technische universität
dortmund

Bachelorarbeit

Verarbeitung von Datenbankoperationen auf heterogener Prozessorarchitektur ARM big.LITTLE

Michael Hein
Juni 2017

Gutachter:
Prof. Dr. Jens Teubner
Thomas Lindemann

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl VI
http://dbis.cs.tu-dortmund.de

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Hintergrund

Im digitalen Zeitalter, wo mehr als 2,3 Milliarden Menschen ein Smartphone benutzen[4] und immer mehr Personen einen Dialog mit ihrem vernetzten Haus anfangen, sind eingebettete Systeme nicht mehr aus dem täglichem Leben wegzudenken. Mit der Verbreitung von mobilen Endgeräten stiegen auch die Anforderungen an die selbigen in den letzten Jahren stark an. Aktuelle Smartphones müssen leistungshungrige Multimediaanwendungen betreiben und dienen oft als Mobile-Office. Nebenbei werden die Bildschirme immer größer und die Geräte immer dünner. Mit handelsüblichen Mehrkernprozessoren ist es jedoch sehr problematisch energieeffiziente Systeme zu betreiben, die gleichzeitig genug Rechenleistung liefern und einen hinreichend kleinen Formfaktor besitzen.

Als effektive Alternative werden deshalb immer häufiger heterogene Prozessorarchitekturen verwendet, bei denen Prozessoren mit unterschiedlichen Leistungsmerkmalen und Aufgaben in Clustern zusammengefasst werden. Diese Technik sieht in der Regel vor, dass neben den Hauptprozessoren, die den Löwenanteil an Rechenleistung liefern, entweder weitere Prozessoren für kurzzeitige Leistungsschübe zugeschaltet werden können, oder für bestimmte Aufgaben optimierte Prozessoren Tasks für die Hauptkerne übernehmen. Diese Bachelorarbeit wird sich mit den sogenannten big.LITTLE-Prozessoren befassen.

Bei der von ARM entwickelten big.LITTLE-Technologie handelt es sich um vorwiegend für den Einsatz in mobilen Endgeräten vorgesehene, heterogene Mehrkernprozessoren. big.LITTLE- Prozessoren enthalten „big“-Kerne, die hohe Rechenleistung bringen und niedrig getaktete „LITTLE“-Kerne, welche auf Energieeffizienz ausgelegt sind. Da die Kerne auf den selben Speicher zugreifen und die gleiche Befehlssatzarchitektur verwenden, ist es möglich, je nach Leistungsanforderung Kerne ab- oder zuzuschalten und Threads dynamisch auf diese zu verteilen. Laut Hersteller soll dies für bestimmte Aufgaben eine deutliche Effizienzverbesserung im Vergleich zu einer homogenen Prozessorstruktur bringen[1].

1.2 Aufgabenstellung und Zielsetzung

In dieser Bachelorarbeit wird ein Hash-Join-Algorithmus auf einem big.LITTLE-Prozessor ausgeführt, um zu untersuchen, inwiefern sich unterschiedliche Thread-Verteilungen zu den „big“- und „LITTLE“-Clustern auf die Energieeffizienz im Bezug auf die Ausführungsgeschwindigkeit auswirken.

In der Entwicklungszeit der big.LITTLE-Prozessoren wurden vorwiegend 3 Modelle zur Thread-Zuweisung verwendet. Beim Clustered Switching werden die Prozessoren in 2 homogene Cluster unterteilt. Der Scheduler verteilt Threads je nach Leistungsbedarf auf einen der beiden Cluster. Der nicht verwendete Cluster wird deaktiviert. Das CPU-Migration-Modell sieht Paarungen von jeweils einem „big“-Kern und einem „LITTLE“-Kern zu einem virtuellen Kern vor. Auf diese virtuellen Kerne kann der Scheduler die Threads verteilen. Abhängig vom Leistungsbedarf wird in jedem virtuellen Kern entweder der „big“-, oder der „LITTLE“-Kern aktiviert. Beim sogenannten Global-Task-Scheduling stehen alle Kerne gleichzeitig zur Verfügung. Der Scheduler entscheidet anhand von Priorität und Leistungsbedarf, welchem Kern ein Thread zugewiesen wird.

Aus Zeit- und Ressourcenmangel beschränke ich mich in meiner Bachelorarbeit darauf, einen bzw. vier Threads durch das Betriebssystem exklusiv auf die „big“- oder „LITTLE“-Kerne verteilen zu lassen.

