5. Übungsblatt

Ausgabe: 12. Juni 2014 · Besprechung: 26. Juni 2014

Parallele Hashjoins

Mit dem Aufgabenblatt 3 haben wir uns die Implementation von Hashjoins auf modernen Architekturen angeschaut. Jetzt wollen wir außerdem noch die Parallelität moderner Multi-Core-Systeme ausnutzen.

Ähnlich wie auf Folie 74 dargestellt, kann man das erreichen, indem man beide Eingaberelationen anfangs aufteilt auf die einzelnen Kerne des Systems. Anders als auf Folie 74 wird die Parallelität allerdings zunächst genutzt, um die Eingaberelationen parallel gemäß ihrer Schlüsselwerte zu partitionieren (gegebenenfalls mehrstufig, wie auf Folie 79 dargestellt).

→ **Welche Annahmen ändern sich dabei im Vergleich zu Folie 74? Wie könnte man vorgehen, um die parallele Verarbeitung zu koordinieren?**

Nach der (parallelen) Partitionierung ergeben sich "von alleine" einzelne Teilaufgaben ($r_i \bowtie s_i$). Deren Abarbeitung kann zum Beispiel durch eine *task queue* zwischen den Kernen/Threads koordiniert werden.

Aufgabe 1: Paralleles Partitionieren

Implementieren Sie eine geeignete Technik, um (zwei) gegebene Eingaberelationen gemäß einem Hashwert parallel zu partitionieren.

Aufgabe 2: Parallele Joinverarbeitung durch *Task Queue*

Implementieren Sie einen *task queue*-Mechanismus, um die einzelnen Teil-Joins $r_i \bowtie s_i$ parallel auszuwerten.

Aufgabe 3: Optimierung Ihrer Hashjoin-Implementation

In der Vorlesung haben Sie zwischenzeitlich außerdem weitere Features moderner Hardware kennen gelernt, die man u. U. zur Beschleunigung verwenden kann, z. B.

- Vektorisierung bzw. das Ausnutzen von SIMD-Operationen
- *Prefetching*, d. h. das spekulative Voraus-Laden von Speicherinhalten zum "Verstecken" von Speicherlatenzen.

Diskutieren Sie, ob solche Techniken auch geeignet sind, die Auswertung Ihrer Hashjoin-Implementation zu beschleunigen.

Hinweis

In verschiedenen Arbeiten wurden unlängst Implementationen vorgestellt, die einen Join-Durchsatz von teilweise mehreren hundert Millionen Tupeln pro Sekunde erreichen. Die Arbeiten beschreiben auch einige der "Tricks", mit denen man In-Memory-Joins noch weiter optimieren kann.

[1] Changkyu Kim *et al.* Sort vs. Hash Revisited: Fast Join Implementation on Modern Multi-Core CPUs. *PVLDB*, 2(2):1378–1389, 2009.

[2] Cagri Balkesen *et al.* Main-Memory Hash Joins on Multi-Core CPUs: Tuning to the Underlying Hardware. In *Proc. of the 29th Int'l Conference on Data Engineering (ICDE)*, Brisbane, Australia, 2013.