
(Bild: zlikovec – Shutterstock)

Automatische Programmierung von eingebetteten Mehrkernprozessoren:

Grafisch parallel programmieren

Die Leistung eingebetteter Systeme wird durch den Einsatz von Mehrkernprozessoren und Hardware-Beschleunigern drastisch gesteigert; gleichzeitig steigt aber auch die Komplexität der Programmierung. Ein neuer Matlab/Scilab-basierter Workflow kann die Arbeit deutlich vereinfachen und beschleunigen.

Von Oliver Oey und Dr.-Ing. Timo Stripf

Eingebettete Systeme sind kleine Computer, welche in einem Gerät integriert sind und auf eine spezielle Aufgabe (z.B. Überwachung, Steuerung oder Regelung) zugeschnitten sind. Die Integration erfordert kleine Abmessungen, was zu Einschränkungen bei den Ressourcen wie Speicher und Rechenleistung führt. Ihr Einsatz stellt in der Regel zudem Anforderungen an die Leistungsaufnahme und die Kosten. Durch immer neue Anwendungsmöglichkeiten wie intelligente Kameras oder autonom fahrende Autos steigt der Bedarf an Rechenleistung dramatisch an. Die aktuellen Einzelkernprozessoren stoßen deshalb an ihre physikalischen Grenzen. Die Lösung liegt darin, die Anzahl der Recheneinheiten (Kerne) und durch speziell an die Aufgabe angepasste Beschleuniger zu ergänzen. Dadurch steigen wiederum die Anforderungen an die Software-Entwicklung: Zur Programmierung der Anwendung(en) kommt nun auch noch die Parallelisierung hinzu. Das wiederum steigert die Komplexität von Tests und Fehlersuche deutlich. Mitarbeiter der emmtrix Technologies GmbH arbeiten an einer automatischen Software-Parallelisierung, um den Herausforderungen in Zukunft Rechnung zu tragen.

Herausforderungen der parallelen Programmierung

Die parallele Programmierung bzw. Parallelisierung hat zum Ziel, eine Anwendung so auf mehrere Recheneinheiten zu verteilen, dass eine effizientere Ausführung erreicht wird. Dies bedeutet häufig, dass die Anwendung schneller ausgeführt wird, kann aber auch verwendet werden, um die Leistungsaufnahme zu senken: Die Leistungsaufnahme eines Prozessors kann durch die Formel $P \sim f \cdot U^2$ dargestellt werden. Dabei ist f die Taktfrequenz und U die Spannung. Bei heutigen Fertigungstechnologien (<45 nm) gilt zudem, dass die Spannung proportional zur Taktfrequenz ist. Die Leistungsaufnahme ist demnach proportional zu f^3. Vergleicht man nun beispielsweise einen Einzelkernprozessor mit einem Mehrkernprozessor mit vier Kernen und der halben Taktfrequenz, so hat dieser die doppelte theoretische Rechenleistung bei nach $P' \approx (1/2)^3 \cdot 4 \cdot P = 1/2\ P$ einer in etwa halbierten Leistungsaufnahme. Um die erhöhte Performance oder die verringerte Leistungsaufnahme zu erreichen, ist es jedoch notwendig, die Anwendung sinnvoll zu parallelisieren.

Bei der Entwicklung von paralleler Software ergeben sich viele Probleme,

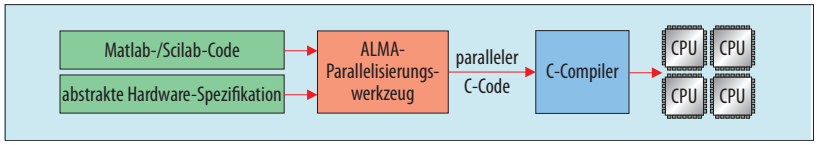

Bild 1. Schematische Darstellung der ALMA-Werkzeugkette.

die bei der sequenziellen Programmierung nicht existieren. Dies ist der Grund, warum die Software-Entwicklung für Mehrkernprozessoren um ein Vielfaches aufwendiger und fehleranfälliger ist im Vergleich zu Einzelkernprozessoren. Im Folgenden wird eine Auswahl der wichtigsten Probleme gegeben:

- **Race Conditions:** Bei Einzelkernprozessoren hat ein Programm eine klare Abarbeitungsreihenfolge. Bei Mehrkernprozessoren kann das Ergebnis einer Berechnung vom zeitlichen Verhalten von einem Kern abhängen. Wenn zwei Kerne die gleiche Variable schreiben, hängt der Wert der Variable davon ab, welcher Kern sie früher schreibt. Man spricht in solchen Situationen von einem kritischen Wettlauf (engl. Race Condition). Das führt in der Regel zu schwer auffindbaren Programmierfehlern, da nur kleine Änderungen beeinflussen können, ob ein Fehler auftritt. Erschwerend kommt hinzu, dass gängige Methoden zur Fehlersuche wie Debugging oder Logging das zeitliche Verhalten des Programms beeinflussen und dafür sorgen können, dass der Fehler gar nicht in Erscheinung tritt. Eine Lösung ist, den Zugriff auf die Variable durch zusätzliche Synchronisation nur in der gewollten Reihenfolge zu erlauben.
- **Deadlocks:** Bei der Synchronisation oder dem Datenaustausch zwischen verschiedenen Kernen muss dafür gesorgt werden, dass ein Kern wartet, bis er die passenden Daten oder Befehle erhält, um mit seiner Arbeit fortzufahren. Bei fehlerhafter Programmierung kann es nun passieren, dass es eine Menge an Kernen gibt, die auf Daten von Kernen warten, die ebenfalls in dieser Menge enthalten sind. Tritt dies auf, kann das Programm nicht mehr weiter abgearbeitet werden und man spricht von einem Deadlock oder einer Verklemmung.
- **Performance:** Das Ziel der parallelen Programmierung ist in der Regel, die Abarbeitungsgeschwindigkeit mit mehreren Kernen im Vergleich zu einem Kern zu erhöhen. Die Performance auf Multicore-Systemen wird durch verschiedene Faktoren behindert: Zum einen existieren nicht parallelisierbare Programmteile, die nur auf einem Kern ausgeführt werden können. Zum anderen führt die notwendige Synchronisation und Kommunikation zwischen Kernen zu Verzögerungen und Overhead. In der Praxis führt das dazu, dass die Performance von einer parallelen Anwendung nur sehr schwer vorhersagbar ist und es verschiedene Anläufe braucht, bis man eine gute Performance erzielt.
- **Auffinden von Fehlern:** Wie schon bei den Race Conditions erwähnt, kann das Auffinden von Fehlern in parallelen Anwendungen sehr viel komplexer werden als im sequenziellen Fall. Wird das Programm zu einer festen Bedingung angehalten, zum Beispiel mit einem Breakpoint im Quellcode, so kann nicht sichergestellt werden, dass immer alle Kerne an exakt der gleichen Instruktion angehalten werden. Somit ist das genaue Verhalten des Mehrkernprozessors nicht genau reproduzierbar und es erfordert in der Regel mehr Durchgänge und somit Zeit, um die Fehler zu finden.
- **Portabilität:** Das parallelisierte Programm ist allein durch die Aufteilung auf mehre Kerne bereits sehr stark an die Zielarchitektur angepasst und kann nicht mehr effizient auf einer Architektur mit einer anderen Anzahl an Kernen ausgeführt werden. Zusammen mit anderen Optimierungen bezüglich der Partitionierung von Daten und eingefügter Synchronisationsbefehle kann der Code nur noch mit größeren Änderungen auf eine andere Architektur portiert werden.

Gemäß einer Studie [1] sorgen die genannten Probleme dafür, dass die Software-Entwicklung für Mehrkernprozessoren dreimal so viele Software-Entwickler benötigt, mindestens 25 % mehr Zeit benötigt und am Ende etwa 4,5-mal so teuer ist.

