Performance Modellierung
Roofline Modell

Theodor Wulff

Inhalte

- Motivation Modellierung
- Roofline Modell
  - Allgemeine Funktionsweise
  - Naive Roofline
  - Bandbreiten Limits
  - CPU Performance Limits
  - Arithmetic Intensity
- Anwendungsbeispiel

Warum modellieren wir?

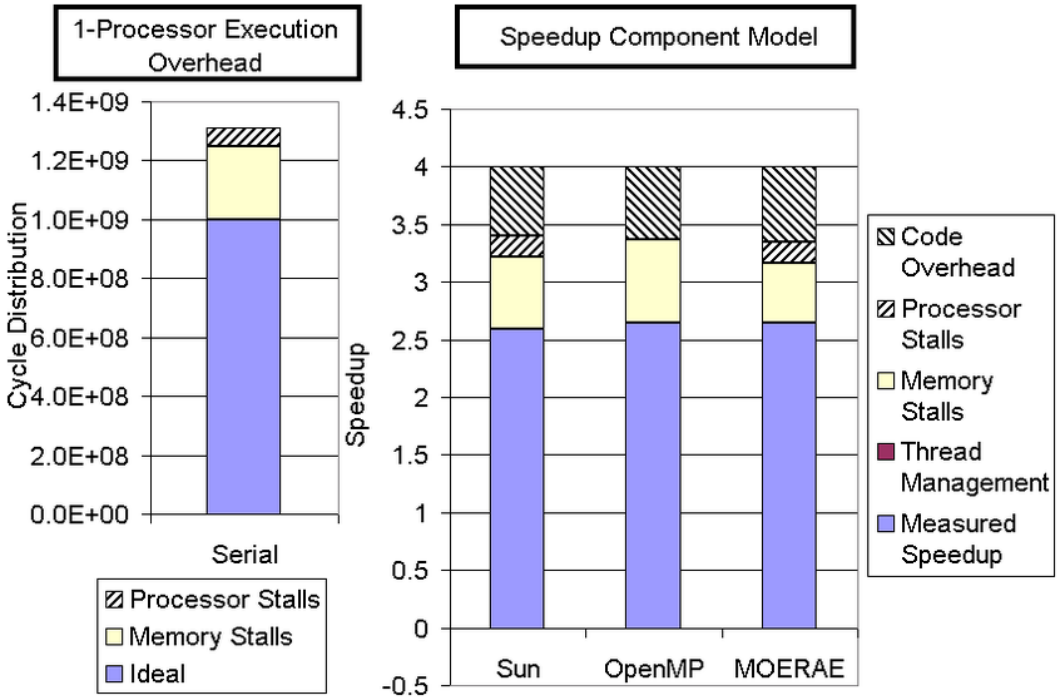
- Voraussagen über das Laufzeitverhalten eines Systems treffen
- Einflüsse zukünftiger Änderungen verstehen
  - Änderung der Scheduling Regeln

→ Modelle sollten verständlich sein: Praxisnutzen und Lesbarkeit beachten

Was modellieren wir?

Stochastische/ Statistische Analyse Modelle

- Sehr genaue Wiedergabe des Systemverhaltens
- Oft keine Optimierungsmöglichkeiten ablesbar

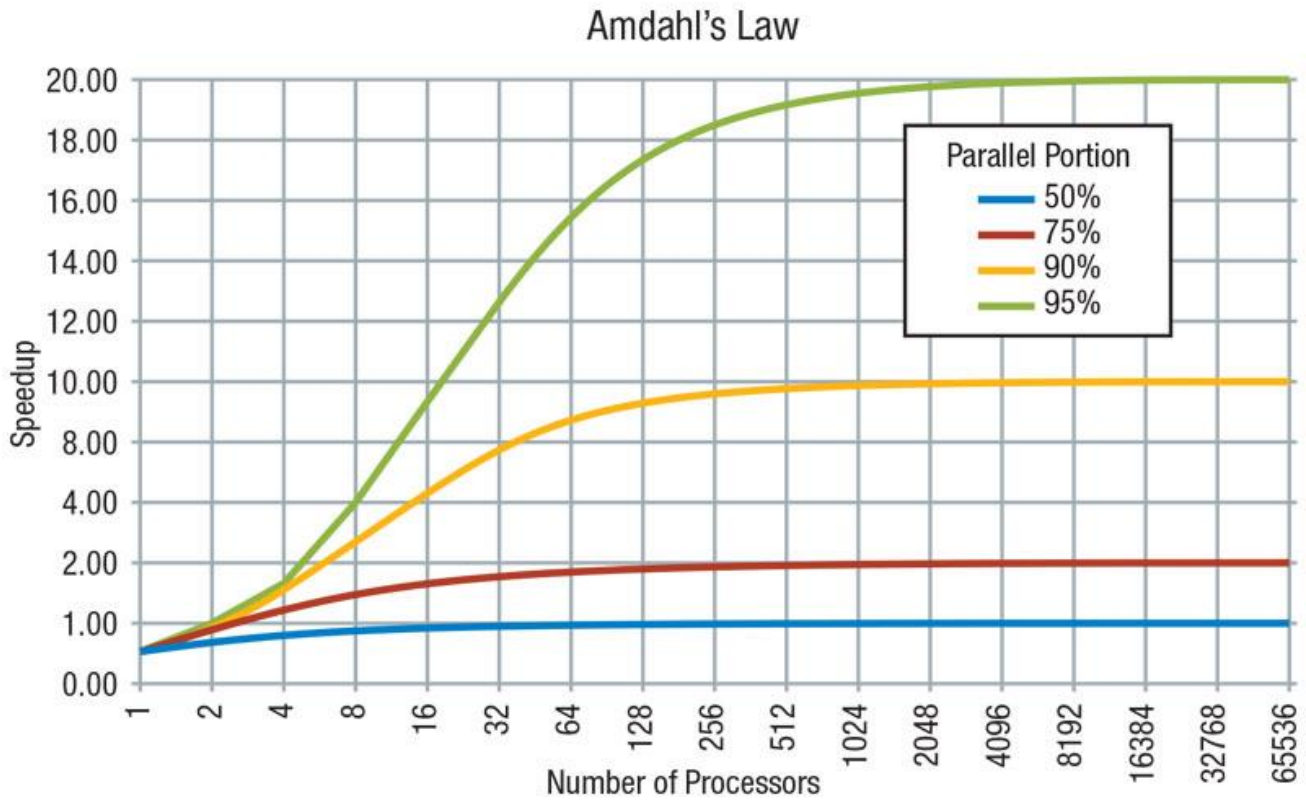


https://www.researchgate.net/figure/Performanceof-the-Major-Loops-in-MDG-P-is-the-number-of-processors-Negative-components_fig3_2470384

Was modellieren wir?