Das Hauptziel der Tests ist es, festzustellen, ob es merkliche Effizienzsteigerungen im Vergleich zu herkömmlichen Mehrkernprozessoren gibt. Des Weiteren wird die Effektivität der „big“- und „LITTLE“-Kerne bei Hash-Operationen direkt verglichen. Dies soll Aufschluss darüber liefern, wie beim Scheduling die Threads besser zugewiesen werden können.

1.3 Aufbau der Arbeit

Das erste Kapitel leitet in das Thema meiner Bachelorarbeit ein, indem Motivation, Aufgabenstellung und Zielsetzung der Arbeit erläutert werden. Kapitel 2 beschäftigt sich mit den Rahmenbedingungen unter denen meine Messungen durchgeführt werden. In Kapitel 3 werden die Messergebnisse für Laufzeit und Energieverbrauch sowie ermittelte Effizienzwerte betrachtet. Anschließend wird ein Fazit gezogen. Kapitel 4 verweist auf verwandte Arbeiten. Das letzte Kapitel enthält ein Schlusswort meinerseits.

Kapitel 2

Versuchsaufbau

2.1 Testhardware

Für meine Arbeit wurde mir ein Einplatinencomputer vom Typ Odroid XU4 bereit gestellt. Dieser wurde mit einem Samsung-Exynos5422-big.LITTLE-Prozessor ausgestattet, der mit jeweils vier „big“- und vier „LITTLE“-Kernen bestückt ist. Der Exynos5422 unterstützt Global-Task-Scheduling und wurde im Galaxy-S5-Smartphone verbaut.

Für einen Leistungsvergleich im Bereich der eingebetteten Systeme wurde ein Odroid C2 zur Verfügung gestellt. Dessen Amlogic-s905-Prozessor wird vorwiegend in Multimedia-Boxen verbaut. Aus diesem Grund ist standardmäßig ein Teil des Arbeitsspeichers für HD-Bildwiedergabe reserviert. Für die durchgeführten Messungen wurde die Reservierung jedoch aufgehoben.

Für Vergleichswerte aus dem gehobenen Leistungssegment wurde ein Poolrechner des Lehrstuhls VI durchgemessen. Im Dell Optiplex 9020 bildet ein Intel-I7-4790-Prozessor mit 3,6 GHz pro Kern die Grundlage für eine leistungsstarke Workstation.

	Odroid XU4	**Optiplex 9020**	**Odroid C2**
CPU	SAMSUNG Exynos 5422 Cortex-A15(2Ghz) & Cortex-A7(1,4Ghz) big.LITTLE, Octa core	Intel I7-4790 (3,6 Ghz)	Amlogic S905 SoC Cortex A53 (ARM v8) 1,5 GHz, QuadCore
GPU	Mali-T628 MP6	Intel HD Graphics 4600	Mali-450 MP3 GPU
Arbeitsspeicher	2 GB LPDDR3 RAM	16GB	2 GB DDR3 RAM
Festplatte	64GB eMMC Modul	128GB Sata SSD	8GB eMMC Modul
Idlebetrieb	~2,2W	~35W	~1,8W

Tabelle 1: Hardwarespezifikationen der Testgeräte. Quellen:[3],[2]

Um die Anzahl an Hintergrundprozessen gering zu halten und möglichst viel Arbeitsspeicher nutzen zu können, wurde auf den beiden Einplatinencomputern ein Minimalbetriebssystem installiert.

Für Stromverbrauchsmessung an den beiden Odroids wurden Ina219-Sensormodule verwendet. Diese konnten über die I^2C-Schnittstelle am Odroid C2 ausgelesen werden. Für den Poolrechner wurden Näherungswerte mit Hilfe eines Verbrauchsmessers am Netzteil ermittelt.

Beide Einplatinencomputer wurden ohne Drosselung der Prozessortaktfrequenz betrieben. Die vom Odroid XU4 zu verwendenden Kerne wurden durch Umgebungsvariablen festgelegt, während die Anzahl gleichzeitig laufender Threads über den Testcode gewählt wurde.

2.2 Testdaten

Als Testdatensätze wurden Rohdaten verwendet, die mit dem TPC-H-Datengenerator erzeugt wurden. Der TPC-H-Benchmark wird von vielen Hardwareherstellern (z. B. Dell, HP) verwendet, um Datenbanksysteme zu testen und die verwendeten Datensätze bieten eine realitätsnahe Komplexität[5]. Um möglichst schnelle Bearbeitungszeiten zu erzielen, werden die Testdatensätze komplett in den Hauptspeicher geladen.

