IN-DATABASE RASTER ANALYTICS: MAP ALGEBRA AND PARALLEL PROCESSING IN ORACLE SPATIAL GEORASTER

Qingyun (Jeffrey) Xie*, Zhihai Zhang, Siva Ravada

Oracle Corporation, One Oracle Drive, Nashua, NH 03062, USA -
(qingyun.xie, zhihai.zhang, siva.ravada)@oracle.com

KEYWORDS: raster, image, database, analytical, processing, query, management, software

ABSTRACT:

Over the past decade several products have been using enterprise database technology to store and manage geospatial imagery and raster data inside RDBMS, which in turn provides the best manageability and security. With the data volume growing exponentially, real-time or near real-time processing and analysis of such big data becomes more challenging. Oracle Spatial GeoRaster, different from most other products, takes the enterprise database-centric approach for both data management and data processing. This paper describes one of the central components of this database-centric approach: the processing engine built completely inside the database. Part of this processing engine is raster algebra, which we call the In-database Raster Analytics. This paper discusses the three key characteristics of this in-database analytics engine and the benefits. First, it moves the data processing closer to the data instead of moving the data to the processing, which helps achieve greater performance by overcoming the bottleneck of computer networks. Second, we designed and implemented a new raster algebra expression language. This language is based on PL/SQL and is currently focused on the "local" function type of map algebra. This language includes general arithmetic, logical and relational operators and any combination of them, which dramatically improves the analytical capability of the GeoRaster database. The third feature is the implementation of parallel processing of such operations to further improve performance. This paper also presents some sample use cases. The testing results demonstrate that this in-database approach for raster analytics can effectively help solve the biggest performance challenges we are facing today with big raster and image data.

1. INTRODUCTION

There are some prominent characteristics of geospatial imagery and raster data. First, they are special and complex data types in comparison with structured and simple data types such as numbers and strings. Second, they require specialized indexing, querying, processing and analyzing algorithms. Thirdly, they are generally huge in size, thus they are "big data" in nature. These mean that we have to build special processing and analysis engines for the geospatial image and raster database. And scalability and performance of such systems are keys to success.

For geospatial image and raster data archiving and management, enterprise RDBMS technologies have been widely used as the foundation. Over the past decade, several products including GeoRaster, RasDaMan, and ArcSDE have demonstrated this database technology (Baumann, 2001. ESRI, 2005. Oracle, 2004). The common feature of these products is to store image and raster data inside RDBMS databases, which in turn provide the best manageability and security. GeoRaster is unique because it takes the database-centric approach (Xie, 2008a. Xie, 2011). This approach not only builds spatial indices but also provides all data management and query operations inside the database itself. It is truly scalable and provides greater performance by removing the need of constantly moving the datasets in and out of the database.

For geospatial image and raster data processing and analyzing, many advanced and highly efficient desktop systems such as ERDAS Imagine and PCI Geomatica and server-based engines such as ArcGIS are readily available. When a large-scale enterprise RDBMS based spatial database is built, such desktop and server-based systems generally can connect to it and then retrieve the imagery and raster data out of the database and process them in the client or another server. However, moving the data between the database and the processing engine is costly given the speed and bandwidth limitations of the computer networks.

With the data volume growing exponentially, real-time and near real-time processing and analysis of such big data becomes more important and urgent. So, building a fast processing and analysis solution for the image and raster databases is critical. Oracle Spatial GeoRaster takes the enterprise database-centric approach for both data management and data processing. This paper presents one of the central components of this database-centric approach: the processing engine built completely inside the database. Part of this processing engine is raster algebra, which we call the In-database Raster Analytics. There are three key features of this in-database raster analytics engine. First, it moves the data processing closer to the data instead of moving the data to the processing. Second, to implement this we designed a new raster algebra language. The third feature is the implementation of parallel processing of such raster operations inside the database. This paper discusses these key characteristics of this in-database analytics engine and the advantages.

2. IN-DATABASE PROCESSING

In-database processing, also known as in-database analytics, refers to the integration of data processing and analytical functionalities into the databases or data warehouses. The basic idea is to eliminate the overhead of moving large data sets from the enterprise databases to separate processing and analytical software applications.

An in-database analytics approach is much faster, more efficient, and more secure than traditional analytics approaches. In-database analytics delivers immediate performance, scalability and security improvements because data never leaves the database until results are filtered and processed (Das, 2010).

