

**TECHNICAL REPORT**

**Exploring the Use of Main Memory Database (MMDB) Technology for the Analysis  
of Gene Expression Microarray Data**

*Prepared by:*

**Nicholas Carriero, PhD<sup>1</sup>**

**Michael V. Osier, PhD<sup>2</sup>**

**Kei-Hoi Cheung, PhD<sup>2,3</sup>**

**Peter Masiar, MS<sup>1</sup>**

**Perry L. Miller, MD, PhD<sup>2,4</sup>**

**Kevin White, PhD<sup>3</sup>**

**Martin Schultz, PhD<sup>1</sup>**

<sup>1</sup>Department of Computer Science, <sup>2</sup>Center for Medical Informatics, <sup>3</sup>Department of  
Genetics, <sup>4</sup>Department of Molecular, Cellular and Developmental Biology

Yale University, New Haven, CT

**April, 2004**

## **Abstract**

The high volume and complexity of microarray data has created both opportunities and challenges for bioinformaticians or computational biologists as the genome-wide analysis of expression data requires advanced database, computational, and statistical approaches. This technical report focuses on the database aspect of microarray informatics research. While relational technology predominates as the platform for current microarray database systems, it has limitations when it comes to large-scale and complex analysis of gene expression data. This type of analysis cannot be handled naturally by relational queries. To augment the relational technology, we describe a lightweight but high-performance approach that allows complex microarray data analysis to be efficiently integrated with a main memory database (MMDB) approach, built upon the aggregated main memory of a cluster of computers. To compare the MMDB approach with the relational approach, we have benchmarked the performance of several test queries. The results suggest that the MMDB approach has the potential to achieve a significant speedup for many analytic retrieval queries.

## 1. Introduction

This report describes a project that is exploring the relative advantages of using a traditional relational database management system (RDBMS) approach compared to a main memory database (MMDB) approach for the analysis of gene expression microarray data. The report describes initial tests that demonstrate the potential for quite dramatic speedup in the time required to perform certain types of analysis using the MMDB approach.

This work is part of a broader project that is exploring the use of high performance computation (HPC) in biomedicine. A great deal of work involving biomedical HPC has focused on the use of parallel computation that allows compute-intensive applications to be executed in parallel, for example on special-purpose parallel machines or on clusters of workstations.

MMDB technology represents a different and complementary form of HPC that allows computations that might otherwise be executed within a conventional database management system to be executed in main memory, thereby avoiding much of the overhead imposed by disk access that is implicit in conventional database systems. The use of MMDB technology is becoming increasingly feasible as the cost of main memory plummets and as an increasingly large amount of such memory is supported by individual workstations. MMDB technology can also be combined with parallel computation in parallel main memory database (PMMDB) applications. For the present research, we used a system called PAMM, which supports PMMDB.

MMDB approaches can be particularly valuable when certain types of analyses need to be performed on data within a database, but when those analyses involve queries that are not an ideal match to the types of queries for which a conventional database is designed. This is particularly true, for example, if a computation involves a complex analysis that iterates over large tables of data, as is often required for the analysis of microarray data.

Experimental microarray data are a challenge to archive and analyze. On the one hand, some attributes of the data (the who, what, where, when and why of the experiments) lend themselves naturally to the organizational structures and retrieval methods of traditional relational databases. On the other hand, the “core” data (microarray spot readings) are large tabular sets of numeric values. The “natural” queries over these data are often more easily conceived (and expressed) in terms of mathematical operations on vectors and matrices, rather than in terms of operations expressed in SQL on the elements of a relational database.

