Whitepaper

Using IBM with OmniSci GPU-accelerated Analytics

Benchmark Testing for IBM Power System AC922 Cognitive Infrastructure for Enterprise AI with NVLink

Table of Contents

Executive Summary

Graphics processing units (GPUs) can make Big Data analytics 100x faster than with systems based on Central Processing Units (CPUs). While the latest CPU can have up to 32 cores, today's GPU has more than 3,500. OmniSci, Inc. harnesses the parallel processing power of GPUs to deliver the world's fastest data queries and visualization. However, most analytic workflows still move data between CPUs and GPUs, and so much of the speedup provided by a GPU can be lost during transfers to a CPU. NVIDIA created NVIDIA® NVLink™ to reduce that bottleneck, and it is only available for CPU-to-GPU transfers on IBM POWER systems. This white paper provides OmniSci performance benchmark results on the IBM Power System AC922 Cognitive Infrastructure for Enterprise AI and shows how NVlink speeds data science workloads shared across NVIDIA® Tesla® V100 GPUs and IBM POWER9 CPUs. This paper is ideal for those looking to add general-purpose computing with GPUs to their existing analytic architectures that include CPUs.

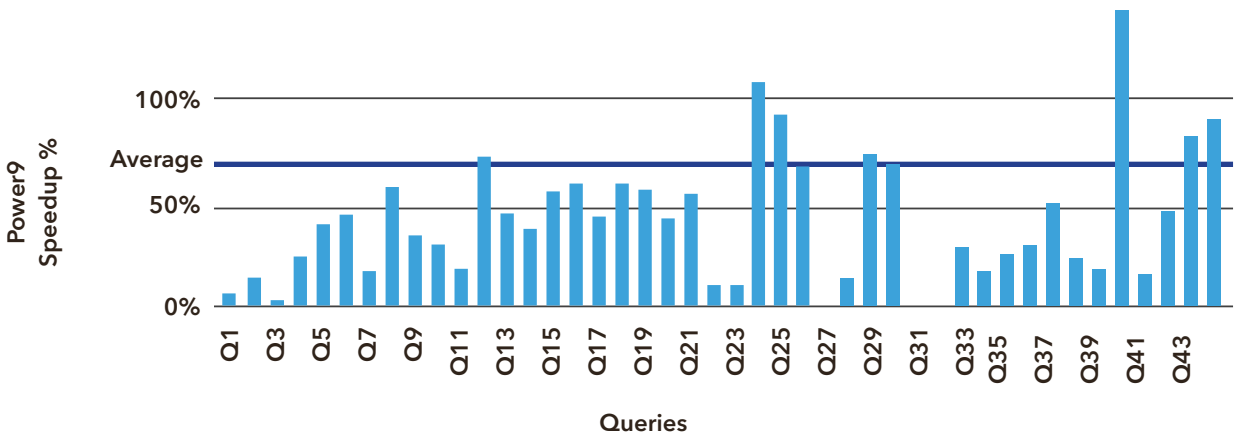
Benchmark Results

We measured an **average speedup of 72% across the 44 SQL queries** representative of the OmniSci software, with speedups up to 198% for Q40. Much of the speed is due to the use of NVIDIA NVLink on IBM Power System AC922, versus the slower GPU-to-CPU data transfers on the traditional PCIe bus used on x86 system.

Speedups on first load and first query are important for these common scenarios:

- Streaming large dataset bigger than GPU RAM capacity - requires copying over older datasets on the GPU. New datasets will be available sooner with faster first load times.
- Multiple large datasets from different sources & columns queried at the same time - e.g., many charts in a dashboard querying different columns, where speedups on first query can speed up the visual analytics.

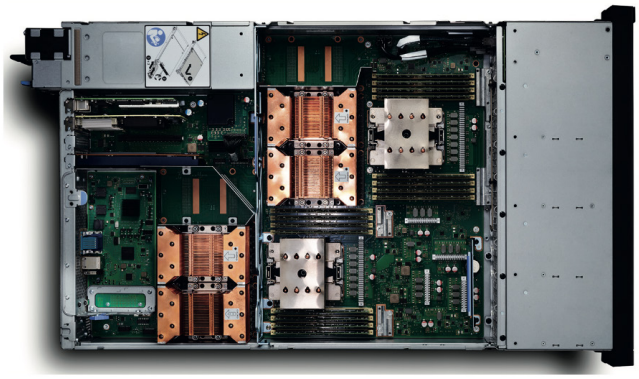
The following chart shows the speedup percentage (%) of benchmark results for "first-run" queries on the IBM Power System AC922 versus an x86 Power9 Speedup.



