SANDIA REPORT

SAND2006-1853
Unlimited Release
Printed April 2006

Data-Centric Computing with the Netezza Architecture

George S. Davidson, Kevin W. Boyack, Ron A. Zacharski,
Stephen C. Helmreich, and Jim R. Cowie

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2006-1853
Unlimited Release
Printed April 2006

Data-Centric Computing with the Netezza Architecture

George S. Davidson and Kevin W. Boyack
Computational Biology
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0310

Ron A. Zacharski, Stephen C. Helmreich, and Jim R. Cowie
Computing Research Laboratory
New Mexico State University
Las Cruces, NM 88003

Abstract

While relational databases have become critically important in business applications and web services, they have played a relatively minor role in scientific computing, which has generally been concerned with modeling and simulation activities. However, massively parallel database architectures are beginning offer the ability to quickly search through terabytes of data with hundred-fold or even thousand-fold speedup over server-based architectures. These new machines may enable an entirely new class of algorithms for scientific applications, especially when the fundamental computation involves searching through abstract graphs. Three examples are examined and results are reported for implementations on a novel, massively parallel database computer, which enabled very high performance. Promising results from (1) computation of bibliographic couplings, (2) graph searches for sub-circuit motifs within integrated circuit netlists, and (3) a new approach to word sense disambiguation in natural language processing, all suggest that the computational science community might be able to make good use of these new database machines.

Acknowledgements

The authors would like to thank Craig Sylvester (Netezza, Inc.) for his significant assistance during this project. We also thank Brent Meyer (Sandia National Laboratories) for his collaboration with respect to the netlist processing project.

CONTENTS

FIGURES

TABLES

1. Introduction

Imagine an *interactive* supercomputer; now, imagine it slicing through terabyte databases in seconds, and that it is at least as easy to use as your PC, or workstation. Clearly, this mind experiment evokes impressions of a very different kind of computing from that which many of us have struggled with since the arrival of massively parallel machines. It suggests more than just a computing capability; it suggests a change in our mental stance with respect to computing and data analysis. However, there is something familiar about this feeling, too. We are reminded of the heady days of Lisp machines (Bawden et al., 1977; Moon, 1985), when supercomputing suddenly suggested more than just floating point computations.

The *read-eval-print* loop at the heart of Lisp programming (Abelson & Sussman, 1996) offers a kind of analysis that is difficult to achieve with the current supercomputing paradigm, which might be called *compile-submit-wait*. The power of immediate interactivity with a computer lies at the heart of tools like Mathematica (Wolfram, 2003), MATLAB (Gilat, 2003), or even BASIC interpreters (Kemeny & Kurtz, 1985), and it stems mostly from the agility of the human mind. As long as one does not have to wait minutes to hours between computational gestures, something amazing happens; one gets problem solving at the speed of human insight.

Think and query, and see the results; then follow that thought with another question as the original problem develops into a cascade of other questions, which are either restated and refined, or perhaps even definitively answered. This *problem→question→better question* cycle is Michel Meyer's problematology (Meyer, 1995) made fast by computing; the speed around this loop is the feature that multiplies insight and enables progress. Unfortunately, these kinds of interactive environments have not been particularly useful with massively parallel computers despite a few interesting experiments (see, for example, the Parallel MATLAB work (Choy & Edelman, 2005)).

One reason this scale of interactivity has not been productively developed for massively parallel computing may have to do with the mismatch of problem sizes that we can think about in an interactive manner vs. the scale of problem that can make good use of a parallel computer. Alternately, it may be that interactive programming models are simply incompatible with large message passing architectures, or maybe it is just hard and we have been too lazy. Whatever the technical obstacles, the economics are particularly daunting when we consider the cost of having a multimillion dollar supercomputer idly waiting on the slow typing and careful analytic thought of a puzzled analyst. On the other hand, maybe this bizarre use of supercomputers is exactly where they are most valuable. Maybe there are data-centric problems where discovery based analysis and data mining fully justify such exorbitant uses of computing.

Here we will consider a massively parallel database machine originally designed for just such analytic searches; a computer that is programmed with database scripts or used directly by means of an interactive SQL *read-eval-print* loop. We will describe this Netezza architecture, the computing problems we used to evaluate the architecture, and our results. We will also share our impressions about the psychological differences we noticed while using this machine in contrast to our experiences with large database servers and other parallel computing environments that we regularly use.

2. The Netezza Architecture

Netezza's architecture (Netezza, 2006), depicted in Figure 1, is a two-tiered system designed to handle very large queries from multiple users. The first tier is a high-performance Linux symmetric multiprocessing host. The host compiles queries received from business information (BI) applications, and generates query execution plans. It then divides a query into a sequence of sub-tasks, or snippets, which can be executed in parallel, and distributes the snippets to the second tier for execution. The host returns the final results to the requesting application thus providing the programming advantages of appearing to be a traditional database server.

