Und um geht es in diesem Kapitel? Wir möchten Ihnen vermitteln ein Verständnis des Designs von Rechnern und zwar geht es ein bisschen auch um Aufwand- und Ertragsanalyse. Also wie viel Aufwand erreicht, wie viel Ertrag im Rechnerdesign. Dazu müssen wir erstmal so ein paar kleine mathematische Betrachtungen machen. Das heißt, wir werden uns anschauen, wie ist Performance definiert, wie ist Speedup definiert, wie hat der Herr Amdahl sein Gesetz hergeleitet, der sehr viel aussagt über die Beschleunigung von Rechnerarchitekturen. Welche Probleme stellen sich mir beim Design, wie ermittle ich einen Instruktionsmix, wie ermittle ich die CPIs meiner Architektur oder meines Designs Metriken kurz gehen, Mips und Megaflops. Wir werden sehen, dass man mit der Durchschnittsbildung, wenn ich also die Geschwindigkeit eines Rechners in irgendeiner Metrik ausdrücken will, dass man da bei der Durchschnittsbildung sehr viel Schmuh treiben kann und wir werden uns damit kritisch auseinandersetzen, was denn eine vernünftige Durchschnittsbildung ist. Wir werden Benchmarks uns ansehen und zum Schluss nochmal ganz kurz auf Performance von eingebetteten Systemen schauen, also Embedded Processors und ganz am Ende das Verhältnis von Kosten und Preis und zwar von Chip und System unterhalten. Gut, worum geht es? Diese Definition wurde auch schon in Grundlagen der Rechneraktivität aufgelegt. Also erstmal ist Leistung ganz normal definiert als Arbeit durch Zeit. Leistung wird im Rechnerbereich, wie wir waren das auch, als Performance bezeichnet. Englisch ist klar, ist Work over Time. genauer ermitteln können. Also Zeit ist relativ einfach die Zeit zu ermitteln und zwar mache ich einfach eine Stoppuhrmessung, das heißt ich bearbeite eine vorgegebene Last, Arbeit oder Work, diese kürzen wir mit W ab und wenn ich beginne mit der Arbeit, dann starte ich die Stoppuhr, wenn die Arbeit beendet ist, dann stoppe ich die Stoppuhr, dann kann ich eine Ausführungszeit ermitteln, nämlich die Execution Time und die ist Endzeit minus wird auch mal als Response Time bezeichnet oder als Latency bezeichnet. Und wir werden sehen, dass es ganz so einfach natürlich nicht ist, weil wenn ich jetzt auf dem Unix-System zum Beispiel diese Ausführungszeit so bestimme, dann muss ich aufpassen, dass da nicht irgendwelche Systemzeiten oder irgendwelche anderen Jobs, die mir dazwischen gekommen sind, auch mitzählen. Also ich muss dann schauen, dass wirklich die Ausführungszeit sich auf die Erledigung meiner Arbeit, also nur auf meinen Prozess bezieht. die Arbeit sein? Im einfachsten Fall ist die Arbeit eine Instruktion, ein Programmbefehl, Assemblerbefehl, je nachdem wie auf welchem Level ich Instruktionen betrachte. Es kann ein ganzer Job sein, es können Transaktionen sein, wenn ich an Datenbanken denke. Es kann alles Mögliche sein, also irgendwelche Aktionen, die mir nützliche Ergebnisse liefern. Und dementsprechend gibt es auch sehr unterschiedliche Metriken, nämlich Mega Instructions per die abgearbeitet werden oder MFLOPs, also Mega Flops, Mega Floating Point Operations per Second oder eben beim Batch System Jobs pro Minute, beim interaktiven System Response Time, beim Datenbanksystem Transaktionen pro Sekunde, TPS und so weiter. Ja, ganz wichtig und das muss ich mir bewusst sein, ich habe es bei der Zeit kurz angesprochen, mit Wahrscheinlichkeiten und Durchschnittswerten gearbeitet. So, stellt sich erstmal die Frage, ja warum ist das eigentlich nicht deterministisches Zeitverhalten, wenn ich eine Stopp-Urmessung mache? Ja, ganz einfach, normalerweise bin ich

auf einem Multi-User, Multitasking-System unterwegs und dieses Zeitverhalten ist eben unter anderem lastabhängig, das heißt, wie viele andere User, wie viele andere Prozesse gerade am Start sind, deshalb kann es kürzer oder länger dauern, also ist ein bisschen von der Last abhängig und deshalb wäre es gut, Messungen alleine auf der Maschine bin und außerdem wird in der Regel so gearbeitet, dass ich nicht eine Messung mache, sondern dass ich viele Messungen mache und dann aus diesen vielen Messungen mir Durchschnittswerte heraus arbeite. Also eine Messung auf solchen Systemen ist in der Regel nicht besonders sinnvoll. Ich muss also öfter messen. Gut. Und wie wir dann so eine relative Performance angegeben, also ich sage einfach, mein Rechner also Rechner X ist N% schneller als Rechner Y ich vergleiche die Performances von Y und X also PY dividiert durch P von X und ich möchte jetzt statt der Performance da stehen haben eben die Formel für die Performance bilde einfach mal den Kehrwert und dann setze ich die Formel für die Performance ein dann komme ich dazu, dass eben das Verhältnis der Leistungen zueinander ist Rechner X dividiert durch die Execution Time auf den Rechner Y, durch die Last auf den Rechner Y. Und das Ganze ist dann in der Formel 1 plus N durch 100, wobei N eben die Prozent sind, die ein Rechner schneller ist, die der Rechner X schneller ist als der Rechner Y. So, jetzt macht das Ganze natürlich nur Sinn, wenn ich diese Rechner für gleiche Lasten vergleiche. Das heißt WX gleich WY, damit kürzt sich das Ganze raus und was übrig bleibt ist einfach das Verhältnis der Ausführungszeiten zueinander. Y im Vergleich zur Performance von X ist die Execution Time auf X durch die Execution Time auf Y. Und das Ganze ist 1 plus N durch 100. Und Sprech ist dann, die Performance von Y ist N% größer als die Performance von X. Ja, also umgekehrt als oben, aber spielt keine Rolle. Ist klar, was gemeint ist. So, vergleichen wir mal zwei Rechner miteinander. 