Ja meine Damen und Herren, ich begrüße Sie zur letzten Vorlesung vor Weihnachten aus der Vorlesung Rechnerstrukturen. wo wir uns mit beschäftigt haben, wie wir Einzelprozessoren, einzelne Cores möglichst beschleunigen können. Natürlich nutzen wir dafür auch Parallelität, indem wir eben Instruction Level Parallelism, also Parallelität auf Befehlsebene versuchen auszunutzen und zum Beispiel, indem wir mehrere Execution Units spendieren, diese methoden haben natürlich sehr lange dazu geführt dass die mikroprozessoren sehr viel schneller wurden und man kann aber feststellen dass so um die jahrtausendwende ein bisschen später diese methoden eigentlich ziemlich ausgereizt waren und gerade solche firmen wie intel die haben sich ja lange geweigert multiprozessoren zu bauen also nicht nur einen nehmen, sondern zwei, drei oder viele Prozessoren herzunehmen, um ein Problem zu lösen. Nachdem aus physikalischen Gründen mehr oder weniger, da gehen wir jetzt gleich darauf ein, dieser Weg nicht mehr weiter zu beschreiten war, zumindest was Aufwand und Ertrag betrifft, hat man sich dann auch bei den Mikroprozessorherstellern entschlossen, Multiprozessoren zu bauen bzw. Multicores zu bauen. Das Kapitel, was wir, oder die Kapitel, die wir im Folgenden behandeln, sehr wichtig, auch was zukünftige Entwicklungen darstellt. Okay, es geht also jetzt um Parallelrechner. Das Kapitel ist überschrieben mit Multimanikos und Multiprozessoren. Wir gehen natürlich noch darauf ein, was der Unterschied ist. Um dieses Kapitel zu verstehen, ist es gut, wenn Sie sich noch so grob an die Grundlagen der Rechnerarchitektur erinnern. Und natürlich auch die Kapitel, die wir bisher besprochen haben in der Vorlesung Rechnerstrukturen. Es geht also in diesem Abschnitt der Vorlesung, in den nächsten Kapiteln, geht es um Multicores, um Parallelverarbeitung, um massiv parallele Systeme und natürlich auch um Rechnerstrukturen, sprich Rechnerarchitektur. gelten atomare Größen. Trotzdem hat sich gezeigt, dass wir immer stärkere Prozessoren oder immer stärkere Rechner brauchen, um mit den Herausforderungen der Zukunft ganz gut fertig zu werden, die technischen Herausforderungen. Und wir wollen in diesem Abschnitt der Vorlesung eben die Möglichkeiten zeigen, die so eine Parallelverarbeitung bietet. Allerdings wollen wir natürlich nicht die Probleme verschweigen, die so eine Parallelverarbeitung mit sich bringt. geht erstmal darum parallelrechner zu motivieren warum es nicht so vielversprechend ist existierende monoprozessoren weiter aufzurüsten wir wollen zeigen welches potenzial in der leistungssteigerung durch parallelität liegt und in dem ersten kapitel in diesem abschnitt möchte ich dann auch noch so kurze vorschau geben was wir alles uns anschauen werden und natürlich paar literaturhinweise Also wir haben ja schon ein paar Mal über das Mursche-Gesetz gesprochen. Und das Erstaunlichste eigentlich am Mursche-Gesetz ist, dass es so lange gegolten hat. Es gilt jetzt seit fast 50 Jahren oder seit mehr als 50 Jahren inzwischen. Und wir sprechen von exponentiellen Wachstum. Und ein exponentielles Wachstum über so eine lange Zeit, das ist mehr als erstaunlich. Also was besagt das Mursche-Gesetz? Da gibt es unterschiedliche Interpretationen. verdopplung der transistordichte auf einem mikroprozessor chip alle 18 bis 24 monate gibt das ist das natürlich ein wort ja dass ich alle eineinhalb bis zwei jahre die anzahl meiner transistoren auf dem chip verdoppeln das muss man erst mal schaffen was sind denn so die direkten folgen aus dem motion

gesetz gut je kleiner die transistoren werden desto schneller kann ich die Schnelle Taktung hängt natürlich direkt mit Performance wieder zusammen, also desto schneller werden eigentlich meine Prozessoren. Auf der anderen Seite ist es natürlich so, je kleiner die Strukturen auf dem Chip werden, desto höher ist die Leistungsaufnahme, Leistungsdichte, wenn wir das ausdrücken in Watt pro Quadratzentimeter. Das heißt, wir werden irgendwann Probleme bekommen bzw. Probleme schon haben in der Wärmeableitung. an Energie in diese Chips reinpumpen, muss natürlich irgendwo in Form von Wärme wieder abgeleitet werden. Da ich immer mehr Strom reinschicken muss und immer schneller takte, ist auch die Leckströme dabei, die dabei entstehen, sind auch immer höher, stellt natürlich ein Problem dar. Also das ist reine Wärme, was ich da reinpumpe. Die macht produktiv nichts, außer den Rechner warm. Und insgesamt lässt sich nachdem dieses Murscher-Gesetz jetzt schon so lange gilt, dass wir so langsam tatsächlich an die physikalischen Grenzen kommen. Also ich höre das natürlich, wir nähern uns den physikalischen Grenzen an seit 30 Jahren, aber man muss sagen, inzwischen sind wir wirklich sehr nah an diesen physikalischen Grenzen. Das heißt, wenn ich auf die Strukturen schaue, die inzwischen auf so einem Chip sind, dann sind wir ein paar Atome breit und so eine Leitung, die muss natürlich mindestens ein paar Atome breit sein, dann, so ein Chip ist sehr klein, aber wenn ich den schnell genug takte, dann ist er immer noch eine Riesenentfernung für Elektronen, die in so einen ganz minimal kleinen Zeitraum von A nach B kommen müssen. Also wir haben Lichtgeschwindigkeit, mit der sich die Elektronen bewegen im Wesentlichen und die brauchen natürlich trotz Lichtgeschwindigkeit ein bisschen Zeit, um von A nach B zu kommen. Ja und dann schon angesprochen, die Leistungsdichte und damit verbundene Kühlproblematik ist auch so ein dickes Problem, was mit physikalischen Grenzen zu tun hat. So, jetzt können wir uns das Ganze historisch betrachten. Wir werden ja auch noch in dieser Vorlesung, entweder heute oder beim nächsten Mal, uns ein bisschen anschauen, die Geschichte der Parallelrechner. Weil ich denke, so historische Betrachtungen sind ganz wichtig, wenn wir in die Zukunft schauen, dass man sieht, wie hat sich denn das Ganze entwickelt und wohin könnte die Reise gehen. gilt das Moore'sche Gesetz und so die ersten Mikroprozessoren oder Vorgänger von Mikroprozessoren, wenn man so will, der 4004 oder 8008, 8086 