Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

Segmentierung parallelisierter Algorithmen im Mixed-Criticality-System

Dipl. Inform. (FH) Marc Bommert
Hochschule RheinMain, Wiesbaden
marc.bommert@hs-rm.de

Herbstreffen der GI-Fachgruppe Betriebssysteme 07. und 08. November 2013

Struktur und Inhalt

1) Kontext der Arbeit
2) OpenMP
3) Mixed Criticality Umgebung
4) OpenMP im MC-Stack
5) Zusammenfassung & Ausblick

Kontext der Arbeit

- Aktuell: Forschungsprojekt AUTOBEST
  - Um Prof. Dr. Robert Kaiser
  - Zielsetzung: „Neuartige“ Betriebssystemkonzepte für automobile Anwendungen
    - Wesentlich: Übertragung von Konzepten aus der Avionik (ARINC-653) auf automotive Plattformen.
  - BMWI-gefördert, Programm ZIM-Koop, 2 Jahre
  - Projektpartner easycore GmbH, Erlangen
  - Präsentiertes Thema ist Anteil d. Forschung (und wird aktuell bearbeitet)

ZIM Zentrales Innovationsprogramm Mittelstand

easycore

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

OpenMP

- Framework zur Parallelprogrammierung
  - Kommunikation über gemeinsamen Speicher
    - Im Kontrast zu MPI: Message Passing Interface (Many Core, NoCs, asymmetrische Multicore-Systeme)
  - Für symmetrische Multicore-Plattformen
  - Methode: Inkrementelle Parallelisierung mittels Präprozessor-Pragmas
  - Weitestgehend transparente Details der Parallelisierung
    - Thread-Erzeugung
    - Synchronisation der Kontrollflüsse
    - Serialisierung des konkurrenten Datenzugriffes

OpenMP

- Ein wesentliches Konstrukt: Parallele for-loops

Code-Beispiel:

```
#pragma omp parallel for schedule (static)
#define N 1000000
for (int i = 0; i < N; i++) {
      result[i] = processData(&data[i]);
}
```

- Annahmen:
  - Unabhängige Schleifeneinzelzyklen
  - „Lange“ Laufzeiten (große Werte von N)
  - Weitestgehend homogene Laufzeiten der Einzelzyklen
- Mögliche Anwendungen: Suchen, Sortieren, Traversieren. Wesentliches Einsatzgebiet: Digitale Bildverarbeitung.

OpenMP

Präprozessoranweisung kann ignoriert werden (Kompilation für Single-Threaded-Betrieb)

```
#pragma omp parallel for schedule (static)
#define N 1000000
for (int i = 0; i < N; i++) {
      result[i] = processData(&data[i]);
}
```

OpenMP

Erzeugung von Worker-Threads bei Detektion des Schlüsselwortes „parallel“

```
#pragma omp parallel for schedule (static)
#define N 1000000
for (int i = 0; i < N; i++) {
      result[i] = processData(&data[i]);
}
```

OpenMP

Nachfolgende for-Schleife ist **statisch** zu parallelisieren

```
#pragma omp parallel for schedule (static)
#define N 1000000
for (int i = 0; i < N; i++) {
      result[i] = processData(&data[i]);
}
```

OpenMP

Kein konkurrenter Datenzugriff, kein Locking

```
#pragma omp parallel for schedule (static)
#define N 1000000
for (int i = 0; i < N; i++) {
      result[i] = processData(&data[i]);
}
```

Implizite Barriere: Join der Worker-Threads und anschließende Fortsetzung des sequentiellen Programmteils

OpenMP

- Meist ein worker je beteiligtem Prozessor
- Folgen der Thread-Synchronisation (Join) nach Abschluss der Parallelberechnung: Inserted Idle Times (IIT), für jene Worker, die „früher“ fertig sind.

  $\Rightarrow$Effizienzeinbuße

  $\Rightarrow$Immer! Weil Fertigstellungszeitpunkte der Teilaufgaben niemals exakt gleich sind.

