PROFESSOR HANSPETER MÖSSENBÖCK

Java **on time**

Professor Hanspeter Mössenböck is head of the Institute for System Software at the Johannes Kepler University, Linz. For more than 10 years, he and his team have been collaborating with Sun Microsystems, and then Oracle, on developing compilers for the programming language Java. Here, he tells us about just-in-time compilers and the innovative work of his team

Why is it necessary to investigate new compilation and optimisation techniques for Java just-in-time (JIT) compliers?

Compiler technology is a well-understood field. However, traditional compilers are batch programs that compile source code to machine code before the code is executed. With JIT compilation the situation has changed. Java programs, for example, start to execute in the interpreter before parts of them are translated to machine code. Compilation happens on the fly during program execution. Thus, many of the well-known compilation techniques cannot be used because they are simply too slow. New, faster techniques need to be developed that allow compilation to happen in the background without noticeable overhead.

What are the advantages to using JIT compilers over static compilers?

JIT compilation allows us to use profiling feedback from the executing program in order to optimise the program for its current input. This cannot be done in static compilation; for example, we can exploit knowledge about path frequencies for reordering the paths in an optimal way, or we can use information about the data types seen so far in order to optimise for the most frequently occurring types. Another advantage of JIT compilers is that they can concentrate on the 'hot spots' of a program, while static compilers always have to compile the whole program. With JIT compilation we can even recompile a method several times with increasing optimisation levels: in the first run we can do a rough and quick compilation, and when the method becomes 'hotter' we can recompile it, spending more time on sophisticated optimisations.

JIT compilation can also be used to change the structure of a program at run time. What is the use for such transformations?

Some programs have to run 24/7 and cannot be shut down for maintenance. It is therefore desirable for maintenance of such programs to be done at run time, which requires a recompilation of affected methods as well as a transformation of already existing data in memory. JIT compilation allows us to do such recompilations selectively. Replacing code and data types at run time is anything but trivial because one has to guarantee the consistency between old and new code parts.

What are you currently working on?

Since programs are compiled while they are executed there is hardly any time for optimisation. Some of our subprojects therefore aim at developing new optimisation techniques that are fast enough to be applied during JIT compilation and still have significant impact on the execution speed. One of our biggest current projects is the development of a new compiler infrastructure called Graal. Alongside this we have a promising project dealing with trace compilation, and we are taking an interesting research direction on our Truffle framework.

Can you reveal more about the Truffle framework?

The Truffle framework is a tree-based execution environment to target the efficient execution of dynamically typed languages such as Javascript, Python or R. The program to be executed is translated to an abstract syntax tree (AST), which is then interpreted. The AST can be modified at run time according to type feedback that is collected during execution. In Javascript, for example, the operand types of an add operation are statically unknown, so the interpreter has to treat them in a general and inefficient way. However, if run time feedback reveals that the operands are always integers, the subtree of the add operation can be replaced with a specialised one in which the operands are integers. The operation can then be performed much more efficiently. The AST in Truffle is therefore a self-optimising representation that adapts to the program being executed. It can also easily be adapted to new languages. Seamless multi-language interoperability is one of the goals of Truffle. Subtrees can originate from different languages, and optimisations on the trees can cross-cut language boundaries.

```
DOMTable.prototype.create = function(dati, thBin){
    xDOMTable = document.createElement("table");
    xtblBody = document.createElement("tbody");
    Element.extend(xDOMTable);
    xDOMTable.setAttribute("id", "domtable");
    Element.extend(xDOMTable);
    if (typeof(dati[0]) != 'object')
        return this.msgTable(typeof(dati) + " not in
[[\"A1\",\"A2\"],[\"B1\",\"B2\"]] format");
    var th = 0;
    var th0 = 0;
//  alert(dati[0][1]);
    for (r = 0; r < dati.length; r++) {
        var domtr = document.createElement("tr");
        Element.extend(domtr);
        th0 = ((r == 0 && (thBin & this.thTop)) || (r ==
(dati.length - 1) && (thBin & this.thBottom))) ? 1 : 0;
        for (c = 0; c < dati[r].length; c++) {
            if (c ==
            this.d = dati[r][c];
            continue;
            }
            if (!th0)
                th = ((c == 0 && (thBin & this.thLeft))
|| (c == (dati[r].length - 1) && (thBin &
this.thRight))) ? 1 : 0;
            var tag = (th || th0) ? "th" : "td";
            var cell = document.createElement(tag);
            Element.extend(cell);
            var tstyle = (tag == "th" ? this.thStyle :
this.tdStyle);
            var tdid =
dati[0][c]+'_'+this.xid;//c+'_'+r;
```

