Performanzanalysen von Smart Systems auf Systemebene

Florian Unterstein
Missing Link Electronics

Abstract

Es werden Tools und Methoden vorgestellt, die es erlauben unter Linux die Rechenleistung und Speicherperformanz (über die gesamte Speicherhierarchie inklusive Caches) von Embedded Systems zu messen. Besonderer Wert wurde darauf gelegt, unverzerrt von der Softwareumgebung (Distribution, Kernel Version etc.) messen zu können. Somit ist ein systemübergreifender Benchmark möglich. In diesem Paper wird erläutert, wie für diesen Zweck die frei verfügbaren Tools CoreMark und LMbench eingesetzt werden können. Ergänzend werden Messergebnisse von TIs OMAP 4, Xilinx Zynq und Nvidias Tegra Plattform präsentiert.

1 Einleitung

Leistungsfähige System on Chip auf ARM Basis dominieren derzeit den Markt für Embedded Anwendungen, besonders der ARM Cortex™-A9 MPCore. Für welchen soll man sich aber entscheiden, wenn man ein System aufbaut? Ist A9 gleich A9? Diese Fragen zu klären ist ein wichtiger erster Schritt in jedem Designprozess, aber nur anhand von Datenblättern nicht ausreichend zu evaluieren. Es bedarf Methoden und Tools, um die Performanz vergleichen zu können. In realen Anwendungen kommt erschwerend hinzu, dass in der Linux Welt die unterschiedlichen Softwareumgebungen unterschiedliche Effekte aufweisen. Ergebnisse sollten aber über verschiedene Systeme hinweg vergleichbar sein, unabhängig von Distribution, Kernel Version oder Compiler. Dieses Paper stellt eine Auswahl an Analyse Tools vor, mit denen es möglich ist, weitgehend unverzerrt von der Softwarekonfiguration unter Linux die Rechenleistung und Speicherperformanz zu untersuchen. Anhand einiger Beispielsysteme wird der praktische Einsatz demonstriert.

2 Benchmarking unter Linux

Nur selten laufen Anwendungen direkt auf dem Prozessor, sondern werden in den meisten Fällen im Kontext eines Betriebssystems ausgeführt. Linux bildet durch seinen umfassenden Software Stack eine hervorragende Grundlage zum effizienten Aufbau von Embedded Systems. Des Weiteren ist es sehr portabel und auf vielen Systemen bereits verfügbar, weswegen es als Betriebssystem für die vorgestellten Benchmarks verwendet wird.
Die Rechenleistung ist im Wesentlichen durch die Taktrate und die ausführbaren Instruktionen pro Takt bestimmt. Daneben spielt die Speicheranbindung eine entscheidende Rolle. Es ist unrealistisch, davon auszugehen, dass der Prozessor jederzeit $Taktrate * Instruktion/Takt$ Anweisungen pro Sekunde abarbeiten kann. In typischen Anwendungen kommt es durch Speicherzugriffe durchaus zu Wartezyklen der CPU. Dieses Problem wird verschärft durch den sogenannten *memory gap*: die Taktraten der CPUs steigen viel schneller als die Geschwindigkeit der Speicher, wodurch die Performance Diskrepanz zwischen den beiden immer größer wird. Daher können Speicherlatenz und -durchsatz zum Flaschenhals eines Systems werden.

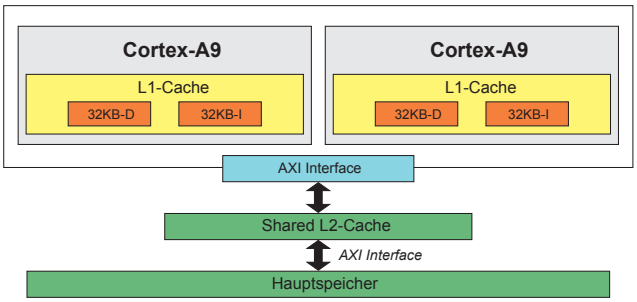

Abbildung 1: Schematischer Aufbau eines Cortex-A9 MPCore

Typischerweise ist der Systemspeicher hierarchisch aufgebaut. Der kleinste und schnellste (und somit auch teuerste) Speicher ist am nächsten an der CPU angebunden. Danach folgen weitere Level mit steigender Kapazität, aber mit niedrigerer Geschwindigkeit. In Abbildung 1 ist schematisch der Aufbau eines ARM Cortex-A9 Dual-Core Prozessors dargestellt. In dieser Konfiguration besitzt jeder Kern einen dedizierten L1-Cache als erste Speicherstufe, zusätzlich gibt es einen Shared Memory L2-Cache den sich beide Kerne teilen. Über einen Bus (hier: AXI Interface) kann schließlich der Hauptspeicher adressiert werden. [1]

