Mixed-Criticality Systeme durch Echtzeitfähigkeitsklassen

Echtzeitanwendungen im Kontext paralleler Programmiermodelle

Tobias Langer, Lukas Osinski und Jürgen Mottok, OTH Regensburg
Tobias Schüle, Siemens AG
Ralph Mader, Continental AG

Moderne eingebettete Systeme erfordern die parallele Ausführung von Anwendungen mit variierenden Kritikalitäten hinsichtlich funktionaler Sicherheit und Echtzeitverhalten. Echtzeitfähigkeitsklassen bilden eine Basis mit der solche Systeme entworfen werden können. Wir präsentieren eine erste Beschreibung des Echtzeitfähigkeitsklassenmodells, sowie seine Referenzimplementierung auf Basis der Embedded Multicore Building Blocks (EMB²), einer Bibliothek für die parallele Programmierung eingebetteter Systeme.

Im Bereich eingebetteter Plattformen lassen sich derzeit zwei Trends verfolgen: Zum einen steigt die Zahl der integrierten Kerne, so dass in Abgrenzung zu Multicore-Systemen bereits von Manycore-Systemen gesprochen wird; zum anderen werden zunehmend auf spezielle Anwendungen zugeschnittene Hardware-Komponenten wie DSPs, GPUs oder FPGAs in Prozessoren integriert, wodurch ein Wandel weg von homogenen hin zu heterogenen Systemen vollzogen wird.

Ein aktuelles Kernproblem ist die optimale Ausnutzung dieser Systeme. Hierbei helfen insbesondere moderne Task-basierte Konzepte für die parallele Programmierung, die mittlerweile in den meisten Sprachen zu finden sind. Mit EMB² steht eine als Open Source Software verfügbare Lösung bereit, die auch auf die Anforderungen an eingebettete Systeme zugeschnitten ist. Es zeichnet sich jedoch ab, dass zukünftige Systeme den parallelen Betrieb von Anwendungen gemischter Kritikalität, hinsichtlich Echtzeitfähigkeit und funktionaler Sicherheitsanforderungen, erlauben müssen.

Das im Folgenden vorgestellte Echtzeitfähigkeitsklassenmodell separiert eingebettete Systeme in Stufen, welche Anwendungen entlang ihrer Kritikalität voneinander isolieren. Zudem beschreiben wir eine Referenzimplementierung einer dieser Kritikalitätsstufen in EMB².

EMB²

Die Embedded Multicore Building Blocks [1][2] sind eine von Siemens entwickelte Bibliothek für die Entwicklung paralleler Anwendungen, insbesondere für heterogene Systems-on-a-Chip. EMB² baut auf der Multicore Task Management API (MTAPI) auf, einer von der Multicore Association standardisierten, Spezifikation für die Umsetzung von Task-Parallelität in eingebetteten Systemen.
Zudem enthält EMB² nicht-blockierenden, threadsicheren Datenstrukturen, parallele Algorithmen sowie Schablonen für die Entwicklung von datenflussorientierten Applikationen.

MTAPI beschreibt Komponenten heterogener eingebetteter Systeme als Knoten (Nodes). Ein Prozessor mit 4 Kernen bildet beispielsweise einen eigenen Knoten, ebenso wie eine GPU oder ein DSP. Diese Nodes stehen dann zur Ausführung von Anwendungen bereit, wobei jeder Knoten sein eigenes Scheduling definiert. Es sei angemerkt, dass MTAPI keine Schedulingstrategien explizit vorschreibt.

MTAPI-Anwendungen bestehen aus mehreren Tasks, dabei bezeichnet ein Task eine bestimmte Aufgabe, sowie den zugehörigen Daten. Die Aufgaben können sowohl durch Programmcode für verschiedene Prozessorarchitekturen als auch durch Hardware realisiert werden.