OpenMP

- „Scheduling“-Methoden von OpenMP
  - zur Auf- und Zuteilung (Segmentierung) der Menge aller Schleifeniterationen zu den worker threads
- Static: Vollstatische Zuteilung
  - Minimaler Overhead, maximale IIT
- Dynamic: Volldynamische Zuteilung
  - Maximaler Overhead, minimale IIT
- Guided
  - Mischung zwischen beiden Methoden (Dynamisches Verfahren mit sukzessive abnehmender chunk size).

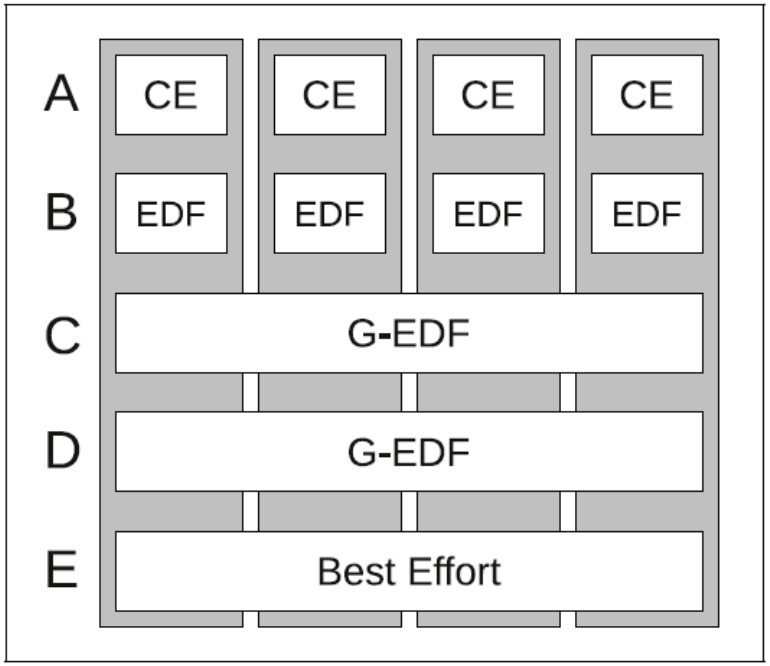
OpenMP

- OpenMP ist für Echtzeitanwendungen eher wenig beachtet/benutzt. Warum?
  - Zuteilung chunks->worker aus globaler Queue
  - Zuteilung worker->cores aus globaler Queue
  - Grundsätzlich möglich, aber o.g. Komplexität des Frameworks steht dem im Wege
  - Reduzierung der Komplexität könnte die Eignung für den Einsatz für Echtzeitsysteme verbessern
  - Auch: Evtl. Hardware-seitige Beschränkungen (Bus-Bandweite, Anbindung des Speichers als Bottleneck)

Mixed Criticality Umgebung

- Zweck: Konsolidierung von Applikation unterschiedlicher Fehlerschwere auf einer (Mehrprozessor-)Plattform
  - Hierarchie der Kritikalitäten: Hochkritische Tasks geben veranschlagte, unverbrauchte Rechenzeit an unterlagerte Schichten ab
  - Rückwirkungsfreier gemeinsamer Betrieb: Keine „Infektion“ der weniger kritischen Abläufe durch den ungleich höheren Verifikationsaufwand höherkritischer Abläufe.