Bound and Bottleneck Analyse

- Amdahls Law
  - Einfluss auf die Laufzeit durch Erhöhung der Prozessoranzahl
  - Simples Erhöhen der Prozessoranzahl hat nicht unbedingt einen deutlichen Einfluss auf die Leistung eines Systems



https://leonardoaraujosantos.gitbooks.io/opencl/content/performance.html

Roofline Modell

Roofline Modell

"We propose an **easy-to-understand**, **visual** performance model that offers **insights** to programmers and architects **on improving parallel software and hardware** for floating point computations."

- *Samuel Williams, Andrew Waterman, and David Patterson*

Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures

Allgemeines

- Angenommen Speicherbandbreite ist die oft einschränkende Ressource
- Bound and Bottleneck Analyse
- Gemessene Performance erlaubt Rückschlüsse auf mögliche Optimierungen
  - Rückschlüsse auf Ende der Optimierungen
- Erweiterbar auf eigene Bedürfnisse

Metrik

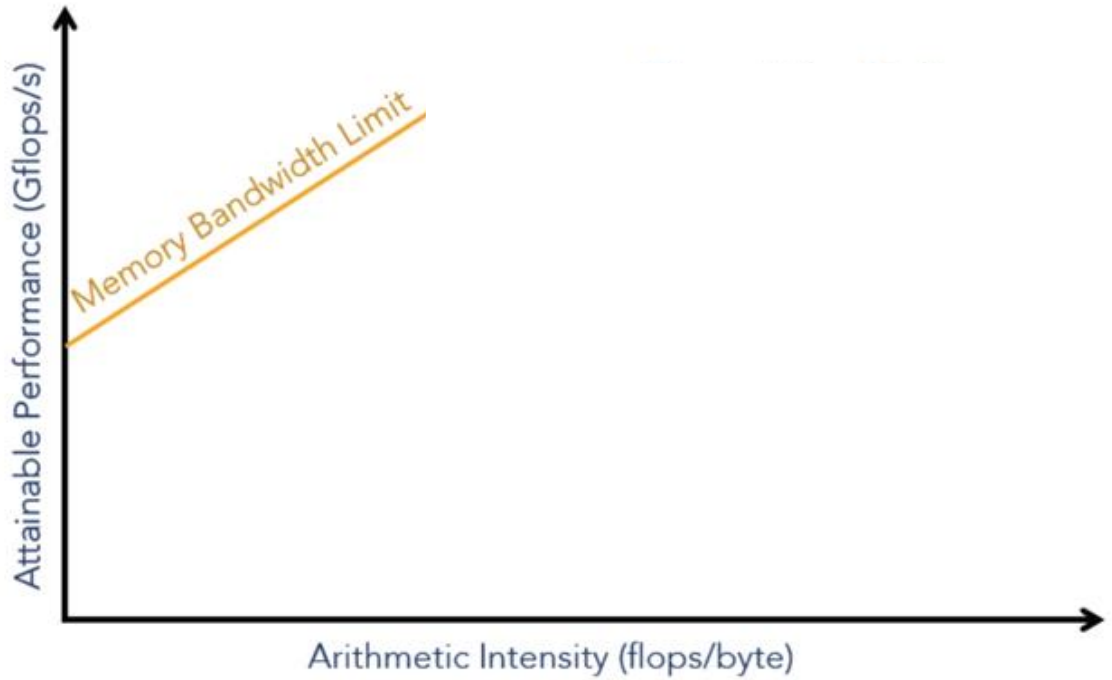
- Berechnungsarbeit: „Floating point operations per second“ FLOP/s → GFLOP/s
- Kommunikation: Byte/s → GB/s

$$"Arithmetic\ Intensity" = \frac{Berechnungsarbeit}{Kommunikation}$$

- Logarithmische Skalierung erleichtert Anpassung an Moore‘s Law

Naive Roofline

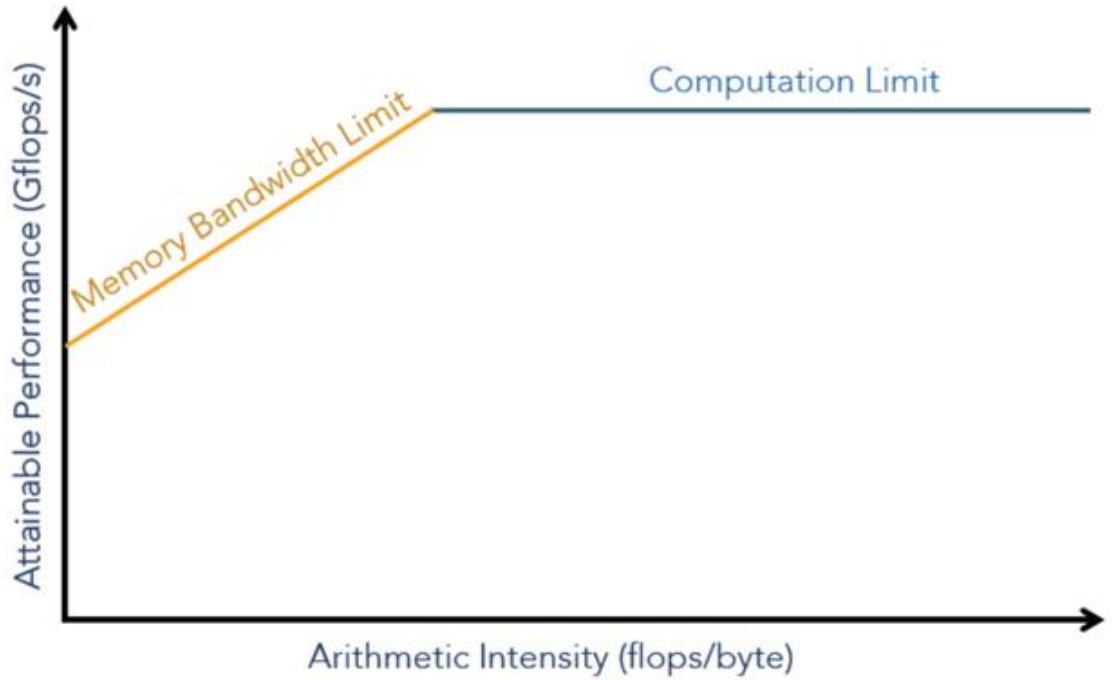
- Grenzen nur durch Bandbreite



https://www.codeproject.com/Articles/1191905/Optimizing-Application-Performance-with-Roofline

