Exercise Session 11

Data Processing on Modern Hardware
263-3502-00L — Fall Semester 2012

Cagri Balkesen
cagri.balkesen@inf.ethz.ch

Department of Computer Science ETH Zurich, Switzerland

6 December 2012

Today's Program

- Discussion: parallel partitioned hash join
- Optimizing the serial version
- Parallelizing with Pthreads
  - First partitioning phase
  - Second partitioning phase & hash join
- Nested-loops join with SIMD

Evaluation Platform

- Experiments on dual-socket Xeon
  - 2.26 Ghz
  - 2 × Quadcore Intel Xeon L5520
  - Hyper-threading = yes → 16 HW threads
  - Microarchitecture: Nehalem (Gainestown)
- Handout code
  - 1st partitioning phase: 2.827 sec
  - 2nd partitioning phase: 3.052 sec
  - 3rd partitioning phase: 2.777 sec
  - Join phase: 1.791 sec
  - Total execution time: **10.601 sec**

Optimized Serial Version

- First partitioning phase 8 $\rightarrow$ 6 radix bits
- Second partitioning phase 8 $\rightarrow$ 6 radix bits
- Then hash join using 14 radix bits (total 26 radix bits)
- Avg tuples per bucket $= 100,000,000/2^{26} \approx 1.5$
- 100,000,000*4(bytes)*2(relations)/64/64
  $\rightarrow$ (avg) 195.3 KiB $<$ 256 KiB (L2)
- Total execution time: **9.423 sec**

Hash Join Function

```
bucket = calloc(NUM_BUCKETS_2, sizeof(uint32_t));
next = malloc(small_rel->num_tuples * sizeof(*next));

// create hash table for smaller relation
for(i = 0; i < (small_end-small_start); i++) {
  key = HASH3(small_rel->tuples[i+small_start].id);
  next[i] = bucket[key];
  bucket[key] = i+1;
}

// perform hash join
for(i = large_start; i < large_end; i++) {
  key = HASH3(large_rel->tuples[i].id);
  for(hit = bucket[key]; hit > 0; hit = next[hit-1]) {
    if(large_rel->tuples[i].id == small_rel->tuples[small_start+hit-1].id) {
      // copy tuples to result buffer -> we skip it for this assignment
      result++;
    }
  }
}
```

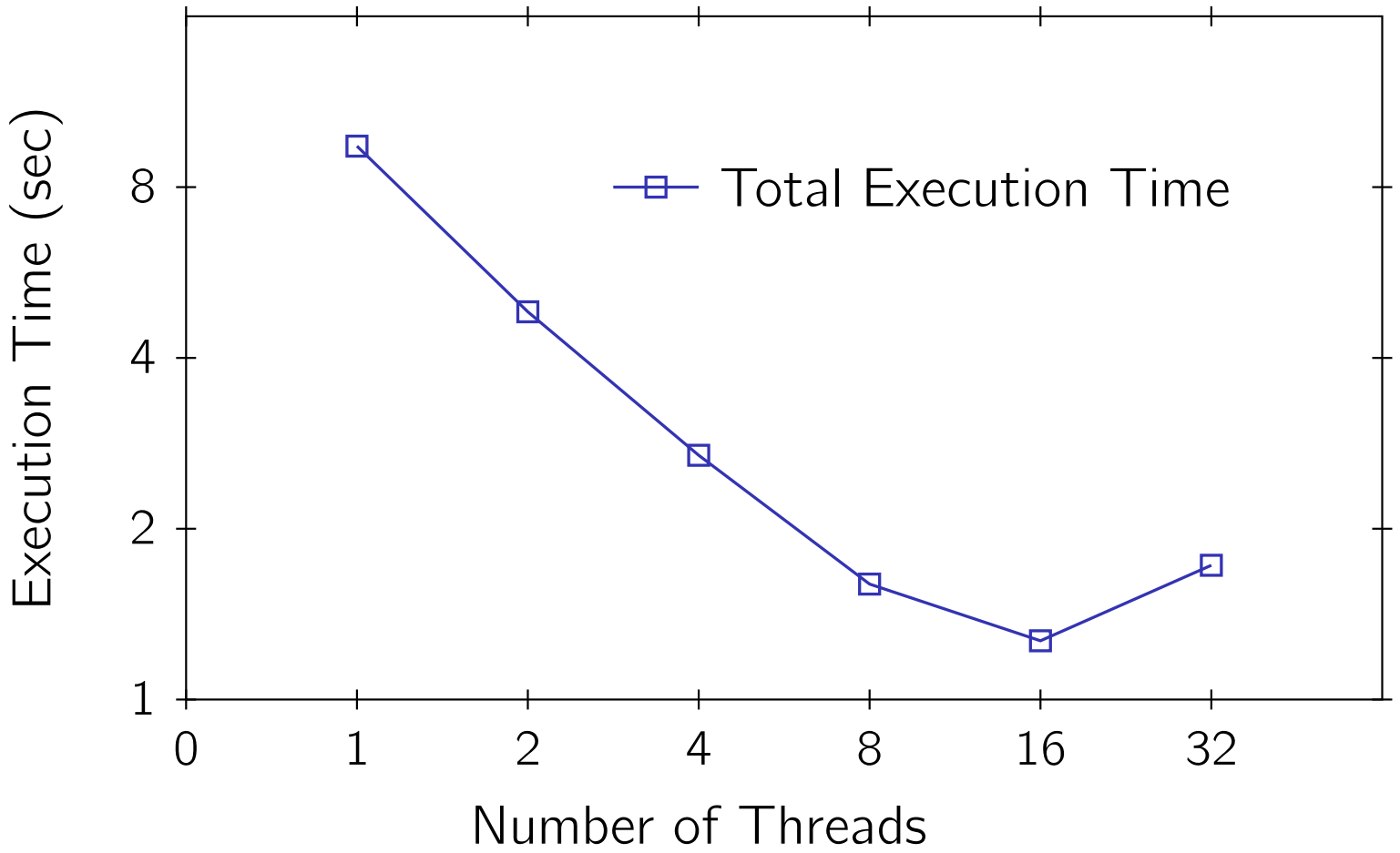
First Partitioning Phase (16 Threads)

1. Use $\tau = 4 \times \mathit{NUMTHREADS}$
2. Compute local historgram for each task (∼0.035 sec/task)
3. Synchronize
4. Merge histograms in each thread (∼0.000010 sec/thread)
5. Compute global prefix sum in each thread (∼0.000002 sec/thread)
6. Compute local prefix sum for each task (∼0.000006 sec/task)
7. Reorder tuples according to local prefix sum of each task (∼0.06 sec/task)
8. Synchronize
9. Total time: ∼**0.365 sec** (16 threads)

Second Partitioning Phase & Hash Join

- 6 radix bits → 64 partitions → 64 tasks (4 tasks/thread)
- Second partitioning phase: (avg) 0.076 sec/task
  → **0.306 sec/thread**
- Join phase: (avg) 0.102 sec/task
  → **0.408 sec/thread**
- Total execution time (16 threads): **1.225 sec**
  (2.26Ghz versus 3.2 GHz [Paper])

Varying the Number of Threads



Nested-Loops Join

- Hash Join: SIMD does not help
  → no efficient scatter/gather SIMD operations
- Let us look at nested-loops instead (R = S = 100,000 tuples)

```
for (i = 0; i < R->num_tuples; i++) {
  for (j = 0; j < S->num_tuples; j++) {
    if(R->tuples[i].id == S->tuples[j].id) {
      matches++;
    }
  }
}
```

- Execution time: **12.732 sec**

SIMD Version of Nested-Loops Join

```
for(i=4; i<=R->num_tuples; i+=4) {

  registerR0 = _mm_set_epi32(R->tuples[i-1].id, R->tuples[i-2].id, ...);
  registerR1 = _mm_shuffle_epi32(registerR0, _MM_SHUFFLE(0,3,2,1));
  registerR2 = _mm_shuffle_epi32(registerR1, _MM_SHUFFLE(0,3,2,1));
  registerR3 = _mm_shuffle_epi32(registerR2, _MM_SHUFFLE(0,3,2,1));

  for (j=4; j<=S->num_tuples; j+=4) {
    registerS0 = _mm_set_epi32(S->tuples[j-1].id, S->tuples[j-2].id, ...);

    accumulator = _mm_add_epi32(accumulator, _mm_cmpeq_epi32(registerR0, registerS0));
    accumulator = _mm_add_epi32(accumulator, _mm_cmpeq_epi32(registerR1, registerS0));
    accumulator = _mm_add_epi32(accumulator, _mm_cmpeq_epi32(registerR2, registerS0));
    accumulator = _mm_add_epi32(accumulator, _mm_cmpeq_epi32(registerR3, registerS0));
  }

  // handle end of relation S
}
// handle end of relation R
```

- Execution time: **2.043 sec**
- Speedup: **6.23**