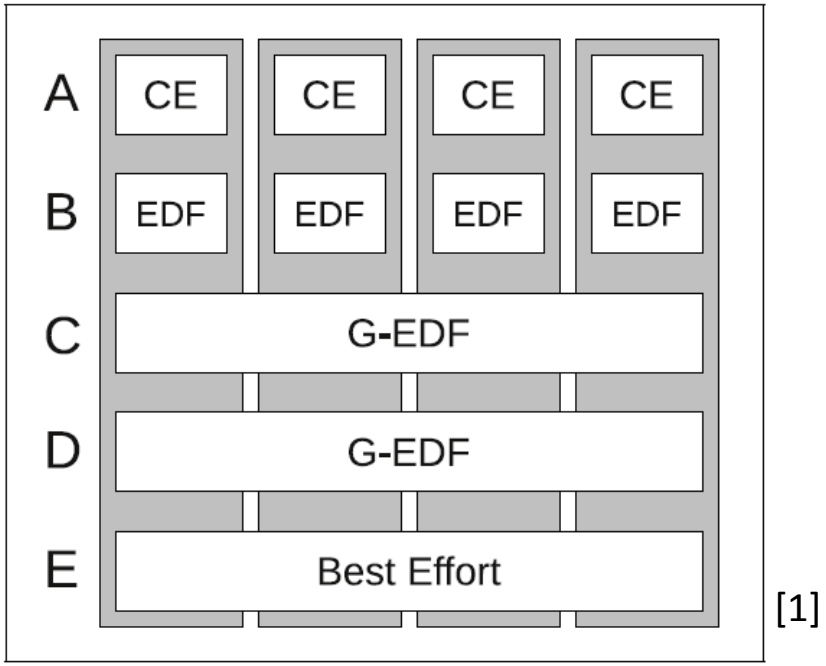
Mixed Criticality Umgebung



[1]

[1] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, and J. A. Scoredos, “Mixed-criticality real-time scheduling for multicore systems,” in Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology, ser. CIT ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1864–1871

Mixed Criticality Umgebung

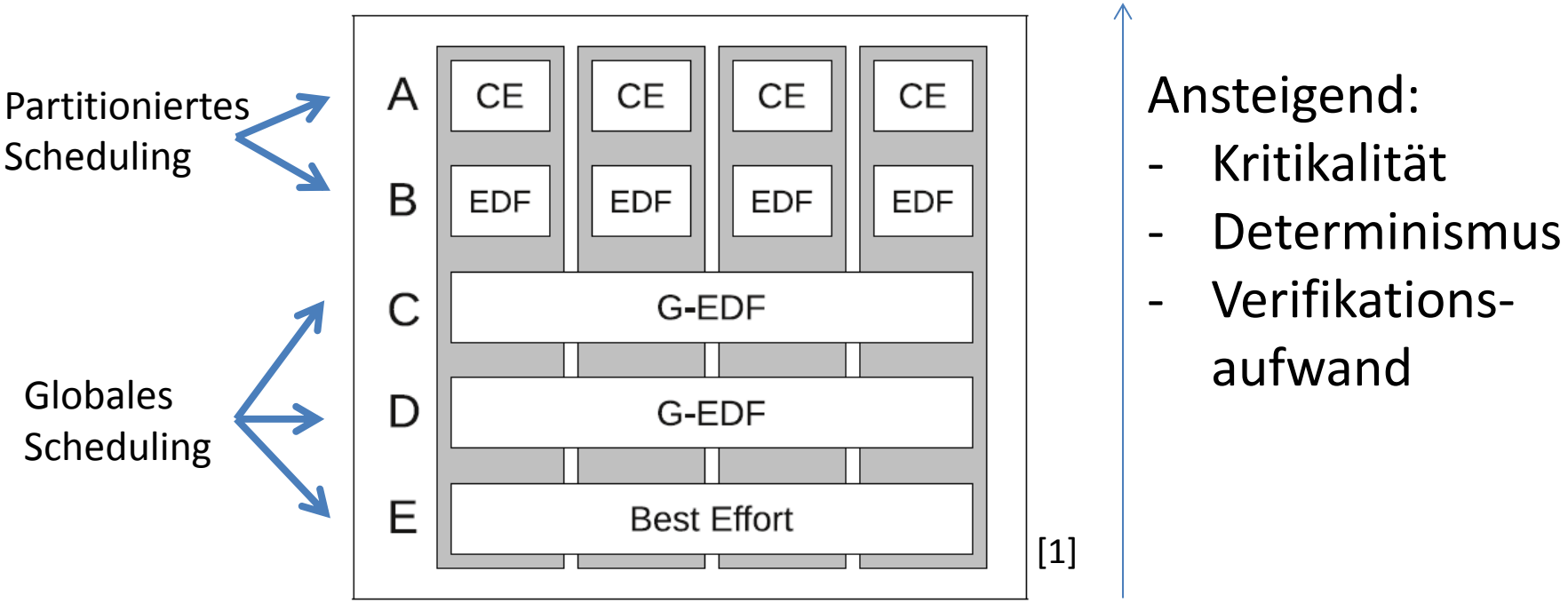


[1]