In-database processing is performed and promoted as a feature by many of the major database and data warehousing vendors, including Oracle, IBM, Teradata, Netezza, Greenplum and Aster Data Systems (Grimes, 2008. Berger, 2009). For example, Oracle Data Mining and Oracle R Enterprise are in-database data analysis engines. Coupled with the power of SQL, they eliminate data movement and duplication, maintain security and minimize latency time from raw data to valuable information.

In-database processing has been successfully used in many high-throughput and mission-critical applications, including fraud detection, pricing and margin analysis. The success of this approach and its applications inspired us to consider the same strategy for image and raster processing and analytics inside Oracle Spatial GeoRaster.

As we mentioned in the introduction, geospatial imagery and raster data are big data. A typical geoimage database has tens or hundreds of terabytes of data. Petabytes of data is not abnormal. Data has "weight" and geospatial image and raster data sets are particularly "heavy". Given that the processing and analysis are data intensive, data locality should always be an important factor in our design and implementation strategy. So we conclude that building an in-database analytics engine should be a good strategy. It moves the data processing closer to the data instead of moving the data to the processing, which helps achieve greater performance by overcoming the bottleneck of computer networks.

3. THE MAP ALGEBRA LANGUAGE

Image and raster data processing and analysis involve a large set of operations, including image geometric corrections, image enhancement and classifications, map algebraic operations, terrain analysis, geostatistics, to name a few. Since map algebra is the basic and most commonly used technique in raster data analysis and GIS modeling, we mainly discuss its implementation in this paper.

Developed through the 1980′s by Professor C. Dana Tomlin as part of his PhD thesis work, Map Algebra is a high-level language providing a framework for performing raster data analysis and cartographic modeling. Map Algebra includes a set of operators, such as arithmetic, boolean, logical, relational, and combinatorial operations. It also includes a set of functions, which are generally classified into four categories: local, focal, zonal and global (Tomlin, 1990).

There are many implementations of Map Algebra. However, the exact syntax and workflow of the expressions and functions could be very different among those implementations, while the concepts and functionality remain the same. Generally, a computing language should include declaration of variables and constants, data processing operations (expressions) and procedures (functions), statements and programs. We think the same should be true for a good Map Algebra implementation.

PL/SQL, the Oracle procedural extension of SQL, is a portable, high-performance transaction-processing language. PL/SQL combines the data-manipulating power of SQL with the processing power of procedural languages. You can control program flow with statements like IF and LOOP. As with other procedural programming languages, you can declare variables, define procedures and functions, and trap runtime errors. PL/SQL lets you break complex problems down into easily understandable procedural code, and reuse this code across multiple applications. When a problem can be solved through plain SQL, you can issue SQL commands directly inside your PL/SQL programs, without learning new APIs. PL/SQL's data types correspond with SQL's column types, making it easy to interchange PL/SQL variables with data inside a table (Oracle, 2012).

Oracle Spatial GeoRaster is completely built inside the enterprise Oracle database server. The PL/SQL language is available to GeoRaster already and the users are mainly using PL/SQL to manage, query and manipulate GeoRaster objects. So, we can further leverage the power of the PL/SQL language. For our geospatial analysis purposes, what this language lacks is the specific map algebra expressions and functions.

To implement this we designed a new raster algebra expression language covering general arithmetic, casting, logical and relational operators as shown below.

```
arithmeticBinaryOp:
          +
        | -
        | *
        | /
comparisonOp:
          =
        | <
        | >
        | >=
        | <=
        | !=
arithmeticUnaryOp:
          +
        | -
booleanBinaryOp:
          &
        | |
booleanUnaryOp:
          !
rangeType:
           castint
         | castonebit
         | casttwobit
         | castfourbit
         | casteightbit
numericFunction:
           abs
         | sqrt
         | exp
         | log
         | ln
         | sin
         | cos
         | tan
         | sinh
         | cosh
         | tanh
         | arcsin
         | arccos
         | arctan
         | ceil
         | floor
ID:
        integer number
constantNumber:
        double number
band:
        integer number
```

```
identifier:
        {ID,band}
        | {band}
unaryArithmeticExpr:
          (arithmeticUnaryOp arithmeticExpr)
binaryArithmeticExpr:
          arithmeticExpr  arithmeticBinaryOp arithmeticExpr
functionArithmeticExpr:
          numericFunction (arithmeticExpr)
arithmeticExpr:
          unaryArithmeticExpr
        | binaryArithmeticExpr
        | functionArithmeticExpr
        | (arithmeticExpr)
        | constantNumber
        | castingExpr
        | identifier
castingExpr:
        rangeType(arithmeticExpr)
unaryBooleanExpr:
          booleanUnaryOp booleanExpr
binaryBooleanExpr:
          booleanExpr booleanBinaryOp booleanExpr
booleanExpr:
          unaryBooleanExpr
        | binaryBooleanExpr
        | (booleanExpr)
        | arithmeticExpr comparisonOp arithmeticExpr
```

