Fallstudie: Terrain Awareness and Warning System

Automatische Parallelisierung eines modellbasierten Designs

Peer Ulbig, Umut Durak, David Müller;
Deutsches Zentrum für Luft- und Raumfahrt e.V.

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

Der Luft- und Raumfahrtsektor verlangt nach neuen Methoden und Ansätzen zur kosteneffizienten Steigerung der Performanz von Anwendungen unter Beibehaltung des Sicherheitsniveaus und der Programmierbarkeit. Ein vielversprechender Ansatz besteht in der Verwendung von Multicore-Architekturen. Um ihr volles Potenzial ausschöpfen zu können, besteht neben der notwendigen Zertifizierung allerdings auch ein Bedarf an Programmierwerkzeugen und -prozessen. Dabei ist die Unterstützung des modellbasierten Entwurfs zur Vereinfachung der Systemmodellierung, der Verifikation und der Validierung von Designentscheidungen entscheidend. In diesem Paper wird daher ein modellbasierter Entwurf für Multicore-Architekturen vorgestellt. Als Fallstudie wird ein „Terrain Awareness and Warning System“ (TAWS) modellbasiert entwickelt und mit Hilfe der interaktiven Parallelisierungslösung emmtrix Parallel Studio (ePS) für die Ausführung auf einem Infineon Aurix TC297B Mikrocontroller parallelisiert.

Der Bedeutungszuwachs der Cyberanteile in Flugzeugen, wie Avioniksysteme, Sensoren, Aktoren und Datenbusse, führte zu einem Paradigmenwechsel von verteilten Onboard-Systemarchitekturen hin zu „Integrated Modular Avionics“ (IMA) [1]. Statt dezentraler und dedizierter Recheneinheiten verbindet man IMA damit, dass mehrere Anwendungen dieselbe Recheneinheit verwenden [2]. Dabei wurden höhere Durchsatzanforderungen immer wichtiger und schließlich auch Multicore- und Multiprozessor-Systeme zu einem wichtigen Thema.

Viele Bemühungen zur Parallelisierung bei der Entwicklung von Flugsystemen mit Multicore-Architekturen konzentrierten sich auf die Anwendbarkeit hinsichtlich der Sicherheitseinschränkungen im Bereich der Avionik [3, 4, 5, 6, 7, 8, 9]. Segregation, Integrität, Berechenbarkeit, Zertifizierungskosten und Leistung sind wichtige Herausforderungen, die es zu bewältigen gilt [8]. Darüber hinaus bleibt eine effektive und effiziente Entwicklungsmethodik für Avionik-Anwendungen mit Multicore-Architekturen eine Forschungsfrage. Im Bereich der Luft- und Raumfahrt müssen noch komplexe Werkzeugketten und Programmierprozesse entwickelt werden, um das volle Potenzial dieser heterogenen parallelen Plattformen der nächsten Generation ausschöpfen zu können.

ARGO (WCET-Aware PaRallelization of Model-Based Applications for Hetero-GeneOus Parallel Systems) ist ein im Rahmen des Forschungs- und Innovationsprogramms Horizon 2020 der Europäischen Union gefördertes Projekt. Es befasst sich

mit der Codegenerierung für Worst Case Execution Time (WCET)-bewusste Parallelisierung von modellbasierten Anwendungen für Multicore-Systeme [10, 11].

Dieses Paper stellt eine Fallstudie vor, in der ein Terrain Awareness and Warning System (TAWS) modellbasiert entwickelt und mit Hilfe der interaktiven Parallelisierungslösung emmtrix Parallel Studio (ePS) für die Ausführung auf dem Infineon AURIX optimiert wurde. Als konkretes Beispiel wurde eine Nachbildung des Enhanced Ground Proximity Warning Systems (EGPWS) gewählt.