3 Einführung in die Tools

3.1 CoreMark

CoreMark ist ein CPU Benchmark, der den CPU Kern mit typischer Last wie Schreib- und Leseoperationen, Integerberechnungen, Operationen auf Listen, Matrixoperationen und CRC-Checks testet. Entwickelt wird er von dem Embedded Microprocessor Benchmark Consortium (EEMBC), zu dem neben vielen anderen ARM, Analog Devices, Intel, VW und Xilinx gehören. [8]
Ziel ist eine einzelne Zahl zur einfachen Bewertung einer CPU zu erhalten, die die Leistung unter realistischer Last wiedergibt. Dies wird dadurch erreicht, dass keine rein synthetischen Berechnungen durchgeführt werden. Stattdessen werden Datenstrukturen und Algorithmen eingesetzt, die in vielen Anwendungsfällen üblich sind. Besonderer Wert wurde darauf gelegt, dass die Rechenlast nicht vom Compiler weg optimiert werden kann. Zwar haben die Compilerversion und -flags Einfluss auf das Ergebnis, aber durch die Bestimmung der Operationen aus dynamischen Daten ist eine Optimierung der Last zur Compilezeit nicht möglich. Um auch Multicore-Prozessoren auslasten zu können, ist es möglich über Compilerflags mehrere Threads bzw. Prozesse zu nutzen. Hierbei ist allerdings zu beachten, dass keinerlei Kopplung zwischen den Threads besteht. Im Normalfall ergibt sich deshalb ein linearer Anstieg der erreichten CoreMarks, was aufgrund von Datenabhängigkeiten und Synchronisierungsproblemen nicht typisch für reale Anwendungen ist.
Die Resultate können auf der Seite des EEMBC eingereicht und veröffentlicht werden. Reproduzierbare Ergebnisse werden durch das EEBMC zertifiziert. In Kombination mit der freien Verfügbarkeit des Quellcodes wird dadurch Manipulation verhindert und eine vertrauenswürdige Bewertung ermöglicht.
Für alle durchgeführten Messungen wurden ausschließlich POSIX Threads genutzt. Der Aufruf um mit *make* zu Kompilieren lautet:

```
$ make compile PORT_DIR=linux\
  XCFLAGS="-DMULTITHREAD=N -DUSE_PTHREAD"
```

wobei *N* die Anzahl der gewünschten Threads ist. Das Target *compile* bewirkt, dass das Programm nur kompiliert und nicht sofort gestartet wird. Insbesondere in Verbindung mit einem Cross-Compiler ist dies besonders nützlich. *PORT_DIR* verweist auf ein Verzeichnis, in dem für die Portierung auf neue Systeme Wrapper definiert werden können, die es z.B. ermöglichen, andere Funktionen zur Zeitbestimmung anzugeben. Da alle Tests auf einem 32-Bit Linux ausgeführt werden, wird hier der bereits mitgelieferte Ordner *linux* gewählt. Wie bereits erwähnt werden die internen Datenstrukturen mit Seeds zur Laufzeit initialisiert. Die Aufrufparameter für einen Performance-Run sind fest vorgegeben:

```
$ coremark 0x0 0x0 0x66 0 7 1 2000
```

In der Ausgabe sind neben den CoreMarks auch der verwendete Compiler samt Flags enthalten.
Wenn die Taktrate eines Prozessors erhöht wird, skaliert die Taktrate des Speichers nicht immer 1:1 mit. Durch den dadurch in Relation zum Prozessor langsameren Speicherzugriff kann es vermehrt zu Wartezyklen kommen, die wiederum die Gesamtperformance negativ beeinflussen. Um solche Skalierungseffekte zu erkennen, ist es hilfreich zusätzlich die CoreMark/MHz zu betrachten. [9]

3.2 LMbench

LMbench ist eine Sammlung einfacher, portabler Benchmark Tools die von Larry McVoy und Carl Staelin entwickelt und betreut wird. Sie steht unter der GNU GPL und ist somit frei verfügbar. LMbench besteht aus spezialisierten, kleinen Programmen, die entwickelt wurden um die Kommunikation zwischen CPU, Caches und Hauptspeicher zu untersuchen und Flaschenhälse aufzuspüren. Im Folgenden wird der Durchsatz und die Latenz der gesamten Speicherhierarchie untersucht, deshalb werden nur die dafür zuständigen Programme im Detail vorgestellt. Benutzt wurde die Version 3.

lat_mem_rd
Mit dem Tool *lat_mem_rd* wird die Latenz bei Lesezugriffen auf den Speicher gemessen. Der Benchmark besteht aus zwei verschachtelten Schleifen, von dem die Äußere die Schrittweite (Stride) und die Innere die Arraygröße variiert. Das Array bildet einen Ring aus Zeigern, die jeweils eine Schrittweite zurück zeigen. In der Schleife wird dann das Zeigerarray durchschritten und dabei die Zugriffszeit aus vielen Durchgängen bestimmt.
Durch Variation der Strides lassen sich Trade-offs im Cache Design aufzeigen: Große Cachezeilen erreichen zwar durch Pre-fetching kleine Latenzen bei Daten mit hoher Lokalität, dafür werden im anderem Fall mehr unnötige Daten in den Cache geladen. Viele Architekturen implementieren zudem einen *critical word first* Mechanismus, der den angeforderten Teil der Cachezeile direkt an die CPU weiterleitet und erst danach die gesamte Zeile in den Cache schreibt. Erfolgt sofort ein weiterer Cache Miss, blockiert der Vorgang bis die Übertragung abgeschlossen ist. Durch diesen Effekt liegt die wahrgenommene Latenz bei Bursts von Misses über der Latenz eines isolierten Zugriffs. Die so ermittelte Latenz nennt man *Back-to-Back Latency*.
Die zurückgegebene Zeit ist die reine Latenz eines Speicherzugriffs. Das bedeutet unter der Annahme, dass eine *load* Instruktion in einem Takt ausgeführt wird, dass von der gesamten Zugriffszeit eine Taktdauer für das Ausführen dieser Instruktion abgezogen wird. Theoretisch wären also Latenzzeiten von 0 ns möglich, wenn das Ergebnis direkt nach dem Takt anliegen würde, in dem der Befehl abgesetzt wurde. [10]

