2012 4th International Conference on Computer Research and Development
IPCSIT vol.39 (2012) © (2012) IACSIT Press, Singapore

MatPro: A Multithreaded Programming Language for Numerical Computing

Upul Bandara, Chathura de Silva

Department of Computer Science and Engineering, Faculty of Engineering, University of Moratuwa, Sri Lanka

upulbandara@gmail.com, chathura@uom.lk

Abstract. Numerical computing is one of the key tools in engineering designs. Due to its utmost importance, researchers have been developing libraries, programming languages, and development environments for numerical computing. Though numerical computing libraries are mature enough for real-world engineering applications, numerical programming languages are not as matured as numerical computing libraries. Today's numerical programming languages are associated with issues, such as poor runtime performance, poor security, and poor utilization of multithreading capabilities of modern processors. In order to address these issues we developed a new programming language named MatPro.
Further, a prototyped compiler was developed for the MatPro programming language and multithreading capabilities was integrated into the MatPro compiler. The multithreaded programming model introduced for this language is based on the dynamic multithreaded programming model. Presently, performance of the MatPro compiler is equal to the performance of the C# compiler and we will be working on enhancing the performance of our compiler.

Keywords: compiler; programming language; matrix; numerical computing; performance

1 Introduction

Numerical computing is one of the key tools in engineering designs. Due to the importance of numerical computing in today's context, researchers have developed a lot of languages and tools for numerical computing. Presently MATLAB [1], Octave [2], and SciLab [3] are the most popular tools for numerical computing. Though these languages are popular amongst engineers and scientists, some shortcomings are associated with these languages. All the languages mentioned above are weakly dynamically typed programming languages. Conrad Sanderson [4] has mentioned issues associated with weakly dynamically typed programming languages. According to him it is possible to write programs in MATLAB or Octave that cannot be completely checked for correct syntax usage at compile time and hence increases the risk of breakage in any deployed programs, or runtime errors occurring during lengthy experiments. Furthermore, all the languages mentioned above are interpreted programming languages. Even though interpreted languages are good for developing prototypes in a short time frame, interpreted languages are not good enough for high-performance applications. Therefore, researchers who develop numerical prototypes in languages such as MATLAB and Octave, in the end ported these prototypes into compiled languages such as C or C++ [4]. Today almost all the computer processors come with more than one processing core. Therefore, in order to get the maximum performance from these processors, effective use of multithreading is essential. Presently all the above mentioned programming languages do not support language level support for multithreading.

Therefore, in order to address these issues we have developed a new programming language called MatPro. MatPro is a strongly and statically typed multithreaded programming language, specially designed for numerical computing. MatPro's multithreaded programming model is based on the dynamic multithreading semantic introduced by Charles E. Leiserson and et al [5]. Source programs written in the MatPro language are compiled to the Common Intermediate Language (CIL) [6], and run inside the Common Language Infrastructure (CLR) [6].

The rest of the paper is organized as follows. In Section II we will be presenting an overview of currently available languages and tools for numerical computing. In Section III we will be discussing the main features of the MatPro programming language. In Section IV we will be discussing the design and construction of the MatPro compiler. In Section V we will be discussing the performance of MatPro compiler. Finally, in Section VI we conclude our paper by discussing the limitations and further improvements of the MatPro language and the MatPro compiler.

2 Related Work

In this section we will discuss several important libraries and programming languages designed for numerical computing.

Linear Algebra PACkage (LAPACK) [7] is one of the popular software libraries for numerical linear algebra. It was originally written in FORTRAN77 [8] and later ported to other languages such as C [9] and C++ [10]. The main goal of LAPACK package is to design and implement a portable and efficient numerical linear algebra package for high-performance computing. LAPACK was designed to be a general purpose library for linear algebra. Therefore, verbosity of the programming interfaces of LAPACK is very high. Hence, it is very hard and error prone to use LAPACK for day to day use.

Armadillo [4] is an open source linear algebra library written in C++ programming language. It was specifically designed for computationally intensive numerical computing applications. One of the main objectives of the Armadillo library is to have a good balance between speeds and ease of use. Since Armadillo is designed to work with the C++ programming language, it is not possible for people who don’t have a good C++ knowledge to use the Armadillo library for their applications.