Der modellbasierte Design-Workflow von ARGO

Die modellbasierte Entwicklung fördert den Aufbau von Modellen und die Generierung ausführbarer Software durch sukzessive Modell-zu-Modell- und Modell-zu-Text-Transformationen [12]. Der modellbasierte Entwurf ist eine Variante der modellbasierten Entwicklung. Er zeichnet sich durch den nahtlosen Einsatz von ausführbaren und grafischen datenflussorientierten Blockdiagrammmodellen und Zustandsautomaten für die Systemspezifikation, sowie den Entwurf und die Implementierung aus. Dabei kommen modellbasierte Entwurfs- und Simulationswerkzeuge wie Scilab/Xcos oder MATLAB/Simulink zum Einsatz [13].

Der modellbasierte ARGO-Workflow (Abb. 1) beginnt mit der Regler-Modellierung, bei der die Anwendungsmodelle in der Scilab/Xcos-Umgebung implementiert werden. Scilab/Xcos ist eine modellbasierte Open Source Design- und Simulationsumgebung [14]. Dabei kann die Scilab-Skriptsprache in einzelnen graphischen Blöcken des Modells verwendet werden, wodurch eine kombinierte graphische und imperative Beschreibung im Modell ermöglicht wird. Die Zielarchitektur wird ebenfalls gleich zu Beginn mit Hilfe einer Architekturbeschreibungssprache (ADL) festgelegt.

Im ersten Schritt (Frontend Tools) wird die Modell-zu-Text-Transformation verwendet, um aus den Scilab/Xcos-Modellen plattformunabhängigen, sequentiellen C-Quellcode zu generieren. Ausgehend davon führt das „GeCoS Source-to-Source Transformation Framework“ [15] Programmtransformationen durch, die z.B. auf Parallelisierung oder Optimierung abzielen. Dazu wird zunächst das C Programm in einen hierarchischen Task-Graphen (HTG) [16] überführt.

Der HTG enthält Informationen über Abhängigkeiten und Ausführungszeiten. So sind zwei Tasks voneinander abhängig, wenn sie auf die gleichen Daten zugreifen. In der Schedulingphase wird der HTG auf die einzelnen Kerne der Multicore-Zielplattform abgebildet. Dabei wird festgelegt, welche Tasks in welcher Reihenfolge auf welchem Kern ausgeführt werden. Wenn zwei Tasks voneinander abhängig sind, dann müssen die Tasks im Schedule nacheinander ausgeführt werden. Im Schritt „Data Management, Synchronization and Code Generation“ aus Abbildung 1 werden die Ergebnisse des Schedulings verwendet, um eine explizite parallele Programmdarstellung mit Kommunikation, Synchronisationen und Adress-Mappings zu erzeugen. Der WCET-Schritt auf Code- und Systemebene berechnet die Multi-Core WCET für die Zielarchitektur. Als Ergebnis wird ein paralleles C-Programm generiert.