EU-Projekt ALMA

Das EU-Projekt ALMA (www.alma-project.eu) hatte zum Ziel, die Parallelisierung durch eine automatisierte Werkzeugkette deutlich zu vereinfachen. Dazu sollte die Komplexität der Hardware vor dem Anwender verborgen werden, der sich auf die abstrakte Programmierung seiner Anwendung konzentrieren konnte. Zum Einsatz kam dabei die Sprache Scilab, welche eine

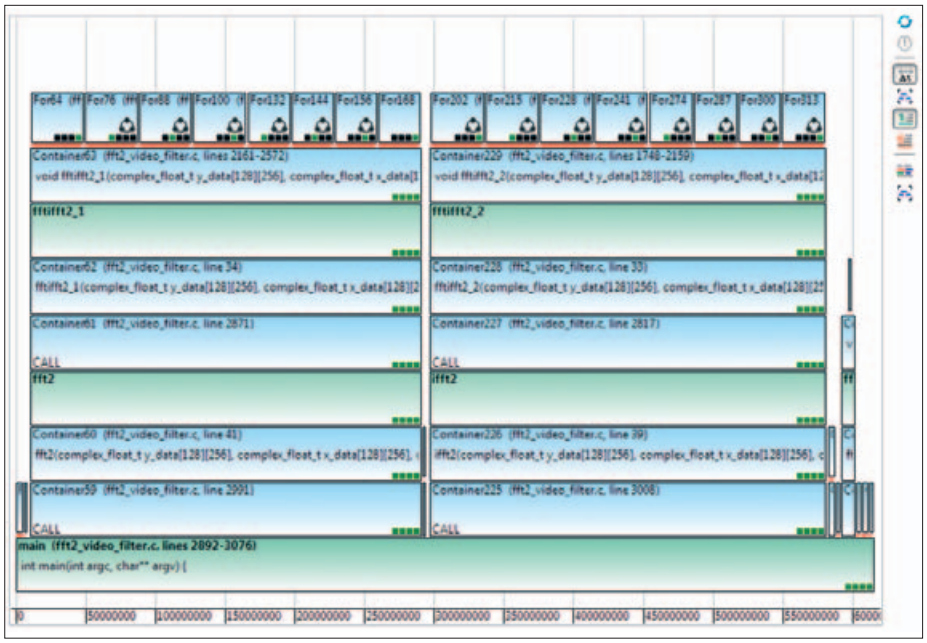

Bild 2. Hierarchische Darstellung eines Programms.

quelloffene Software für numerische Berechnungen ähnlich wie Matlab ist. Um die Hardware zu beschreiben, wurde eine Architekturbeschreibungssprache verwendet. Somit können alle wichtigen Eigenschaften der Hardware in einem standardisierten Format bereitgestellt werden.

Das Projekt lief von 2011 bis 2015 und konnte mit einem Proof of Principle in Form einer rudimentären Werkzeugkette (**Bild 1**), die erfolgreich mit zwei Testanwendungen evaluiert wurde, und den folgenden wichtigen Erkenntnissen beendet werden:

- ➜ Die automatische Parallelisierung kann die Entwicklung von parallelen Anwendungen sehr stark beschleunigen. Eine vollständige Automatisierung hat allerdings zur Folge, dass der Software-Entwickler außen vor gelassen wird und die Aufteilung auf die Kerne nur schwer nachvollziehbar ist. Im Evaluierungsprozess zeigte sich, dass Software-Entwickler die Parallelisierung mitbestimmen wollen, um mehr Einfluss darauf nehmen zu können, wie sich das Programm auf der Hardware verhält.
- ➜ Die Verwendung von numerischen Programmiersprachen eignet sich gut dafür, Algorithmen zu testen und zu entwickeln, ohne dass die Zielarchitektur dabei beachtet werden muss. Sie sind auch flexibel genug, um in verschiedenen Branchen für viele Anwendungen verwendet zu werden, wie durch unterschiedliche Testfälle in der Bildverarbeitung und der Telekommunikation gezeigt wurde.
- ➜ Zudem konnte durch die Wahl der Eingangssprache sichergestellt werden, dass sich der Code sehr gut für die Parallelisierung eignet. Als Beispiel seien hier Pointer in C/C++ genannt, die eine komplexere Analyse erfordern, um die reinen Änderungen an Daten zu erkennen. Pointer existieren in Scilab und Matlab nicht.

