# Falsche Rechenergebnisse durch OpenBLAS 0.3.20 in Containern auf den i02-Knoten (Sapphire Rapids)

Alexander Schlüter, TU Darmstadt — Lichtenberg-Cluster
5. September 2026

## Kurzfassung

Finite-Elemente-Rechnungen in einem Apptainer-Container (Ubuntu 22.04, DOLFINx/PETSc/MUMPS) scheitern auf den i02-Knoten reproduzierbar im ersten Lastschritt mit „numerisch singuläre Matrix", während dieselbe Software auf i01-Knoten seit Monaten fehlerfrei rechnet. Ursache ist nicht die Anwendung, sondern die BLAS-Bibliothek im Container: **OpenBLAS 0.3.20 (Ubuntu-22.04-Paket, `DYNAMIC_ARCH`) erkennt die Xeon Platinum 8470Q der i02-Knoten als „Cooperlake" und liefert mit diesem Kernel falsche Ergebnisse.** Schon ein einfaches Matrixprodukt weicht um etwa 22 vom Referenzwert ab (erwartet $10^{-12}$). Nichts stürzt ab — die Zahlen sind nur falsch. Der Workaround `OPENBLAS_CORETYPE=SkylakeX` behebt das vollständig; ein Produktivjob konvergiert damit auf Anhieb. Betroffen ist potenziell jeder Container auf i02, der über numpy, scipy, PETSc oder MUMPS auf diese OpenBLAS-Version zugreift — und viele FEniCS-/DOLFINx-/wissenschaftliche Python-Images basieren auf Ubuntu 22.04.

## Umgebung

| | |
|---|---|
| Knoten | `mpsd0001` (i02), Intel Xeon Platinum 8470Q, Sapphire Rapids; 8 × 13 Kerne, 490 GB |
| Container | Apptainer, Ubuntu 22.04: DOLFINx 0.7.3, PETSc 3.20 mit MUMPS, MPICH 4.1.2, numpy mit `libopenblas.so.0` (`openblas-pthread`) |
| OpenBLAS | 0.3.20, `DYNAMIC_ARCH`, `NO_AFFINITY`; gewählter Kernel laut `openblas_get_corename()`: **Cooperlake** |
| Vergleich i01 | `mpsc*`, ältere Xeon-Generation (4 × 24 Kerne, 365 GB): dieselbe Software rechnet korrekt |

## Beispiel 1 — BLAS-Selbsttest (nur numpy, kein FE, kein MPI)

Ein 600×600-Matrixprodukt wird einmal über BLAS (`A @ B`, dgemm) und einmal ohne BLAS (`numpy.einsum`) berechnet; zusätzlich das Residuum eines linearen Gleichungssystems (LAPACK dgesv). Der vollständige Test (`blas_check.py`) liegt im beigefügten Archiv; Kern:

```
rng = np.random.default_rng(0)
A = rng.random((600, 600)); B = rng.random((600, 600))
d_gemm = np.abs(A @ B - np.einsum("ij,jk->ik", A, B)).max()   # erwartet ~1e-12
x = rng.random(600); b = A @ x
d_gesv = np.abs(A @ np.linalg.solve(A, b) - b).max()          # erwartet ~1e-10
```

Gemessen am 05.09.2026 auf `mpsd0001` (SLURM-Job 54482116, 1 Kern):

| `OPENBLAS_CORETYPE` | gewählter Kernel | dgemm-Abweichung | dgesv-Residuum | Bewertung |
|---|---|---|---|---|
| nicht gesetzt | **Cooperlake** | **$2{,}19 \cdot 10^{1}$** | **$5{,}52 \cdot 10^{4}$** | falsch |
| `SkylakeX` | SkylakeX | $5{,}4 \cdot 10^{-13}$ | $2{,}6 \cdot 10^{-13}$ | korrekt |
| `Haswell` | Haswell | $5{,}1 \cdot 10^{-13}$ | $2{,}8 \cdot 10^{-13}$ | korrekt |

Die Matrixeinträge liegen bei etwa 150 (Summen von 600 Produkten aus [0, 1]); eine Abweichung von 22 bedeutet, dass einzelne Einträge schlicht falsch sind.

