Vorlesung 9

Aufgabe 1

How do bandwidth-bound computations differ from compute-bound computations?

Bandwidth-bound = Laden von RAM in die CPU ist teuer, auch wenn CPU sehr schnell ist, muss sie trotzdem auf die Daten warten und bei Compute-bound ist der Engpass nicht die Datenübertragung, sondern die tatsächliche Leistung der CPU, und wie schnell die Operationen ausgeführt werden, hängt von der CPU selbst ab

Aufgabe 2

Explain why temporal locality and spatial locality can improve program performance.

Temporal locality

Programme haben oft Schleifen oder wiederholte Zugriffe auf dieselben Variablen => wenn Daten gerade verwendet wurden, sind sie wahrscheinlich immer noch im Cache => Zugriff auf den Cache ist viel schneller als der Zugriff auf RAM => durch temporale Lokalität ist die Wahrscheinlichkeit höher, dass die benötigten Daten schon im Cache sind => das verkürzt Zugriffszeiten

Spatial locality

Caches laden nicht nur einzelne Daten, sondern ganze Cachelines => wenn also ein Programm auf ein Element zugreift, werden auch die danebenliegenden Daten geladen => bei nachfolgenden Zugriffen auf benachbarte Speicheradressen (z.B in einem Array) => diese Daten liegen schon im Cache => das verkürzt Zugriffszeiten

Aufgabe 3

Select one slide from the lecture, research more about the topic, and report on it.

Ich habe mir die Folie zu Streaming Stores ausgesucht.

Normalerweise wenn ein Programm Daten in den Speicher schreibt, werden sie zuerst in den Cache geladen, verändert und dann irgendwann zurück in den RAM. Bei großen Datenmengen kann das ineffizient sein, vor allem wenn sie nur einmal geschrieben und danach nicht mehr gelesen werden.

Das ist aus 2 Gründen blöd:

1) Cache füllt sich mit Daten, die nicht wiederverwendet werden, und verdrängt andere nützliche Daten
2) Daten erst in den Cache zu schreiben und dann in den RAM kostet unnötig Zeit

Um das technisch zu realisieren, gibt es Instruktionen, z.B MOVNTDQ (Move Non-Temporal Double Quadword) in SSE/AVX. Mit diesen Befehlen werden die Daten „markiert“ und dann über „Write Combining Buffer“ in den RAM geschrieben. White Combining ist eine kleine Puffer-Zone in der CPU, sie fasst Schreiboperationen zu größeren Blöcken um die Anzahl der Speicherzugriffe zu reduzieren.

Aufgabe 4

Read the paper, Discuss two things you find particularly interesting.

1. Ich fand interessant, wie wichtig und unterschiedlich der Umgang mit Schleifen sein kann:

„Loop Interchange“ – Reihenfolge von 2 verschachtelten Schleifen wird vertauscht. Das Ziel: Lokalität verbessern, Stride reduzieren, weniger RAM Zugriffe.

„Loop Fusion“ – 2 benachbarten Schleifen mit gleichem Iterationsraum werden zu einer Schleife kombiniert. Ziel: weniger Overhead (weil weniger Verwaltungsaufwand mit Zähler erhöhen, Bedingungen prüfen..), auch bessere Lokalität, weniger Cache-Misses.

„Loop Blocking“ – teilt eine große Schleife in kleinere cache-freundliche Blöcke auf. Das Ziel: bessere Datenwiederverwendung

2. Als Fortsetzung von Punkt 1: Schleifenoptimierungen reichen nicht, wenn die Speicheranordnung selbst ineffizient ist. Da gibt es andere „Data-Layout“-Optimierungstechniken:

"Array Padding“ (das hatten wir schon in der VL), die Idee ist – wenn 2 Arrays in der selber Cakommcheline landen und abwechselnd angesprochen werden – kann man Padding zwischen Arrays einfügen, sodass sie unterschiedliche Cachelines nutzen => weniger Kollisionen im Cache

„Array Merging“ – wenn 2 Arrays zusammen genutzt werden, aber weit auseinander im Speicher liegen (schlechte Lokalität), kann man sie zu einer Struktur kombinieren, damit sie nebeneinander liegen => bessere Cache-Nutzung und weniger Misses

„Array Transpose“ – wenn ein Array ineffizient angesprochen wird (z. B. spaltenweise bei row-major order), kann man die Dimensionen vertauschen (so ähnlich wie Loop Interchange) => reduziert Strike, verbessert Lokalität