?
?
?
?
?
?
?
?
?
?
?
?
?
?
?
?
?
?
?
≈
5*million KV operations per second (op/s) per core) and low latency (microsecond-scale) by reducing networking overhead.*
?
?
?
?
??
≈
110*ns. A CPU core can issue several memory access instructions concurrently when they fall in the instruction window, limite*
store units in a core (measured to be 3 ≈
4 *in our CPU) [?, ?, ?]. In our CPU, we measure a max throughput of* 29.3*M random* 64*B accesses per second per core. On the other h*
byte KV pair typically requires ≈
100*ns computation or* ≈
500 *instructions, which is too large to fit in the instruction window (measured to be* 100 ≈
200). *When interleaved with computation, the performance a CPU core degrades to only* 5.5*MOps. An optimization is to batch r*
core throughput to 7.9*MOps in our CPU, which is still far less than the throughput of the host DRAM.*
≈
90 *nanoseconds on our platform (measured with [?], hardware details in sec.??). For* 64-
byte access granularity, the random read latency including data copy is ≈
110*ns. This latency can be partially hidden by the out-*
of-
order execution engine in modern CPUs, which can issue a few memory accesses in parallel. However, the parallelism is limite
the number of load-
store units (LSUs) per core k (measured to be 3 ≈
4 *on our CPU) and the instruction window size W (measured to be* 100 ≈
200 *instructions on our CPU) [?, ?, ?]. When there are enough memory operations in the instruction window, k of them can be iss*
??
≈
100*ns computation or* ≈
500 *instructions, which is far larger than the instruction window size. Consequently, when the random memory access instru*
?
?
??

$$\frac{1}{RandAccessThroughput} = \frac{MemLatency}{Parallelism} \tag{1}$$

$$\frac{1}{KVopThroughput} = ComputationTime + \frac{MemLatency}{min(BatchSize, Parallelism)} \tag{2}$$

≈
10*Mop/s*), 10 ≈
20*x slower than a single DDR channel.*
?
?
?
?
?
?
?
?
?
?
?
two-
sided
??
??
??
one-
sided
??
≈
150*Mop/s* [?]) *provided by RDMA NICs, it is challenging to find an efficient match between RDMA primitives and key-*
value operations. For a write (PUT or atomic) operation, multiple network round-
trips and memory accesses may be required to handle hash conflicts, memory allocation and fetch/save non-
inline data. RDMA does not support transactions. Clients must synchronize with each other to ensure consistency by either usi
based KVS, , Pilaf [?], FaRM [?] and HERD [?] recommend using one-
sided RDMA for GET operations only. For PUT operations, they fall back to two-
sided RDMA as RPC and let the remote CPU do the actual work. Throughput of write-
intensive workload is still bottlenecked by CPU cores.
≈
2*Mop/s* [?].
?
?
?
?
?
?
?