CFK: Aggressive Compileroptimierung

Eine Million Teilchen

17. September 2026

1 Fragestellung und Versuchsplan

CPP_G4, 17. September 2026. Welche Compileroptimierung beschleunigt die bestehende C++-Architektur, ohne fuer die Forschung wichtige Eigenschaften unbemerkt aufzugeben?

Eine feste uniforme Wolke mit 1000000 Teilchen im Einheitsquadrat, exakt dieselbe Eingabedatei wie CPP_G3. Gesamtmasse 1, G=1, keine Glaettung, klassisches Oeffnungskriterium mit theta=0.5. Alle 1000000 Beschleunigungen werden mit zwoelf OpenMP-Threads berechnet. Der Gitterbau bleibt seriell.

Vier Builds desselben numerischen Quellcodes, je drei getrennte vollstaendige Programmaufrufe. Die Reihenfolge wechselt zwischen den Runden; Kompilierung und Tests sind vor der Messserie abgeschlossen. Kein gleichzeitiger Benchmark. OMP_DYNAMIC=FALSE; keine zusaetzliche Threadbindung oder Kontrolle von Takt und Hintergrundlast.

Dies sind technische Wiederholungen einer Wolke, keine unabhaengigen Stichproben. Die Empfehlung gilt zunaechst fuer diese CPU und diesen statischen classic-Durchgang, nicht automatisch fuer andere Verteilungen, das Integer-Kriterium oder eine Zeitintegration.

2 Compilerprofile

GCC 13.3, C++20, Boost 1.83 und OpenMP auf Intel Core Ultra 7 255U. Alle Builds verwenden -O3 -DNDEBUG. LTO wird durch die CMake-Option CMAKE_INTERPROCEDURAL_OPTIMIZATION=ON aktiviert; die tatsaechlichen Compile- und Linkoptionen sind archiviert.

Profil	Zusaetzliche Optionen	FP-Kontraktion
Referenz	keine	off
Native + LTO	-march=native, LTO	off
+ FMA	-march=native, LTO	fast
+ Fast-Math	-march=native, LTO, -ffast-math	fast

3 Was diese Optionen bedeuten

Native erlaubt Instruktionen der lokalen CPU; der erzeugte Code ist nicht mehr fuer beliebige andere Prozessoren portabel. LTO optimiert ueber Quelldateigrenzen hinweg. Beide Optionen erlauben fuer sich keine beliebige Umordnung der Gleitkommasummen.

FMA kann a*b+c mit nur einer abschliessenden Rundung berechnen. Abweichungen von der bisherigen Referenz sind deshalb nicht automatisch schlechtere physikalische Genauigkeit. Die Referenz ist ein etablierter Barnes-Hut-Build, keine exakte Kraftloesung.

Fast-Math geht deutlich weiter: unter anderem Umordnung von Operationen und Annahmen ueber das Fehlen von NaN und Unendlich. Damit koennen vorhandene Fehlerpruefungen unwirksam werden. Deshalb messen wir es als Experiment, nicht als stillschweigend akzeptierte Produktionsoption. Kein pauschales -Ofast: der hier verwendete Schalter grenzt das Experiment auf Fast-Math ein.

4 Gesamtlaufzeit

Mediane aus drei Aufrufen, inklusive Einlesen, Gitterbau, Kraftbaum, Kraftrechnung und Ausgabe. Peak-RSS stammt von GNU time und bezieht sich nur auf den jeweiligen C++-Prozess.

Profil	Median [s]	Bereich [s]	Faktor	RSS [MiB]
Referenz	37.14	36.59-38.56	1.00	748.0
Native + LTO	37.00	36.87-38.47	1.00	747.9
+ FMA	38.18	37.65-38.91	0.97	747.6
+ Fast-Math	36.68	34.86-38.43	1.01	747.6

5 Zeitanteile

Kraftbaum umfasst Validierung, Aggregate und vorbereitete Geometrie. Die Mediane der einzelnen Phasen sind nicht additiv zum Median der Prozesszeit; Ausgabe und Zwischenarbeiten kommen hinzu.

Profil	Einlesen [s]	Gitter [s]	Kraftbaum [s]	Kraefte [s]
Referenz	0.344	16.996	11.099	7.615
Native + LTO	0.339	17.254	10.864	7.552
+ FMA	0.350	17.625	11.327	7.513
+ Fast-Math	0.346	17.413	10.536	7.242

6 Streuung der Kraftzeit

Auch der isolierte Kraftdurchgang beinhaltet OpenMP-Start und Ergebnisallokation. Es wird kein Threadzahl-Speedup gemessen, sondern derselbe Zwoelf-Thread-Durchgang mit anderen Compileroptionen.

Profil	Minimum [s]	Maximum [s]	Faktor der Mediane
Referenz	7.579	8.051	1.000
Native + LTO	7.407	7.658	1.008
+ FMA	7.359	7.564	1.014
+ Fast-Math	7.190	7.386	1.052

7 Ergebnisgleichheit und Tests

Fuer jeden Build wurden vollstaendige Knotencodes, zwingende Teilungen und Blattzuordnungen des Millionen-Gitters per SHA-256 mit CPP_G3 verglichen. Alle vier Gitter stimmen ueberein. Jede Kraftausgabe wird ausserhalb des C++-Programms mit NumPy auf endliche Werte und gegen die alte Referenz geprueft.

Referenzgleich bedeutet bitgleiche double-Komponenten zu CPP_G3. Wiederholbar bedeutet bitgleiche Ausgaben zwischen den drei Aufrufen desselben Builds. Die Zaehler erfassen besuchte, geoeffnete und akzeptierte Knoten. Tests bezeichnet beide vorhandenen nativen Testsuiten, einschliesslich Grenzfaellen.