About IBM Power Systems AC922: The Fastest GPUs and CPUs with the NVLink Interconnect

Modern AI, HPC and analytics workloads are driving an ever-growing set of data intensive challenges that can only be met with accelerated infrastructure. To help meet these demands, IBM Power Systems has designed AC922 for the AI Era, the best server for Enterprise AI. The AC922 leverages IBM's new POWER9™ processor with a myriad of modern connectivity capabilities yielding up to 5.6x the data movement over the antiquated PCIe Gen 3 buses found in x86. IBM Power Systems deliver the only architecture enabling NVLink between CPUs and GPUs, unlocking new potential for accelerated computing. As the speed and size of those analytics increased, the load exposed a clear latency choke point: bandwidth for CPU-to-GPU data transfers. NVIDIA's NVLink breaks that bottleneck.

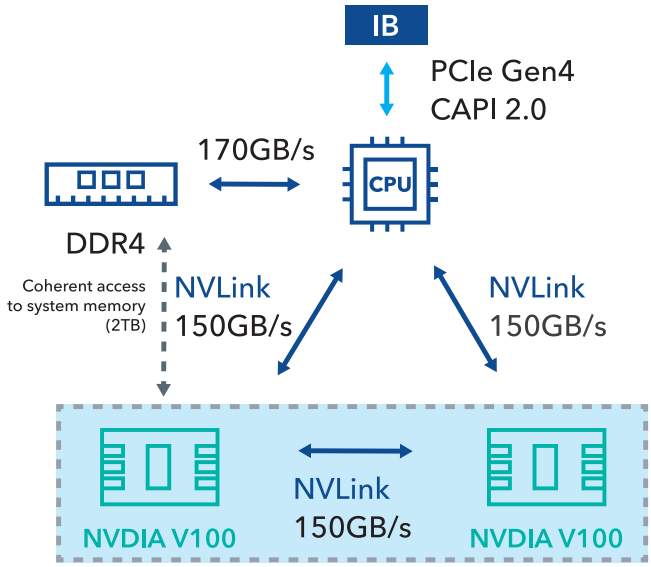
Much of that analytic acceleration comes from a new chip: the POWER9® processor with NVIDIA® NVLink™. The IBM POWER9 is a state of the art CPU in its own right, but because it comes with NVLink, it works together seamlessly with the NVIDIA® Tesla® V100 GPU. NVLink accelerates CPU-to-GPU analytic pipelines up to 5.6 times compared to systems without NVLink, and that gives businesses far faster insight.



Power Systems AC922 internal view

IBM Power System AC922 for Cognitive Infrastructure for Enterprise AI is the first to provide the NVLink Interconnect between CPU and GPU, and it delivers up to 5.6x the data movement of PCIe Gen 3.

In fact, this is the only configuration on the market that combines NVLink, Power9 processors and Tesla GPU processors.The following image shows the power of that relationship. NVIDIA's NVLINK technology transfers data between CPUs and GPUs at 100-150 gigabytes per second.



Source: IBM Power Advanced Compute (AC) AC922 Server Datasheet

OmniSci with IBM Power System AC922 Cognitive Infrastructure for Enterprise AI: A Powerful Coupling for Interactive Analytics

IBM Power System AC922 Cognitive Infrastructure for Enterprise AI delivers big data processing power at truly interactive speeds, over a hybrid GPU-CPU system. Because OmniSci makes big data analytics far faster on both GPUs and CPUs, OmniSci running on those systems delivers the fastest analytic capabilities and it abstracts the underlying hardware that makes that possible. As far as the analyst or data scientist is concerned, it's just very fast and easy to use.