Naive Roofline

- Grenzen nur durch Bandbreite und CPU Performance

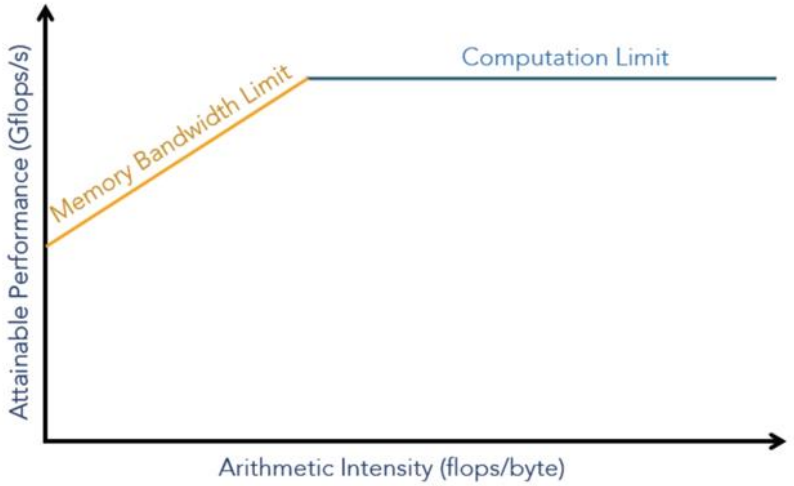


https://www.codeproject.com/Articles/1191905/Optimizing-Application-Performance-with-Roofline

Naive Roofline

- Grenzen nur durch Bandbreite und Performance

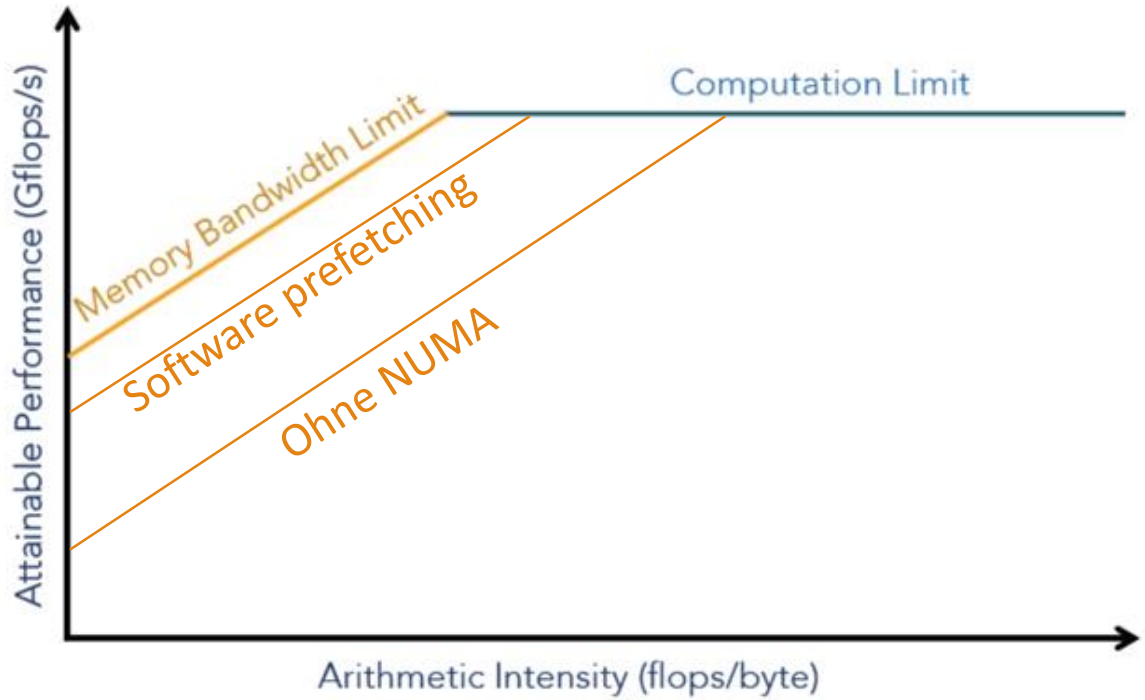
$$Erreichbare\ Performance = min \begin{cases} max.\ Floating\ Point\ Performance \\ max.\ Speicherbandbreite\ * Arithmetic\ Intensity \end{cases}$$



https://www.codeproject.com/Articles/1191905/Optimizing-Application-Performance-with-Roofline

Bandbreiten Limits

- DRAM Bandbreite
  - $Parallelismus * Frequenz$
- Software Prefetching
- NUMA – Non-uniform memory access



Bandbreiten Limits

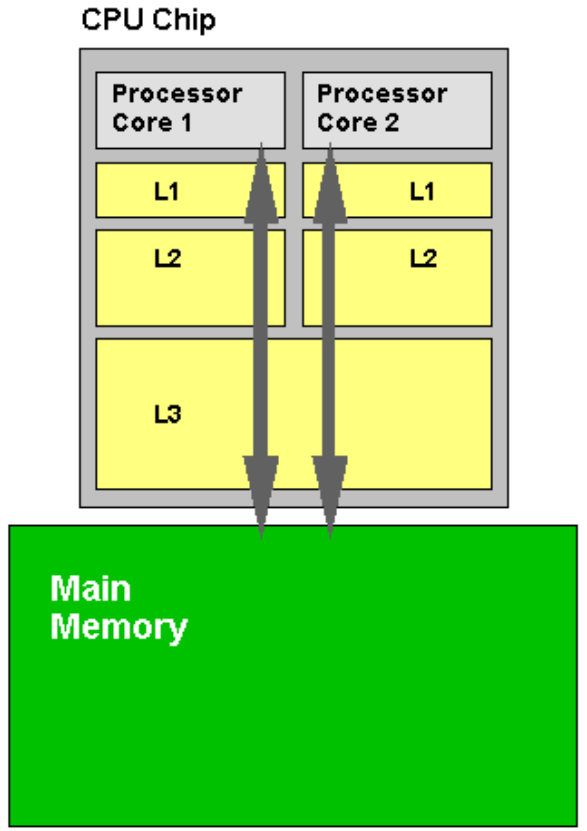
Latenz

- Cache Line Transfer
  - Datentransfereinheit zwischen Cache und Hauptspeicher
- Little‘s Law
  - $L = \lambda W$
  - L = durchschnittlich wartende Items
  - W = durchschnittliche Wartezeit
  - λ = durchschnittlich ankommende Items pro Zeiteinheit

