Just-in-time Compilation (JIT)

Seminar: Automation, Compilers, and Code-Generation
Chair: High Performance and Automatic Computing
RWTH AACHEN UNIVERSITY
Safdar Dabeer Khan
Outline

- Overview
  - Static Compilation
  - Virtual machines
  - Traditional Approaches
- JIT
  - Defining JIT
  - JIT: A Combination of two traditional approaches
- Working Mechanism
  - Conceptual Idea
  - Technical steps
  - Advantages & Drawbacks
- Applications
- Exploring JIT & java
  - Compilation in java
  - VM & JVM
  - JIT in JVM
  - Runtime optimizations by JIT
- Startup Delay and Possible Optimizations
  - Java HotSpot
Compiler and Interpreter

Basic conceptual view of both techniques, remember they are not mutually exclusive.

- Performs actions described by high level program.
- Generate machine code and then walk a parse tree for execution
  OR
- Generate and execute intermediate software-defined instructions.
- Perform some conversion work every time a statement or function is executed

- Produce machine code directly executable by computer hardware.
- Generates a stand-alone machine code program.
- Can make almost all conversions from source to the machine level at once.
Hybrid Compilation/Interpretation

Which one is better?

- Compilation
- Interpretation
- Hybrid
Static Compilation

- Translate from high level language to machine code.
- All bindings are done at compile time.
- Linking is done during the creation of an executable.
- Linker resolves the referenced symbols.

👍 Robust, better security, before hand optimization, reduced start-up cost

👎 Compatibility concerns, Less opportunity for performance improvement, dynamic traits exploitation, infeasible speculative optimization
Ahead of Time Compilation (AOT)

“Performs compilation before execution rather than during execution.”

- Trade offs:
  - Memory
  - Starting time
  - Portability
  - Optimizations
Outline

- Overview
  - Static Compilation
  - Virtual machines
  - Traditional Approaches
- JIT
  - Defining JIT
  - JIT: A Combination of two traditional approaches
- Working Mechanism
  - Conceptual Idea
  - Technical steps
  - Advantages & Drawbacks
- Applications
- Exploring JIT & java
  - Compilation in java
  - VM & JVM
  - JIT in JVM
  - Runtime optimizations by JIT
- Startup Delay and Possible Optimizations
  - Java HotSpot
“Just-In-Time (JIT) compilation, also known as dynamic translation, is compilation done during execution of a program at run time rather than prior to execution” [*]
JIT: A combination of approaches

- JIT compiler represents a hybrid approach.
- “Speed of compiled code” and “Flexibility of Interpretation”
- Combining two approaches brings pros and cons of both techniques.

- Selectively compile the most frequently executing methods to native code during execution.
Cont.

- Conceptual view of JIT
  - Translate continuously.
  - Perform caching of compiled code.
  - Minimizes lag on future execution of same code during a given run.
Outline

- Overview
  - Static Compilation
  - Virtual machines
  - Traditional Approaches

- JIT
  - Defining JIT
  - JIT: A Combination of two traditional approaches

- Working Mechanism
  - Conceptual Idea
  - Technical steps
  - Advantages & Drawbacks

- Applications

- Exploring JIT & java
  - Compilation in java
  - VM & JVM
  - JIT in JVM
  - Runtime optimizations by JIT

- Startup Delay and Possible Optimizations
  - Java HotSpot
Conceptual Idea

- Compiler
Cont..

- Interpreter

```plaintext
SourceFile.src
1. X = 10;
2. Y = 20;
3. Display(z);
4. Z = X + Y;
5. Display(A);
6. Display(z);

ObjectFile.obj
1BC01AD676
1001016A7
```
Cont.

- JIT
Cont.

- JIT
Technical Steps

- We can divide JIT into distinct phases mainly:
  - Machine code creation at runtime.
  - Machine code execution at runtime.

- Machine code creation

  This step is similar to what every compiler does with slight difference.
  - Create machine code at program run time.
  - Use building blocks for keeping code in memory for execution later.
  - Easier to write
Machine code execution (involved roughly three main steps):

- Allocate a readable, writable and executable chunk of memory on the heap.
- Copy the machine code implementing intermediate code into this chunk.
- Execute code from this chunk by casting it to a function pointer and calling through it.

