**1. Unterschiede zwischen Bandbreiten-gebundenen und Rechen-gebundenen Berechnungen**

Bei rechengebundenen (compute-bound) Berechnungen ist die CPU der Engpass. Das bedeutet, dass die Geschwindigkeit des Programms hauptsächlich von der Rechenleistung der CPU abhängt. Ein typisches Beispiel ist die numerische Integration zur Berechnung von Pi (wie in der Vorlesung gezeigt). Hier kommt es darauf an, die Algorithmen effizient zu gestalten und, wenn möglich, parallele Berechnungen zu nutzen, um die Ausführungszeit zu verkürzen.

Im Gegensatz dazu sind bandbreitengebundene (bandwidth-bound) Berechnungen dadurch gekennzeichnet, dass der Engpass beim Speicherzugriff liegt. Ein Beispiel hierfür ist die Summierung aller Elemente einer Matrix (auch wie in der Vorlesung gezeigt), bei der die Zugriffe auf den Hauptspeicher dominieren. Die Optimierung solcher Berechnungen erfordert insbesondere Maßnahmen zur Verbesserung der Cache-Nutzung, wie die Berücksichtigung von räumlicher (spatial locality) und zeitlicher (temporal locality) Lokalität.

**2. Verbesserung der Programmausführung durch Temporal und Spatial Locality**

Die zeitliche Lokalität (Temporal Locality) bezieht sich darauf, dass ein Speicherbereich, der einmal referenziert wurde, in naher Zukunft erneut genutzt wird. Dies ist besonders vorteilhaft, da die entsprechenden Daten häufig im Cache verbleiben und so weniger Zugriffe auf den langsamen Hauptspeicher notwendig sind.

Die räumliche Lokalität (Spatial Locality) beschreibt, dass bei der Referenzierung eines Speicherbereichs benachbarte Speicherbereiche wahrscheinlich bald darauf ebenfalls referenziert werden. Dies ermöglicht eine effiziente Nutzung von Cache-Lines, insbesondere bei Stride-1-Zugriffen, bei denen Daten sequentiell im Speicher abgelegt und genutzt werden. Beide Konzepte tragen wesentlich dazu bei, die Effizienz von Programmen zu steigern, indem sie Speicherzugriffe minimieren.

**3. Recherche zu einem Thema der Vorlesung: „Streaming Stores“**

Streaming Stores ermöglichen es, Daten direkt in den Arbeitsspeicher zu schreiben, wobei der Cache bewusst umgangen wird. Diese Technik ist besonders effizient bei speichergebundenen Aufgaben, bei denen die geschriebenen Daten später nicht erneut gelesen werden. Ein typisches Beispiel für die Implementierung von Streaming Stores sind AVX-Instruktionen, die Daten direkt in den RAM schreiben.

Allerdings birgt die Nutzung von Streaming Stores auch Risiken. Eine falsche Anwendung kann zu Leistungseinbrüchen führen, etwa durch unnötige Invalidierungen von Cache-Lines. Daher ist ein fundiertes Verständnis der Speicherzugriffscharakteristik des Programms notwendig, um diese Technik effektiv einzusetzen.

**4. Diskussion über den Artikel „Cache Optimization Techniques“**

Ein interessantes Konzept aus dem Artikel sind die Cache-oblivious Algorithms. Diese Algorithmen sind so gestaltet, dass sie die Cache-Hierarchien optimal nutzen, ohne explizite Parameter wie Cache-Größe oder Cache-Line-Länge zu kennen. Sie basieren auf rekursiven Divide-and-Conquer-Ansätzen, die die Anzahl der Cache Misses reduzieren. Der Vorteil liegt in der Robustheit gegenüber unterschiedlichen Hardware-Konfigurationen, während die Implementierung solcher Algorithmen jedoch komplex ist.

Ein weiteres Thema sind Blocking-Techniken, bei denen Datenblöcke so angepasst werden, dass sie vollständig in den Cache passen und wiederverwendet werden können. Diese Technik verbessert die Cache-Trefferquote, insbesondere bei speicherintensiven Anwendungen. Beide Ansätze zeigen, wie wichtig ein tiefes Verständnis der Speicherarchitektur ist, um die Programmausführung zu optimieren.  
(Fun Fact: im letzten Semester haben wir bei Professor Bücker in der Veranstaltung Efficient Computing genau dasselbe Paper behandelt, da das Thema der Cache Optimization ein großer Teil der Veranstaltung war)