Bild 3. Scheduling-Ansicht eines Programms mit Darstellung der Prozessorlast. Blöcke, die die meiste Zeit benötigen, sind sofort sichtbar.

Weiterentwicklung für den Industrieeinsatz

Im Rahmen des EU-Projekts ALMA hat emmtrix Technologies daran mitgearbeitet, den wissenschaftlichen Proof of Principle zu erbringen, und es sich jetzt zur Aufgabe gemacht, den Ansatz noch weiter an die industriellen Bedürfnisse anzupassen. Da in ALMA zunächst die Evaluation des Konzepts im Vordergrund stand, reichte es aus, die Werkzeugkette über Skripte steuern zu können. Anwender bekamen Informationen über die Güte der Parallelisierung erst nach der Ausführung in einem Simulator oder direkt auf der Hardware. Der erste Schritt hin zu einer intuitiveren Bedienung war es, eine grafische Aufbereitung des Programms und schnelle Rückmeldungen über getroffene Entscheidungen bereitzustellen.

Die grafische Darstellung über Hierarchieebenen ist in **Bild 2** dargestellt. Auf der X-Achse ist die Zeit in Form von Taktzyklen der Zielarchitektur aufgetragen. Auf der Y-Achse ist die Hierarchie des Beispielprogramms dargestellt. Eine neue Hierarchieebene wird hinzugefügt, wenn ein neuer Block oder Funktionsaufruf im Programm gefunden wird. Das gewählte Beispiel besteht zum größten Teil aus zwei Funktionsaufrufen für die zweidimensionale Fourier-Transformation und ihre Inverse (fft2 und ifft2). Sie werden verwendet, um Daten aus dem Zeitbereich in den Frequenzbereich um-

zuwandeln und umgekehrt. Diese Transformationen werden in vielen Bereichen eingesetzt, wenn mit größeren Datenmengen gearbeitet wird, da manche Berechnungen im Frequenzbereich deutlich einfacher zu erledigen sind. Die Transformation und ihre Inverse haben einen sehr ähnlichen Aufbau. Folgt man den Hierarchien, trifft man auf weitere Containerblöcke und Funktionsaufrufe, bis man auf der obersten Ebene einige For-Schleifen findet. Mit dieser Darstellung ist es sehr einfach möglich, sich einen Überblick über ein Programm zu verschaffen und die Codeteile zu identifizieren, deren Berechnung am meisten Zeit benötigen. Zusätzlich werden in der unteren rechten Ecke eines jeden Blocks die aktuell zugewiesenen Prozessoren dargestellt. Im gewählten Beispiel ist es ein Vierkernprozessor, bei dem die For-Schleifen auf der obersten Ebene auf die verfügbaren Kerne verteilt wurden.

Bild 4. An die Industrie angepasster Workflow.

Die hierarchische Darstellung hilft bei der Analyse des sequenziellen Programms und erlaubt es, Parallelisierungsentscheidungen wie die Zuweisung von Blöcken auf Kerne zu treffen. Um die parallele Ausführung des Programms zu evaluieren, wird zusätzlich eine Darstellung der Prozessorlast über der Zeit in einer Scheduling-Ansicht wie in **Bild 3** dargestellt. Für jeden Kern des gewählten Cortex-A7-Prozessors sind die relevanten Blöcke, die die meiste Zeit beanspruchen, dargestellt. Nicht jeder Teil einer Anwendung kann parallelisiert werden, wie man in diesem Beispiel am Anfang des Programms sieht. Die Initialisierung findet auf Kern 0 statt; alle anderen Prozessoren müssen warten, bis sie mit ihren Berechnungen beginnen können.