Tasks werden innerhalb der Knoten verwaltet und ausgeführt. EMB² realisiert dies durch prioritätenbasiertes Work Stealing (eine allgemeine Beschreibung von Work Stealing ist in [3] zu finden). Jeder Kern verfügt über eine Queue pro Prioritätsstufe, welche abgearbeitet wird. Sobald ein Kern leerläuft, „stiehlt“ er sich einen unbearbeiteten Task von einem anderen Kern.

Die Verarbeitung der Tasks verläuft nach Run-to-Completion-Semantik. Einmal gestartet, werden Tasks nicht mehr unterbrochen.
Dies reduziert bei parallelen Anwendungen den Overhead durch Threads, da weder Kontextwechsel notwendig sind, noch Tasks oder Daten zwischen Kernen migriert werden müssen. Außerdem werden potenzielle Race Conditions unterbunden.

Globales Echtzeitscheduling
Mit steigenden Anforderungen an eingebettete Systeme haben Multi- und Manycore-Systeme auch im Bereich der Echtzeitsysteme Einzug gehalten. Diese Systeme stellen immer noch eine große Herausforderung dar, insbesondere das Scheduling. Genaue Analysen der Task-Systeme hinsichtlich ihres Zeitverhaltens gestalten sich oft wesentlich schwieriger, als bei vergleichbaren Single-Core-Systemen.

Daher werden häufig Anwendungstasks bereits zur Designzeit fest auf einzelne Kerne alloziert und dann durch bekannte Single-Core-Schedulingverfahren wie Earliest Deadline First (EDF), Deadline Monotonic oder Rate Monotonic Scheduling verwaltet.
Gerade für Systeme mit wenigen Kernen mag dieser Ansatz zufriedenstellend funktionieren, dennoch ist er mit einer Reihe von Problemen verbunden. Das zugehörige kombinatorische Problem wächst rasant mit der Anzahl der Kerne und der Tasks, sodass die Aufteilung schnell sehr aufwendig wird. Auch lässt eine feste Verteilung der Tasks nur schwer eine optimale Auslastung des Systems zu.

Globale Schedulingverfahren sind eine Alternative zur Task-Partitionierung. Hier werden zur Verarbeitung anstehende Tasks erst zur Laufzeit durch den Scheduler auf freie Kerne des Systems verteilt.

Ein bewährtes Verfahren aus dem Bereich der globalen Schedulingverfahren für Echtzeitsysteme ist Global EDF (GEDF). Bei diesem Vorgehen kann durch Taskmigration jedoch zusätzlicher Overhead entstehen. Gerade bei eingebetteten Systemen mit komplexeren Speicherhierarchien können hier erhebliche Kosten entstehen. Diese Kosten können eingespart werden, wenn Tasks, ähnlich dem EMB²-Modell, nach Run-to-Completion Semantik ausgeführt werden, also nicht-präemptives Scheduling eingesetzt wird. In [4] zeigten Guan et al., dass nicht-präemptives GEDF insbesondere auf Systemen mit vielen Kernen hinsichtlich der Einhaltung von Deadlines vergleichbares Verhalten zu präemptivem GEDF zeigt. Echtzeitgarantien können weiterhin gewährleistet werden.

Echtzeitfähigkeitsklassen
Im Folgenden beschreiben wir die Echtzeitfähigkeitsklassen, welche den zeitgleichen Betrieb von Anwendungen mit verschiedenen Anforderungen, sowohl an Echtzeitfähigkeit als auch an funktionale Sicherheit, erlauben.

Die Klassen spezifizieren festgelegte, unveränderliche Ausführungsmodi. Ein Ausführungsmodus beschreibt, wie die Anwendung ausgeführt wird (Schedulingverfahren, Unterbrechbarkeit von Tasks), und legt den synchronisierten Zugriff auf gemeinsame Ressourcen fest. Abhängig von der Anwendungskritikalität kann der Zugriff auf gemeinsame Ressourcen durch wartefreie Algorithmen ermöglicht werden. Falls notwendig werden explizite Synchronisationsmechanismen, wie Semaphoren und Mutexe, verwendet. Somit wird festgelegt wie und ob Anwendungen verschiedener Betriebsklassen hinsichtlich ihrer Betriebsmittel miteinander interagieren. Damit wird eine Isolation der Anwendungen zueinander garantiert.

