CFK: Kompakte Kraftbaumindizes

100000 Teilchen, Phasenprofil und Thread-Diagnose

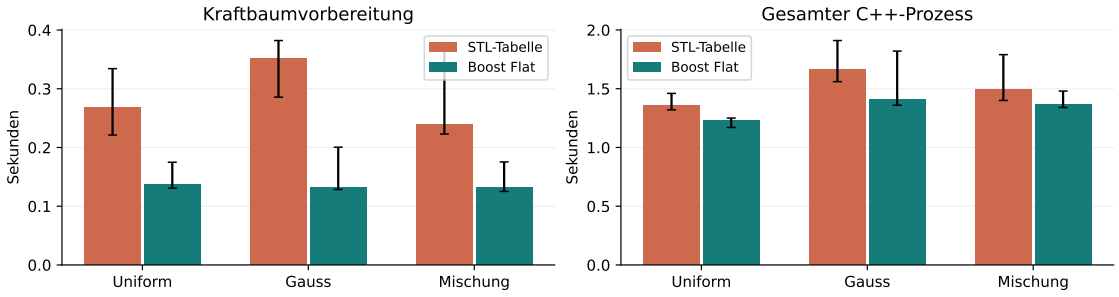
17. September 2026

1 Ergebnisse

Verglichen werden die bisherige STL-Hash-Tabelle und eine kompakte Boost-Flat-Tabelle. Beide Programme besitzen dieselbe neue Zeitinstrumentierung. Es werden keine Pruefungen entfernt und keine mathematischen Regeln geaendert. Die Hauptserie verwendet 100000 statt einer Million Teilchen: fuenf Wiederholungen je Wolke und Variante, insgesamt 30 Laeufe. Alle Beschleunigungen und Durchlaufzaehler stimmen bitweise mit den eingefrorenen Referenzen ueberein.

Wolke	STL vorb. [s]	Flat vorb. [s]	Faktor	STL ges. [s]	Flat ges. [s]
Uniform	0.2676	0.1365	1.96	1.36	1.23
Gauss	0.3519	0.1325	2.66	1.66	1.41
Mischung	0.2400	0.1326	1.81	1.49	1.37

Mediane aus fuenf identischen Eingaben; keine unabhaengigen physikalischen Stichproben. Vier Vorbereitungsthreads, zwoelf fuer Gitterzuordnung und Kraefte. Die Gesamtdauer enthaelt auch JSON-Ein-/Ausgabe. Fehlerbalken zeigen Min–Max, keine Konfidenzintervalle. Peak-RSS des C++-Prozesses (STL zu Flat): Uniform 71.4 zu 80.8 MiB, Gauss 67.2 zu 72.6 MiB, Mischung 63.2 zu 72.6 MiB.

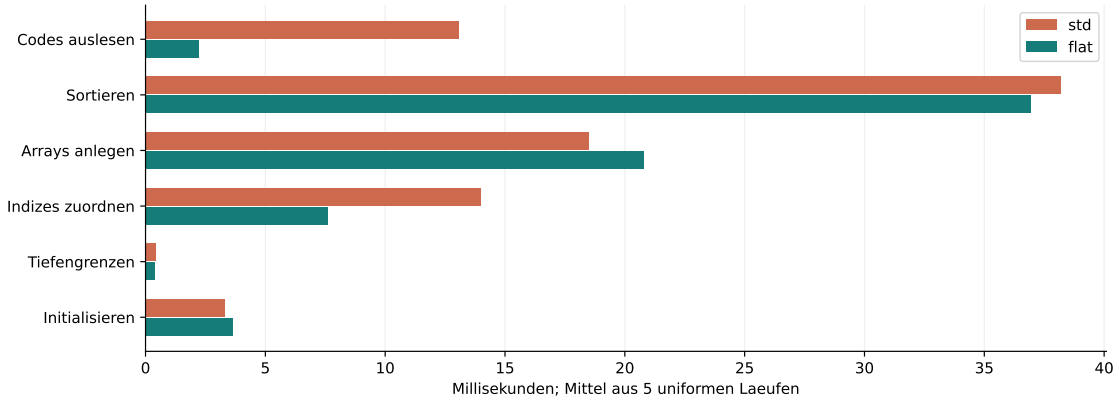


1.1 Genuegen 100000 Teilchen?

Ja, fuer schnelle Variantenvergleiche und Tests. Cache- und Kapazitaetseffekte koennen sich bei groesseren Eingaben aendern. Der einzelne Millionenlauf bestaetigt deshalb die groessere Eingabe, ersetzt aber keine statistische Skalierungsserie. Einzelner Millionen-Validierungslauf: Vorbereitung 1.764 s, Gesamt 17.14 s, Peak 729.2 MiB. Keine neue Millionen-Zeitserie.