ist vielleicht ein Begriff aus Ende der 70er Jahre, da reden wir von Tausenden bis Zehntausenden von Transistoren, die auf so einem Chip untergebracht waren. Wir haben hier auf der Y-Achse unser Wachstum in Transistoren ausgedrückt, und endet bei 10 Milliarden. Wir sehen ein schönes exponentielles Wachstum hier aufgezeichnet. Und wir haben hier unsere Jahresleiste, die beginnt 1970 und endet 2015. Und das ist eben der Beweis dessen, was ich gerade gesagt habe, nämlich dass das Moesche Law zumindest aus Intel-Sicht die letzten 50 Jahre gültig war. Wir haben also hier die ganzen Generationen von Mikroprozessoren, die Intel so hervorgebracht hat. Schlagzeilen gemacht hat, war praktisch der 8086er und dann haben wir hier die Pentium-Geschichte, die hier noch läuft und dann schließlich mündet in die i3, i5, i7 Architekturen mit unterschiedlichen Bezeichnungen für die Plattformen. Inzwischen sind wir also bei deutlich mehr wieder andere Metriken einführen und es ausdrücken, je nachdem was man

eben mit seiner Zeichnung ausdrücken möchte. Also die Folie vorher war eben das Wachstum in Transistoren. Diese Folie hier, die geht ein bisschen auf die Technologie ein, beginnt dementsprechend in der Zeitleiste ein bisschen früher. Also wir starten hier im Jahr 1900 und enden im Jahr 2025 als Vorhersage. Und man hat also hier, zu den mit 30er Jahren, da gab es, wenn man überhaupt von Rechnern reden will, also es waren diese mehr oder weniger mechanischen Rechenmaschinen, also elektromechanische Rechenmaschinen, die es da gegeben hat. Dann hat man einen kurzen Zeit lang mit Relays gearbeitet, so in den 40er Jahren, wo man so die ersten echten Rechner praktisch gebaut hat, oder zumindest Vorarbeiten zu den ersten Rechnern geleistet hat. mit Röhren gearbeitet. Das wurde aber relativ schnell dann von der Transistor-Technologie abgelöst, so 60er bis 70er Jahre und dann Anfang der 70er, wie wir gerade auf der Folie vorher gesehen haben, da begann dann das Zeitalter der integrierten Schaltkreise, also der ICs und da befinden wir uns natürlich immer noch. ist, das ist die Calculations per Second per 1000 Dollar, also wie viele Berechnungen pro Sekunde kann ich für 1000 Dollar durchführen und da sieht man natürlich auch ein ganz deutliches exponentielles Wachstum wieder, dass eben Berechnungen natürlich immer billiger werden. Das andere, was diese Kurve wohl aussagen will, ist, jetzt sind wir wieder ein bisschen wirklich den Menschen überlegen. Das heißt, die Angst, die man ja dann immer hat, das ist da unten in einem Satz ausgedrückt, the singularity is near when humans transcend biology. Also wenn wir eine Technologie entwickeln, die der Biologie überlegen ist, dann wird künstliche Intelligenz natürlich interessant. So, die sagen, dass wir im Jahr 2020, was die Technologie betrifft, eines Maushirns haben und dass wir, wenn wir die Leistungsstärke eines menschlichen Hirns übertreffen wollen, da einfach noch ein bisschen arbeiten müssen. Aber andererseits sagen die natürlich voraus, dass es in greifbare Nähe rückt. Also diese Vorhersage ist von 2013, aber es hat sich nicht allzu viel verändert seitdem. zumindest hardwaretechnisch langsam in die nähe kommen dass wir wenn wir die schlau programmieren ja die künstliche intelligenz zum tragen kommen das heißt dass wir irgendwann den menschlichen überlegen sein werden was die technologie betrifft und wird es dann natürlich auch software technisch umsetzen können das ist wieder eine ganz andere frage ja so ein paar meister uns gibt es immer wenn man sich in der rechnerentwicklung anschaut also der intel pentium war sicherlich ein das letzte Jahrtausend gebaut, also von 1993 bis 1999. Im Laufe der Entwicklung wurde der natürlich immer schneller getaktet, das heißt mit 60 MHz begonnen und am Ende dann 300 MHz getaktet. Frontside-Bus auch ein bisschen schneller geworden von 50 bis 66 MHz. Es gab schon einen Cache-on-Chip, 16 KB bis 32 KB. Die Fertigungstechnologie 800 Nanometer bis 250 Nanometer. 20, 25 Nanometer, das heißt nochmal 10 mal besser. Und der Befehlssatz, der darauf lief, Maschinebefehlssatz, ist eben diese x86 Maschinebefehlssatzarchitektur. Ja, wo sind wir heute? Heute sind wir doch eher bei der i7 Architektur, Intel i7 Ivy Bridge. Und was haben wir? Wir haben keinen Monoprozessor mehr, sondern wir haben einen Quad-Core Prozessor. die Arbeit leisten, die parallel zusammenarbeiten. Wir haben mehrere Caches, wir haben eine ganze Cache-Hierarchie, also ein L1-Cache, der ist immer noch im Kilobyte-Bereich, L2-Cache, da sind

wir schon fast beim Megabyte, und dann L3-Cache, da sind wir dann deutlich im Megabit-Bereich. Wir haben natürlich vier Kerne, die wir mit Befehlen und Daten versorgen müssen, dementsprechend aufwendig ist das Interface zum Speicher, Dual-Channel, also man versucht hier auch wieder Parallelität zu nutzen, dass man auf mehreren Kanälen gleichzeitig auf diesen Speicher zugreift. Wir haben ja in Grundlagen der Rechner-Architektur diese Speicherchips schon besprochen, DDR3 Speicherkontroller, entsprechenden schnellen PCI-Express-Controller, damit hier möglichst schnell auf diesen Speicher zugegriffen werden kann. Außerdem, ich habe ja jetzt wirklich sehr viele Transistoren, dann eine kleine GPU unterzubringen, also eine Graphical Processing Unit. Das heißt, ich kann normale Grafik locker vom Chip erledigen lassen, brauche also keine spezielle Grafikkarte. Die brauche ich dann erst, wenn ich Gamer bin und tolle Spiele machen will. Für normale Textverarbeitung, Büroverarbeitung, normales Processing reicht die GPU, die da integriert ist. Immer wichtiger wird natürlich auch die Verlustleistung, das heißt die Leistungsaufnahme von so einem Chip. viele unterschiedliche Varianten von dieser i7 Architektur, je nachdem was ich brauche, auf Leistung getrimmt oder auf Verbrauch getrimmt. Der erste i7 kam so im Jahr 2012 auf den Markt, Fertigungstechnik ist inzwischen 22 Nanometer und das ändert sich natürlich, aber irgendwann als er auf den Markt kam, war man bei einer Darauf waren 1,4 Milliarden Transistoren inklusive