Mit folgendem Befehl wird eine Messung ausgeführt:

```
$ taskset 0x1 lat_mem_rd -N 1 -P 1 size stride
```

taskset 0x1 bindet die Ausführung von *lat_mem_rd* an den CPU Kern 0, da für die Latenzmessung kein Multitasking erwünscht ist. Die Arraygröße wurde für alle Tests auf 128 MB gesetzt. Wichtig ist, dass sie größer als die L2 Cachegröße gewählt wird, um auch den Hauptspeicher abzudecken. *stride* sollte nicht kleiner als eine Cache Zeile gewählt werden, da sich ansonsten die Cache Zeile selbst wie ein Pre-fetching Mechanismus verhält und mehrere Werte auf einmal übertragen werden. In Abbildung 2 ist dies für die Schrittweite von 16 Byte und einer Cache Zeilengröße von 32 Byte zu sehen. Für 16 Byte Strides ist die gemessene Latenz wegen des genannten Effekts in etwa halb so groß wie ein Zugriff in Wirklichkeit benötigt.
Die Ausgabe besteht aus der Arraygröße in MB und der zugehörigen Latenz. Für die Auswertung empfiehlt es sich, die Werte mit dem Logarithmus der Arraygröße als x-Achse zu Plotten. Wie in Abbildung 2 zu sehen, ergeben sich mehrere Plateaus mit gleichen Latenzen. Diese korrespondieren mit der Speicherhierarchie und der Größe der L1- und L2-Caches.
Die Latenz hängt unter anderem auch vom CPU Takt ab. Um tieferen Einblick in die Effizienz der Anbindung unabhängig von der Taktung zu erlangen, kann die Latenz auf CPU Zyklen normiert werden:
$Latenz(CPUTakten) = Latenz(s) * Taktrate(Hz)$

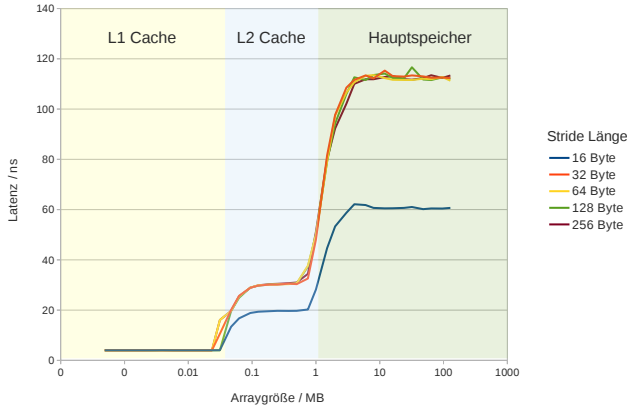

Abbildung 2: Speicherlatenz für verschiedene Strides

bw_mem
Um die Speicherbandbreite zu bestimmen, bietet LMbench das Tool *bw_mem* an. Damit können insgesamt neun verschiedene Arten von Speichertransfers durchgeführt und gemessen werden, im Folgenden werden nur der einfache *read* und *write* Modus genutzt. Das Programm allokiert die doppelte Menge der übergebenen Speichergröße und initialisiert diese auf Null. Anschließend wird die benötigte Zeit gemessen, um aus der ersten Hälfte in die Zweite zu kopieren.
Die Option *-N* legt die Iterationen fest, *-P* die parallelen Threads:

```
$ bw_mem -N x -P y size rd|wr
```

Um einzelne Speicherbereiche zu messen, muss die Speichergröße *size* so gewählt werden, dass sie den gewünschten Bereich nicht überschreitet. Für den maximalen Durchsatz auf Mehrkernprozessoren muss zudem mit mehreren Threads gearbeitet werden. Besonders zu beachten ist dabei die Speicherarchitektur, da sich Threads möglicherweise gemeinsame Caches teilen. Angenommen ein Dual-Core Prozessor besitzt 32KB L1-Cache und 1MB shared L2-Cache und es soll der L1-Cache getestet werden. Dann werden zwei Prozesse mit je der halben Cachegröße benötigt, da *bw_mem* doppelt so viel Speicher belegt wie übergeben wird. Der passende Beispielaufruf ist:

```
$ bw_mem -N 1 -P 2 16K rd
```

Soll dagegen der L2-Cache getestet werden, so wird zunächst der 1MB Speicher zwischen den beiden Kernen geteilt, da der Cache shared ist. Die Threads bekommen dann jeweils die Hälfte der 512KB pro Kern zugeteilt:

```
$ bw_mem -N 1 -P 2 256K rd
```

Die Anzahl der Iterationen wurde für dieses Beispiel auf 1 gesetzt.
Auch hier lassen sich in Abbildung 3 für die Lesegeschwindigkeit die Caches gut erkennen. Für einen aussagekräftigen Schreibtest ist es wichtig, dass die Schreibstrategie des Cache passend konfiguriert ist. In Abbildung 3 ist ein Fall aufgezeigt, in dem der Cache als *write-through* eingestellt ist und somit bei jedem Schreibzugriff in den übergeordneten Speicher schreibt. Dadurch wird immer auf den Hauptspeicher zugegriffen und die Caches sind nicht erfassbar.

