Echtzeitfähige Code-Generierung in virtualisierenden Ausführungsumgebungen

Martin Däumler und Matthias Werner

TU Chemnitz
Fakultät für Informatik
D-09107 Chemnitz
mdae|mwerner@cs.tu-chemnitz.de

Virtualisierende Ausführungsumgebungen wie Implementierungen der "Java Virtual Maschine Specification" (JVM Specification) oder der "Common Language Infrastructure" (CLI), die Nutzer-Applikationen ausführen, eröffnen neue Herausforderungen bei der Entwicklung von Software für Echtzeitsysteme. Eine virtualisierende Ausführungsumgebung wird häufig als virtuelle Maschine (VM) realisiert. Eine solche VM bietet einen plattformunabhängigen virtuellen Befehlssatz , der von dem realen Rechensystem abstrahiert. Gäste der VM, d.h., Anwendungen, werden in Zwischencode ausgeliefert, der aus Instruktionen des virtuellen Befehlssatzes besteht. Das erhöht die Portabilität, da nur die VM auf das reale Rechensystem portiert werden muss.

Zur Ausführung muss der Zwischencode in nativ ausführbaren Code des realen Rechensystems überführt werden. Gängige VM-Implementierungen nutzen dynamische Compilierung mit einem "Just-in-Time"-Compiler (JIT-Compiler). Die Übersetzung eines Stücks Zwischencode wird typischerweise bis zur ersten Ausführung dieses Stück Codes verzögert. Das wird als "*lazy compilation*" bezeichnet und stellt eine Optimierung des Speicherbedarfs und der Zeit dar, da Code bedarfsgerecht erzeugt wird. Dadurch können die Ausführungszeiten eines Stück Zwischencodes bei mehrmaliger Ausführung erheblich schwanken. Das steht im Widerspruch zu deterministischen Ausführungszeiten, wie sie in einem Echtzeitsystem notwendig sind. Dieser Vortrag stellt ein Konzept zur Erzielung deterministischer Ausführungszeiten in VMs vor. Das Ziel ist die Ausführung einer Anwendung, ohne das zur Ausführungszeit nativer Code erzeugt oder modifiziert wird. Damit lässt sich ein Determinismus erreichen, der z.B. für eine in C programmierte Anwendung typisch ist. Weitere Quellen zeitlichen Nichtdeterminismus wie automatische Speicherverwaltung oder Synchronisierungs- und Priorisierungsaspekte werden hier, ebenso wie WCET-Analyse, nicht betrachtet. Neben dem hohen zeitlichen Determinismus gibt es weitere Anforderungen, um Anwendungsprogrammierer nicht unnötig einzuschränken. So soll der Zwischencode bzw. der Quell-Code, der in den Zwischencode übersetzt wird, nicht modifiziert werden müssen, um von dem gesteigerten Determinismus profitieren zu können und um Legacy-Code erfassen zu können. Daher muss die Lösung dynamische Sprachfeatures von VMs unterstützen, auch wenn für diese keine Echtzeitgarantien gegeben werden können. Um die zugrundeliegende Hardware effizient zu nutzen bzw. um eine hohe Performanz zu erreichen, wird auf In-

terpretation des Zwischencodes verzichtet. Zudem soll sich die Startzeit einer Anwendung nicht unnötig erhöhen.

Bei der Lösung des Problems wurde sich für die Adaption einer ausgereiften VM – der freien CLI-Implementierung Mono [Xam12] als Versuchsträger – entschieden. So kann auf einen großen Funktionsumfang und bewährte Komponenten wie den Klassenlader und Code-Generator zurückgegriffen und die Klassenbibliothek wiederverwendet werden. In einem ersten Schritt wird durch eine Vor-Compilierung in der Startphase der VM nativer Code generiert. Es werden alle CLI-Assemblies verarbeitet, die, ausgehend vom Assembly der Anwendung, referenziert werden [SDW11]. Diese Methode wurde einer Code-Coverage-Analyse vorgezogen, da diese ebenfalls zur Startzeit beitragen würde. Der erzeugte native Code enthält unbestimmte Referenzen zu noch nicht oder später vor-compilierten Code. Diese verzweigen in die VM um bei Bedarf Code zu laden bzw. zu generieren. In einem zweiten Schritt werden die unbestimmten Referenzen aufgelöst [DW12]. Im Gegensatz zur Vor-Compilierung erfolgt das auf einer hardwarenahen Ebene. Der so optimierte native Code enthält für nicht-dynamische Teile keine unbestimmten Referenzen, so dass er ohne Eingreifen der VM ausgeführt werden kann.

Für die Code-Generierung wurde Monos JIT-Compiler verwendet. Alternativ bietet Mono einen Ahead-of-Time-Compiler (AOT-Compiler), der Zwischencode in positionsunabhängigen nativen Code übersetzt und persistent speichert. Zur Laufzeit wird dieser native geladen und ausgeführt. Das "*lazy loading*" ist eine Quelle zeitlichen Nicht-Determinismus. Zudem enthält positionsunabhängiger Code tendenziell mehr (unbestimmte) Referenzen als JIT-compilierter Code, wodurch sich Performanz-Nachteile ergeben können. Eine weitere Untersuchung mit dem AOT-Compiler als Code-Generator und angepasster Referenz-Auflösung [DW13] ergab eine Reduktion der Startzeit von ca. 20% gegenüber den JIT-basierten Ansatz.

Die Diskussion des Ansatzes wird mit Ergebnissen mehrerer Experimente unterstützt, die in erster Linie die Vorhersagbarkeit der Ausführungszeiten und die Startzeit einer Anwendung betrachten. Um einen Vergleich existierender verschiedener Konzepte zur echtzeitfähigen Code- Generierung zu erhalten, wurden für die Experimente unter anderem etablierte Echtzeit-Lösungen aus dem Java-Bereich herangezogen.

Literaturverzeichnis

[DW12] Däumler, Martin ; Werner, Matthias: Optimierung der Code-Generierung virtualisierender Ausführungsumgebungen zur Erzielung deterministischer Ausführungszeiten. In: Halang, WolfgangA. (Hrsg.) ; Holleczek, Peter (Hrsg.): *Kommunikation unter Echtzeitbedingungen* Bd. 1. Springer Berlin Heidelberg, 2012, S. 29–38

[DW13] Däumler, Martin ; Werner, Matthias: Reducing startup time of a deterministic virtualizing runtime environment. In: *Proceedings of the 16th Inter-*

national Workshop on Software and Compilers for Embedded Systems. New York, NY, USA : ACM, 2013 (M-SCOPES '13). – ISBN 978–1–4503–2142–6, 48–57

[SDW11] Schepeljanski, Alexej ; Däumler, Martin ; Werner, Matthias: Entwicklung einer echtzeitfähigen CLI-Laufzeitumgebung für den Einsatz in der Automatisierungstechnik. In: Halang, WolfgangA. (Hrsg.) ; Holleczek, Peter (Hrsg.): *Eingebettete Systeme* Bd. 1. Springer Berlin Heidelberg, 2011, S. 21–30

[Xam12] Xamarin: *Mono*. www.mono-project.com, 2012