Um genug Arbeitsspeicher für die benötigten Operationen zur Verfügung stellen zu können, wurde als Skalierungsfaktor für die Rohdatenerstellung 0,5 und 1 gewählt, was ca. 500 MB bzw. 1 GB an Rohdaten entspricht. Der Odroid C2 hatte Probleme bei der Speicherverwaltung und konnte deshalb nur mit Skalierungsfaktor 0,5 getestet werden.

2.3 Algorithmus

Der Algorithmus führt zwei Equi-Joins mit unterschiedlich großer R-Tabelle und identischer S-Tabelle durch. Für jedes Tupel in S gibt es Tupel mit gleichem Index in R. Somit ist die maximale Anzahl an Joinoperationen garantiert. Zum weiteren Vergleich wurde eine Query aus dem TPC-H-Benchmark nachgebaut, die einen der Equi-Joins um mehrere Filterprädikate erweitert.

Die Funktion des Hash-Join-Algorithmus lässt sich kurz zusammenfassen: In der Build-Phase wird die kleinere der beiden Tabellen nach Wahl eines passenden Indizes in eine Hash-Tabelle geladen. Daraufhin wird in der Scan-Phase die zweite Tabelle durchlaufen und für jedes Tupel der Index in der Hashtabelle gesucht. Sollte in der Hashtabelle ein Tupel mit gleichem Index vorhanden sein, dann werden beide Tupel gejoint. Bei der Query werden in Build- und Scan-Phase zusätzlich die Tupel mit Filterprädikaten untersucht. Tupel, welche die zusätzlichen Einschränkungen nicht erfüllen, werden in den beiden Phasen übersprungen. Erfolgreiche Join-Operationen werden nicht ausgegeben, sondern einfach mit einem Counter gezählt, um nicht unnötige I/O-Operationen mitzumessen.

Als Programmiersprache wurde C gewählt. Um den Code multithreading-fähig zu machen wird die OpenMP-Bibliothek verwendet. Kompiliert wurde der Code mit dem für das entsprechende System vorgesehenen GCC. Zur Zeitmessung wurde eine monotonische Uhr des Betriebssystems verwendet.

Kapitel 3

Ergebnisse

Der Messalgorithmus wird mit drei verschiedenen Joins durchgeführt, die in dieser Bachelorarbeit fortlaufend mit Join1 bis Join3 angesprochen werden. Alle drei Joins verwenden Tabellen aus dem TPC-H-Benchmark. Die Tabellen und ihre Attribute werden mit den vom Benchmark vorgegebenen Namen angesprochen.

Join1 ist ein reiner Equi-Join zwischen den Tabellen *part* und *lineitem*. Bei Skalierungsfaktor 0,5 enthält *part* 100.000 und *lineitem* ca. 3 Millionen Tupel. Join2 joint die Tabelle *orders* mit *lineitem*, wobei *order* 750.000 Tupel enthält. Join3 ist eine Nachbildung von Query 14, welche die gleichen Tabellen wie Join1 unter Berücksichtigung bestimmter Prädikate bearbeitet.

Bei Join1 und Join2 werden etwa 3 Millionen Joinoperationen durchgeführt. Bei Join3 werden diese auf ca. 38.000 reduziert. Mit Skalierungsfaktor 1 verdoppeln sich die Anzahl an Tupeln und Joinoperationen.

Anhang A enthält eine ausführliche Auflistung der Testdaten.

3.1 Laufzeit

Bei Betrachtung von Abbildung 3.1 ist es offensichtlich, dass die „LITTLE“-Kerne im direkten Laufzeitvergleich nicht mit den anderen Kernen mithalten können. Auf Grund der Optimierung dieses Prozessortyps auf Energieeffizienz ist ihr schlechtes Abschneiden in Bezug auf Laufzeit jedoch nicht überraschend.

Beim Vergleich von „big“- und „LITTLE“-Kernen fällt auf, dass bei reinen Equi-Joins sich der Laufzeitunterschied verringert, wenn die geladenen Tabellen beide mehr Tupel enthalten. Wenn nur die R-Tabelle größer wird, treten die Verbesserungen nur beim Multi-Threading ein. Die Laufzeitreduzierung von Join3 im Vergleich zu Join1 ist im Single-Core-Betrieb auf „LITTLE“-Prozessoren deutlich geringer als auf „big“-Prozessoren; im Multi-Threading-Betrieb mit vier Threads bleibt das Verhältnis konstant. Bei Erhöhung

des Skalierungsfaktors nimmt der Laufzeitunterschied deutlich zu. Im Vergleich der Joins scheinen die „LITTLE“-Kerne mit den reinen Equi-Joins deutlich besser arbeiten zu können.