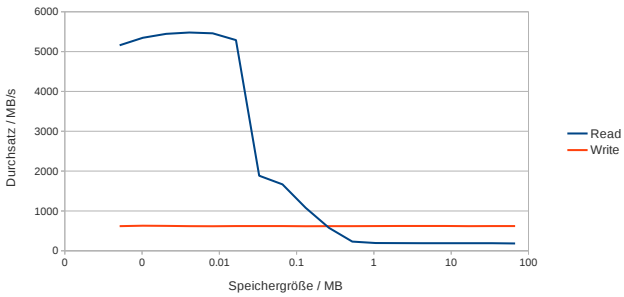

Abbildung 3: Lese- und Schreibdurchsatz für write-through Caches

4 Messungen

4.1 Testsysteme

Abschließend werden Messergebnisse einiger ausgewählter ARM sowie eines x86 Testsystems vorgestellt. Auf allen

Systemen mit Ausnahme des Toradex Boards kam Ubuntu 12.04 zum Einsatz. Auf dem Toradex Colibri T20 lief ein angepasstes Angstrom Linux, dass vom Hersteller bereit gestellt wurde.

PandaBoard Das PandaBoard [4] ist ein Community getriebenes Entwicklungsboard und setzt auf einem OMAP4430 Chip von Texas Instruments [5] auf.

SoC	TI OMAP4430
Prozessor	Dual-Core ARM Cortex-A9
Taktrate	1GHz
L1 Cache	32KB I / 32KB D
L2 Cache	1MB shared
Hauptspeicher	1GB DDR2

ISEE IGEPv2 Das IGEPv2 Board von ISEE [3] ist ein Einplatinen-Computer für den Einsatz im industriellen Bereich. Das Board kann im erweiterten Temperaturbereich von -40° C bis +80° C betrieben werden und basiert auf dem OMAP3530 SoC.

SoC	TI OMAP3550
Prozessor	ARM Cortex-A8
Taktrate	720MHz
L1 Cache	32KB I / 32KB D
L2 Cache	256KB
Hauptspeicher	512MB DDR

Toradex Colibri T20 Das Colibri T20 Computer Modul von Toradex [6] ist ein Steckmodul, das über keine eigenen I/O Konnektoren verfügt. Über eine Schnittstelle kann es mit verschiedenen Carrier Boards von Toradex betrieben werden. Das Testmodul wurde auf einem Iris Carrier Board [7] installiert, das alle grundlegenden Anschlüsse wie USB, RJ45 Ethernet und DVI zur Verfügung stellt.

SoC	Nvidia Tegra2
Prozessor	Dual-Core ARM Cortex-A9
Taktrate	1GHz
L1 Cache	32KB I / 32KB D
L2 Cache	1MB shared
Hauptspeicher	512MB DDR2

Xilinx Zynq-7000 Evaluation Board Bei dem Zynq-7000 von Xilinx handelt es sich um eine Mischform aus FPGA und SoC mit ARM Cortex-A9 Prozessor. Xilinx geht dabei einen neuen Weg und stellt nicht das FPGA in den Vordergrund und integriert den Prozessor, sondern konzipierte die Plattform Prozessor zentrisch. Dadurch erscheint es für Software Entwickler wie ein gewöhnlicher ARM Mikroprozessor, bietet aber die Möglichkeit, über den programmierbaren Logikteil der Hardware die Flexibilität eines FPGA zu nutzen.

SoC	Xilinx Zynq-7000
Prozessor	Dual-Core ARM Cortex-A9
Taktrate	666MHz
L1 Cache	32KB I / 32KB D
L2 Cache	512KB shared
Hauptspeicher	1GB DDR3

Asus Eeepc 1000H Das 2008 erschienene Netbook Eeepc 1000H [2] von Asus ist das einzige vollwertige Notebook mit x86 Architektur im Vergleich. Dennoch bietet sich an, es für einen Architkturvergleich heranzuziehen, da das Konzept der Netbooks ähnliche Ziele verfolgt, wie die anderen verwendeten Plattformen: Mobilität und Energieeffizienz.

Prozessor	Intel Atom N270
Taktrate	1.6GHz
L1 Cache	32KB I / 24KB D
L2 Cache	512KB
Hauptspeicher	1GB DDR2

4.2 Ergebnisse

Die erreichten CoreMarks in absoluten Werten und auf MHz normiert sind in Abbildungen 4 und 5 dargestellt. In den folgenden Diagrammen 6 und 7 sind der Lese- und Schreibduchsatz im Vergleich dargestellt. Die Speicherlatenz ist zunächst pro System für verschiedene Strides (Abb. 8 bis 12) und in Abbildung 13 bzw. 14 im Vergleich aufgeführt.

Literatur

[1] Arm cortex-a9 mpcore. http://www.arm.com/products/processors/cortex-a/cortex-a9.php.

[2] Asus eee pc 1000h. http://www.asus.com/Eee/Eee_PC/Eee_PC_1000H.

[3] Isee igepv2. http://www.isee.biz/products/processor-boards/igepv2-board.

[4] Pandaboard. http://pandaboard.org/.

[5] Texas instruments omap4430. http://www.ti.com/product/omap4430.

[6] Toradex colibri t20. http://www.toradex.com/node/906/Modules/Colibri-T20.

[7] Toradex iris carrier board. http://www.toradex.com/node/863/Colibri-Computer-On-Module-Carrier-Boards/Iris.

[8] EMBEDDED MICROPROCESSOR BENCHMARK CONSORTIUM. Coremark. http://www.coremark.org.

[9] GAL-ON, S., AND LEVY, M. Exploring coremark – a benchmark maximizing simplicity and efficacy. http://www.eembc.org/techlit/whitepaper.php#coremark.

[10] MCVOY, L., AND STAELIN, C. lmbench: Portable tools for performance analysis. http://www.bitmover.com/lmbench/lmbench-usenix.pdf.