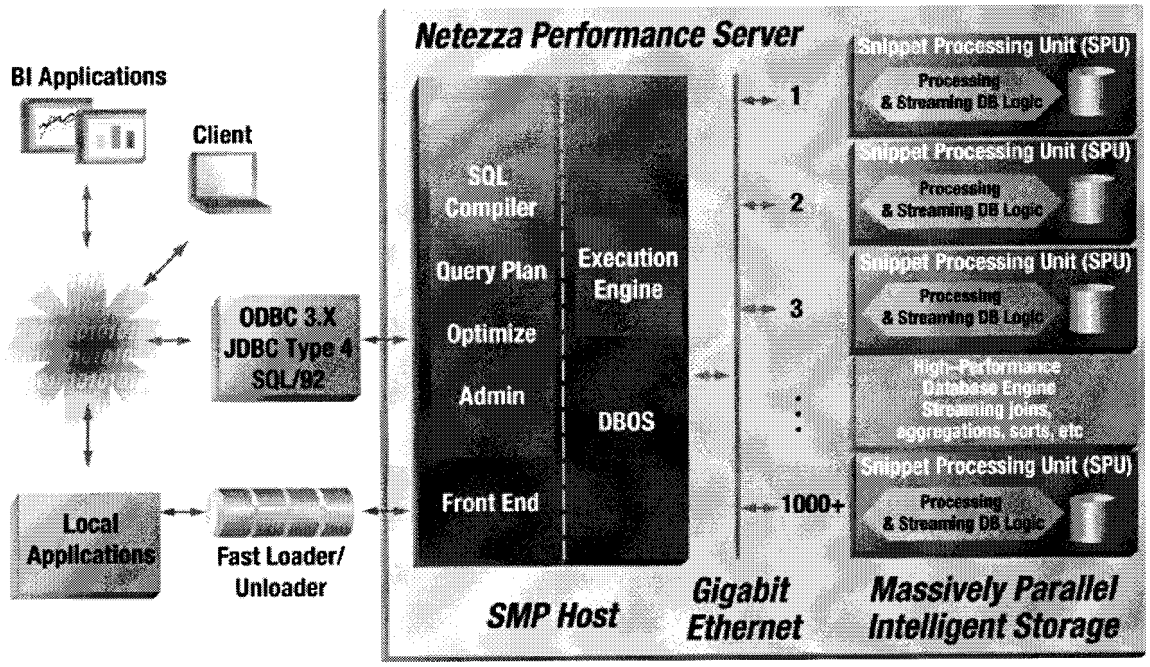

Figure 1. The Netezza Performance Server consists of a closely coupled server with many parallel Snippet Processing Units, each with their own disk and streaming database logic chip to perform fast pattern matching.

The second tier consists of dozens to hundreds or thousands of Snippet Processing Units (SPUs) operating in parallel. Each SPU is an intelligent query processing and storage node, and consists of a powerful commodity processor, dedicated memory, a disk drive and a field-programmable disk controller with hard-wired logic to manage data flows and process queries at the disk level, as depicted in Figure 2. The massively parallel, shared-nothing SPU blades provide the performance advantages of massively parallel processors.

Nearly all query processing is done at the SPU level, with each SPU operating on its portion of the database. All operations that easily lend themselves to parallel processing (including record operations, parsing, filtering, projecting, interlocking and logging) are performed by the SPU nodes, which significantly reduces the amount of data moved within the system. Operations on sets of intermediate results, such as sorts, joins and aggregates, are executed primarily on the SPUs, but can also be done on the host, depending on the processing cost of that operation.

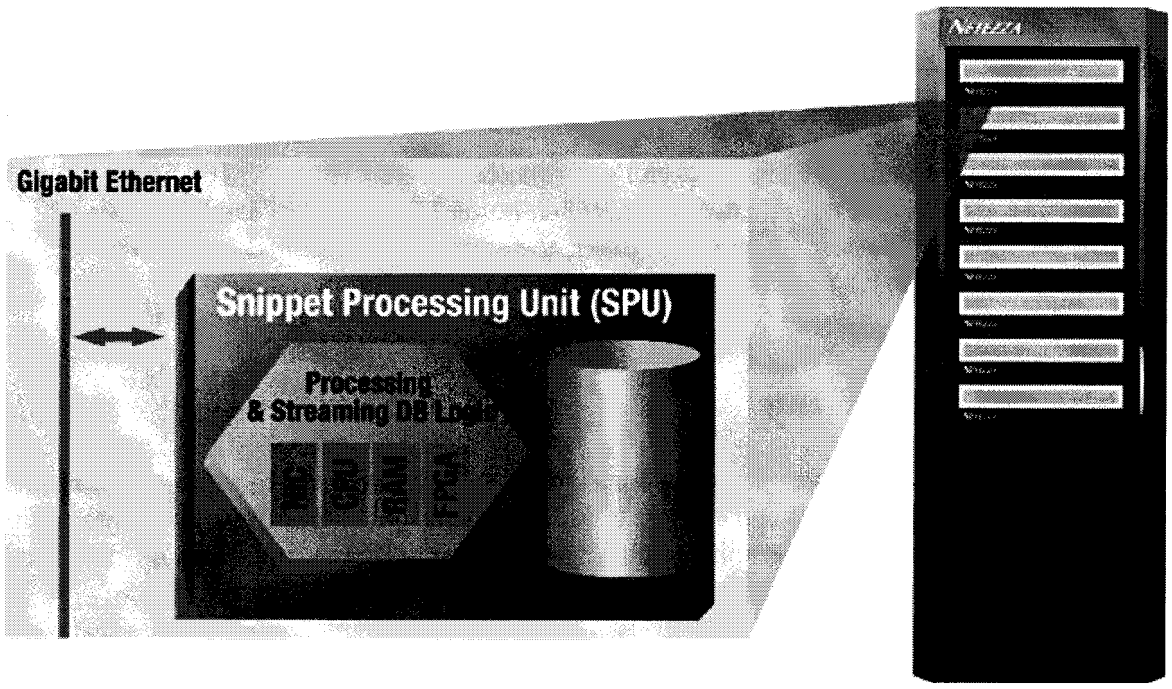

Figure 2. Snippet Processing Units are connected by Gigabit Ethernet to both the host server and to the other SPUs.

