Applying Reconfigurable Hardware to the Analysis of Multispectral and Hyperspectral Imagery

Miriam Leeser[1], Pavle Belanovic[1], Michael Estlick[1],
Maya Gokhale[2], John J. Szymanski[2], and James Theiler[2]

[1]Department of Electrical and Computer Engineering
Northeastern University
Boston, Massachusetts 02115

[2]Space and Remote Sensing Sciences Group
Los Alamos National Laboratory
Los Alamos, New Mexico 87545

ABSTRACT

Unsupervised clustering is a powerful technique for processing multispectral and hyperspectral images. Last year, we reported on an implementation of k-means clustering for multispectral images. Our implementation in reconfigurable hardware processed 10 channel multispectral images two orders of magnitude faster than a software implementation of the same algorithm. The advantage of using reconfigurable hardware to accelerate k-means clustering is clear; the disadvantage is the hardware implementation worked for one specific dataset. It is a non-trivial task to change this implementation to handle a dataset with different number of spectral channels, bits per spectral channel, or number of pixels; or to change the number of clusters. These changes required knowledge of the hardware design process and could take several days of a designer's time.

Since multispectral data sets come in many shapes and sizes, being able to easily change the k-means implementation for these different data sets is important. For this reason, we have developed a parameterized implementation of the k-means algorithm. Our design is parameterized by the number of pixels in an image, the number of channels per pixel, and the number of bits per channel as well as the number of clusters. These parameters can easily be changed in a few minutes by someone not familiar with the design process. The resulting implementation is very close in performance to the original hardware implementation. It has the added advantage that the parameterized design compiles approximately three times faster than the original.

Keywords: hyperspectral, k-means, image processing, algorithm, field-programmable gate array (FPGA)

1. INTRODUCTION

Modern hyperspectral imagers can produce data cubes with hundreds of spectral channels and millions of pixels. One way to cope with this massive quantity is to organize the data so that pixels with similar spectral content are clustered together in the same category. This provides both a compression of the data and a segmentation of the image that can be useful for other image processing tasks downstream.

The classic approach for segmentation of multidimensional data is the k-means algorithm; this is an iterative method that produces successively better segmentations.[1,2] It is a simple algorithm but the computational expense can be considerable, particularly for clustering large hyperspectral images into many categories. Last year we reported on results of the ASAPP (Accelerating Segmentation and Pixel Purity) project in implementing the k-means algorithm in field-programmable gate array (FPGA) hardware.[3]

The standard software implementation of k-means uses floating-point arithmetic and Euclidean distances. By fixing the precision of the computation and by employing the Manhattan distance metric, we were able to fit more distance-computation calculations on the chip, obtain a higher degree of fine-grain parallelism, and therefore faster performance. The price of these optimizations was slightly less optimal clusters. We reported on the use of different

Send correspondence to Miriam Leeser: E-mail: mel@ece.neu.edu

distance metrics and of different levels of precision truncation, and their effect on total within-class variance of a cluster.[3] The hardware implementation presented was designed for 10 channel, 12 bits per channel multispectral images based on MTI datasets. Our results showed that the hardware implementation of k-means ran two orders of magnitude faster than the software version, with an average increase in within-class variance of about 3 percent.

The major drawback of the hardware implementation is that it required in-depth knowledge of hardware design to accomplish. In addition, it works only for a dataset with 10 channels per pixel, 12 bits per channel, and 8 clusters. These parameters were hardwired into the design in many different locations. In addition, a change in one parameter has other effects such as requiring a wider accumulator in another part of the design. Changing these values requires knowledge of hardware design, and may take several days to accomplish.

Multispectral data sets come in many shapes and sizes.[4,5] Our goal is to make the k-means hardware implementation more accessible to an image analyst by making it easier for the analyst to adapt the implementation to different data sets. This translates to the requirement of making the parameters easier to change. An analyst may wish to change the number of clusters, or apply k-means to a different dataset. In this paper, we present a parameterized k-means implementation that allows the user to easily change the parameters without requiring knowledge of the implementation.