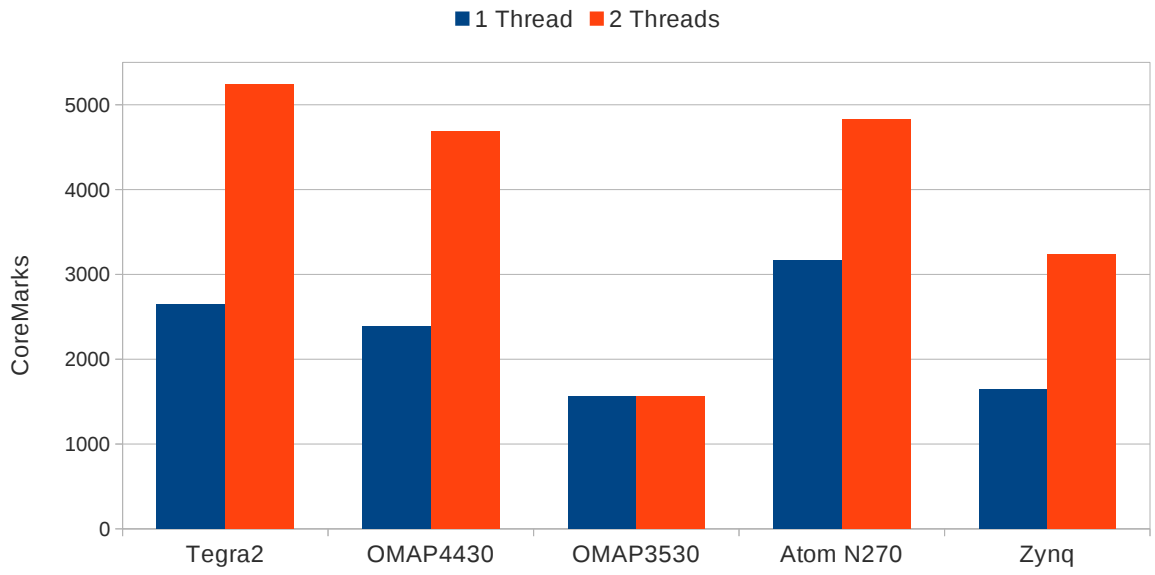

Abbildung 4: Vergleich CoreMarks

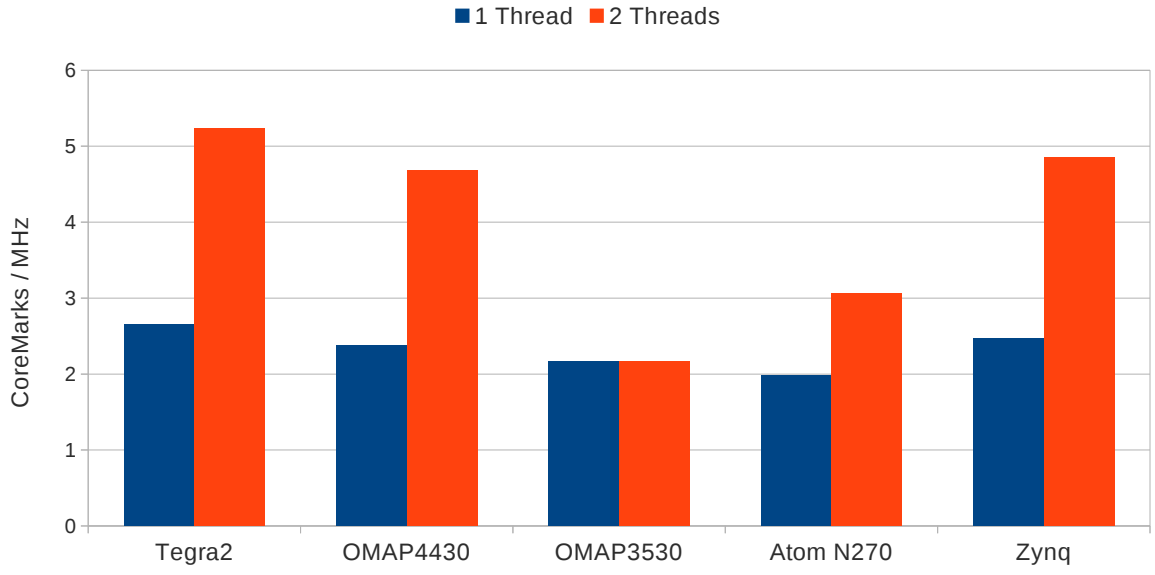

Abbildung 5: Vergleich CoreMarks / MHz

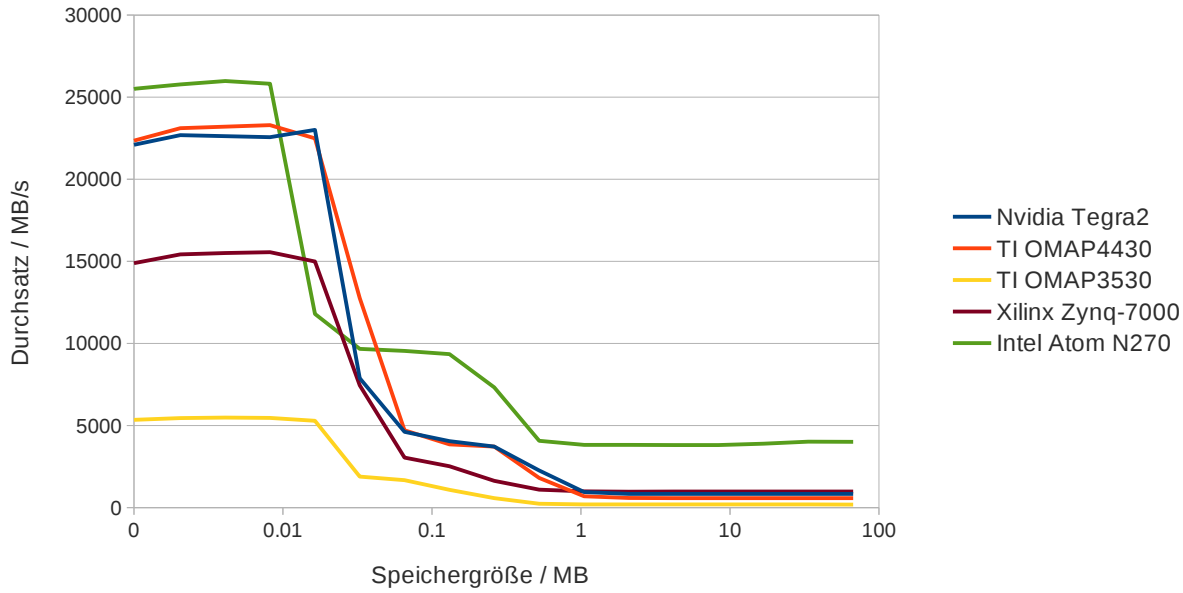