## 2. Background

### 2.1 Microarray Databases

The growing use of DNA microarrays in biomedical research has given a major impetus to bioinformatics research, and several gene expression databases have been developed in the past several years. While some of these databases (e.g., RAD [1], GeneX [2], LAD [3], YMD [4], SMD [5]) are intended for use by individual laboratories, centers, or institutions with specific needs, others (e.g., ArrayExpress [6] and GEO [7])

are designed to serve as public repositories of microarray data in a standardized format (MIAME [8]). Despite their differences, most (if not all) of these databases are implemented using RDBMSs such as Oracle and Sybase. While these database systems provide a flexible data management and querying capability, they have a potential “impedance mismatch” when performing large-scale and complex analyses that are often required for interpreting gene expression patterns. Most of these analyses cannot be most naturally implemented by relational queries. They have often been carried out by external programs, written in procedural/statistical programming languages such as C and R, on large datasets extracted from the database systems.

## 2.2. Main Memory Databases

The concept of main-memory databases has been revisited periodically over the past decade or two (see for example [9-11]). Much of the previous work was performed when main memory was not as inexpensive as it is now, and not as available in large quantities on lower-end workstations. Also, the focus of previous work has often been on preserving RDBMS functionality in a main memory environment. In contrast:

- 1) We are not aiming for a full-fledged MM DBMS (even without transaction support), but a focused approach tailored to the needs of a particular set of tables.
- 2) While we want to simplify integration with RDBMS, we are not wedded to relational model *per se*. Indeed, we see providing alternate programmatic interfaces to the data as potentially enhancing simplicity, expressivity, flexibility and efficiency.
- 3) We are working at a time when memory prices justify extravagant use of memory.
- 4) We are working at a time when 64-bit address spaces are becoming common, making for inexpensive "memoryful" systems, that can in turn be aggregated to form clusters with even larger total main-memory capacity.

A previous deployment of PAMM, the PMMDB approach used in the present work, is described in [12].

## 3. YMD and the Challenges of Analyzing Microarray Data

Yale Microarray Database (YMD) is an institutional database designed to help support many different microarray researchers at Yale, including collaborators at other institutions [4]. YMD currently provides a variety of functionality, including certain query and analysis capabilities for microarray data sets using YMD’s RDBMS (Oracle). A Web interface allows users to access and query this Oracle database. In YMD, the primary data involves a series of hybridizations, arrays hybridized with RNA samples corresponding to different experimental conditions. Each hybridization is linked to 1) annotation data describing the samples studied, and 2) raw data files including the array spot image (in TIF format) and the scanned image data file (in GenePix Results (GPR) File format). For the analyses described in this report, YMD also stores the data from selected GPR files in a single large table, the columns of this table correspond to the GPR file column headers. The data for one hybridization are appended to those for another hybridization. This table currently has over 6 million rows and its size will continue to

increase. The same data modeling approach was used by other microarray databases (e.g., SMD).

Within YMD it is straightforward to handle retrieval of microarray data sets based on experiment attributes contained in the annotation of each hybridization. It is not straightforward, however, to handle retrieval of data sets (across multiple hybridizations) based on spot properties. A query of this nature might have the following form: “Retrieve spot data from a set of hybridizations for all spots that have a specified property for one or more of a specified set of hybridizations.” To answer such a query in a reasonable amount of time from a traditional RDBMS is challenging and typically requires using some combination of indexing and precomputation.

Indexing and precomputation, however, are not panaceas and come at a cost. First, they add to the over-all management burden of the database application and to the time needed to add new data to the database. Second, they have the potential to induce “sclerosis” since efficient interaction with the data within the database tends to be on the database’s terms, dictated by the indices and precomputed values, and not on the user’s terms, dictated by the scientific questions of interest. Third, precomputation increases the “distance” between the raw data and the end user, making it harder to interpret (and debug) unexpected results.

The fact that spot data are stored within one large table raises some scaling issues as well. It is anticipated that both the number of hybridizations, as well as the amount of data per hybridization, will be increasing dramatically in the foreseeable future. This will only compound the cost of indexing and precomputation, and has the potential to severely stress the overall RDBMS.

