errors from the output string and shows the top five errors in the error stream in the output panel.

Algorithm is as follows
- Extract the input from the Panel or take a String
- Tokenize the input string and store in a array input [ ]
- Scan each token
- For i-> 1 to length of tokens
- If it starts with ERC
- take the complete ERC code
- Check the code in database
- Retrieve the complete information and stores in a string
- Display the errors in error stream

**D. Parser**

Parser is one of the important classes. It is used to evaluate the expression. The expression can be given in two ways. It directly takes the input panel and calls the ExtractInput class or it can directly take the string which is helpful in case of Command Line Input. It divides the inputs into tokens and passes the each token to respective parsing module.

Algorithm is as follows
- Extract the input from the Panel or take a String
- Tokenize the input string and store in a array input [ ]
- For i -> 0 to length of input
- If input[i] is a valid character
- Add to output
- input[i] is an expression
- Solve using expression tree and add to output
- If input [i] starts with integration
- Call parseIntegration Module
- If input [i] starts with differentiation
- Use parseDifferentiation method
- If input[i] starts with —sum
- Call parseSum method and add the result to output
- If input[i] starts with —permu
- Call parsePermutation method and add the result to output
- If input[i] contains
- Call parseFact and update the output
If input[i] is any trigonometric function
- Call respective parser and update the output E.

**E. Expression Evaluation**

Expression Evaluation is class which is used to find the output of an expression. It uses stack to convert the expression into postfix expression and then evaluate it.

**Algorithm Infix to Postfix Conversion.**

- Define a stack
- Go through each character in the string
- If it is between 0 and 9, append it to output string.
- If it is left brace push to stack
- If it is operator */+- then
- If the stack is empty push it to the stack
- If the stack is not empty then start a loop:
  - If the top of the stack has higher precedence
  - Then pop and append to output string
  - Else break
- Push to the stack
- If it is right brace then
  - While stack not empty and top not equal to left brace
  - Pop from stack and append to output string
  - Finally pop out the left brace.
- If there is any input in the stack pop and append to the output string.

**F. Postfix Evaluation**

Algorithm is as follows

- Scan the Postfix string from left to right.
- Initialize an empty stack.
- If the scanned character is an operand, add it to the stack. If the scanned character is an operator, there will be at least two operands in the stack.
- If the scanned character is an Operator, then we store the top most element of the stack (top Stack) in a variable temp.
- Pop the stack.
- Now evaluate top Stack (Operator) temp.
- Let the result of this operation be res.
- Pop the stack and Push res into the stack.
- Repeat this step till all the characters are scanned.

**VIII. Working Mechanism**

At first the symbolic expression is extracted and converted into pseudo expression. Then the expression is tokenized and hence given to parser. Parser takes the input as array of tokens. Then a thread pool is created in which each thread of the thread pool picks a token and processes the token by calling the corresponding parse method.

The figure 2 below shows that four threads are there in the thread pool and each thread processes a token i.e. Integration is done using one thread while derivates, permutation and series evaluation can be done concurrently.
VIII. RESULT & DISCUSSION

The distinguishing feature of the application are the Java API's and MPL algorithms which are the building blocks of its highly efficient parser. The hold the capability to yield successful results for the multiple complex input provided by the user. An interactive interface existence inculcates a better user interaction with the product. Due to introduction of parallelism we get better results when compared to sequential results. As shows in figure 3 & Figure 4.

*****Sample input 1*****

\[
\begin{align*}
(x^2 + \alpha^2) dx + \sqrt{625} + \sqrt{64} + \log_{10}(10) + h(x) + \sin(1 + 2 + 3 + 4 + 5 + x) e^x + \\
\log_{10}(27) + \int \cos(a) \, da
\end{align*}
\]

*****Sample Output 1*****

\[
25.3410486349161 + x^2 + \int 3 + x^3 \int \ln(a) + K + \\
\sin(15.0 + x) + \sin(a) + K
\]

Figure 3 Inputs and Output

*****Sample input 2*****

\[
2 \sin\left(\sum_{i=1}^{10} (x^2 + x + 1)\right) + \int (x^2 + x + y) \, dx + \log_{10}(\log_{10}(100)) + 5 y_2 + \\
\frac{d}{dx} (y^2 + y + e)
\]

*****Sample Output*****

\[
3.31U + 6 \sin(y) + 1U + 2U + Y
\]

Figure 4 Inputs and Output

CONCLUSION

Designing an application that deals with all kinds of mathematical problems is next to impossible. Hence we had picked up a set of classes and solved problems belonging to those classes which were successfully accomplished. This CAS was designed taking into account the existing applications in the market to make the subject an interesting stuff.

This application was developed keeping in mind the other software like Lingo, Maple, Mat-Lab, Mathematica etc and features of Java by which the functionalities can be achieved. Since the paper named above require extensive knowledge of the subject and learning of certain languages, hence we tried our best. The application however can be extended in order to make it efficient by toning down its properties adding new functionalities and wide range of inputs.
ACKNOWLEDGMENT

The authors wish to thank Dr. Manuj Darbari without his essential guidance, this research paper would not have been possible.

REFERENCES