Abbildung 6: Vergleich Speicher Lesedurchsatz

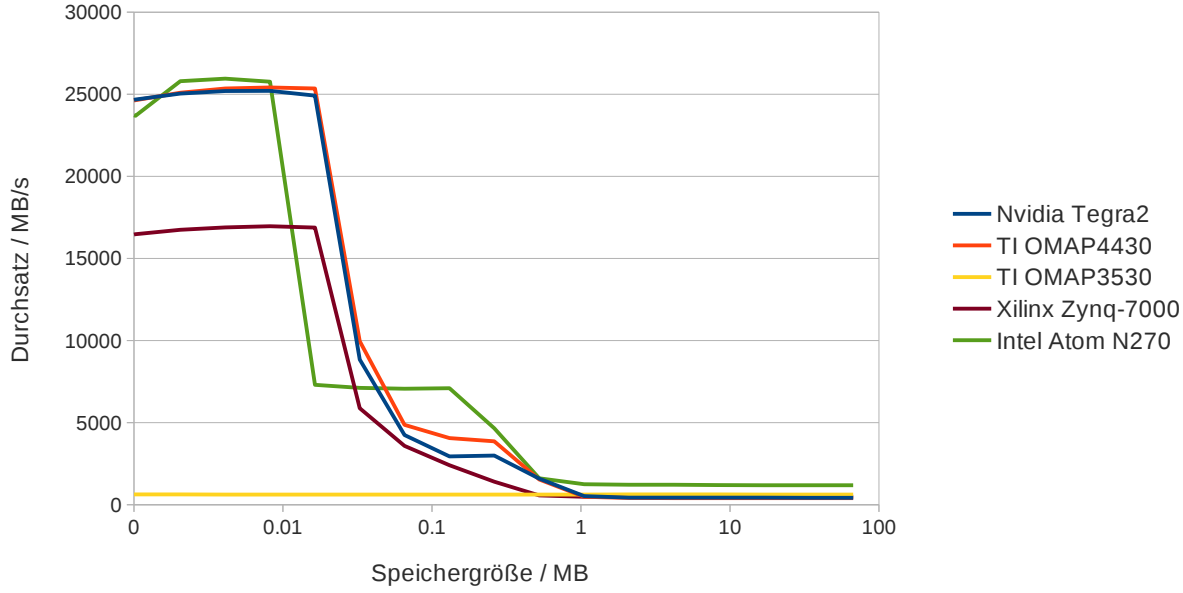

Abbildung 7: Vergleich Speicher Schreibduchsatz

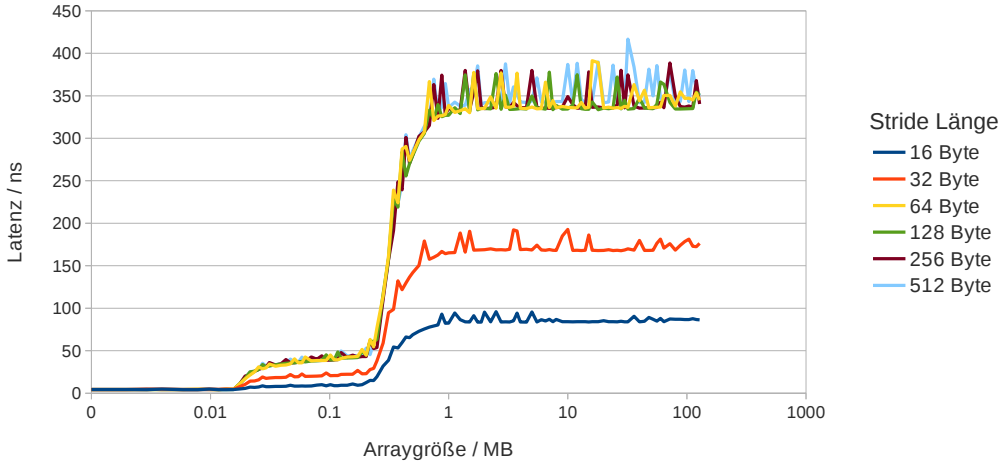

Abbildung 8: Speicherlatenz TI OMAP3550 (ISEE IGEPv2)