The "identifier" in the expression refers to a raster layer of a GeoRaster object. It is either a single band number if there is only one GeoRaster object involved, or a pair of (ID, band) where ID refers to one of GeoRaster objects in the expression and band refers to a specific band of that GeoRaster object.

We also developed four major procedures including arithmetic operation, conditional query, classify and cell value-based update as follows.

sdo_geor_ra.rastermathop: runs arithmeticExpr operations.
sdo_geor_ra.findcells: searches cells based on booleanExpr.
sdo_geor_ra.classify: applies arithmeticExpr to cells and then segments the raster.
sdo_geor_ra.rasterupdate: updates cells of a raster based booleanExpr.

Each of these procedures take many layers from one or many GeoRaster objects, apply booleanExpr and/or arithmeticExpr expressions over those layers, do the specific algebraic computation or modeling, and output a new GeoRaster object. The expressions can be defined in any way based the syntax of the expression language above.

4. PARALLEL PROCESSING

As we mentioned in the introduction, scalability and performance of such systems are also keys to success. The scalability of GeoRaster in the database has been mainly solved by the design of the GeoRaster data model, the control of memory usage in the GeoRaster engine, and the application of Oracle GRID Computing technologies (Xie, 2006. Xie, 2008a. Xie 2008b). This scalability applies to this in-database map algebra as well. So, our focus here is mainly about performance of the processing engine.

Performance depends upon the design and implementation of the in-database processing strategy, the processing algorithms, speed of I/O, flexible memory utilization, to name a few. Given that modern computers are mostly multicore or have multiple CPUs, parallel processing becomes a very important solution for speedup. Parallel processing divides a large task into many smaller tasks, and executes the smaller tasks concurrently in different CPU's or on several computing nodes. As a result, the larger task completes more quickly. The major benefits of parallelism are speedup (faster) and scaleup (more users) for massive data processing operations. So it should be an essential factor in our software implementation. Note that concurrency is already part of the GeoRaster database, which can help improve the speed of massive data processing too (Xie, 2006).

The Oracle database provides a powerful SQL parallel execution engine that can run almost any SQL-based operation – DDL, DML and queries – in the Oracle Database in parallel. When you execute a SQL statement in the Oracle Database it is decomposed into individual steps or row-sources, which are identified as separate lines in an execution plan (Dijcks, 2010).

With this parallel execution framework, however, the individual raster processing functions, such as mosaic and raster algebra operations, cannot be directly parallelized without some special implementation. This is because each of the heavy image processing and raster manipulation operations is not purely row-based and has its own logic in how the raster data (or raster blocks) are internally processed.

There are several ways to leverage the oracle parallel execution engine, among which pipelined and parallel table function is an important aspect of parallelism. Table functions can be used and controlled by any user. The goal of a set of table functions is to build a parallel processing pipeline leveraging the parallel processing framework in the database (Oracle 2008. Dijcks, 2010). We leverage table functions to encapsulate complex logic in a PL/SQL construct so that we can process different subsets of the data of a GeoRaster object in parallel. To parallelize those operations we have to begin with explicitly controlling the level of degree of parallelism and deciding what subsets of the data to be handled in each subprocess. We used the output raster to split the whole region into subsets and the total number of subsets is decided by the degree of parallelism (DOP), which can be controlled by user input. Then the Oracle parallel execution framework will split the whole task into different subprocesses based on the total number of subsets and each subprocess will process one of the subsets independently. When all subsets are finished, the whole process is done.