der GPU, die da drauf ist. Getaktet wurde das Ganze zwischen 2 und 3 GHz. Und da gibt es natürlich jede Menge unterschiedliche Modelle, Ausführungen dieses Chips. Je nachdem, wo Sie das einsetzen. Ob Sie es im Laptop einsetzen, im Desktop oder im Server vielleicht einsetzen. Mit speziellen Hardware-Eigenschaften für die Aufgaben. Wir brauchen immer mehr Leistung und das ist ja ein Grund, warum wir Parallelverarbeitung betreiben. Und wir sehen es aber auch, wenn wir uns unsere gute alte Performance-Formel anschauen. Die haben wir ja schon in der Gera-Vorlesung eingeführt. Und wir haben irgendwann mal gesagt, okay, die Rechner-Performance, die ich erreichen kann, ist im Wesentlichen der Takt dividiert durch Cycles per Instruction ideal. Cycles per Instruction oder auch Instructions per Cycle, wenn ich superskalar bin, plus einen bestimmten Delay, weil natürlich ist die Realität ja oft ein bisschen unterschiedlich von der Theorie und ich muss halt mal auf den Speicher warten, ich habe irgendwelche Datenkonflikte und dementsprechend treten da Stalls in der Pipeline auf und ähnliches und das muss ich natürlich zusammenfassen in einem Faktor von Delay. Dann habe ich noch die Mächtigkeit meines Befehlssatzes, es macht ja einen Unterschied, Risk-Architektur habe mit relativ kurzen, knackigen Befehlen, die natürlich auch nicht so viel tun und verglichen mit einer VLEW-Architektur, also Very Large Instruction Word, wo ich sehr lange Befehle habe, meistens mehrere Befehle in einem Befehlswort, die natürlich dann entsprechend mächtiger sind. Das heißt, ich muss das Ganze mit so einem Mächtigkeitsfaktor wichten. Und dann habe ich natürlich noch den Faktor PF, also effektive Parallelität. Und Wir haben uns bis jetzt in der Vorlesung genau auf diese Faktoren hier, also auf diesen Teil der Formel konzentriert. Das haben wir jetzt ziemlich ausgereizt mit Superskalarität, mit VLEW-Betrachtungen, was wir alles gemacht haben. bin. Dass ich das nicht sein werde, darauf werden wir eben eingehen. Gut, das

heißt, wenn wir den vorderen Teil der Formel hier ausgereizt haben, durch alle möglichen architektonischen Betrachtungen, es soll nicht sein, dass irgendwann nochmal ein Geistesblitz kommt, wo ich, keine Ahnung, hier unten nochmal beschleunige oder hier oben. Ja gut, hier oben werde ich eher weniger beschleunigen, weil das Ganze einfach zu warm wird. Also viel schneller takten ist nicht drin. Wir haben ja dieses Plateau mir der seit über zehn Jahren inzwischen, weil es einfach technologisch nichts bringt, noch höher zu takten. Das heißt, wenn sich Änderungen ergeben, dann in dem Faktor oder hier unten, aber was sich schon ein bisschen abzeichnet, wenn man sich die neuen Architekturen anschaut, dass eben dieser Faktor PF, also die effektive Parallelisierung, dabei eine sehr wichtige Rolle spielen wird. Ja, wie gesagt, also Parallelität gibt es ja auf Wir haben auch schon ganz gut gesehen, wo die überall genutzt wird. Also erstmal natürlich in der Wortparallelität, das heißt, gehe ich jetzt davon aus, dass ein Wort 16-Bit-breit ist, 32-Bit-breit ist oder 64-Bit-breit ist, also alle modernen Architekturen haben mittlerweile 64-Bit-breit. Wir sehen Pipelining überall, also arithmetisches Pipelining, Befehlspipelining ist in den Prozessoren eingebaut. Diese ersten drei Ebenen, die beziehen sich natürlich immer auf Instruction-Level-Parallelism, also wie viel Parallelität steckt in meinen Befehlen drin und wie kann ich die nutzen. Diese Parallelität ist aufgrund von Daten- und Kontrollflussabhängigkeiten natürlich gering. Wir erinnern uns an den Aufwand, den wir treiben mussten, um Sprungvorhersage zu treffen, sind. Und dementsprechend kann ich hier nicht beliebig parallelisieren, sondern ich habe eine bestimmte Parallelität, die ich nutzen kann, aber die ist auch deutlich begrenzt. Und dann hat man sich also entschlossen, irgendwann, bevor man jetzt versucht, irgendwelche unmöglichen Verringerungen zu machen, um so einen Monoprozessor noch schneller zu machen, warum dupliziere ich nicht einfach diese Monoprozessoren und baue mir eine wo ich leistungsstarke Einzelkerne zusammenarbeiten lasse. Das ist also die Grundidee. Ich schalte einfach viele Prozessoren zusammen. Warum hat man das nicht früher gemacht? Naja, wir werden es gleich sehen. Es gibt natürlich, wenn ich mit Multiprozessoren arbeite, so ein paar Überzeugungsprobleme. Also man sagt immer, die Programmierung ist deutlich komplizierter, über meine Hardware machen. Das brauche ich beim Monoprozessor nicht, weil der Compiler übernimmt mir das. Ich drücke meinen Algorithmus in einer höheren Programmiersprache aus, egal ob die jetzt Java heißt, C++ oder irgendwas anderes heißt und der Compiler hat die Aufgabe, das dann auf die Maschine zu bringen und dann auch die Maschine möglichst gut zu nutzen. Das heißt, als Programmierer mache ich mir in der Regel keine Gedanken über die Hardware. bei einem Multiprozessor auch so funktioniert, das funktioniert aber nur bedingt so, also automatische Parallelisierung ist ein bisschen schwierig, wie wir noch sehen werden. Ja, wo sind dann eben die Grenzen entstanden bei den Monoprozessoren? Wir haben schon darüber gesprochen, die Lichtgeschwindigkeit, warum? Ja, wenn ich von einem Takt von 10 GHz ausgehen würde, den wir ja nicht erreicht haben, sich mein Elektron in einem Taktzyklus um 3 cm maximal. Selbst wenn ich mir so einen Chip anschaue, da bin ich schon im Zentimeterbereich, kann das dann zu Problemen führen, dass das Elektron nicht mehr von A nach B rechtzeitig

