Plattformübergreifende Software für Multicore & FPGAs

Portable MATLAB®, Simulink und Scilab-Softwareentwicklung

Timo Stripf, Michael Rückauer und Oliver Oey, emmtrix Technologies GmbH

Neuste eingebettete Systeme werden zunehmend mit leistungsfähigen Multicore-Prozessoren sowie FPGA-Beschleunigern ausgestattet. Das ist notwendig, um die immer höheren Performanzanforderungen von Anwendungen bei möglichst niedrigem Energieverbrauch zu erfüllen. Die Programmierung solcher heterogenen Multicore-Systeme in Unternehmen wird derzeit überwiegend manuell realisiert. Dies ist sehr komplex, teuer und zeitaufwändig und stellt Software-Entwickler vor eine große Herausforderung. Viele Unternehmen scheuen aus diesem Grund den Einsatz von Multicore sogar gänzlich.
Im Markt für eingebettete Systeme wird händeringend nach Programmierlösungen für Multicore-Systeme gesucht. In diesem Vortrag werden Werkzeuge für die automatisierte C-Code Generierung und Parallelisierung aus MATLAB®, Simulink und Scilab für heterogene eingebettete Multicore-Systeme vorgestellt.

Motivation

Laut Studien [1] ist die Programmierung von eingebetteten Mehrkernsystemen 4,5-mal so teuer und benötigt 3-mal so viele Software-Ingenieure wie die Programmierung von Einkernsystemen und dauert 25% länger. Das Problem hierbei ist insbesondere der Mehraufwand durch die Verteilung der Aufgaben auf die einzelnen Kerne, der zusätzliche Testaufwand, sowie Probleme durch zusätzliche Fehlerquellen wie Race Conditions oder Deadlocks. Insbesondere letzteres sind Fehler, die besonders schwierig zu Debuggen sind und daher von Entwicklern gefürchtet werden. Bei heterogenen Systemen (im Vergleich zu homogenen Systemen) kommt zusätzlich noch der Mehraufwand hinzu, entscheiden zu müssen, welcher Anwendungsteil auf welchem Prozessortyp oder Beschleuniger ausgeführt werden soll, um ein effizientes Ergebnis bezüglich Performanz & Energieverbrauch zu erzielen.

Die emmtrix Technologies GmbH hat ein Entwicklungswerkzeug entwickelt, welches die Probleme bei der parallelen Softwareentwicklung für eingebettete heterogene Mehrkernprozessoren adressiert. Mittels automatischer Parallelisierung und Codegenerierung werden die aufwändigen Aufgaben automatisiert, sowie die Parallelisierung innerhalb einer graphischen Benutzeroberfläche erheblich vereinfacht. Als Eingabe werden MATLAB®-Skripte, Simulink-Modelle sowie die Open-Source-Alternative Scilab unterstützt. Der Programmierer kann sich auf die Entwicklung des Algorithmus konzentrieren, während die Anpassungen an die Zielhardware automatisch durch die Werkzeugkette erledigt werden. Schließlich kann die Lösung direkt auf der Zielhardware ausgeführt und getestet werden. Diese Automatisierung der Parallelisierung ermöglicht Rapid Prototyping, was den Aufwand des Entwicklungszyklus zur Evaluation einer parallelen Lösung erheblich reduziert.

Beispielanwendung & Testumgebung
Als Beispielanwendung dient ein überschaubares Teilproblem aus der industriellen Bildverarbeitung, das als MATLAB®-Programm implementiert ist. Es wird ein Bild von einer Kamera eingelesen, eine Kantendetektion mittels eines Sobel-Filters durchgeführt, sowie das Ergebnis auf einem Bildschirm ausgegeben.

Als Zielsystem kommt ein ZedBoard [2] mit einem Zynq-7000 FPGA (XC7Z020) zum Einsatz, der einen Dual-Core Cortex A9 Prozessor mit FPGA-Logik kombiniert. Die programmierbare FPGA-Logik dient sowohl zur Bildakquise von der angeschlossenen Kamera, zur Ausgabe des Bildes auf dem Monitor, als auch als Hardwarebeschleuniger für Teile der Beispielanwendung. Auf dem System läuft ein Linux-Betriebssystem.