OmniSciDB is open source, designed from the ground up to leverage the massive parallelism of GPUs. By efficiently combining the parallel power of multiple GPUs per server, OmniSciDB can execute queries over billions of records in milliseconds. That power and speed can scale with distributed computing across multiple high availability (HA) servers. Since OmniSciDB was open sourced in May 2017, partners like IBM and NVIDIA have rallied behind OmniSciDB as a standard for GPU analytics.

OmniSci Immerse is the GPU-powered visualization client, designed from the beginning to take advantage of the lightning-fast SQL and in situ rendering capabilities of OmniSciDB.

Every time a data scientist interacts with the OmniSci Immerse dashboard, it generates a set of SQL queries to OmniSciDB and the dashboard refreshes in milliseconds—even with tables containing billions of rows. Its speed comes from the fact that a GPU node contains more than a thousand times more processing cores than a CPU node, and those cores can process data in parallel.

In addition, OmniSci Immerse can leverage the native rendering engine built into OmniSciDB. For example, instead of having to send a billion points from server-to-client to render a pointmap, OmniSci Immerse can request a rendered PNG from the server. Unlike typical business intelligence systems, OmniSci Immerse lets the analyst instantly generalize massive datasets at whichever level of detail is useful, without down-sampling or aggregating any individual datapoint.

That GPU horsepower makes OmniSci Immerse a far superior window through which data scientists or business analysts can work with their data. They can explore data in its raw form. Any time an analyst changes one data parameter, all other charts, graphs and maps cross-filter to redraw the entire visualization dashboard in that new context. As the analyst becomes more familiar with the data, she can easily update her dashboard and add new visual objects. Compare that to speed of iteration to the status quo on a CPU system. It might take minutes or hours to script and run each query on a billion-row dataset and then more time to visualize and interpret the results.

IBM Power System AC922 Cognitive Infrastructure for Enterprise AI servers give enterprises the type of speed at scale that data scientists and business leaders need to stay ahead of new business challenges. These analytics require both the scale of big data and sub-second query speed, for use cases like:

- Telecommunications Network Reliability Analysis
- Vehicle Telematics Analysis
- Investment Banking Alternative Data Insights
- Utility Smart Meter Analysis
- Oil & Gas Well Log Analysis
- Pharmaceutical Clinical Trial Analysis
- Cyber Incident Investigation
- Defense and Intelligence GEOINT

OmniSci Benchmark Testing of IBM Power System AC922 Cognitive Infrastructure for Enterprise AI Demonstrates the Power of NVLink

The OmniSci collaboration around IBM Power System AC922 Cognitive Infrastructure for Enterprise AI demonstrates the superior performance and real-world impact of coupling two of the highest-performing processors: the Tesla V100 and POWER9.

In June 2019, OmniSci ran preliminary benchmark testing to quantify the acceleration of SQL queries on Power processors, which have the advantage of NVLink between both CPU-GPU and GPU-GPU, versus X86 CPUs, which have NVLink between the GPU-GPU but not CPU-GPU.

Benchmark System and Query Specifications

The following table shows the specifications of the two systems tested.

Test system and configurations with NVLink ("AC922"):

- IBM Power System AC922 Cognitive Infrastructure for Enterprise AI
- GPUs: 4 x NVIDIA Tesla V100 SXM2
- CPUs: IBM Power9, 2 x 16 cores
- Memory: 256GB

Test system and configurations without NVLink ("X86"):

- Intel(R) Xeon(R) CPU @ 2.30GHz
- GPUs: 4 x NVIDIA Tesla V100 SXM2
- CPUs: Intel X86, 48 cores
- Memory: 312GB

Benchmark Queries & Results in Detail

The following tables show the query text and the benchmark results for "first-run" queries on the IBM Power System AC922 versus an x86 system.

Benchmark Queries

The benchmark test ran 44 queries that test various aspects of common SQL queries, including filtering, filtering by different data types, group bys, different cardinalities in the data, different calculations/aggregations (e.g., calculating the average), extracting data (e.g., extracting the year from a timestamp), grouping by that extracted data, etc.