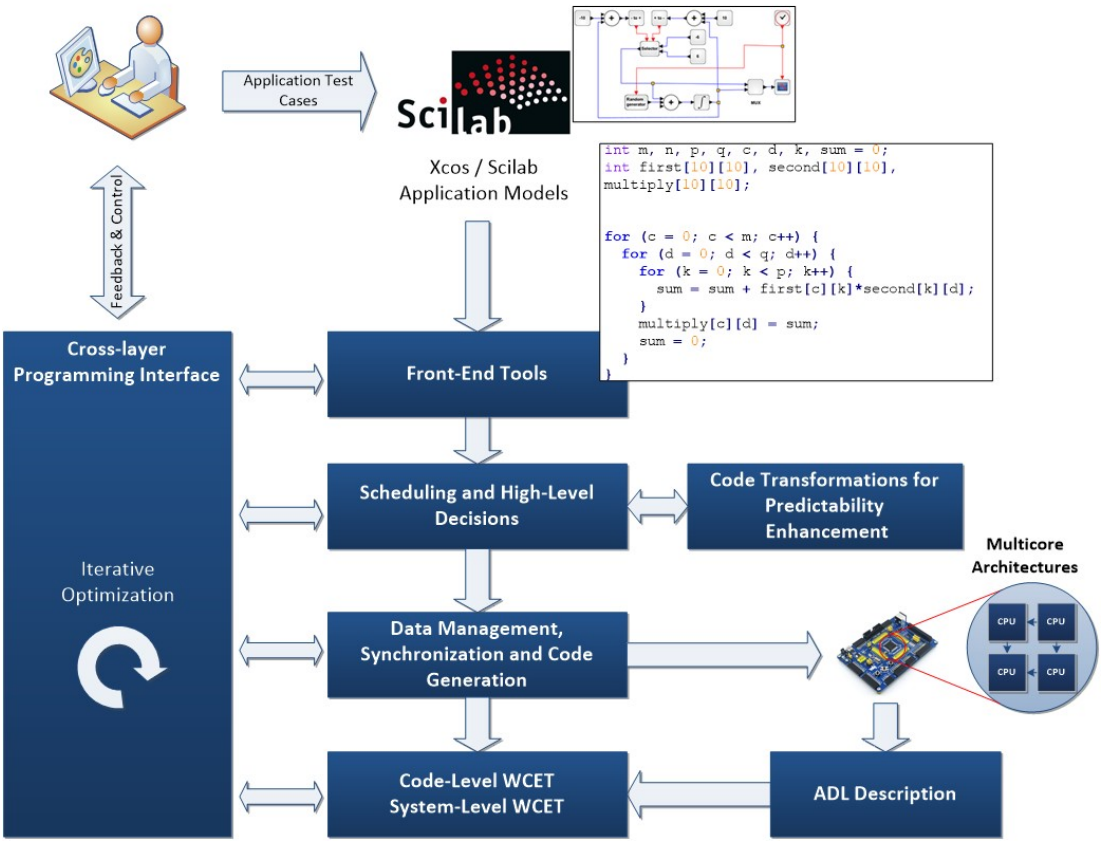

Abb.1: Der modellbasierte Design-Workflow von ARGO

Terrain Awareness and Warning Systems (TAWS)

Controlled Flight Into Terrain (CFIT) war für viele Todesfälle in der Zivilluftfahrt verantwortlich, bis die Installation von Terrain Awareness and Warning Systems (TAWS) von vielen Luftfahrtbehörden vorgeschrieben wurde (vgl. Federal Aviation Administration, [17]). Es gibt verschiedene TAWS-Optionen auf dem Markt, die eine Reihe von Funktionen zur Verfügung stellen, um dem Risiko von CFIT entgegenzuwirken. T3CAS von ACSS [18] und TAWS von Universal Avionics [19] sind zwei davon. Ein kurzer Vergleich dieser Systeme ist in [20] zu finden. Das Enhanced Ground Proximity Warning System (EGPWS) ist ein weiteres Beispiel für ein TAWS, es wurde von Honeywell [21] entwickelt und dient in den folgenden Abschnitten als Vorbild.

Die Kernfunktion des EGPWS besteht darin, visuelle und akustische Warnungen zwischen 30 ft und 2450 ft über Grund (AGL) zu erzeugen, um CFIT zu vermeiden. Diese Warnungen sind in fünf Modi unterteilt:

Modus 1 Gefährliche Sinkgeschwindigkeit: Warnungen vor überhöhter Sinkgeschwindigkeit in allen Flugphasen.

Modus 2 Gefährliche Bodenannäherungsrate: Warnungen zum Schutz des Flugzeugs vor dem Aufprall auf den Boden, wenn das Gelände in Relation zum Flugzeug schnell ansteigt.

Modus 3 Höhenverlust nach Start: Warnungen, wenn ein signifikanter Höhenverlust nach dem Start oder während eines Go Around-Manövers in einer niedrigen Flughöhe festgestellt wird.

Modus 4 Unsicheres Gelände: Warnungen, wenn keine ausreichende Bodenfreiheit bezüglich Flugphase, Flugzeugkonfiguration und Geschwindigkeit vorhanden ist.

Modus 5 Abweichung unter Gleitpfad: Warnungen, wenn das Flugzeug bei der Instrumentenlandung zu weit unter dem Gleitpfad fliegt.