Timely developments

Java is used by over 10 million people worldwide. Nonetheless, its compilation process is more complex than for other programming languages, making Java more dependent on high-quality compilers. Now, researchers at **Johannes Kepler University Linz** are developing novel optimisation techniques for Java

JAVA HAS COME a long way since it first launched in 1995, and is widely believed to be faster than most other programming languages. Its popularity is indisputable, reporting 10 million users in 2012. However, while languages like C, C++ or Fortran are directly translated to machine code, Java programs are first translated to a compact bytecode representation. This representation is portable – it can be interpreted by a Java virtual machine (VM) on almost any computer – but is much slower than pure machine code.

To address this, VMs compile the most frequently executed parts of a program to machine code. This service is on demand and is known as a just-in-time (JIT) compilation. Since JIT compilation happens in the background while the program to be compiled is interpreted, it has to be as fast as possible, while also producing high-quality code. Here, Hanspeter Mössenböck, Professor of Computer Science and head of the Institute for System Software at the Johannes Kepler University Linz, sees an enormous research potential. Working alongside five PhD and four Master's students, Mössenböck has collaborated with Oracle, the major vendor of Java, for over 10 years. Together they work to optimise Java JIT compilers and investigate new compilation techniques.

OPTIMISATION

The team's work centres around discovering new intermediate program representations as well as program analyses and transformations that improve the performance of applications running on Oracle's Java Hotspot VM. These optimisations are as diverse as developing a fast register allocator, a safe check elimination technique, algorithms for eliminating redundant computations, and fast inlining of method calls, among others.

A special challenge of JIT compiler optimisations is that they take effect while the compiled program is being executed. For example, if the VM decides to merge two data types due to a compiler optimisation, all existing objects of these data types have to be merged in memory. This then has to happen transparently in the background while the program continues execution. Central to the team's work is that these optimisations must be as efficient as possible. Yet the optimisations must produce high-quality code. Balancing these often conflicting goals requires the team to regularly develop completely novel algorithms.

INTELLIGENCE

JUST-IN-TIME COMPILATION

OBJECTIVES

To devise novel techniques for just-in-time compilation and optimisation in order to improve the performance of applications running on Oracle's Java Hotspot virtual machine. Topics include: dynamic and transparent optimisations, dynamic code evolution, trace compilation, self-optimising interpreters and multi-language virtual machines.

KEY COLLABORATORS

Dr Thomas Würthinger, Oracle Labs

Professor Walter Binder, Universitá della Svizzera Italiana

FUNDING

Oracle Labs

FWF Austrian Science Fund (P 22493-N18)

CONTACT

Professor Hanspeter Mössenböck
Head, Institute for System Software

Johannes Kepler University Linz
Altenbergerstraße 69
A-4040 Linz
Austria

T +43 732 2468 4340
E hanspeter.moessenboeck@jku.at

www.fwf.ac.at/de/abstracts/abstract.asp?L=D&PROJ=P22493

HANSPETER MÖSSENBÖCK is a full professor of Computer Science and head of the Institute for System Software at the Johannes Kepler University Linz. Before he joined JKU, he was an assistant professor at ETH Zurich where he worked with Niklaus Wirth on the development of the Oberon language and system. He is mainly interested in programming languages, compilers, system software and software architectures, and is the author of several books on software development. For more than 10 years he and his team have been collaborating with Oracle (previously Sun Microsystems) on just-in-time compilers and virtual machines. His current project team consists of five PhD and four Master's students.