Ansteigend:
- Kritikalität
- Determinismus
- Verifikations-aufwand

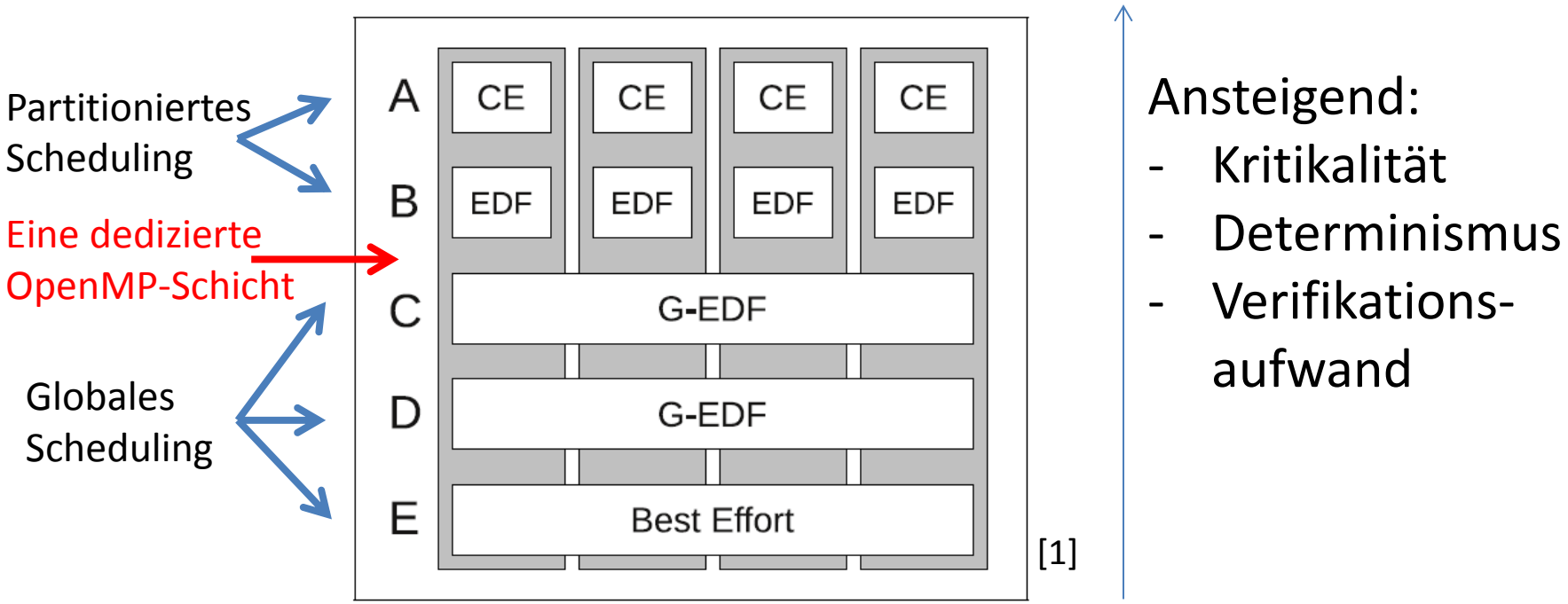
[1] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, and J. A. Scoredos, “Mixed-criticality real-time scheduling for multicore systems,” in Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology, ser. CIT ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1864–1871

Mixed Criticality Umgebung



[1] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, and J. A. Scoredos, “Mixed-criticality real-time scheduling for multicore systems,” in Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology, ser. CIT ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1864–1871

OpenMP im MC-Stack



[1] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, and J. A. Scoredos, “Mixed-criticality real-time scheduling for multicore systems,” in Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology, ser. CIT ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1864–1871