Abbildung 2 verdeutlicht den ersten Modus des EGPWS. Die drei Flugzeuge haben die gleiche Höhe von etwa 2000 ft, aber unterschiedliche Sinkgeschwindigkeiten, was durch ihre Position und Ausrichtung in der Grafik verdeutlicht wird. Während sich das grüne Flugzeug in einem sicheren Flugzustand befindet, wird durch die Sinkrate des orangefarbenen Flugzeugs eine Warnung ausgelöst. Das rote Flugzeug sinkt in Relation zu seiner geringen Höhe viel zu schnell und erfordert ein sofortiges Eingreifen des Piloten, weshalb eine eindringlichere Warnung ausgegeben wird.

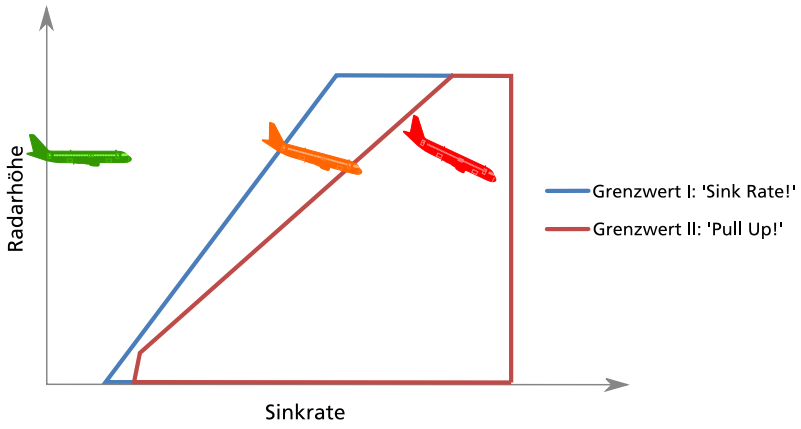

Abb. 2: EGPWS-Modus 1: Gefährliche Sinkgeschwindigkeit.

Zusätzlich zu den fünf genannten Modi bietet das EGPWS einige erweiterte Funktionen, wie das „Terrain Look Ahead Alerting“ und das „Terrain Alerting and Display“, die auf einer Geländedatenbank basieren. Beim Terrain Look Ahead Alerting wird bis zu einer Entfernung, die bei der aktuellen Fluggeschwindigkeit 60 s Flugzeit entspricht, anhand der Geländedatenbank bestimmt, ob eine Kollision mit dem Gelände droht oder nicht. In einem solchen Fall würde das System eine Ermahnung ausgeben. Kann es darüber hinaus bereits innerhalb der nächsten 30 s Flugzeit zu einer Kollision kommen, kommt es zu einer intensiveren Warnung durch das System. Diese Zusammenhänge verdeutlicht Abbildung 3.

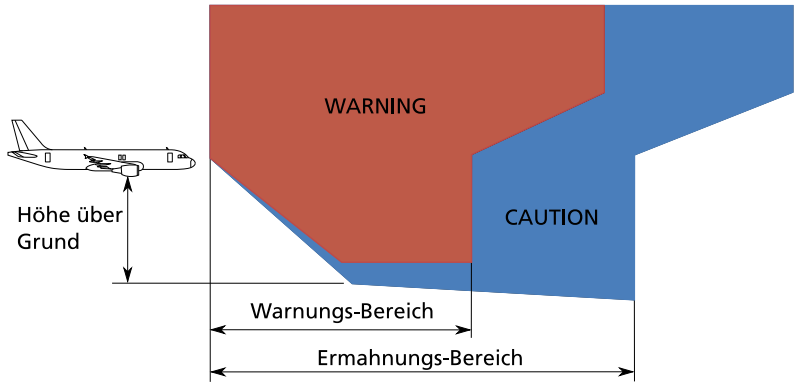

Abb. 3: Prinzip des „Terrain Look Ahead Alerting“ beim EGPWS.

Modellierung des TAWS in Xcos

Dem modellbasierten Design-Workflow von ARGO entsprechend wurden die fünf Basismodi des EGPWS in Scilab/Xcos nachgebildet. Abbildung 4 zeigt beispielhaft die Nachbildung des EGPWS-Modus 1 (Abb. 2) im Scilab/Xcos-Modell.