Bandbreiten Limits

Cache Line Spatial Locality

- Spatial Locality
  - Nah beieinander liegende Adressen haben gute Chancen in kurzer Zeit geladen zu werden
- Cache Miss resultiert in Laden einer Cache Line
  - Verbraucht Bandbreite
  - Weniger als die Hälfte der geladenen Daten wird i.d.R. genutzt



https://www.pcmag.com/encyclopedia/term/39177/cache

Performance Limits

Instruction Level Parallelism (ILP)

- Anzahl der parallel ausführbaren Operationen

Zur Verdeutlichung:

1. e = a + b
2. f = c + d
3. g = e * f

- 1 und 2 haben keine Abhängigkeiten -> parallel ausführbar
- 3 hängt von 1 und 2 ab -> nicht parallel ausführbar

Performance Limits

Instruction Level Parallelism (ILP)

- Anzahl der parallel ausführbaren Operationen

Zur Verdeutlichung:

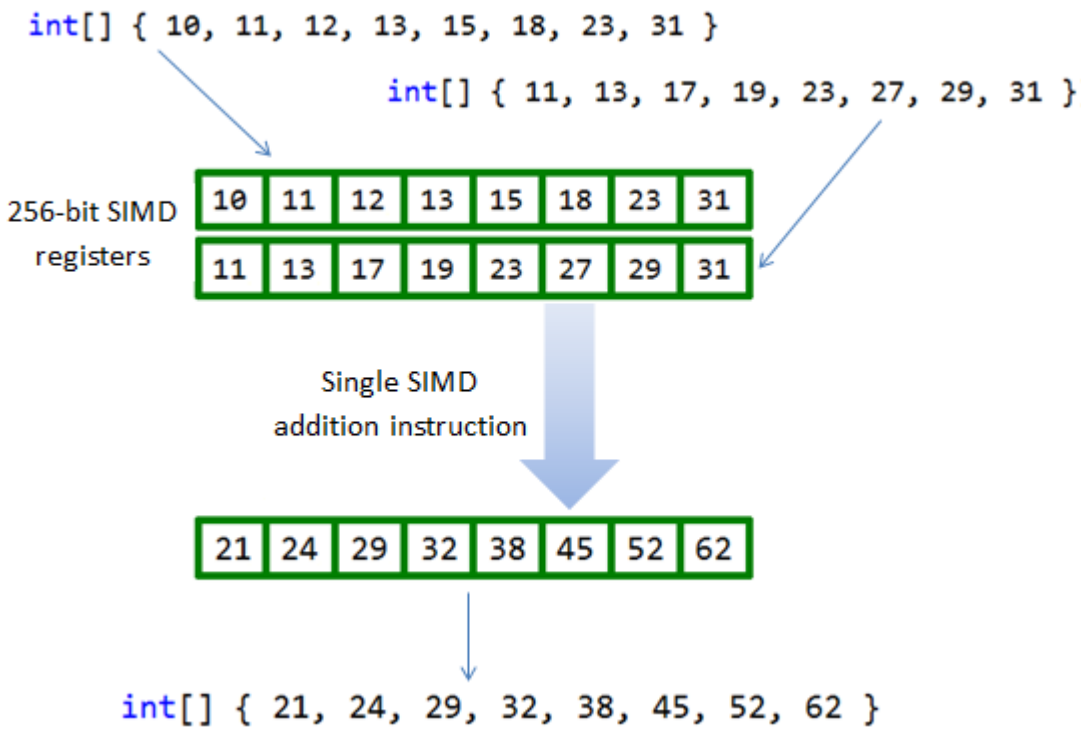
1. e = a + b
2. f = c + d
3. g = e * f

Angenommen eine Operation lässt sich in einer Zeiteinheit ausführen:

→ ILP = 3/2

Performance Limits

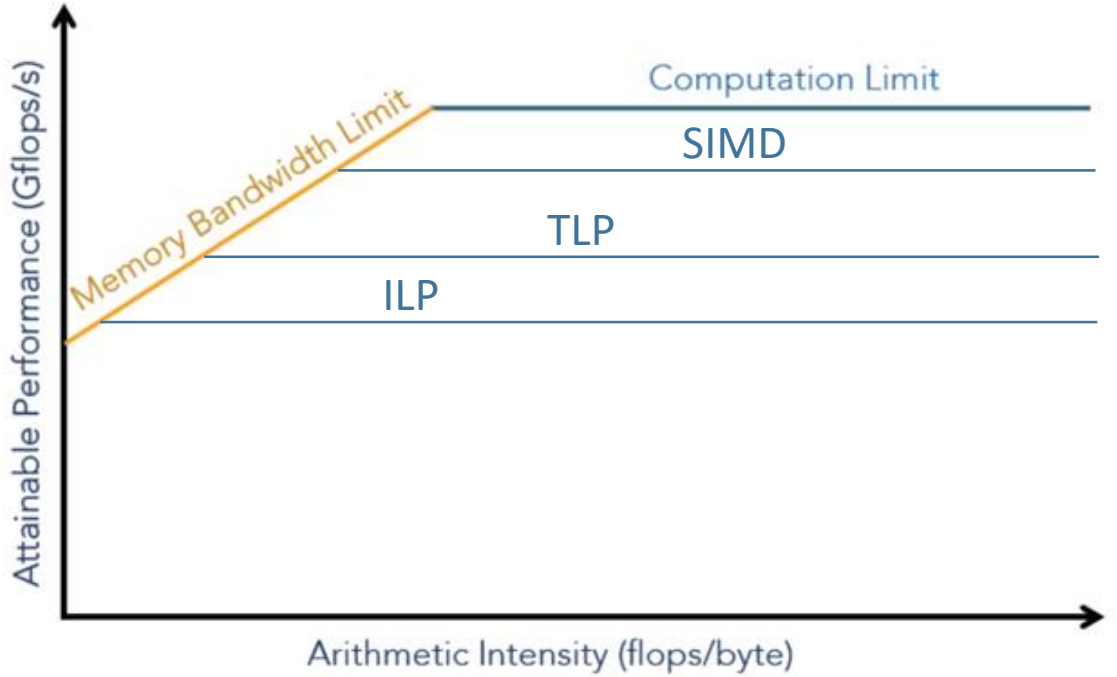
- Instruction Level Parallelism (ILP)
- Task Level Parallelism (TLP)
  - Paralleles Ausführen von Threads
- SIMD – Single Instruction Multiple Data



https://instil.co/2016/03/21/parallelism-on-a-single-core-simd-with-c/

Performance Limits

- Instruction Level Parallelism (ILP)
- Task Level Parallelism (TLP)
  - Paralleles Ausführen von Threads
- SIMD – Single Instruction Multiple Data



https://www.codeproject.com/Articles/1191905/Optimizing-Application-Performance-with-Roofline