Intelligent Query Streaming is performed on each SPU by a Field-Programmable Gate Array (FPGA) chip that functions as the disk controller, but which is also capable of basic processing as data is read from the disk. The system is able to run critical database query functions such as parsing, filtering and projecting at full disk reading speed, while maintaining full ACID (Atomicity, Consistency, Isolation, and Durability) transactional operations of the database. Data flows from disk to memory in a single laminar stream, rather than as a series of disjointed steps that would require materializing partial results.

To achieve high performance, the storage interconnection, which is a bottleneck with traditional systems, is eliminated by directly attaching the disks so that data can stream straight into the FPGA for initial query filtering. Then, to further reduce the workload on the central server, the intermediate query tasks are performed in parallel on the SPUs. The internal network traffic has been reduced by approximately two orders of magnitude by only using the system-wide, gigabit Ethernet network to transmit intermediate results, rather than the entire collection of raw data. As a result, the I/O bus and memory bus on the host computer are only used for assembling final results.

In general-purpose systems, I/O bottlenecks are commonplace as the system is scaled to accommodate complex queries. On the other hand, in this architecture, additional arrays of snippet processing units can be added to the NPS system without impacting I/O performance because query processing involves just a minute fraction of the data traffic associated with

traditional systems, and because storage and processing are tightly coupled into a single unit. The autonomy of the SPUs creates further conditions for a highly scalable system, allowing SPUs to be added without worrying about coordination with other units. As a result, growing data volumes can be managed and accommodated with orderly, predictable investments.

This report covers an evaluation of the Netezza Performance Server 8150 with 108 active Snippet Processing Units, each with its own disk, which holds $1/108^{th}$ of the database. By uniformly distributing the data across all of the disks, the applications can make use up to 4.5 terabytes for one large database or an aggregate combination of different databases.

3. Computational Experiments

The investigation included three very large database problems; each related to important national security issues. The team developed and demonstrated: (1) the largest literature citation graph that has been computed, to date; (2) graph search algorithms for netlist verifications of integrated circuits; and (3) a novel approach to better automatic language translation. Each of these investigations will be briefly described in their own subsections. It is interesting to note that each of these applications ultimately involves graph algorithms, though we did not initially select them with that in mind. In retrospect, this observation may just reflect the coming importance of graph algorithms in general, and the need for architectures that support them.

3.1. Computing literature graphs

Citation graphs are a representation of intellectual history as it develops and is documented in the published literature, for instance papers in the set of scientific journals. These graphs represent publications as nodes and then use directed arcs to represent references to earlier papers, based on the authors' own included reference citations. Citation graphs have played a central role in the field of bibliometrics (the scientific study of the structure, development, and influence of scientific fields), starting with the pioneering work of Price (Price, 1965).

The direct citation graph itself is rather sparsely connected as, on average, each paper only cites two dozen previous papers. The bibliographic research community is also interested in identifying publications that are more weakly coupled. For instance, bibliographic coupling is said to occur when two papers cite one or more common, earlier papers. Computing these graphs inevitably leads to a combinatorial explosion of the number of edges between nodes. Historically, most studies have been done on very small datasets (e.g., hundreds of papers, thousands of citations). The limited size of these studies has not been by choice, but was simply constrained by the available computing resources.

With the recent increases in computing capabilities, we are no longer limited to analysis of small literature graphs. We regularly conduct science and technology policy studies that rely on the generation of maps of very large citation graphs, which we create from one-year slices of the published record (Klavans & Boyack, 2006). These graphs are each based on roughly one million papers containing 25 million citations. Calculation of the bibliographic coupling coefficients (400 million couplings) for one of these graphs is just barely possible with high-end

personal computers. However, we wanted to create a much clearer picture by using more than a single year of publications. On the other hand, we did not want to invest the resources to redevelop the entire system in a language other than SQL, which would be required if we were to use our MPP computers.

Given that constraint, and that the problem size scales as $\mathbf{O}(n^2)$ with the number of citations, calculation of a 20-30 year map was beyond our reach. However, the full problem was believed to be well within the capabilities of the Netezza machine. Thus, we decided to test the Netezza parallel database computer by attempting to calculate the bibliographic coupling graph from our largest collection of citation data.

The full dataset included 16.09M papers containing 388M citations. The ultimate calculation of the bibliographic coupling graph for this set generated over 47 billion couplings, as shown in Table 1. Though the full problem could not be run on our dual-processor Xeon 3.0 workstation using MySQL, smaller problems were completed to provide benchmark comparisons. In particular, we were able to measure the time to compute bibliographic couplings for 4M and 26M citation pairs with the smaller system, which allows the comparison of run time on the base system to the time required on the Netezza computer.

	Small	**Medium**	**Large**
Citation pairs	4 million	26 million	388 million
BC coefficients	68 million	400 million	47 billion
Coefficient storage			2 TB
Time (Xeon)	160 min.	40 hr.	n/a
Time (Netezza)	2.3 min.	0.38 hr.	50 hr.

Table 1. The sizes and timing results for three different bibliographic database computations.

Speedups of 50-100 times were obtained on the small and medium test sets by using the Netezza machine, which is consistent with the use of approximately 100 Netezza Snippet Processing Units (SPU) vs. a Xeon processor.