In summary, where the RDBMS technology is a good fit, it works well, but there are areas of suboptimal fit in processing microarray data, and there is also a looming issue of the need to handle much larger numbers of larger microarray data sets. We believe that the areas of suboptimal fit can be addressed by integrating the base RDBMS with a system designed to work efficiently with the spot data directly.

#### **4. Deployment of MMDB in YMD Setting**

Since our goal is to provide a system to address the areas of suboptimal fit, but not to eliminate the RDBMS, we also need to simplify integration of the MMDB with the RDBMS. We utilized PAMM, a PMMDB system developed at the Yale Department of Computer Science and Scientific Computing Associates, which provides support for moving tables from an RDBMS to a MMDB and for manipulating the data in the MMDB environment. PAMM accepts “CREATE TABLE” statements describing those tables that will be extracted from the main RDBMS and loaded into the MMDB. PAMM then automatically generates:

- 1) a utility to convert extracted tables from text form to the internal binary form of the MMDB system,
- 2) a utility to convert the binary form to text form,
- 3) a module of accessor functions to access data in the tables in “true” order and in “sorted” order,

- 4) a utility to query a table via either logical ANDs or ORs involving ranges of column values,
- 5) a utility to generate some statistics about the contents of a table,
- 6) for testing purposes, a utility to populate a table of arbitrary size with random data.

For testing purposes, the resulting utilities are typically invoked from a command line or shell script. They serve as data exchange components, simple query front ends, and testing interfaces.

The tabular data extracted from the RDBMS is converted via one of the automatically generated utilities into a file containing the tabular data stored in the binary format used by the MMDB system. The binary format structures the table data row by row and includes indexing information that allows the information in a column to be accessed in sorted order. These binary files can be mapped directly into the address space of a program and so support extremely efficient data loading. This conversion need only be done once and takes on the order of seconds per hybridization.

An API is provided that, in conjunction with the accessor functions, offers a programmatic interface. In addition to the accessor functions, routines are provided for retrieval and for sorting intermediate results. The API also includes routines to load tables from files containing data stored in the system's binary format. Using these routines, a developer can work with multiple instances of a given table, loading them in as needed. At the moment, the interface is targeted to the C programming language, but nothing in it is intrinsic to C, and so it would be straightforward to target to other development environments. To make the approach more concrete, Figure 1 shows an example program that uses the programmatic API built to manipulate GPR data.

```
// Loop over all the hybridizations
for (hyb = 0; hyb < numHybs; ++hyb) {
    if (hybs[hyb].tPtr) {
        // If hybs[hyb].tPtr is not zero, then the hyb should be analyzed.

        GprDataTable      *dataPtr = (GprDataTable *)hybs[hyb].tPtr;
        int                hiVal, hiX, loVal, loX, i;

        // loVal and hiVal should specify a range of values for Flag.
        loVal = 0;
        // Because we access via the sorted permutation, the last entry
        // is the greatest.
        hiVal = GprDataFlagSortedAccessor(dataPtr, dataPtr->numRows-1);

        // Use API to find indices bracketing the range.
        if (!lookupRangeInCol(dataPtr->Flag, dataPtr->numRows,
                               &loVal, &hiVal, &loX, &hiX))

            // If we found something, set a bit indicating which
            // records are in the range.
            for (i = loX; i < hiX; ++i)
                bitVector[hyb][dataPtr->Flag->sortedPermutation[i]] |= FlagBit;
    }
}
```

**Figure 1:** To illustrate how PAMM allows processing of its data, this figure shows C language code that utilizes PAMM's automatically generated API, including predefined table types (e.g., GprDataTable), and accessor functions (e.g., GprDataFlagSortedAccessor). This code looks for all spots in a set of hybridizations whose Flag value is greater than or equal to zero.