Example: (for details, please visit link)

```c
void* alloc_executable_memory(size_t size) {
    void* ptr = mmap(0, size,
                     PROT_READ | PROT_WRITE | PROT_EXEC,
                     MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
    if (ptr == (void*)-1) {
        perror("mmap");
        return NULL;
    }
    return ptr;
}
```

```c
long add4(long num) {
    return num + 4;
}
```

```c
void emit_code_into_memory(unsigned char* m) {
    unsigned char code[] = {
        0x48, 0x89, 0xf8,       // mov %rdi, %rax
        0x48, 0x83, 0xc0, 0x04, // add $4, %rax
        0xc3                     // ret
    };
    memcpy(m, code, sizeof(code));
}
```

```c
const size_t SIZE = 1024;
typedef long (*JittedFunc)(long);
void run_from_rwx() {
    void* m = alloc_executable_memory(SIZE);
    emit_code_into_memory(m);
    JittedFunc func = m;
    int result = func(2);
    printf("result = %d\n", result);
}
```
Advantages and Drawbacks

- Faster execution.
- Easier handling of late bound data types.
- Enforce security guarantees.
- Can be optimized to targeted CPU and operating system
- Portable byte code.
- Can use profile information to perform optimizations.
- Can perform other many different runtime optimizations.

- Startup delay.
- Limited AOT optimizations because of time.
- Compiler should be packaged inside virtual machine.
- Can not perform complex optimizations which are possible with static compilation.
- Maintenance and debugging can be a headache.
- Security concerns
Outline

- Overview
  - Static Compilation
  - Virtual machines
  - Traditional Approaches
- JIT
  - Defining JIT
  - JIT: A Combination of two traditional approaches
- Working Mechanism
  - Conceptual Idea
  - Technical steps
  - Advantages & Drawbacks
- Applications
- Exploring JIT & java
  - Compilation in java
  - VM & JVM
  - JIT in JVM
  - Runtime optimizations by JIT
- Startup Delay and Possible Optimizations
  - Java HotSpot
Applications

Many different companies/organizations have adopted JIT in their tools, some of renown are:

- **Oracle Java**
  - The Just-In-Time (JIT) compiler is a component of the Java™ Runtime Environment that improves the performance of Java applications at run time.

- **Microsoft .NET Framework**
  - The JIT compiler is part of the Common Language Runtime (CLR). The CLR manages the execution of all .NET applications.

- **JIT in web browsers**
  - Trace Monkey is a trace based JIT compiler used by Mozilla Firefox browser to run JavaScript programs

- **LLVM**
  - intro
Outline

- Overview
  - Static Compilation
  - Virtual machines
  - Traditional Approaches
- JIT
  - Defining JIT
  - JIT: A Combination of two traditional approaches
- Working Mechanism
  - Conceptual Idea
  - Technical steps
  - Advantages & Drawbacks
- Applications
- Exploring JIT & java
  - Compilation in java
  - VM & JVM
  - JIT in JVM
  - Runtime optimizations by JIT
- Startup Delay and Possible Optimizations
  - Java HotSpot
Compilation in java

Conceptual view of code compilation in java.

- Java source code is compiled by java compiler resulting in JVM readable java byte code.
- JVM performs following two main steps:
  - Compiles byte code at runtime into machine readable instructions
  - Execute compiled machine readable code

Java source code → Java compiler → Java byte code → JVM → JIT placed inside JVM

Operating System (OS) → Hardware
Virtual Machine

- Different kind of virtual machines provide different functions.

- Some of the important goals of VM to consider:
  - Portability.
  - Bridge the gap between compilers and interpreters.

- A virtual machine need at least following three basic components:
  - Interpreter
  - Runtime Supporting System
  - Collection of libraries

- Some of the major concerns:
  - Efficiency
  - Multiple VM’s concurrency issue.
  - Compatibility with host for malware protection.
JVM comprises following main features:

- **Runtime**
  - Mainly handles class loading, byte code verification and other required functions.
- **JIT**
  - Profiling, compilation plans, optimizations
- **Garbage Collection**
JIT in JVM

Improves the performance of Java programs by compiling byte code into native machine code at run time.

- JIT compiler is by default enabled, however it gets activated when a Java method is called.

- Performs on runtime: Method code → Machine code

  JVM having the machine code does need to interpret it, results in improving processor time and memory usage

- JIT compilation threshold helps to take action.

- JIT recompilation threshold helps to make optimization decisions.
JIT’ing requires Profiling

- Collect data during execution:
  - Executed functions
  - Executed paths
  - Branches
  - Parameter values

- Collecting data at right time:
  - Early or late phase
  - Continue or intermittent way

- Collecting data by:
  - Sampling
  - Program instrumentation
  - Using hardware performance measures

- Use collected data for:
  - Optimizations
Runtime Optimizations by JIT

During the compilation performed by JIT, it performs following main optimization steps:

- Inlining
- Local optimizations
- Control flow optimizations
- Global optimizations
- Native code generation
Inlining

“Replaces a function call site with the body of the called function” [*]

- Trees of smaller methods are "inlined", into the trees of their callers.

```
Before
int addAll(int max) {
    int accum = 0;
    for (int i = 0; i < max; i++) {
        accum = add(accum, i);
    }
    return accum;
}
int add(int a, int b) { return a + b; }

After
int addAll(int max) {
    int accum = 0;
    for (int i = 0; i < max; i++) {
        accum = accum + i;
    }
    return accum;
}
```
Optimizations performed in this phase are:

- **Trivial Inlining**
  - Inlining short, simple functions can save both time and space

- **Call graph inlining**
  - Create a call graph and evaluate important parts by traversing.