MATLAB is a numerical computing environment, and a fourth generation programming language designed and developed by MathWorks Inc [11]. MATLAB consists of a rich set of features in order to manipulate matrices, plotting graphs, data manipulation, and experimenting with algorithms. MATLAB makes it quite easy to code complicated algorithms involving vectors and matrices. The MATLAB programming language is a weakly, dynamically typed interpreted programming language.

GNU Octave is a high-level language primarily designed for numerical computing. Octave provides a set of tools for solving common numerical linear algebra problems, such as finding roots of nonlinear equations and manipulating polynomials. It is freely available under the GNU General Public License (GPL) [2].

As mentioned in Section I, Conrad Sanderson [4] has discussed issues associated with MATLAB and Octave.

3 MatPro language

The philosophy of the MatPro language is that, people with little knowledge in programming, should be able to write non-trivial mathematical programs in multithreaded fashion with minimum effort. Further, the MatPro compiler should be able to generate typed-safe executable programs with acceptable level of performance.

Program 1 Solving a system of linear equations in MatPro

```
    %System of linear equations
    % x  + 3y - 2z = 5
    % 3x + 5y + 6z = 7
    % 2x + 4y + 3z = 8
    mat A = [1,3,-2; 3,5,6; 2,4,3]
    mat b = [5;7;8]
    mat result = Inverse(A)*b
```

MatPro’s built-in data types are **int**, **double**, **string**, **bool**, and **mat**. In addition to these, arrays are also a supported data type in the MatPro language. Further, MatPro comes with selection and iterative constructs. For example, for iterations MatPro uses the **for** and **while** loops whilst, for selection MatPro uses the **if-else** construct.

In addition to the common scalar operations, MatPro is capable of handling most of the common matrix operations. For example, the following MatPro program solves a system of linear equations.

3.1 Multithreaded Programming in MatPro

One of the key features of MatPro language is its multithreading support. Unlike other mathematical programming languages, multithreading is a built-in feature in the MatPro language. The multithreading programming model in MatPro is based on the dynamic multithreading model described by Charles E. Leiserson and et al [5].

spawn, sync, and **parallel** are the main keywords used in the dynamic multithreading model. The keywords **spawn** and **sync**, are used to represent nested parallelism, whereas the **parallel** keyword is used to represent parallel loops. According to Charles E. Leiserson and et al [5], the dynamic multithreading model has some advantages over other multithreading models as given below:

- Simple extension to serial programs using three keywords. The programmer can describe multithreading programs by adding three keywords to the serial version of the program.
- Many multithreading algorithms involving nested parallelism follow naturally from the divide-and-conquer paradigm.
- The model is faithful to how parallel-computing practice is evolving.

In this research we have implemented nested parallelism using the **spawn**, and **sync** keywords. Parallel loops will be implemented in the next version of the language. In the rest of the section we will be presenting several parallel programs written in the MatPro language.

Program 2 Simple multithreaded program

```
%A simple multithreaded program
func int MethodOne(int x){
    return x
}
func int MethodTwo(int} y){
    return y
}
%simple driver
int x = spawn MethodOne(100)
int y = spawn MethodTwo(200)
sync
print "x :" + x
print "y :" + y
```

Program 3 Fibonacci number calculator

```
%Recursive Fibonacci number
%calculator
func int Fib(int x){
    if x <= 1 {
        return 1
    }else{
        int p = Fib(x-1)
        int q =  spawn Fib(x-2)
        sync
        return q +p
    }
}
%Sample driver program
int fibNumber = Fib(5,5)
print "Answer: "+fibNumber
```

Program 2 shows a simple multithreaded program written in the MatPro language. The program spawns two threads and two methods identified as MethodOne, and MethodTwo are running on these two threads. Furthermore, at the end of the program the two methods return their output to the main thread. The **sync** statement blocks the main thread, till the two spawned threads complete their work.

Program 3 calculates the Fibonacci sequence using the multithreaded approach. The Fib method takes an integer as an argument say x and returns x^{th} Fibonacci number of the Fibonacci sequence. The Fib method spawns a separate thread to calculate Fib(x-1) and waits at **sync** until Fib(x-1) returns its value.