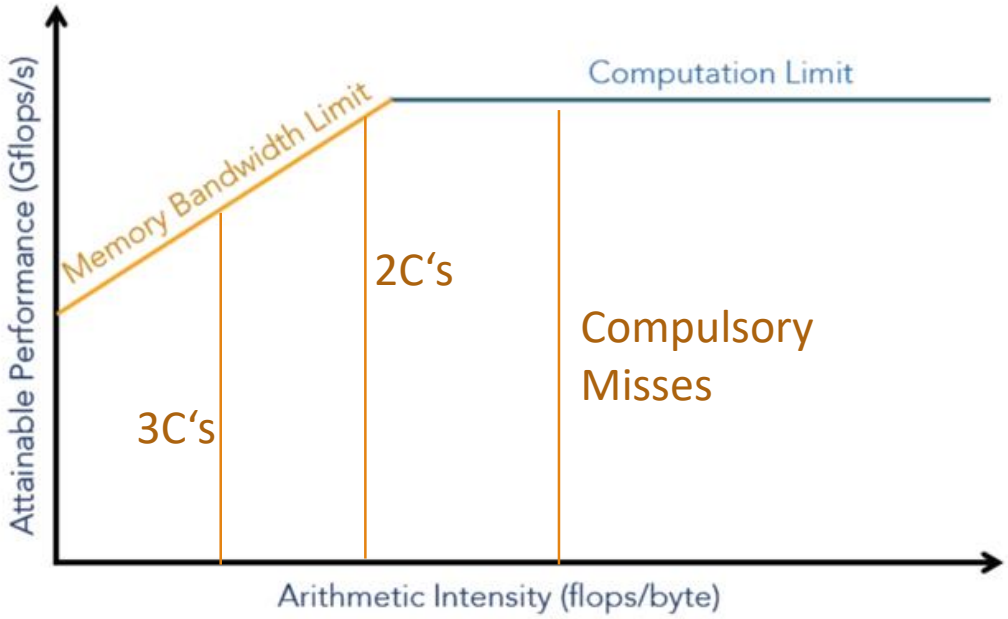
Arithmetic Intensity Walls

- Cache Organisation beeinflusst Arithmetic Intensity
- Arten von Cache Misses als Intensity Walls
  - Compulsory Miss – Erster Aufruf bei leerem Cache
  - Capacity Miss – Cache kann nicht alle Daten halten
  - Conflict Miss – 2 Blöcke werden auf den selben Ort gemapped

$$"Arithmetic\ Intensity" = \frac{Berechnungsarbeit}{Kommunikation}$$

Arithmetic Intensity Walls

- Cache Organisation beeinflusst Arithmetic Intensity
- Arten von Cache Misses als Intensity Walls
  - Compulsory Miss
  - Capacity Miss
  - Conflict Miss

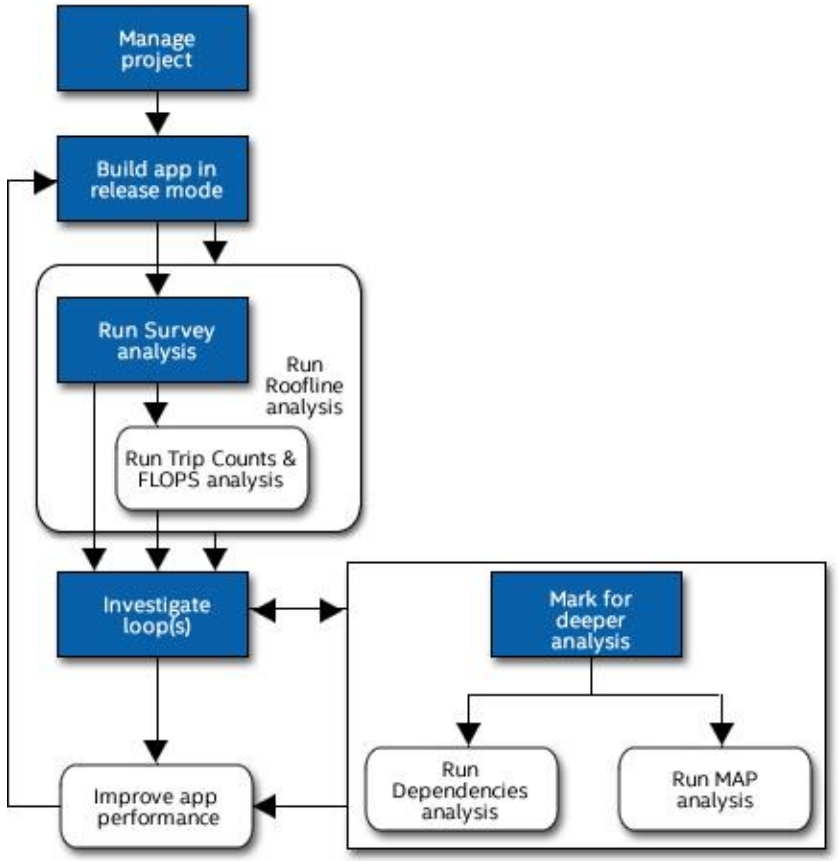


https://www.codeproject.com/Articles/1191905/Optimizing-Application-Performance-with-Roofline

Wie lässt sich das Modell anwenden?