As shown in the column labeled 'Large', we were able to make use of the size of the Netezza machine (4.5 TB disk storage space, 4.5 TB disk swap), as well as its speed. Our final, bibliographic coupling network is, to our knowledge, the largest citation graph ever calculated. Further, because the Netezza parallel database engine is based on SQL, queries suitable for the base Xeon 3.0 system were executed in parallel, without the need to expose the parallel architecture (with, for instance, message passing) to the code developer, which greatly simplified our development and debugging work.

3.2. Graph search through an integrated circuit netlist

The interconnection networks within integrated circuits are examples of the large scale graphs (more than one million nodes) that we use in the development of parallel graph algorithm libraries. We had access to the netlist, or connections between internal components, for a part

designed by a colleague at Sandia National Laboratories. The device consists of about 3.5M transistors interconnected by approximately 10M conductors (which will be called wires, whether or not they are realized as metallization).

These designs are specified with hardware description languages, which compile to functional circuit descriptions (or macros) specifying elements and interconnections that implement the functions of the desired circuit. These macros are themselves a low-level programming language with input and output arguments and internal variables used for specifying local connections. Each line within the macro specifies a call to another macro, or specifies an instantiation of a physically realizable element, for instance, resistors, diodes, and transistors. For example, consider Figure 3, which shows a typical macro calling out only realizable components: three N-type MOSFET transistors and three P-type MOSFET transistors. This netlist was created by logic design software; however, a netlist can also be assembled by hand, or can even be inferred from X-ray images of actual components.

By whatever means they are constructed, these netlists are a source of large-scale graphs that are very useful for testing graph algorithms; they are, also, associated with very practical reliability and fabrication issues. However, the scale of these graphs is such that humans need computational assistance to work with them. This need was the motivation for exploring how an interactive, massively parallel database computer might help an analyst explore large netlists for anomalies.

Given the short evaluation period during which we tested the Netezza machine, we focused on a single question: can these machines be used to search for isomorphic graphs, and in particular, how efficiently can this computing environment identify all the instances of the sub-circuits specified by the *and2d1* macro shown in Figure 3. Given a relational database representation of a netlist, it is simple to craft an SQL query that will select sets of transistors of the proper PMOS or NMOS types that also have the appropriate connectivity. We faced two issues with regard to this test: how best to create a database version of the netlist information, and how to make queries that could be executed in a reasonable time, for example in just a few seconds.

To load the database, a program was written to transform the macro representations into C code which *executed* the macros such that each execution resulted in either calling lower level macros, or if the macro being called was at the lowest level, i.e., a realizable device, then a line was written to a text file recording the instance number, device type, and the signal names coming through the macro parameter slots. The resulting file was loaded as a parallel database with random, but level, distribution across the individual disks on the 116 Snippet Processing Units.[1]

[1] In a traditional database system, indexing dramatically speeds up join times. The need for indexing is avoided on the Netezza architecture because processing is done at the disk level – i.e., the FPGA operates on the data as it is read from the disk. Thus, indexing times are avoided with this machine. However, proper distribution of tables across the available SPUs is critical to achieve high performance. Tables should be distributed across SPUs using the same fields that would be indexed in a traditional system, thus enabling each processor to run on its portion of the data with little or no need for communication between the SPUs.

```
c and2d1 Cell A1 O    A2 I    Z I     VSS G    VDD P;;
* 5 pins
* 7 nets
* 6 instances
I M1 N  Z  n7 VSS VSS;  L 3.5e-07  W 3.5e-06;
I M2 N  n8 A2 VSS VSS;  L 3.5e-07  W 3.5e-06;
I M3 N  n7 A1  n8 VSS;  L 3.5e-07  W 3.5e-06;
I M4 P  Z  n7 VDD VDD;  L 4.5e-07  W 6.5e-06;
I M5 P  n7 A2 VDD VDD;  L 4.5e-07  W 6.5e-06;
I M6 P  n7 A1 VDD VDD;  L 4.5e-07  W 6.5e-06;
e
```

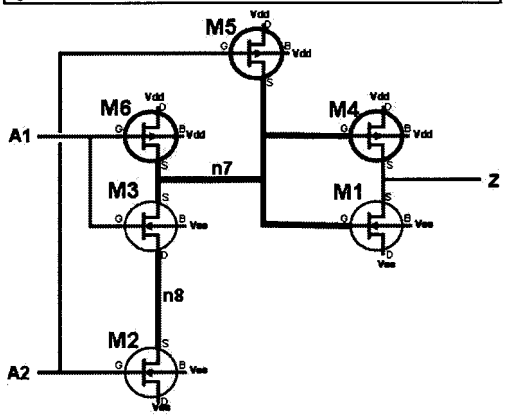

Figure 3. The macro *and2d1*, which specifies a two input AND function with a buffered output Z. The realization requires three n-type and three p-type transistors. Notice that p-type transistors have a larger length (L) and width (W) specification; also, note the two local connection nets (drawn in bold) labelled n7 and n8, which are important clues for identifying isomorphic circuits within the collection of 3.5 million transistors in the original component.

It was quickly determined that a straight forward approach was intractable; that is to say, combinatorial explosions prevent the construction of a final result table by joining the various possibilities for each realizable element in the macro. Even doing the searches for patterns matching the *and2d1* circuit in several sequential steps exhausted the available disk space after several hours of computation.

With our initial enthusiasm and hope for a simple solution crushed, we carefully examined the combinatorial explosions and realized that we would only be able to make progress if we could greatly limit the sizes of the joined tables. Fortunately, a powerful clue presented itself when we recognized that we had special knowledge about the connectivity strictly within a macro. While we could have no idea about the number of terminations for signal names coming into, or leaving, the macro definition, we can know the exact termination counts for those signals that are solely used within the macro (consider the connections n7 and n8 depicted in Figure 3 by the bold lines).