kommen kann. Das heißt, außer dass das Ding unendlich heiß wird, ist das nächste Problem, dass die Elektronen vielleicht nicht schnell genug von A nach B kommen. ausgehe, da ist es kein Problem. Da sprechen wir von 3 Metern, die so ein Elektron zurücklegen kann. Und wenn ich da jetzt auf den Chip schaue, dann ist natürlich 3 Meter eine Entfernung, die nicht auftritt. Wenn ich allerdings im Zentimeterbereich bin, dann kann sowas passieren. Und das nächste ist, ich muss ja immer Aufwand und Ertrag betrachten. Und wenn die Technologie Verbesserungen einfach nur mit unendlich großem Aufwand erreicht werden können, dann bringt mir das nichts mehr. Das heißt, da muss ich einfach mal sagen, okay, die Technologie neuen Ansatz suchen. Und der nächste Punkt haben wir gerade besprochen, die Parallelität auf Instruktions-Ebene ist begrenzt. Das sehe ich einfach. Wir haben Datenabhängigkeiten, wir haben Kontrollflussabhängigkeiten, also komme ich da auch nicht weiter auf der Stelle. Das heißt, wenn ich überhaupt noch Leistungssteigerung will, dann muss ich versuchen, eben alle Möglichkeiten Dieses Beispiel haben wir schon mal, ich glaube in den Grundlagen der Rechnerarchitektur nochmal besprochen, warum es vorteilhaft sein kann, auch bei schlechter Auslastung mehr Transistoren für den Multiprozessor zu spendieren. Beispiel und wenn ich dann mit diesen 20 Millionen Transistoren so ein Multiprozessor aufbaue oder ein Multicore aufbaue, dann habe ich natürlich nur 1,2 Millionen Transistoren pro Core, habe aber 16 Cores und wir bleiben bei 1 GHz Taktung. Das CPI ist natürlich deutlich höher, weil die Cores, die sind nicht so ausgereizt in der Architektur wie der Monoprozessor, sind also ein bisschen einfacher, aber die sind nicht zehnmal langsamer, sondern die sind halt dreimal gut 3 mal langsam und selbst wenn ich das schaffe jetzt diesen Rechner nur zu 50% auszulasten, bin ich immer noch 5 mal so schnell als der Monoprozessor also das ist natürlich ein konstruiertes Beispiel, aber das soll einfach verdeutlichen dass da diese Technologie links der Monoprozessor komplett ausgereizt ist wenn ich da einfach ein paar wenige Transistoren verwende ich nicht so viel weniger schnell bin und deshalb lohnt es sich da eben so ein technologisch Weg zu machen. Gut, das sind die Good News. Natürlich gibt es auch Bad News. Ja, also was, warum hat man nicht schon viel früher Multiprozessoren gebaut? Um ehrlich zu sein, hat es in den, ja ich habe von 86 bis 91 promoviert in der Zeit, da war Parallelrechner mal kurz en vogue. mal gegangen, so Anfang der 90er Jahre war so ein richtiger Hype mit Parallelrechnern. Diese Projekte sind, das muss man im Rückblick einfach so zugeben, alle gnadenlos gescheitert. Warum? Weil in dieser Zeit die Mikroprozessoren einfach so viel schneller geworden sind, jedes Jahr so viel schneller geworden sind, bis man so einen Parallelrechner fertig entwickelt Mikroprozessoren eingesetzt. Bis der Parallelrechner mit seinem Verbindungsnetzwerk, mit seiner Anbindung fertig war, hat es einen Mikroprozessor gegeben, der genauso schnell war wie der Multiprozessor. Und dann hat der Programmierer natürlich gesagt, warum soll ich mich denn mit einem Multiprozessor herumärgern, wenn ich in der gleichen Leistungsstufe einen Monoprozessor bekomme. Debugging ist schwieriger auf Multiprozessoren. Alles ist schwieriger, warum soll ich das machen? Dann hat es dann gegeben, da hat man ein paar hundert Millionen versenkt und hat dann einfach das Thema Parallelrechner einschlafen

lassen, erfolglos, weil eben die Monoprozessoren immer noch sehr schnell, viel schneller worden sind und erst in den 2000er Jahren, als dann Intel eingesehen hat, dass man jetzt Multicore bauen muss, dass man mit dem Single-Core nicht mehr weiterkommt, dann hat das Thema Parallelrechner wieder so richtig an Bedeutung gewonnen und ist jetzt momentan wieder top aktuell. Für mich ist es erstaunlich, weil ich habe die erste Phase schon mitgemacht und manchmal denkt man ja, die neuen Wissenschaftler finden das Rad jetzt komplett neu, weil sie nicht in alle Veröffentlichungen schauen. Also viele der Probleme, die jetzt diskutiert werden, kommen mir alle sehr bekannt vor. Aber es wird einfach nochmal diskutiert. Aber die Probleme haben sich nicht gelöst. Das ist ja das, was mich beruhigt. Also die gibt es immer noch und deshalb gibt es viel Okay, was sind denn die Probleme bei den Multiprozessoren? Ich habe erstmal jetzt auf einmal einen neuen Endos-Parameter, die Topologie. Das heißt, wie verbinde ich denn meine Prozessoren? Baue ich ein speichergekoppeltes System? Baue ich ein Message-Parsing-System? Baue ich ein hybrides System? Und diese Topologie spiegelt sich natürlich auch irgendwo in der Programmierung wider. Das heißt, es wird, wenn ich automatisch parallelisieren will, relativ kompliziert, Compiler zu schreiben. und selbst die können natürlich nur statisch parallelisieren. Das heißt, nicht jede Anwendung ist auch geeignet, auf so einem System zu laufen. Es gibt also Anwendungen, die haben sehr wenig Parallelität, die sind streng sequenziell. Die werde ich nicht erfolgreich auf so einem Parallelrechner implementieren. Das heißt, da muss ich den dahinterliegenden Algorithmus genauer betrachten. Ich muss schauen, ob ich den Algorithmus ändern kann, dass ich da ein bisschen mehr Parallelität reinbringe. bevor ich den implementiere. Wenn ich von vielen Prozessoren spreche, dann ist es tatsächlich so, dass die Auslastung aufgrund der Datenkonflikte, Kontrollkonflikte normalerweise schlecht ist, dass ich also froh sein kann, wenn ich 20 bis 50% Auslastung erreiche. Also Produktivarbeit zwischen 20 und 50%. Die Programmierung, wenn ich wirklich Leistung rausholen will, aus so einer Maschine, die ist ein bisschen schwierig. weil ich als Programmierer mir jetzt Gedanken über die Hardware machen muss. Das ist für mich als Programmierer erstmal ungewohnt. Und wer schon mal parallele Programme geschrieben hat, der weiß, dass das menschliche Hirn nicht besonders gut ist, parallel zu denken. Und deshalb ist es halt ein Programmiermodell, was man sich erstmal dran gewöhnen muss. Dann ist es ja so, dass so eine Verhaltensanalyse, das heißt, was auf dem Parallelrechner sehr, sehr schwierig ist, weil es ja oft nicht deterministisch, asynchron passiert und natürlich ist auch das Debugging schwierig. Wenn ich einen Monoprozessor habe, dann ist es total easy. Da habe ich einen Debugger, ich kann sequenziell jeden Programmstück durchgehen. Beim Parallelrechner passiert auf 1000 Prozessoren irgendwas anderes gleichzeitig und auf welchen dieser 1000 Prozessoren schaue ich jetzt gerade drauf. Kann ich das überhaupt? In dem Moment, wo ich drauf schaue, beeinflusse ich mein Laufzeitverhalten. eben oft so beim Parallelenprogrammieren, dass ich einen Fehler drin hatte in meinem Programm. In dem Moment, wo ich den Debugger angeworfen habe, das heißt zwischendrin das Programm mal unterbrochen habe, war der Fehler weg, weil eben dieser Fehler zum Beispiel ein Synchronisationsfehler war. Also das Debugging auf einer Parallelenmaschine,