Although the system described lays the foundation for the automatic integration of the MMDB approach with YMD, we are currently developing the approach in a standalone test environment. The MMDB can be loaded with data 1) from the RDBMS or 2) from the “raw” GPR files (the output of the GENEPIX image analysis software). We currently work with the latter. It would be quite straightforward to link the MMDB approach to the YMD RDBMS. We envision that this process will ultimately be automated and perhaps triggered whenever a new hybridization is added to the main RDBMS.

## 5. Performance Tests of the MMDB vs. RDBMS Approach

To assess the overall feasibility of the MMDB approach, and to compare it to the YMD RDBMS approach, we ran a set of tests.

1. One test addressed the question of whether the MMDB approach could free us from the need for pre-computation, discussed above.
2. The second test used three representative queries of the sort that YMD’s query interface is designed to support.

For the tests described in this report, PAMM ran on one CPU of a parallel cluster, with a 2.4 GHz Xeon processor and 2 GB of main memory, running the Linux operating system. YMD currently runs on an IBM model 7025-F50 machine with dual PowerPC\_604 332 MHz processors and 1 GB of main memory, running the AIX 4.3.3 operating system and Oracle version 8.1.7.4.

### 5.1 Exploring the Need for Precomputation

$$rc_i \leftarrow \begin{cases} \frac{ExperimentForeground_i - ExperimentBackground_i}{referenceForeground_i - referenceBackground_i} \\ \text{or} \\ \text{Null if } (ExperimentForeground_i \leq ExperimentBackground_i) \\ \text{or} \\ (referenceForeground_i \leq referenceBackground_i) \end{cases}$$

$$normFactor \leftarrow \frac{1}{10^{\sum \log_{10} rc_i}} \quad \forall i \therefore 1 \leq rc_i \leq 10 \text{ and } spot_i \text{ is "good"}$$

$$ratioCorrect_i \leftarrow rc_i * normFactor$$

**Figure 2:** This figure shows the logic by which the value of ratio correct is calculated for each spot i of a hybridization. First a value  $rc_i$  is derived from four values associated with each spot. Then a normalization factor for the hybridization as a whole is calculated based on all the  $rc_i$  values. Finally for each spot i, a ratio correct value is computed by multiplying  $rc_i$  by the normalization factor. As described in the text, these values are precomputing in YMD and computed on the fly in the MMDB system.

One of the quantities of interest when working with microarray data is the “ratio correct” value (a normalized ratio used in YMD). Figure 2 outlines the logic involved in calculating this value. One ratio correct value is computed for each spot of a

hybridization. Recognizing the importance of these values for other queries, and considering the cost of computing these values on demand within the traditional RDBMS setting, YMD currently precomputes these values and stores them with each hybridization.

Our first test compared a batch extraction of these precomputed values, for “good” spots, from the YMD to a batch extraction of the same ratio correct values *computed on the fly* from the MMDB. 91 hybridizations were used. Each hybridization presents data for ~20,000 spots. The overall size of a hybridization in ASCII format is ~4.7 MB.

Table 1 shows the results of this analysis. In discussing these results, we should note that both systems have some sensitivity to caching in one form or another, so the first run of a repeated set of runs tends to take somewhat longer (to pull disk data into memory caches). As a result, we list the first run time and the subsequent run times separately. For the MMDB system, the cache was explicitly flushed before each “first” run. For YMD, each “first” run was performed on a different day to allow plenty of time for the cache to be purged in the natural course of database operation. Each entry in the table lists the average time for 10 runs, followed by the range of those times in parentheses.

These figures show a speedup using MMDB of ~17x for the first run and ~100x for subsequent runs. Thus, even though the MMDB system was doing a significant computation on the fly that was precomputed with the RDBMS, it was still much faster. While one could fruitfully explore these numbers in much greater detail, for the purposes of this initial study it is sufficient to note that they support the hypothesis that moving to main memory obviates the need for at least some precomputations and validates the general intuition that for this kind of computation, simple memory-resident data structures can be decidedly more efficient.