Beim Vergleich vom Poolrechner mit den „LITTLE“-Kernen fällt auf, dass der Laufzeitunterschied bei größerem Skalierfaktor bei jedem Join stark zunimmt. Beim Vergleich der einzelnen Joins lassen sich die gleichen Veränderungen wie bei der Gegenüberstellung von „big“- und „LITTLE“-Kernen feststellen. Die „big“-Kerne verhalten sich beim Vergleich mit dem Poolrechner wie die „LITTLE“-Kerne. Die Laufzeitunterschiede steigen lediglich beim Multi-Threading weniger stark an.

Der Odroid C2 verhält sich im Laufzeitvergleich wie eine hochskalierte Variante des „LITTLE“-Clusters. Lediglich in der Multi-Threading-Variante von Join1 kam es zu Laufzeiteinbußen seitens des C2. Die „big“-Kerne hängen den C2 in allen Messungen ab, wobei der Laufzeitunterschied größtenteils konstant bleibt.

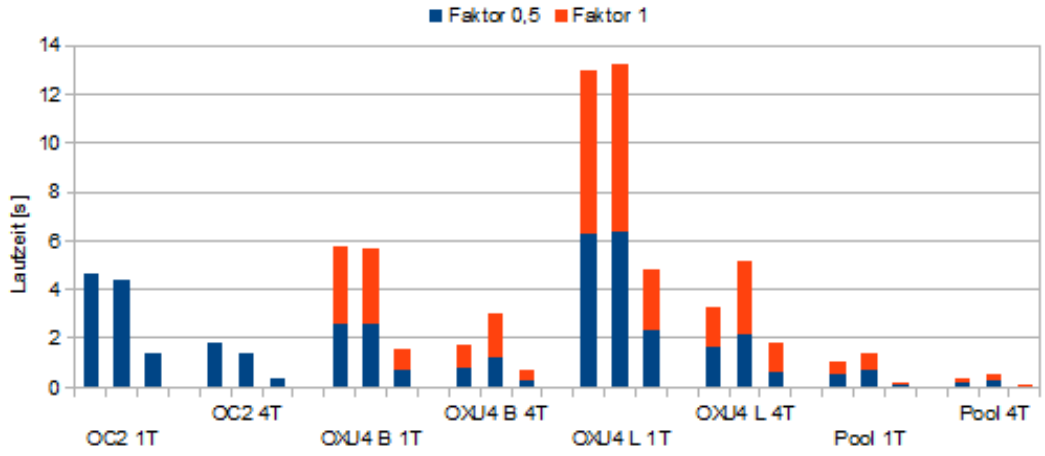

Abbildung 3.1: Laufzeit aller durchgeführten Messungen. Von links nach rechts werden jeweils Join 1 bis 3 aufgelistet. Die „big“-Prozessoren werden über B, die „LITTLE“-Prozessoren über L referenziert. Die Zahl vor dem T gibt die Anzahl Threads an.

3.2 Verbrauch

Das Verbrauchsverhalten von „big“- und „LITTLE“-Kernen zeigt nur geringe Schwankungen bei Equi-Joins. Unabhängig von den gejointen Tabellen verbrauchen die „LITTLE“-Kerne zwar ca. 10 bis 20% mehr Energie wenn nur ein Kern genutzt wird, jedoch benötigen sie 30% weniger wenn sie mit Multi-Threading verwendet werden. Bei Join3 mit Skalierungsfaktor 0,5 verbrauchen sie sowohl mit einem als auch mit vier Threads geringfügig weniger als die „big“-Kerne. Bei Faktor 1 verbrauchen sie allerdings je nach Betriebsart 50% (Single-Threading) bzw. 15% (Multi-Threading) mehr Energie als die „big“-Kerne.