Query	Query Text
Q1	select count(*) from ##TAB##
Q2	select carrier_name, count(*) from ##TAB## group by carrier_name
Q3	select carrier_name, avg(arrdelay) from ##TAB## group by carrier_name
Q4	select origin_name, dest_name, avg(arrdelay) from ##TAB## group by origin_name, dest_name
Q5	select date_trunc(month,dep_timestamp) as ym, avg(arrdelay) as del from ##TAB## group by ym
Q6	select dest_name, extract(month from dep_timestamp) as m, extract(year from dep_timestamp) as y, avg(arrdelay) as del from ##TAB## group by dest_name,y,m
Q7	select count(*) from ##TAB## where origin_name = 'Lambert-St Louis International'
Q8	select count(*) from ##TAB## where origin_name='Lambert-St Louis International' and dest_name = 'Lincoln Municipal'
Q9	select uniquecarrier,flightnum,dep_timestamp,dest_lat from ##TAB## where origin_name = 'Lambert-St Louis International' and flightnum=586
Q10	select origin_name, dest_name, avg(arrdelay), avg(depdelay), avg(arrdelay * depdelay) from ##TAB## group by origin_name, dest_name
Q11	select uniquecarrier,flightnum,dep_timestamp,dest_lat from ##TAB## where origin_name = 'Lambert-St Louis International' and flightnum=586 limit 5000
Q12	SELECT ##TAB##.carrier_name as key0,AVG(##TAB##.depdelay) AS x,AVG(##TAB##.arrdelay) AS y,COUNT(*) AS size FROM ##TAB## WHERE ((##TAB##.dep_timestamp >= TIMESTAMP(0) '1996-07-26 16:30:06' AND ##TAB##.dep_timestamp < TIMESTAMP(0) '1997-05-16 16:30:06')) GROUP BY key0 ORDER BY size DESC LIMIT 50
Q13	SELECT extract(month from ##TAB##.arr_timestamp) as key0, extract(isodow from ##TAB##.arr_timestamp) as key1, COUNT(*) AS color FROM ##TAB## WHERE ((##TAB##.dep_timestamp >= TIMESTAMP(0) '1996-07-28 00:00:00' AND ##TAB##.dep_timestamp < TIMESTAMP(0) '1997-05-18 00:00:00')) GROUP BY key0, key1 ORDER BY key0, key1
Q14	SELECT ##TAB##.dest_state as key0, AVG(##TAB##.arrdelay) AS val FROM ##TAB## WHERE ((##TAB##.dep_timestamp >= TIMESTAMP(0) '1996-07-28 00:00:00' AND ##TAB##.dep_timestamp < TIMESTAMP(0) '1997-05-18 00:00:00')) GROUP BY key0 ORDER BY key0
Q15	SELECT ##TAB##.dest_state as key0, AVG(##TAB##.arrdelay) AS val FROM ##TAB## WHERE ((##TAB##.dep_timestamp >= TIMESTAMP(0) '1996-07-28 00:00:00' AND ##TAB##.dep_timestamp < TIMESTAMP(0) '1997-05-18 00:00:00')) GROUP BY key0 ORDER BY key0
Q16	select origin_name, dest_name, avg(arrdelay), avg(depdelay), avg(arrdelay + depdelay) from ##TAB## group by origin_name, dest_name