Im Folgenden ist der Grundaufbau der MATLAB®-Anwendung gezeigt:

```
function edgeImage = sobel(originalImage, threshold)
  k = [ 1  2  1;
        0  0  0;
       -1 -2 -1];
  H = conv2(double(originalImage),k,'same');
  V = conv2(double(originalImage),k','same');
  E = sqrt(H.*H + V.*V);
  edgeImage = uint8((E > threshold) * 255);
end
```

Der Grundaufbau des Filters ist, dass zunächst zwei 2D-Faltungsfilter (conv2) für die Erkennung der horizontalen und vertikalen Kanten angewandt werden. Als Faltungsmatrix wird k bzw. die Transponierte k‘ verwendet. Richtungsunabhängige Kanteninformationen werden durch die Kombination beider Ergebnisse erzielt. Als Ausgabebild werden Kanten angezeigt, sobald ein Schwellwert überschritten ist. In unserem Beispiel wird 0,7 angenommen.

Abbildung 1: Beispiel des Sobel-Filters

Generierung sequentiellen C-Codes
Als erster Arbeitsschritt wird das MATLAB®-Programm zunächst in sequentiellen C-Quellcode für die Zielarchitektur übersetzt. Hierbei kommt der emmtrix Code Generator zum Einsatz. Dieser unterstützt sowohl MATLAB®- und Scilab-Skripte als auch Simulink-Modelle. Der generierte C-Code ist bereits für die Parallelisierung vorbereitet.

Ein wichtiger Aspekt ist hierbei die Modellierung des Interface-Codes.

```
function main()
  image = interface_indata();
  gray = rgb2gray(image);
  edgeIm = sobel(gray, 0.7);
  interface_outdata(gray2rgba(edgeIm));
end
```

Der Interface-Code nimmt die Eingangsdaten entgegen und gibt die Ausgangsdaten aus. Dieser ist unterschiedlich, je nachdem wo und in welcher Umgebung der Code ausgeführt wird. Zur Evaluation des Programmes auf dem PC kann z.B. interface_indata innerhalb der MATLAB®-Anwendung auf eine Bilddatei zugreifen und interface_outdata schreibt das Ergebnis auf die Festplatte. Auf diese Art wurde z.B. auch das Bild in Abbildung 1 erzeugt. Für eine spätere Ausführung auf der Hardware wird der Interface-Code dann durch eine Ansteuerung der Kamera bzw. des Bildschirms ersetzt.

Analyse des Programms

Nach der Generierung des sequentiellen Programms wird dieses auf dem PC mit Testeingangsdaten ausgeführt. Hierbei wird ein Profiling durchgeführt, mit dem die Ausführungszeiten für die eingestellte Zielarchitektur ermittelt werden. Als Ergebnis wird dann im emmtrix Parallel Studio [3] eine hierarchische Programmdarstellung angezeigt, die die Ausführungszeiten der verschiedenen Programmteile zeigt. Auf der horizontalen Achse ist die Zeit gegeben. Auf der untersten Ebene ist die main-Funktion, die das gesamte Programme repräsentiert und sich über die ganze Zeit erstreckt. Auf der zweiten Ebene befindet sich am Anfang und Ende die Funktionsaufrufe für die Interface-Funktionen. In der Mitte ist der größte Block der Aufruf der sobel-Funktion. Innerhalb der Funktion kann man gut die beiden 2D-Falterungsfilter (conv2_1 und conv2_2) sowie die For-Schleife zur Kombination der beiden Faltungsergebnisse erkennen. Tiefere Ebenen werden in diesem Beispiel aus Übersichtszwecken nicht mehr dargestellt.

