# The Mining Integrated Programming Language (MIPL)

*A simple rule-based, portable, high performance, dynamic programming language for Data Mining*

**A White Paper**

**February 22, 2012**

**Younghoon Jeon, Young Hoon Jung, Jinhyung Park, Wonjoon Song, Akshai Sarma**

**{yj2231, yj2244, jp2105, dws2127, as4107}@columbia.edu**

## 1. Introduction to MIPL

MIPL is a language for Data Mining which provides methods to implement various sophisticated matrix-based algorithms and to retrieve the computed data easily by offering a novel, convenient way that converts matrices into a set of facts and vice versa.

## 1.1 Design Goals of MIPL

Most of the languages that provide matrix-based computations, such as MATLAB or R, require the user to be fluent in complex syntaxes and a number of functions for matrix manipulations. MIPL, on the other hand, provides two simple yet different aspects of the language to every user independent of their proficiency in Data Mining. The language handles different skills and interests by separating the data retrieval part of the language from the algorithm implementation part, each of which plays a distinct role in the language. In addition to MIPL's approach to matrix-based algorithm implementation, its flexible architecture allows it to easily accelerate and parallelize the code using GPGPU [1] and MapReduce [2].

### 1.1.1 FRQ: Facts, Rules and Queries

The data retrieval part of the language is called FRQ, which stands for Facts, Rules, and Queries. From the user perspective, this part of MIPL is similar Prolog [3], which supports logic programming in a declarative way. However, the syntax is extended for the purposes of Data Mining and supports various useful constructs such as regular expressions in rules and facts, allowing the user to express complex sentences elegantly and concisely. This also allows the user to very easily describe the data to be retrieved by defining rules and facts and then defining queries. On the top of MIPL's FRQ, even a casual programmer who has no knowledge of MATLAB [4], can easily write a data retrieval program, utilizing pre-written algorithms.

#### 1.1.2 JOB: For Efficient Matrix-based Algorithm Development

While the Prolog like syntax in the FRQ provides the users an easy way to retrieve data, it is not a viable method for the development of algorithms to use for the data retrieval. The FRQ's syntax, which uses nested list types is not convenient nor is it optimized for complex matrix operations or parallelization. To solve this issue, MIPL has a second component, called JOB, which uses separated syntax from FRQ. JOB provides the algorithm implementation part that takes inspiration from MATLAB's syntax, to provide an efficient means of representation for algorithm creation. JOB is intended for serious users, who have sufficient programming experience and want to create a new "JOB". To facilitate this, JOB also provides various linear algebra primitives that constitute key building blocks for various sets of Data Mining algorithms. By separating JOB from FRQ, MIPL can be used by users with a varying degree of Data Mining and programming experiences. It frees the users who simply want to create a data retrieval program from understanding or creating complex algorithms by separating and delegating this task to more serious developers or researchers.

#### 1.1.3 MFC: Matrix Facts Conversion

FRQ and JOB are logically separated for the user but can work together in synergy through the MFC, which stands for Matrix Facts Conversion. This is one of the key concepts of MIPL. The MFC allows facts in FRQ to be converted to matrices, used for input arguments of a JOB. Conversely, the output matrix from JOB can be converted to a set of facts through MFC, so that FRQ users can use these facts in combination with other rules and facts for further computation. The MFC collects a rule with a set of facts under a matrix, much like how it would be represented as a table.

## 2. MIPL Language Concept

In order to characterize MIPL, we define some of the key features of our language. MIPL is a platform and architecture neutral, simple yet powerful, portable, dynamic, and high performance language.

### 2.1 Architecture Neutral

Similar to Java, MIPL's target code is designed to run on a virtual machine. MIPL currently produces Java Bytecode itself as the compilation output, allowing it to be run on Java Virtual Machines for architecture neutrality.

### 2.1.1. (Even) Virtual Machine Neutral

MIPL is not limited to a particular virtual machine. Because MIPL is open, through the modular plugin architecture, the MIPL compiler can be augmented to generate different types of intermediate codes at the behest of its users. By default, MIPL generates Java Bytecode that runs on Java Virtual Machine. Users are free to insert a new code generator so that the generated code can run on a new type of virtual machine. This means MIPL is independent from virtual machines as well as from architecture.