OpenMP im MC-Stack

- Warum denn genau dort?
  - Wir benötigen mehr als einen Prozessor
    - D.h. wir müssen unterhalb der partitionierten Schichten A und B angeordnet sein.
  - Wir wollen nicht auf der Restrechenzeit eines G-EDF planen müssen.
    - Hohe Dynamik durch EDF
    - Aber auf Schicht B wird auch schon P-EDF verwandt?
      - Korrekt, ist aber in partitionierter Form „Kontrollierbarer“
      - Oftmals wird in der Praxis stattdessen RMS angewandt
        - Leichter implementierbar (FP), Determinismus im Fehlerfall.

OpenMP im MC-Stack

- Problemstellung beim Betrieb von OpenMP im MC-Stack:
  - Es verbleiben ungleiche und sehr variable Restrechenkapazitäten zwischen den Prozessoren auf der OpenMP-Parallelisierungsschicht.
  - Dies führt u.U. (statische Segmentierung) zu enorm großen IITs bis hin zum Verhungern parallelisierter Task-Segmente.
  - Schlechte Effizienz parallelisierten Codes.
  - Grund: Das Framework berücksichtigt nicht, auf unterlagerter Schicht betrieben zu werden. Es ist naiv.

OpenMP im MC-Stack

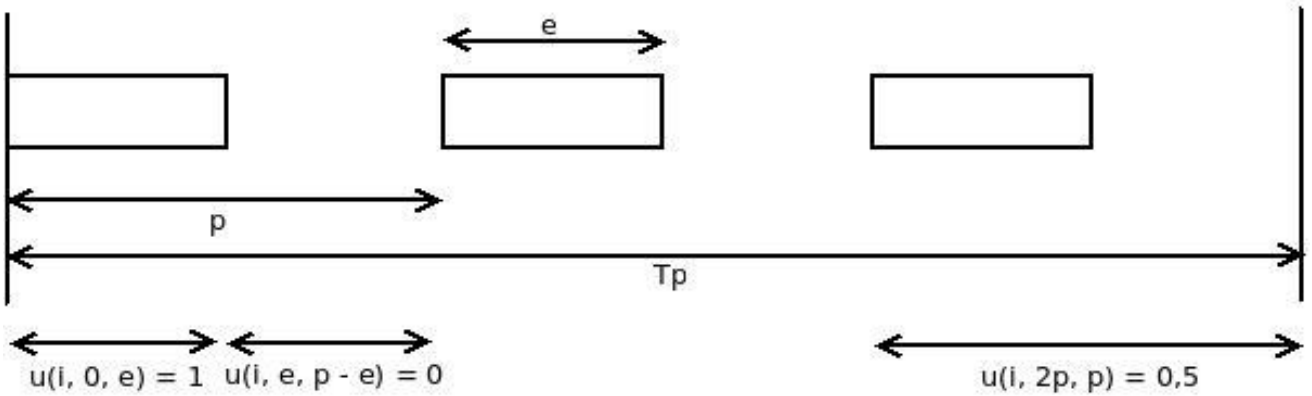
- Lösung: Hinzufügen von Intelligenz zur OMP-Implementierung.
  - Durch angepasstes OpenMP „Scheduling“ (Segmentierung der Gesamtanzahl an Schleifendurchläufen).
  - Unter Berücksichtigung statischer Planungsparameter der höherkritischen Schichten
    - Task-Parameter auf den höchsten Schichten sind üblicherweise bestens bekannt
  - Ziel: Effizienz erhöhen, Echtzeitfähigkeit der parallelisierten Tasks

OpenMP im MC-Stack

- Utilization functions
  - Werden statisch zur Integrationszeit des MC-Systems definiert.
  - Liefern einen normalisierten Wert, der über die Auslastung des betr. Prozessors im angegebenen Zeitraum informiert.
  - Zur Bekanntmachung des erwarteten Verhaltens partitionierter Tasksets
    - Wir definieren zwei Utilization-Funktionen: Eine Worst-case utilization u_{WCET} und eine best-case utilization u_{BCET}
  - Wir nehmen eine voll-periodische Planung auf höchster Schicht an, und eine gewisse Harmonie
    - Wesentliche Annahme: Es kann eine Hyperperiode T_p (kgV aller Task-Perioden) bestimmt werden
    - Diese Hyperperidoe T_p des Gesamtsystems ist die Periode der Utilization functions
    - Nicht alle Konstellationen von Task-Sets sind handhabbar

