CFK: Optimierungen der Aufbaupipeline

Kontrollierter Vergleich mit 100000 Teilchen

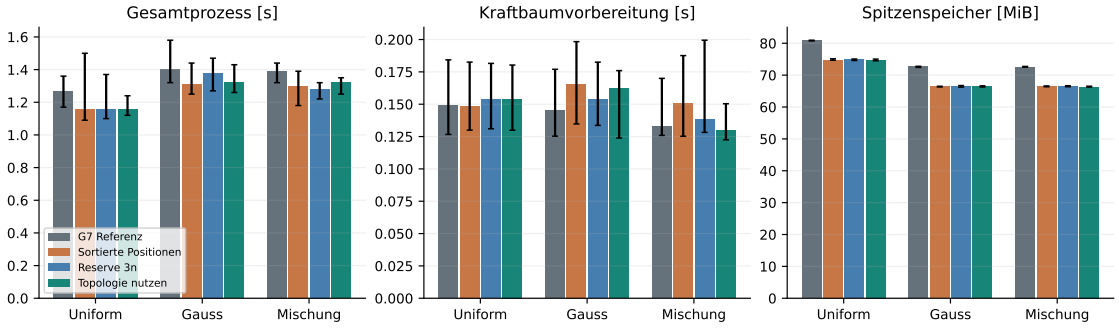
17. September 2026

1 Ergebnis und Einordnung

Drei Optimierungen wurden implementiert und getrennt verglichen: zusammenhaengend gespeicherte Positionen fuer die Duplikatpruefung im Gitterbau, eine kleinere Anfangsreserve der Indextabelle und die Wiederverwendung der sortierten Gittertopologie. Alle 60 Kraftlaeufe liefern bitweise dieselben Ergebnisse wie die eingefrorene Referenz. Die Konformitaetsregeln und die mathematische Kraftrechnung bleiben unveraendert.

Wolke	G7 [s]	G8 [s]	Ersparnis [%]	G7 [MiB]	G8 [MiB]
Uniform	1.27	1.16	8.7	80.8	74.8
Gauss	1.40	1.32	5.7	72.7	66.5
Mischung	1.39	1.32	5.0	72.6	66.3

Mediane aus je fuenf Wiederholungen identischer Eingaben. Die Gesamtdauer umfasst den C++-Prozess einschliesslich Ein-/Ausgabe. Peak-RSS ist der Spitzenspeicher des Prozesses, keine Summe von Containerkapazitaeten.



Fehlerbalken zeigen die Spannweite Min–Max, keine Konfidenzintervalle. Die vier Stufen bauen aufeinander auf; die letzte nutzt die Topologie. Die Gesamtverbesserung ist sichtbar, aber die Kraftbaumvorbereitung allein wird nicht konsistent schneller. Der robuste praktische Nutzen liegt hier vor allem in der neuen Gitter-Duplikatpruefung und ihrem kleineren Speicherbedarf.

2 Algorithmen und Sicherheitsgrenzen

2.1 Positionen sortieren statt einzeln allokieren

Bisher wurde jede Position in einen geordneten Suchbaum eingefuegt. Jetzt werden die Paare (x,y) in einem Vektor gesammelt, lexikographisch sortiert und benachbarte Eintraege auf Gleichheit geprueft. Beide Verfahren benoetigen $O(n \log n)$ Vergleiche. Der Vektor vermeidet jedoch Einzelallokationen und pointerbasierte Suchpfade. Sein Speicher wird vor dem eigentlichen Gitteraufbau freigegeben. Endlichkeit und Gebietszugehoerigkeit werden vor dem Sortieren geprueft; positive und negative Null gelten als gleiche Position.

2.2 Reservierung ist keine Kapazitaetsgrenze

Ohne vorhandene Topologie wird anfangs fuer $3n$ statt $4n$ Indizes reserviert. Bei Bedarf darf die Tabelle weiterhin wachsen. Die Zahl der Vorfahren kann bei stark konzentrierten Eingaben deutlich groesser ausfallen. Ein Faktor drei ist daher eine Speicherheuristik, keine mathematische Schranke. Mit Topologie wird fuer die exakte Zahl massentragender Knoten reserviert. In dieser Serie meldet die Boost-Tabelle in allen neuen Varianten dieselbe interne Kapazitaetsstufe (491519 Buckets). Die kleinere angeforderte Reserve erzeugt hier deshalb keinen nachgewiesenen Kapazitaetsgewinn. Buckets sind keine Bytes. Fuer andere Teilchenzahlen ist eine neue Kapazitaetsmessung noetig.