The motivation behind parameterizing the k-means algorithm is the vastly different datasets available. For example, MTI, IKONOS and Landsat images have different numbers of spectral bands, pixels, etc. The 10 channel images we use are derived from the AVIRIS data to mock up MTI data. Real MTI data[6] has 4 visible and 12 IR bands, so sometimes an analyst wished to process 4 channels of data, sometimes 12 and sometimes 16. A typical MTI image has roughly 1200×3200 pixels for visible imagery (5m ground resolution), and 300×800 for IR imagery (20m resolution). A number of other sensors are summarized by NASA.[7] IKONOS imagery is 4 spectral bands (R,G,B,NIR) + 1 panchromatic band; it has 1m resolution on the ground. Images are 1000×1000 pixels. Landsat 7 has eight bands: 6 spectral, 1 thermal, and 1 panchromatic. Ground resolution varies from 30m for spectral, 60m for thermal, and 15m for panchromatic. A typical image size is 183×170 km, or over $5K \times 5K$ pixels.[8] Many analysts use subimages when processing this data. Multspectral images come from other sources as well. For example, we have some medical data of cancerous tissue that is 705×601 pixels and 24 channels. There are reasons for clustering each of these data sets. This requires varying the parameters specified.

2. K-MEANS CLUSTERING

Given a set of N pixels, each composed of C spectral channels, and represented as a point in C-dimensional Euclidean space (that is, $\mathbf{x}_n \in \mathcal{R}^C$, with $n = 1, \ldots, N$); we partition the pixels into K clusters with the property that pixels in the same cluster are spectrally similar. Each cluster is associated with a “prototype” or “center” value which is representative of (and close to) the pixels in that class.

One measure of the quality of a partition is the within-class variance; this is the sum of squared (Euclidean) distances from each pixel to that pixel's cluster center.

For a fixed partition, the optimal (in the sense of minimum within-class variance) location for each center is the mean of all pixels in each class. And for a fixed choice of centers, the optimal partition assigns points to the cluster whose centers are closest. The k-means clustering algorithm provides an iterative scheme that operates over a fixed number (K) of clusters, while attempting to simultaneously optimize center locations and pixels assignments.

From an initial sampling, the algorithm loops over all the data points, and reassigns each to the cluster whose center it is closest to. After the pass through the data, the cluster centers are recomputed. Each iteration reduces the total within-class variance for the clustering, so it is guaranteed that after enough iterations, the algorithm will converge, and further passes will not reassign points. It bears remarking that this is a local minimum.

There are many variants of the basic k-means clustering algorithm. In some variants, the number of clusters (K) is altered as part of the algorithm.[9] Other variants of k-means update the cluster centers each time a point is reassigned to a new cluster. Others vary in the way initialization is performed and termination is determined.

We have chosen a k-means algorithm that is particularly well-suited to hardware implementation. K is chosen by the image analyst before the implementation is generated. The algorithm runs by first assigning every point to a new cluster (a complete iteration through all the pixels in the image) and second computing the new cluster centers. Initialization and termination are determined in software, allowing for many different options. For example, the implementation may terminate after no points are reassigned or after a fixed number of iterations. These are standard variants on the basic k-means algorithm.

3. EFFICIENT K-MEANS ALGORITHM FOR HARDWARE IMPLEMENTATION

Our implementation includes other variations that are seldom found in software implementations, but result in more efficient hardware implementation. These include using a different distance metric, and minimizing bitwidths of the input data.