2 Was wurde geaendert?

Die Wurzel traegt Code 1, ihre Nachfolger $2k$ und $2k + 1$. Fuer einen belegten Blattcode k sind seine Vorfahren $k \gg j$. Diese Codes werden weiterhin gesammelt und eindeutig indiziert. Geaendert wurde nur die temporaere Tabelle Code zu Speicherindex: `std::unordered_map` wird durch `boost::unordered_flat_map` ersetzt. Die kompaktere Speicherung vermeidet viele Einzelallokationen und reduziert pointerbasierte Zugriffe. Es werden keine Iteratoren oder Referenzen ueber Einfuegungen hinweg aufbewahrt. Nach der Indexierung wird die Tabelle nur noch gelesen, auch in parallelen Abschnitten. Kompakte Eintraege bedeuten nicht automatisch weniger Peak-RSS: Beide Tabellen reservieren weiterhin fuer vier Eintraege je Teilchen. Unterschiedliche Kapazitaetsstufen, Reserven und Speicherwiederverwendung koennen den Prozessbedarf erhoehen. Die Reservierungsstrategie wurde hier bewusst nicht zugleich veraendert; ihre Anpassung ist ein separater, lohnender Vergleich. Eine andere interne Iterationsreihenfolge beeinflusst nicht die Rechnung: Die Codes werden weiterhin aufsteigend sortiert. Knotenanordnung, Kindreihenfolge und Gleitkommaoperationen bleiben damit unveraendert. Mit der CMake-Option CFK_FLAT_INDEX=OFF ist die bisherige Tabelle weiterhin als Vergleichsvariante verfuegbar. Bei der uniformen Wolke sinkt Vorfahren sammeln im Median von 0.0665 auf 0.0082 s; Sortieren/Indexieren von 0.0882 auf 0.0665 s.



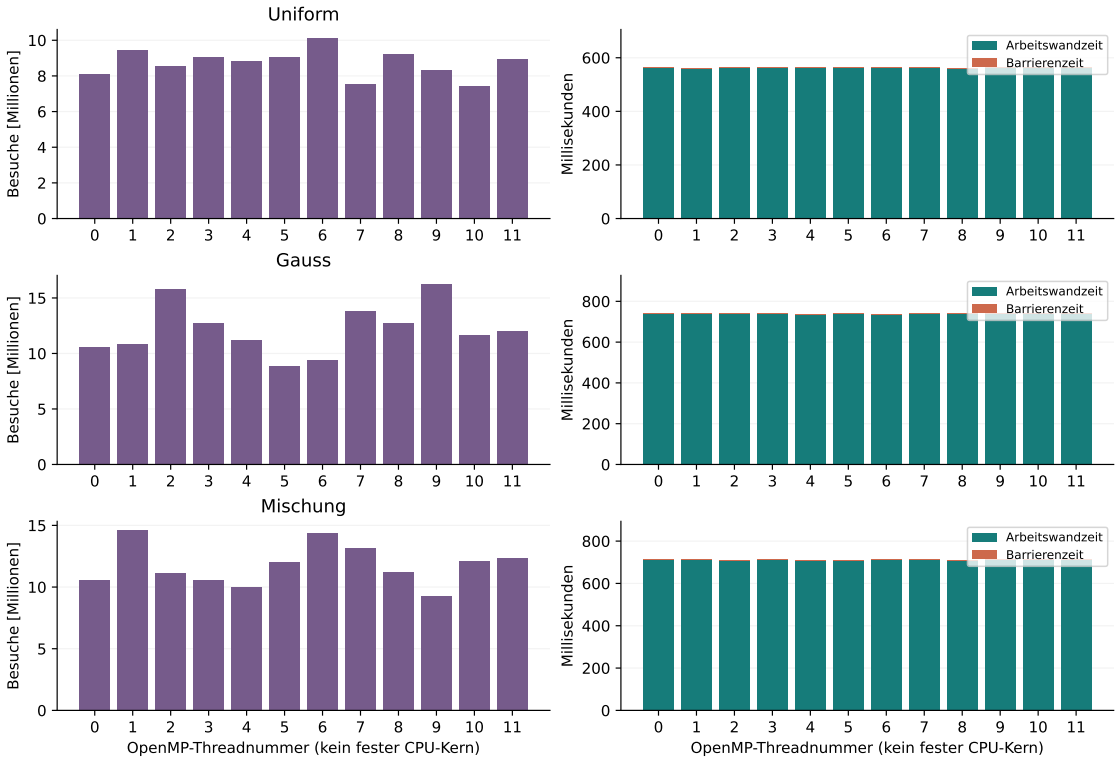
Die feineren Zeiten sind Bestandteile der bisherigen Phase Sortieren/Indexieren, keine zusaetzlichen Zeiten. Neben dem Sortieren werden Codeextraktion, Arrayallokation, Indexzuordnung, Tiefengrenzen und Knoteninitialisierung getrennt gemessen. Aus Diagramm und Rohdaten laesst sich erkennen, ob eine parallele Sortierung ueberhaupt den groessten Restaufwand treffen wuerde.