Workflow

- Performance Modellierung als fester Bestandteil der Entwicklung
- Kein voreiliges Modellieren



https://software.intel.com/en-us/advisor-user-guide-vectorization-workflow-diagram

Anwendungsbeispiel

- 2 einfache Beispielalgorithmen, die Additionen ausführen
  - X = Ya + Yb → niedrige Arithmetic Intensity
  - X = Ya + Ya + Yb + Yb + Yb → hohe Arithmetic Intensity

- Roofline Funktion des Intel Advisor 2019
- Rechnerarchitektur:
  - CPU: Intel Core i5-6500 @ 3,20 GHz
  - Kerne: 4

Datenset

Arrays der Länge 1328

- Arrays aus Structs (AoS)
- Structs mit Arrays (SoA)
- Arrayzugriffszeit < Structzugriffszeit

Loops mit je 10.000.000 Iterationen

Skalaroperationen

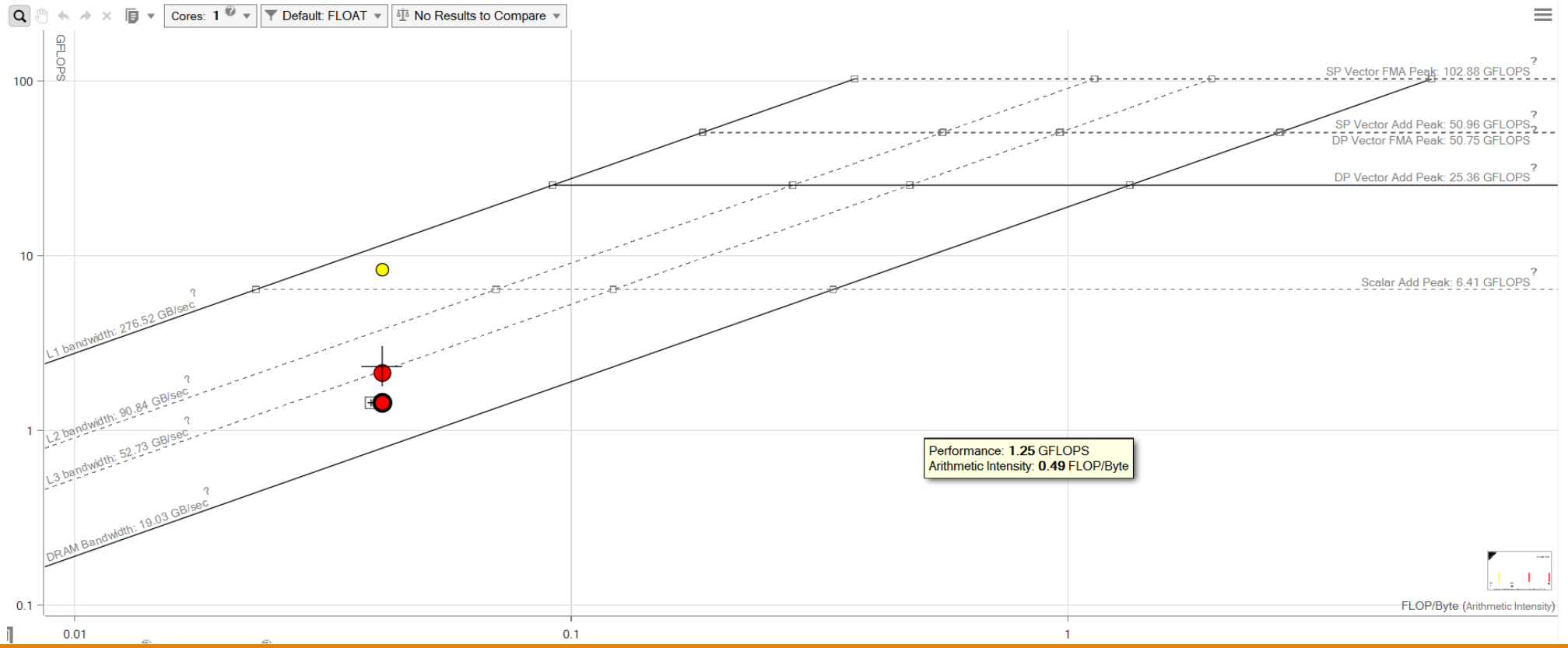
- Eine Operation pro Wertepaar

Vektoroperationen

- Eine Operation wird auf mehrere Werte angewandt

Niedrige Arithmetic Intensity

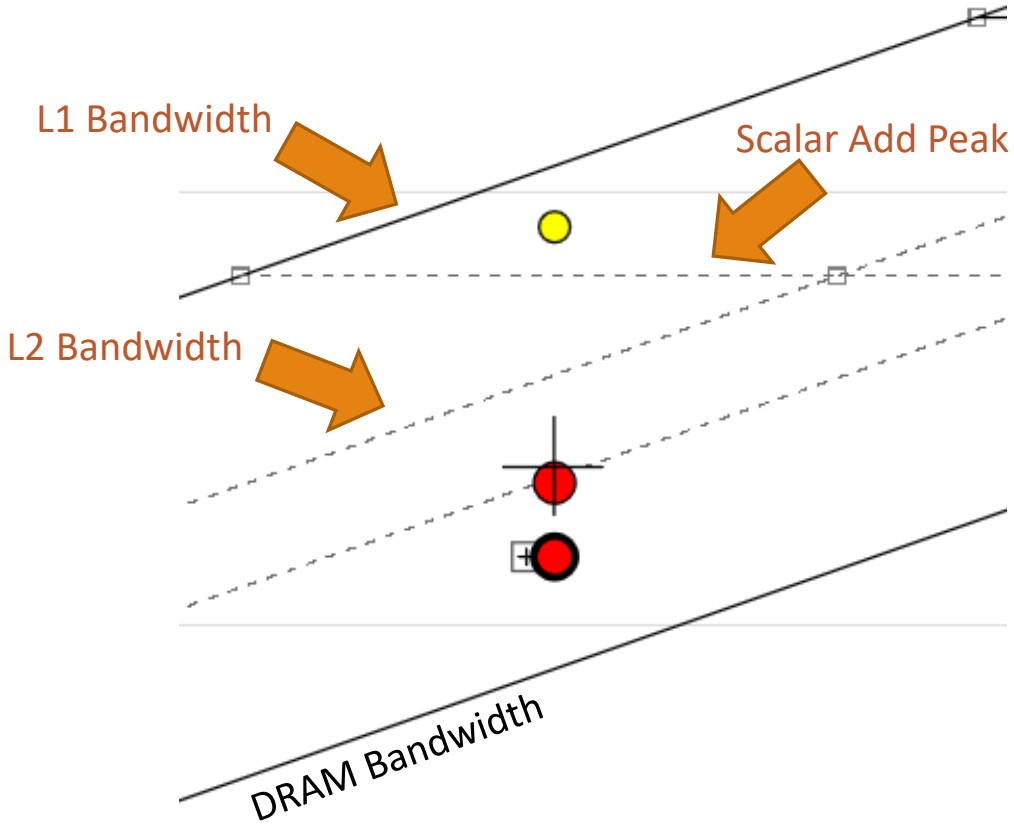
Algorithmus: X = Ya + Yb



Niedrige Arithmetic Intensity

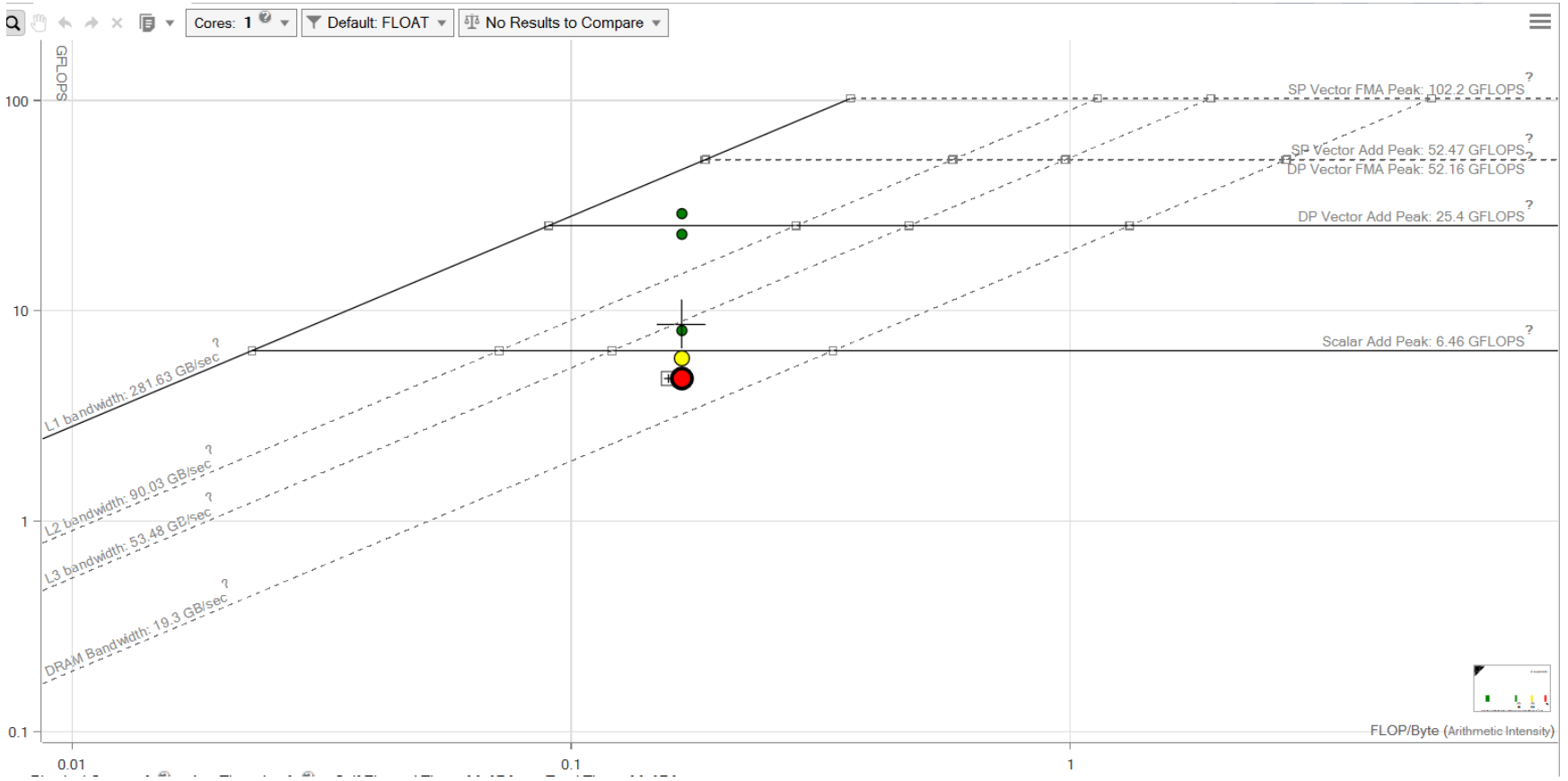
- Algorithmus: X = Ya + Yb
  1. AoS – keine Vektorisierung
  2. SoA – keine Vektorisierung
  3. SoA - Vektorisierung