	<u>YMD</u>	<u>MMDB</u>
First run	733 (670 - 823)	43.3 (37.1 - 50.8)
Subsequent runs	500 (498 - 502)	4.6 (4.5 - 4.6)

**Table 1:** Each entry shows the average time (in seconds) of 10 runs followed by the range of times, as described in the text.

## 5.2 Comparing Three Representative Queries

The second set of tests uses the type of query provided by YMD’s Web interface. Using this interface, users can specify queries as follows.

1. First the user specifies a Boolean set of criteria to be applied to each spot of each hybridization, in a specified set of target hybridizations. Using these criteria, YMD creates a list of “eligible” spots. A spot is eligible if it meets the criteria in *any* of the specified hybridizations.
2. YMD then produces the following output for each hybridization, for each eligible spot (if the hybridization includes the spot):
  - a) the ratio correct value,

b) “1” if the spot does not meet the criteria for any reason *except* conditions based on flag or filter\_flag attributes, or “NULL” otherwise.

Three representative queries were used in the tests.

1. Query 1 A spot is eligible if the spot's filter\_flag attribute is "good", and the spot's flag attribute is "not bad", and the spot's DNA ID attribute is not “blank.”
2. Query 2 A spot is eligible if the criteria of Query 1 apply, and the spot's ratio\_correct value  $\geq 1.3$ , and the spot's ref\_correct  $\geq 750$ .
3. Query 3 A spot is eligible using the same criteria as Query 2, but  $\log_2$ (ratio correct) is output rather than ratio correct.

We ran these three queries for 10 hybridizations, because YMD’s Web interface currently allows a maximum of ten hybridizations to be analyzed in one query (for efficiency purposes). As with the previous test, the MMDB system computes values like ratio correct and ref correct (an intermediate value used during the ratio correct computation) on the fly, while YMD has these saved as precomputed values. The MMDB trials were run from the command line. (We have not implemented an MMDB interface flexible enough to accept a general query, so the test query logic is encoded in C). The YMD trials were run from the YMD Web interface.

Table 2 shows the results of these tests. Here again, because of the effects of memory caching, the first runs for both approaches were longer than subsequent runs, and are listed separately. Each entry in the table lists the average time for five YMD-first runs and the average time for 10 runs for the other entries, followed by the range of times for the runs. These figures show a speedup of ~90-140x for the first run and a speedup of ~200-300x for subsequent runs.

	YMD		MMDB	
	<u>First</u>	<u>Subsequent</u>	<u>First</u>	<u>Subsequent</u>
Query 1	402 (377 - 416)	372 (364 - 382)	2.9 (2.8 - 2.9)	1.2 (1.1 - 1.3)
Query 2	203 (126 - 260)	102 (99 - 105)	2.3 (2.3 - 2.4)	0.50 (0.49 - 0.55)
Query 3	205 (181 - 297)	160 (155 - 170)	2.3 (2.3 - 2.5)	0.52 (0.50 - 0.63)

**Table 2:** Each entry shows the average time (in seconds) of 5-10 runs followed by the range of times, as described in the text.

### 5.3 Interpreting These Results

The goal of the current project is to establish the general characteristics of performance of the MMDB in a deployed setting. The results of these tests strongly suggest that the MMDB approach has the potential to dramatically improve performance in certain types of microarray analyses compared to an RDBMS. The results derived for MMDB vs. RDBMS, however, cannot be directly compared for several reasons, including the following.

1. The hardware used for the two systems was different. (It is worth pointing out, however, that this is simply a reflection of the resources at hand. The MMDB test platform is not particularly exotic. If we were to bring the MMDB online today, the hardware used would be similar, if not identical, to the test hardware. So from that perspective, the comparative times are meaningful.)
2. In the two approaches, the analysis was invoked in a different fashion. The YMD analyses were invoked over the Internet using YMD's Web interface, whereas the MMDB analyses were invoked via a command line interface.
3. The performance figures of YMD are potentially influenced by competing use of the RDBMS.
4. The format of the output produced by the two approaches were also different. The MMDB format results in about twice as many bytes. (We did, however, compare the data values produced to make sure they were the same. As discussed later, this validation exercise was valuable in its own right as it helped uncover some bugs in both systems and ambiguities in the overall specifications.)

