Ein Fork-Join-Parallelismus im Mixed-Criticality-Umfeld

Dipl.-Inform. (FH) Marc Bommert

Workshop „Echtzeit 2015“

13.11.2015, Boppard am Rhein

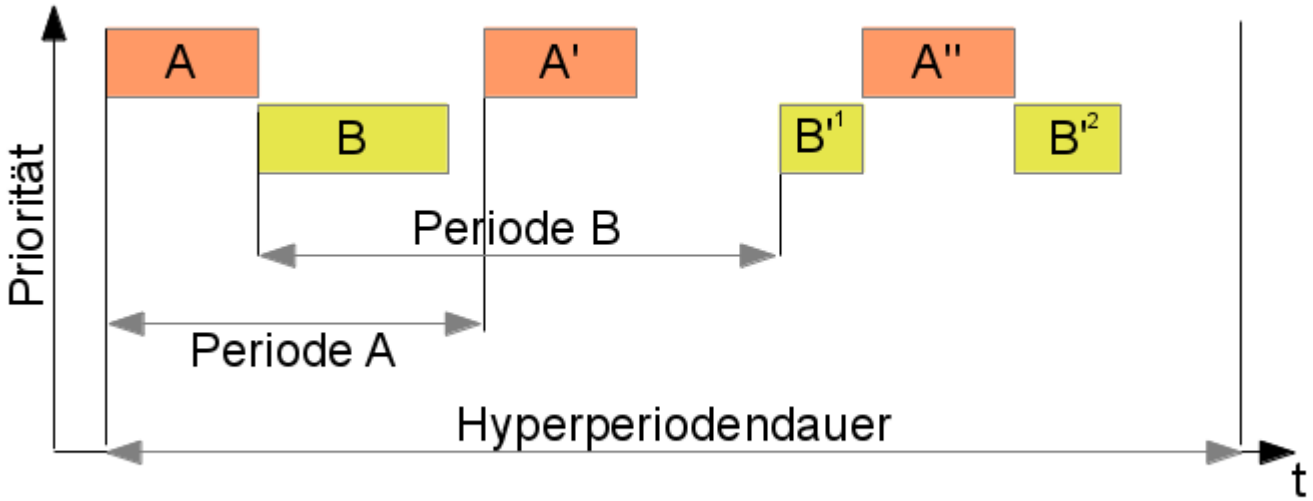
NOSCo – New Operating System Concepts
Hochschule RheinMain University of Applied Sciences
Universität Koblenz-Landau

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

Outline

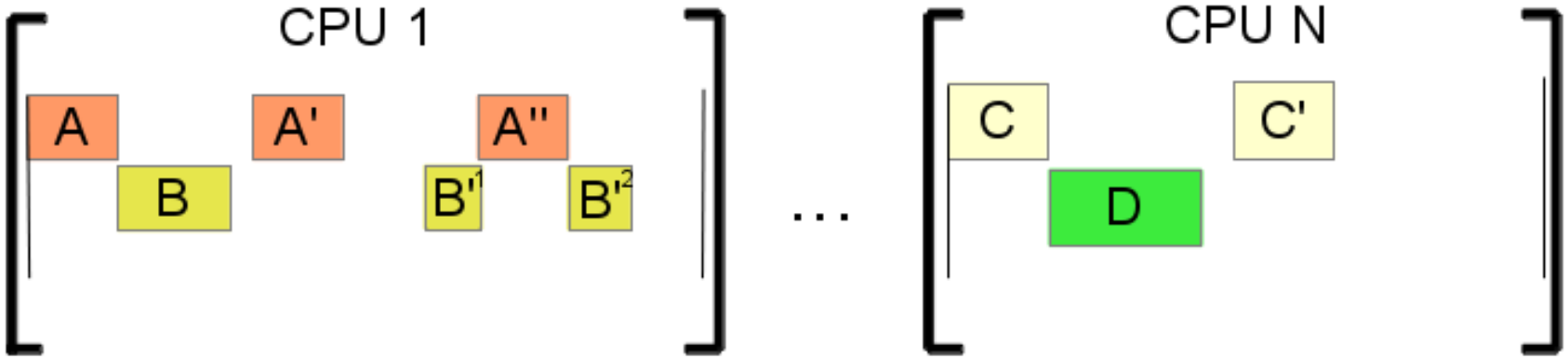
- Modell: Scheduling-Hierarchie
- Problemdefinition
- Untergeordneter Fork-Join-Parallelismus
  - Segmentierbarkeit
  - Einplanbarkeit
- Essenz: Abbildung der *Auslastungsfunktion*
- Evaluation
- Conclusio und Ausblick