Points are assigned to the cluster centers to which they are closest; for the minimum within-class variance criterion, "closest" is defined in terms of the Euclidean distance. Consider a point $\mathbf{x}$ and cluster center $\mathbf{c}$ where i indexes the spectral components of each. The Euclidean distance is defined as:

$$\|\mathbf{x}-\mathbf{c}\|^2 = \sum_i |x_i - c_i|^2. \tag{1}$$

But other distance measures can also be used; for instance, the general family of p-metrics (for which the Euclidean distance is the special case $p = 2$) is given by:

$$\|\mathbf{x}-\mathbf{c}\|^p = \sum_i |x_i - c_i|^p. \tag{2}$$

To perform a k-means iteration, one must compute the distance from every point to every center. If there are N points, K centers, and C spectral channels, then there will be $O(NKC)$ operations. For the Euclidean distance, each operation requires computing the square of a number.

The Euclidean distance has several advantages. For one, the distance is rotationally invariant. Furthermore, minimizing the Euclidean distance minimizes the within-class variance. On the other hand, the Euclidean distance is much more expensive to implement in hardware. The Manhattan distance, corresponding to $p = 1$, is the sum of absolute values of the coordinate differences; the Max distance, corresponding to $p = \infty$ is the maximum of the absolute values of the coordinate differences. Both these alternative distance metrics were investigated because neither requires any multiplication. In addition, a linear combination of the two was considered.

Data independent and data dependent experiments showed that the Manhattan distance could be used with a small loss in quality of the resulting clusters. The resulting hardware is much more compact than that using the Euclidean distance. This translates to significant speedup in run-times since more parallelism can be implemented.

Another area we investigated was minimizing bitwidths to keep the hardware implementation compact and to increase parallelism. In hardware implementations, the number of bits can be chosen to optimize the size of the hardware design. An advantage of using the Manhattan distance is that the growth in bitwidth in the datapath is smaller than that generated by the Euclidean distance, since the squared values in the Euclidean distance produce twice the bitwidth.

In addition, we investigated the number of bits per channel that influenced the outcome of clustering. Since a large amount of data is being compressed to a very small number of clusters, we hypothesized that much of the input data was not contributing to the result. This hypothesis was borne out in both data independent and data dependent experiments. For AVIRIS images, half the input bits per channel could be ignored with a very small loss in quality, provided a few extra bits were used to represent the cluster means.[10] The result of doing this is the generation of more efficient hardware. The analyst must be able to easily change the number of bits per channel in the implementation to take advantage of this optimization.

4. K-MEANS HARDWARE IMPLEMENTATION

The k-means clustering algorithm consists of two steps: (1)assigning each pixel in the image to one of the K clusters, and (2) recomputing the cluster centers. In order to recompute the cluster centers, not only must the number of pixels assigned to a cluster be computed, but the values of all pixels assigned to each cluster must be accumulated on a channel by channel basis. Our hardware implementation includes step 1 of the algorithm as well as all the accumulation. Cluster centers are computed on the host, but the image data is accessed only by the hardware. Hence, inputs to the hardware implementation are all pixels in the image and the values of all K clusters; outputs are cluster assignments for each pixel and accumulated values for each cluster.

Our design, shown in Figure 1, is fully pipelined. Each clock cycle, a new input pixel is read from memory. The hardware consists of a datapath for computing the nearest cluster center, a pixel shift register to keep the pixel values