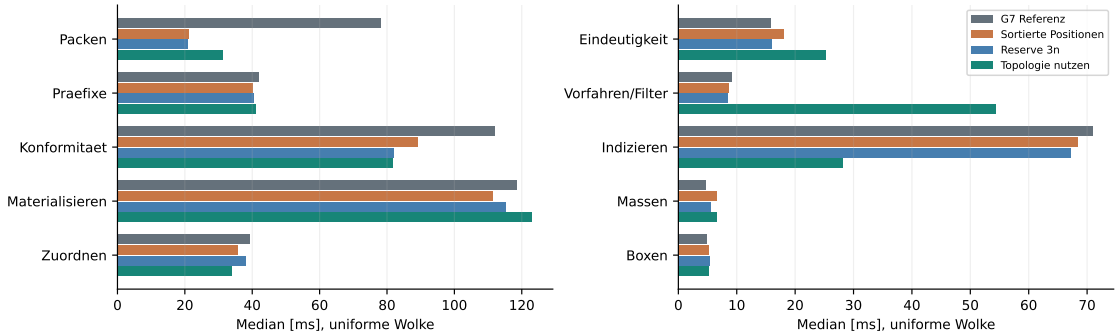
2.3 Massentragende Topologie aus dem CFK-Baum

Die Wurzel ist 1, die Kinder eines Codes k sind $2k$ und $2k+1$, der Elterncode ist $k \gg 1 = \lfloor k/2 \rfloor$. Sei T die vollstaendige sortierte Gitterknotenmenge und P die Menge belegter Blattcodes. Benoetigt wird

$$T_M = \{\lfloor p/2^j \rfloor : p \in P, \quad 0 \le j \le \lfloor \log_2 p \rfloor\}.$$

Der neue Weg sucht die belegten Blaetter binaer in T, markiert sie und propagiert die Markierungen in absteigender Codereihenfolge zu den Eltern. Elterncodes werden dabei ebenfalls monoton kleiner; ein rueckwaerts laufender Index genuegt zur Elternsuche. Anschliessendes Filtern behaelt genau die markierten Codes in ihrer urspruenglichen sortierten Reihenfolge. Mit $N = |T|$ betraegt dieser Schritt $O(N + n \log N)$ Zeit und N Bytes fuer temporaere Markierungen, zusaetzlich zum Ergebnisvektor. Die Markierungen werden vor dem Aufbau der Indextabelle freigegeben. Leere Gitterzellen gelangen nicht in den Kraftbaum. Der uebergebene Speicher wird nach der Konstruktion nicht referenziert. Geprueft werden Wurzel, strenge Sortierung, Geschwisterpaare, existierende Eltern sowie die Existenz und Blatteigenschaft aller Teilchencodes. Die geometrische Konformitaet bleibt Aufgabe des Gitterbauers. Saemtliche bisherigen Eingabe-, Massen-, ID-, Zuordnungs- und Ueberlaufpruefungen bleiben erhalten.

3 Phasen, Tests und Reproduzierbarkeit



Die Phase Vorfahren umfasst beim neuen Weg auch Topologievalidierung, Markierung und Filtern. Diese geaenderte Bedeutung ist bei Vergleichen wichtig. Das separate Auslesen und Sortieren der Kraftbaumcodes entfaellt, aber das bedeutet nicht, dass die gesamte Vorbereitung automatisch schneller wird. Die binaere Suche der Teilchencodes im vollstaendigen Gitter bleibt Aufwand.