By making use of this specific knowledge, we could create sequences of queries that greatly narrowed the possibilities for each of the transistors that could have been specified by the *and2d1* macro. For example, it is possible to recognize that any transistor that might correspond to M1 within any *and2d1* circuit would have to be an N-type transistor, and that its second slot (corresponding to n7) would have to have degree=5. Further, its third and fourth slot would have to have the same signal name (VSS).

Using these simple constraints the set of 3,515,459 unique instance numbers for the design can be reduced to only 50,040 possibilities for M1, see Table 2. Likewise, M2 must be a type N transistor, where the second slot has a connection of degree 2, and third and fourth slots wired to the same signal (VSS), which reduces to only 57,812 possible instances. Table 2 shows number of possibilities for each individual transistor in the macro *and2d1*.

Possible	Number of Instances
M1	50,400
M2	57,812
M3	27,920
M4	30,952
M5	31,857
M6	31,857

Table 2. The number of transistors that could correspond to each of the six instances in the *and2d1* macro without considering pairwise constraints.

By combining these possibilities, subject to connection constraints within the *and2d1* macro, we find 11,031 instances of six transistor sets consistent with the wiring specified by the *and2d1* macro, see Table 3.

Macro Names	Rows Found
M1M3	16,360
M1M2M3	16,047
M1M2M3M4	13,135
M1M2M3M4M5	11,079
M1M2M3M4M5M6	11,031

Table 3, the reduction in possibilities as further couplings are applied. So, while there were 50,400 possible M1 transistors, the count reduces to 16,360 when the joint constraints for M1 and M3 are considered. Considering the joint constraints among all six transistors, there are 11,031 possible *and2d1* macro instances.

From the execution log, it was possible to discover that the macro *and2d1* actually ran only 9,688 times. The difference seems to be 1,343 sets of transistors that have the same wiring as that specifically called out by the *and2d1* macro, but which were generated as parts of macros other than *and2d1*.

The instance numbers that were not within the range for the actual *and2d1* macro calls were extracted and manually examined. For example, the six transistors shown in Table 4 were identified as an example of a circuit isomorphic to the pattern specified by *and2d1*, even though they were not the result of an *and2d1* macro call.

M1	M2	M3	M4	M5	M6
24156	24171	24170	24155	24169	24168

Table 4, the unique instance numbers for six transistor matching the constraints for the *and2d1* macro; however, these cannot have been synthesized by a macro use because the instance numbers are not sequential.

Clearly, these instance numbers are not from an actual *and2d1* macro call, otherwise the instance numbers would be sequential from 24156 through 24161. By reconstructing the details about the transistor with instance number 24156, one finds this transistor was created by the macro *M_C_U12_N1*.

Note that this identification is exactly an example of searching for a particular circuit pattern and unexpectedly finding more than were anticipated. In this case, one might conclude that the designer, or the circuit generation software, found it easier to emit calls that implement the equivalent of *and2d1* rather than call the macro itself. It is doubtful that any program can ever completely determine that the unexpected sub-circuits follow the intent (and specifications) of a designer. However, this example shows that the computer can be used to greatly limit the number of possibilities that must be manually checked when looking through the circuit for possible unwanted, additional circuits.

Interestingly, while the naïve, brute force SQL search never terminated, the individual queries for determining the possibilities for M1 through M6, and then the various joins from these primary tables all ran in about 10 seconds, each. This speed is sufficient to maintain interactive, analytic attention spans.

3.3. Graph search through an ontology for word sense disambiguation

Most automatic language translation systems make use of the statistical properties of written text to make a Bayesian estimate for a possible translation. However, these algorithms never actually understand the textual meaning itself. An alternate approach, pursued by researchers at New Mexico State University (NMSU), aims for higher quality translations by means of computationally expensive, knowledge intensive reasoning about sentence meanings. Our results suggest that this approach, when combined with a massively parallel computer, is now computationally feasible. The new algorithms should scale linearly to make use of even the largest Netezza machine (600 processors with 27 terabytes of storage). Linear scaling, together with the massively serial nature of streaming document sources, suggests that tens of thousands of processors could be employed for intelligence applications. Interestingly, even better web-searches can also be enabled by this approach, which attempts to actually understand the word senses within the queries and throughout the text being searched.

Word sense disambiguation deals with how to pick out the correct sense, or definition, of an instance of a word given its current context. For example consider the possible confusions associated with uttering the words, "*Bush gave Rice the book.*" We are able to unambiguously understand this phrase by means of constraints around the concept of giving (*someone* gives *something* to *someone*) and external knowledge about current events (by recognizing two public

figures as a possible resolution for the giver and receiver of the book). Word sense disambiguation is a necessary component in a wide range of natural language processing applications including machine translation, information retrieval, and speech synthesis, where it is often necessary to understand the word sense usage in order to make the correct pronunciation.

Martin Cooper (Cooper, 2005) estimates that each word represents, on average, 1.3 distinct concepts. Dictionaries list an average of 2.0 senses per word; however, there is great variability. Ng and Lee (Ng & Lee, 1996) report that on average there are 7.8 senses per noun and 12 per verb for the 191 most ambiguous words. In early work, Pimsleur (Pimsleur, 1957) showed that the naïve approach of just picking the most common sense for each word produces results that are 80% correct, cited in (Ide & Veronis, 1998). Pimsleur sketched out a method that distinguished additional meanings, which was around 90% accurate. An implementation of his ideas was made by Madhu and Lytle (Madhu & Lytle, 1965), which used Bayesian methods and produced about 90% correct disambiguation results. While substantial research has since been conducted, relatively little progress has been made across the intervening five years to improve this accuracy. Moreover, many current studies which report results at 90% or better, involve only a few words, which are also, typically, nouns (Ide & Veronis, 1998).