das ist absolut schwierig. Und dann zumindest war es früher so, bis man dann so einen Rechner zusammengebaut hat Betriebssystem dazu gebaut hat, einen Compiler dazu gebaut hat, sind ein paar Jahre ins Land gegangen und dann gab es eben den nächsten Rechner, der schneller war. Das heißt, ich muss ja so ein System eben ja alles dafür neu machen, also neue Compiler entwickeln, dann muss ich unter Umständen eine dynamische Lastverteilung implementieren. Ich muss meine kurz optimieren und das dauert natürlich bis sowas fertig ist. Argument, was natürlich immer kommt ist, es gibt Millionen bis Milliarden Lines of Code und die funktionieren nicht im Parallelmodus. Natürlich kann ich so gehen, dass die DECK Software auf dem Monoprozessor weiterlaufen lassen ohne Probleme, aber eben nicht, wenn ich da jetzt 1000 Prozessoren verwende, da kann ich nicht so ein altes C-Programm einfach draufschmeißen und hoffen, dass 1000 Prozessoren das unter Umständen langsamer sein. Also das ist natürlich alles ein Problem. Also wenn man sich diese Seite anschaut, dann kriegt man vielleicht ein Gefühl dafür, warum Intel sich so lange geweigert hat, mal die Kosten zu bauen. Das ist natürlich seit langem bekannt, wo die Probleme sind. Und man hat sich also sehr, sehr davor gescheut, parallel zu gehen, bis es halt gar nicht mehr anders gegangen ist. Ja, warum brauche ich denn immer noch leistungsfähigere ganz gut auf, aus auf meinem System. Mein Laptop läuft inzwischen 8 Stunden und macht ja eigentlich auch alles, was es soll. Aber es hat die Vergangenheit einfach gezeigt, dass diese leistungsfähigen Systeme letztendlich auch eine wesentliche Triebkraft der Weltwirtschaft darstellen. Ich kann einfach mehr in kürzerer Zeit erledigen und deshalb brauche ich sowohl auf dem Desktop als auch natürlich im Hintergrund irgendwelche dicken Server und eingebettete Systeme mit der entsprechenden Leistung. Was natürlich Grenzen des Wachstums darstellt, das sind zwei Walls, die wir inzwischen sehen, ganz klar, deutlich vor uns sehen. Das ist die sogenannte Memory Wall und das ist die Power Wall. Was sind diese beiden Walls? im Wesentlichen hervorgerufen durch die Verzögerung beim Speicherzugriff. Das heißt, es gibt die sogenannte CL, also Cus Latency. Das ist die Zeit, die es dauert vom Absenden des Lesebefehls, also der Prozessor sagt, okay, Load, bis zum Erhalt der Daten. Das ist also diese Verzögerung, die da auftritt. Und dann gibt es noch die RCD Time, also Rust to Cus Delay. wie lange dauert es vom ersten Datum, das ich bekommen habe, bis zum nächsten Datum, das ich bekommen habe. Und die Gesamtzugriffszeit ist natürlich CL plus RCD, also Cast Latency plus Rastocast Delay. Und wenn wir uns jetzt die Speichertypen anschauen, dann sehen wir, dass diese Zeiten sich eben nicht so wahnsinnig verändert haben wie Geschwindigkeiten beim Prozessor. anschauen, da haben wir eine Zählzeit von 18,7 Nanosekunden und bei neueren Systemen ist sie sogar wieder ein bisschen leicht gewachsen. Also wir sind mal, die DDR3-Speicher waren mal bei 8,2 Nanosekunden, inzwischen sind wir bei 9,5 Nanosekunden. Warum sind die hier trotzdem schneller? Naja, weil ich mehr Interleaving, mehr Parallelität nutze und diese Technologieveränderung hat dazu geführt, dass diese Zeit wieder ein bisschen weil ich eben sehr viel parallel mache. Und die RCD-Zeit, da sieht man ein ähnliches Verhalten, auch die ist eben nicht mehr deutlich schneller geworden. Das heißt, das Bild verdeutlicht ganz gut unser großes Dilemma, dass also die Prozessoren immer noch sehr viel schneller

werden, die Speicher allerdings eben nicht sehr viel schneller werden und das Einzige, was uns derzeit noch rettet, 2, 3 und ein Hauptspeicher dazwischen. Und dann natürlich Zugriff auf den Hauptspeicher möglichst parallelisiert, das heißt 4 Channels gleichzeitig, die da bedient werden, um überhaupt einigermaßen rechtzeitig die Caches vorzuladen, damit der Prozessor arbeiten kann. Also das Grundproblem sind die unterschiedlichen Taktraten der Prozessorchips und D-RAM-Chips. dann dementsprechend, wenn ich eine Metrik ansetze, zum Beispiel wie viele Befehle kann ich pro Speicherzugriff ausführen, dann steigt eben in dieser Metrik die Zugriffslatenz. Und das nächste sind Architekturänderungen wie Superskalartechnik, dass ich also ganze Blöcke von Befehlen hole. Dadurch brauche ich natürlich wieder eine größere Bandbreite im Instruction Cache. Ich habe also einen erhöhten Bedarf an Speicherzugriffen. gemacht dass ich habe da wie architekturen fahre das heißt dass ich extra cash für befehle und daten aber trotzdem habe ich da eben noch probleme und dann jetzt mache ich mal die course und das macht die probleme natürlich noch schlimmer ich habe ja schon probleme monochrome mit daten zu versorgen jetzt muss ich auf einmal viele kurs mit daten versorgen und ja das macht meine situation was den speicher-begriff natürlich noch mal deutlich schlimmer Gut, letztendlich haben wir das Grundproblem dann, dass ich nicht schnell genug Daten und Befehle vom Chip extern in diesem Speicher nachladen kann, um den Prozessor auszulasten und als direkte Konsequenz habe ich halt meine sehr komplexe Speicher-Hierarchie. war Platz auf dem Chip, um Registersätze größer zu machen. Andererseits muss ich diese Registersätze retten beim Prozesswechsel, also kann die nicht unendlich groß machen. Es wird einfach beliebig kompliziert an dieser Stelle. Wir haben die Harvard-Architektur schon eingeführt, getrennte Code- und Datencaches. Und wir haben natürlich auch das auf allen Ebenen, das heißt sowohl auf den großen