2.1 Pruefungen bleiben erhalten

Laengen, Positionen, Duplikate, IDs, Blattzuordnung, Ueberlappungen, Massen und numerische Ueberlaeufe werden weiter geprueft. Erfolgreiche Tests machen ungueltige zukuenftige Eingaben nicht unmoeglich. Eine spaetere Wiederverwendung garantierter Eigenschaften muss ueber einen abgesicherten internen Uebergang erfolgen, nicht ueber ein pauschales Abschalten der Validierung an frei veraenderlichen Objekten.

3 Load-Balancing der Kraftrechnung

Der neue optionale Diagnosemodus misst pro Thread Knotenbesuche, Arbeitswandzeit und Zeit an der Abschlussbarriere. Die Hauptzeitserie laeuft ohne Diagnose. Die folgenden Werte stammen aus je einem separaten Diagnoselauf; sie sind eine erste Beobachtung, keine Verteilungsstatistik.



Unterschiedliche Besuchszahlen bei aehnlichen Abschlusszeiten sind mit gutem dynamischem Lastausgleich vereinbar, insbesondere auf einer hybriden CPU. Arbeitswandzeit enthaelt auch Scheduling und Betriebssystemunterbrechungen. Die Barrierenzeit ist keine direkte Messung brachliegender CPU-Leistung. Diagnose fuer Gitterzuordnung und Vorbereitungs-Teilbaeume ist noch nicht implementiert.

4 Protokoll, Interpretation und naechste Schritte

Wolke	Threads	Kraftzeit [s]	Warten [%]	max/min Besuche
uniform	4	1.2881	0.10	1.51
uniform	12	0.5663	0.37	1.36
gaussian	4	1.6552	0.05	1.48
gaussian	12	0.7427	0.30	1.84
mixture	4	1.7443	0.06	1.29
mixture	12	0.7162	0.28	1.57

Der Warteanteil ist $\sum_i w_i / \sum_i (a_i + w_i)$, mit Arbeitswandzeit a_i und Barrierenzeit w_i. Erfasst sind die tatsaechlich verwendeten OpenMP-Threads; die Threadnummern identifizieren keine fest gebundenen Prozessorkerne.

4.1 Reproduzierbarkeit

GCC 13.3, Boost 1.83, C++20, Release mit -O3, ohne Fast-Math und ohne FMA-Kontraktion. Beide Varianten haben dieselben Flags bis auf die Wahl der Indextabelle. OMP_DYNAMIC=FALSE; keine Kernbindung oder Taktkontrolle. Die Reihenfolge der Varianten und Wolken wechselt deterministisch. Waehrend der Messserie werden keine Builds gestartet. Newtonsche Kraftrechnung mit classic-MAC, $\theta = 0.5$, $G = 1$, $\epsilon = 0$ und allen Zielen. Drei zusaetzliche Integer-MAC-Laeufe pruefen die archivierten 128 Ziele je Wolke. Die sechs Diagnoselaeufe verwenden alle Ziele bei vier bzw. zwoelf Kraftthreads. Alle Kraefte werden bitweise gegen CPP_G2 bzw. im Millionenfall CPP_G3 geprueft. Native Tests pruefen unter anderem Randkoordinaten, Tiefe 63, beide MACs, fehlerhafte Eingaben und die Summen der Threadzaehler. Release-Tests beider Tabellenvarianten sowie ASan/UBSan mit OpenMP werden ausgefuehrt. LeakSanitizer bleibt wegen der Umgebung deaktiviert; kein spezieller Race-Detector. Testprotokolle liegen im Rohdatenarchiv; das Protokoll des zusaetzlichen Sanitizerlaufs ohne OpenMP liegt im Berichtsordner.

4.2 Bedienung und Archiv

```
cfk_force --input input.json --output diagnose.json --all
  --threads 12 --prepare-threads 4 --diagnostics
python3 python/benchmark_index.py --output experiments/cpp_g7_neu
python3 python/summarize_index.py experiments/cpp_g7_neu
  --report ../Berichte/2026-09-17/CPP_G7_neu
```

Fortsetzungszeilen jeweils zu einem Befehl verbinden. Ohne die Option `--million` bleibt die Benchmarkserie bei 100000 Teilchen. Bei wiederholten Kraftauswertungen enthaelt JSON die Threaddiagnose der letzten Auswertung. Die Diagnose ist standardmaessig ausgeschaltet. Rohdaten, Quellstand, Programme, Flags und SHA-256-Manifest: CPP/experiments/cpp_g7_index. Sicherung vor der Aenderung: Sicherungen/2026-09-17_vor_CPP_G7/Quellstand.tar.gz.

4.3 Weitere Vorschlaege

Zuerst lohnt die Reservierungsstrategie der Flat-Tabelle, um den Speichermehrbedarf zu reduzieren. Fuer weitere Laufzeitgewinne sollte der Gitterbau untersucht werden: Er verwendet beim Packen noch die alte baumbasierte Duplikatpruefung. Der Millionen-Kontrolllauf misst dafuer rund 3.1 Sekunden, deutlich mehr als die gesamte Kraftbaumvorbereitung. Danach bietet sich die Wiederverwendung bereits sortierter Gittertopologie an. Die Lastdiagnose der Gitter-/Teilbaeume bleibt eine gesonderte Erweiterung.