ARM big.LITTLE Processing

Energieeffiziente asymetrische Multicore

Sebastian Humenda (shumenda at gmx punkt de)

30.06.2013

Lizenz

Inhaltsverzeichnis

1 Einführung und Definitionen

Immer höhere Anforderungen an mobile Plattformen, aber auch steigende Energiepreise machen es nötig, dass Rechnersysteme zunehmend energieeffizient arbeiten. Laut [7] sind große Firmen bereits dazu übergegangen für ihre Rechenzentren Maschinen zu erwerben, die die beste Leistung relativ zum Energieverbrauch haben und nicht absolut die beste Leistung. Besonders der expandierende Markt der Smartphones stellt ganz neue Anforderungen an Energie und Leistung [11, S. 1f.]. Auf mobilen Plattformen sollen hochauflösende Videos wiedergegeben werden, Videotelefonie zur Verfügung stehen, graphikaufwendige Spiele gespielt werden können und vieles mehr, was insgesamt hohe Leistungsanforderungen an die Geräte sind. Nach [12, S. 1 f.] sind Spiele und Browsing mit die stärksten Akku beanspruchenden Anwendungen, wohingegen SMS verfassen o. ä. wesentlich weniger Leistung benötigt, aber üblicherweise mit dem gleichen Prozessor ausgeführt wird. Weiterhin wird ebd. davon gesprochen, dass LCDs immer mehr Strom verbrauchen, weshalb die Lebenszeit von Akkus um einen weiteren Faktor verkürzt wird. Deswegen erhält Energieoptimierung bei der Entwicklung neuer Prozessoren und eingebetteter Systeme hohe Priorität.

Ein traditioneller Ansatz zur Leistungsoptimierung war anfangs vor allem die Taktratensteigerung. Später begann man weitere Kerne (zur parallelen Ausführung von mehreren Aufgaben gleichzeitig) in einen Prozessor zu integrieren.

Beim Hinzufügen von (gleichen) Kernen zu einem Prozessor erhalten wir homogene Mehrkernprozessoren mit einem gemeinsamen Befehlssatz[1]. Zwei Kerne bedeuten nicht nur (theoretisch) doppelte Leistung, sondern auch verdoppelte Leistungsaufnahme. Dynamische Taktratenanpassung, zum Ausgleich des hohen Energieverbrauchs,[2]ist bei weitem nicht das Maximum der Optimierung der Energieeffizienz, bestätigt auch [13].

Asymmetrische Mehrkernprozessoren sind nun die nächste Stufe in der Optimierung der Energieeffizienz. Bei diesen Prozessoren unterscheidet sich neben der Taktrate auch die Mikroarchitektur, wie z. B. die Pipelinelänge oder eine sequenzielle Befehlsreihenfolge gegenüber OOO-Ausführung (out-of-order). Man unterscheidet AMP-Systeme u. a. danach, ob sie den gleichen oder verschiedene Befehlssätze verwenden.

Eine Möglichkeit zur Ausführung von fließkommaintensive Rechnungen ist die Verlagerung dieser Operationen auf die GPU, der Rest des Programmes läuft weiterhin auf der CPU. GPU und CPU verwenden allerdings verschiedene Befehlssätze, was die Komplexität der Software erhöht. Hingegen kann man auch zwei Kerne in einen Prozessor integrieren, die den gleichen Befehlssatz haben und bei denen sich lediglich die Mikroarchitektur unterscheidet. Man kann sich das zum Beispiel anhand mehrerer Generationen von x86-CPUs verdeutlichen. Von Pentium Pro bis Pentium 4 wuchs

[1] Im engl. als single-ISA homogenous multicore

[2] Bei dynamischen Taktraten in homogenen Mehrprozessorsystemen wird oft auch der Begriff asynchroner Mehrkernprozessor verwendet.

die Pipelinelänge ständig, gleichsam Energieverbrauch und Leistung. In [8] kombinierte man dann zwei Generationen in ein System und erhielt ein asymmetrisches Mehrkernsystem mit gleichem Befehlssatz.[3]

In dieser Ausarbeitung wird auf den Aufbau und die Eigenschaften von big.LITTLE eingegangen werden, auf dessen Realisierung in eingebetteten Systemen und auf die Herausforderungen an die Software. Dies geht Hand in Hand mit der Erläuterung einiger Kernkonzepte von AMP-Systemen. Hingegen wird kurz auf das Konkurrenzprodukt von NVIDIA eingegangen werden, welches andere Merkmale als big.LITTLE aufweist und welches einige Klassifizierungsprobleme aufwirft.

2 Vorstellung der Hardware

In diesem Kapitel sollen die beiden Prozessoren Cortex-A7 und Cortex-A15, sowie das big.LITTLE-System im Ganzen vorgestellt werden.

2.1 Genereller Aufbau

Das von ARM vorgestellte big.LITTLE verbindet zwei Prozessoren mit unterschiedlichen Merkmalen auf einem „System on a Chip" (kurz SoC). Das Hauptaugenmerk liegt bei big.LITTLE auf den beiden Prozessoren, aber die Referenzplattform von ARM hat noch weitere, teils lediglich empfohlene Komponenten definiert (vgl. [11]). Dazu gehören zum Beispiel der auf Abbildung 1 sichtbare GIC, aber auch die nicht dargestellte jedoch von ARM empfohlene Mali Graphics Card (vgl. [11].

Das Ziel beim Entwurf von big.LITTLE war nach [6, 11] die möglichst energieeffiziente Ausführung von Anwendungen. Gleichzeitig soll das System sehr gut für rechenintensive Aufgaben skalierbar sein.

Die erste Version von big.LITTLE, vorgestellt in [11], beinhaltet, wie auch auf der Abbildung 1 ersichtlich, den energieeffizienzen Prozessor Cortex-A7 und den leistungsstarken Prozessor Cortex-A15. Wichtig ist, dass man zur Vereinfachung immer von einem Cortex-A7 bzw. Cortex-A15 spricht, aber damit jeweils der Kerntyp gemeint ist, da pro SoC mehrere Prozessoren eines Typs vorhanden sein können.

Im oberen Teil der Abbildung ist ferner der GIC, der Generic Interrupt Controller, zu sehen. Er ist in der Lage im laufenden Betrieb, je nach Anforderung, die Interrupts von einem zu einem anderen Prozessor zu migrieren. Unten im Bild sieht man den CCI 400 Port, der kurz in Kap. 2.3 erläutert wird.

[3] Bei der Betrachtung des Befehlssatzes hat man Erweiterungen außer Acht gelassen.

Mittlerweile wurde das Konzept um zwei Prozessoren erweitert; laut [6] kann nun der Cortex-A53 als energieeffizienter und der Cortex-A57 als leistungsstarker Kern eingesetzt werden. An dieser Stelle soll vor allem die Variante mit den Prozessoren Cortex-A7 und Cortex-A15 beschrieben werden, da die neuere Variante vor allem aktualisierte Prozessoren in das System bringt. Die Idee der Technologie bleibt jedoch identisch. Lediglich die Spanne zwischen „groß" (leistungsstark) und „klein" (energieeffizient) wächst mit dem Cortex-A53 und dem Cortex-A57[4] (vgl. [5]).

Forschung zum Nutzen und der Verwendung von asymmetrischen Mehrkernprozessoren erfolgt schon länger. In [13] wird ein asymmetrisches System (unter Nutzung von Simulation) evaluiert, bei dem mehrere Generationen von Prozessoren einer Architektur in einem Mehrkernprozessor auf einem SoC kombiniert wurden. Damit erhielt man ein asymmetrisches Mehrkernsystem mit einem Befehlssatz, bei dem die Mikroarchitektur verschieden war. Als Resultat dieser Forschung ergaben sich Anforderungen an ein System, bei dem die Kerne unterschiedliche Leistungs- und Energieniveaus haben. Diese kann der Leser im folgenden nutzen, um das big.LITTLE-System und dessen im Kapitel 5 vorgestellte Konkurrenz selbst zu beurteilen:

- Features: Ausgeglichenheit zwischen hohen Funktionsumfang und möglichst wenig Bauteilen zur Stromeinsparung
- Fähigkeit ein Executable zu teilen (echte Thread-Parallelität[5])
- Auswahl von mind. zwei verschiedenen Kernklassen, leistungs- und energieoptimiert
- wenig Überlappung der Arbeitspunkte der jeweiligen Prozessoren zueinander
- Verkürzung des kritischen Pfades bei der Migration von Arbeitslasten

2.2 Prozessortypen

Die Prozessoren Cortex-A7 und Cortex-A15 implementieren beide den ARMv7-Befehlssatz, wodurch sie sich aus Sicht der Software gleich verhalten. Trotzdem sind, wie im folgenden vorgestellt, die Unterschiede in der Mikroarchitektur teils enorm. Durch den ARMv7-Befehlssatz bringen diese Prozessoren Hardwarevirtualisierung, LPAE (Large Physical Address Extension)[6], NEON[7] und andere Features mit (vgl. [20, 3, 2]). Die

[4] Die Angaben sind nach momentanem Stand der Veröffentlichung; der Cortex-A57 ist der leistungsstärkste von ARM erhältliche Prozessor und der Cortex-A53 der energieeffizienteste.

[5] Es gibt zwei Modi, in denen ein AMP-System betrieben werden kann, welche in Kap. 3 auf Seite 6 erläutert werden. Genannte Voraussetzung bezieht sich auf den gleichzeitigen Betrieb von asymmetrischen Kernen.

[6] LPAE: zur Adressierung von großen Hauptspeichern unter 32-Bit Systemen.

[7] NEON: SIMD-Erweiterung für den ARM-Befehlssatz.

beiden Prozessoren haben eigene L1 und L2 Caches (vgl. [11]). Zwar wäre platztechnisch ein gemeinsamer L2-Cache günstiger gewesen, doch erlaubt dieses Design eine Energieoptimierung im L2-Cache für den Cortex-A7.

Um eine schnelle Migration der Arbeitslast zwischen den Prozessoren zu erlauben, sind die beiden Kerne über einen AMBA 4-Hochgeschwindigkeitsbus[8] miteinander verbunden. Über diesen lassen sich auch weitere Kerne verbinden, es sind pro Prozessortyp 1–4 Prozessoren möglich. Im folgenden wird jeweils von den beiden „Kernen" gesprochen, obwohl es sich um mehrere Kerne von einem Kerntyp handeln kann.

2.2.1 ARM Cortex-A7

Der Cortex-A7 ist der kleine Kern der big.LITTLE-Kombination, wobei „klein" relativ zum leistungsstärkeren Cortex-A15 gesehen werden muss. Er verbraucht nach Angaben von [11] nur ein Bruchteil der Energie des aktuellen Cortex-A8 mit höherer Leistung. So soll er 60 % energieeffizienter als der Cortex-A8 sein und dennoch 20 % mehr Leistung besitzen (vgl. [3]). Ebd. wird betont, dass Energieeffizienz das Hauptaugenmerk beim Entwurf dieses Prozessors war.

Wie bei Abbildung 2 zu sehen ist der Cortex-A7 ein dual-issue-Prozessor[9] unter Einhaltung der Befehlsreihenfolge und einer Pipeline von 8–10 Stufen. Gerade die Pipelinelänge unterscheidet sich stark von dem Cortex-A15; laut [11] hängt der Energieverbrauch stark von der Länge der Pipeline ab, was eine kürzere Pipeline für diesen Prozessor sinnvoll erscheinen lässt. Ferner verfügt er über eine Sprungvorhersage.

L1- und L2-Caches sind für den Cortex-A7 privat.[10] Zur Einsparung von Chipfläche hätte nach [11] ein gemeinsamer L2-Cache für beide Prozessortypen große Vorteile gebracht, hingegen kann bei privaten L2-Caches das Hauptaugenmerk auf Energieeffizienz gelegt werden. Er kann mit 64 Bit Bandbreite Load/Store-Operationen und mit 128 Bitbreite auf den AMBA-Bus zugreifen und hat damit die doppelte Speicherbandbreite als der Cortex-A5 (vgl. [6, 3]).

Der Cortex-A7 ist auf bis zu vier Kerne aufrüstbar. Zusätzlich können weitere Koprozessoren für Multimedia, Sicherheit und änlichem dazu konfiguriert werden. Jeder der einzelnen Prozessoren kann, sofern er nicht benötigt wird, einfach zur Energieoptimierung ab- und wieder angeschaltet werden (vgl. [3]).

2.2.2 ARM Cortex-A15

Im Gegensatz zum Cortex-A7 wurde beim Cortex-A15 das Augenmerk mehr auf Leistung gelegt, wobei auch dieser Prozessor sehr stromsparend ist. Der Prozessor besitzt eine OOO-Pipeline mit 15–24 Stufen und kann im Schnitt drei Befehle pro Takt an die Funktionseinheiten weiterreichen (sustained triple issue). Ferner hat er von jeder Funktionseinheit[11] im Vergleich zum Cortex-A7 mehr Ausführungen. ([11]. Auch wurden die Funktionseinheiten nach [2] selbst verbessert, z. B. die SIMD-Erweiterung NEON und die Sprungvorhersage.

Aufgrund der höheren Leistungsmerkmale unterscheidet sich der Cortex-A15 vom Cortex-A7 auch in der Cachetopologie. Der L2-Cache ist nicht nur integriert und extern betreibbar, auch Geschwindigkeit und Speichervolumen wurden vergrößert. Zusätzlich ist der L2-Cache in der Lage einfache Fehler zu korrigieren und zweifache Fehler zu erkennen. (vgl. [2, 11]).

Damit werden ganz klar nicht nur Smartphones und Tablets fokussiert, sondern auch Multimedia- und Streamingplattformen sowie Router, etc. Diese vordefinierten Profile von ARM unterscheiden sich vor allem in Taktrate und Anzahl der Kerne, die von 1 GHz bis zu $2{,}5$ GHz reichen (vgl. [11, 2]).

Die Leistung des Cortex-A15 ist gemessen an seinem Energieverbrauch laut [1] enorm. Er ist einer der leistungsstärksten Prozessoren, den es bis zu diesem Zeitpunkt von ARM gibt.

2.3 Vergleich beider Prozessoren

Beide Prozessoren Verfügen über einen CCI 400-Port (Cache Coherent Interconnect)[12] wodurch sie eine schnelle Speicheranbindung besitzen und Caches untereinander kohärent halten können. Damit sind die beiden Prozessoren in Kombination in einem big.LITTLE-System in der Lage, die Eingangs aufgestellte Anforderung, einen möglichst kurzen kritischen Pfad für die Migration bereitzustellen, zu erfüllen.

Bei der Abbildung auf Seite 19 sieht man das Leistungs- zu Energieverhältnis beider Prozessoren und deren Überlappung. Hier wird deutlich, dass ARM bewusst die Arbeitspunkte auseinander optimiert hat (vgl. [11]).

Den Leistungsvorteil vom Cortex-A15 gegenüber dem Cortex-A7 und den Energieeffizienzvorteil des Cortex-A7 gegenüber dem Cortex-A15 kann in der Tabelle 1 für einige

[8] AMBA: offene Bus-Protokollfamilie von ARM zur Verbindung von Komponenten in SoC-Systemen.
[9] Dual-Issue: Es können pro Takt bis zu zwei Befehle zu den Funktionseinheiten durchgereicht werden.
[10] An dieser Stelle bezieht sich „privat" auf das big.LITTLE-System. Der L2-Cache ist nur privat für den einen Kerntyp, aber bei z. B. mehreren Cortex-A7 untereinander ist der L2-Cache geteilter Speicher.
[11] Interne Funktionseinheiten des Prozessors wie Fließkommaeinheit, Ganzzahleinheit, etc.
[12] CCI 400: Protokoll aus AMBA 4-Protokollfamilie zur Cachekoherenz der beiden Kerntypen.

beispielhafte Anwendungen betrachtet werden. Obwohl die Benchmarks als absolute Messwerte nicht zur Rate gezogen werden sollten,[13] sind sie für eine relative Einschätzung durchaus geeignet, da hier die Unterschiede der Leistungs- bzw. Energieeffizienzmerkmale der Mikroarchitekturen gezeigt werden soll. Anschaulich sieht man die für den Cortex-A7 angesprochenen Optimierungen im L2-Cache-Bereich, die hier durch eine Energieeffizienzsteigerung von einem Faktor von $3{,}4$ sichtbar werden. Besonders beeindruckend ist der Energieeffizienzgewinn mit dem Faktor $3{,}8$ bei FDCT, was nahelegt, wie hoch die Energieeinsparung für bestimmte Anwendung sein kann.

Tabelle 1: Leistungs- und Energieverbrauch von Cortex-A7 und Cortex-A15 im Vergleich

Benchmark	Cortex-A7→Cortex-A15 (Leistung)	Cortex-A15→Cortex-A7 (Energieeffizienz)
Dhrystone	1,9 ×	3,5 ×
FDCT	2,3 ×	3,8 ×
IMDCT	3,0 ×	3,0 ×
MemCopy L1	1,9 ×	2,3 ×
MemCopy L2	1,9 ×	3,4 ×

(Quelle: [11, S. 3])

3 Betriebsmodi von big.LITTLE

Wie in [11] vorgestellt, kann big.LITTLE prinzipiell in zwei Modi betrieben werden. Bei dem einen Modus wird die Arbeitslast auf den für die Anforderung am besten passenden Kern abgebildet, beim anderen Modus hingegen werden beide Kerne genutzt.

3.1 big.LITTLE AMP

Im „big.LITTLE AMP Use Model" sind beide Kerne aktiv und das Betriebssystem verteilt die Last je nach Anforderung und Speicherintensität selbstständig ([11, 12]). Dieser Modus ist klassischer asymmetrischer Mehrprozessorbetrieb. Zur Nutzung von asymmetrischen Mehrprozessorsystemen ist bereits viel geforscht worden. So untersuchen z. B. [17], [7] und [19] den Einfluss von Scheduling auf asymmetrische Mehrprozessorsysteme. Einig ist man sich vor allem darüber, dass ohne Betriebssystemunterstützung die erhoffte Effizienzsteigerung in asymmetrischen Mehrprozessorsystemen nicht eintritt, was in Kap. 4.2.1 behandelt wird.

[13] Laut [14] ist z. B. Dhrystone nicht als Benchmark geeignet, da es aus den 80er Jahren des letzten Jahrhunderts stammt und eher CPU's aus damaliger Zeit mit den damaligen Anforderungen benchmarkt.

Da in diesem Modus beide Kern(typen) aktiv sind, können Spitzenlasten vom System abgefangen werden. Dies erlaubt im Vergleich zu SMP-Systemen trotzdem Energieeinsparung, da weniger CPU-intensive Aufgaben auf kleine Kerne ausgelagert werden können. Damit erhalten wir zusätzliche Parallelität (Thread-Level-Parallelität), ohne dass für die weniger CPU-intensiven Aufgaben „große" Kerne wie bei SMP-Systemen zum Einsatz kommen. (vgl. [11, 12, 8]).

3.2 big.LITTLE Migrationsmodell

Im „big.LITTLE Task Migration Use Model" ist immer nur ein Kerntyp aktiv und das Betriebssystem wird, transparent oder intransparent von einem Kern auf den anderen Kern migriert, ganz nach benötigter Energieeffizienz bzw. geforderter Leistung. Transparents bezieht sich dabei darauf, dass das Betriebssystem je nach dem die Migration selbst vornimmt oder von einem Hypervisor vorgenommen wird. Für den ersten Fall stellt ARM Software zur Verfügung, damit bereits existierende Betriebssysteme die Vorteile dieser Architektur ohne Anpassung nutzen können (vgl. [11, 12]). Das Betriebssystem läuft damit virtualisiert und nutzt die Features des ARMV7-Befehlssatzes zur Virtualisierung ([11]). Wird diese Software nicht genutzt, muss das Betriebssystem selbstständig die Migration anstoßen und die erforderlichen Schritte ausführen.

Im Umschalten liegt die Stärke dieses Modus (siehe 3.2.1). Gerade wenn die o. g. Software von ARM zur Migration verwendet wird, kann mit aktuellen Betriebssystemen in diesem Modus sehr hohe Rechenleistung genutzt werden. Trotzdem können durch das Umschalten auf den Cortex-A7 noch höhere Akkulaufzeiten für mobile Geräte durch erhöhte Stromeinsparung erreicht werden.

3.2.1 Ablauf der Migration

Die Umschaltung zwischen den Kernen kann transparent[14] und intransparent für das Betriebssystem erfolgen. Sie wurde stark optimiert, sodass nunmehr nur noch 20.000 Instruktionen nötig sind, d. h. in 20 μs auf einer CPU mit 1 GHz Taktrate die Umschaltung geschehen kann. Da heutige CPUs oftmals schneller als 1 GHz getaktet sind, ist die Umschaltzeit sogar noch etwas geringer. Dieser geringe Overhead fördert häufiges Umschalten zwischen den Kernen, um die möglichst korrekte Abbildung von Arbeitslast $\mapsto$ Kern zu erreichen.

Wenn das Betriebssystem von einen auf den anderen Kern migriert werden soll, so gibt es einen startenden und einen auslaufenden Prozessor. Auf dem Flussdiagramm der Abbildung 5 sind jeweils die Schritte der Migration untereinander. Jeder nach unten verlaufende Strang steht für einen Prozessor und zwei horizontal auf der gleichen

[14] In [11] bedeutet transparents, dass das Betriebssystem durch den Hypervisorcode von ARM nicht merkt, auf welchen Kern(en) es läuft.

Höhe angeordneten Aktionen geschehen im Ablauf gleichzeitig. Farblich ist der aktive Prozessor rot und der inaktive bläulich, was sich im Verlauf umkehrt.

Wenn beim noch inaktiven startenden Prozessor das Signal zur Aktivierung eintritt, wird dieser eingeschaltet, invalidiert seine Caches und aktiviert L2-Cache-Snooping (vgl. [11]). Wie ersichtlich führt der auslaufende Prozessor währendessen die Ausführung fort. Damit wird die sog. warming-up-Phase sinnvoll ausgefüllt. Erst wenn der startende Prozessor bereit ist, beginnt der auslaufende Prozessor nach einem Signal mit der Sicherung des Status. Im Grunde ist es der Kontext und zusätzliche interne Zustände der Pipeline.

Nachdem die Daten beim startenden Prozessor eingetroffen sind, stellt er diese wieder her. Hierbei hilft nach [12, 11] die 1-zu-1 Abbildung der Register (aufgrund der Architekturgleichheit), wodurch keine Zeit auf „Umsortierung" der Registerinhalte verwand werden muss.

Wie anfangs auch bei dem auslaufenden Prozessor, so führt jetzt der startende Prozessor die Ausführung der Anwendung(en) weiter fort, während der auslaufende Prozessor die letzten Schritte bis zur Deaktivierung vornimmt. Bevor der auslaufende Prozessor den L2-Cache leert, wartet er eine gewisse Zeit mit aktiven L2-Snooping, damit eventuell geänderte Daten über den CCI 400-Port übertragen werden können. Nach dem Leeren des Caches und dem Deaktivieren des L2-Cache-Snoopings kann der auslaufende Prozessor abgeschaltet werden.

Bei der schematischen Darstellung wurden, wie man leicht erkennt, Details stark vereinfacht. Diesen Vorgang in 20.000 Instruktionen, wie bereits eingangs erwähnt, ausfzuführen, ist daher eine starke Optimierung.

4 Herausforderungen an Software für big.LITTLE und verwandte Systeme

Die big.LITTLE-Architektur stellt an die Software, besonders an das Betriebssystem, neue Anforderungen. Das grundsätzlich immer wiederkehrende Problem ist dabei die fehlende Beurteilung der Asymmetrie.

Zwei Probleme treten für beide Betriebsmodi auf, weshalb diese bereits an dieser Stelle erläutert werden. In [11] wird davon gesprochen, dass die Register gleich sind (u. a. ein Merkmal für einen gemeinsamen Befehlssatz). Gleichsam wird angemerkt, dass sich zwei Register unterscheiden: CP15 und CPU-ID. Die CPU-ID muss sich auf jeden Fall unterscheiden, damit das Betriebssystem weiß, auf welchem Kerntyp es läuft. Das CP15-Register speichert die Cache-Topologie des Kerns. Wie bereits in Kapitel 2.2.1 erläutert, ist die Topologie des Cortex-A7 verändert worden, um energieeffizienter zu arbeiten. Wenn nun das Betriebssystem nach einer Migration auf dem anderen Kern

läuft, muss es auf die Veränderung gefasst sein. Nach [8] nehmen aktuelle Betriebssysteme meist an, dass die Register, die sie zuerst vorfanden, sich nie ändern. Diese Annahme kann aber aufgrund der Unterschiedlichkeit sogar bis zum Absturz des Systems führen. Dieses Problem gilt natürlich auch für den AMP-Betrieb, bei dem je nach Clock-Interrupt das Betriebssystem auf einem anderen Kern ausgeführt wird. Auch hier muss das Betriebssystem vor dem Zugriff auf Register prüfen, auf welcher CPU es sich befindet.

Sehr eng verwandt damit ist das Problem, dass viele Routinen des Betriebssystems zum Startzeitpunkt geladen werden. In den Architekturen ändern sich von Generation zu Generation immer wieder kleine Details, die dann eine Anpassung der betroffenen Routinen erfordert. Da die Auswahl zum Startzeitpunkt erfolgt, ist die Routine nur für den Startprozessor passend, nicht allerdings für die anderen Prozessoren. Das führt entweder zu Ineffizienz oder Abstürzen des Systems. [8] konnte dies speziell für den Linuxkern nachweisen, wies aber darauf hin, dass diese Probleme höchstwahrscheinlich auch bei anderen Betriebssystemen bestehen.

4.1 Problemstellungen für das big.LITTLE Migrationsmodell

Um die Migration von dem einen auf den anderen Kern vorzunehmen, hat man zwei Möglichkeiten. Entweder das Betriebssystem nimmt die Migration selbst vor oder sie wird von einer Schicht unterhalb des Betriebssystems vorgenommen, wodurch das Betriebssystem dann zur Middleware[15] wird.

Wenn das Betriebssystem die Migration selbstständig vornimmt, muss es den vollständigen Migrationsprozess (als Routine) implementiert haben (vgl. [11, 12]). Ein Fallstrick nach der Migration sind allerdings, wie auf der vorherigen Seite beschrieben, die verschiedenen CP15- und CPU-ID-Register. Da im big.LITTLE Migrationsmodell immer nur ein Kerntyp aktiv ist, bietet es sich für Leistungs- und Energieeffizienzoptimierung den Kerntyp zu wechseln, wenn jeweils die eine der beiden Eigenschaften benötigt wird. Eine Migration zwischen zwei Prozessoren kann teuer sein, wie zum Beispiel [13] bereits experimentell nachwies. Es ist daher wichtig, dass das System darauf rücksicht nimmt und nicht zu häufig zwischen den Kerntypen wechselt.

ARM stellt in [11] einen Switcher vor, der Unterhalb des Betriebssystems läuft und die Migration an dessen Stelle vornimmt; für diese Aufgabe kommt die Unterstützung von Hardwarevirtualisierung der Prozessoren zum Einsatz. 2.2 auf Seite 3). Diese Software überwacht den Leistungsbedarf des Betriebssystems und migriert es je nach Anforderung auf den jeweils anderen Kern. Ferner abstrahiert der Switcher von den geringen mikroarchitekturiellen Unterschieden. Dieser Switcher erlaubt es jedes für die ARM-Architektur erhältliche Betriebssystem zu verwenden.

[15] Begriff für eine Schicht zwischen Hardware-naher Software und Anwendungssoftware.

4.2 Problemstellungen beim asymmetrischen Mehrprozessorbetrieb

Der Mehrprozessorbetrieb, wie in Kap. 3.1 auf Seite 6 vorgestellt, fordert die Software in besonderem Maße, da aktuelle Systeme für homogene bzw. asynchrone Mehrprozessorsysteme ausgelegt sind. Um die Features des big.LITTLE AMP-Betriebes voll nutzen zu können, muss das verwendete Betriebssystem Unterstützung für big.LITTLE beinhalten. So lange der Switcher von ARM verwendet wird, ist AMP nicht möglich, da dafür das Betriebssystem nicht virtualisiert laufen darf. Unterstützung für AMP-Betrieb ist allerdings in aktuellen Betriebssystem mangelhaft (vgl. [8, 9][16]). Einerseits sind, wie auch schon 4 auf Seite 8 erklärt, die aktuellen Systeme auf die Unterschiede zwischen den CPU-Registern nicht eingestellt, andererseits können sie die Leistungsunterschiede nicht beurteilen und auch die Energieeffizienzabschätzung ist mangelhaft.

Laut [8] verwendete Linux nur einen CPU-Idle-Treiber, was bei homogenen Systemen ausreichend ist; der Treiber, zu der CPU auf dem Linux zuerst ausgeführt wurde, ist aber nicht immer optimal für den jeweils anderen Kern, da dieser andere Energiesparmöglichkeiten haben kann. Wie bereits erwähnt wird die Routine zum Startzeitpunkt gewählt, auf einem AMP-System läuft das Betriebssystem aber jeweils auf anderen CPUs. Dieses Problem wurde allerdings nach [9] in Linux 3.8 gelöst.

Ein weiterer Punkt ist das dynamische Hinzufügen und Entfernen von CPU-Kernen, falls Kerne zum Energiesparen abgeschaltet oder Kerne zur Leistungssteigerung zugeschaltet werden sollen. Dafür kann es sogar nötig sein, dass die Boot-CPU deaktiviert wird, worauf zumindest Linux lange Zeit, bis Linux 3.8, nicht vorbereitet war. Dies wird auch manchmal als CPU-Hotplugging bezeichnet und ist bis jetzt nur für x86-Systeme vollständig implementiert (vgl. [9]).

4.2.1 Allgemeines zu Scheduling in AMP-Systemen

Das Scheduling stellt eine besondere Herausforderung dar. Heutige Scheduler sind wie bereits erwähnt nicht in der Lage mikroarchitekturielle Unterschiede der Kerne in die Prozessplanung einzubeziehen. Nach [8] berücksichtigen momentane Systeme lediglich Unterschiede beim DVFS (dynamic voltage and frequency scaling).

Die wichtigste Planungsfrage ist, welches der korrekte Kern für die aktuelle Anwendung, bzw. den aktuellen Thread ist. Jeder Scheduler gibt auf diese Frage eine leicht andere Antwort. [19] stellt z. B. fest, dass die Annahme, langsamere Kerne wären für speicherintensive Anwendungen grundsätzlich besser geeignet, nicht korrekt ist.

[16] [9] wird an dieser Stelle als Quelle aufgeführt, da dort erwähnt wird, dass einige Änderungen für den Betrieb von Linux auf big.LITTLE vollzogen wurden. Alle Probleme sollten damit aber noch nicht behoben sein.

In [7] wurden zusätzlich Untersuchungen von dem Ausführungsverhalten verschiedener Java-Anwendungen vorgenommen. Laut den Autoren sind virtuelle Maschinen dankbare Ziele für Optimierungen, da virtuale Maschinen immer mit Leistungseinbußen einhergehen und mit AMP-Systemen die Möglichkeit besteht diese zu relativieren. Android setzt beispielsweise auf eine virtuelle Maschine namens Dalvik[17] wodurch die dort gewonnenen Erkenntnisse in direkter Anwendung beim Scheduling eingesetzt werden könnten. In der Veröffentlichung wurde unter anderem untersucht, wieviel der insgesamten Ausführzeit für Garbage Collection, Just-In-Time-Kompilation oder für Interpretation verwendet wurden. Ferner wurden Empfehlungen gegeben, welche Kerntypen sich für die jeweiligen Threads der virtuellen Maschine eignen. Für big.LITTLE könnte man beispielsweise, wenn man vom big.LITTLE Mehrprozessormodell ausgeht, Interpretation und JIT auf dem schnellen Kern und GC auf dem langsamen Kern ausführen, da gerade letzteres besonders hohe Speicherbandbreite, aber keine CPU-Intensität besitzt.

4.2.2 Comprehensive scheduler for Asymetric Multicore Processors (*CAMP*)

Der in [17] vorgestellte CAMP-Scheduler möchte das Problem beheben, dass asymmetrische Scheduler nur auf Energieeffizienz oder Leistung optimieren, denn dies wird laut den Autoren nicht einer AMP-Plattform gerecht. „Leistung" war in dieser Veröffentlichung die Threadparallelität (TLP). Zusätzlich soll noch die durchschnittliche Befehlszahl pro Takt maximiert werden. So sollen beispielsweise sequenzielle Programme auf großen, OOOE superskkalaren Kernen und TLP-intensive auf vielen kleinen Kernen ausgeführt werden. Ebd. wird erwähnt, dass frühere Forschungen gezeigt haben, dass für hohe Parallelität sowieso viele kleinere Kerne besser sind.

Dafür schätzt man den Beschleunigungsfaktor (**BF**), den man bei der Migration von dem einen auf den anderen Kern erhält. Die zuverlässigste Variante ist natürlich das Ausführen von dem Thread auf beiden Kernen (Sampling-based), dieser Overhead ist allerdings nicht zweckdienlich. Deshalb greift man in [17] auf eine modellbasierte Schätzung zurück. Ein Teil der Daten für die Planungsentscheidung wird aus einer statischen Tabelle bezogen, um die Schätzung der mikroarchitekturiellen Unterschiede zur vereinfachen. Diese Tabelle wurde zuvor mit einer Maschinenlernsoftware aus sehr vielen Messdaten aufgebaut. Ferner werden noch viele andere zur Laufzeit gesammelte Daten einbezogen und mit einer komplexen Formel zu einer Entscheidung herangezogen.

Der Beschleunigungsfaktor ist nur für Programme mit einem Thread von Nutzen, Bei mehreren Threads kann es aber sinnvoll sein, die Threads als zu einem Programm gehörig zu betrachten. In diesem Fall hilft uns die Metrik **BF** aber nicht weiter. Deswegen wurde ebd. eine neue Metrik **AG** (Auslastungsgrad) eingeführt. Dieser gibt an, ob ein

[17] Siehe Projektseite: `http://code.google.com/p/dalvik/`

Programm von der Migration profitiert, unter Berücksichtigung der Speicherintensität und der Threadparallellität. Für einen einzelnen Thread gilt **BF** $=$ **AG**. Für Multithreadanwendungen ist die Berechnung sehr komplex. In der Veröffentlichung wurde dann erst ein analytischer Ansatz besprochen, der aber vom Rechenaufwand nicht praxistauglich war. Anschließend wurde eine vereinfachte Formel aufgezeigt, die durch Experimente approximiert wurde.

Die Autoren verglichen ihren Algorithmus mit drei anderen Algorithmen, unter andrem mit Round-Rubin und einem anderen AMP-Scheduler und er erzielte immer bessere Ergebnisse.

4.2.3 Performance Impact Estimation (*PIE*)

In [19] wird eine Methode vorgestellt, mit der jeweils das Betriebssystem auf dem aktuellen Kern abschätzt, ob es Vorteile brächte (den aktuellen) Thread bzw. das aktuelle Programm auf einem anderen Kern auszuführen. Dabei wird die Schätzung mit dem aktuellen Kern verglichen. Es wird mehrfach darauf hingewiesen, dass alleine die Beurteilung nach Speicherintensität nicht ausreichend ist, wie sie zum Beispiel in [13, Kap. 1] vorgeschlagen wird. Oft wurden speicherintensive Aufgaben grundsätzlich auf langsamere Kerne verlagert, was laut den Autoren nicht immer günstig ist. Als Beispiel wird ein Thread mit wenig Speicherebenparallelität (MLP), keinen Datenabhängigkeiten und hoher Befehlsebenenparallelität (ILP) angeführt. Solch Threads können sehr gut auf kleinen Kernen mit fester Befehlsreihenfolge ausgeführt werden. Bei hoher Speicherparallelität und ähnlichem ist hingegen ein großer Kern mit OOO-Ausführung sinnvoll.

Um die Abschätzung des Leistungsgewinns korrekt vorzunehmen, wird die Speicherintensität zusammen mit den Eigenschaften der Mikroarchitektur des anderen Prozessors vorgenommen. Die Speicherzugriffsrate wird hier mit einer Formel aus drei Komponenten gebildet

1. LLC-Miss-Rate: Anzahl der Fehlzugriffe des Caches letzter Ebene
2. Latenz pro Fehlzugriff[18]
3. Anzahl der gleichzeitigen und verbleibenden Fehlzugriffe

Es wurden Daten wie Größe des Reorderbuffers o. ä. einbezogen, um möglichst genaue Schätzungen zu erreichen. Für die Formeln, auch für die Beurteilung der OOOE vs. inorder-Prozessoren sei auf die Veröffentlichung verwiesen.

Dieser Algorithmus ist laut den Autoren besser als alle bis dahin vorgestellten Algorithmen, gerade durch die Einbeziehung vieler vorhandener Daten über den Prozessor.

[18] In dieser Veröffentlichung ging man davon aus, dass die Cachetopologien der einzelnen Prozessoren gleich sind.

Trotzdem wird eine gewisse Hardwareunterstützung benötigt, in Summe etwa 15 Byte an Registerspeicher.

5 Vergleich von verschiedenen asymmetrischen Systemen

5.1 Vergleich mit asynchronen Systemen

AMP und speziell big.LITTLE bietet im Vergleich zu synchronen und asynchronen Mehrkernsystemen deutliche Vorteile in Hinsicht Energieeffizienz (vgl. [12, 8, 7]).

[12] führt dazu in Kap. 3 ein Beispiel an (siehe Abbildung). Gegeben seien zwei Kerne mit unterschiedlicher Taktung. Der Kern mit niedrigerer Taktung spart zwar Energie, hat aber im Großen und Ganzen auch weniger Leistung. Asymmetrische Systeme sind noch stromsparender, da dort nicht nur die Taktrate, sondern auch die Mikroarchitektur variiert wurde. Bei Mehrprozessorsystemen müssen von der Snoop Control Unit (SCU) ständig Daten von einen auf den anderen Prozessor übertragen werden; durch die verschiedenen Taktraten müssen beide Kerne aber aufeinander warten, was zu zusätzlichen Leistungseinbußen im System führt. Dies kann zwar, je nach Design, auch bei asymmetrischen Systemen auftreten, da dort aber die Energieeinsparung wesentlich höher ist, ist der Gesamtverlust geringer.

Die grundsätzlichen Vorteile von asymmetrischen Mehrprozessorsystemen im Vergleich zu homogenen und asynchronen Systemen sollen hier nicht erneut behandelt werden, da durch die vorangegangenen Kapitel deutlich geworden sein sollte, dass asymmetrische Systeme zwar etwas weniger Leistung als klassische SMP-Systeme besitzen, aber deutlich energieeffizienter arbeiten.

5.2 Vergleich von Samsung Exynos 5 Octa und NVIDIA Tegra 4 — ARM big.LITTLE gegenüber NVIDIA VSMP

big.LITTLE wurde bisher von mehreren Firmen aufgegriffen. Bei Samsung wird big.LITTLE in SoC's mit der Serienbezeichnung Samsung Exynos produziert und vertrieben. Die aktuellste ist die Exynos 5 Octa-Serie, bei der die maximale Konfiguration von [11] verwendet wird, indem man vier Cortex-A7 mit vier Cortex-A15 verbindet (siehe Abbildung auf Seite 22). Dieser Prozessor wird in Mobiltelefon der Samsung S4-Reihe verbaut, wobei die ersten Modelle mit einem Qualcomm-Prozessor und erst die Modelle Ende 2013 mit dem Samsung Exynos Octa ausgeliefert werden (vgl. [10]).

NVIDIA hat als direkte Konkurrenz das Tegra 4 System vorgestellt. Wie bei der Abbildung auf Seite 22 ersichtlich, wird auf vier-PLUS-einen Kern gesetzt. Es ist fünfmal der Cortex-A15, laut [14] läuft der fünfte Kern allerdings mit geringer Frequenz und wurde

mit einer speziellen Halbleitertechnologie gefertigt[19] und verbraucht deshalb weniger Energie. Er soll aber dessen ungeachtet trotzdem für Musik- und einfache Videowiedergabe, E-Mail und Hintergrundanwendungen geeignet sein.

Die Technologie dahinter nennt sich VSMP und übernimmt die Migration von der laufenden Software[20] transparent.[21] Da nie beide Kernvarianten gleichzeitig betrieben werden, ist keine Hardware für Cachekohärenz notwendig, was laut [14] zur Einsparung an Hardware und deren Energieverbrauch führt. Damit ist NVIDIA's Tegra 4 mit dem Migrationsmodus (siehe Kap. 3.2 auf Seite 7) vergleichbar, aber nicht mit dem big.LITTLE AMP.

Die Frage ob VSMP asymmetrisch ist, ist je nach verwendeter Definition schwierig. Laut [15] ist das System asynchron; durch die verwendete Halbleitertechnik haben die Prozessoren allerdings wie auch bei big.LITTLE asymmetrische energetische Eigenschaften. Unklar ist auch, in welcher Konfiguration der „kleinere" Cortex-A15 verwendet wird, da nach [11] die Pipelinelänge bei der Herstellung variieren kann und dies sehr wohl ein mikroarchitekturieller Unterschied ist.
Laut [14, 15] bietet dieses Vorgehen den Vorteil, dass bestehende Software nicht angepasst werden muss und die Hardware nach Arbeitslast entscheiden kann, welche und wieviele Kerne genutzt werden. Dies gilt allerdings auch für big.LITTLE bei Verwendung des ARM Switchers.

Ferner unterscheiden sich die beiden Systeme, wie bei der Abbildung auf Seite 22 ersichtlich, im Aufbau der Caches. Bei ARM hat jeder Kerntyp seinen eigenen L2-Cache, um ihn jeweils auf Leistung- oder Energieeffizienz zu optimieren (vgl. [11]). NVIDIA setzt ebenfalls auf getrennte L2-Caches, wobei der Cache für den energieeffizienten fünften Kern lediglich $512\,\mathrm{k}$B groß ist, während der übliche L2-Cache $2\,\mathrm{M}$B umfässt (siehe Abb. 5.2). Der große Cache kann dann bei nicht-Benutzung, wie auch bei big.LITTLE, abgeschaltet werden. (vgl. [11, 15, 14]).

Welches nun das bessere System ist, kann man nicht so einfach sagen. In Benchmarks schneidet NVIDIAs Tegra 4 bei AnTuTu oder Quadrant um Längen besser ab als Samsungs Exynos Octa (vgl. [15, 18]). Hingegen migriert, wie bereits in Kap. 3.2.1 beschrieben, big.LITTLE in $20\,\mu\mathrm{s}$ (bei einer CPU mit $1\,\mathrm{GHz}$) und NVIDIAs VSMP benötigt nach [14] $2\,\mathrm{ms}$ und damit deutlich länger. Außerdem weißt [14] darauf hin, dass durch

[19] In [14] wird erläutert, dass Halbleiter, die für hohe Taktraten optimiert sind, höhere Leckströme aufweisen; da für den 5. Kern keine hohe Taktrate benötigt wird, wurde auf eine andere Halbleitertechnologie umgestellt.

[20] NVIDIA erwähnt in [14], dass die dynamische Frequenz- und Spannungsskalierung sowie das CPU-Hotplugging von einer Software übernommen wird; der Einsatz ist wahrscheinlich ähnlich wie bei big.LITTLE.

[21] In [14] wird „transparent" ebenfalls wie in [11] (siehe Kap. 3.2.1 auf Seite 7) dafür gebraucht, dass das Betriebssystem die Umschaltung nicht bemerkt.

die Homogenität[22] aktuelle Systeme keine falschen Scheduling-Entscheidungen treffen. Über die tatsächliche Energieeffizienz, im Vergleich zur Leistungsstärke, ist noch nichts bekannt, [18] bescheinigt dem Samsung Octa lediglich eine höhere Laufzeit als seinen Vorgängern. Ferner hat Samsung bereits ein Referenzmodell vorgestellt, das Samsung GalaxyS4, für NVIDIA existiert noch kein Modell.

6 Zusammenfassung

In dieser Veröffentlichung wurde das big.LITTLE-System von ARM vorgestellt. Dieses System kommt den Anforderungen nach immer mehr Leistung in mobilen Plattformen, aber auch immer mehr Energieeffizienz gleichsam nach. Um diese Anforderungen zu erfüllen, gibt es bei jedem big.LITTLE-System zwei verschiedene Kerntypen. Der Cortex-A7 ist der energieeffiziente Kern (LITTLE) und ermöglicht hohe Akkulaufzeiten, wohingegen der Cortex-A15 hohe Leistung liefert. Dadurch kann man je nach Anforderungsprofil wenig rechenintensive Aufgaben, wie e-Mails verfassen, Texte lesen oder SMS schreiben, auf dem energieeffizienzen Kern ausführen. Gleichzeitig steht auf dem gleichen SoC der leistungsstarke Kerntyp zur Verfügung, der für Browsing und andere rechentechnisch anspruchsvollere Aufgaben wie 3d-Karten oder Spiele genutzt werden kann.

Um diese Eigenschaften optimal nutzen zu können, müssen die Programme auf beiden Kerntypen konsistent ausgeführt werden, weshalb sich ARM für Kerne mit einem gleichen Befehlssatz entschied. Dennoch ist die Beurteilung, welcher Kern für welche Aufgabe am geeignetsten ist, nicht trivial. Das liegt u. a. daran, dass der Scheduler die mikroarchitekturiellen Unterschiede der Kerntypen dafür in Betracht ziehen müsste, diese Angaben liegen ihm aber nicht vor.

big.LITTLE wird zukünftigen Mobilgeräten wie Smartphones oder Tablets längere Laufzeiten ermöglichen, da der kleine, energieffiziente Kern viele alltägliche Aufgaben bewältigen kann. Gleichzeitig profitieren diese mobilen Plattformen enorm von dem leistungsstarken Cortex-A15 ([11, S. 2], [1]).

Literatur

[1] Anandtech Inc. Testing arm's cortex a15. `http://www.anandtech.com/show/6422/samsung-chromebook-xe303-review-testing-arms-cortex-a15/6`, 03 2013.

[22] Die Migration, sowie CPU-Hotplugging (Ein- und Ausschalten von einzelnen Kernen) übernimmt eine unter dem Betriebbssystem laufende Software von NVIDIA. Das System hat durch dieses Vorgehen System hat dadurch entweder vier gleiche oder einen kleinen Kern zur Verfügung, die Verteilung und Distribution übernimmt NVIDIA's Software (vgl. [14]).

[2] ARM Corporation. Cortex-a15 processor. http://www.arm.com/products/processors/cortex-a/cortex-a15.php, 09 2012.

[3] ARM Corporation. Cortex-a7 processor. http://www.arm.com/products/processors/cortex-a/cortex-a7.php, 09 2012.

[4] ARM Corporation. Amba open specifications. http://wwwarmcom/products/system-ip/amba/amba-open-specificationsphp, 03 2013.

[5] ARM Corporation. Arm cortex-a57. http://wwwarmcom/products/processors/cortex-a50/cortex-a57-processorphp, 03 2013.

[6] ARM Corporation. big.little processing. http://wwwarmcom/products/processors/technologies/biglittleprocessingphp, 03 2013.

[7] Ting Cao, Stephen M Blackburn, Tiejun Gao, and Kathryn S McKinley. The yin and yang of power and performance for asymmetric hardware and managed software. In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, pages 225–236, Washington, DC, USA, 2012. IEEE Computer Society.

[8] Nagabhushan Chitlur, Ganapati Srinivasa, Scott Hahn, P K Gupta, Dheeraj Reddy, David Koufaty, Paul Brett, Abirami Prabhakaran, Li Zhao, Nelson Ijih, Suchit Subhaschandra, Sabina Grover, Xiaowei Jiang, and Ravi Iyer. Quickia: Exploring heterogeneous architectures on real prototypes. In *Proceedings of the 2012 IEEE 18th International Symposium on High-Performance Computer Architecture*, HPCA '12, pages 1–8, Washington, DC, USA, 2012. IEEE Computer Society.

[9] Jonathan Corbet. Linux 3.8 merge window part 1. https://lwn.net/Articles/528893/, Dec 2012.

[10] Andrew Cunningham. Samsung's exynos 5 octa: Checking out the chip inside the galaxy s 4. *Ars Technica*, 03 2013.

[11] Peter Greenhalgh. Biglittle processing with arm cortex-a15 & cortex-a7. http://www.arm.com/files/downloads/big.LITTLE_Final.pdf, 09 2011.

[12] Cho Hyun-Duk, Chung Kisuk, and Kim Taehoon. Benefits of the biglittle architecture. http://www.samsung.com/global/business/semiconductor/minisite/Exynos/data/benefits.pdf, Feb 2012.

[13] Rakesh Kumar, Keith I Farkas, Norman P Jouppi, Parthasarathy Ranganathan, and Dean M Tullsen. Single-isa heterogeneous multi-core architectures: The potential for processor power reduction. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 81–, Washington, DC, USA, 2003. IEEE Computer Society.

[14] NVIDIA Corporation. Variable smp – a multi-core cpu architecture for low power and high performance. http://www.nvidia.com/content/PDF/tegra_white_papers/tegra-whitepaper-0911b.pdf, 2011.

[15] NVIDIA corporation. Nvidia tegra 4 family cpu architecture. http://www.nvidia.com/docs/IO/116757/NVIDIA_Quad_a15_whitepaper_FINALv2.pdf, 03 2013.

[16] NVIDIA Corporation. Tegra 4 superchip processors. http://www.nvidia.com/object/tegra-4-processor.html, 03 2013.

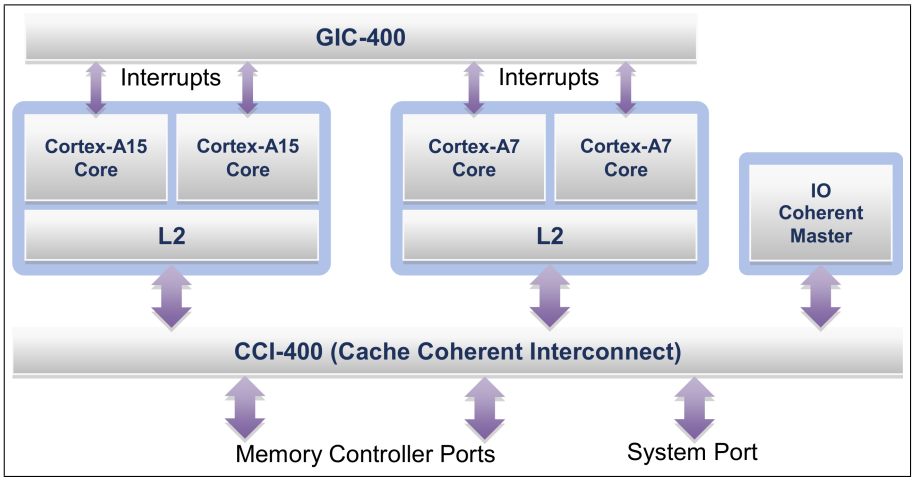
[17] Juan Carlos Saez, Alexandra Fedorova, David Koufaty, and Manuel Prieto. Leveraging core specialization via os scheduling to improve performance on asymmetric multicore systems. *ACM Trans Comput Syst*, 30(2):6:1–6:38, April 2012.

[18] Tabtech.de. Samsung galaxy s4 exynos 5410 octa-core im benchmark. http://www.tabtech.de/android-bzw-google-tablet-pc-news/samsung-galaxy-s4-exynos-5410-octa-core-im-benchmark, 03 2013.

[19] Kenzo Van Craeynest, Aamer Jaleel, Lieven Eeckhout, Paolo Narvaez, and Joel Emer. Scheduling heterogeneous multi-cores through performance impact estimation (pie). *SIGARCH Comput. Archit. News*, 40(3):213–224, 06 2012.

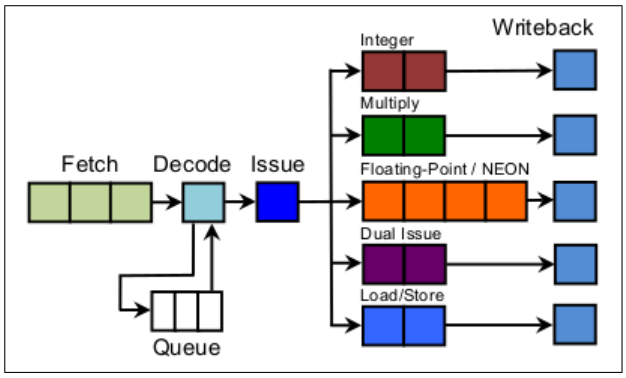
[20] Wikipedia. Arm-architecture. http://enwikipediaorg/w/indexphp?title=ARM_architecture&oldid=544275831, 03 2013.

Abbildung 1: Schematischer Aufbau big.LITTLE



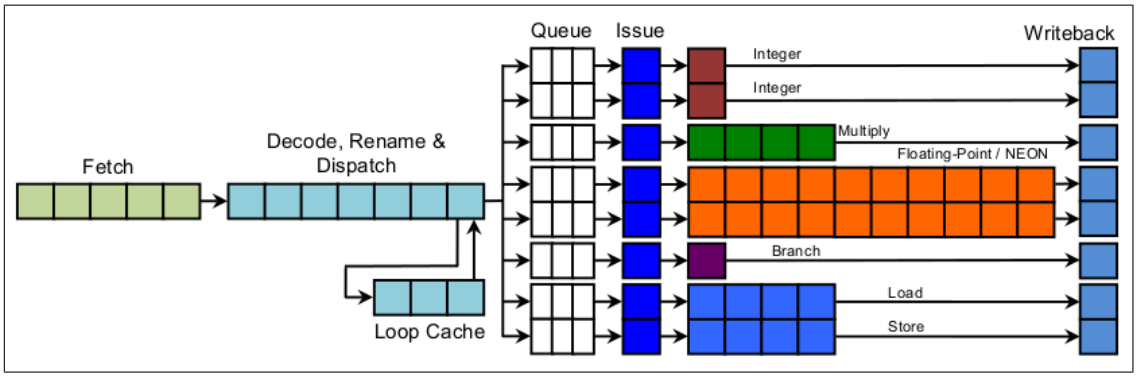
Quelle: [11]

Abbildung 2: Cortex-A7



Quelle: [11]

Abbildung 3: Cortex-A15



Quelle: [11]