As an example, the following conditional query finds all pixels in a three-band image where the cell value of the first band is greater than 10 and less than 50, the cell value of the second band is greater than or equal to 100 and less than 150, and the cell value of the third band is greater than 200 and less than 245. The result is a new image of all pixels meeting the query condition. The parameter 'parallel=4' means the process will be parallelized into 4 processes, each of which will process a quarter of the original image simultaneously, thus the overall performance will be improved significantly.

```
declare
  geor   SDO_GEORASTER; -- source image
  geor1  SDO_GEORASTER; -- result image
begin
  select georaster into geor from georaster_table where georid = 1;
  select georaster into geor1 from georaster_table where georid = 2 for update;
  sdo_geor_ra.findcells (
    geor,
    '(({0}>10)&({0}<50)&{1}>=100)&({1}<150)&({2}>200)&({2}<245)',
    null, geor1, null, 'false',
```

```
    'parallel=4');
  update georaster_table set georaster = geor1 where georid = 2;
  commit;
end;
```

We conducted some initial tests on the parallel implementation of sdo_geor_ra.findcells as well as sdo_geor_ra.classify. The sdo_geor.findcells test uses the above script to query an image based on all three bands while the sdo_geor_ra.classify test segments the first band of the three-band image into 12 classes. We used a x86_64 Linux Server to do the tests. It has 4 Intel(R) Xeon(R) X5670 CPUs and the CPU speed is 2.93GHz. The total memory is 8GB. The operating system is Red Hat Enterprise Linux Server 5.4. We used four 3-band images of different sizes. The results are shown in table 1 and 2.

Table 1. Execution time in seconds of sdo_geor.findcells with and without parallelism

Degree of Parallel ism	image size and image dimension size in row x column x band			
	238.7MB 8930x8912x3	954.8MB 17860x17824x3	2.15GB 26790x26736x3	3.82GB 35720x35648x3
1	17.33	45.14	87.80	153.36
2	15.33	26.43	55.19	106.00
4	11.67	23.33	49.80	80.79
8	10.12	20.68	40.92	74.63

Table 2. Execution time in seconds of sdo_geor.classify with and without parallelism

Degree of Parallelism	image size and dimension size in row x column x band			
	238.7MB 8930x8912x3	954.8MB 17860x17824x3	2.15GB 26790x26736x3	3.82GB 35720x35648x3
1	6.47	44.03	80.50	138.45
2	4.89	36.33	58.19	86.68
4	3.86	26.52	44.84	77.55
8	3.50	21.35	41.09	68.00

There is only 1 disk on this machine so the I/O contention among parallelized subprocesses is very high. That has a big impact on the performance numbers. However, even with only one disk, table 1 and 2 show that when the raster algebra operation is parallelized into 2 to 8 subprocesses, the processing operation is significantly faster than the same operation without parallelism. In addition, the overall performance improvement scales very well with image size increasing as shown in table 1 and 2. There are still more room for improvement yet to be done. However, we can reasonably assume the performance improvement could be much better if the machine has more CPU's and more memory, and particularly if a high-speed storage cluster (using Oracle ASM technology) or a high-end machine such as Oracle Exadata Database Machine is used. Some database tuning techniques will help improve parallel performance as well.

5. APPLICATIONS

Currently, the raster algebra engine implementation is focused only on the "local" function type of map algebra and is designed to work with the standard PL/SQL language and run completely inside the database. Using the PL/SQL and the raster algebra expressions and functions, users can implement a wide range of applications, such as applying complex pixel queries in the database, editing a raster based on raster cell values and formulated query conditions, segmenting images or classifying a thematic map, and conducting cartographic modeling over a large number of rasters and images of unlimited size. The engine runs these algebra expressions and functions as single processes inside the database and each of those processes can be parallelized, thus dramatically improves the analytical capability and performance of the GeoRaster database.

Map Algebra is mainly used in cartographic modeling and is considered an essential component of any GIS systems. These applications and the importance of the map algebra expressions and functions are well known. Due to the lack of testing dataset of thematic layers for a case study area and the easy access of Landsat imagery, we only use Normalized Difference Vegetation Index (NDVI) and Tasseled Cap Transformation (TCT) as our application examples in this paper to demonstrate the capability.

