Virtualisierungskosten

Sven Fiergolla

January 13, 2020

Benutzte Hardware:
- AMD Ryzen 5 2600X Six-Core Processor (12 Threads) 3.6 GHz Base Clock, 4.2 GHz Boost
- 16GB 3200MHz Ram
- Samsung evo ssd

1 Aufgabe 1: "Virtualisierungskosten (Nativ)"

Performance results (FLOPS and IOPs in G-FLOPS):

	time (real)	time(usr)	time (sys)	Number of FLOPs	Number of IOPs
native	0m4.101s	0m2.096s	0m0.000s	14.149101	14.437010
native	0m4.089s	0m2.084s	0m0.002s	14.036330	14.435551
native	0m4.096s	0m2.088s	0m0.002s	14.228415	14.425814
native	0m4.096s	0m2.090s	0m0.001s	14.195853	14.451477
native	0m4.103s	0m2.091s	0m0.000s	14.184424	14.555823
native (average)	0m4.94s	0m2.091s	0m0.001s	14.196281	14.452789

Table 1: Native Performance

	time (real)	time(usr)	time (sys)	Number of FLOPs	Number of IOPs
vm	0m11.225s	0m9.087s	0m0.041s	3.28	3.29
vm	0m11.089s	0m9.084s	0m0.042s	3.29	3.55
vm	0m10.996s	0m9.101s	0m0.039s	3.48	3.60
vm	0m10.527s	0m8.436s	0m0.027s	3.36	3.56
vm (average)	0m11.1s	0m8.9s	0m0.038s	3.35	3.56

Table 2: Virtual Performance

Zur Rirtualisierung wurde qemu benutzt, da es bei meinem Hostsystem Fedora bereits vorinstalliert ist. Um den Effekt besser sichtbar zu machen wurde auf eine Installation von KVM (Kernel based Virtual Machine) verzichtet und normale virtualisierung über qemu verwendet. Der Performanceunterschied ist

sehr deutlich erkennbar, die virtualisierte Lösung hat nur ein virtel der Hostsystemleistung aufzuweisen. Zu erwähnen ist jedoch, das nur 4 der 6 Kerne dem Gastssytem zugeteilt wurden um eine Benutzung des Hosts zu gewährleisten, was ebenfalls die Leistung vermindert. Ein Unterschied zwischen IOPs und FLOPs ist nicht deutlich zu erkennen, obwohl zu erwarten wäre, dass die IOPs nicht so stark von der Virtualisierung betroffen sind.

2 Aufgabe 2: "Virtualisierungskosten (JVM/CLR)"

Aufgabe 2 konnte nur bedingt erfüllt werden, es ist nicht gelungen die Ergebnisse oder die Programme sinnvoll in Relation zu setzten außer über eie relative Performance in FLOPs und IOPs, welche aber sehr stark abweichten. Dies hing von einer Menge Faktoren ab, wie zb die Anzahl durchgeführte Operationen, das JDK etc. Bei weniger Iterationen schien der Unterschied deutlicher zu sein als bei vielen Iterationen. Mit dem 1.8 OpenJdk wurden ca 7 GFLOPs erreicht, mit der GraalVM sogar fast 9. Allgemein kann man jedoch sagen, dass die Initialisierung der Laufzeitumgebung einiges an Ressourcen und Zeit in Anspruch nimmt, danach jedoch fast ähnliche Performance wie natives Ausführen erreicht.

3 Aufgabe 3: "Container (Docker)"

Genutztes C Programm aus Aufgabe 1, docker container ersellt von:
FROM ubuntu
COPY . .
CMD bash -c 'time ./CPU/CPUBenchmark 100000000 1'

docker build -t benchmark .

Ausführung als:
time docker run –it benchmark

sodass time ein mal vom Hostsymtem und ein mal von der Bash aus dem Ubuntu Container ausgeführt wird, das Hostsystem misst die Zeit inclusive des Starten des COntainers, das time innerhalb lediglich den Aufruf des benchmark Programms.

Schnell wird klar, dass die eigentliche Performance des programms nicht beeinflusst ist, da die Anzahl an Operation quasi idetisch zur nativen Auführung ist. Jedoch benotigt das starten des containers etwas Zeit in Anspruch, was in der time Auswertung des Hostsystems als time (sys) auffällt.

	time (real)	time(usr)	time (sys)	Number of FLOPs	Number of IOPs
in docker	0m4.120s	0m2.113s	0m0.002s	14.026831	14.29935
system time	0m4.421s	0m0.025s	0m0.020s		
in docker	0m4.103s	0m2.095s	0m0.001s	14.172581	14.373986
system time	0m4.711s	0m0.028s	0m0.013s		
in docker	0m4.104s	0m2.097s	0m0.002s	14.167995	14.364781
system time	0m4.704s	0m0.021s	0m0.021s		
in docker(average)	0m4.109s	0m2.099s	0m0.002s	14.196281	14.452789
system (average)	0m4.160s	0m0.025s	0m0.018s		

Table 3: Native in docker Performance