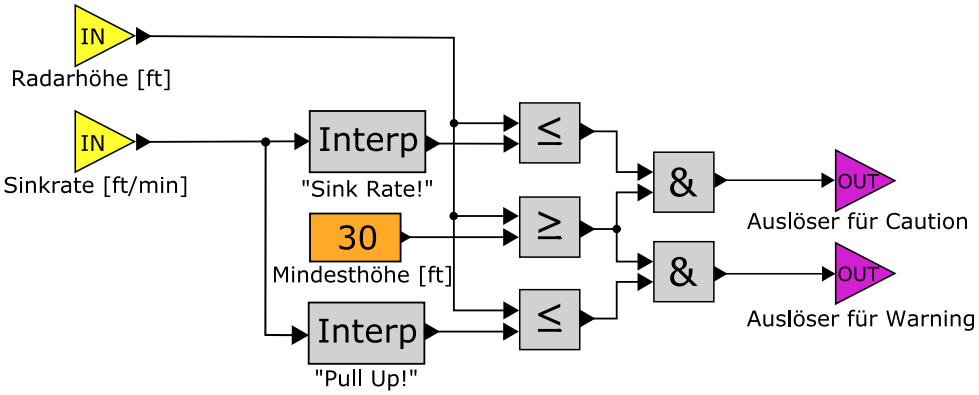

Abb. 4: Nachbildung des EGPWS-Modus 1 in Scilab/Xcos.

Das Terrain Look Ahead Alerting und das Terrain Alerting and Display wurden zudem als Scilab-Skripte implementiert.

Automatische Parallelisierung des TAWS

Die Parallelisierung des TAWS nach Vorbild des EGPWS erfolgt mit dem übergeordneten Ziel, dass das System anschließend für Testzwecke auf verschiedenen Wegen mit dem Air VEhicle Simulator (AVES) des DLR Braunschweig kommunizieren können soll. Zunächst ist eine Processor-in-the-Loop Testreihe geplant, bei der für das EGPWS üblicherweise eingesetzte Datenbusse, wie beispielsweise ARINC 429, durch eine einzige Ethernet-Verbindung zum Simulator ersetzt werden.

Das Zielsystem für die Processor-in-the-Loop Testreihe ist das Aurix TC297 Starter Kit, welches einen Aurix TC297B Mikrocontroller mit drei Kernen und integrierter Ethernet Schnittstelle (MAC) bietet. Die Bitübertragungsschicht (PHY) ist als inte-

grierter Schaltkreis auf dem Starter Kit vorhanden. Weiterhin wird das Echtzeitbetriebssystem FreeRTOS in einer für das Zielsystem portierten Version eingesetzt. Auf einem Kern soll als Ethernet Task (eth_task) die asynchrone Kommunikation mit dem Flugsimulator betrieben werden. Auf den beiden verbleibenden Kernen wird die hardwareunabhängige parallelisierte Logik des EGPWS als Application Task (app_task) ausgeführt. Theoretisch führen zwar alle Kerne beide Tasks aus, in Abhängigkeit vom jeweiligen Kern gibt jedoch einer der beiden Tasks durch unmittelbares Schlafenlegen die entsprechende CPU wieder frei. Der Datenaustausch zwischen den Tasks erfolgt über einen gemeinsamen, durch einen Mutex geschützten Speicherbereich (Shared Memory) in der Local Memory Unit (LMU) des Aurix TC297B. Auf diese Weise werden die physikalischen Schnittstellen für die TAWS-Logik durch Speicherzugriffe abstrahiert. Abbildung 5 verdeutlicht diese Ansätze.

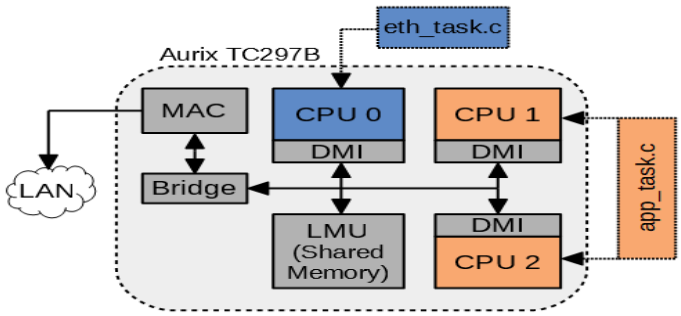

Abb. 5: Vereinfacht dargestellte Processor-in-the-Loop Konfiguration des parallelisierten TAWS.