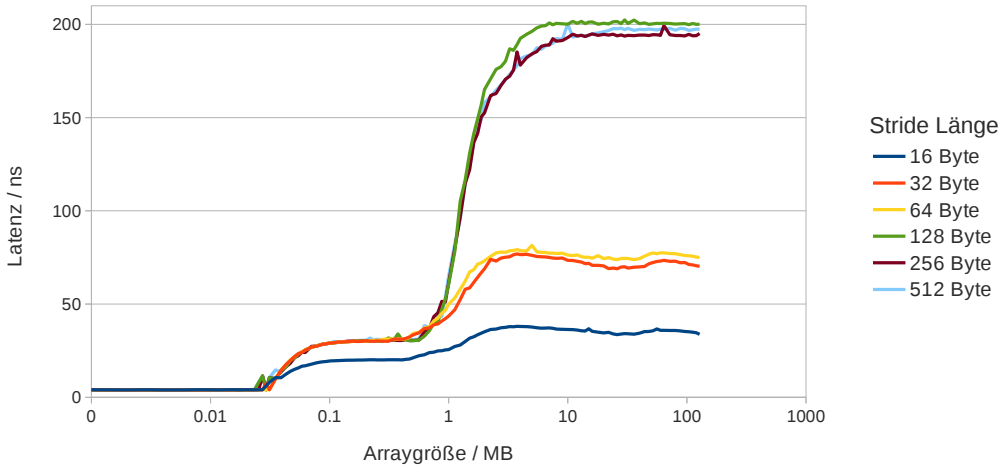

Abbildung 9: Speicherlatenz TI OMAP4430 (PandaBoard)

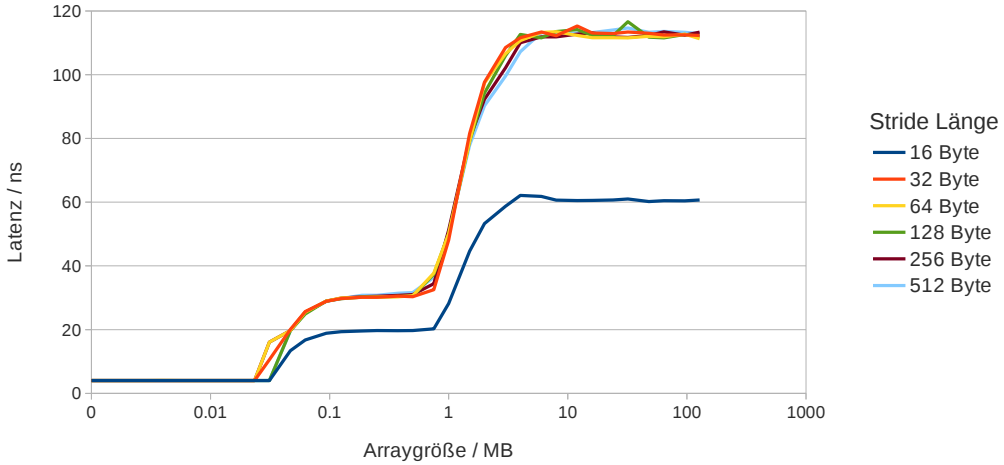

Abbildung 10: Speicherlatenz Nvidia Tegra2 (Toradex Colibri T20)