Beim Vergleich von Poolrechner und den „LITTLE“-Kernen zeigt sich, dass der Poolrechner deutlich mehr Energie für die gleichen Operationen verbraucht. Mit größerem Ska-

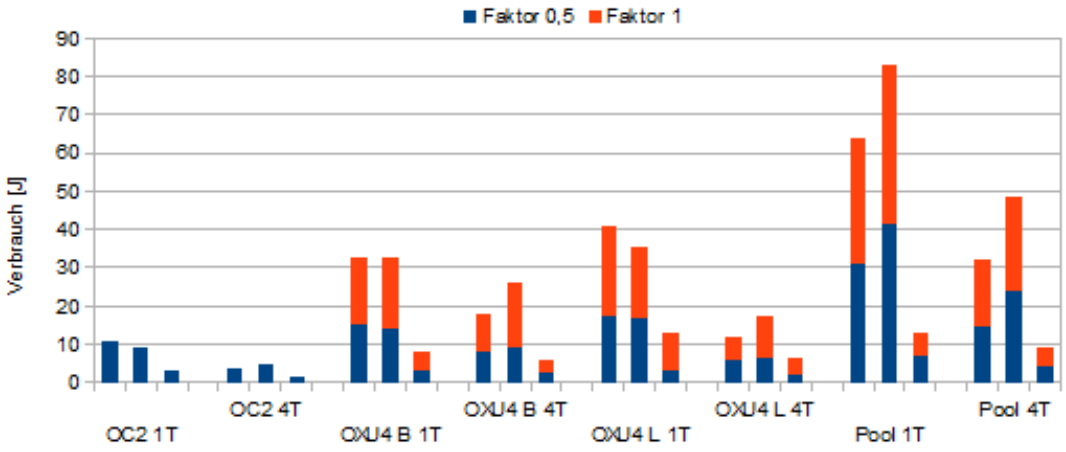

Abbildung 3.2: Energieverbrauch aller durchgeführten Messungen. Von links nach rechts werden jeweils Join 1 bis 3 aufgelistet. Die „big“-Prozessoren werden über B, die „LITTLE“-Prozessoren über L referenziert. Die Zahl vor dem T gibt die Anzahl Threads an.

lierungsfaktor wird der Unterschied jedoch geringer. Bei Join3 und Faktor 1 verbraucht der Poolrechner im Single-Core-Betrieb sogar gleich viel Energie.

Die „big“-Kerne brauchen in allen Messungen weniger Energie als der Poolrechner. Bei Messungen mit einem Thread ist der Unterschied zwischen „big“-Prozessor und Poolsystem größer als im Multi-Threading-Betrieb. Mit größerem Skalierungsfaktor reduziert sich der Unterschied.

Der Odroid C2 verbraucht bei fast allen Joins weniger als die Prozessoren des XU4. Lediglich wenn Join3 mit nur einem Thread bearbeitet wird, verbrauchen der XU4 geringfügig weniger.

Beim Poolrechner fällt auf, dass der Verbrauch um den Faktor 2 ansteigt, wenn der Skalierungsfaktor von 0,5 auf 1 erhöht wird. Beim Odroid XU4 ist der Mehrverbrauch höher und übersteigt in einigen Messungen sogar den Faktor 3.

3.3 Energieeffizienz

Zur Bewertung der Energieeffizienz betrachte ich zwei Größen: Das Energie-Zeit-Produkt (EDP) und die Anzahl der gejointen Tupel pro verbauchtem Joule (fortlaufend abgekürzt als EE2). Beim EDP-Vegleich mit Skalierungsfaktor 0,5 fällt auf, dass die Kerne des Odroid XU4 beim Multi-Threading verhältnismäßig nahe beieinander liegen. Der Odroid C2 zeigt bei allen Joins gleich gute oder bessere Effizienz. Der Poolrechner gleicht seinen höheren Verbrauch durch bessere Laufzeitwerte aus und erhält somit deutlich bessere EDP-Werte als der Rest. Bei Tests mit nur einem Thread verschlechtern sich die Effizienzwerte für alle Geräte um den Faktor 5 und aufwärts, wobei die „LITTLE“-Kerne generell das schlechteste Effizienzverhalten aufweisen. Wenn der Skalierungsfaktor von 0,5 auf 1 erhöht wird, steigt