Die Unterteilung der Anwendungen in Echtzeitfähigkeitsklassen erfolgt dabei entsprechend ihrer Anforderungen hinsichtlich funktionaler Sicherheit und der daraus resultierenden Echtzeitfähigkeit.

Abbildung *1* zeigt eine Übersicht über das Modell der Echtzeitfähigkeitsklassen.
Ausgehen von der nichtmodifizierten EMB²-Bibliothek erstrecken sie sich, graduell steigend, hin zu harten Echtzeitgarantien.

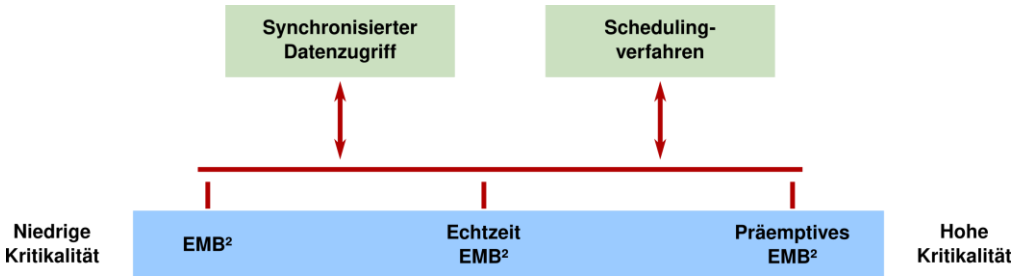

Abbildung 1: Überblick über die Echtzeitfähigkeitsklassen

Echtzeitfähigkeitsklasse „EMB²“
In der EMB² Klasse findet sich die EMB²-Bibliothek nach aktuellem Stand wieder. Anwendungstasks werden durch Round-Robin gleichmäßig auf die Knoten des Systems verteilt. Dort werden sie im Anschluss durch prioritätsbasiertes Work Stealing abgearbeitet, wodurch ein hoher Durchsatz garantiert wird. Die Tasks laufen dabei nach Run-to-Completion Semantik.

Der Zugriff auf gemeinsame Daten wird durch nicht-blockierende Datenstrukturen geschützt.

Echtzeitfähigkeitsklasse „Echtzeit EMB²“
Ebenso wie in der EMB² Klasse werden Tasks in der Echtzeit Klasse mit Run-to-Completion-Semantik abgearbeitet. Der Zugriff auf gemeinsame Ressourcen erfolgt auch über nicht-blockierende Datenstrukturen.

Anstelle von Work-Stealing kommt nicht-präemptives GEDF zum Einsatz. Dadurch werden Tasks mit geringerem Durchsatz abgearbeitet, allerdings können Echtzeitgarantien für die Fertigstellung der Tasks gegeben werden.

Echtzeitfähigkeitsklasse „Präemptives EMB²“
Die Präemptives EMB² Klasse beschreibt klassische, harte Echtzeitsysteme. Tasks können während ihrer Ausführung unterbrochen werden. Dadurch werden negative Einflüsse auf andere Tasks minimiert.

Anstelle der bisherigen nicht-blockierenden Datenstrukturen treten hier blockierende Datenstrukturen und Synchronisationsmittel (bspw. Semaphoren und Mutexe). Mögliche Prioritätsinversionen beim Zugriff auf gemeinsame Ressourcen werden durch den Einsatz entsprechender Protokolle verhindert.

Umsetzung der Echtzeitfähigkeitsklassen in EMB²
Als Referenzimplementierung der Echtzeitfähigkeitsklassen wurde das Echtzeitscheduling, wie es in der Echtzeit EMB² Klasse beschrieben ist, in EMB² umgesetzt. Dazu wurde die MTAPI-Implementierung erweitert und die EMB²-Schnittstellen dahingehend ausgebaut, dass Taskdeadlines bei der Benutzung der parallelen Algorithmen angegeben werden können.