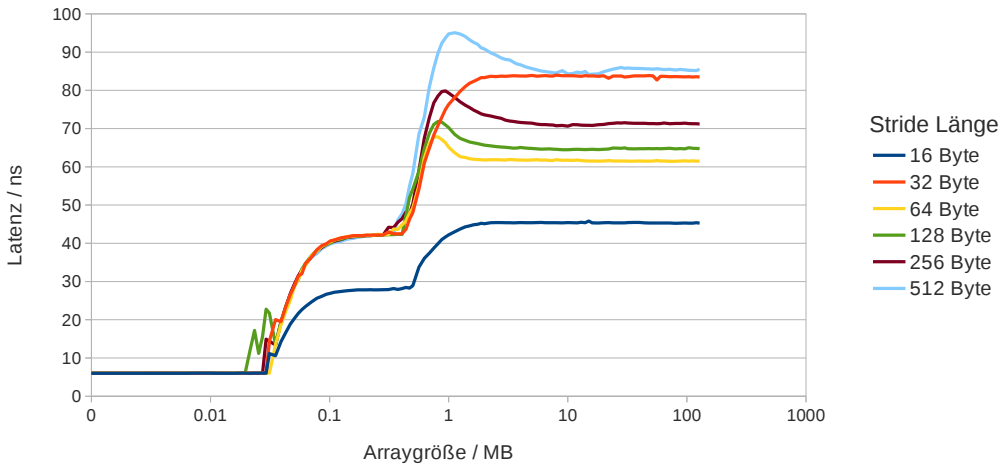

Abbildung 11: Speicherlatenz Xilinx Zynq-7000

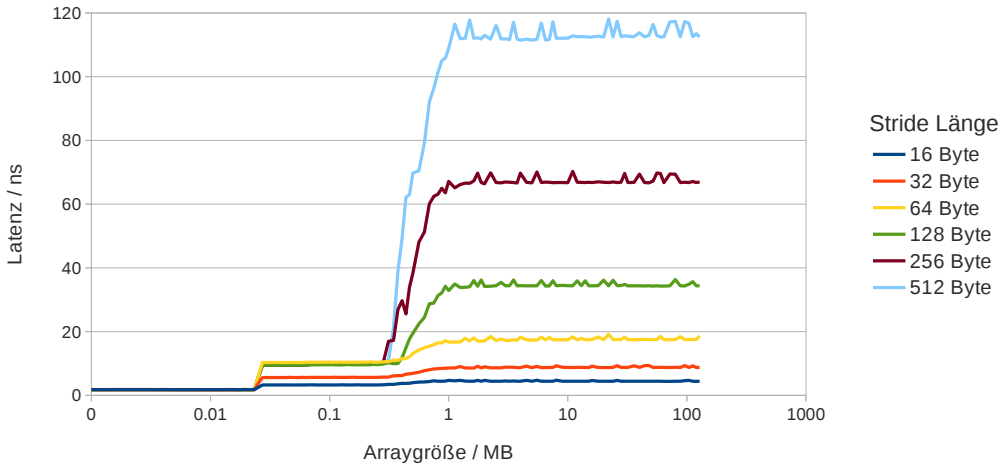

Abbildung 12: Speicherlatenz Intel Atom N270 (Asus Eeepc)

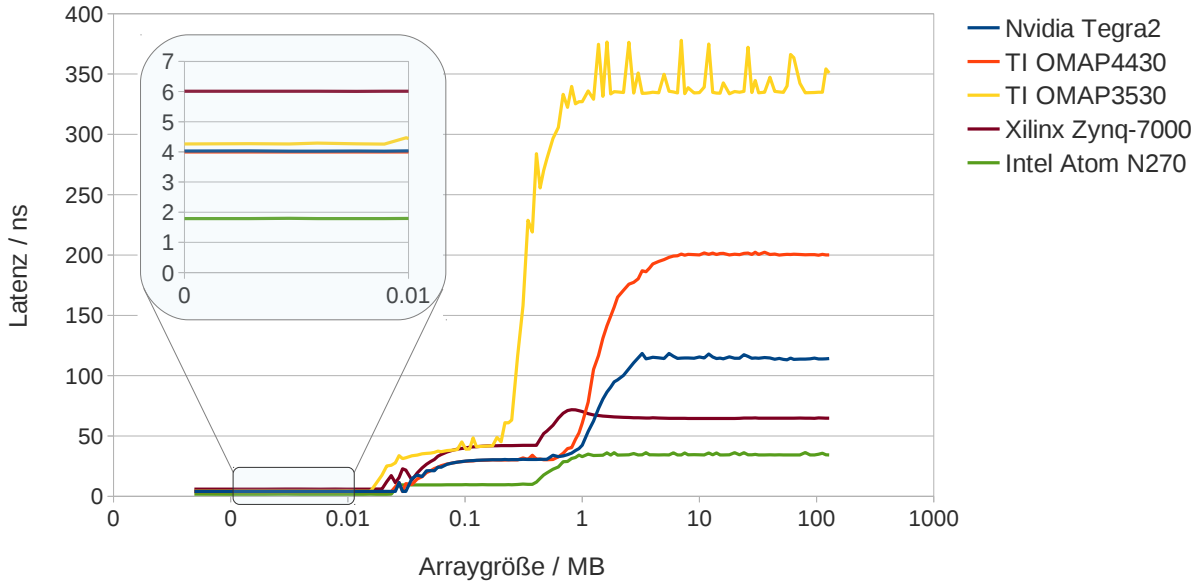

Abbildung 13: Vergleich Speicherlatenz

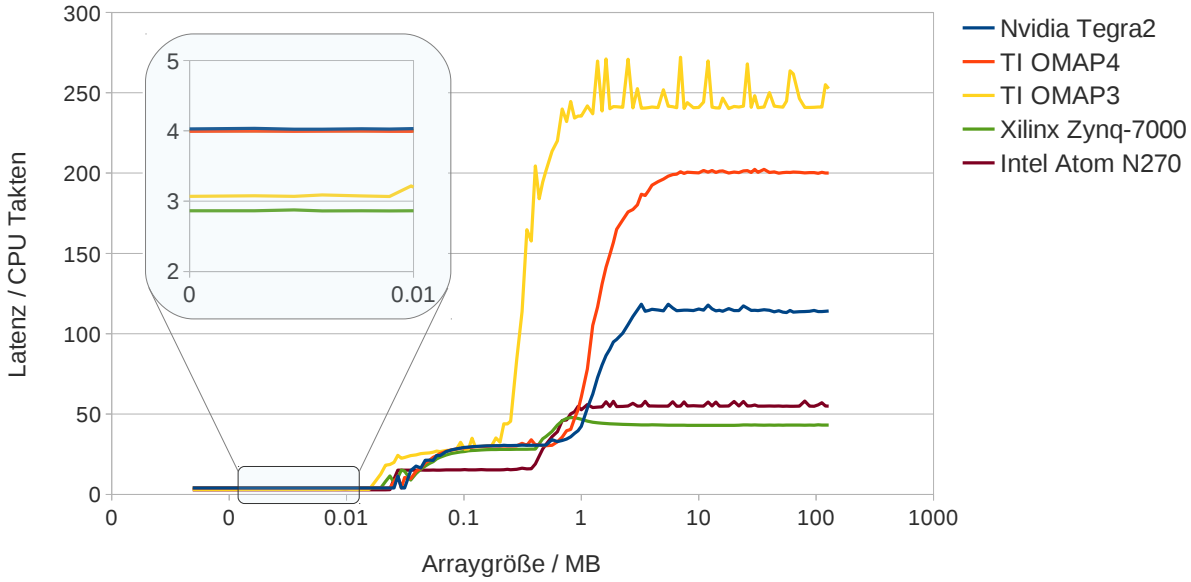

Abbildung 14: Vergleich Speicherlatenz in CPU Takten