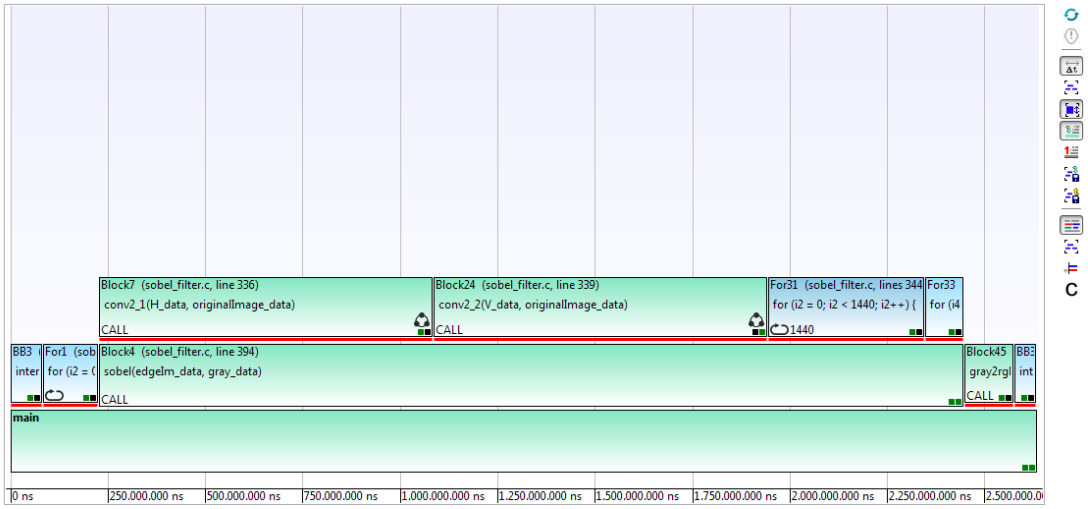

Abbildung 2: Hierarchische Darstellung des Sobel-Filters

Parallelisierung des Programms
Zunächst wird das Programm für den Dual-Core-Prozessor des Zynq-Systems parallelisiert. Hier hat der automatische Scheduler bereits eine sinnvolle Verteilung gefunden. Ansonsten kann der Benutzer in der hierarchischen Darstellung immer in den Prozess eingreifen und z.B. den Prozessorkern pro Programmteil/-block festlegen. Damit behält der Benutzer stets die volle Kontrolle über die Parallelisierung.

Ein Ergebnis der Parallelisierung besteht darin, welche Programmteile gemeinsam auf einem Prozessor abgearbeitet werden bzw. wie tief ein Programm aufgebrochen werden soll. Dies ist in der oberen Darstellung anhand der roten Linie erkennbar. Anhand der Kästchen rechts unten ist die Zuweisung auf den jeweiligen Prozessorkern erkennbar.

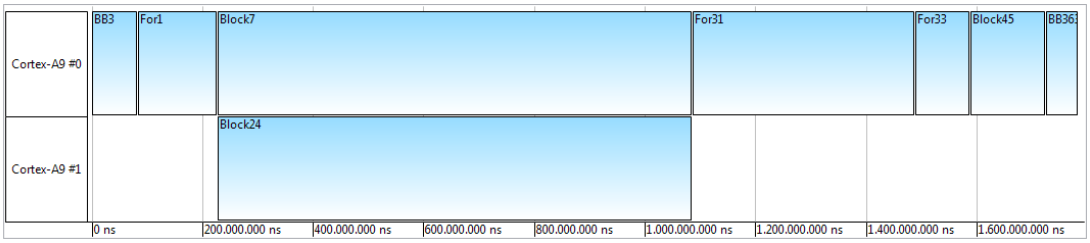

Abbildung 3: Paralleles Schedule auf zwei Prozessorkernen

Diese Programmteile findet man ebenfalls in der Scheduling-Ansicht wieder, die den Ablauf des parallelen Programms repräsentiert. Hier kann man erkennen, dass die beiden 2D-Faltungsoperationen auf zwei unterschiedliche Prozessoren verteilt wurden und dadurch die Anwendung um ca. 50% beschleunigt wird.

Rapid Prototyping
Das emmtrix Parallel Studio erlaubt das direkte Ausführen der parallelisierten Anwendung auf der Zielarchitektur. Dazu wird zunächst eine automatische Codegenerierung basierend auf dem berechneten Schedule durchgeführt und paralleler Code erzeugt. Für den parallelen Code kann unter verschiedenen APIs wie z.B. MPI, Prozesse oder pthreads ausgewählt werden. Für die Zynq-Zielarchitektur bieten sich pthreads (POSIX Threads) an, da die Architektur über einen gemeinsamen Speicher verfügt.

Für das Rapid Prototyping wird der generierte Code mit einem Template für das Zielsystem kombiniert. Im Template ist die Initialisierung, Hauptschleife, die Ansteuerung der Kamera und des Bildschirms sowie die Hardware-Implementierung der Interface-Funktionen realisiert.