Query	Query Text
Q17	select date_trunc(month, dep_timestamp_32_fixed) as ym, avg(arrdelay) as del from ##TAB## group by ym
Q18	select date_trunc(month, dep_timestamp_3) as ym, avg(arrdelay) as del from ##TAB## group by ym
Q19	select date_trunc(month, dep_timestamp_9) as ym, avg(arrdelay) as del from ##TAB## group by ym
Q20	select dest_name, extract(month from dep_timestamp_32_fixed) as m, extract(year from dep_timestamp_32_fixed) as y, avg(arrdelay) as del from ##TAB## group by dest_name, y, m
Q21	select dest_name, extract(month from dep_timestamp_6) as m, extract(year from dep_timestamp_6) as y, avg(arrdelay) as del from ##TAB## group by dest_name, y, m
Q22	select dest_name, extract(month from dep_timestamp_9) as m, extract(year from dep_timestamp_9) as y, avg(arrdelay) as del from ##TAB## group by dest_name, y, m
Q23	SELECT ##TAB##.carrier_name as key0, AVG(##TAB##.depdelay) AS x, AVG(##TAB##.arrdelay) AS y, COUNT(*) AS size FROM ##TAB## WHERE ((##TAB##.dep_timestamp_32_fixed >= TIMESTAMP(0) '1996-07-26 16:30:06' AND ##TAB##.dep_timestamp_32_fixed < TIMESTAMP(0) '1997-05-16 16:30:06')) GROUP BY key0 ORDER BY size DESC LIMIT 50
Q24	SELECT ##TAB##.carrier_name as key0, AVG(##TAB##.depdelay) AS x, AVG(##TAB##.arrdelay) AS y, COUNT(*) AS size FROM ##TAB## WHERE ((##TAB##.dep_timestamp_3 >= TIMESTAMP(3) '1996-07-26 16:30:06.000' AND ##TAB##.dep_timestamp_3 < TIMESTAMP(3) '1997-05-16 16:30:06.000')) GROUP BY key0 ORDER BY size DESC LIMIT 50
Q25	SELECT ##TAB##.carrier_name as key0, AVG(##TAB##.depdelay) AS x, AVG(##TAB##.arrdelay) AS y, COUNT(*) AS size FROM ##TAB## WHERE ((##TAB##.dep_timestamp_6 >= TIMESTAMP(6) '1996-07-26 16:30:06.000000' AND ##TAB##.dep_timestamp_6 < TIMESTAMP(6) '1997-05-16 16:30:06.000000')) GROUP BY key0 ORDER BY size DESC LIMIT 50
Q26	SELECT ##TAB##.carrier_name as key0, AVG(##TAB##.depdelay) AS x, AVG(##TAB##.arrdelay) AS y, COUNT(*) AS size FROM ##TAB## WHERE ((##TAB##.dep_timestamp_6 >= TIMESTAMP(6) '1996-07-26 16:30:06.000000' AND ##TAB##.dep_timestamp_6 < TIMESTAMP(6) '1997-05-16 16:30:06.000000')) GROUP BY key0 ORDER BY size DESC LIMIT 50
Q27	SELECT extract(month from ##TAB##.arr_timestamp_32_fixed) as key0, extract(isodow from ##TAB##.arr_timestamp_32_fixed) as key1, COUNT(*) AS color FROM ##TAB## WHERE ((##TAB##.dep_timestamp_32_fixed >= TIMESTAMP(0) '1996-07-28 00:00:00' AND ##TAB##.dep_timestamp_32_fixed < TIMESTAMP(0) '1997-05-18 00:00:00')) GROUP BY key0, key1 ORDER BY key0, key1
Q28	SELECT extract(month from ##TAB##.arr_timestamp_3) as key0, extract(isodow from ##TAB##.arr_timestamp_3) as key1, COUNT(*) AS color FROM ##TAB## WHERE ((##TAB##.dep_timestamp_3 >= TIMESTAMP(3) '1996-07-28 00:00:00.000' AND ##TAB##.dep_timestamp_3 < TIMESTAMP(3) '1997-05-18 00:00:00.000')) GROUP BY key0, key1 ORDER BY key0, key1
Q29	SELECT extract(month from ##TAB##.arr_timestamp_6) as key0, extract(isodow from ##TAB##.arr_timestamp_6) as key1, COUNT(*) AS color FROM ##TAB## WHERE ((##TAB##.dep_timestamp_6 >= TIMESTAMP(6) '1996-07-28 00:00:00.000000' AND ##TAB##.dep_timestamp_6 < TIMESTAMP(6) '1997-05-18 00:00:00.000000')) GROUP BY key0, key1 ORDER BY key0, key1