A number of different methods have been used to address the word sense disambiguation task and these methods fall into two broad categories, statistical and symbolic. Statistical techniques initially centered on supervised disambiguation techniques such as Bayesian classification. For example, Gale (Gale, Church, & Yarowsky, 1992) describes a system based on Bayesian classification that is correct for about 90% of the occurrences of the 6 ambiguous nouns examined. Recently, there has been an interest in unsupervised approaches; for example, Yarowsky (Yarowsky, 1995) describes an algorithm that is 96% accurate.

The symbolic approach described here is based on Katz and Fodor (Katz & Fodor, 1963). In that seminal work, the authors present the rudiments of a theory of disambiguation using selectional restrictions and type hierarchies. There has been a substantial amount of work, both NMSU, and elsewhere, to devise well-constructed algorithms and implementations of this basic idea; for example, see Guthrie *et al.* (Gutherie, Guthrie, Wilks, & Homa, 1991), Cowie *et al* (Cowie, Guthrie, & Guthrie, 1992), Mahesh *et al.* (Mahesh et al., 1997), and Viegas *et al.* (Viegas, Mahesh, & Nirenburg, 1999). This prior work has been extremely promising, but has suffered from a significant problem; the computational time required to process naturally occurring text of any complexity has been too long to be of practical use. This processing complexity motivated the evaluation of the Netezza architecture to see if the approach could be improved to enable its use with realistically applications.

Part of the lexical meaning of the earlier example involving the word *give* is that it requires a human subject. An initial step in having a computer understand the meaning of the sentence is to look up each word in a lexicon. Using this lexicon we would discover that the word *bush* is ambiguous, i.e., we will find that it has multiple entries in the lexicon. It could mean the current president of the United States, a shrub, or a collection of hair. Likewise, the word *Rice* could mean a person named Rice, or rice (the grain), or a university in Houston. *Give* could mean deliver or bequeath, and so on. Thus, just the sentence fragment, *Bush gave Rice*, is ambiguous between 18 different readings (3 x 3 x 2). The word sense disambiguation task is to filter out

those possibilities that don't make semantic sense. For example, if we knew that the agent of *give* (in this case the subject) needed to be a human, we could eliminate the readings where the word *bush* referred to a plant or hair. We perform this task by using an ontology, which is a relationship between textual representations of words and each of their possible meanings, augmented by restrictions relevant to each word sense.

Representing an ontology within a relational database was prototyped using a single Linux desktop PC. We used a MySQL 4.0.22 database server and Python 2.3.4. The size of the database was approximately 50,000 lexical entries per language, and 27,000 root concepts in the ontology, which were queried from python scripts using, for example, predicate functions such as **isaP(***Bush, HUMAN***)**. As knowing Bush is the President, and that President is a public office, which is held by a human, this **isaP()** predicate can determine that Bush is HUMAN. Notice that this traversal through the ontology requires recursions of unpredictable depth, as shown by the bold lines in the code in Figure 3. This recursion is the source of the significant time required to perform word sense disambiguation using ontological methods. For example, using the Xeon processor and environment described above, 6.97 seconds were required to disambiguate the test sentence "Bush gave Rice the Book." Larger blocks of text would, of course take longer; worse, the space requirements explode such that database queries on this workstation were intractable with sentences of reasonable sizes. The Netezza architecture was evaluated to see if these limits could be overcome with a powerful, massively parallel database computer.

```
def isaP(X, Y):
            if X == Y:
                  return true
            else:
                  parents = isa(X)
                  if parents == []:
                        return false
                  for parent in parents:
                        if isaP(parent, Y):
                              return true
                        return false
```

Figure 4. The pseudo code for asking the is-a question.

We started with the MikroKosmos ontology (Mahesh & Nirenburg, 1995), which is a set of assertions arranged in triplets, see Table 5, each addressing a single concept, which may have multiple named slots (with instances like Agent, Destination, or IS-A, etc.), and zero or more constraints associated with that slot and, hence, the word sense tied to that concept. For example, a constraint might require the Agent to be HUMAN, as in the case of a president, or require that the Agent is a PLANT in the case of rice (the grass).

Concept	Slot	Constraint
Give	Agent	Human
Give	Destination	Animal
President	IS-A	Human
Bush	IS-A	Plant

Table 5. A partial view of the ontology showing the relationship slot and the constraint applicable to any resolution for the slot.

To avoid recursions, but at the expense of greatly expanding the database, the ontology can be unfolded by adding new triples for every possible resolution for a constraint. Thus, as the MikroKosmos ontology tells us that OPPONENT, PARTNER, and TEACHER, etc., are all particular instantiations of HUMAN one can add triples like (Give, Agent OPPONENT), (Give, Agent, PARTNER), (Give, Agent, TEACHER), etc., to the database. Notice that TEACHER could be expanded to LECTURER or to PROFESSOR, etc., which indicates how rapidly this kind of unfolding can explode. Nevertheless, limited unfolding does make a difference in the search times. A set of test sentences that required 262 seconds on the Xeon system required only 5.58 seconds on the Netezza Performance Server 8150 with 108 active processing units. For these timing runs, the test sentences were submitted individually, one at a time. However, simultaneous queries could be processed in a pipelined manner while a single pass is made through the database, so bigger blocks of text could be processed at nearly the same speed as individual test sentences.