Ein Real-Time Task-Set



- Ablaufplanung ist durch folgende Task-Parameter bestimmt:
  - Periode
  - Deadline (entspricht analytisch ermittelter WCET, Sicherheitsspielraum)
- Konservativer Ansatz
  - Harmonisches System/Feste Gesamtperiodizität
  - Statische Offline-Planung (hier: RMS)

Mixed Criticality: Partitioniertes Scheduling im Mehrprozessorsystem

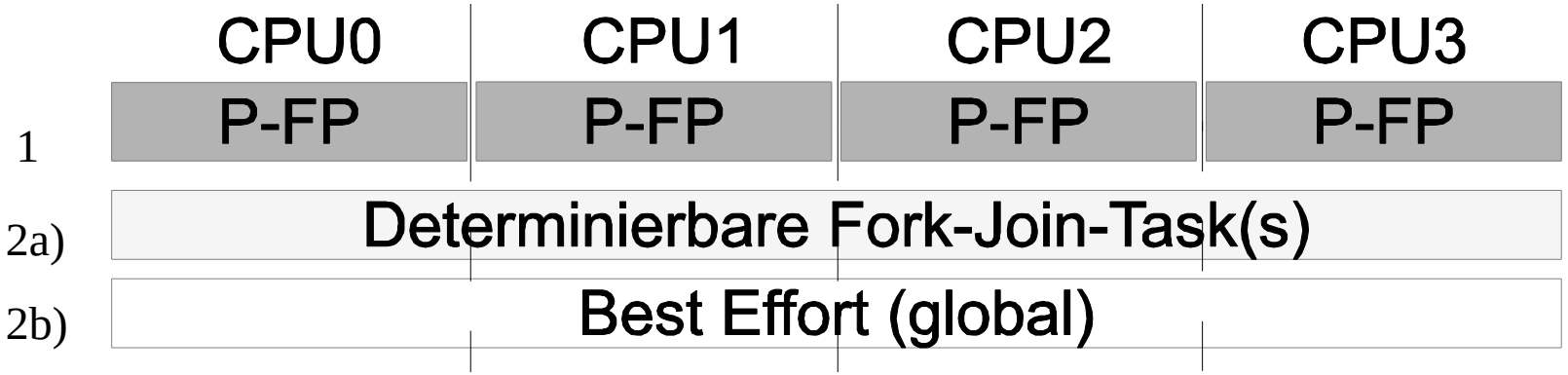


- SMP-System: Partitioniertes RT-Scheduling
- Auslastung $u < 1$
  - Berechnungskapazität nicht voll genutzt
  - Restrechenkapazität für unterlagerte Schicht „übrig“
- Modell: Einfügen einer unterlagerten dedizierten Planungsschicht für parallelisierte Berechnungen

Unterlagerter Fork-Join-Parallelismus

Hierarchie der Scheduling-Ebenen:

1) Partitioniertes Scheduling nach fixen Prioritäten (RMS)
2) Globales Scheduling in unterlagerten Ebenen
    a) Dedizierte Planungsschicht für Fork-Join-Tasks
    b) Unterste Best-Effort-Schicht: Nicht näher definiert



Unterlagerter Parallelismus: Problemdefinition

- Die nach Ausführung der RT-Tasks verbleibenden Restrechenkapazitäten sind fragmentiert.
  - Ggf. nachteilig für parallelisierte Berechnungen
  - Für den Worst-Case-Fall ist der Systemablauf bekannt
- Aber: Das parallel zu lösende Problem ist jedoch ebenfalls segmentierbar
  - Können wir für jeden möglichen Worst-Case-Systemzustand die optimale Segmentierung finden?