Desktop-Prozessorebenen als auch schon heutzutage getrennte Code und Datenspeicher eingeführt werden, weil selbst diese einfachen Prozessoren sonst nicht in der Lage werden, sich ausreichend schnell mit Daten und Befehlen zu versorgen. Ja, so eine Speicherhierarchie bringt immer Kohärenzprobleme, die ich natürlich am besten hardwaretechnisch irgendwie lösen muss. Ja, dieses Bild zeigt das Ganze nochmal, ausgehend startend genormt auf 1980 und dann schauen Da gilt Moore's Law, das ist also die Geschwindigkeit oder die Performance der Prozessoren, wie sich die entwickelt hat von 1980 bis 2010. Und hier unten ist praktisch die Entwicklung der Speicherchips und wir sehen da so ein Gap, das natürlich wächst und zwar 50% pro Jahr. Das macht die Sache eben entsprechend kompliziert. Und entsprechend komplexer werden die Speicher-hierarchien. Innerhalb des Mikroprozessors, also On-Chip, haben wir keinen Cache. Und im Jahr 2010 haben wir einen 3-Level-Cache On-Chip und einen 4-Level-Cache Off-Chip. Und der erste On-Chip-Cache war im Jahr 1989 mit 8 KB Größe. Und der erste L2-Cache 1995 und 2003 konsequenterweise der erste L3-Cache im Ethanium-2 mit 6 MB Größe. Das sind alles direkte Konsequenzen aus diesem Prozessor Memory Gap. Und da gibt es auch momentan noch keine echte Lösung. Dieses Gap wächst weiterhin. Ja, das war also das Problem der Memory, wo das ungelöst ist, das wir vor uns sehen. Also da muss man mal schauen, was die Zukunft so bringt. Ob es dann irgendwann Level 5 und Level 6

Cache gibt. Powerwall. Firmen wie Intel, die versuchen natürlich langfristige Prognosen zu machen, wie ist ihre Roadmap und die SIA, also Semiconductor Industry Association, die hat in 1998 eine Prognose gewagt, dass sie sagt, okay, im Jahr 2008 werden wir unsere Chips so mit 6 GHz takten und im Jahr 2011 werden wir unsere Chips mit 10 GHz takten. Man sieht an dieser natürlich mal wieder, dass es gefährlich ist Prognosen zu machen, man kann auch ganz schön daneben liegen und wir wissen ja jetzt, dass Anfang der 2000er Jahre dann diese Stagnation eingetreten ist, das heißt wir sind so richtig über diese 3 bis 4 GHz Takt eigentlich nicht hinausgekommen. Gut, Intel war ein bisschen pessimistischer, hat 2004 noch davon gesprochen, Natürlich, ich will ja innerhalb eines Taktes Befehle vervollständigen. Und wenn ich dann an meinen Befehlspipelining schaue, dann muss ich die einzelnen Pipeline-Stofen natürlich schnell genug machen, um so einen 5 GHz Takt zu fahren. Und dementsprechend hat man natürlich in Pentium 4 die Pipe in dieser Prescott-Mikroarchitektur relativ lang gemacht, um dann tatsächlich später mal so hoch takten zu können. Aber das ist natürlich nicht eingetreten, wie wir jetzt wissen. Es ist relativ zügig gegangen bis ungefähr 3,4 bis 4 GHz und dann war Ende im Gelände. Das heißt, dann hat der Powerwall zugeschlagen, weil einfach die Hitzeentwicklung zu groß war und die Hitze nicht mehr abgeleitet werden konnte. Das heißt, im Jahr 2008, da haben wir eben keinen Prozessor, der mit 5 GHz getaktet ist, sondern der schnellste Prozessor im Jahr 2008, das war der Core 2 Extreme. 45nm Technologie und getaktet mit 3,2 GHz, das heißt also weit entfernt 5 GHz ist ja fast doppelt so viel, was eigentlich prognostiziert war. Gut, trotzdem natürlich ein schöner, schneller Rechner mit allem was schick ist zu der Zeit im Jahr 2008, aber eben ein bisschen weg von der Prognose. die höheren taktraten klar wenn ich schneller takte muss ich ein bisschen mehr energie da rein führen als wenn ich langsamer takte das heißt der takt ist in etwa proportional zur leistungsaufnahme hoch 3 also das ist immer schlecht so ein kubisches wachstum und nachdem dann der chip so heiß wird habe ich natürlich erhöhte thermische Verluste. Und das macht es noch heißer und dann sind meine Verluste noch höher. Das heißt, die Grenzen der Luftkühlung sind deutlich erreicht und ich kann nicht mehr schneller takten, weil sonst schmilzt einfach mein Chip. Und dann kann ich nicht mehr weiterarbeiten. Gut, also Taktung, Ende im Gelände. Aber was ich immer noch schaffe, aufgrund der Verkleinerung meiner Strukturen, Transistorzahl auf dem Chip immer noch verdoppeln kann. Alle 18 bis 24 Monate. Also das gilt derzeit immer noch. Wie lange werden wir sehen. Ich kann nicht mehr Parallelität rausholen als in so einem Programm drinsteckt auf Befehlsebene. Das heißt architektonisch kann ich im Monochor nichts mehr machen dann kann man sagen, dass bei normalen Programmen ich ungefähr zwei bis drei Befehle mit Tricks und Spekulationen, vielleicht vier bis fünf Befehle Parallelität habe, also Sachen, die ich gleichzeitig ausführen kann. Wenn ich mehr will, dann muss ich noch mehr spekulieren. Und ich kann natürlich immer fehlspekulieren, was mich Strafe kostet. Darüber haben wir ja gesprochen in den letzten Vorlesungen. Also Aufwand und Ertrag bringt einfach nichts. ist noch weiter vergrößert, dass der Effekt auch nur marginal ist, also Riesenaufwand mit wenig Ertrag. Diese Kurven zeigen das natürlich ganz gut, wir sind bei, ich glaube, ich bin mir nicht so ganz sicher, ob wir die 5

Nanometer Technologie erreicht haben, diese Kurven hier, die sind natürlich ein bisschen unschön, das heißt, wir haben auf der X-Achse Achse unsere Leckströme angezeigt und je nach Technologie 65 Nanometer bis 14 Nanometer Technologie sieht man natürlich ganz deutlich je schneller wir takten, desto höher sind die Leckströme und die Leckströme sind letztendlich nichts anderes als Wärme, die ich abführen muss und das ist natürlich nur begrenzt möglich auf dieser kleinen Chipfläche, die ich zur Verfügung habe Gut, also Memory Wall, Power Wall ist ein möglicher Ausweg Multi-Cores und Many-Cores. Das heißt, die Grundidee von Intel war dann, dass ich eben mehrere vollständige Prozessoren, die jetzt als Cores bezeichnet werden, auf einem Prozessorchip packe. Da kann ich natürlich den Vorteil der Duplikation zum Beispiel verwenden, dass ich zwei oder vier identische Cores da baue, dann brauche ich den bloß einmal entwickeln und dann Copy-Paste in meinem