4 MatPro Compiler

The MatPro compiler is completely written in the C# programming language. In order to improve the compilation speed and the flexibility of the compiler code base, we implemented the scanner and the parser of the MatPro compiler by hand. Since MatPro is a strongly, statically typed programming language, type checking is essential and it should be done during the compile time. The MatPro compiler performs type checking while building the Abstract Syntax Tree (AST) [12]. The next two phases of the compiler are building the intermediate representation from the parse tree and performing machine independent optimization. We have omitted these two steps due to the following reasons:

- Safonov, Vladimir O. [13] has pointed out that”Due to the two-step architecture of code generation on Microsoft.NET–compile-time generation of MSIL code and runtime generation (JIT compilation) of the native target platform code–the developers of the compiler from a source language to MSIL may not take care of any optimizations. All optimizations are delayed until runtime and JIT compilation.”

- Converting AST to a low level representation such as three address code will not add any value addition, since the MatPro compiler doesn’t do any machine independent optimization due to the reason mentioned above. Furthermore, converting AST to a low level representation will add an extra overhead to the compilation process of MatPro programs.

Therefore, instead of converting to a low level intermediate representation such as, three address code and performing machine independent optimization, we directly generate the CIL by walking the AST using the visitor pattern [14].

4.1 Generating CIL for MatPro Programs

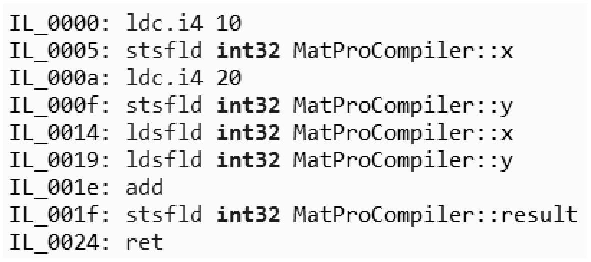
Code generation for MatPro programs can be divided into two parts: code generation for serial MatPro programs and code generation for parallel MatPro programs.

Generating CIL for the serial MatPro source codes is a straightforward process. For example, consider the MatPro program shown in Program 4.

Program 4 Integer addition in MatPro

```
int x = 10
int y = 20
int result = x + y
```

Figure 1 shows the generated CIL for the above MatPro program. According to the generated CIL, it first loads the constant 10 into the evaluation stack and stores it in the variable x. Similarly, it loads 20 into the evaluation stack and stores it in the variable y. Next it loads variable x and y into the evaluation stack and does the addition operation. Finally it stores the output of the addition in the result variable.



```
IL_0000: ldc.i4 10
IL_0005: stsfld int32 MatProCompiler::x
IL_000a: ldc.i4 20
IL_000f: stsfld int32 MatProCompiler::y
IL_0014: ldsfld int32 MatProCompiler::x
IL_0019: ldsfld int32 MatProCompiler::y
IL_001e: add
IL_001f: stsfld int32 MatProCompiler::result
IL_0024: ret
```

Figure 1 CIL code for adding two integers

Similarly we can produce CIL for serial codes involving matrix operation. For implementing matrix operations such as matrix addition, matrix inversion we used a separate library called Math.NET Numerics [15].

Generating CIL for the parallel MatPro code is complicated compared to generating CIL for serial MatPro programs. Further, even though it is possible to implement multithreading using basic facilities provided by CIL, it is more manageable when using a separate threading library for this purpose. In the .NET platform the most convenient and efficient library for parallel programming is the Task Parallel Library (TPL) [16]. TPL scales the degree of concurrency dynamically in order to most efficiently use all the processors available in the system [17]. Further, TPL handles partitioning of the work, scheduling of threads in the Thread Pool, cancellation support, state management, and other low-level details [17]. As described in section III, most of the dynamic multithreading features are present in TPL. This was the main reason for using TPL for implementing multithreading in the MatPro compiler.

5 Performance of MatPro Compiler

Same as other software packages, compilers also should be tested and evaluated before being used for real-world applications. Therefore, we have conducted several performance tests in order to gauge the performance of the MatPro compiler. All the tests were carried out under the following conditions:

- Operating System: Windows 7 Starter 32 bit.
- Computer Speed: Intel(R) Atom (TM) CPU N450 @ 1.66GHz.
- Compiler Memory: 2 GB.

5.1 Performance of Matrix Operations

We have selected several computationally intensive matrix operations in order to measure the performance of matrix operations. Matrix operations of the MatPro language are compared with matrix operations of the C# language and C++ language. For implementing the matrix operations in C#, we used the Math.NET Numerics [15] matrix library, which is the same library we used to implement the matrix data type in the MatPro language. For implementing matrix operations in C++ language, we used the Armadillo [4] matrix library. In this section we will describe these matrix operations and will present the relative performance of these operations between MatPro, C#, and C++ languages. Table I shows the matrix operations used for matrix benchmarking.