Had the test performance timings comparing the two approaches been close, then these factors would clearly be significant considerations. Because the results using MMDB were so clearly superior to those using RDBMS, we feel that these factors do not influence the overall implications of the results. What we see here justifies the assertion that this approach is "sufficiently" fast in the sense that the query processing per se is no longer the rate limiting step in the user's interaction with YMD via the query page. Further, the architecture of the MMDB and the hybridization-by-hybridization nature of the data sets suggest that a similar level of performance will be achievable via parallelization as the data set size and number scale.

## **6. Discussion**

### **6.1 MMDB vs. RDBMS for Use with Microarray Data**

When used for complex spot retrievals as described above, YMD shares certain characteristics with many other large-scale data mining problems, including 1) large data sets, 2) a need for flexible and fluid access to data, often "against the grain" (inconsistent with any pre-existing indexing or precomputation scheme), and 3) no requirement for basic RDBMS machinery such as transactions, locking, logging (data mining is typically read-only in character).

These characteristics suggest that much of an RDBMS's functionality is essentially "deadweight" for these problems. A good portion of what remains in an RDBMS is there to efficiently manage access to a large, disk-resident, data set via a small main memory: complex caching and query massaging subsystems designed to keep disk access down and data cache hit rates high.

In contrast MMDB technology will increasingly allow us to access a large data set via a very large memory. Recent hardware developments have had a major impact on what is feasible for a large memory. As of July, 2003, the price for semiconductor

memory has dropped to ~\$0.50/MB. System boards can now easily accommodate several gigabytes for workaday machines. In addition, the advent of 64-bit CPUs has led to the introduction of server-class machines with 8-16GB of memory at workstation prices. Thus a small cluster of machines can provide, in aggregate, ~100 GB of main memory at a cost that is likely to fit comfortably within the computing budget of a department or a good size laboratory.

With 100 GB of main memory, microarray spot data could be memory resident in its entirety (or, at worst, loaded, overlay fashion, from each microarray data set stored in a binary format that may be directly mapped into memory). If this is done, then one is no longer concerned about efficient access to disk-based data, and so a traditional RDBMS adds very little.

## **6.2 Using MMDB vs. Hand-Coding the Logic**

One alternative to using either an RDBMS or an MMDB approach is to extract the microarray spot data and analyze that data with a hand-coded analysis routine. PAMM's PMMDB approach provides a number of potentially useful capabilities compared to hand-coding the logic. It preserves the naming of tables and attributes so that if data is extracted from a RDBMS, all the names are automatically the same. It automates the transfer of data from an RDBMS to MMDB. It provides an automatically generated library of routines that access and analyze the data in a variety of potentially useful ways. It also automatically builds indices of the data to facilitate that access and analysis. In addition, PAMM is designed to support parallel MMDB capabilities, so that as increasingly large amounts of data are analyzed, one can migrate natural to a PMMDB approach where different parts of the analysis (e.g., the analysis of different hybridizations) are run simultaneously on multiple processors.

## **7. Summary**

It is not particularly surprising that a “one-size fits all” approach is not altogether appropriate for microarray data analysis given the strikingly diverse character of the types of retrieval queries of interest. We have demonstrated here that it is feasible to augment the RDBMS with a lightweight, high performance system for the more analytic retrieval queries. In addition to enhancing performance, this system has the potential to simplify the RDBMS by reducing or eliminating the need for certain indices and precomputed values.