OpenMP im MC-Stack

- Utilization functions: Beispiel
  - Ein partitioniertes Taskset i: Eine Task (2e=p)
    - e = Periodendauer der Task, e = WCET der Task



OpenMP im MC-Stack

- Distribution functions
  - Steuern die Segmentierung der Gesamtzahl der Schleifendurchläufe
  - Werden einmalig bei Eintritt in die Parallelsequenz konsultiert
  - Nutzen die Utilization functions
  - Wir definieren zwei Verteilungsfunktionen
    - Static weighted distribution
    - Hybrid distribution

OpenMP im MC-Stack

- Static Weighted distribution
  - Proportionale Verteilung der Gesamtzahl Schleifendurchläufe auf die Prozessoren gemäß des Anteils der Rechenzeit pro Prozessor pro Hyper-Periode:

$$z(i, N, m) = \frac{1 - u_{WCET(i,0,T_p)}}{\sum_{k=1}^{m}(1 - u_{WCET(k,0,T_p)})} * \mathrm{N}$$

- Vorteile
  - Minimaler Overhead für die Parallelisierung
  - Höchster Determinismus / Minimale Varianz der Worker-Laufzeit & gute Analysierbarkeit des Zeitverhaltens, da keinerlei Synchronisation im OpenMP-Framework
- Nachteile
  - Im Mittel hohe IIT, da die Einzellaufzeiten der hochkritischen Tasks in den seltensten Fällen der WCET entsprechen.

OpenMP im MC-Stack

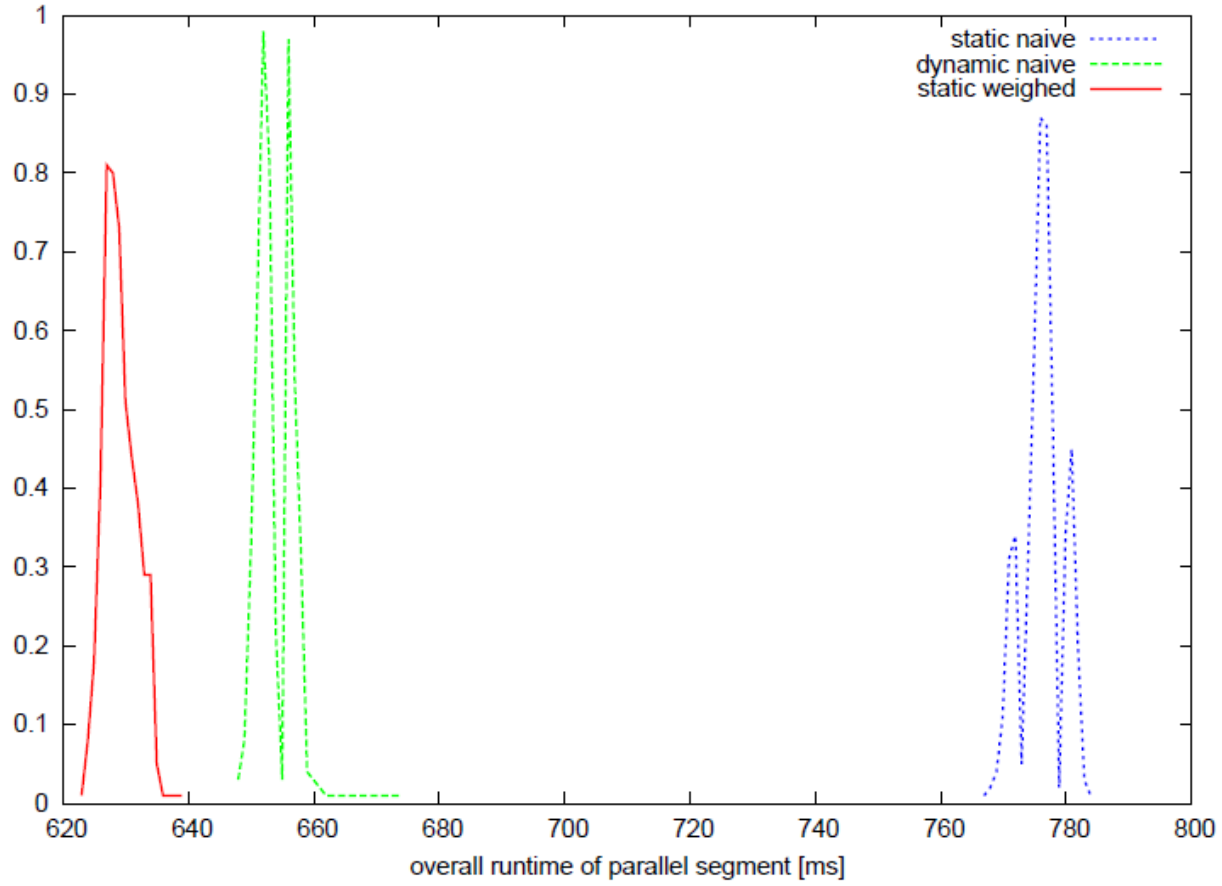
- Hybrid distribution
  - Teildynamische Verteilung einer Untermenge der Schleifendurchläufe nachdem der statische Anteil verarbeitet wurde.
  - Trade-Off: IIT wird reduziert (backfilling) zu Lasten eines erhöhten Overhead
  - Proportionen zwischen statisch verteilter Anzahl Schleifendurchläufe N_s zur dynamisch verteilten Anzahl Schleifendurchläufe N_d werden bestimmt als:

$$N_s = N - N_d = \mathrm{N} * \frac{\sum_{i=1}^{m}(1 - u_{WCET(i,0,T_p)})}{\sum_{i=1}^{m}(1 - u_{BCET(i,0,T_p)})}$$

- Vorteile
  - Geringere IIT als der vollstatische Ansatz
- Nachteile
  - Overhead ist gegenüber dem vollstatischen Ansatz erhöht
  - Work-stealing: Determinismus vermindert / Höhere Varianz der Worker-Laufzeiten durch Locking, erschwerte Analysierbarkeit des Zeitverhaltens wegen Synchronisation im OpenMP-Framework.

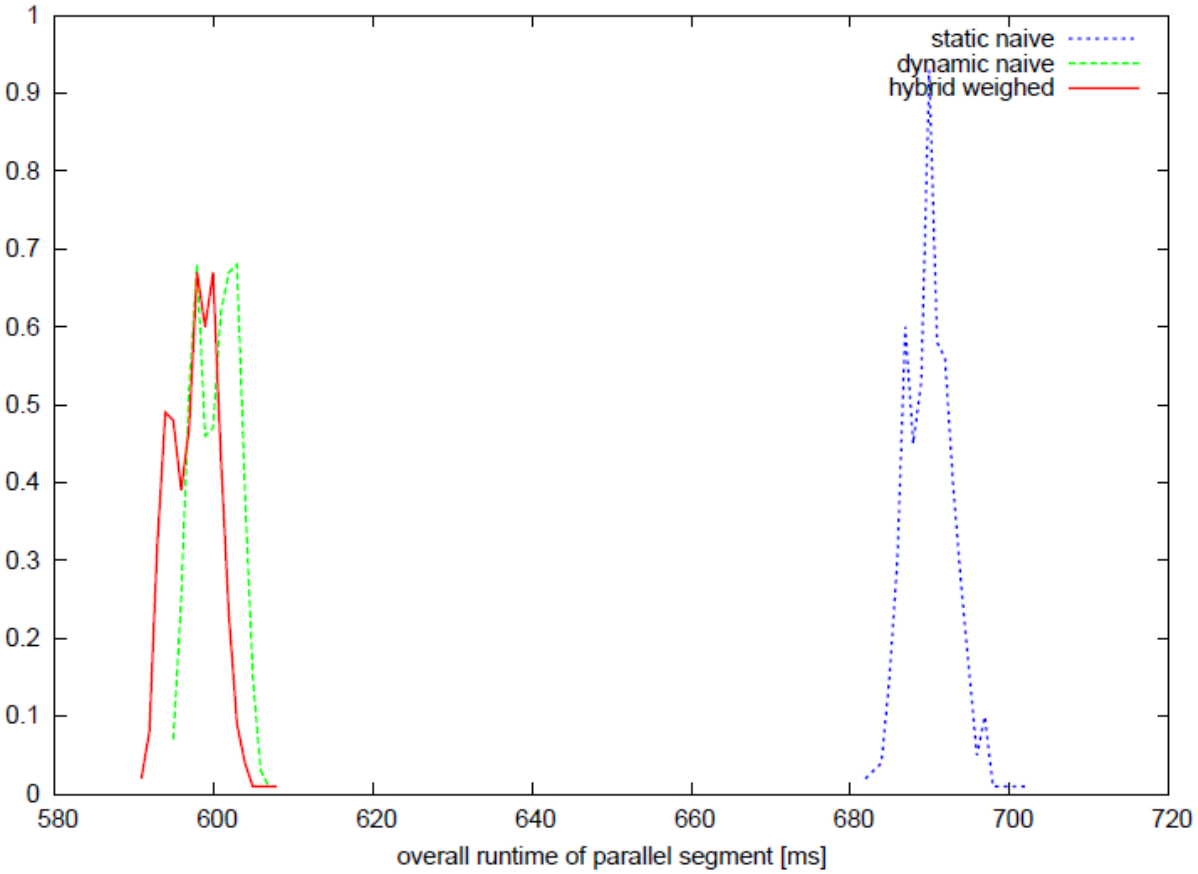
OpenMP im MC-Stack

- Qualitative Simulation / Bewertung



OpenMP im MC-Stack

- Qualitative Simulation / Bewertung



OpenMP im MC-Stack

- Echtzeitfähigkeit des Ansatzes: Ausblick
  - Wesentlich: Pinning der worker auf einzelne Cores
    - Reduzierung des „Schedulings“ auf die Auf- und Zuteilung von Schleifenzyklen zu worker-Threads
  - Busbandweiten sind Problem
    - Annahme derzeit : Busbandweiten jederzeit mehr als ausreichend
  - Locking (konkurrenter Datenzugriff) im Parallelsegment nicht berücksichtigt
    - Wird Bestandteil weiterer Forschungsarbeit

Zusammenfassung