Per OpenMP parallelisierte Schleife: Segmentierbarkeit am Beispiel

Beispiel

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

Parallelisiert die Schleife zwischen *m* Prozessoren

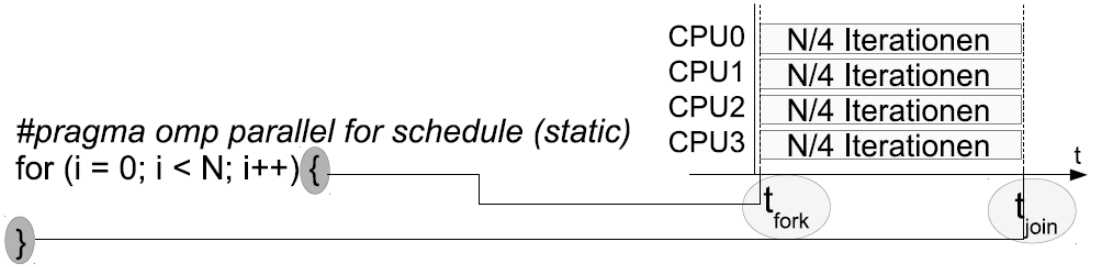
Verteilungsstrategien: *static*, *dynamic* oder *guided*

Wie sehen die resultierenden Workloads aus?

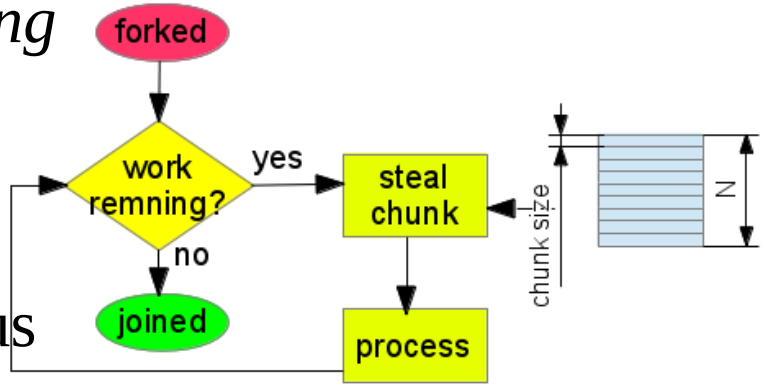
Wie ist die Einplanbarkeit dieser Workloads?

OpenMP-Segmentierungsstrategien

- *Static:* N/m Iterationen per Prozessor



- *Dynamic:* Aktives *Work-Stealing* in definierter *Chunk Size*



- *Guided:* Hybrider Mechanismus Nicht weiter betrachtet.
- *Auto*: Implementation defined

OpenMP-Segmentierungsstrategien: Kriterium der Einplanbarkeit

- Optimierungsziel: Alle *m* Prozessoren schließen die Berechnung der zugeordneten Teilaufgabe zum gleichen Zeitpunkt ab.
  - Sonst: Einfügung von Wartezyklen (Inserted-Idle-Time, IIT) für einzelne Prozessoren
  - Annahme: Parallele Schleife wird sporadisch aktiv
  - Behauptung: Parallele Berechnung durch geeignete Segmentierung optimierbar (d.h. besser einplanbar)

Vorarbeit: Statisch-gewichtete Segmentierung

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

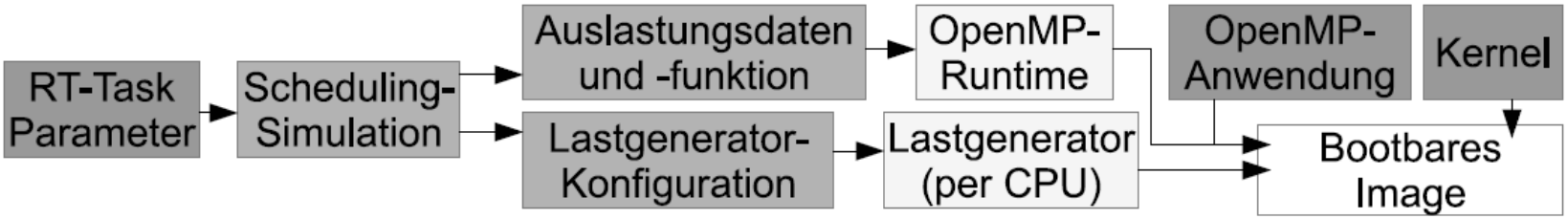
- Verteilungsfunktion z berechnet die Anzahl der Iterationen N_i für den Prozessor i [1 .. m] anhand der Gesamtiterationszahl N
- N_i geht proportional zum Verhältnis der für den jeweiligen Prozessor verbleibenden Rechenzeit zu der gesamten verbleibenden Rechenzeit
  - Bereitgestellt per Auslastungsfunktion $u_{WCET}(i, t, dt)$

Essenz: Erhebung und SW-Abbildung der Auslastungsfunktion

$$u_{WCET}(i, t, dt)$$

- Statische RT-Planungsdaten zur Laufzeit explizit zur Verfügung stellen
  - Geringe Abfragekosten erwünscht
  - Abfragbar durch Tasks auf der unterlagerten Parallelschicht
  - Abfrageparameter: Start-Offset und Slot-Dauer
    - Abfrage für beliebige Zeitbereiche innerhalb der Hyperperiode

Technische Umsetzung: Blockdiagramm



- Vollständige Eigenentwicklung
  - Generierung von Code und Konfiguration aus der Simulation heraus
  - Statische Bindung aller Komponenten
- Lastgenerator auf Basis von Kernel-Anpassung
  - Exaktere Kontrolle der Task-Aktivzeiten (WCET)
- Minimale OpenMP-Runtime (Subset/parallel-for)

Technische Umsetzung: Auslastungsfunktion

- Datenmodell
  - Linearfolge von Timestamp-Zweiertupeln
    - sortiert, per CPU
    - Struktur ($t_{OCCUPIED}$, t_{IDLE}), ($t_{OCCUPIED}$, t_{IDLE}), …
  - Logisch: Binärer Suchbaum (divide et impera)
- Abfragefunktionalität
  - Suche des Startzeitpunktes auf dem Feld mit logarithmischem Aufwand
  - Akkumulation der Auslastung mit linearem Aufwand
  - Maßgeblich ist die Anzahl der Scheduling-Ereignisse innerhalb des Abfragebereiches, nicht der Dauer.

Evaluation

- Mikrobenchmarks

Anzahl Tupel	Bereich der Abfrage	Dauer [TSC Ticks]([ns])		
		Minimum	Maximum	Mittel
512	0 .. (HP - 1)	15573(5001)	17720(5690)	15737(5053)
1024	0 .. (HP - 1)	31372(10074)	33839(10867)	31785(10207)
2048	0 .. (HP - 1)	62371(20030)	64516(20718)	62741(20148)
4096	0 .. (HP - 1)	124729(40055)	126924(40760)	125150(40191)

Evaluation

- Mikrobenchmarks

Anzahl Tupel	**Bereich der Abfrage**	**Dauer [TSC Ticks]([ns])**		
		Minimum	Maximum	Mittel
512	0 .. (HP - 1)	15573(5001)	17720(5690)	15737(5053)
1024	0 .. (HP - 1)	31372(10074)	33839(10867)	31785(10207)
2048	0 .. (HP - 1)	62371(20030)	64516(20718)	62741(20148)
4096	0 .. (HP - 1)	124729(40055)	126924(40760)	125150(40191)
512	(HP - 2) .. (HP - 1)	390(125)	810(260)	416(133)
1024	(HP - 2) .. (HP - 1)	445(142)	855(274)	478(153)
2048	(HP - 2) .. (HP - 1)	478(153)	979(314)	500(160)
4096	(HP - 2) .. (HP - 1)	566(181)	1189(381)	594(190)