- **Tail recursion elimination**
  - Similar to tail-call elimination with added constraint i.e. calling itself.

- **Virtual call guard optimizations**
  - Perform by devirtualization

What about?
Local Optimizations

“Improve small portion of code at a time”

Mainly includes:

- Local data flow analyses and optimizations
  - Information collection about the data flow values across basic blocks.
  - Compute data flow equations and optimize such as:
    - Ambiguous or duplicate definitions
    - Remove redundant expressions

- Register usage optimization

- Simplifications of Java™ idioms
  - VarargsCollectionFactoryMethod
Control flow optimizations

“Analyze the flow of control inside a code section and rearrange code paths to improve the efficiency.”

Mainly includes:
- Code reordering
- Loop optimizations
  - Inversion
  - Reduction
  - Versioning and specialization
- Switch analysis
- Dead code elimination
Global optimizations

“Perform optimizations on entire method at once”

Mainly includes:

- Global data flow analyses and optimizations
- Optimizing garbage collection and memory allocation
- Partial redundancy elimination
- Optimizing synchronizations
Native Code Generation

Performing optimization during native code generation depends upon the underlying architecture, generally it performs:

- Translation of method trees into machine code.
- Perform minor optimizations as required.
Outline

- Overview
  - Static Compilation
  - Virtual machines
  - Traditional Approaches
- JIT
  - Defining JIT
  - JIT: A Combination of two traditional approaches
- Working Mechanism
  - Conceptual Idea
  - Technical steps
  - Advantages & Drawbacks
- Applications
- Exploring JIT & java
  - Compilation in java
  - VM & JVM
  - JIT in JVM
  - Runtime optimizations by JIT
- **Startup Delay and Possible Optimizations**
  - Java HotSpot
Startup Delay by JIT

- Time taken by JIT to load and compile the byte code cause delay in preliminary execution. This initial delay is known as “startup delay”

- For having better generated code, JIT performs more optimizations which also increase startup delay.

  \[ \text{Increased Optimizations} \quad \alpha \quad \text{Better Code Generation} \]

  \[ \text{Increased Optimizations} \quad \alpha \quad \text{Startup delay} \]

- Increased startup delay can also be because of IO-bound operations
HotSpot

“Combines interpretation, profiling, and dynamic compilation”

- Initially it runs as an interpreter and only compiles the "hot" code
- Performs profiling to identify frequently execute code sections.
- Time is saved by not compiling the infrequent code.
- Profiling data help to improve decision making for optimizations.
- Apply adaptive optimization technology, includes:
  - HotSpot Detection
  - Method Inlining
  - Dynamic Deoptimization
HotSpot comes with two compilers:

- **The client compiler**
  - Reduce application startup time.
  - Reduce memory footprint.
  - Less time for compilation

- **The server compiler**
  - Intended for long-running server applications.
  - Maximize peak operating speed.
  - Apply complex optimizations.
HotSpot Optimizations

HotSpot include number of complex and advanced optimizations, some of them are mentioned below:

- Deep inlining:
  - Method inlining combined with global analysis and dynamic deoptimization

- Fast instanceof/checkcast
  - Accelerating the dynamic type tests

- Range check elimination:
  - Surety about the index bound to remove index bound check.

- Loop unrolling:
  - Enables faster loop execution

- Feedback-directed optimizations:
References

- http://slideplayer.com/slide/6971785/
- http://slideplayer.com/slide/5821419/
- http://www.slideshare.net/ZeroTurnaround/vladimir-ivanovjvmjitcompilationoverview-24613146
- http://eli.thegreenplace.net/2013/11/05/how-to-jit-an-introduction
- https://www.youtube.com/watch?v=8yo1gQT7U74
- http://www.oracle.com/technetwork/java/whitepaper-135217.html#scalability
Outline

- Overview
  - Static Compilation
  - Virtual machines
  - Traditional Approaches
- JIT
  - Defining JIT
  - JIT: A Combination of two traditional approaches
- Working Mechanism
  - Conceptual Idea
  - Technical steps
  - Advantages & Drawbacks
- Applications
- Exploring JIT & java
  - Compilation in java
  - VM & JVM
  - JIT in JVM
  - Runtime optimizations by JIT
- Startup Delay and Possible Optimizations
  - Java HotSpot
Thanks for your precious time 😊
Any Questions ?