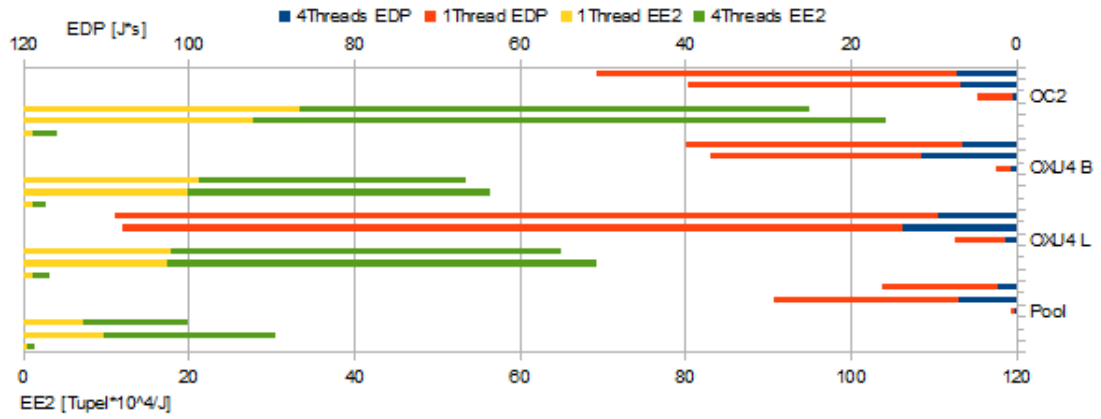

Abbildung 3.3: Energieeffizienzvergleich aller Testgeräte anhand von EDP und gejointen Tupeln pro verbrauchtes J. Von oben nach unten werden jeweils Join 1 bis 3 aufgelistet. Die „big“-Prozessoren werden über B, die „LITTLE“-Prozessoren über L referenziert. Die Ergebnisse sind nur für Skalierfaktor 0,5.

der EDP-Wert des Poolrechners in allen Messungen um ungefähr den Faktor 4 an. Unter diesen Faktor kommt der Odroid XU4 in keiner Messung, sondern verschlechtert sich teilweise um den Faktor 5 und mehr.

Beim Vergleich der EE2-Werte mit Skalierungsfaktor 0,5 ist der Poolrechner hingegen der große Verlierer. Unabhängig von der Anzahl der verwendeten Threads kommen die Einplatinencomputer in allen Messungen auf mindestens doppelt so hohe Effizienzwerte. Während die einzelnen Cluster des Odroid XU4s mit ihren EE2-Werten noch relativ nahe bei einander liegen, kommt der C2 in den meisten Messungen im Vergleich mit dem Poolrechner sogar auf einen Verhältnisfaktor von 3. Beim Skalierungsfaktor 1 bleiben die EE2-Werte des Poolrechners konstant, während der Odroid XU4 an Effizienz verliert. Insbesondere bei den „LITTLE“-Kernen reduzieren sich die EE2-Werte für den Join3 um ca. 50%.

3.4 Fazit

Der Vergleich des Odroid XU4 mit den anderen Testgeräten zeigt definitiv Schwächen der „big“- und „LITTLE“-Kerne auf. Der Laufzeitunterschied von „big“- und „LITTLE“-Kernen verringerte sich wenn Equi-Joins mit größerer R-Tabelle durchgeführt wurden, während die Effizienz der „LITTLE“-Kerne vor allem bei Durchführung von Join3 oft litt. Mühlbauer et al. stellen dar, dass „LITTLE“-Kerne sich besser eignen um große Hashtabellen zu erzeugen, jedoch bei Stringoperationen weniger effizient als „big“- Kerne sind [7]. Dies entspricht meinen Beobachtungen. Generell gilt aus Sicht der Energieeffizienz, dass es sich für den Odroid XU4 nicht lohnt, Joins mit nur einem Thread zu bearbeiten. Bei Verbrauchswerten und Effizienz im Bezug auf EE2 hat der Odroid XU4 deutlich besser abgeschnitten als

der Poolrechner. Allerdings nehmen die Vorteile drastisch ab wenn größere Datenmengen verarbeitet werden. Der Odroid C2 hat im Gesamtvergleich zum XU4 bessere Effizienzwerte gezeigt und liegt bei der Laufzeitanalyse zwischen den „big“- und „LITTLE“-Kernen.