Die finalen C-Dateien werden auf das Zielsystem übertragen und dort direkt lokal in Linux kompiliert. Dadurch erspart man sich auf dem Host-System die Einrichtung eines Cross-Compilers. Nach der Kompilierung kann es direkt auf dem Zielsystem ausgeführt werden und man kann seine Anwendung live testen. Der gesamte Vorgang von der sequentiellen Codegenerierung bis zur Ausführung der Anwendung ist voll automatisiert und läuft in kurzer Zeit durch die gesamte Toolchain.

Hardware Profiling

Neben dem funktionalen Test der Anwendung, wird beim Rapid Prototyping noch die Performanz auf der Zielarchitektur evaluiert. Dazu werden im Anwendungscode Profiling-Befehle integriert. Diese messen auf der Hardware das Schedule der parallelen Anwendung und erlauben somit einen Vergleich mit dem vorher berechneten Schedule.

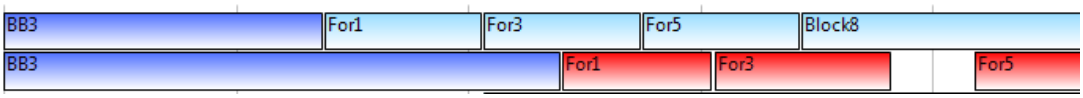

Abbildung 4: Vergleich von Messung und Vorberechnung

Im emmtrix Parallel Studio können die Unterschiede visualisiert werden. In Abbildung 4 ist der Unterschied auf dem ersten Prozessorkern vom berechneten zum gemessenen Schedule zu sehen. Hierbei ergibt sich insbesondere eine Abweichung für den Interface-Code (BB3). Bei einem weiteren Durchlauf können die Abweichungen dann berücksichtigt werden.

Unterstützung von heterogenen Systemen

Der vorgestellte Workflow für die Parallelisierung lässt sich auch einfach auf FPGA-Beschleunigern anwenden. In unserem Beispiel möchten wir jetzt einen der beiden Faltungsfilter auf dem FPGA ausführen. Per Rechtsklick auf den Block30 bzw. die conv2_2 Funktion, kann dieser manuell auf den FPGA gesetzt werden. Im Hintergrund wird dann per High-Level Synthese von Xilinx [4] das C-Teilprogramm als Hardwarebeschleuniger synthetisiert. Nach einer Synthese von ca. 1 Minute stehen dann erste Performanzinformationen zur Verfügung, die dann für das Schedule verwendet werden können.