With help from Netezza, we were able to develop a method for interactively exploring a range of alternative solutions. The Netezza architecture enabled us to quickly build tables on-the-fly to support this interactive way of exploring the problem. Instead of the current supercomputer paradigm of *compile-submit-wait*, we were able to perform our investigations using a more natural problem solving approach. As a result of these investigations, we were able to more compactly encode our triples using a single integer, for example (buy, agent HUMAN) was uniquely coded as 17645217185676, which not only took less space, and hence shortened the time to scan through the entire database, but also simplified the matching computation from string comparisons to a single integer compare, both of which improved performance.

As a final test, we processed the following text fragment:

> *Roche buys Doctor Andreu. The Roche Group acquired the laboratory it was announced. The communication did not specify the amount. Sources placed the amount at 10 million pesetas. The product will be produced under the same name. The product will be commercialized under the same name. Dr. Andreu, whose fame rests on its pills...*

This text was run through a pre-processor, which identified parts of speech, and tagged proper names with semantic categories. Even here, perfection is not found. The pre-processor, for instance, identified (not unreasonably) Doctor Andreu as a person, not an organization.

All the subsequent pre-processing was done by hand while mimicking, as well as possible, the current state of the art automatic processing. Only clause level constraints were examined (though there are constraints in the ontology and lexicon between modifiers (adjectives/adverbs)) and their modified nouns and verbs. Inter-clausal relationships were also not examined, except for relative clauses, where an appropriate identification of the head was assumed (however, the baseline system did evaluate such modifier constraints). Nor were prepositional phrases included that did not carry specific thematic roles in the verbal construction.

The end result of this process was a set of triples representing the possible semantic meanings of the sentence. For example, the first sentence *Roche buys Doctor Andreu* was converted to:

buy agent human
buy theme human.

The six sentences in the test fragment, shown above, were converted to 69 triplets. The Netezza run over this data required a surprisingly short 0.23 seconds per query.

Historically, the incorporation of word sense disambiguation techniques into practical natural language processing applications has been hampered by the significant amount of time required for disambiguation. Therefore, we interpret our good result as identifying an opportunity to greatly improve practical applications of natural language processing. For example, this work suggests that adding a word sense disambiguation module to machine translation would improve translations and only delay processing by a few seconds if the proper machine and architecture were used.

In addition, we find that using this approach could actually improve the quality of word sense disambiguation and other natural language processing related tasks. Having the ontology expanded lays open the possibility of tuning it more directly to handle metonymical[2] and metaphorical[3] readings, which have remained out of reach for machine translation systems and knowledge extraction programs. Having rapid and accurate processing of the ontology makes it feasible to use it in other NLP tasks—parsing disambiguation, reference resolution, etc. — to which it has heretofore not been readily applied. The interactivity and rapidity of processing also makes debugging and improving both ontology and algorithms a much more tractable task as the effect of changes can be evaluated almost instantly.

4. Discussion

The Netezza architecture is the simplest parallel computer we have ever used. Installation was a snap; the rack was rolled onto the floor and connected to power and our network; that was it! Programming the machine is also simple, as the parallel nature of the database is not directly

[2] Metonymy – a figure of speech using the name of one thing for that of another, but related thing, as in "lands belonging to the crown."
[3] Metaphor – a figure of speech where a word literally denoting one thing is used in place of another to suggest a likeness or analogy between the two things.

exposed except with respect to the initial distribution of the data over the nodes when tables are being created. Regular SQL serves as the sole programming language (currently), and hence is exactly as easy to use on the parallel Netezza as on a workstation, or even on a personal computer. Interestingly, much of the performance tuning required for typical database applications is not needed, nor is it even possible with this machine, which is a real advantage because such tuning requires expensive, highly trained database management analysts. The cost of continued ownership for this machine is thus less than would be expected for other data warehousing solutions.

In evaluating our results from just three non-business related applications, it is important to remember that we do not have a long tradition of using database machines to do science applications, so it is unlikely that we stumbled across the best matches between this new architecture and our needs in the few short weeks of our evaluation. The architecture is intriguing; however, it is important to consider the possibility that we could have done the equivalent computations on our existing large-scale machines. For example, the literature citation studies could have been coded in C with MPI, but the SQL approach was very transparent, and was a direct port from workstations with smaller databases.

Interestingly, both the Word Sense Disambiguation project and the Netlist Graph Search project were able to code SQL implementations that failed (horribly) to meet the anticipated performance goals. Both applications eventually achieved superior performance; how that performance was achieved is particularly intriguing. Both applications require algorithms for search through a large graph, and both eventually preprocessed those graphs to compile a critical amount of information into a single integer at each node of the graph so that the searches could be accomplished with very simple and fast matching constraints.

The highly interactive nature of the Netezza architecture made this research possible, but the final result is very likely amenable to implementation on our massively parallel message passing machines. Further, as long as we are working with small enough graphs, and they are all going to be much smaller when the encoded information is condensed to these integer representations, the distributed memory, message passing machines may be much faster than implementations on database architectures, which inherently make use of disks. In considering the history of these two applications, one might wonder if an interactive database machine would best be used to develop and prototype production algorithms that could then be rewritten for even greater performance on MPP message passing machines.