Kapitel 4

Verwandte Arbeiten

Mühlbauer et al.[8] sehen in der Verwendung von heterogenen Prozessorarchitekturen einen Lösungsansatz um einen Teil der Einschränkungen zu umgehen, die bei der Schalierung von Prozessorleistung auftreten. Sie führen erfolgreiche Schritte zur Optimierung der Energieeffizenz von big.LITTLE-Prozessoren, bei der Durchführung von Datenbankoperationen, durch. Im Angesicht der zunehmenden Einschränkungen von Chipgröße und Wärmeabstrahlung weisen Mühlbauer et al. darauf hin, dass ein Cluster von vier „LITTLE"-Kernen, in allen Messungen, bessere Laufzeit- und Energieverbrachswerte als ein einzelner „big"-Kern erziehlt. Dabei nehmen die vier „LITTLE"-Kerne weniger Platz auf der Platine ein als der „big"-Kern. Dass bessere Laufzeit- und Energieverhalten der „LITTLE"-Kerne konnte ich in meinen Messungen bestätigen. Jedoch wurde meine Herstelleranfrage bezüglich der Abmaße der unterschiedlichen Kerne meines Prozessors ignoriert. Das Größenverhältnis sollte sich jedoch nicht zu stark unterscheiden, da Mühlbauer et al. einen Vorgänger meines Exynos5422 verwenden.

Cheng et al.[6] untersuchen einen alternativen Lösungansatz zur Verbesserung der Energieeffizienz von Datenbankoperationen auf eingebetteten Systemen. Sie nutzen eine Onboard-GPU als Coprozessor für einen ARM Cortex-A9 und stellen fest, dass die GPU für bestimmte Operationen, wie z. B. Sorts oder Hash-Joins, deutlich effizienter als der Multi-Core-Prozessor läuft. Bei optimierter Lastverteilung im kombinierten Betrieb von GPU und CPU können somit bessere Effizienzwerte als bei einer herkömmlich Workstation mit Multi-Core-Prozessor erzielt werden.

Kapitel 5

Schlusswort

Auch schon ohne die Verwendung fortgeschrittener Schedulingverfahren wie Global-Task-Switching zeigt der Odroid XU4 Potential, was die energieeffiziente Ausführung von Hash-Joins betrifft. Zwar weist der XU4 im Vergleich zum C2 schlechtere Energieeffizienzwerte auf, bietet durch seine heterogene Prozessorarchitektur jedoch zusätzliches Optimierungspotential.

Verglichen mit der Workstation bietet diese deutlich mehr Rechenleistung bei einer ebenso deutlich schlechteren Energieeffizienz. Ein Ansatz eine ähnliche Rechenleistung wie eine Workstation unter Ausnutzung des geringeren Energieverbrauchs des XU4 zu erreichen, wäre Clustering. Allerdings müsste geprüft werden inwiefern sich der Overhead des Clusterings auf die Energieeffizienz auswirkt. Die Erforschung dieser Möglichkeiten sei jedoch anderen überlassen.

Anhang A

Gesamtübersicht der Messergebnisse

Die folgende Tabelle enthält eine Aufstellung aller aufgenommenen Messwerte und berechnete Energieeffizienzwerte. T1 und T2 geben die Anzahl der eingelesenen Tupel an. J steht für die Anzahl der durchgeführten Joins.

Messergebnisse

Join1	Faktor 0,5	T1=100000	T2=2999671	J=2999671			Faktor 1	T1=200000	T2=6001215	J=6001215		
	Laufzeit [s]	Verbrauch [J]	Build [s]	Probe [s]	EDP [J*s]	EE2 [Tupel/J]	Laufzeit [s]	Verbrauch [J]	Build [s]	Probe [s]	EDP [J*s]	EE2 [Tupel/J]
OC2 1T	4,68	10,86	0,14	4,54	50,82	276212,8						
OC2 4T	1,86	3,92	0,038	1,822	7,29	765222,2						
OXU4 B 1T	2,64	15,1	0,1	2,54	39,86	198653,7	5,79	32,87	0,19	5,6	190,32	182574,2
OXU4 B 4T	0,79	8,22	0,046	0,744	6,49	364923,5	1,77	18	0,09	1,68	31,86	333400,8
OXU4 L 1T	6,33	17,2	0,25	6,08	108,88	174399,5	13	40,98	0,49	12,51	532,74	146442,5
OXU4 L 4T	1,63	5,78	0,07	1,56	9,42	518974,2	3,3	12,15	0,14	3,16	40,10	493927,2
Pool 1T	0,53	30,793	0,017	0,513	16,32	97414,1	1,08	63,936	0,035	1,045	69,05	93862,8
Pool 4T	0,16	14,48	0,007	0,153	2,32	207159,6	0,35	32,025	0,014	0,336	11,21	187391,6