Layout. Natürlich muss ich die noch irgendwie vernünftig miteinander verbinden. so die letzten zehn jahre was hat man so auf dem markt gesehen typischerweise so 2 bis 16 cores für general purpose mikroprozessoren und bei eingebetteten prozessoren 8 bis 9 cores also eher eine kleine anzahl von cores was natürlich sich ein bisschen verändert hat sind die grafikkarten Purpose Core, der möglichst leistungsstark ist zu bauen, sondern sehr viele Spezialkores, die eben spezielle Aufgaben besonders gut lösen können, aber nicht General Purpose sind. Gut, wenn man jetzt das Moose-Gesetz so noch ein paar Jahre weiterführen möchte, dann kann man sagen, okay, wir verdoppeln jetzt nicht mehr die Chips, die Transistoren pro Chip, sondern wir verdoppeln die Cores pro Chip. Und wenn man von gehen, also eine Verdopplung alle 18 Monate, eine Vervierfachung alle 3 Jahre, dann wäre man im Jahr 2020 bei einem K-Cores, also bei 1000 Cores und wir sind im Jahr 2020 und wir sehen die 1000 Cores allerdings nur bei Grafikbeschleunigern und nicht bei Intel-Prozessoren. Und das ist natürlich auch eine der großen Fragen in der Zukunft ist, was ist denn jetzt so die die intel architektur wenige leistungsstarke course oder die entweder architektur sehr viele course die spezialisiert sind für aufgaben gibt es da irgendwo ein optimum natürlich ist es vom anwendungszweck abhängig general purpose sagt man heute sind ist der intel ansatz besser grafik ist der entweder ansatz besser aber wie wird es in zukunft ausschauen kann ich ich habe so viel einfach Spezialkores bauen. Das heißt, dass ich einen Core habe, der kann besonders gut General Purpose, einen Core, der kann besonders gut Multimedia, einen Core, der kann besonders gut Datenbanken oder sowas. Ich habe ja wieder einen sehr, sehr großen Entwicklungsraum und wo liegen jetzt meine Optima in diesem Entwicklungsraum? Nur zum Sprachgebrauch, also sobald man 32 oder 64 Prozessoren hat, da spricht man von Many Cores. Andere sind Multicores und Manycores, eben viele Cores. Auch diese Zahl kann sich im Laufe der Zeit ändern. Also es hat mal Zeiten gegeben, da haben wir schon von 16 Prozessoren von Manycores gesprochen. Okay, so, jetzt haben wir die zwei Walls, also Memory Wall, Power Wall im Blick. Jetzt macht sich, wenn wir uns entscheiden, Multi- und Manycores zu bauen, natürlich eine neue Wall auf. ist jetzt ein Parallelism-Wall. Also, was steckt dahinter? Dahinter steckt, dass natürlich alle heutigen Programme, dass die DEC-Software, dass die erstmal rein sequenziell programmiert wurde und da eben kein Augenmerk auf Parallelver-

arbeitung gelegt wurde. Dementsprechend gibt es wenig Programmierer, die parallele Programmiererfahrung haben. Dann das nächste Problem ist, wenn ich auf die Anwendungen schaue, genügend Parallelität? Auf Anni fällt mir Grafik ein, da finde ich immer Parallelität. Dann fallen mir die theoretischen Physiker ein und irgendwelche Biologen mit Molekularmodellen. Da steckt in der Regel sehr viel Parallelität drin. Aber das sind Nischenanwendungen. Ist denn im Allgemeinen genügend Parallelität vorhanden, um solche Rechner auszulasten? Und wie sieht so ein Ausgabesystem zum Manico aus? Das sind alles noch ungelöste Fragen. Die nächste Frage ist natürlich jetzt wieder nach der Anzahl der Cores. Was ist die optimale Anzahl von Cores, die ich da bauen kann? Das kann man auch allgemein nicht beantworten. Das hängt von der Anwendung ab. Letztendlich ist es eine Kosten-Nutzen-Frage, die ich entscheiden muss. Und ich muss natürlich auch für Stückzahlen sorgen. Das heißt, so eine Entwicklung ist wahnsinnig teuer und die lohnt sich nur, wenn ich eine entsprechende Anzahl von Prozessoren auch verkaufe. kann passieren durch die Parallelism Wall? Naja, es kann passieren, dass Intel die Lust daran verliert, noch mehr Cost auf einen Chip zu bringen, das heißt die Erhöhung der Kurzweil bricht ab, da eben ich nicht genügend parallelisierbare Anwendungen zum Beispiel finde. Ja, das ist so ein bisschen die Roadmap von Intel, dass man sagt, ok, wir fangen da für skalare und parallele Anwendungen und die werde ich auch weiterhin bauen. Und dass es natürlich noch eine andere Richtung gibt, eben diese Many-Core-Error, so im Jahr 2010 beginnend. Dafür brauche ich aber natürlich massiv parallele Anwendungen, um solche Rechner auch nutzen zu können. parallelen Programmierung, wenn ich jetzt so ein Multi- oder Minicore habe, erstmal natürlich mein Programmiermodell, also wie entwerfe ich überhaupt ein paralleles Programm, also paralleles Software Engineering, das ist ein neues Themengebiet, ist ein bisschen anders als herkömmliche Software Engineering. Dann muss ich mir natürlich eine schöne Programmierumgebung schaffen, also mit Debugger und alles, was man so braucht, die müssen natürlich erst entwickelt werden, Es gibt schöne Programmierumgebungen für die sequentielle Programmierung, aber das Gleiche, den gleichen Komfort brauche ich natürlich auch für parallele Programmierung. Ich muss möglicherweise neue Programmiersprachen entwerfen, erfinden, parallele Sprachen. In der Regel passiert es heutzutage so, dass ich meine sequentielle Sprache erweitere und ein paar Konstrukte für die Kommunikation, also wenn ich Message Passing mache, gekoppelte Systeme, parallele Systeme habe, dann brauche ich halt Konstrukte, damit ich Private und Shared Daten anlegen kann, dann brauche ich irgendwelche Synchronisationen auf den Shared Daten, die ich da machen kann. Das sind also in der Regel Erweiterungen von sequentiellen Sprachen. Aber ist denn das wirklich das, was ich möchte? Möchte ich denn nicht viel lieber eine Sprache entwickeln, die von vornherein diese Parallelität mit integriert hat? Das eröffnet natürlich ganz neue Möglichkeiten. natürlich so ganz neue Sprachen, ja, es ist immer gefährlich. Auf der anderen Seite muss ich sagen, habe ich im Laufe meiner Karriere so viele Programmiersprachen kommen und gehen sehen, warum soll da nicht auch mal eine parallele Sprache kommen? Also mein erstes Programm habe ich in Algoil 60 geschrieben, dann habe ich Modula zweimal