Die grafische Benutzeroberfläche (**Bild 4**) unterstützt den Anwender bei seinen Entscheidungen; viele andere Tätigkeiten können durch Automatisierung komplett übernommen werden. Durch automatisches Einfügen der Synchronisation und die Verteilung von Daten kann sichergestellt werden, dass keine Race Conditions oder Deadlocks auftreten. Dies wird durch eine Festlegung von klaren Regeln erreicht, die dafür sorgen, dass die Kommunikation immer in der bereits vorher berechneten Reihenfolge abläuft. Für eine Abschätzung dieser Zeiten wird ein internes Modell der Zielarchitektur verwendet. Dieses beinhaltet sowohl die Zeiten für sämtliche Befehle als auch für die Kommunikation und Synchronisation. Die Methoden zur Ermittlung dieser Werte können von einer rein statischen Analyse des Quellcodes über die Ausführung auf einem Simulator bis hin zur Ausführung auf dem tatsächlichen Zielsystem erfolgen.

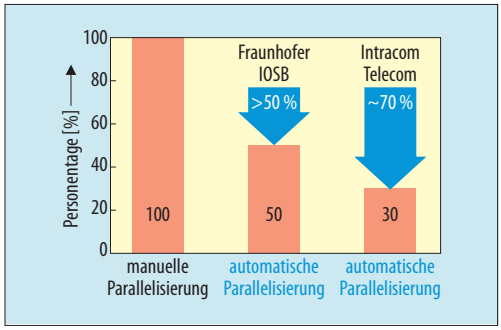

Bild 5. Zeitersparnis durch Einsatz von automatischer Parallelisierung.

Halbe Entwicklungszeit

Die entwickelte Werkzeugkette wurde mit zwei industriellen Anwendungen evaluiert. Im Fokus stand dabei die erreichte Einsparung an Entwicklungszeit im Vergleich zu einer rein manuellen Parallelisierung. Für die Tests kam die X2014-Plattform der Firma Recore Systems zum Einsatz.

Die erste Anwendung stammt vom Fraunhofer IOSB und kommt aus dem Bereich der industriellen Bildverarbeitung. Die skaleninvariante Merkmalstransformation (engl. Scale-invariant Feature Transform, kurz SIFT) wird verwendet, um Objekte auf einem Video wiedererkennen zu können, und kann für verschiedene Aufgaben wie beispielsweise Überwachung oder Schildererkennung verwendet werden. Mit Hilfe der automatischen Parallelisierung konnte eine Beschleunigung um den Faktor 1,86 beim Einsatz von zwei Prozessorkernen erreicht werden. Beim Wechsel auf vier Kerne konnte eine Beschleunigung von 2,6 erzielt werden. Dabei konnte die Entwicklungszeit um rund 50 % im Vergleich zur manuellen Parallelisierung reduziert werden.

Die zweite Anwendung stammt aus dem Bereich der Telekommunikation von der Firma Intracom Telecom. Betrachtet wurde die vollständige Basisbandkette der Sendeseite nach dem IEEE-802.16e-Standard. Diese Datenvorverarbeitung wird eingesetzt, um Daten nach dem Wimax-Standard zu versenden. Bei der Verwendung von zwei beziehungsweise vier Kernen wurde eine Beschleunigung um den Faktor 1,7 respektive 2,77 erreicht (**Bild 5**). Dabei konnte die Entwicklungszeit um etwa 70 % reduziert werden.

jk

Literatur

[1] **VDC Research: Next Generation Embedded Hardware Architectures Driving Onset of Project Delays, Costs Overruns, and Software Development Changes. September 2010.**

Oliver Oey
ist Leiter der Produktentwicklung bei der emmtrix Technologies GmbH. Zuvor arbeitete er in verschiedenen Forschungsprojekten im Bereich paralleler Software-Entwicklung am Fraunhofer IOSB und dem Karlsruher Institut für Technologie (KIT). Herr Oey studierte Elektro- und Informationstechnik am KIT.
oliver.oey@emmtrix.com

Dr.-Ing. Timo Stripf
ist technischer Geschäftsführer der emmtrix Technologies GmbH. Herr Stripf war für die Leitung des EU-Forschungsprojekts ALMA mitverantwortlich. Er studierte am Karlsruher Institut für Technologie (KIT) Informatik und promovierte in Elektrotechnik im Bereich Compilerbau.
timo.stripf@emmtrix.com