Calling Functions Dynamically Generated By Eclipse OMR JitBuilder

Georgiy Krylov, Gerhard W. Dueck, Kenneth B. Kent
Faculty of Computer Science, University of New Brunswick
Younes Manton
IBM Canada
{gkrylov|gdueck|ken}@unb.ca {ymanton@ca.ibm.com}

Problem Statement

Calling dynamically generated functions is a complex task in terms of setting up the parameters and return values. This is a study of the complexity and performance of multiple approaches to addressing this issue.

Definitions

- **Eclipse OMR** – Collection of language-agnostic tools for language runtime environment construction
- **Eclipse OMR JitBuilder** – an API, simplifying access to the compiler.
- **Libffi** – Multi-platform open source project assisting with setting up function calls

Example scenario

- Interpreter stack is composed of `Value`s –union type for `int32`, `int64`, `float`, `double`, `128-bit int`
- A function pointer of `void (*) (...)` type is compiled;
- Interpreter stack changes before call
- Call is executed
- Interpreter stack changes after call

Complexity of the problems grows exponentially

- Number of types the language supports
- Number of parameters and their order
- Number of return values (0 or 1)
- Void return type is supported, so `fn();` is a valid call

The maximum number of possible calls is

$$\#OfTypes \sum_{p=0}^{max\#OfParameters} (\#OfTypes - 1)^p$$

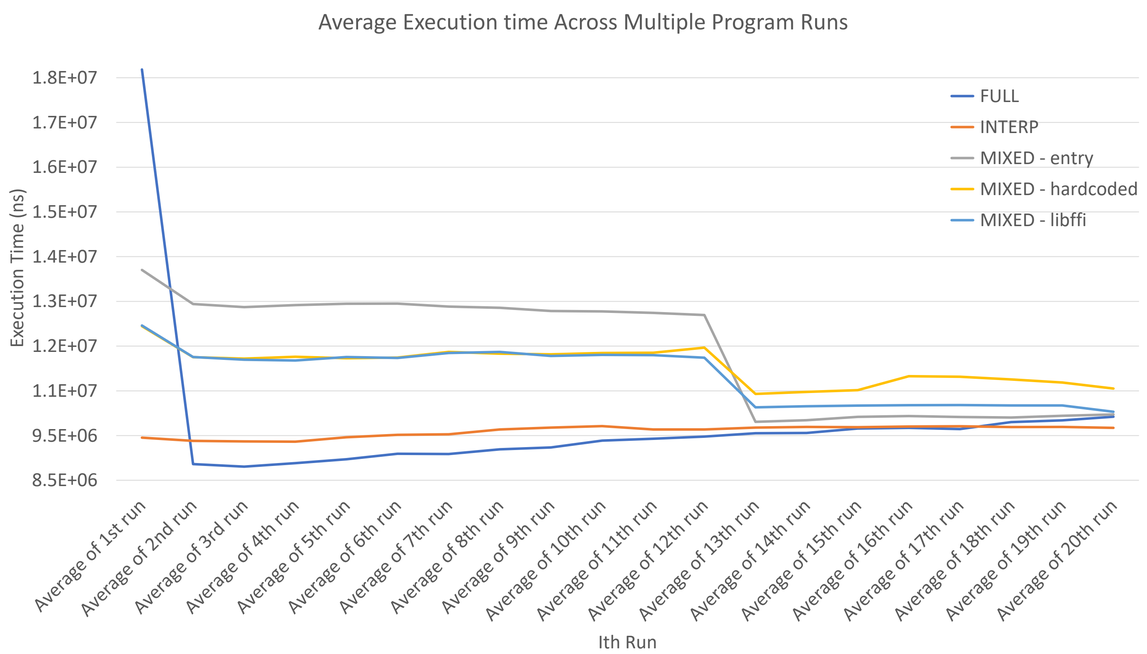
The program complexity grows exponentially

```
switch(numberOfParams){
 case 1 :
  switch(numberOfReturnValues){
   case 1:
    switch(returnType){
     case Type::I32:
      switch (param0Type){
       case Type::I32:{
uint32_t rv;
rv=reinterpret_cast<uint32_t (*)(uint32_t)>(fn)(param0);
Push(rv);
```

Proposed solutions and their characteristics

- Hard-coding a switch table
- Generating functions using OMR JitBuilder to modify the interpreter stack and execute call – entry functions
- Using a library solution

	Scalability	Ease of use	Performance
Hardcoded	Worst	Average	Better
Entry functions	Easy	Average	Varying



For a program that has ten methods calling one another in a chain

Future work

- Preparing a call at runtime using
  `#include <functional>`
- Explore boost library

CAS–Atlantic