In remote sensing, NDVI was one of the most successfully and widely used vegetation index (VI), which can quickly identify vegetated areas and monitor plant growth or their "condition". Using Landsat TM imagery, the standard NDVI computation formula is (TM4 – TM3) / (TM4 + TM3). The following script takes a Landsat 7 ETM+ image and compute the NDVI, which is stored as another raster of floating number data type. Note, in our algebra language, band number starts with 0, so the formula translates into the expression '({3}-{2})/({3}+{2})'.

```
declare
  geor1      MDSYS.SDO_GEORASTER;
  geor2      MDSYS.SDO_GEORASTER;
begin
  -- source ETM+ image
  select georaster into geor1 from georaster_table where georid = 2;
  -- to store NDVI
  select georaster into geor2 from georaster_table where georid = 3 for update;
  mdsys.sdo_geor_ra.rasterMathOp(geor1,
      SDO_STRING_ARRAY('({3}-{2})/({3}+{2})'),
      'celldepth=32bit_real',geor2);
  update georaster_table set georaster = geor2 where georid = 3;
  commit;
end;
```

Figure 1 shows a small area of the original ETM+ 543 image and the resulting NDVI image after running the above script.

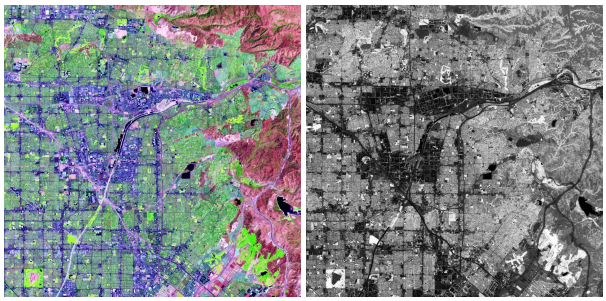

Fig. 1, ETM+ 543 color image (left) and NDVI image (right). Image Courtesy of PCI Geomatics.

The concept of tasseled cap transformation is a useful tool for compressing spectral data into a few bands associated with physical scene characteristics (Crist and Cicone 1984). TCT helps analyze the physical ground features. With Landsat imagery, it uses 5 bands of either original digital number (DN) or reflectance data to generate 6 new bands, each of which represents different ground features. The 6 resulting bands are

generally called (soil) brightness, (vegetation) greenness, (soil and canopy) wetness, haze, TC5, and TC6. Each or a combination of them is useful for different applications, including crop growth monitoring and analysis, biomass study, agriculture planning, to name a few. In this test, we used a full scene Landsat 5 TM image and the Landsat 5 Tasseled Cap Transformation coefficients for DN Data (Crist and Cicone, 1986). The following script takes a TM image as input, automatically generates the TCT expression based on the coefficient matrix, execute the expression, and create a new image holding the results.

```
declare
  type array_type is varray(6) of binary_double;
  type array_array_type is varray(6) of array_type;
  tct_coeff array_array_type := array_array_type(
    array_type( 0.3561,  0.3972,  0.3904,  0.6966,  0.2286,  0.1596),
    array_type(-0.3344, -0.3544, -0.4556,  0.6966, -0.0242, -0.2630),
    array_type( 0.2626,  0.2141,  0.0926,  0.0656, -0.7629, -0.5388),
    array_type( 0.0805, -0.0498,  0.1950, -0.1327,  0.5752, -0.7775),
    array_type(-0.7252, -0.0202,  0.6683,  0.0631, -0.1494, -0.0274),
    array_type( 0.4000, -0.8172,  0.3832,  0.0602, -0.1095,  0.0985));
  i integer;
  gr1 sdo_georaster;
  gr2 sdo_georaster;
  stmt varchar2(5024);
begin
  select georaster into gr1 from georaster_table where georid = 2;
  select georaster into gr2 from georaster_table where georid = 4 for update;
  stmt:='';
  i:=1;
  -- the following code generates the TCT expression
  LOOP
      stmt:=stmt||
        '''('||trim(to_char(tct_coeff(i)(1),'0.9999'))|| ')*{0}+' ||
        '('||trim(to_char(tct_coeff(i)(2),'0.9999'))|| ')*{1}+' ||
        '('||trim(to_char(tct_coeff(i)(3),'0.9999'))|| ')*{2}+' ||
        '('||trim(to_char(tct_coeff(i)(4),'0.9999'))|| ')*{3}+' ||
        '('||trim(to_char(tct_coeff(i)(5),'0.9999'))|| ')*{4}+' ||
        '('||trim(to_char(tct_coeff(i)(6),'0.9999'))|| ')*{6}''' ;
     IF(i<6) THEN
        stmt:=stmt||',';
     END if;
     i := i + 1;
     IF i > 6 THEN
      EXIT;
     END IF;
  END LOOP;
  stmt := 'call sdo_geor_ra.rasterMathOp(:1,SDO_STRING2_ARRAY('||stmt||'),'||
      '''celldepth=32BIT_REAL'',:2)';
  execute immediate stmt using gr1,in out gr2;
  update georaster_table set georaster = gr2 where georid = 4;
  commit;
end;
```