Der Ausgangspunkt des ARGO-Workflows für die Parallelisierung des TAWS ist zunächst Scilab. Hier wird das Xcos-Modell des TAWS in Scilab-Code übersetzt. Dabei werden neben der TAWS-Schrittfunktion Dateneingangs- und -ausgangsfunktionen sowie eine Szenario-Datei erzeugt, welche Eingangs-, Ausgangs- und Zustandsvariablen initialisiert und die genannten Funktionen nacheinander aufruft. Das so erzeugte Scilab-Szenario ist wiederum die Ausgangsbasis für die eigentliche Parallelisierung mit Hilfe des ePS. Ein neues Projekt mit der Zielhardware Aurix TC297 Starter Kit und der Eingangsprogrammiersprache Scilab wird im ePS erstellt und der zuvor generierte Scilab-Code eingebunden.

Im nächsten Schritt wird aus dem Scilab-Code sequentieller C-Code generiert. Dieser Schritt beeinflusst bereits maßgeblich den später erzeugten parallelen C-Code. Codeoptimierungen in Bezug auf Wertebereiche der Eingangs-, Ausgangs-, und Zustandsvariablen im konkreten Scilab-Szenario werden unterdrückt. So wird sichergestellt, dass der C-Code die TAWS-Schrittfunktion für alle möglichen Kombinationen und Abfolgen von Eingangswerten unabhängig vom gewählten Scilab-Szenario richtig wiedergibt. Zudem werden die Dateneingangs- sowie die Datenausgangsfunktion in separaten Quellcode ausgelagert. Auf diese Weise werden nur deren Funktionsaufrufe bei der späteren Parallelisierung berücksichtigt, im Gegensatz zur TAWS-Schrittfunktion jedoch nicht deren Inhalte. Dieses Vorgehen ist beim TAWS sinnvoll, da die Dateneingangs- und die Datenausgangsfunktion nachträglich für den Zu-

griff auf den gemeinsamen Speicher mit dem Ethernet Task umgeschrieben werden sollen.

Es folgt die automatische Generierung des parallelen C-Codes aus dem sequentiellen C-Code. Wie zuvor beschrieben, soll dieser nur für zwei Kerne als Application Task parallelisiert werden. Dazu wird für den ersten Kern ein sogenanntes „Core Constraint“ gesetzt, das diesen Kern von der Parallelisierung ausschließt. Dieser Kern steht somit exklusiv für die asynchrone Kommunikation mit dem Simulator zur Verfügung. Das Resultat der Parallelisierung sind dedizierte Funktionen für die jeweiligen Kerne und die Funktion app_task(), die diese in Abhängigkeit des ermittelten Kernes aufruft. Die Funktion main_p0() des Application Tasks für den ersten Kern ist wie erwartet leer. Main_p1() und main_p2() enthalten die jeweiligen Anteile der parallelisierten TAWS-Logik, Aufrufe der Datenein- und -ausgangsfunktion und zusätzliche API-Aufrufe, welche zum Datenaustausch untereinander und zur Sicherstellung der Synchronität genutzt werden. Der Ethernet Task ist im Gegensatz zum Application Task kein Resultat der Parallelisierung, sondern wird manuell für die Ausführung auf dem ersten Kern implementiert.

Im ePS lässt sich das Ergebnis der automatischen Parallelisierung im „Multicore Schedule View“ mit dedizierten Spuren für jeden Kern grafisch nachvollziehen. Durch Klicken auf Codeelemente werden deren Beziehungen zu anderen Codeelementen mit Pfeilverbindungen angedeutet. In Abbildung 6 kann beispielsweise Kern 1 (Tricore#1 in Abb. 6) die For-Schleife „For16“ erst berechnen, wenn Kern 2 (Tricore#2 in Abb. 6) das Codeelement „BB236“ beendet hat.