3.1 Messplan

Drei eingefrorene Wolken (uniform, Gauss, Mischung), jeweils 100000 Teilchen; vier Stufen und fuenf Wiederholungen ergeben 60 Hauptlaeufe. Die Variantenfolge rotiert deterministisch und wird teilweise umgekehrt. Alle Laeufe erfolgen nacheinander ohne parallele Builds. Zwoelf Threads fuer Gitterzuordnung und Kraefte, vier fuer die Kraftbaumvorbereitung. Die hier veraenderten Sortier-/Filterverfahren selbst sind seriell. Es handelt sich nicht um eine neue Thread-Skalierungsserie. GCC 13.3, Boost 1.83, Release: `-O3 -DNDEBUG -ffp-contract=off`; kein Fast-Math. `OMP_DYNAMIC=FALSE`, keine feste Kernbindung oder Taktkontrolle. Classic-MAC mit Theta 0.5, G=1, ohne Glaettung, alle Ziele. Zusaetzlich drei Integer-MAC-Pruefungen mit je 128 archivierten Zielen. Die drei separat erzeugten Gitter stimmen in Codes, Teilchenzuordnung, Teilungen und Tiefenstatistik mit der Referenz ueberein.

3.2 Tests und Grenzen

Release-Tests fuer beide Reservierungsfaktoren, OpenMP-Tests mit begrenzter Threadzahl und ASan/UBSan mit und ohne OpenMP bestanden. Geprueft werden Tiefe 63, Randpunkte, subnormale Koordinaten, leere Baeume, manipulierte Topologie, doppelte Positionen, NaN/Unendlich und die Lebensdauer des uebergebenen Gitters. Tests vergleichen auch Knotenanordnung, Verknuepfung, Massen, Schwerpunkte und Boxen beider Aufbauwege. LeakSanitizer ist in dieser Umgebung deaktiviert; ein spezieller Race-Detector wurde nicht verwendet. Fuenf Wiederholungen beschreiben die lokale Laufzeitstreuung, nicht die Unsicherheit ueber beliebige Teilchenverteilungen. Ein Vorteil bei einer Million Teilchen ist durch diese Serie nicht belegt. Fuer Entwicklung reichen 100000 Teilchen; spaetere Skalierungsentscheidungen brauchen wenige groessere Kontrolllaeufe. Es wurden hier keine Millionenlaeufe gestartet.

4 Bedienung und weitere Schritte

Das Kommandozeilenprogramm nutzt bei selbst gebautem Gitter standardmaessig die Topologie. Mit `-no-reuse-topology` bleibt der eigenstaendige Kraftbaumaufbau erreichbar. Die Bibliothek kann wie bisher ohne Topologie aufgerufen werden; der dritte Konstruktorparameter ist optional. `CFK_INDEX_RESERVE_FACTOR` ist ein CMake-Parameter (1 bis 8).

```
python3 python/benchmark_pipeline.py --output experiments/g8_neu
python3 python/summarize_pipeline.py experiments/g8_neu
  --report ../Berichte/2026-09-17/CPP_G8_neu
```

Die letzten beiden Zeilen bilden einen Befehl. Das Benchmarkprogramm benoetigt die im Quelltext bezeichneten Vergleichsbinaries und Referenzdateien; ein neuer Ausgabeordner verhindert das Ueberschreiben des Archivs. Die Auswertung prueft zuerst dessen SHA-256-Manifest. Rohdaten, Eingaben, Programme, Buildflags, Testlogs und Quellsnapshot: CPP/experiments/cpp_g8_pipeline. Rueckkehrpunkt: Sicherungen/2026-09-17_vor_CPP_G8/Quellstand.tar.gz. API-Erlaeuterung: CPP/docs/pipeline.md. Messwerte: messwerte.csv im Berichtsordner.

4.1 Empfehlung

Die sortierbasierte Gitterpruefung beibehalten. Die kleinere Reserve ist korrekt, aber fuer diese Groesse kein eigener Speichergewinn. Den Topologiepfad als getestete Alternative erhalten, ohne ihn bereits als gesicherten Geschwindigkeitsgewinn zu bezeichnen. Bei knappen Laufzeitbudgets beide Wege fuer die tatsaechliche Eingabegroesse vergleichen. Als naechste gezielte Optimierung bietet sich an, die Teilchenblattcodes einmal zu sortieren und linear mit der vorhandenen Topologie abzugleichen, statt fuer jedes Blatt eine binaere Suche auszufuehren. Danach sind Materialisierung und Konformitaetsabschluss des Gitters die naechsten Profilkandidaten. Mehr Threads allein loesen deren Datenabhaengigkeiten nicht.