TABLE I. MATRIX OPERATIONS USED FOR BENCHMARKING

Benchmark	Description
Matrix Multiplication	Matrix size starts with 2x2 square matrix and double the matrix size in each iteration up to 1024x1024.
Matrix Inversion	Matrix size starts with 2x2 square matrix and each iteration double the matrix size up to 1024x1024.

Graphs in Figure 2, and Figure 3 shows the performance comparisons between the three languages for matrix multiplication and inversion. According to these two graphs matrix multiplication and inversion performances of C# and MatPro languages are identical. Moreover, matrix multiplication and inversion performance of C++ language is better than C# and MatPro languages.

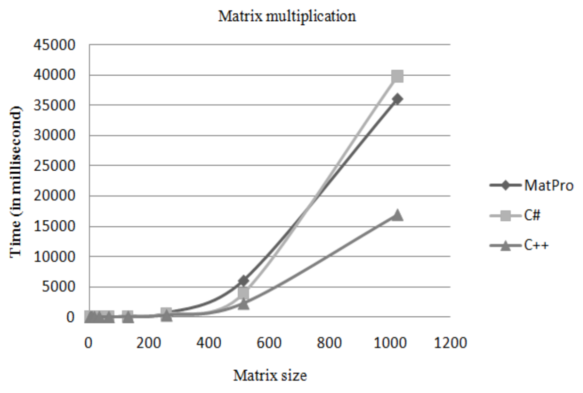


Figure 2 Performance comparison of matrix multiplication

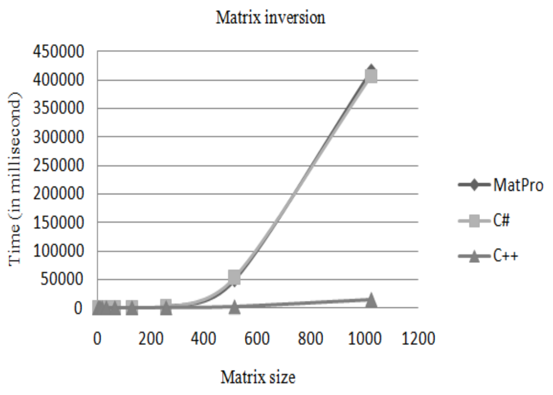


Figure 3 Performance comparison of matrix inversion

5.2 Performance of Parallel Programs

In this section we present the performance of parallel programs written in MatPro language. We used four programs and each program was implemented in both parallel and serial fashion. Then we used these eight programs as a benchmark in order to take measurements and compare the performance between parallel and serial programs. Table II shows the descriptions of these programs.

TABLE II. PROGRAMS USED FOR BENCHMARKING PARALLEL AND SERIAL PROGRAMS IN MATPRO

No:	Program name	Description
1	Ackermann [18]	Calculates Ackermann(3,13) three times in both parallel and serial techniques.
2	Trapezoidal Rule [19]	Calculates (log $(1/x^3)$ - log $(1/x^2)$) from 1.0 to 10000.0 both in parallel and serial techniques.
3	Simpson's rule [19]	Calculates (log $(1/x^3)$ - log $(1/x^2)$) from 1.0 to 10000.0 both in parallel and serial techniques.
4	Bubble sort [5]	Sorting 10 integer arrays, each consists of 50,000 elements using both parallel and serial techniques.

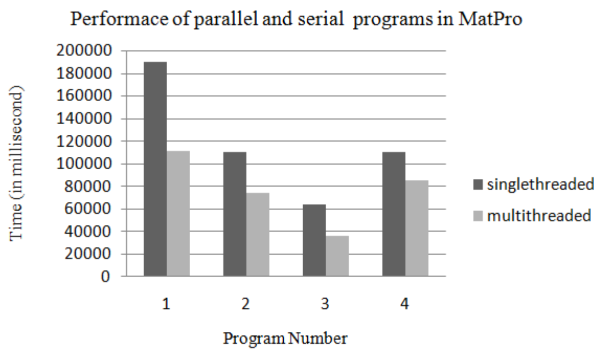


Figure 4 Performance of parallel and serial programs in MatPro

The runtime performances of above benchmarks are shown in Figure 4. According to Figure 4 it is clear that, multithreading in MatPro has improved the performance of programs discussed in Table II. Further, it is to be noted that, it is not possible to convert any serial program into a parallel program and improve the performance. But if there is a possibility to convert a serial program to a parallel program without destroying the semantics of the serial program, it is possible to gain a good performance improvement by using parallel programs in MatPro language.