Query	Query Text
Q30	SELECT extract(month from ##TAB##.arr_timestamp_6) as key0, extract(isodow from ##TAB##.arr_timestamp_6) as key1, COUNT(*) AS color FROM ##TAB## WHERE ((##TAB##.dep_timestamp_6 >= TIMESTAMP(6) '1996-07-28 00:00:00.000000' AND ##TAB##.dep_timestamp_6 < TIMESTAMP(6) '1997-05-18 00:00:00.000000')) GROUP BY key0, key1 ORDER BY key0, key1
Q31	select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty, sum(l_extendedprice) as sum_base_price, sum(l_extendedprice * (1 - l_discount)) as sum_disc_price, sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge, avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price, avg(l_discount) as avg_disc, count(*) as count_order from lineitem where l_shipdate <= date '1998-12-01' - interval '90' day (3) group by l_returnflag, l_linestatus order by l_returnflag, l_linestatus
Q32	select l_orderkey, sum(l_extendedprice * (1 - l_discount)) as revenue, o_orderdate, o_shippriori
Q33	select n_name, sum(l_extendedprice * (1 - l_discount)) as revenue from customer, orders, lineitem, supplier, nation, region where c_custkey = o_custkey and l_orderkey = o_orderkey and l_suppkey = s_suppkey and c_nationkey = s_nationkey and s_nationkey = n_nationkey and n_regionkey = r_regionkey and r_name = 'ASIA' and o_orderdate >= date '1994-01-01' and o_orderdate < date '1994-01-01' + interval '1' year group by n_name order by revenue desc
Q34	select sum(l_extendedprice * l_discount) as revenue from lineitem where l_shipdate >= date '1994-01-01' and l_shipdate < date '1994-01-01' + interval '1' year and l_discount between.06 - 0.01 and.06 + 0.01 and l_quantity < 24
Q35	select supp_nation, cust_nation, l_year, sum(volume) as revenue from (select n1.n_name as supp_nation, n2.n_name as cust_nation, extract(year from l_shipdate) as l_year, l_extendedprice * (1 - l_discount) as volume from supplier, lineitem, orders, customer, nation n1, nation n2 where s_suppkey = l_suppkey and o_orderkey = l_orderkey and c_custkey = o_custkey and s_nationkey = n1.n_nationkey and c_nationkey = n2.n_nationkey and ((n1.n_name = 'FRANCE' and n2.n_name = 'GERMANY') or (n1.n_name = 'GERMANY' and n2.n_name = 'FRANCE')) and l_shipdate between date '1995-01-01' and date '1996-12-31') as shipping group by supp_nation, cust_nation, l_year order by supp_nation, cust_nation, l_year
Q36	select o_year, sum(case when nation = 'BRAZIL' then volume else 0 end) / sum(volume) as mkt_share from (select extract(year from o_orderdate) as o_year, l_extendedprice * (1 - l_discount) as volume, n2.n_name as nation from part, supplier, lineitem, orders, customer, nation n1, nation n2, region where p_partkey = l_partkey and s_suppkey = l_suppkey and l_orderkey = o_orderkey and o_custkey = c_custkey and c_nationkey = n1.n_nationkey and n1.n_regionkey = r_regionkey and r_name = 'AMERICA' and s_nationkey = n2.n_nationkey and o_orderdate between date '1995-01-01' and date '1996-12-31' and p_type = 'ECONOMY ANODIZED STEEL') as all_nations group by o_year order by o_year
Q37	select c_custkey, c_name, sum(l_extendedprice * (1 - l_discount)) as revenue, c_acctbal, n_name, c_address, c_phone, c_comment from customer, orders, lineitem, nation where c_custkey = o_custkey and l_orderkey = o_orderkey and o_orderdate >= date '1993-10-01' and o_orderdate < date '1993-10-01' + interval '3' month and l_returnflag = 'R' and c_nationkey = n_nationkey group by c_custkey, c_name, c_acctbal, c_phone, n_name, c_address, c_comment order by revenue desc
Q38	select l_shipmode, sum(case when o_orderpriority = '1-URGENT' or o_orderpriority = '2-HIGH' then 1 else 0 end) as high_line_count, sum(case when o_orderpriority <> '1-URGENT' and o_orderpriority <> '2-HIGH' then 1 else 0 end) as low_line_count from orders, lineitem where o_orderkey = l_orderkey and l_shipmode in ('MAIL', 'SHIP') and l_commitdate < l_receiptdate and l_shipdate < l_commitdate and l_receiptdate >= date '1994-01-01' and l_receiptdate < date '1994-01-01' + interval '1' year group by l_shipmode order by l_shipmode