The actual raster algebra expression and command, i.e., the "stmt", generated by the above script is as follows (reformatted a bit for readability). Note that you can also directly apply this expression to make the above script look even simpler.

```
sdo_geor_ra.rasterMathOp(:1,  SDO_STRING2_ARRAY(
   '(0.3561)*{0}+(0.3972)*{1}+(0.3904)*{2}+(0.6966)*{3}+(0.2286)*{4}+(0.1596)*{6}',
   '(-0.3344)*{0}+(-0.3544)*{1}+(-0.4556)*{2}+(0.6966)*{3}+(-0.0242)*{4}+(-0.2630)*{6}',
   '(0.2626)*{0}+(0.2141)*{1}+(0.0926)*{2}+(0.0656)*{3} +(-0.7629)*{4}+(-0.5388)*{6}',
   '(0.0805)*{0}+(-0.0498)*{1}+(0.1950)*{2}+(-0.1327)*{3}+(0.5752)*{4}+(-0.7775)*{6}',
   '(-0.7252)*{0}+(-0.0202)*{1}+(0.6683)*{2}+(0.0631)*{3}+(-0.1494)*{4}+(-0.0274)*{6}',
   '(0.4000)*{0}+(-0.8172)*{1}+(0.3832)*{2}+(0.0602)*{3}+(-0.1095)*{4}+(0.0985)*{6}'),
   'celldepth=32Bit_REAL',  :2)
```

The source image (the file name is L5044034_03420110918.tif, available from the U.S. Geological Survey) covers the San Francisco Bay area and is 377MB (7091 rows, 7961 columns, 7 bands and 8bit integer) in size. The result image is 1.26GB (7091 rows, 7961 columns, 6 bands, and 32bit float) in size. In repeated tests on a low-end commodity linux machine, the total execution time (without parallelism) is only 3 minutes or less. Figure 2 shows a small subset of the source image, and the brightness, greenness and wetness resulted from the TCT.

Using the same raster algebra language, users can also very easily add some additional scripts in the above PL/SQL block to convert the 32bit floating number image into 8 bit integer image and in the mean time apply image stretching (simply another map algebra expression) on the TCT image to generate a new GeoRaster object for visualization and analysis purposes.

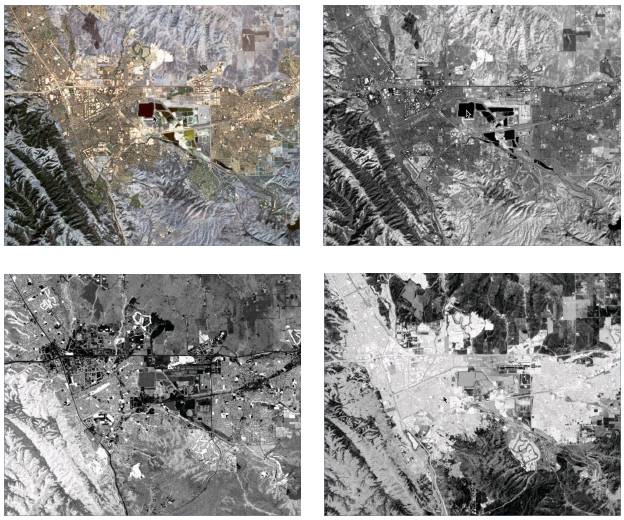

Figure 2. TM 123 Color Image (upper left), Brightness (upper right), Greenness (lower left) and Wetness (lower right). TM Image Courtesy of the U.S. Geological Survey

As shown in the syntax and the above examples, the PL/SQL language and the map algebra expressions are powerful and flexible for users to easily implement numerous processing and analytical applications. In addition to the optimized implementation of raster algebra algorithms and the embedded parallel processing, users can further leverage the power of Oracle Enterprise GRID Computing infrastructure to quickly process and analyze thousands of images and rasters stored in the GeoRaster database concurrently and on a global basis (Xie, 2008a).

6. CONCLUSIONS

Satellite imagery, airborne photographs and other geospatial raster data are complex data types that require specialized database management systems and analysis solutions. Unprecedented data volume plus real time or near-real time archiving and processing requirements of such data dictate extreme scalability and performance of such systems and solutions. Oracle Spatial GeoRaster takes an enterprise database-centric approach by enhancing Oracle database server to solve the database management challenges and achieve virtually unlimited scalability and great performance. The in-database raster analytics engine proposed in this paper enhances Oracle Spatial GeoRaster database management system by embedding some analytical algorithms inside the database allowing data to be processed where the data is stored. This approach coupled with parallel processing capabilities offers great performance benefits for many basic and commonly used

database management operations. This also removes the need of separate solutions for some GIS and business applications. For traditional remote sensing and GIS applications, specialized image processing packages and GIS solutions are still required. However, such third party solutions can also benefit greatly in performance from this analytics engine by pushing some basic data processing and filtering operations into the database so that less data is retrieved and transported into the client for further processing and analysis.

7. REFERENCES

Baumann, P., 2001. Web-enabled Raster GIS Services for Large Image and Map Databases. In: *5th Int'l Workshop on Query Processing and Multimedia Issues in Distributed Systems (QPMIDS'2001)*, Munich, Germany, September 3-4, 2001

Berger C., 2009. Oracle Data Mining 11g: Competing on In-Database Analytics, An Oracle Technical White Paper. http://www.oracle.com/technetwork/database/options/odm/oracledatamining11gwpv3-130702.pdf (16 March 2012)

Crist, E., R. Cicone, 1984. A physically-based transformation of Thematic Mapper data -- the TM Tasseled Cap. *IEEE Trans. on Geosciences and Remote Sensing*, GE-22: 256-263.

Crist, E., R. Laurin, R. Cicone, 1986. Vegetation and soils information contained in transformed Thematic Mapper data. In: *Proceedings of IGARSS '86 Symposium*, 1465-70. Ref. ESA SP-254. Paris: European Space Agency.

Das J., 2010. Adding Competitive Muscle with In-Database Analytics. Database Trends and Applications. http://www.dbta.com/Articles/Editorial/Trends-and-applications/Adding-Competitive-Muscle-with-In-Database-Analytics-67126.aspx (16 March 2012)

Dijcks, J., H. Baer, M. Colgan, 2010. Oracle Database Parallel Execution Fundamentals, An Oracle Technical Whitepaper. http://www.oracle.com/technetwork/articles/datawarehouse/twp-parallel-execution-fundamentals-133639.pdf (16 March 2012)

ESRI, 2005. Raster Data in ArcSDE® 9.1 - An ESRI White Paper. http://www.esri.com/library/whitepapers/pdfs/arcsde91-raster.pdf (16 March 2012)

Grimes, S., 2008. In-Database Analytics: A Passing Lane for Complex Analysis. Information Week. http://www.informationweek.com/news/software/bi/212500351?cid=RSSfeed_IE_News (16 March 2012)

Oracle, 2004. *Oracle Spatial GeoRaster, 10g Release 1 (10.1)*.

Oracle, 2008. Oracle Database Data Cartridge Developer's Guide 11*g* Release 1 (11.1). http://docs.oracle.com/cd/B28359_01/appdev.111/b28425/pipe_paral_tbl_ref.htm#i76723 (16 March 2012)

Oracle, 2012. Oracle Database PL/SQL Language Reference 11*g* Release 2 (11.2). http://docs.oracle.com/cd/E11882_01/appdev.112/e25519.pdf (16 March 2012)

Tomlin, D., 1990. *Geographic Information Systems and Cartographic Modeling*. Prentice Hall Inc.

Xie, Q., Z. Li, W. Xu, 2006. Using Enterprise Grid Computing Technologies to Manage Large-Scale Geoimage And Raster Databases. In: *the Proceedings of ASPRS 2006 Annual Conference*, Reno, Nevada, May 1 – 5, 2006.

Xie, Q., S. Ravada, W. Xu, Z. Zhang, 2008a. An Enterprise Database-centric Approach for Geospatial Image Management and Processing. In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Beijing, China, Vol. XXXVII. Part B4. pp. 199 – 204.

Xie, Q., 2008b. Oracle Spatial, Raster Data. *Encyclopedia of GIS*, Shashi Shekhar and Hui Xiong (editors), Springer. pp. 826 - 832.

Xie, Q., J. Sharma, J. Ihm, 2011. Oracle Spatial 11g GeoRaster, An Oracle Technical White Paper. http://download.oracle.com/otndocs/products/spatial/pdf/spatial11gr2_georaster_twp.pdf (16 March 2012)