Join2	Faktor 0,5	T1=750000	T2=2999671	J=2999671			Faktor 1	T1=1500000	T2=6001215	J=6001215		
	Laufzeit [s]	Verbrauch [J]	Build [s]	Probe [s]	EDP [J*s]	EE2 [Tupel/J]	Laufzeit [s]	Verbrauch [J]	Build [s]	Probe [s]	EDP [J*s]	EE2 [Tupel/J]
OC2 1T	4,4	9,01	1,11	3,29	39,64	332926,9						
OC2 4T	1,41	4,86	0,3	1,11	6,85	617216,3						
OXU4 B 1T	2,6	14,21	0,85	1,75	36,95	211095,8	5,66	32,67	1,74	3,92	184,91	183691,9
OXU4 B 4T	1,25	9,29	0,41	0,84	11,61	322892,5	3	26,25	0,88	2,12	78,75	228617,7
OXU4 L 1T	6,41	16,85	1,91	4,5	108,01	178022,0	13,22	35,32	4,04	9,18	466,93	169909,8
OXU4 L 4T	2,15	6,36	0,6	1,55	13,67	471646,4	5,16	17,36	1,7	3,46	89,58	345692,1
Pool 1T	0,71	41,251	0,17	0,54	29,29	72717,5	1,43	83,083	0,37	1,06	118,81	72231,6
Pool 4T	0,29	24,041	0,07	0,22	6,97	124773,1	0,57	48,507	0,14	0,43	27,65	123718,5

Join3	Faktor 0,5	T1=100000	T2=2999671	J=37909			Faktor 1	T1=200000	T2=6001215	J=75983		
	Laufzeit [s]	Verbrauch [J]	Build [s]	Probe [s]	EDP [J*s]	EE2 [Tupel/J]	Laufzeit [s]	Verbrauch [J]	Build [s]	Probe [s]	EDP [J*s]	EE2 [Tupel/J]
OC2 1T	1,4	3,39	1,17	0,23	4,75	11182,6						
OC2 4T	0,4	1,28	0,33	0,07	0,51	29616,4						
OXU4 B 1T	0,75	3,22	0,59	0,16	2,42	11773,0	1,57	8,24	1,32	0,25	12,94	9221,2
OXU4 B 4T	0,32	2,32	0,249	0,071	0,74	16340,1	0,71	5,7	0,59	0,12	4,05	13330,4
OXU4 L 1T	2,36	3,14	1,98	0,38	7,41	12072,9	4,83	12,9	4,31	0,52	62,31	5890,2
OXU4 L 4T	0,65	1,96	0,522	0,128	1,27	19341,3	1,8	6,55	1,59	0,21	11,79	11600,5
Pool 1T	0,11	6,875	0,085	0,025	0,76	5514,0	0,22	13,024	0,166	0,054	2,87	5834,1
Pool 4T	0,05	4,255	0,041	0,009	0,21	8909,3	0,11	9,361	0,077	0,033	1,03	8117,0

Abbildungsverzeichnis

Literaturverzeichnis

[1] *big.LITTLE Technology - ARM.* `https://www.arm.com/products/processors/technologies/biglittleprocessing.php`. Besucht am 17. Juni 2017.

[2] *Hardkernel Ltd. ODROID-C2.* `http://www.hardkernel.com/main/products/prdt_info.php?g_code=G145457216438`. Besucht am 17. Juni 2017.

[3] *Hardkernel Ltd. ODROID-XU4.* `http://www.hardkernel.com/main/products/prdt_info.php?g_code=G143452239825&tab_idx=1`. Besucht am 17. Juni 2017.

[4] *Smartphones - Nutzer weltweit 2012-2020 Prognose.* `https://de.statista.com/statistik/daten/studie/309656/umfrage/prognose-zur-anzahl-der-smartphone-nutzer-weltweit/`. Besucht am 17. Juni 2017.

[5] *TPC-H - Homepage.* `http://www.tpc.org/tpch/`. Besucht am 17. Juni 2017.

[6] Cheng, X., B. He und C. T. Lau: *Energy-Efficient Query Processing on Embedded CPU-GPU Architectures.* DaMoN '15, Seiten 1,5, 2015.

[7] Muehlbauer, T., W. Ruediger, R. Seilbeck, A. Kemper und T. Neumann: *Heterogeneity-Conscious Parallel Query Execution: Getting a better mileage while driving faster!* DaMoN '14, Seite 3, 2014.

[8] Muehlbauer, T., W. Ruediger, R. Seilbeck, A. Kemper und T. Neumann: *Heterogeneity-Conscious Parallel Query Execution: Getting a better mileage while driving faster!* DaMoN '14, Seiten 1,4, 2014.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 18. Juni 2017

Michael Hein