## Beispiel 2 — Poisson-Problem mit MUMPS

Dasselbe Bild eine Ebene höher: ein Standard-Poisson-Problem (Einheitswürfel, 117 649

Freiheitsgrade) im Container mit DOLFINx gelöst, direkte Faktorisierung.

| `OPENBLAS_CORETYPE` | **MUMPS (LU)** | **PETSc-eigene LU** |
|---|---|---|
| nicht gesetzt | Lösung enthält **NaN** | falsche Lösung |
| `SkylakeX` | konsistent mit PETSc-LU | konsistent mit MUMPS |

## Beispiel 3 — Produktivjob vorher/nachher

Identischer FE-Job (elasto-plastische Homogenisierung, 2,8 Mio. Freiheitsgrade, 32 MPI-Ränge, ein i02-Knoten). Einziger Unterschied ist die Umgebungsvariable.

| **SLURM-Job** | `OPENBLAS_CORETYPE` | **erster Lastschritt** |
|---|---|---|
| 54482042 | nicht gesetzt | 24 Versuche, 24 × keine Konvergenz, MUMPS `INFO(1) = -10` (numerisch singulär), Abbruch bei $\Delta t < 10^{-11}$ |
| 54482126 | `SkylakeX` | konvergiert nach 1 Newton-Iteration; 3 Schritte in 5 Minuten, 0 verworfen |

Nach dem Setzen der Variable in den Jobskripten laufen inzwischen 20 Produktivjobs parallel auf i02-Knoten, alle mit konvergierenden Schritten. Vorher waren 49 Jobs identisch gescheitert; weil das Fehlerbild wie ein Modell- oder Solverproblem aussieht (Konvergenz, Singularität), lief die Fehlersuche zunächst zwei Tage in Richtung MPI, JIT-Cache und Netztopologie.

## Einordnung

OpenBLAS erhielt ein eigenes Sapphire-Rapids-Target erst mit 0.3.21; die davor liegenden Versionen wählen per `DYNAMIC_ARCH` den Cooperlake-Kernel, dessen AVX-512-Pfade auf dieser CPU falsche Ergebnisse liefern. Ubuntu 22.04 (Jammy) liefert genau 0.3.20 aus und ist Basis vieler wissenschaftlicher Container. Fehlerbilder in Anwendungen sind uneinheitlich — falsche Werte, NaN, „singuläre Matrix", ausbleibende Konvergenz — und werden leicht dem eigenen Code zugeschrieben. Auf i01 tritt der Fehler nicht auf, weil dort die korrekte Kernel-Familie gewählt wird.

## Workaround und Empfehlung

Sofort wirksam, ohne Änderung am Container: `export OPENBLAS_CORETYPE=SkylakeX` vor dem Container-Aufruf, bei Apptainer zusätzlich `APPTAINERENV_OPENBLAS_CORETYPE=SkylakeX`. Auf i01 ist die Variable unschädlich. Dauerhaft: Container mit OpenBLAS ≥ 0.3.24 (z. B. Ubuntu 24.04) oder numpy-Wheels mit eigenem OpenBLAS. Für den Cluster wäre ein Hinweis in der Dokumentation zu i02 und Containern hilfreich, gegebenenfalls ein Default für `OPENBLAS_CORETYPE` im Job-Environment der i02-Knoten.

## Nachrechnen in zwei Minuten

Das beigefügte Archiv `openblas_i02_check.zip` enthält `blas_check.py`, ein SLURM-Skript, das beide Varianten auf einem i02-Knoten laufen lässt, und die erwartete Ausgabe. Es genügt ein beliebiges Ubuntu-22.04-Image mit numpy, etwa:

```
unzip openblas_i02_check.zip && cd openblas_i02_check
apptainer pull docker://dolfinx/dolfinx:v0.7.3
sbatch run_check.sbatch dolfinx_v0.7.3.sif      # i02, 1 Kern, < 1 min
cat blas-check-*.out
```

Ohne Argument prüft `run_check.sbatch` das System-Python des Knotens; `sbatch -C i01 run_check.sbatch <sif>` liefert die Gegenprobe auf i01.