Query	Query Text
Q39	select 100.00 * sum(case when p_type like 'PROMO%' then cast(l_extendedprice as FLOAT) * (1 - l_discount) else 0 end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue from lineitem, part where l_partkey = p_partkey and l_shipdate >= date '1995-09-01' and l_shipdate < date '1995-09-01' + interval '1' month
Q40	select p_brand, p_type, p_size, count(distinct ps_suppkey) as supplier_cnt from partsupp, part where p_partkey = ps_partkey and p_brand <> 'Brand#45' and p_type not like 'MEDIUM POLISHED%' and p_size in (49, 14, 23, 45, 19, 3, 36, 9) and ps_suppkey not in (select s_suppkey from supplier where s_comment like '%Customer%Complaints%') group by p_brand, p_type, p_size order by supplier_cnt desc, p_brand, p_type, p_size
Q41	TPCHQ19.sql|select sum(l_extendedprice * (1 - l_discount)) as revenue from lineitem, part where (p_partkey = l_partkey and p_brand = 'Brand#12' and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG') and l_quantity >= 1 and l_quantity <= 1 + 10 and p_size between 1 and 5 and l_shipmode in ('AIR', 'AIR REG') and l_shipinstruct = 'DELIVER IN PERSON') or (p_partkey = l_partkey and p_brand = 'Brand#23' and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK') and l_quantity >= 10 and l_quantity <= 10 + 10 and p_size between 1 and 10 and l_shipmode in ('AIR', 'AIR REG') and l_shipinstruct = 'DELIVER IN PERSON') or (p_partkey = l_partkey and p_brand = 'Brand#34' and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG') and l_quantity >= 20 and l_quantity <= 20 + 10 and p_size between 1 and 15 and l_shipmode in ('AIR', 'AIR REG') and l_shipinstruct = 'DELIVER IN PERSON')
Q42	TWQ01.sql|SELECT UNNEST(hashtags) as key0, COUNT(*) AS val FROM ##TAB## GROUP BY key0 ORDER BY val DESC LIMIT 227
Q43	TWQ02.sql|select sender_id, count(*) val from ##TAB## group by sender_id order by val desc limit 100
Q44	TWQ03.sql|select sender_id, origin, count(*) val from ##TAB## group by sender_id, origin order by val desc limit 5

Benchmark Results

We measured an average speedup of 72% across the 44 SQL queries representative of the OmniSci software, with speedups up to 198% for Q40.

Query #	AC922 First Run (ms)	X86 First Run (ms)	Speedup
Q1	334	360	8%
Q2	862	993	15%
Q3	1105	1139	3%
Q4	1618	2042	26%
Q5	1055	1493	42%
Q6	269	393	46%
Q7	478	567	19%
Q8	574	914	59%
Q9	268	364	36%
Q10	401	523	30%
Q11	361	434	20%
Q12	147	256	74%
Q13	336	493	47%
Q14	281	391	39%
Q15	295	463	57%
Q16	774	1,252	62%
Q17	397	577	45%
Q18	570	922	62%
Q19	584	928	59%

Query #	AC922 First Run (ms)	X86 First Run (ms)	Speedup
Q20	719	1,037	44%
Q21	925	1,449	57%
Q22	1,303	1,359	4%
Q23	353	393	11%
Q24	240	503	110%
Q25	241	470	95%
Q26	239	405	69%
Q27	270	271	0%
Q28	422	486	15%
Q29	273	480	76%
Q30	271	462	70%
Q31	1,136	1,217	7%
Q32	2,688	2,949	10%
Q33	1,094	1,435	31%
Q34	421	500	19%
Q35	1,110	1,406	27%
Q36	1,306	1,717	31%
Q37	5,528	8,379	52%
Q38	908	1,130	24%
Q39	601	717	19%
Q40	8,951	26,644	198%

Query #	AC922 First Run (ms)	X86 First Run (ms)	Speedup
Q41	665	783	18%
Q42	680	1,011	49%
Q43	2,310	4,275	85%
Q44	3,252	6,254	92%
AVERAGE	1,059	1,824	72%

Bring IBM Power System AC922 Cognitive Infrastructure for Enterprise AI, NVIDIA NVLink and OmniSci to Your Team

Learn More:

Take advantage of GPU-accelerated analytics to answer questions that have been out of reach before the rise of GPUs. Contact IBM and OmniSci (**sales@omnisci.com**) today to learn more about the unprecedented power of OmniSci on IBM Power System AC922 Cognitive Infrastructure for Enterprise AI.

For further information visit:
www.omnisci.com

In Partnership with: