ELSEVIER

Available online at www.sciencedirect.com
SCIENCE DIRECT®

Microprocessors and Microsystems 28 (2004) 213–222

MICROPROCESSORS AND MICROSYSTEMS

www.elsevier.com/locate/micpro

Case study of a functional genomics application for an FPGA-based coprocessor☆

Tom Van Court[a,*], Martin C. Herbordt[b,1], Richard J. Barton[c,2]

[a]*Department of Electrical and Computer Engineering, Boston University, 413 Photonics Building, 8 Saint Mary's Street, Boston, MA 02215, USA*
[b]*Department of Electrical and Computer Engineering, Boston University, 324 Photonics Building; 8 Saint Mary's Street, Boston, MA 02215, USA*
[c]*Department of Electrical and Computer Engineering, University of Houston, Houston, TX 77204, USA*

Received 20 October 2003; revised 16 February 2004; accepted 4 March 2004

Abstract

Although microarrays are already having a tremendous impact on biomedical science, they still present great computational challenges. We examine a particular problem involving the computation of linear regressions on a large number of vector combinations in a high-dimensional parameter space, a problem that was found to be virtually intractable on a PC cluster. We observe that characteristics of this problem map particularly well to FPGAs and confirm this with an implementation that results in a 1600-fold speed-up over an optimized serial implementation. Some of the other contributions involve the data routing structure, the analysis of bit-width allocation, and the handling of missing data. Since this problem is representative of many in functional genomics, part of the overall significance of this work is that it points to a new area of applicability for FPGA coprocessors.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Computational coprocessor; FPGA application; Bioinformatics; Microarray data analysis; Steiner system

1. Introduction

Microarrays measure simultaneously the expression products of thousands of genes in a tissue sample and so are being used to investigate a number of critical biology questions [1,3,6,10,12,22]. Among these are (paraphrasing from pages 19 to 20 of Ref. [17]): Given the effect of 5000 drugs on various cancer cell lines, which gene is most predictive of the responsiveness of the cell line to a particular chemotherapeutic agent? or Is there a group of genes that can serve to distinguish the outcomes of patients with disease *ijk* who are otherwise indistinguishable? or Which of all known genes have a pattern of expression similar to those genes regulated by factor *xyz*?

As exciting as this usage is, microarray analysis is extremely challenging. One issue is that the data are noisy and the noise is often difficult to characterize. Another issue is that the number of measured quantities is invariably much larger than the number of samples. This results in an underconstrained system not amenable to traditional statistical analysis such as finding correlations. As a result of these and other difficulties, techniques are used (often derived from machine learning) that provide a focus of attention, or a visualization, from which biological significance can be inferred. Among these are various forms of clustering, inference nets, and decision trees. Their computational complexity ranges from the trivial to the intractable. However, given the cost of obtaining microarray data, the fact that further biological interpretation is usually still required, and the value of many of the most rudimentary computations, most analysis applications are tailored to run fairly quickly on ordinary PCs.

It is undoubtedly the case, however, that biologists would like to ask far more complex questions and that increased computational capability would help to answer them. With applications such as those listed above, however, even a slightly harder question can result in an increase by orders of magnitude in computation. This is true of the problem we investigate here.

☆ A preliminary version of this work appeared in the Proceedings of the 13th International Conference on Field Programmable Logic and Applications (FPL 2003).

* Corresponding author. Tel.: +1-617-353-2840.
E-mail address: tvancour@bu.edu (T. Van Court).
[1] Tel.: +1-617-353-9850.
[2] Tel.: +1-713-743-4454.

0141-9331/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.micpro.2004.03.005

Kim et al. [16] would like to find a set of genes whose expression can be used to determine whether liver tissue samples are metastatic or non-metastatic. For biological reasons, it is likely that three genes is an appropriate number to make this determination. Kim et al. further propose that use of linear regression would be appropriate to evaluate the gene subsets over the available samples. Since there are tens of thousands of potential genes, 10^{11} to 10^{12} data subsets need to be processed. Although simple to implement, he reported that this computation was intractable even on his small cluster of PCs.

We decided to investigate the prospects of supporting this computation on an FPGA-based coprocessor. There are many reasons:

- It is an excellent match computationally. The data-set size, the amount of computation per datum, the nature of the individual computations, and the data-type size all are favorable to FPGAs.
- This computation is representative of a large number of similar computations in microarray analysis as well as in broader bioinformatics and computational biology (BCB). Therefore, demonstrating the efficacy for this problem would have much wider significance. Note that others (e.g. Refs. [11,24,27]) have shown similar success (to what we show here) for another application in bioinformatics; that is, of applying dynamic programming to sequence analysis. However, the computational characteristics of that particular application bear no resemblance to the problem addressed here.
- There is a large number—perhaps thousands—of potential users. Therefore, a low-cost distributed solution is much more attractive than a centralized resource such as a large-scale cluster. This is especially true since (as we will show) it would take a very large cluster to match performance.
- The state of algorithmics in microarray analysis (and some other areas of bioinformatics) is one of flux. Therefore, a solution based on a generic PC-and-coprocessor may be more attractive than hardwired alternatives such as ASICs. The same would be true with respect to turn-key software/hardware systems, such as those provided by a number of vendors, assuming that they provided solutions to these problems at all. Also, both of these 'hardwired' alternatives are extremely expensive.

What we have found is that we can obtain a speed-up of a factor of 1600 over an optimized serial version running on a 1.7 GHz Pentium IV PC. These results have so far been achieved in simulation using post place-and-route timing for the Xilinx XC2VP100-7.

Our most basic contribution is the speed-up for this particular problem: data sets of nearly 10,000 genes can be examined in less than 20 min instead of 19 days. Other contributions have to do with the actual implementation, including a novel data routing structure, development of an architectural family of computations, the analysis of the bit-width allocation (precision management), and our handling of missing data. Finally, as this problem has similar characteristics to many others in microarray analysis, we show that there is potential for broader applicability of FPGA coprocessors in this very important domain.

The rest of this paper is organized as follows. In Section 3 we present the problem formally and describe the serial implementation. There follows a description of the FPGA design and implementation including an analysis of data path widths. Then comes a description of extensions. We conclude with a discussion.

2. Application detail and serial implementation

The data to be analyzed are derived from n microarrays, each a sample from a diseased or healthy (control) tissue. The data consist of a binary diagnosis and an expression value for each of the genes being analyzed. Expression values are tabulated in a matrix with row vectors corresponding to microarrays and column vectors corresponding to particular genes. The outcomes are tabulated in a column vector $\mathbf{Y}$. The technique used is to compute the linear regressions for all three-way combinations of genes. Pearson's R^2 defines the goodness of fit for each regression. Examining a standard statistics reference [23], we find that the estimators $\hat{\beta}_0, \ldots, \hat{\beta}_n$ comprising the column vector $\hat{\boldsymbol{\beta}}$ can be computed as follows:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{Y}$$

The first column of $\mathbf{X}$ consists of n 1s and each remaining column consists of the n expression values for one gene of the subset being considered in this particular combination. However, since much of computational complexity results from the inversion, it is important to reduce its rank. This is done by centering the data, which results in a $\hat{\boldsymbol{\beta}} = \hat{\beta}_1, \ldots, \hat{\beta}_n$ of

$$\hat{\boldsymbol{\beta}} = [(\mathbf{X}^+)^{\mathrm{T}}\mathbf{X}^+]^{-1}(\mathbf{X}^+)^{\mathrm{T}}\mathbf{Y}^+$$

and

$$R^2 = \frac{\hat{\beta}(\mathbf{X}^+)^{\mathrm{T}}\mathbf{Y}^+}{(\mathbf{Y}^+)^{\mathrm{T}}Y^+}$$

where $\mathbf{X}^+$ is the $n \times 3$ matrix containing column vectors with elements $X_{ij} - \bar{\mathbf{X}}_i$ and $\mathbf{Y}^+$ is the column vector containing the values of $Y_j - \bar{\mathbf{Y}}$. Note that in centered mode we do not need to compute $\hat{\beta}_0$ and thus do not need the column of ones in $\mathbf{X}$. We therefore operate on 3×3 matrices rather than 4×4.

The covariance matrix

$$c_{ij} = (\mathbf{X}_i - \bar{\mathbf{X}}_i)^{\mathrm{T}}(\mathbf{X}_j - \bar{\mathbf{X}}_j)$$

can also be written as

$$nc_{ij} = n\sum_k (X_{ik}X_{jk}) - \left(\sum_k X_{ik}\right)\left(\sum_k X_{jk}\right)$$

The factor of n cancels in later operations, and eliminates the need to perform the division implied by $\bar{X}_i$. The entire computation for each gene combination thus partitions into two parts: the first consists of the nine dot products and four summations while the second consists of computing the covariance matrix, inverting the matrix, and using those results to obtain $\hat{\boldsymbol{\beta}}$ and R^2.

In our tests we used data derived from a human breast tumor study by Perou et al. [22]. The data are typical of those generated in microarray studies and consist of expressions of 9218 genes (including controls) from 84 microarray samples. Raw expression data are ratios. As is standard practice, we took the log, normalized, and rounded the data to integer values in the range -4 to $+4$. Using four bits is on the high end of information per sample; often only two bits are used. The result vector is binary: 0/1 for diseased/healthy.

An important consideration in microarray analysis is dealing with missing data. That is, the microarray value for a gene/sample expression is sometimes unreadable; in that case no value at all is reported. In the Perou data set, 46% of genes contain missing data. If not handled properly, missing data can dominate the regressions and render results meaningless. We take a simple approach: for a given combination of three genes, for each sample with missing data, we eliminate that sample from consideration for all genes. The combination is rejected completely if too many healthy samples are dropped.

Statistical literature [18] refers to this as a form of complete-case analysis within any one combination of genes, and available-case analysis in selecting combinations of genes. A χ^2 test of Perou's data shows that the frequency of missing values is not decisively different among healthy samples than among diseased samples. Thus, the missing data matches the Missing At Random assumption and complete-case analysis appears justified.

When implementing the algorithm, one notes that each dot-product and sum is used a large number of times. It seems to make sense to precompute *all* of the $n \times n$ dot products, then use them in the $\binom{n}{3}$ inversions later. This eliminates roughly a factor of n dot products. Unfortunately, this does not account for missing data: each dot-product can have drop-outs not just from data missing from the two vectors being multiplied, but also from the third vector of the set. Since this third vector changes for every set, it follows that all dot products must be recomputed for every iteration.

We created two versions of the serial code. One was a mirror of the FPGA implementation and was used to verify results, especially with respect to maintaining precision. The other was used to generate timing and so was highly optimized for serial execution on a modern processor.

The R^2 for each set of genes was computed in 10.1 μs on a 1.7 GHz Pentium IV PC. Because of the tremendous data locality, there was no drop-off in performance with respect to the number of gene combinations evaluated. This means that evaluating three-way combinations of 1000 genes takes about half an hour, while three-way combinations of 10,000 genes takes more than 19 days, and 20,000 genes takes more than five months. Clearly L1 cache and available ILP are being used to a very high degree, the latter not surprisingly due to the numerous multiply–accumulate (MAC) computations and the hardwired invert.

Still, these results confirm our initial assumption: that this computation, while perhaps not 'heroic' in the grand-challenge sense, is still outside the realm of usage in exploratory data analysis. For this and similar computations to be readily usable as part of an analysis toolkit, days need to be reduced to minutes.

3. FPGA methods, implementation, and results

3.1. Hardware model

The overall hardware model is simply an FPGA on a PCI board plugged into a PC. As the amount of reuse per datum is very high, details about the particular board and interface do not have a significant impact on our results: only a KB/second input rate needs to be supported. Several commercial products meet these criteria (e.g. Refs. [2,21]).

The rest of the discussion in this section describes simulations, synthesis, and place-and-route in the Xilinx ISE 5.2.02i environment [25] for Virtex-II Pro XC2VP100 gate array [26] and anticipates implementation on a generic coprocessor board when one becomes available for this chip in 2004. The Virtex-II Pro product family is especially interesting here because of its large gate count, large on-chip memory, and high number of dedicated multipliers. Implementations on other devices in this family follow from the one described here using analogous optimizations and are described in Section 4.

We use standard VHDL language constructs throughout the design. Chip-specific synthesis controls are used only for indicating where RAMs be synthesized and for choosing between hardwired and synthesized (LUT-based) multipliers.

3.2. Parallel algorithm

As is often the case, the serial algorithm needs to be substantially reworked to obtain maximal performance on an FPGA. The key differences are: (i) use of minimal initial data type size and use of precision management to keep this minimality throughout the computation and (ii) parallelizing the computation, which results in hundreds of pipelines and a complex data distribution network to service those pipelines. There are also differences that may be termed

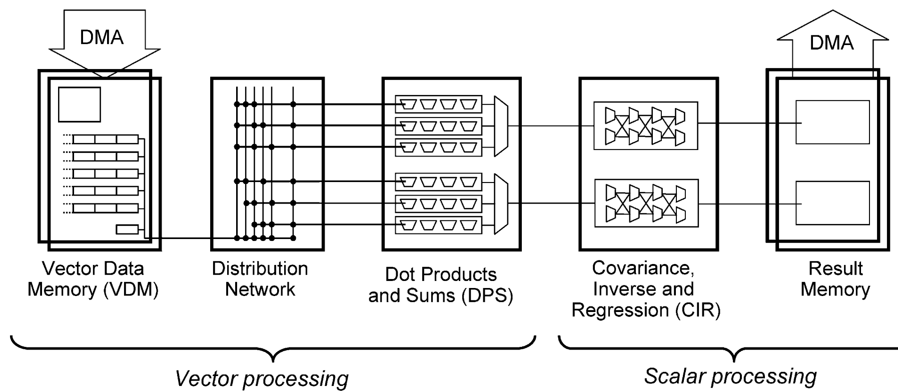

Fig. 1. Schematic of entire computation.

technology-oriented rather than algorithmic. These include use of combinational logic to perform safety checks in parallel, hardwiring substantial computations, performing boolean operations with single gates, and avoiding division.

The FPGA implementation is summarized in Fig. 1. The circuitry consists of three parts: (i) vector data memory and distribution network (VDM) and the two computational segments described in Section 2: (ii) dot-products and summations (DPS) and (iii) covariance matrix, inverse, and regression (CIR).

We now show how the serial algorithm maps onto the parallel hardware. For clarity, we assume 10,000 vectors (genes per microarray) of length 100 (number of micro-arrays/samples in the medical study). Recall that the outer loop requires computing regressions of all $\binom{10,000}{3} = 1.66 \times 10^{11}$ three-way combinations of vectors. In the parallel version, the outer loop consists of choosing nine vectors at a time (of the 10,000) and processing in parallel all $\binom{9}{3} = 84$ combinations of those nine vectors. The processing within each of the 84 combinations is also done in parallel. In one iteration of the outer loop the following steps occur:

- The vector memory provides elements from the nine vectors in parallel; the 100 elements/vector are provided serially so nine streams of data are sent in parallel to the distribution network.
- The distribution network replicates the nine streams to feed the 84 combinations of three vectors (for a total of 252 streams of 100 elements) into the 84 DPS units.
- Each to the 84 DPS units performs in parallel all of the dot products and summations required for a single three-way vector combination.
- The outputs of the DPS units are scalar elements of the covariance matrix. Processing the matrices is done by the CIRs. Because the CIRs are much faster than the DPSs, each CIR is fed by a number of DPSs.
- The results from the best combinations (above some threshold) are retained in result memory before transfer to the host.

We can currently store about 10% of the entire data set on-chip, so the vector memory must be reloaded periodically. However, even with each vector being used with eight others in parallel, vectors are still used in a large number of nine-way combinations before they need to be swapped.

We next give details of the implementation, starting with the DPS and CIR. In the following sections we talk about optimizations and safety checks and then about timing, integration, and speed-matching. At that point the responsibilities of the VDM are clear and it is then described.

3.3. DPS and CIR overview

Each DPS accepts four data vectors, three X values and one Y. The Y represents the diagnosis, 0 or 1 for cancerous or healthy samples, respectively. The X values represent expression levels, encoded as four bit values. The encoding represents a symmetric range from $-N$ to $+N$. Earlier, we noted that the application works well when expression data is quantized to a range of -4 to 4 (encoded as 0–8). A special value (15, binary 1111) is a Not-a-Number (NaN) code that represents missing or invalid input data. The DPS unit, illustrated in Fig. 2, consists of: counters to tally valid data sets and Y values, accumulators to total the X vectors and XY dot products, and MAC sections to total the X_iX_j dot products and X_i^2.

Note the handling of missing data. In Fig. 2, the box labeled = NaN? detects missing data and propagates that fact to the MAC units where the accumulation of the invalid summands is blocked. If, for example X_{1i} is a missing value, then the summands $X_{1i}^2, X_{1i}X_{2i}, X_{1i}X_{3i}$ and $X_{1i}Y_i$ are obviously meaningless. Following Ref. [18], X_{2i}, X_{2i}^2, X_{3i}, X_{3i}^2, $X_{2i}X_{3i}$, $X_{2i}Yi$, $X_{3i}Y_i$, and Y_i are also omitted from their respective sums and from the vector length count.

This DPS computation takes advantage of the 1-bit Y values in several ways. First, the summation of Y values reduces from an accumulator to a counter. Second, the summation of Y^2 values is redundant. Given 0 and 1 as the only possible Y values, the sums of Y and Y^2 are identical. Third, the XY sums are accumulators, conditionally adding

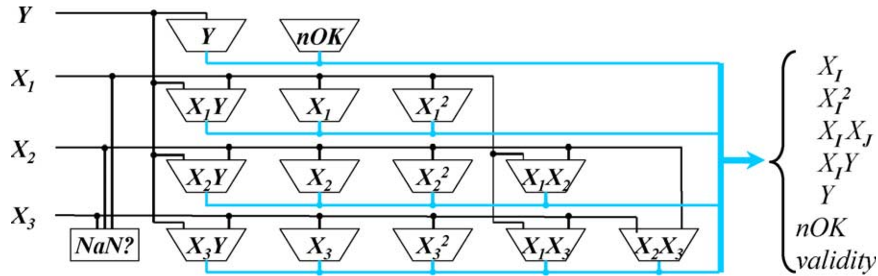

Fig. 2. A dot-products and summations (DPS) unit.

X elements depending on Y being 0 or 1. Only the dot products of X vectors require true multiplication, 4×4 bit products of unsigned values, giving 8 bit results or less.

Once the vector is processed, DPS results are latched for input to the CIR (see Fig. 3). DPS results consist of outputs from all accumulators in Fig. 2, a valid data counter, and the validity indicator (overflow and minimum-Y tests—details below). The result is presented broadside to the CIR. That section consists entirely of unclocked combinational logic, including adders, subtracters, and multipliers. It first computes the 3×3 covariance matrix from the dot products. That feeds the closed form inversion of the covariance matrix.

3.4. DPS and CIR details: optimizations and safety checks

One fundamental optimization is to minimize the datapath width at all points. Worst-case calculations of the minimum width are simple and safe, but result in a far more conservative implementation than necessary. We address this by: (i) a priori determining the maximum datapath widths and (ii) confirming that overflow has not occurred. There are two aspects to datapath width determination: the bits that can be dropped at the high end (because of worst cases that never occur in practice) and bits that can be dropped at the low end (because the loss of precision is insignificant).

The a priori path width determination is done in two ways: empirical and theoretical. On the theoretical side, the need for less significant bits is estimated using interval arithmetic [9]. Precision is verified empirically by sampling test data off-line. That is, covariance and inversion calculations are simulated on samples of the actual data. Then the covariance and inverse matrices are multiplied together. The ideal result would be an identity matrix; measured precision is acceptable if results lie within a specified distance of the ideal. Note that although this application resembles common signal processing in some ways, the meanings of truncation errors differ. Existing techniques for determining numbers of significant bits are based on controlling signal-to-noise ratios [8] throughout the computation. It is difficult to assign meaning to an SNR in this statistical inference application; instead, our analysis is based on confidence intervals. Part of our ongoing research includes formalization and automation of that analysis.

Also on the theoretical side, worst-case analysis sets an upper bound on the number of high-order bits required. It is interesting that most authors (e.g. Ref. [20]) perform worst-case bit allocation in whole bit increments, even when their

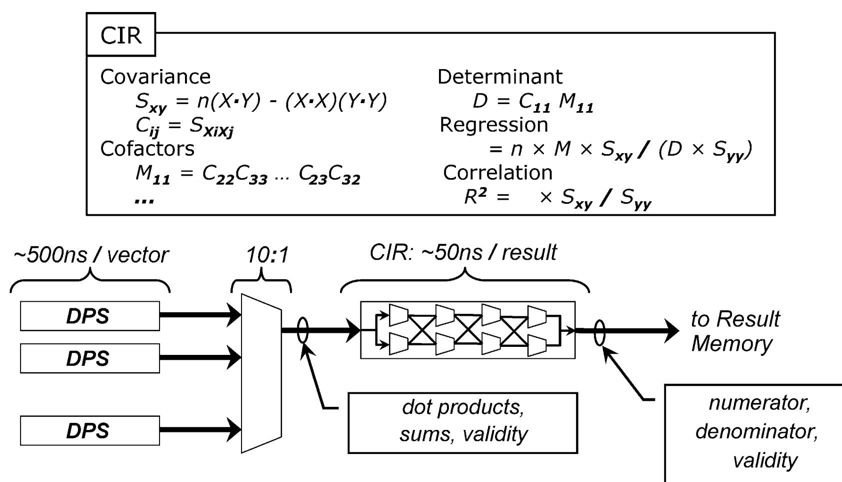

Fig. 3. A covariance matrix, inverse, and regression (CIR) unit and its interfaces to DPS units and result memory.

bitwidth allocation is otherwise sensitive to application data values. For example, consider the accumulator needed to add up 84 terms in a dot-product of vectors with nine element values, 0–8, as described above. Naively, that would require $7 + 4 + 4 = 15$ bits. In fact, the accumulator need only hold $\lceil \log_2(84 \times 8 \times 8) \rceil = 13$ bits to handle the worst-case for this application.

Empirical estimates of high order bit requirements also come from measurements of calculations on sample data. The calculation is performed off-line, for a meaningful subset of the application cases, and the number of bits used at each step is measured. This gives statistics of actual bit usage. High-order bits are allocated to cover the samples observed, plus some margin based on observed standard deviations. The implementation, then, handles all data values that can reasonably be expected. To be thorough, however, the FPGA logic checks for overflow at each step. The DPS and CIR stages both present validity indicators, stating whether an erroneous calculation was detected at any point.

As noted earlier, some regressions may be invalid because missing data values leave too few $Y = 1$ (healthy) samples. Fig. 2 shows the 'Y sum valid?' check, a configurable test requiring some minimum number of healthy samples in a data vector. If inadequate data with $Y = 1$ appears, that also marks the whole calculation as invalid. A similar test of the number of $Y = 0$ samples is not necessary for this data set.

In the CIR, one optimization is the use of closed form inversion. Although this method has many problems when applied to large matrices, it works well with these matrix, vector, and data sizes. Besides the obvious speed advantage, hardwiring allows us to keep cofactors and determinant separate for independent use in computing correlation coefficients and regression.

Another optimization is that all multiplications in the CIR use the FPGA's block multipliers. The DPS, with smaller operands, uses multipliers built from logic. Further, optimization may be possible in the CIR by exchanging some block multipliers for multipliers built from logic for sufficiently small operands.

Various other obvious optimizations were found not to be beneficial due to timing and resource allocation tradeoffs. These include pipelining the CIR and processing DPS vectors in parallel.

3.5. Putting together the DPS and CIR: timing and speed-matching

Overall throughput is described later; here we begin the description of timing in the context of speed-matching the parallel DPS's and CIRs. Timing is computed statically, using post place-and-route (PAR) results. PAR is completely automatic, with no manual guidance, timing constraints, or floor-planning. Timing estimates are based on synthesis of nine CIR units, each supplied by 10 DPS units (90 DPSs total; see Fig. 3 for one CIR/10 DPS unit) and VDM as described below. The target is a XC2VP100 gate array with speed grade −7. The entire design, minus 'glue' needed to attach the application logic to its host environment, occupies 73% of the logic slices and 94% of the block multipliers. This design is naturally the product of several iterations: throughput balancing depends on the timings of synthesized circuits, but the circuits are designed to balance throughput in each section.

Post-PAR timing analysis indicates a DPS clock of 5.35 ns while propagation delay through the CIR to its result register is 47.36 ns. Using conservative approximations, this implies a 5.5 ns DPS clock rate and very roughly a 10:1 ratio of CIR delay to DPS rate. The system clock is set to the DPS rate. A separate counter divides the system rate down so that CIR results are latched and new results presented to the CIR at 1/10 the DPS clock rate.

Given data vectors of length ~ 80, the DPS requires about 500 ns to compute a result for one set of vectors. The CIR vs. DPS imbalance, about 50 vs. 500 ns, is conspicuous. The other conspicuous imbalance is in the resources used by each section. Each CIR requires 48 of the target FPGA's 444 block multipliers. The DPS, however, uses only 4×4-bit products, which can be built more effectively from ordinary logic. A DPS uses smaller, less specialized logic resources, under 1% of the chip total. Considering both execution time and logic resources used, to achieve maximum logic activity, each CIR serves 10 DPS units. Fig. 3 shows how this is done.

The DPSs all start and end their computations at the same time, latch their results, and begin processing the next set of vectors. After latching the DPS results, separate logic presents results from each DPS to its CIR sequentially. A counter (not shown in Fig. 3) holds input stable for the CIR propagation delay, then latches the CIR result and reads the next saved DPS result.

These design values (vector length, CIR propagation delay, etc.) and available resources are all parameters specific to the problem and technology at hand. The effect of varying these parameters is described in Section 4.

3.6. Feeding the pipelines: vector data memory

Recall that each DPS unit processes three vectors and that there are 90 DPS units. Therefore, the VDM must simultaneously distribute 270 vector streams (see Fig. 4). A general (but impractical) solution would be to store all $v = 10{,}000$ vectors on chip in a 270-ported memory. Size itself is not the problem: the entire data set fits in 1 MB. Our solution leverages the v^2 reuse of each vector in a hierarchical structure.

We observe that nine vectors form $\binom{9}{3} = 84$ combinations and so are nearly sufficient to keep the 90 DPS units busy. This leaves six DPSs idle, the price paid for efficient memory access. A simple network (see Fig. 5) distributes vector data to the DPSs.

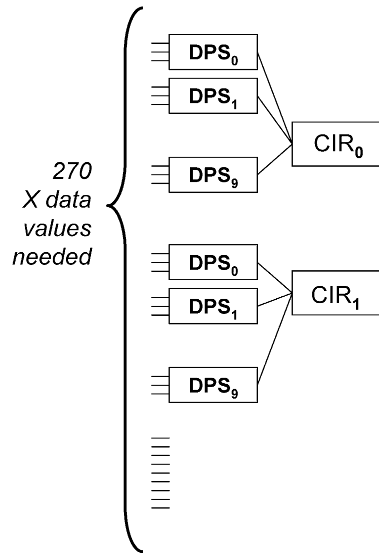

Fig. 4. VDM requirements are derived from the number and specification of the DPS pipelines.

Each vector store unit (labeled $X_1 \cdots X_m$ in Fig. 6 and within the VDM in Fig. 5) feeds one of the distribution network's nine X inputs and is indexed by an independent address register (labeled $\text{index}_1 \cdots \text{index}_m$). The sequence of vectors is stored in the Vector Select Ram. The chip's 16 Kb RAM blocks can each hold 50 vectors, so vector storage uses only a small amount of the available RAM. Vector store units and the Vector Select RAM are loaded from off-chip. Double buffering ensures that loading does not interfere with data access for computation. Note that many more values are read from the vector stores than are loaded into them, since each vector is reused in many size-9 combinations, so the DPS should never be idle while the VDM is reloaded.

Operation is as follows. A set of nine vectors is chosen at the beginning of a vector computation, one from each vector store unit, as indicated by the address registers. Successive vector elements are read by incrementing all of the data address registers in unison, through the length of the vector. At the end of a vector, the address registers are reloaded from the Vector Select RAM and the next set of data vectors is ready for transfer. The Y vector is reused with all sets of X vectors.

This VDM design supports the solution of the following combinatorial problem: choosing sets of size 9 (or k) from a set of 9218 (or v) elements such that every size-3 subset of those v appears in one size-k set (again, see Fig. 5). This is exactly the Steiner System $S(3, k, v)$. Generating algorithms exist for these size-k sets [13], but are not amenable to FPGA implementation. Off-line computation generates that set of size-k sets, and loads the vector stores and Vector Select RAM accordingly. The Vector Select RAM and vector stores must be reloaded $10^5 - 10^6$ times to cover all size-3 sets; however, this is a trivial requirement for a run of several minutes.

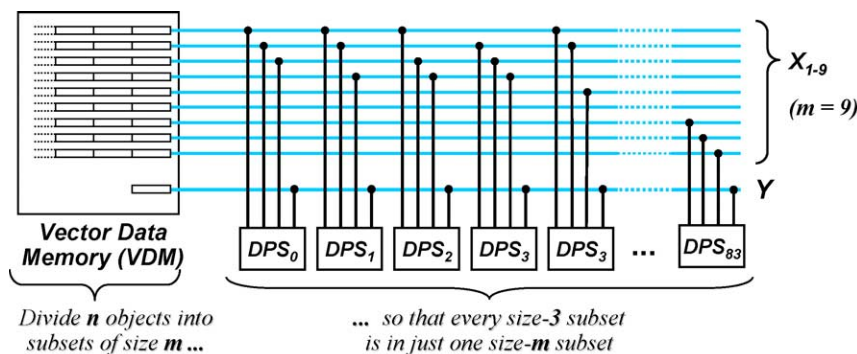

Fig. 5. The Vector Data Memory (VDM) network. Note that $m = 9$ vectors are sufficient to process 84 sets simultaneously.

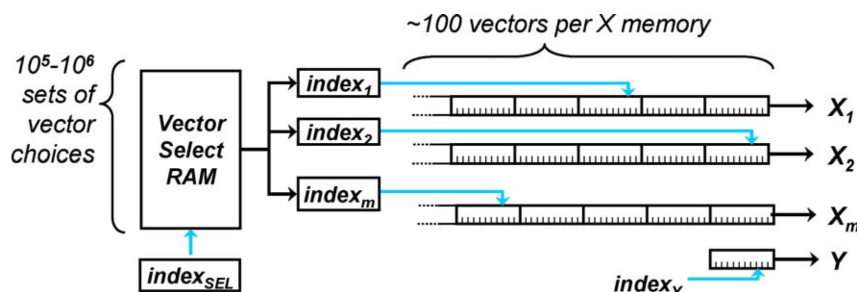

Fig. 6. The Vector Data Memory indexing structures. For each combination of 84 sets, the index registers step through the appropriate set of $m = 9$ vectors. The sequence of initial values of the index registers, stored in the Vector Select Ram, is determined off-line.

Table 1
Resource allocations for other FPGAs and vector sizes

Xilinx model	Hard multipliers	CIRs	DPSs-per-CIR	X data RAMs	DPSs, triplets
2vp125	556	11	20	12	220, 220
			15	11	165, 165
			11	10	121, 120
			8	9	88, 84
2vp100	444	9	14	10	126, 120
			10	9	90, 84
			7	8	63, 56
2vp70	328	6	20	10	120, 120
			12	9	84, 84
			10	8	60, 56

Triplets refers to the number of DPS's actually used after accounting for combinatorial inefficiency.

3.7. Performance

As described, the basic component of our design consists of a pipeline with a single CIR (~50 ns) and single DPS (~500 ns for a length-80 vector). This gives a speed-up of a factor of 20 over the serial PC version. The chosen Virtex II Pro easily fits nine of these units (the critical resource being the multipliers) increasing the speed-up to 180. Each CIR serves 10 DPSs reducing the cycle time of the original unit to 50 ns and increasing the total speed-up to a factor of 1800. Combinatoric inefficiency in the VDM reduces this factor to 1600. These results are summarized in Table 1.

4. Extensions

Being a new application/technology combination, this work lends itself to myriad extensions.

4.1. Resource allocation for other FPGAs and data set sizes

The size of the data input for microarray computations is severely constrained by the application itself: the number of vectors is limited by the number of sites on the microarray and the vector size by the number of microarrays. In any case, the number of vectors has virtually no effect on either the hardware configuration or the throughput. The vector length, however, affects the ratio of DPS's to CIRs. This and the effect of the computational resources of the target FPGA on throughput is summarized in Table 2. The number of hard multipliers per chip is the constraining resource and is likely to remain so up to DPS/CIR ratio of at least 20 (corresponding to a doubling of vector length). Table 3 shows resource allocations when the FPGAs and the vector lengths are varied. *X Data Rams* refers to the number of vectors being read simultaneously from the Vector Data Management unit. Recall that this number grows very slowly as it suffices for the number, taken three-at-a-time, be larger than the number of DPS's. The throughput follows almost immediately from the resource allocation.