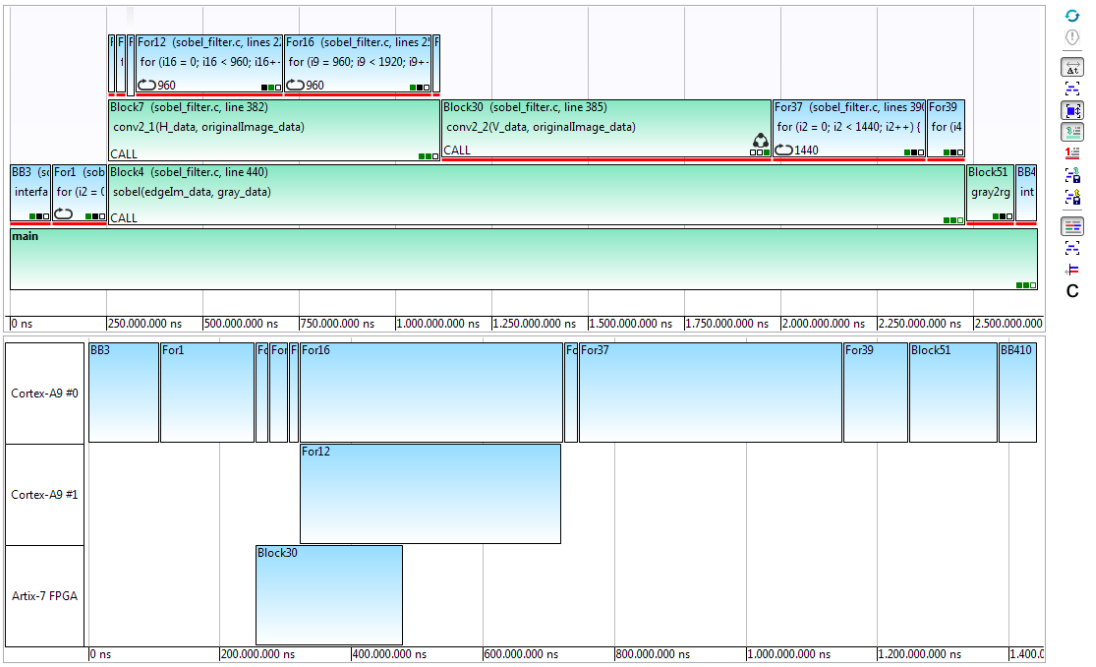

Abbildung 5: Verwendung von FPGA-Beschleuniger

Die frei gewordenen Kapazitäten auf dem zweiten Prozessorkern kann man dann z.B. durch Aufteilung des ersten Faltungsfilters auf zwei Prozessorkernen nutzen, wodurch insgesamt die Performanz der Anwendung weiter verbessert wird.

Zusammenfassung

In diesem Artikel wurden die emmtrix Werkzeuge vorgestellt, welche die Programmierung von eingebetteten heterogenen Multicore-Systemen erleichtern, indem die sonst zeitaufwändige Parallelisierung automatisiert durchgeführt wird. Bei Evaluationen konnte gezeigt werden, dass dadurch der Entwicklungsaufwand zwischen 50-80% [5] reduziert werden kann.

An einem Beispiel aus der industriellen Bildverarbeitung wurde eine MATLAB®-Anwendung für ein Zynq-System optimiert. Zunächst wurde der MATLAB®-Code in sequentiellen C-Code übersetzt und die Performanz der Programmteile in einer hierarchischen Darstellung angezeigt. Die hierarchische Darstellung erlaubt hierbei die Analyse des Programms und die Steuerung der Parallelisierung. Der Benutzer kann hier z.B. festlegen, auf welchem Prozessorkern ein Anwendungsteil ausgeführt werden soll. Zusätzlich können auch einzelne Programmteile auf den FPGA ausgelagert werden. In einer Scheduling-Ansicht wird dann der genaue Ablauf der parallelisierten Anwendung angezeigt. Mittels Codegenerierung wird automatisch paralleler C-Code erzeugt. Dieser kann direkt für die Zielarchitektur kompiliert und auf dieser ausgeführt werden, wodurch Rapid Prototyping ermöglicht wird.

Literatur- und Quellenverzeichnis

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

[2] „ZedBoard™ - Xilinx Zynq®-7000 All Programmable SoC," [Online]. Available: http://zedboard.org/product/zedboard.

[3] „emmtrix Parallel Studio," emmtrix Technologies GmbH, 2016. [Online]. Available: http://www.emmtrix.com/products/emmtrix-parallel-studio.

[4] „Vivado High-Level Synthesis," Xilinx, Inc., [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html.

[5] „ALMA Project, Test cases evaluation report," 2015. [Online]. Available: http://www.alma-project.eu/downloads/FP7-ICT-2011-7-287733-WP5-D5.6.pdf.

Autoren

Dr.-Ing. Timo Stripf hat am Karlsruhe Institut für Technologie (KIT) Informatik studiert und erlangte seinen Doktor in Elektrotechnik im Bereich Compilerbau im Dezember 2013. Er beschäftigt sich bereits seit Anfang 2012 mit automatischer Parallelisierung. Heute ist er technischer Geschäftsführer der emmtrix Technologies GmbH.

Kontakt

emmtrix Technologies GmbH,
Engesserstraße 5, 76131 Karlsruhe
Internet: www.emmtrix.com
Email: stripf@emmtrix.com
Telefon: +49 (721) 608-46884

Dipl.-Inform. Michael Rückauer hat ein Diplom in Informatik von der Universität Karlsruhe, heute Karlsruher Institut für Technolog (KIT). Er ist Experte für Mapping und Scheduling auf heterogenen Systemen sowie FPGA-Hardwareentwicklung und High-Level Synthese. Er ist Senior Engineer bei der emmtrix Technologies GmbH.

Dipl.-Ing. Oliver Oey hat ein Diplom in Elektrotechnik vom Karlsruher Institut für Technologie (KIT). Er ist Experte in der Parallelisierung von Anwendungen und der parallelen Codegenerierung. Er ist Senior Engineer bei der emmtrix Technologies GmbH.