Errata for "Building Efficient Query Engines in a High-Level Language" (PVLDB 7(10):853-864)

Yannis Klonatos [1], Christoph Koch [1], Tiark Rompf [1,2], and Hassan Chafi [2]

[1] École Polytechnique Fédérale de Lausanne (EPFL) {firstname}.{lastname}@epfl.ch
[2] Oracle Labs {firstname}.{lastname}@oracle.com

This is in response to recent feedback from our peers that calls for a number of clarifications regarding the experimental section of our paper.

Table 1 clarifies which optimizations are used in each evaluated flavor of LegoBase. *HyPer-simulated is a configuration of the LegoBase codebase* that mirrors the HyPer system as closely as possible by just activating some of the main optimizations and deactivating others: using the push engine, with operator inlining active but data structure optimizations and data layout transformations turned off. In addition, as noted in our paper (footnote 8), the actual HyPer system uses query plans generated by its own query optimizer while HyPer-simulated uses query plans from DBX.

We use TPC-H queries and generated data *at scaling factor 8* to evaluate the impact of our compilation techniques.

For all evaluated systems, reported query evaluation times *only include the execution time of the query* and exclude the time taken for query optimization/compilation and loading the data into the main-memory data structures. We note that the data structure specialization optimizations of Section 3 of our paper (whenever used) are a *gentle form of pre-computation*, as would arguably be loading row data from disk into a main-memory column store.

System	Data Struct. Opt. (Sect. 3.2)	Change Data Layout (Sect. 3.3)	Operator Inlining (Sect. 2.1)	Push Engine Opt. (Sect. 3.1.1)
Volcano-Style (LLVM)	×	×	×	×
Volcano-Style (GCC)	×	×	×	×
HyPer-simulated	×	×	✓	✓
LegoBase (Scala)	✓	✓	✓	✓
LegoBase (C)	✓	✓	✓	✓

Table 1: Optimizations used in each evaluated system.

Since the current prototype of LegoBase only supports the generation of single-threaded code, *all systems are forced into single-threaded execution*.

The HyPer version used in the camera ready version of our paper is v0.4-226, which was provided to us by the HyPer developers in March 2014. This version was run single-threaded by disabling all CPU cores but one in the configuration of the Linux kernel. Table 2 also presents performance results from the latest HyPer version available (v0.4-452) as of July 2014. We note that for some queries (e.g., Q2, Q5, Q9, and Q16), there is a significant improvement from the old to the new HyPer binary. We observed that the HyPer query optimizer changed between the two binaries, and that the new version generates more efficient query plans. In addition, the developers of HyPer informed us of a command line parameter that allows HyPer to run single-threaded (PARALLEL=off), which was not used in the results of the camera ready version of our paper (as described above). By specifying this parameter, we observed that the performance of the new binary was slightly improved.

For completeness, Table 2 presents the original *absolute performance results* for all evaluated systems in the paper (which only presents relative numbers in the form of speedups), plus the results for HyPer v0.4-452 (with the setting PARALLEL=off) run in the same experimental environment.

The sentence *"LegoBase [...] can get an additional 5.3× speedup, for a total average 7.7× performance improvement"* in the 4th paragraph of Section 4.1 should be read as follows: a) LegoBase performs 7.7× better than DBX on average across all TPC-H queries and b) LegoBase achieves a 5.3× additional speedup compared to the speedup that HyPer achieves over DBX (the latter being 2.44×). LegoBase achieves an average speedup of 3.8× over HyPer (v0.4-226) across the TPC-H queries.

System	Q1	Q2	Q3	Q4	Q5	Q6	Q7
DBX	1790	396	1528	960	19879	882	969
Volcano-Style (LLVM)	1366	2865	2850	3639	210553	653	9735
Volcano-Style (GCC)	1700	3175	3417	3416	228936	578	9245
HyPer (v0.4-226)	861	227	1229	811	2542	302	860
HyPer (v0.4-452)	779	43	892	622	338	198	798
HyPer-simulated	540	244	1058	1018	40987	287	724
LegoBase (Scala)	202	140	323	460	3592	180	413
LegoBase (C)	168	108	195	283	2220	100	209

System	Q8	Q9	Q10	Q11	Q12	Q13	Q14
DBX	2172	3346	985	461	881	13593	823
Volcano-Style (LLVM)	24752	38729	4598	823	3288	24161	784
Volcano-Style (GCC)	24511	36658	4280	701	3665	25159	628
HyPer (v0.4-226)	518	2747	943	184	782	5101	260
HyPer (v0.4-452)	493	2139	565	102	485	2333	197
HyPer-simulated	1581	4102	967	210	1026	4623	252
LegoBase (Scala)	760	1303	1012	218	649	1748	389
LegoBase (C)	462	447	488	105	281	604	188

System	Q15	Q16	Q17	Q18	Q19	Q20	Q21	Q22
DBX	578	12793	1224	4535	6432	744	1977	447
Volcano-Style (LLVM)	928	18354	2196	11363	5655	5293	19368	1453
Volcano-Style (GCC)	787	16463	2768	12978	6107	5251	22253	1659
HyPer (v0.4-226)	415	1678	693	2935	1630	658	1809	420
HyPer (v0.4-452)	229	590	490	3682	1421	277	1321	212
HyPer-simulated	255	2906	635	1052	952	709	2019	525
LegoBase (Scala)	426	4601	270	1152	2862	1710	3759	827
LegoBase (C)	134	1326	75	245	371	495	669	191

Table 2: Absolute performance results (in milliseconds).

Proceedings of the VLDB Endowment, Vol. 7, No. 13