## 2.2 Simple

MIPL's FRQ is simple to use. A user with only a little programming experience can quickly learn the syntax of the FRQ and create a complete Data Mining program very quickly. However, the simplicity does not sacrifice the capabilities of MIPL as JOB can be used to develop by experienced users to generate powerful new algorithms by using and defining in-built operations.

## 2.3 Portable

As MIPL is both architecture and virtual machine independent, it gives the user high portability. When a certain virtual machine is not portable to the target machine, the MIPL user can replace the code generation module with one that does match the target.

## 2.4 Dynamic

All rules, facts, and JOBs are evaluated at run time upon the user's query from the command prompt or the MIPL program. This means, dynamic type checking including variable types or matrix computability is determined at run time, unlike static typing, which is done at compile time.

### 2.4.1 Dynamic and Weak Typing

Each variable is declared without a type. In other words, MIPL is dynamically typed. Also, MIPL employs weak typing, or implicit type casting, for which MIPL checks type compatibility while converting variables at runtime. With dynamic typing and type conversion, MIPL users can produce code very fast, similar to a scripting language.

### 2.4.2 Lazy Rule Binding

MIPL deals with the binding of rules and facts in the FRQ at runtime, deferring the actual execution of rule binding. This provides for fast and easy compilation as well as optimized execution. MIPL includes a command prompt that allows the user to enter queries. As the query is applied, binding is done on the rules and facts.

## 2.5 High Performance

MIPL is designed to have high performance. With the vast amounts of data dealt with in the Data Mining field, it is imperative that the language designed for it be of high performance. As such, MIPL’s flexible architecture allows the users to replace plugins for execution module to enable notable optimization techniques, which are Parallelization and GPGPU computation.

### 2.5.1 Implicitly Parallel

MIPL’s default execution module includes a MapReduce component which will be used for MIPL JOB code execution. Because JOB code consists of matrix computations, it is inherently parallelizable. Thus, after generating code, the MIPL execution environment can be linked to run on the MapReduce framework to efficiently compute on large data sets and complex rules, in a fault-tolerant and scalable way.

### 2.5.2 Hardware Accelerated

As recent computing nodes come with GPUs and GPUs performs well in matrix computations, MIPL also provides a way to utilize accelerators, in most cases GPUs, on the local or remote nodes. MIPL's default execution module utilizes GPUs on MapReduce nodes through OpenGL APIs.

## 3. MIPL Use Case Scenarios

Here are some brief use case scenarios that can happen with MIPL.

### 3.1 Advertisement Pricing

When a web-platform administrator needs to determine how much of advertisement pricing is appropriate for each web page, the PageRank algorithm can be used. With page relation data gathered and a PageRank algorithm implemented in MIPL, the user can calculate proper values for advertisement on each page by launching PageRank every day.

### 3.2 Weather Forecast

Another obvious example is in weather forecast, as most of the users want to retrieve weather forecast information easily while not having to deal with sophisticated algorithms like regression or random forest and rather they be hidden behind the scene. Once a weather forecast algorithm is implemented in JOB, the user can retrieve the forecasted weather data from the massively huge raw data, by giving several conditions, like in California, from 3-6PM tomorrow.

## 4. Summary

The MIPL is a simple yet powerful programming language that exposes distinctly optimized features for the naïve Data Mining users and advanced Data Mining developers. Apart from its simplicity, MIPL's dynamic, high performance, and portable characteristics ensure that its users reach their goals in the optimal way.

## 5. Reference

[1] VECPAR 2006, S. Ohshima, Parallel Processing of Matrix Multiplication in a CPU and GPU Heterogeneous Environment, International Meeting on High Performance Computing for Computational Science, 2006

[2] J. Dean, MapReduce: Simplified Data Processing on Large Clusters, USENIX Symposium on Operating Systems Design and Implementation, 2004

[3] Mathworks MATLAB homepage: http://www.mathworks.com/products/matlab

[4] SWI-Prolog homepage: http://www.swi-prolog.org