Table 2
Effect of resource constraints on throughput

Constraint	Determines
Multipliers-per-chip/multipliers-per-CIR	No. of CIRs
Vector-length × DPS-cycle-time	DPS time per vector
DPS-time/CIR-time	DPSs per CIR
DPS's-per-CIR × No.-of-CIRs	No. of DPSs
MAX m s.t. $\binom{m}{3} \leq$ No.-of-DPSs	No. of X memories
$\binom{n}{3}$/No. of DPSs	Combinatoric efficiency
min (DPS-time/DPSs-per-CIR, CIR time)	Vector time
No.-of-DPSs × combinatoric-efficiency/vector time	Throughput

Table 3
Performance summary

Hardware	Time	Per work done per hardware
DPS vector time	5 × ~100 = ~500 ns	Per combination per DPS DPSs
10 DPSs per CIR	500/10 = 50 ns	Per combination per CIR
9 CIRs per chip	50/9 = 5.5 ns	Per combination per chip
93% Combinatoric inefficiency	5.5/0.93 = 5.9 ns	Per combination per chip
FPGA	~710 s	For all combinations per chip
PC	10 μs = 10^4 ns	Per combination per PC
PC	~2 weeks	For all combinations per PC

Ratio of PC time to FPGA time is ~1600.

4.2. Larger combinations of genes

Although the set size of three emerged from domain knowledge, being able to evaluate larger set sizes would also be useful. Assuming (at first) that the configuration beyond the VDM remains the same as does the performance requirement (<20 min for the result), the combinatorics dictate a bound of about 1500 genes for a set size of four and 500 genes for a set size of five. Reductions such as these are already typical in current microarray computations and are done using various filtering techniques. These present an obvious trade-off: more filtering means better performance, but also an increased likelihood that important results will be eliminated unintentionally. As with many types of computations, we see that the end-result of a 1000-fold speed-up is less an increase throughput and more an improvement in quality.

The prominent effect of set size on the back-end is in the matrix inversion. With a set size ≥4, our method of a hardwired combinational circuit using rational arithmetic may be untenable and Gaussian elimination may be

required. In that case, it is likely that the throughput of the entire computation would reduce to the throughput of the parallel inversions. More likely would be a change in the back-end to a simpler metric, for example, one related to correlation.

4.3. Changing the back-end

As we just mentioned, linear regression is only one of several possible methods to score combinations of genes. We have also examined the use of a linear discriminator [22] where the figure of merit is a Mahalanobis distance measurement between the X values corresponding to $Y = 1$ and $Y = 0$ [15]. The overall implementation follows that of the regressor, with many components being reused and overall resource allocation and timing also being similar. Other distance measurements under investigation include standard tests of the statistical significance of differences between the two sets of X values.

4.4. Use as part of higher order tool-kit

Research into Microarray analysis is perhaps only in its initial stages: they were invented less than ten years ago, their usage has a growth rate of 70% per year, and papers on microarray analysis techniques provide a large (and increasing) faction of those presented at bioinformatics symposia. One of the key aspects of this work is therefore design is for extendability. In particular, the components were built with reuse in mind and so are highly parameterized. Also, we have considered the generalization of higher-level system components to enable their interchangeability while maintaining maximum resource allocation and throughput.

5. Discussion

We have described work in accelerating a computation in functional genomics by using an FPGA coprocessor. The 1000-fold + speed-up derived from our FPGA implementation achieves our goal of reducing the duration of this computation from days to minutes. This is done while keeping the system cost (hardware and IT support) low. It is therefore quite plausible that this and related techniques could indeed become part of a computational toolbox for functional genomics, broadening the types of inferences possible from microarray data.

This application differs fundamentally from others in BCB to which computational coprocessors have been applied [5,4,7,14,19]. Those have resembled the most typical usage of FPGAs: deeply pipelined computational structures such as are used in data communications and signal processing. Rather, we have applied the massive computational and communication capability within the FPGA to manage an on-chip combinatorial explosion. Also in contrast to previous bioinformatics coprocessors is that the present system performs much more complex arithmetic than is required for the common BCB targets of acceleration (e.g. Needleman–Wunsch string matching and BLAST). As such it requires non-trivial optimization of resources, especially the higher level components such as the hard multipliers.

In Section 1, we described why the potential for speed-up was so great for this application, even though it already can use a serial processor at close to maximum capacity. Here we present those points in a slightly different way and state that a large number of BCB problems have a similar characteristic structure. In particular, they have:

- A high-dimensional parameter set that must be searched or enumerated to find an optimal solution,
- Simple performance criteria (score functions) derived from the parameters,
- A decomposable search strategy and/or score function,
- A large but manageable sample data set that must be processed to evaluate the score function for each candidate element in the parameter set, and
- Data elements representable in modest numbers of bits,
- Significant reuse of data elements across candidate solutions.

Acknowledgements

Al Conti performed extensive debugging and also implemented some of the extensions. We would like to thank Xilinx for the generous donations of their products. We would also like the thank the anonymous reviewers for their helpful suggestions. This work was supported in part by the National Science Foundation through CAREER award No. 9702483 and by the Texas Advanced Research Program (Advanced Technology Program) under grant No. 003652-952.

References

[1] U. Alon, D. Notterman, K. Gish, S. Ybarra, D. Mack, A. Levine, Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays, Proceedings of the National Academy of Science 96 (1999) 6745–6750.

[2] Avnet, Xilinx Virtex-II Pro Development Kit, Avnet, Phoenix, AZ, 2003.

[3] M. Bittner, et al., Molecular classification of cutaneous malignant melanoma by gene expression profiling, Nature 406 (2000) 536–540.

[4] H.-M. Bluehgen, T. Noll, A programmable processor for approximate string matching with high throughput rate, Proceedings of the International Conference on Application Specific Systems, Architectures, and Processors, 2000, pp. 309–316.

[5] M. Borah, R. Bajwa, S. Hannenhalli, M. Irwin, A SIMD solution to the sequence comparison problem on the MGAP, Proceedings of

the International Conference on Application Specific Array Processors, 1994, pp. 336–345.
[6] A. Butte, P. Tamayo, D. Slonim, T. Golub, I. Kohane, Discovering functional relationships between rna expression and chemotherapeutic susceptibility using relevance networks, Proceedings of the National Academy of Science 97 (22) (2000) 12182–12186.
[7] E. Chow, T. Hunkapiller, J. Peterson, Biological information signal processor, Proceedings of the Conference on Application Specific Array Processors, 1991, pp. 144–160.
[8] G. Constantinides, P. Cheung, W. Luk, Optimum wordlength allocation, Proceedings of FPGA Custom Computing Machines, 2002.
[9] E. Hansen, Topics in Interval Analysis, Clarendon Press, Oxford, 1969.
[10] M. Eisen, P. Spellman, P. Brown, D. Botstein, Cluster analysis and display of genome-wide expression patterns, Proceedings of the National Academy of Science 95 (1998) 14863–14868.
[11] S. Guccione, E. Keller, Gene matching using JBits, Proceedings of Field Programmable Logic and Applications, 2002, pp. 1168–1171.
[12] E. Gunther, D. Stone, R. Gerwien, P. Bento, M. Heyes, Prediction of clinical drug efficacy by classification of drug-induced genomic expression profiles in vitro, Proceedings of the National Academy of Science 100 (16) (2003) 9608–9613.
[13] A. Hartman, A fundamental construction of 3-designs, Discrete Mathematics 124 (1994) 107–132.
[14] D. Hoang, Searching genetic databases on splash 2, Proceedings of IEEE Workshop FPGAs for Custom Computing Machines (1993) 185–191.
[15] R. Johnson, D. Wichern, Applied Multivariate Statistical Analysis, Prentice-Hall, Upper Saddle River, NJ, 2002.
[16] Y. Kim, M.-J. Noh, T.-D. Han, S.-D. Kim, S.-B. Yang, Finding genes for cancer classification: Many genes and small number of samples, Second Annual Houston Forum on Cancer Genomics and Informatics, 2001.
[17] I. Kohane, A. Kho, A. Butte, Microarrays for an Integrative Genomics, MIT Press, Cambridge, MA, 2003.
[18] R. Little, D. Rubin, Statistical Analysis with Missing Data, Wiley/Interscience, New York, 2002.
[19] D. Lopresti, P-NAC: a systolic array for comparing nucleic acid sequences, IEEE Computer 20 (7) (1987) 98–99.
[20] S. Mahlke, R. Ravindran, M. Schlansker, R. Schreiber, T. Sherwood, Bitwidth cognizant architecture synthesis of custom hardware accelerators, IEEE Transaction on CAD of Integrated Circuits and Systems 20 (2001) 1355–1370.
[21] Nallatech, Dime-II Series Motherboards, Nallatech, Glasgow, 2003.
[22] C. Perou, et al., Prediction of clinical outcome with microarray data: a partial least squares discriminant analysis (PLS-DA) approach, Human Genetics 112 (2003) 581–592.
[23] T. Ryan, Modern Regression Methods, Wiley, New York, 1997.
[24] Time Logic Corp, web site: www.timelogic.com, 2003.
[25] Xilinx, Integrated Software Environment, Xilinx, San Jose, CA, 2002.
[26] Xilinx, Virtex-II Pro Platform FPGA User Guide, Xilinx, San Jose, CA, 2002.
[27] Y. Yamaguchi, Y. Miyajima, T. Maruyama, A. Konagaya, High speed homology search using run-time reconfiguration, Proceedings of Field Programmable Logic and Applications, 2002, pp. 281–291.

Tom Van Court received the BS degree in Engineering from Cornell University and the MS degree in Computer Science from Boston University. He is currently pursuing the PhD degree at Boston University in Electrical and Computer Engineering. His current research centers on the use of FPGAs as application-specific coprocessors and on adapting software design technology to FPGA-based computing. He also teaches software development in the CS department of BU's Metropolitan College. Van Court also has industrial experience in operating system development, networking, device control programming, and embedded systems.

Martin Herbordt received the B.A. degree in Physics and Philosophy from the University of Pennsylvania and the M.S. and Ph.D. degrees in Computer Science from the University of Massachusetts. He is currently Associate Professor of Electrical and Computer Engineering at Boston University. His research interests are in Computer Architecture and include design automation, using configurable circuits for computation, vision architecture, and switch design.

Richard J. Barton received the BA degree in Actuarial Science and Finance, the MS degree in Mathematics, and the PhD degree in Electrical Engineering, all from the University of Illinois. He is currently Assistant Professor of Electrical and Computer Engineering Department at the University of Houston. Dr. Barton's research and teaching interests span many different aspects of statistical signal processing. Past research contributions have been in robust signal detection and estimation, signal detection in the presence of long-term dependent noise, applications of higher-order statistics to modulation design on communication channels, biological sequence analysis, and applications of wavelet transforms and higher-order statistics to pattern recognition and signal classification. His current research interests include the impact of signal dimension and channel uncertainty on wireless communication channel capacity, the development of stochastic approaches to computational electromagnetics for channel modeling and circuit analysis., and applications of statistical signal processing to computational biology systems.