6 Conclusion and Future Works

In this paper we presented a new programming language called MatPro for numerical computing. The main feature of the MatPro language is its built-in support for multithreading. MatPro's multithreading model is based on the dynamic multithreading model.

Designing, implementing, and testing a programming language and a compiler within a very short time period is not possible. Therefore, we can observe several limitations in the current version of the compiler. Among those limitations, inadequate performance of matrix operations is the most noticeable limitation of the MatPro compiler. Presently we are using Math.Net Numerics [15] library for implementing the matrix data type in the MatPro compiler. According to our testing, performance of matrix operations in MatPro language depends on the performance of the Math.Net Numerics library. Therefore, in the future we will be using a high-performance library for implementing matrix data type in the MatPro compiler.

Furthermore, we will be adding new features to the MatPro in order to improve the performance and expressiveness of the MatPro language. Among those features, the most important features are given below:

- Present version of the compiler doesn't support exception handling. In order to develop robust programs in MatPro we believe exception handling is an essential feature we need to add to the MatPro compiler.
- Present version of the compiler doesn't support file I/O operations. Therefore, we will be adding this feature in the next version of the compiler.
- Ability to calling precompiled MatPro libraries from a MatPro program is an essential feature missing in the current version of the MatPro compiler. We will be implementing this in the next version of the MatPro compiler.
- Since MatPro compiler generates intermediate language instructions for Microsoft's .NET platform, it is an added advantage to facilitate inter-operability between MatPro and other .NET languages.
- The present version of the compiler has very limited mathematical functions. In the next version of the MatPro compiler we will add a lot more mathematical functions to the MatPro compiler.

7 References

[1]. "MATLAB - the language of technical computing [online]." http://www.mathworks.com/products/matlab/, Accessed [2011-JAN-23].

[2]. "Octave- high-level language for numerical computing [online]." Availalbe: http://www.gnu.org/software/octave/, Accessed [2011-JAN-23].

[3]. "Scilab - the free software for numerical computation [online]." Available:http://www.scilab.org/, Accessed [2011-JAN-23].

[4]. C. Sanderson, “Armadillo: An open source c++ linear algebra library for fast prototyping and computationally intensive experiments,” tech. rep., Technical report, NICTA, 2010.

[5]. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, Third Edition. The MIT Press, 3 ed., Sept. 2009.

[6]. J. Richter, CLR via C#, Second Edition. Microsoft Press, 2nd ed. ed., Mar. 2006.

[7]. “Clapack f2c’ed version of lapack [online].” Available: http://www.netlib.org/clapack/, Accessed [2011-JAN-23].

[8]. M. Kupferschmid, Classical Fortran: Programming for Engineering and Scientific Applications, Second Edition. CRC Press, 2 ed., Jan. 2009.

[9]. B. W. Kernighan and D. M. Ritchie, C Programming Language. Prentice Hall, 2 ed., Apr. 1988.

[10]. B. Stroustrup, The C++ Programming Language: Special Edition. Addison-Wesley Professional, 3 ed., Feb. 2000.

[11]. “Mathworks - matlab and simulink for technical computing [online].” Availalbe: http://www.mathworks.com/, Accessed [2011-JAN-23].

[12]. A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, Compilers: Principles, Techniques, & Tools with Gradiance (pkg). Addison Wesley, 2 ed., Oct. 2007.

[13]. V. O. Safonov, Trustworthy Compilers. Wiley, 1 ed., Mar. 2010.

[14]. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1 ed., Nov. 1994.

[15]. “Math.NET numerics[online].” Available: http://mathnetnumerics.codeplex.com/,Accessed [2011-JAN-23].

[16]. J. Albahari and B. Albahari, C# 4.0 in a Nutshell: The Definitive Reference. O’Reilly Media, fourth edition ed., Feb. 2010.

[17]. “Task parallel library [online].” http://msdn.microsoft.com/enus/ library/dd460717.aspx, Accessed [2011-AUG-23].

[18]. B. W. Kernighan and R. Pike, Unix Programming Environment. Prentice Hall Ptr, Mar. 1984.

[19]. F. B. Hildebrand and Mathematics, Introduction to Numerical Analysis: Second Edition. Dover Publications, 2 ed., June 1987.