Mössenböck describes the innovative aspect of their work as lying in the aggressiveness of their optimisations: "We often generate optimised machine code based on optimistic assumptions and later fall back to interpreted code if those assumptions turn out to be wrong". The group is also faced with the challenge that optimisations have to modify the code and the data of a running program without breaking it – something they would not have to face with static compilers.

SUCCESSFUL TRACK RECORD

Over the years the team has developed a number of optimisations that have been used in the product version of Oracle's HotSpot Java. These include a linear scan register allocator faster than traditional register allocators, which were traditionally based on a graph-colouring approach. Other optimisations include: escape analysis, in which data objects that do not escape their allocation context can be transformed into simple variables; object inlining, in which closely related data objects are automatically merged in memory in order to speed up memory management and data access; and bounds check elimination, where accesses to arrays can be optimised to avoid checks if the array index is in the valid range.

The team has developed a number of optimisations that have been used in the product version of Oracle's HotSpot Java

CURRENT PROJECTS

Among the latest developments is a new compiler infrastructure called Graal, an open source project for the OpenJDK community. The research aims to develop a new Java JIT compiler that allows for easier and more aggressive optimisations. As such, it is based on a novel intermediate program representation and uses optimistic assumptions extensively. Written in Java, the development of optimisations can benefit from the use of a high-level language.

Alongside Graal, the researchers are also developing innovative trace compilation, a special form of JIT compilation. Most JIT compilers translate frequently executed methods; however, a trace compiler translates frequently executed paths through a program. This enables the compilation unit to be smaller and more precise, and therefore the compilation to be faster. Paths also carry more context information than methods and therefore allow more aggressive optimisations. "One of the most rewarding compiler optimisations is code inlining: instead of invoking a method, its code is directly embedded at the call site," Mössenböck explains. Trace compilation allows more aggressive inlining than method-based compilation, because traces are smaller and more accurate than methods in the sense that they carry more context information.

FUTURE PLANS

In order to continue their innovative work, the group hopes to profit from more profiling feedback and more context information. For example, if profiling would tell the compiler that a certain method is always called from the same program site and with the same range of parameter values, the compiler could exploit this knowledge for more sophisticated optimisations.

An exciting prospect is the planned development of JIT compilation of dynamically typed languages such as Javascript or Ruby. Compilers have traditionally been unable to produce highly efficient code as these languages lack static type information. However, the team hopes to develop runtime feedback, which could collect type information during the execution of such programs. When enough information was available, parts of the programs could then be JIT-compiled to code which is as efficient as that produced from a statically typed language.

Looking towards the longer term impact of his group's work, Mössenböck strikes a particularly positive note: "I consider research to be particularly exciting if it has potential to be used in practice. As we are collaborating with Oracle, some of our research results have a fair chance of being incorporated into the Java product and may therefore be used by millions of developers and consumers across the world." It is this translational approach that enables the Institute for System Software at the Johannes Kepler University Linz to attract excellent PhD students from all over the world, and to lead the way in their collaboration with Oracle.

```
            if (r==0)
                tdid = dati[0][c]+'_0';

            cell.setAttribute("id", tdid);
            var cellText =
document.createTextNode(dati[r][c]);
            if (tag == "th") {
                var oThis = this;
                this.colonne.push(tdid);
                this.colvals.push(dati[r][c]);

                Event.observe(cell,"click",function(){
                    asc = ordarr[1] == 'ASC' ? 'DESC' :
'ASC';

                    buf = this.innerHTML.split('<');
                    ord = buf[0];
```