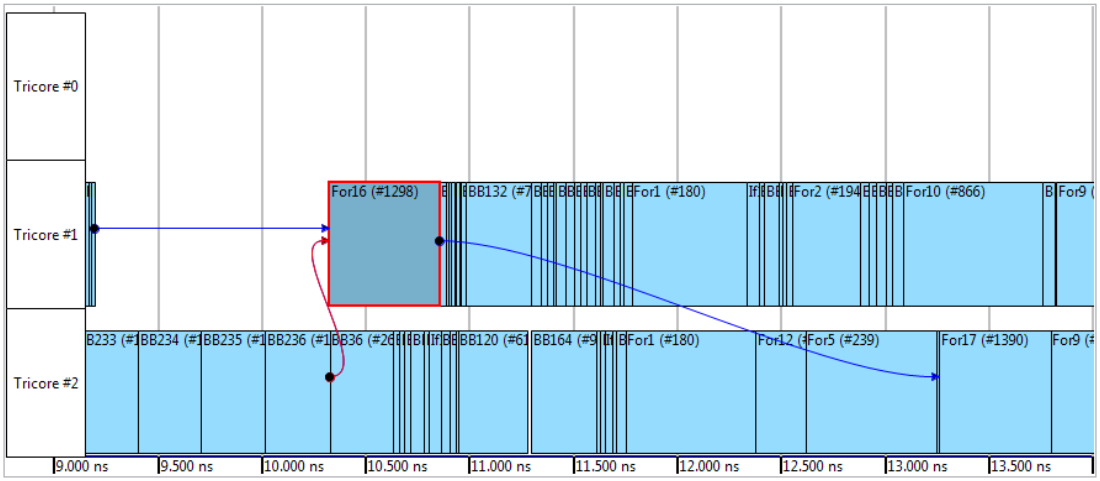

Abb. 6: Auszug aus dem Programmablauf des parallelisierten TAWS im „Multicore Schedule View“ des emmtrix Parallel Studios (ePS).

Die vom ePS prognostizierte WCET des parallelisierten TAWS wird im Vergleich zum sequentiellen Code um 25% verringert. Der systemspezifische hohe zusätzliche Kommunikationsaufwand und damit teilweise verbundene Wartezeiten einzelner

Kerne begründen die Abweichung von der theoretisch maximal möglichen Laufzeitreduzierung um 50% für zwei Kerne.

Zusammenfassung

In diesem Paper wird ein modellbasierter Entwurf für Multicore-Prozessoren vorgestellt. Als Fallstudie wurde ein Terrain Awareness and Warning System (TAWS) nach Vorbild des Enhanced Ground Proximity Warning Systems (EGPWS) modellbasiert in Xcos entwickelt. Das Xcos-Modell wurde zunächst in Scilab-Code überführt. Mit Hilfe der interaktiven Parallelisierungslösung emmtrix Parallel Studio (ePS) wurde der Scilab-Code zunächst in sequentiellen C-Code transformiert und danach für die Ausführung auf einem Infineon Aurix TC297B Mikrocontroller parallelisiert und optimiert. Die ersten Ergebnisse zeigen durch die Parallelisierung bereits eine Reduzierung der Laufzeit von 25%. Für die Zukunft ist bezüglich des ePS eine Optimierung der Kommunikation geplant sowie eine Portierung auf die nächste Generation von Infineon Aurix mit sechs Kernen angedacht.

Literatur- und Quellenverzeichnis

[1] Prisaznuk, P. J. 1992. “Integrated modular avionics”. In *Proceedings of the IEEE 1992 National Aerospace and Electronics Conference*, Dayton, OH.

[2] Watkins, C. B. and R. Walter. 2007. “Transitioning from federated avionics architectures to integrated modular avionics”. In *IEEE/AIAA 26th Digital Avionics Systems Conference*, Dallas, TX.

[3] Nowotsch, J. and M. Paulitsch. 2012. “Leveraging multi-core computing architectures in avionics”. In *9th European Dependable Computing Conference (EDCC)*, Sibiu, Romania.