Wie zuvor ausgeführt, werden die Tasks eines Knoten bei in EMB² Klasse durch prioritätsbasiertes Work Stealing verarbeitet. Dabei verwaltet jeder Kern die zur Ausführung stehenden Tasks in Abhängigkeit ihrer Priorität in einer eigenen Queue.

Zur Umsetzung des globalen EDF-Algorithmus wurde in der MTAPI-Implementierung die lokalen Queues durch eine knotenweite Task-Queue ersetzt. Diese enthält alle, zur Ausführung bereitstehenden Tasks, sortiert nach Deadline.

Durch die zentralisierte Queue kann garantiert werden, dass zu jedem Schedulingzeitpunkt der Task mit der nächsten Deadline ausgeführt wird. Damit werden die Anforderungen für GEDF erfüllt. Die Schedulingzeitpunkte finden immer dann statt, wenn ein Task abgearbeitet wurde.

Der Schedulingmodus für einen Knoten kann bei der Initialisierung festgelegt werden:

```
embb::mtapi::NodeAttribute attr;
attr.SetSchedulerMode(GLOBAL_EDF);
embb::mtapi::Node::Initialize(DOMAIN_ID, NODE_ID, attr);
```

Nach der Initialisierung der Knoten können mithilfe sogenannter Execution Policies Tasks mit definierter Deadline gestartet werden:

```
auto deadline_duration = embb::base::DurationMilliseconds(40);
embb::mtapi::ExecutionPolicy deadline(deadline_duration);
embb::mtapi::TaskAttribute deadline_attribute;
deadline_attribute.SetPolicy(deadline);
node.Start(task, arguments, results, deadline_attribute);
```

Ferner ist es möglich, mehrere, zusammengehörige Tasks mit einer gemeinsamen Deadline zu starten, wie es beispielsweise für parallele Algorithmen notwendig ist

(der durch die ForEach-Schleife parallel auszuführende Code ist im Folgenden durch eine Lambda-Funktion gegeben):

```
std::vector<int> values = {1, 2, 3, 4, 5, 6, 7, 8, 9};
embb::mtapi::ExecutionPolicy deadline(deadline_duration);
embb::algorithms::ForEach(values.begin(), values.end(),
[] (int& val) {val *= val;}, deadline_policy);
```

Zusammenfassung

Aufgrund funktionaler wie nicht-funktionaler Anforderungen müssen moderne Echtzeitsysteme die parallele Ausführung von Anwendungen mit verschiedenen Ansprüchen an Echtzeitfähigkeit und funktionale Sicherheit ermöglichen. Mithilfe der vorgestellten Echtzeitfähigkeitsklassen lassen sich solche Systeme besser beschreiben und umsetzen. Zudem wurde eine Referenzimplementierung für das Scheduling von Echtzeitanwendungen nach dem Global-EDF-Verfahren in EMB² vorgestellt. In weiteren Schritten soll das von uns entwickelte Konzept hinsichtlich bekannter Echtzeitmetriken untersucht, sowie für eine Automotive Plattform portiert werden.

Literaturverzeichnis

[1] T. Schüle. „EMB² = Parallel + Heterogen – Parallele Programmierung von Systems-on-a-Chip“. ESE Kongress, 2015.

[2]: T. Schüle. „Embedded Multicore Building Blocks – Parallel Programming made Easy“. Embedded World, 2015.

[3] Robert D. Blumofe, Charles E. Leiserson. „Scheduling multithreaded computations by work stealing“. Journal of the ACM, Band 46, Nr. 5, 1999.

[4]: N. Guan, W. Yi, Z. Gu, Q. Deng, G. Yu. „New Schedulability Test Conditions for Non-preemptive Scheduling on Multiprocessor Platforms“. Real-Time Systems Symposium, 2008.