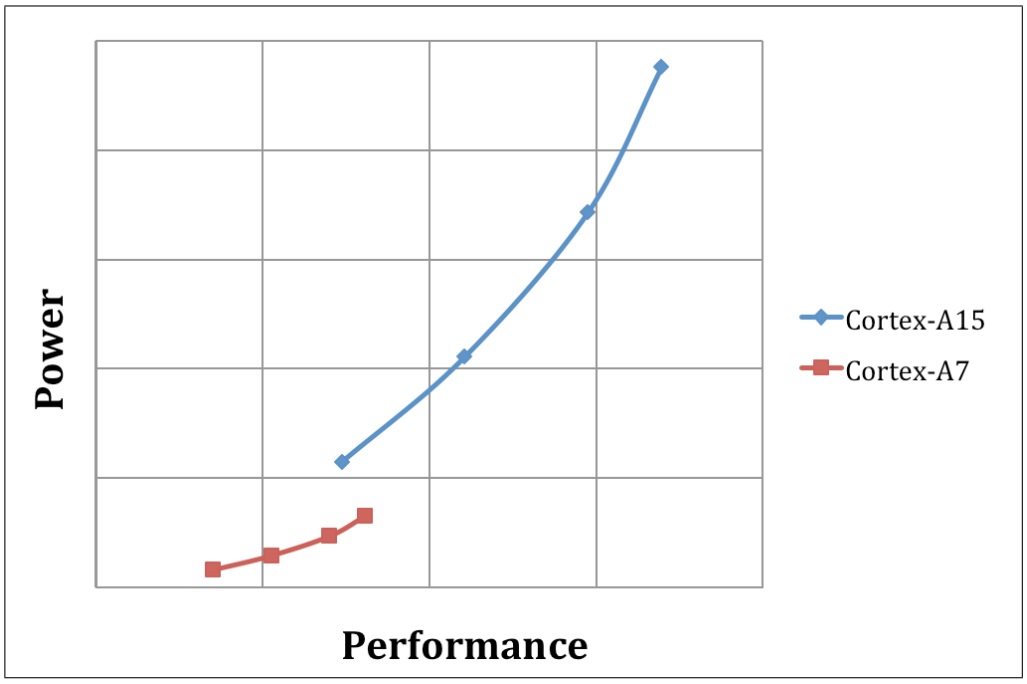
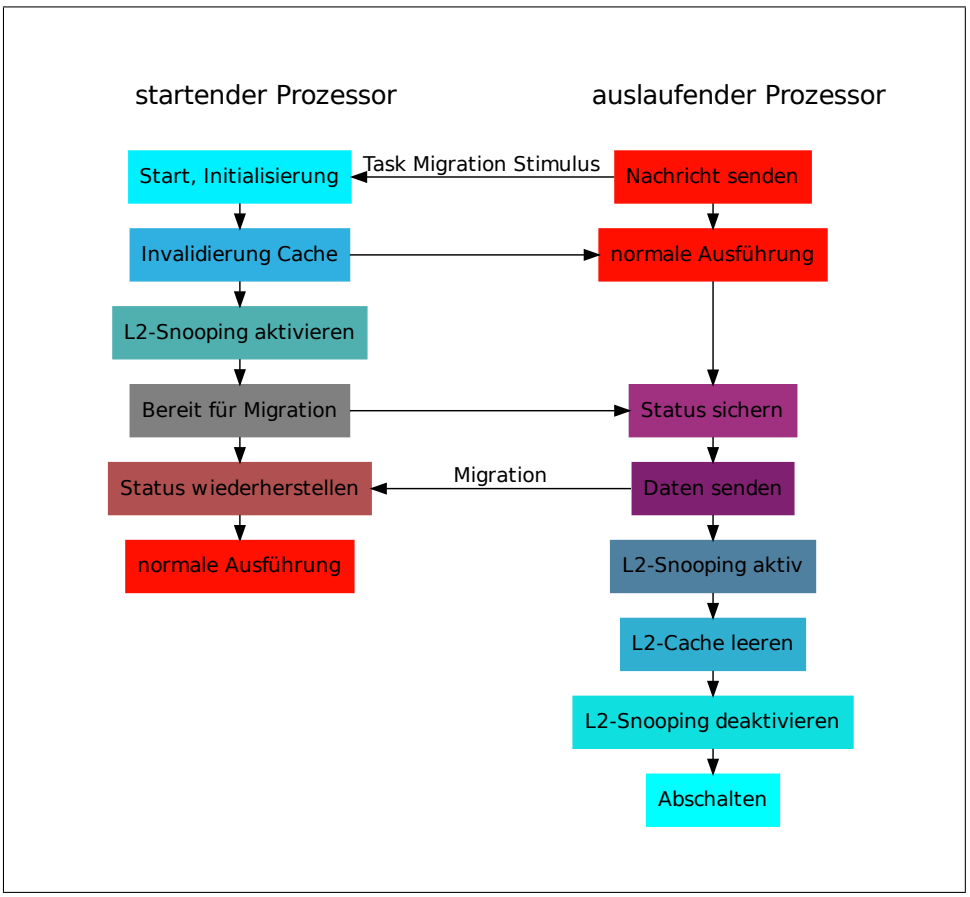


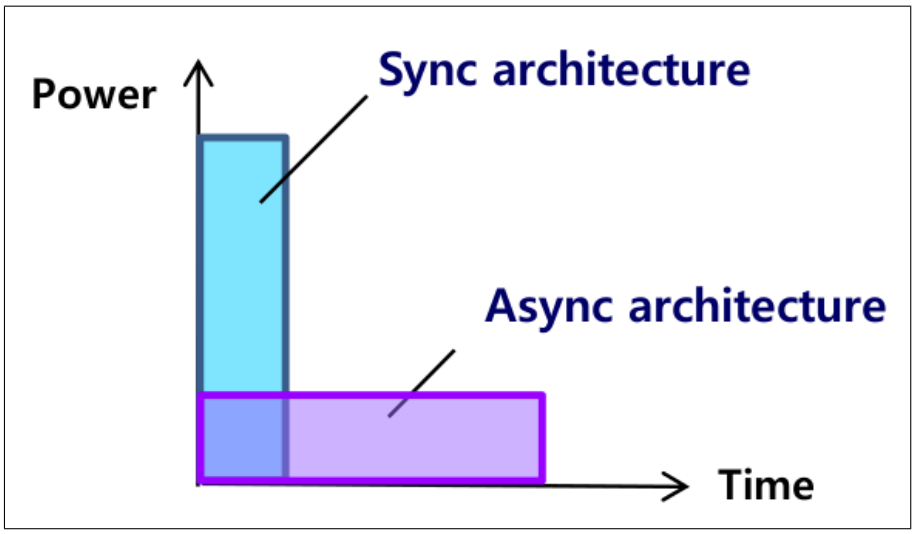
Abbildung 4: Arbeitspunkte des Cortex-A7 und des Cortex-A15 (Quelle: [11])

Abbildung 5: Ablauf der Migration



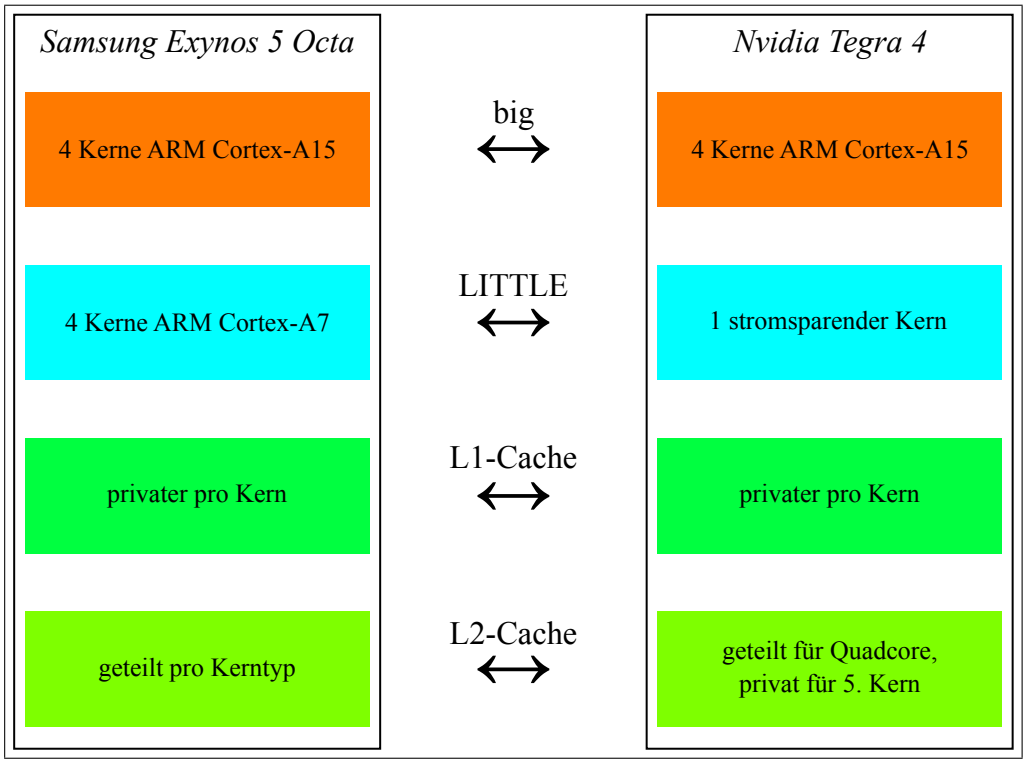
Quelle der Daten: [11]

Abbildung 6: Vergleich von asynchronen und synchronen Mehrprozessorsystemen



Quelle: [12, Kap. 3]

Abbildung 7: Schematischer Vergleich von NVIDIA Tegra 4 und Samsung Exynos 5 Octa



Quelle der Daten: [10, 15, 16]