- Es wurde beschrieben wie eine Implementierung des OpenMP-Framework in einer Mixed-Criticality-Umgebung möglichst effizient umgesetzt werden kann.
- Die Umsetzung basiert auf Verteilungsfunktionen, die sich statische Utilization-Information aus partitionierten hochkritischen Tasksets zu Nutze machen.

Zusammenfassung

- Es wurde dargelegt, dass der Ansatz (festes Mapping zwischen Worker-Threads und Prozessoren) eine Komplexitätsreduktion bewirkt, die die Eignung für den Echtzeiteinsatz potentiell verbessert.
- Die grundsätzliche Eignung der Methode (Effizienzsteigerung) wurde am konstruierten Beispiel durch Simulation gezeigt.

Ausblick

- Das vorgestellte Thema wird aktuell bearbeitet und prototypisch implementiert.
  - Anschließend kann eine Bewertung erfolgen.
  - Definition von Schnittstellen zur Integrationsdomäne sind nötig (Utilization Functions aus Task-Parametern)
- Danach ist die weitere Betrachtung möglich:
  - Kürzere Laufzeit der Parallelsequenzen (kleine N)
  - Konkurrenter Datenzugriff in der Parallelsequenz
  - Anwendung auf übrige Parallelisierungskonstrukte
  - …

Vielen Dank für Ihre Aufmerksamkeit.

Fragen? Diskussionsbedarf?

Person, Kontext & Thema