Autoren

Tobias Langer (M.Sc.) promoviert im Bereich Echtzeitscheduling für Multi- und Manycore-Systeme im Rahmen des FORMUS³IC-Projektes am Laboratory for Safe and Secure Systems (LaS³) der OTH Regensburg. Er beschäftigt sich dabei insbesondere mit Skalierbarkeit von Schedulingverfahren auf parallelen Systemen, sowie Parallelität auf Task-Ebene.

E-Mail: tobias.langer@oth-regensburg.de

Lukas Osinski (M.Sc.) promoviert im Bereich Funktionale Sicherheit für Multi- und Manycore-Systeme im Rahmen des FORMUS³IC-Projektes am Laboratory for Safe and Secure Systems (LaS³) der OTH Regensburg. Er beschäftigt sich dabei insbesondere mit Software-Verfahren für fehlertolerante Systeme.

.

E-Mail: lukas.osinski@oth-regensburg.de

Prof. Dr. Jürgen Mottok ist Projektleiter und lehrt Informatik an der OTH Regensburg. Seine Lehrgebiete sind Software Engineering, Programmiersprachen, Echtzeitsysteme und Functional Safety. Er leitet wissenschaftlich das Laboratory for Safe and Secure Systems (LaS³, http://www.las3.de) in Regensburg, ist Beirat des Bavarian Cluster of IT-Security and Safety, Beirat des Automotive Forum Sicherheit Software Systeme, Beirat des ASQF Safety, Mitglied des Leitungsgremiums der Regionalgruppe Ostbayern der Gesellschaft für Informatik, Organisator des Fachdidaktik-Arbeitskreises Software Engineering der Bayerischen Hochschulen, Vorsitzender des Lehren von Software Engineering e.V. (LeSE e.V.) und Projektleiter der mit kooperativen Promotionsverfahren ausgestatteten Forschungsprojekte VitaS³, S³OP, S³EMO, AMALTHEA, AMALTHEA4public, FORMUS³IC, ZeloS³, FraLa, S³CORE und EVELIN. Prof. Dr. Jürgen Mottok ist in Programmkomitees zahlreicher wissenschaftlicher Konferenzen vertreten.
Er ist Träger des Preises für herausragende Lehre, der vom Bayerischen Staatsministerium für Wissenschaft, Forschung und Kunst im Jahr 2010 vergeben wurde. Am 4.12.2015 wurde Prof. Dr. Jürgen Mottok der „Preis für besondere Leistungen bei der Zusammenarbeit zwischen Wirtschaft und Wissenschaft“ verliehen. Prof. Mottok war an den erfolgreichen Ausgründungen der LaS³-Spin-offs Timing Architects und iNTENCE beteiligt

E-Mail: juergen.mottok@oth-regensburg.de

Dr. Tobias Schüle ist Senior Key Expert Engineer, zertifizierter Software-Architekt und Projektleiter bei Siemens Corporate Technology, der zentralen Forschung und Entwicklung von Siemens. Sein Hauptinteresse gilt neben der parallelen Programmierung der Architektur und Entwicklung von eingebetteten Systemen. Er ist Autor zahlreicher Veröffentlichungen und Coautor des Buchs „Multicore-Software“

.

E-Mail: tobias.schuele@siemens.com

Ralph Mader ist Principal Technical Expert für "Powertrain Software Architecture Embedded Systems" bei der Continental AG. Seit 2006 arbeitet Ralph Mader im Bereich Entwurf und Auswahl von Mikrocontrollern im Bereich Powertrain, sowie deren effizienten Einsatz. Weiterhin leitet er seit 2010 die Entwicklung von Multicore Software Architekturen für Motorsteuerungssysteme. Dabei vertritt er Continental in einschlägigen Forschungsprojekten wie S³Core, FORMUS³IC und Proxima

E-Mail: ralph.mader@continental-corporation.com