Profil	Referenzgleich	Wiederholbar	Zaehler gleich	Tests bestanden
Referenz	ja	ja	ja	ja
Native + LTO	ja	ja	ja	ja
+ FMA	nein	ja	ja	ja
+ Fast-Math	nein	ja	ja	NEIN

8 Numerische Abweichungen

Verglichen wird mit CPP_G3, nicht mit einer neuen direkten O(n^2)-Rechnung. L2-relativ ist die euklidische Norm aller Differenzen geteilt durch die Norm aller Referenzbeschleunigungen. Pro Teilchen gilt r_i=||a_i-a_i(ref)||/||a_i(ref)||; bei Null im Nenner wird die kleinste positive normale double-Zahl verwendet. Angegeben sind das 99-Prozent-Quantil und Maximum dieser relativen Vektordifferenz. Je Profil wird der groesste Wert der drei Wiederholungen berichtet. Die globale L2-Norm kann von nahen Paaren mit grossen Kraeften dominiert werden; die teilchenweisen Kennzahlen ergaenzen sie deshalb. Relative Werte sind bei fast verschwindender Referenzbeschleunigung besonders empfindlich.

Alle Kraftausgaben erfuellen die bisherige komponentenweise Vergleichstoleranz rtol=5e-12, atol=1e-9.

Profil	Andere Komponenten	L2-relativ	r: Q99	r: Maximum
Referenz	0	0	0	0
Native + LTO	0	0	0	0
+ FMA	1859099	1.39e-16	1.97e-14	2.14e-12
+ Fast-Math	1878628	1.73e-16	1.97e-14	2.14e-12

9 Grenzfaelle und Aussagegrenzen

Nicht bestandene Testsuiten: + Fast-Math. Die vollstaendigen Fehler stehen in den jeweiligen tests.log-Dateien. Ein erfolgreicher Millionenfall ersetzt diese Grenzfalltests nicht.

Beim Fast-Math-Build melden die Krafttests drei fehlende Ausnahmen: NaN als Position, Ueberlauf der Gesamtmasse und Ueberlauf einer Kraft. Die Gittertests brechen ausserdem bei einem Grenzfall mit einer Koordinaten-/Duplikat-Ausnahme ab. Diese konkreten Regressionen sind ein Ausschlussgrund fuer den unveraenderten Produktionscode.

Auch ein reproduzierbarer aggressiver Build garantiert keine Bitgleichheit zwischen Compilern oder CPU-Typen. In einer spaeteren dynamischen Simulation koennen kleine Kraftabweichungen anwachsen. Eine solche Langzeituntersuchung wurde hier nicht gemacht.

Drei Wiederholungen erlauben eine erste lokale Empfehlung, keine gesicherte Rangfolge bei Unterschieden innerhalb der Laufzeitschwankungen. Native und LTO werden gemeinsam variiert; ihre individuellen Anteile wurden nicht isoliert.

10 Erste Empfehlung

Unter den hier getesteten referenzgleichen Profilen mit bestandenen Tests hat Native + LTO den kleinsten Median der Gesamtzeit. Dieses Profil ist die erste Alternative zur Referenz, wenn hohe Geschwindigkeit und die bisherigen numerischen Ergebnisse gleichzeitig erhalten bleiben sollen. Die Messbereiche muessen bei knappen Unterschieden mitberuecksichtigt werden.

Der Medianvorteil gegenueber der Referenz betraegt nur 0.4 Prozent. Daraus leite ich noch keinen belastbaren Gesamtvorteil ab und lasse den bisherigen Referenzbuild als Standard unveraendert. Native + LTO ohne Fast-Math und ohne FMA bleibt ein konservativer lokaler Optimierungskandidat.

Die Kraftphase belegt im Referenzlauf ungefaehr 21 Prozent der Gesamtzeit. Zehn Prozent weniger Kraftzeit sparen deshalb nur etwa 2.1 Prozent Gesamtzeit, solange der Aufbau unveraendert bleibt. Verbesserungen an der Blattzuordnung und den Speicherzugriffen sind deshalb ein wichtigerer naechster Ansatz als weitere pauschale Flags.

Den bisherigen portablen Referenzbuild und die archivierten Binaerdateien beibehalten. CPU-spezifische Builds separat kennzeichnen. FMA nur dann als eigenstaendiges numerisches Profil verwenden, wenn die gemessene Beschleunigung den Verzicht auf Referenz-Bitgleichheit rechtfertigt. Ein Profil mit fehlgeschlagenen Sicherheits- oder Geometrietests wird nicht fuer Produktionsrechnungen empfohlen.

11 Reproduktion und Quellen

Die Serie dauerte einschliesslich Bauen und Tests 714.0 Sekunden. 100 archivierte Dateien wurden vor dieser Auswertung verifiziert. Rohdaten: CPP/experiments/cpp_g4_compiler. Archiviert sind Eingabe, Quellen, Programme, Buildoptionen, CPU-Merkmale, Einzelmessungen und Ergebnispruefungen.

Erneute Ausfuehrung aus CFK/CPP mit neuem Ausgabeverzeichnis: python3 python/benchmark_compiler.py –output experiments/cpp_g4_neu. Auswertung: python3 python/summarize_compiler.py experiments/cpp_g4_neu.

Compilersemantik: GCC 13.3 Handbuch, Optimize Options und x86 Options; abgerufen am 17. September 2026. Die Beschreibung der Optionen beruht auf diesen Quellen, die Zeit- und Gleichheitsaussagen auf den lokalen Messungen.

`https://gcc.gnu.org/onlinedocs/gcc-13.3.0/gcc/Optimize-Options.html`

`https://gcc.gnu.org/onlinedocs/gcc-13.3.0/gcc/x86-Options.html`