programmiert, weil es heißt, das ist eine tolle Sprache, dann zwischendrin mal Pascal, dann C programmiert, das heißt, es ist ja ja wo steht geschrieben, dass alle Programme keine Ahnung Scheme oder Java sein müssen, in 5 Jahren kann das wieder ganz anders ausschauen ja dann brauche ich Programmiermodelle für Multicores also für Speicherkopplung zum Beispiel, da gibt es schon Ansätze von OpenMP oder POSIX Threads, die man da verwenden kann und natürlich auch, wenn ich jetzt ManyCores habe, die dann wie wir gleich sehen werden in der Architektur durchaus auch Message Passing gekoppelt sind, also Nachrichten gekoppelt, weil ich ganz einfach über Speicherkopplung nur eine sehr begrenzte Anzahl von Prozessoren koppeln kann. Und wenn ich eben viele Prozessoren habe, dann bleibt mir nichts anderes übrig, als die Nachrichten zu koppeln. Und auch da brauche ich natürlich irgendwelche Programmiermodelle. Und dann gibt es zum Beispiel MPI, also Message Passing Interface. Das ist so eine mehr oder weniger genormte Programmiererweiterung, die es mir erlaubt, Nachrichten Ja, also, wenn wir uns jetzt das Thema Multi- und Many-Cos anschauen, dann muss man einfach sagen, was die Rechnerarchitektur betrifft, ist das natürlich die wesentliche Herausforderung der Informatik für die nächsten Jahre. Das heißt, ich muss die Dinge erstmal bauen und dann muss ich die Dinge programmieren. Das sind also zwei Richtungen. Einmal eher so die technische Informatik, weil die Dinge bauen, die Dinge programmieren. Das ist dann mehr die praktische Informatik. in allen Feldern zu tun aufgrund der Technologie Malti und Manicor. Das macht vielleicht dann auch so die nächsten Kapitel dieser Vorlesung interessanter. Ja, jetzt könnte man natürlich sagen, ja brauche ich denn wirklich diese Leistung? Woher kommt denn dieses Gerücht, dass ich ständig mehr Leistung brauche? Ja, ich brauche sie tatsächlich. mehr simuliert als früher. Also egal in welchem Bereich, ob sie Maschinenbau, Autos gegen die Wand fahren oder ob sie in der Chemie irgendwelche Moleküle miteinander reagieren lassen, es wird vielfach eben simuliert mit großen numerischen Programmen und die brauchen Leistung. Es gibt neue nicht-numerische Anwendungsgebiete, die natürlich auch Leistung brauchen. Dann habe ich sehr viele verteilte Systeme heutzutage, die miteinander kommunizieren müssen. Auch Dann in der Industrie, ja eben gerade angesprochen, große Simulationsaufgaben. Dann haben wir die optische Bildverarbeitung und Schriftenerkennung. Also die Systeme, die sind ja auch immer ausgeklügelt. Die Systeme bei der Post, die handgeschriebene Adressen erkennen, die können inzwischen besser lesen als das menschliche Auge. Da stecken natürlich auch unheimliche Algorithmen dahinter, die dann entsprechend in Real-Time zuschlagen müssen. Banken, Börseninformationssysteme, ist ja auch immer wieder in der Presse, je schneller man eine Information hat, desto schneller kann man natürlich reagieren. Dann, wenn Sie sich überlegen, Google, Amazon, diese Firmen, welche Serverformen die betreiben, was da alles dahinter steckt, an Leistung natürlich und sehr viel in der Informatik, da muss man auch mal realistisch sein, sind einfach Daten, die irgendwo gespeichert werden, und die können natürlich auch nicht schnell genug sein. Also das Arbeitsfeld ist groß, das heißt wir werden auch in Zukunft noch mehr Performance brauchen, zu Kosten, die man sich natürlich leisten können muss. Was vielleicht in Zukunft noch ein bisschen wichtiger wird, ist die Leistungsaufnahme, das heißt die Metrik, wie viel

Watt brauche ich pro Instruktion. sie möchten natürlich dass die akku länger als einen tag hält und deshalb sind natürlich stromsparende schnelle prozessoren auch wieder ein thema was immer wichtiger aber auch im smartphone wissen sie auch da haben sie auch multi kurs drin also so ein modernes smartphone hatten acht oder vier oder acht kurs die da am werkeln sind also die die architektur der multi kurs die ist natürlich in energiesparenden Prozessoren verbaut. Gut, also, worum geht es jetzt die letzten drei, vier Vorlesungen in diesem Semester, also bis Ende Januar? Wir haben uns jetzt gerade so eine Motivation und eine Einführung angeschaut, warum brauchen wir überhaupt Parallelrechner, warum sind die in Zukunft wichtig? Was für mich auch immer wichtig ist, dass ich mal so einen kleinen geschichtlichen Blick mir gönne, Das heißt, man kann sehr viel aus Geschichte lernen, auch wenn es immer wieder Menschen gibt, die das nicht tun. Wir als Wissenschaftler sollten doch mal immer zurückschauen, um daraus unsere Schlüsse für die Zukunft zu ziehen. Deshalb haben wir so die wichtigsten Milestones bei der Entwicklung von Parallelrechnern uns nochmal kurz Revue passieren lassen. Wir werden uns Best-Practice-Beispiele anschauen, das heißt Architekturbeispiele, die eben maßgeblich am Erfolg von Parallelrechnern beteiligt waren. dann haben wir natürlich immer das Problem der Verbindungsnetzwerke. Wir haben Komponenten, wir haben Prozessoren, wir haben Speicher. Das muss alles irgendwie möglichst gut miteinander verbunden werden. Da gibt es natürlich unterschiedliche Möglichkeiten, die wir betrachten wollen. Und wir wollen ein bisschen auch mathematische Betrachtungen machen, das heißt quantitative Betrachtungen. Wie kann ich denn Parallelität überhaupt messen? Und wie kann ich denn möglicherweise Vorhersagen treffen, x mal so schnell ist oder nicht. Bevor ich das mühsam implementiere, dass ich mir einfach vorher mal anschaue, was steckt denn eigentlich an Parallelitäten zum Programm drin. Das ist also der Stoff, mit dem wir uns noch beschäftigen wollen. Gut, es gibt ein bisschen Literatur. Das sind hier die Hinweise, wo Sie interessante Informationen zum Thema Parallelrechner, Programmierung finden, dass sie brauchen diese Literatur natürlich nicht unbedingt, sondern Vorlesungsstoff befindet sich im Wesentlichen auf den Folien und bei meinen Erklärungen, wenn jemand das Thema allerdings natürlich speziell interessiert, dann kann er da gerne nochmal nachschauen.