10 Sekunden und für die gleiche Last hat der Rechner X die Execution Time von 15 Sekunden. Wenn ich jetzt ein Verkäufer bin vom Rechner Y, dann sage ich natürlich, ja mein Rechner Y, der ist 50% schneller als der Rechner X. Ganz einfach, indem ich die Formel verwende, Py zu Px ist Execution Time zueinander dividiert und zwar X durch Y ist 15 durch 10 und das Ganze Hier haben wir unser n, also unser n ist 50% in dem Fall. So, wenn ich jetzt der unterlegene Konkurrent bin, dann möchte ich das Ganze natürlich ein bisschen positiver sehen. Und dann sage ich einfach, ja gut, mein Rechner x, der ist nur 33% langsamer als der Rechner y. Ja, wie gibt es jetzt das? Ja klar, ich vergleiche jetzt x zur Leistung von y. Dementsprechend muss ich die Execution Type von y durch die Execution Type von x teilen. Da kommen dann 10 Fünftel raus, Klarkerwert. Und das Ganze ist 1 minus 1 Drittel. Also das ist eine Frage, die uns als Ingenieure leider auch beschäftigt. Die Wirtschaftswissenschaftler können davon ein Lied singen. Was nehme ich denn als 100%? Ist die Leistung von Y 100% oder ist die Leistung von X 100%? Und dementsprechend kommen da halt unterschiedliche Werte dabei raus. Und die Marketingleute, die machen nicht viel anderes als solche Spielereien. Prozent als Verkaufselement. Wir wollen das Ganze ein bisschen sachlicher betrachten und natürlich möglichst korrekt alles ermitteln. Definieren wir erstmal ein Speedup. Was ist der Speedup oder wie ist der Speedup definiert? Der Speedup S ist das Verhältnis der Performance nach

Verbesserungen der Architektur PY zur vorherigen Performance PX bei gleicher Last. durch px und das ist die formel wie wir sie gerade kennengelernt haben ist execution time von x durch execution time von y ist eben 1 plus n durch 100 dementsprechend sagt man auch maschine y hat einen speed up von n prozent im vergleich zu maschine x soweit nichts neues genau wie auf der folie vorher nicht mehr schneller weiterkomme mit meinen Ausführungszeiten, dann gehe ich einfach parallel. Und wenn ich viele Sachen gleichzeitig nebenläufig, also parallel löse, ja dann werde ich sehr, sehr schnell. Und der Herr Amdahl hat ziemlich schnell eine Beobachtung gemacht, nämlich, dass wir von komplexen Systemen sprechen mit sehr vielen unterschiedlichen Komponenten und dass diese Komponenten voneinander abhängig sind. ich muss natürlich alle Komponenten auf diesem kritischen Pfad so schnell wie möglich machen. Das heißt also, dass die Gesamtleistung, also die Total Performance von meinem System, ist auch von den schnellen und von den langsamen Komponenten abhängig. Und besonders langsame Komponenten, also beim von Neumann Rechner ist es zum Beispiel der Speicherzugriff, wenn Sie sich erinnern an die Grundlagen der Rechnerarchitektur, diese besonders langsamen Komponenten, die werden als Bottlenecks bezeichnet, also Flaschenhals. ob die sich im kritischen Pfad befinden und wenn sie sich im kritischen Pfad befinden, dann ist es natürlich sehr schlecht, was die Performance betrifft. Amdahl hat es aber nicht nur so allgemein formuliert, sondern er hat es auch qualitativ unterlegt, das heißt er hat eine Formel hergeleitet, die das Ganze mal ein bisschen deutlicher machen soll. Was hat er gemacht? Er hat erstmal betrachtet, die zu erledigende Arbeit wie, Es gibt sicherlich Bestandteile, die ich nicht beschleunigen kann. Diese Bestandteile, die ich nicht beschleunigen kann, die sind zum Beispiel durch Datenabhängigkeiten bedingt, durch irgendwelche Eigenschaften der Hardware bestimmt, das eben nicht weiter beschleunigt werden kann. Also ganz allgemein einfach eine Menge an Arbeit, die nicht beschleunigt wird. Und dann gibt es eine Menge an Arbeit, die ich sehr gut beschleunigen kann. W2. So, und er sagt einfach, okay, meine gesamt zu erledigende Arbeit ist die Summe aus den beiden. Also ein Teil der Arbeit kann beschleunigt werden, ein Teil der Arbeit kann nicht beschleunigt werden. Gesamtmenge W ist Summe W1 plus W2. So, jetzt will ich den relativen Anteil herstellen, das kann ich ganz einfach machen, indem ich sage, okay, der beschleunigte Anteil der Arbeit W, also relativer Anteil, Accelerated Fraction, daher kommt die Abkürzung FACC, Fraction für Anteil und ACC für Acceleration. Und dieser relative Anteil ist natürlich W2 im Verhältnis zu W, also W2 zu W. Und jetzt möchte er noch wissen, was ist jetzt der Speed-Up des beschleunigten Anteils meiner Arbeit. Und dafür definiert er zwei Performances. die Performance High, das ist das, wenn er im beschleunigten Modus läuft, also das ist die schnellere Performance meines Rechners, wenn er praktisch parallel läuft und die Performance