[4] Karray, H., M. Paulitsch, B. Koppenhöfer, and D. Geiger. 2013. “Design and implementation of a degraded vision landing aid application on a multicore processor architecture for safety-critical application”. In *16th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, Paderborn, Germany.

[5] Koppenhöfer, B., and D. Geiger. 2016. “EMC2 Use Case:Hybrid Avionics Integrated Architecture Demonstrator”. In *HiPEAC, Workshop EMC²*, Prague, Czech Republic.

[6] Agrou, H., P. Sainrat, M. Gatti, and P. Toillon. 2012. “Mastering the behavior of multi-core systems to match avionics requirements”. In *AIAA 31st Digital Avionics Systems Conference (DASC)*, Williamsburg, VA.

[7] Löfwenmark, A., and S. Nadjm-Tehrani. 2014. “Challenges in future avionic systems on multi-core platforms”. In *International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Naples, Italy.

[8] Sander, O., F. Bapp, L. Dieudonne, T. Sandmann, and J. Becker. 2017. “The promised future of multi-core processors in avionics systems”. *CEAS Aeronautical Journal* vol.8, no. 1, pp. 143-155.

[9] Bapp, F., and J.Becker. 2018. “Advances in Avionic Platforms: Multi-Core Systems”. In *Advances in Aeronautical Informatics: Towards Flight 4.0*, edited by U.Durak, J.Becker, S.Hartmann and N.Voros, Springer.

[10] Derrien, S., I. Puaut, P. Alefragis, M. Bednara, H. Bucher, C. David, Y. Debray, U. Durak, I. Fassi, C. Ferdinand, and D. Hardy. 2017. “WCET-aware parallelization of model-based applications for multi-cores: The ARGO approach”. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Lausanne, Switzerland.

[11] Durak, U., D. Müller, J. Becker, N. Voros, P. Alefragis, T. Stripf, P.A. Agnel, G. Rauwerda and K. Sunesen. 2016. “Model-based development of Enhanced Ground Proximity Warning System for heterogeneous multi-core architectures”. In *ASIM 2016*, Dresden, Germany.

[12] Gasevic, D., D. Djuric, and V. Devedic. 2009. *Model driven engineering and ontology development.* Springer Science & Business Media.

[13] Stürmer, I., M. Conrad, I Fey and H Dörr. 2006. “Experiences with model and autocode reviews in model-based software development”. In *Proceedings of the 2006 International Workshop on Software Engineering for Automotive Systems*, Shanghai, China.

[14] Campbell, S.L., J.P. Chancelier, and R. Nikoukhah. 2010. *Modeling and Simulation in Scilab/Scicos with ScicosLab 4.4*. Springer New York.

[15] Floch, A., T. Yuki, A. El-Moussawi, A. Morvan, K. Martin, M. Naullet, M. Alle, L. L'Hours, N. Simon, S. Derrien, and F. Charot. 2013. “GeCoS: A framework for prototyping custom hardware design flows”. In *IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, Eindhoven, Netherlands.

[16] Girkar, M. and C.D. Polychronopoulos. 1994. “The hierarchical task graph as a universal intermediate representation”. *International Journal of Parallel Programming*, vol.22, no.5, pp.519-551.

[17] Federal Aviation Administration. 2000. “Installation of Terrain Awareness and Warning Systems (TAWS) Approved for Part 23 Airplanes”, Advisory Circular 23-18, Washington, D.C.

[18] ACSS. 2018. “T3CAS”. http://www.l3aviationproducts.com/products/t3cas/. Letzter Zugriff am 16.08.2018.

[19] Universal Avionics. 2018. “TAWS | Terrain Awareness and Warning System”. http://www.uasc.com/home/shop/avionics/taws. Letzter Zugriff am 16.08.2018.

[20] Smith, D. 2005. “Traffic Alert Collision Avoidance Systems—TCAS Buyer’s Guide”. *Pilot’s Guide to Avionics 2005*, pp. 34-41.

[21] Honeywell. 2018. “Terrain & Traffic Awareness”. https://aerospace.honeywell.com/en/product-listing/terrain-and-traffic-awareness. Letzter Zugriff am 16.08.2018.