Evaluation

- Mikrobenchmarks

Anzahl Tupel	**Bereich der Abfrage**	**Dauer [TSC Ticks]([ns])**		
		Minimum	Maximum	Mittel
512	0 .. (HP - 1)	15573(5001)	17720(5690)	15737(5053)
1024	0 .. (HP - 1)	31372(10074)	33839(10867)	31785(10207)
2048	0 .. (HP - 1)	62371(20030)	64516(20718)	62741(20148)
4096	0 .. (HP - 1)	124729(40055)	126924(40760)	125150(40191)
512	(HP - 2) .. (HP - 1)	390(125)	810(260)	416(133)
1024	(HP - 2) .. (HP - 1)	445(142)	855(274)	478(153)
2048	(HP - 2) .. (HP - 1)	478(153)	979(314)	500(160)
4096	(HP - 2) .. (HP - 1)	566(181)	1189(381)	594(190)
4096	0 .. (10% * HP)	12558(4032)	14617(4694)	12720(4084)
4096	0 .. (20% * HP)	24947(8011)	27063(8691)	25129(8070)
4096	0 .. (30% * HP)	37450(12026)	39516(12690)	37695(12105)
4096	0 .. (40% * HP)	49848(16008)	52190(16760)	50183(16115)

Evaluation 2

- Benchmarking der statischen gewichteten Verteilung, N = 800, M = 8, HP = 10ms

Ausführung	Last	Laufzeit$[ns]$	Speedup	$\delta t_{start}[ns]$	$\delta t_{stop}[ns]$	overhead$[ns]$
sequentiell	keine	112614666	-	-	-	-
par. statisch	keine	19941334	464%	22200	44119	641
par. statisch	Task-Sets	24583165	358%	26147	4790057	232
par. gewichtet	Task-Sets	23459024	380%	25656	3458662	1218

Abschätzung

- Verbesserung des Speedup trotz erhöhtem Overhead
- Lasten: CPU0 (u = 0,075), CPU1 … CPU7 (u = 0,05)
  - Minimale Differenz: Sehr geringes Optimierungspotential
  - Dennoch messbarer Kosten-Nutzen-Vorteil feststellbar

Conclusio

- Die Bereitstellung statischer Auslastungsdaten
  - Für eine dedizierte Fork-Join-Planungsschicht
    - Die einer partitionierten Planungsschicht (P-FP) hierarchisch untergeordnet ist (Kritikalitätshierarchie)
  - Gewonnen aus einer Offline-Simulation
    - Auf Basis von (z.B. analytisch) ermittelten Task-Parametern
  - Ermöglicht die verbesserte Segmentierung von parallelisierten Algorithmen auf unterlagerter Schicht
    - Vereinfachte Determinierbarkeit
    - Effizienzverbesserung
  - Strenge Orientierung am Worst-Case-Szenario
    - Vorrangiges Ziel: Determinismus (& vereinfachte Determinierbarkeit per Simplifizierung)

Ausblick

- Systematische, quantitative Evaluierung
  - Derzeit nur exemplarisch & rein qualitativ
- Erweiterbarkeit des grundlegenden Ansatzes
  - Weitere Verwendungen der zur Laufzeit verfügbaren statischen Auslastungsinformation
- Analyse übriger OpenMP-Konstrukte
- Definition des mathematischen Modells, kontinuierliche Überprüfung
  - Scheduling-Overhead bisher unberücksichtigt

return 0;

Vielen Dank für Ihre Aufmerksamkeit

Theoretische Vorarbeit:

Bommert, Marc:
Schedule-aware Distribution of Parallel Load in a Mixed Criticality Environment.
In: Proceedings of the 7th Junior Researcher Workshop on Real-Time Computing (JRWRTC '13) (21st International Conference on Real-Time and Network Systems (RTNS '13) Sophia Antipolis, France), 2013