In our opinion, the evaluation was a huge success. At one of the earliest planning meetings, we hoped to find one or two of the applications that would enable an important breakthrough, such that something previously impossible would be readily accessible. It is a bit early to tell, but it seems possible that the natural language application (Word Sense Disambiguation) meets this criterion. Certainly, the achieved speedup is significant, and enables computations that were not previously possible. Further, the improvement in quality is such that new uses can already be seen.

With these results, it would seem possible to immediately begin applying the technique to projects of national interest. One possible project might be a high throughput, high quality translation system for streaming text from around the world, which could be important to national security and/or homeland defense. Another timely project might combine these results with our work on early syndromic detection systems for natural or terrorist-induced epidemics. For example, Chapman, et al. (Chapman et al., 2005) have shown that NLP techniques can be used to winnow through 'chief complaint' reports at emergency rooms to produce early indicators of certain kinds of infections. The techniques and massively parallel computing enabled by our Netezza evaluation could be applied to very large collections of reports from around the US, or even around the world, to detect early indications of infections. This idea is particularly timely given the concerns about the possibility of millions of deaths from avian flu this year or in the near future.

5. References

Abelson, H., & Sussman, G. J. (1996). *Structure and interpretation of computer programs* (2nd ed.): The MIT Press.

Bawden, A., Greenblatt, R., Halloway, J., Knight, T., Moon, D., & Weinreb, D. (1977). *LISP Machine Progress Report* (No. A.I. Memo 444). Cambridge, MA: MIT AI Lab.

Chapman, W. W., Christensen, L. M., Wagner, M. M., Haug, P. J., Ivoanov, O., Dowling, J. N., et al. (2005). Classifying free-text triage chief complaints into syndromic categories with natural language processing. *Artificial Intelligence in Medicine, 33*(1), 31-40.

Choy, R., & Edelman, A. (2005). Parallel MATLAB: Doing it right. *Proceedings of the IEEE, 93*(2), 331-341.

Cooper, M. (2005). A mathematical model of historical semantics and the grouping of word meanings into concepts. *Computational Linguistics, 31*, 227-248.

Cowie, J., Guthrie, J., & Guthrie, L. (1992). *Lexical disambiguation using simulated annealing.* Paper presented at the 14th International Conference on Computational Linguistics, *COLING'92*, Nantes, France.

Gale, W. A., Church, K. W., & Yarowsky, D. (1992). A method for disambiguating words senses in a large corpus. *Computers and Humanities, 26*, 415-439.

Gilat, A. (2003). *MATLAB: an introduction with applications*: Wiley.

Gutherie, J. A., Guthrie, L., Wilks, Y., & Homa, A. (1991). *Subject-dependent co-occurrence and word sense disambiguation.* Paper presented at the 29th Annual Meeting of the Association for Computational Linguistics, Berkeley, California.

Ide, N., & Veronis, J. (1998). Word sense disambiguation: The state of the art. *Computational Linguistics, 24*(1), 1-40.

Katz, J. J., & Fodor, J. A. (1963). The structure of a semantic theory. *Language, 39*, 170-210.

Kemeny, J. G., & Kurtz, T. E. (1985). *Back to BASIC: the history, corruption, and future of the language*: Addison-Wesley.

Klavans, R., & Boyack, K. W. (2006). Quantitative evaluation of large maps of science. *Scientometrics*, in press.

Madhu, S., & Lytle, D. W. (1965). A figure of merit technique for the resolution of non-grammatical ambiguity. *Mechanical translation, 8*(2), 9-13.

Mahesh, K., & Nirenburg, S. (1995, Aug. 19-20, 1995). *A situated Ontology for Practical NLP.* Paper presented at the Workshop on Basic Ontological Issues in Knowledge Sahring, International Joint Conference on Artificial Intelligence (IJCAI-95), Montreal, Canada.

Mahesh, K., Nirenburg, S., Beale, S., Viegas, E., Raskin, V., & Onyshkevych, B. (1997). *Word sense disambiguation: Why statistics when we have these numbers?* Paper presented at the 7th International Conference on Theoretical and Methodological Issues in Machine Translation, Santa Fe, New Mexico.

Meyer, M. (1995). *Of Problematology: Philosophy, science, and language* (D. Jamison, Trans.): University of Chicago Press.

Moon, D. A. (1985, July 1985). *Architecture of the Symbolics 3600.* Paper presented at the Proceedings of the 12th Annual Symposium on Computer Architecture.

Netezza. (2006). Figures and descriptions used with permission, Netezza, Netezza Inc., (www.netezza.com).

Ng, H. T., & Lee, H. B. (1996). *Integrating multiple knowledge sources to disambiguate word sense: An examplar-based approach.* Paper presented at the 34th Annual Meeting of the Association for Computational Linguistics, University of California, Santa Cruz, California.

Pimsleur, P. (1957). Semantic frequency counts. *Machine Translation, 4*(1-2), 11-13.

Price, D. J. D. (1965). Networks of scientific papers. *Science, 149*(3683), 510-515.

Viegas, E., Mahesh, K., & Nirenburg, S. (1999). Semantics in action. In P. Saint-Dizier (Ed.), *Pedicative forms in natural language and lexical knowledge bases*: Dordrecht: Kluwer Academic Press.

Wolfram, S. (2003). *The mathematica book* (Fifth Edition ed.): Wolfram Media.

Yarowsky, D. (1995). *Unsupervised word sense disambiguation rivaling supervised methods.* Paper presented at the 33rd Annual Meeting of the Association for Computational Linguistics, Cambridge, Massachusetts.

Sandia National Laboratories