In fact, although we have yet to integrate the MMDB operationally with the RDBMS, the system has nonetheless already begun to prove its worth. Our tests have demonstrated that if nothing else, the MMDB is sufficiently capable that it can serve as a validation tool. With relatively little coding effort and extremely fast execution times, we are able to generate full output for various test queries. Comparing this output with that from the RDBMS has allowed us to validate the latter. In the process, we have identified ambiguities in specifications and uncovered bugs in both systems.

Finally, while we have focused on microarray data sets, we believe that the method described is applicable more broadly, especially to other “mixed mode” settings in bioinformatics.

## Acknowledgements

This research is supported in part by NIH grants U24 DK58776 and K25 HG02378, by NIH grants T15 LM07056 and P20 LM07253 from the National Library of Medicine, and by NSF grant DBI-0135442.

## References

1. Stoeckert C, Pizarro A, Manduchi E, Gibson M., Brunk B, Crabtree J, Schug S, Shen-Orr S, Overton GC. A relational schema for both array-based and SAGE gene expression experiments. *Bioinformatics* 17(4): 300-308, 2001.
2. H. Mangalam, J. Stewart, J. Zhou, K. Schlauch, M. Waugh, G. Chen, A. D. Farmer, G. Colello, and J. W. Weller. GeneX: An Open Source gene expression database and integrated tool set. *IBM Systems Journal* 40(2): 552-69, 2001.
3. Killion PJ, Sherlock G, Iyer VR. The Longhorn Array Database (LAD): An Open-Source, MIAME compliant implementation of the Stanford Microarray Database (SMD). *BMC Bioinformatics* 4(1):32, 2003.
4. Cheung KH, White K, Hager J, Gerstein M, Reinke V, Nelson K, Masiar P, Srivastava R, Li Y, Li J, Zhao H, Li J, Allison DB, Snyder M, Miller P, Williams K. YMD: a microarray database for large-scale gene expression analysis. *Proc AMIA Symp.* 140-4, 2002.
5. Gollub J, Ball CA, Binkley G, Demeter J, Finkelstein DB, Hebert JM, Hernandez-Boussard T, Jin H, Kaloper M, Matese JC, Schroeder M, Brown PO, Botstein D, Sherlock G. The Stanford Microarray Database: data access and quality assessment tools. *Nucleic Acids Res.* 31(1):94-6, 2003.
6. Brazma A, Parkinson H, Sarkans U, Shojatalab M, Vilo J, Abeygunawardena N, Holloway E, Kapushesky M, Kemmeren P, Lara GG, Oezcimen A, Rocca-Serra P, Sansone SA. ArrayExpress--a public repository for microarray gene expression data at the EBI. *Nucleic Acids Res.* 31(1):68-71, 2003.
7. Edgar R, Domrachev M, Lash AE. Gene Expression Omnibus: NCBI gene expression and hybridization array data repository. *Nucleic Acids Res.* 30(1):207-10, 2002.
8. Brazma A, Hingamp P, Quackenbush J, Sherlock G, Spellman P, Stoeckert C, Aach J, Ansorge W, Ball CA, Causton HC, Gaasterland T, Glenisson P, Holstege FC, Kim IF, Markowitz V, Matese JC, Parkinson H, Robinson A, Sarkans U, Schulze-Kremer S, Stewart J, Taylor R, Vilo J, Vingron M. Minimum information about a microarray experiment (MIAME)-toward standards for microarray data. *Nat Genet.* 29(4):373, 2001.
9. DeWitt DJ, Katz RH, Olken F., Shapiro LD, Stonebraker M, Wood DA. Implementation techniques for main memory database systems. *SIGMOD Conference* 1984: 1-8.
10. Eich MH. MARS: The design of a main memory database machine. *IWDM* 1987: 325-338.
11. Boncz PA. Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications. Ph.D. Thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, May 2002.

12. Price RD, Bandy VA, Carriero NJ, An Alternative for Very Large Databases That Saves Money and Development Time. Technology Review Journal, Fall/Winter 2002: 61-78.