- Niedrige Arithmetic Intensity lässt wenig arithmetische Optimierungen zu



Hohe Arithmetic Intensity

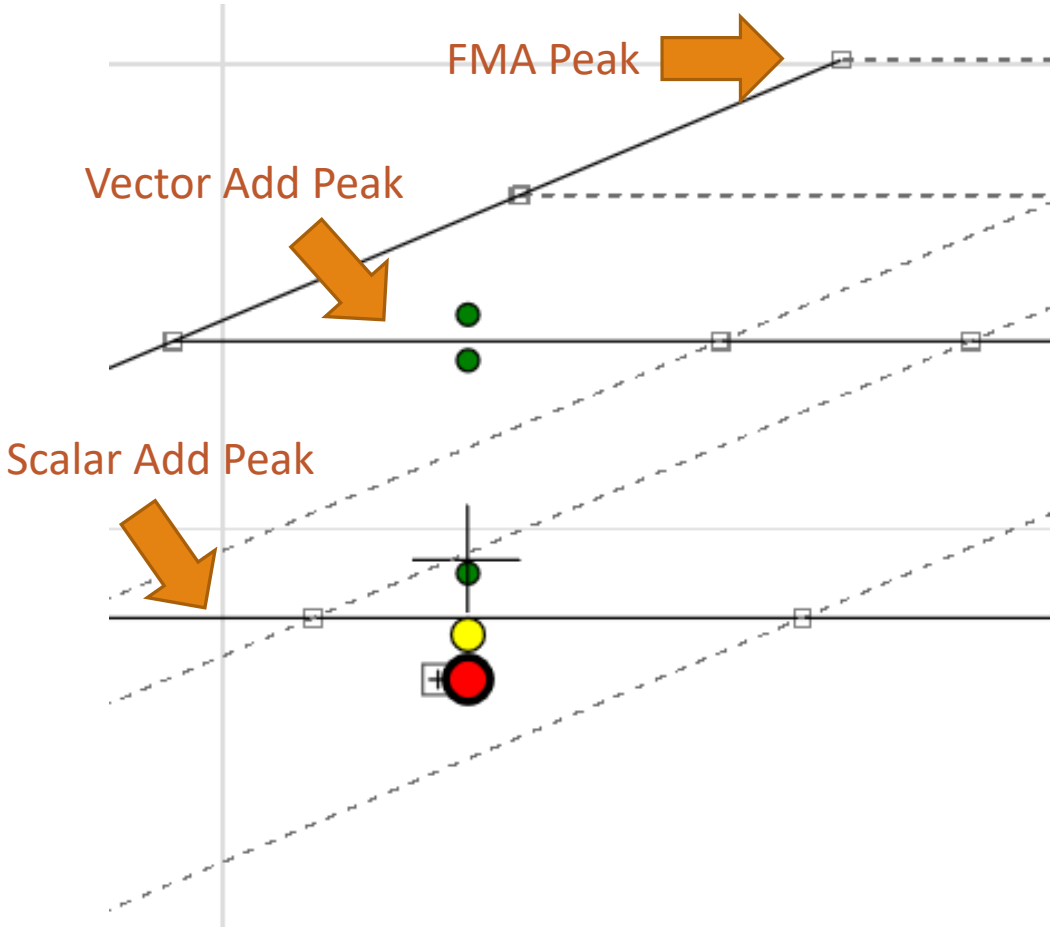
Algorithmus: X = Ya + Ya + Yb + Yb + Yb



Hohe Arithmetic Intensity

- Algorithmus: X = Ya + Ya + Yb + Yb + Yb
- FMA = Fused Multiply Add

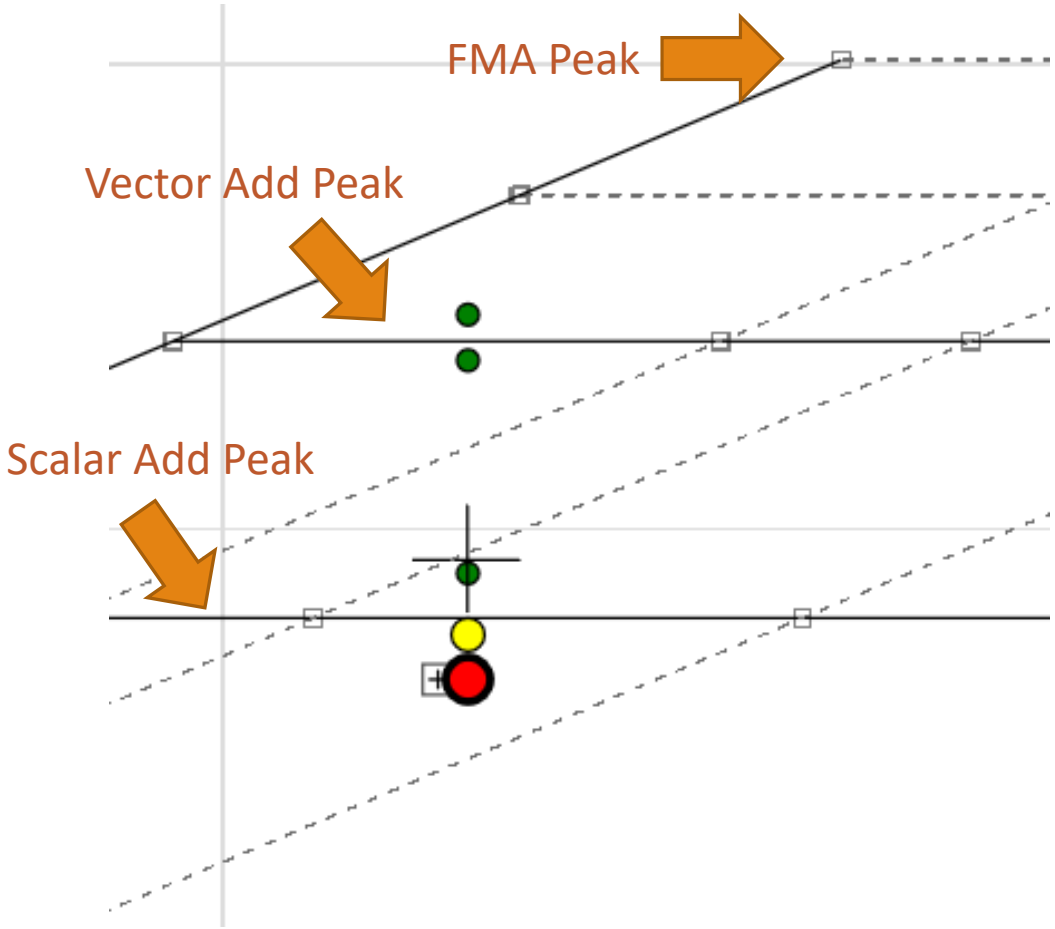
1. AoS – keine Vektorisierung
2. SoA – keine Vektorisierung
3. AoS – Vektorisierung
4. SoA – Vektorisierung
5. SoA – Vektorisierung mit FMAs



Hohe Arithmetic Intensity

- Algorithmus: X = Ya + Ya + Yb + Yb + Yb
- FMA = Fused Multiply Add

- Performance Gewinn durch Vektorisierung und Strukturveränderung
- Kaum weitere Optimierung möglich
  - Bandbreiten Limit fast erreicht



Abschließend

- Roofline Modell
  - Intuitiv lesbar
  - Erkenntnisse über Optimierungsmöglichkeiten
  - Relativ kostengünstig

- Noch viele weitere Möglichkeiten das Modell anzupassen
  - Z.B. Arithmetic Intensity → Operational Intensity

Quellen

- https://www.d.umn.edu/~gshute/arch/cache-performance.xhtml
- https://www.pcmag.com/encyclopedia/term/39177/cache
- https://instil.co/2016/03/21/parallelism-on-a-single-core-simd-with-c/
- S. Kumar and C. Wilkerson, "Exploiting spatial locality in data caches using spatial footprints," Proceedings. 25th Annual International Symposium on Computer Architecture (Cat. No.98CB36235), Barcelona, Spain, 1998, pp. 357-368.
- https://de.scribd.com/doc/33700101/Instruction-Level-Parallelism#scribd
- http://www.inf.fu-berlin.de/lehre/SS00/19540-V/bookupdate/kahmann_kretzschmar/amdahl.html
- https://medium.com/software-design/why-software-developers-should-care-about-cpu-caches-8da04355bb8a
- D. C. Little, John & Graves, Stephen. (2008). Little's Law. Building Intuition: Insights from Basic Operations Management Models and Principles. 81-100.
- https://whatis.techtarget.com/definition/NUMA-non-uniform-memory-access
- https://techdifferences.com/difference-between-array-and-structure.html
- https://www.pcmag.com/encyclopedia/term/48164/numa
- http://crd.lbl.gov/departments/computer-science/PAR/research/roofline/
- https://courses.cs.washington.edu/courses/cse378/02sp/sections/section9-2.html
- S. Williams, "The Roofline Model", *chapter in Performance Tuning of Scientific Applications,* edited by David H. Bailey, Robert F. Lucas, Samuel W. Williams, (CRC Press: 2010)

Letzter Zugriff: 05.12.2018