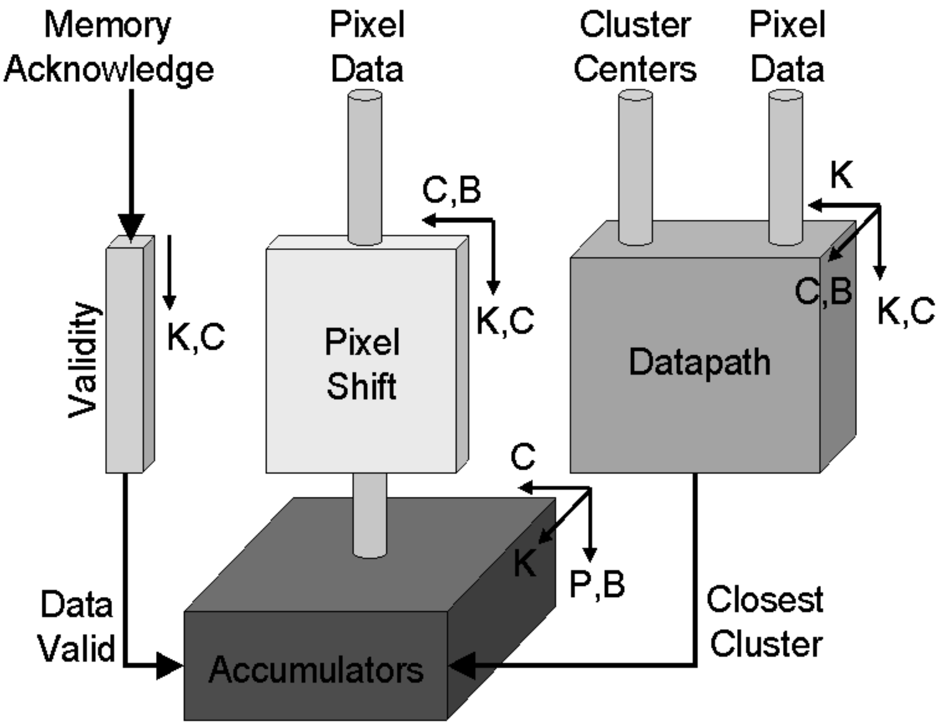

Figure 1. K-means Hardware Design

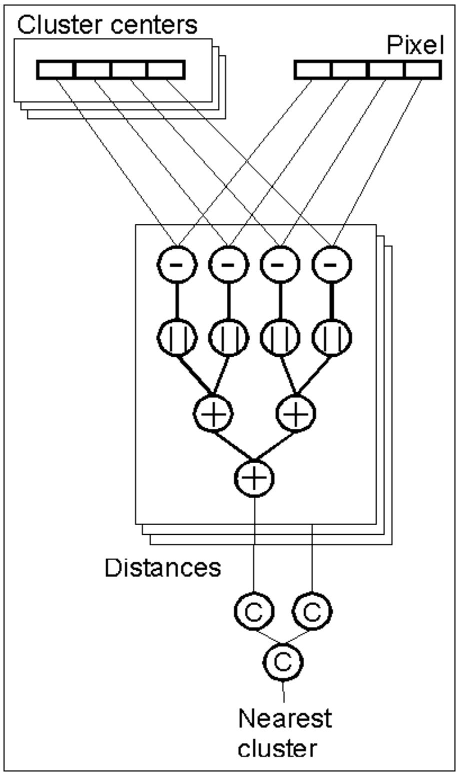

Figure 2. Datapath

with their corresponding cluster center, hardware for keeping track of the validity of the output, and an accumulation block for accumulating the pixel values for pixels assigned to each cluster.

The datapath section of the design assigns each pixel to the closest cluster. One pixel is read from the memory at each clock cycle and placed at the input to the datapath. All spectral channels of the pixel appear at the input to the datapath simultaneously. The other input to the datapath circuit is the center of each of the K clusters. The output of the datapath circuit is a cluster number, indicating which of the K clusters the pixel was assigned to. The Manhattan distance is used.

Figure 2 shows the datapath circuit in more detail. First the distance is calculated in parallel between the pixel and each of the K clusters. Calculating the Manhattan distance involves three operations performed serially: subtraction, absolute value and addition. Then, the results of these K distance calculations are compared to find the minimum. Both the addition of channel distance calculations and the comparison of the total distance calculation require tree structures to compute. The result is the number of the cluster center closest to the pixel.

The accumulators add the pixel value, on a channel by channel basis, to the cluster to which the pixel is being assigned. This requires the output of the datapath circuit as well as the pixel value, which was retrieved from memory for the datapath calculation. The pixel value must appear at the input to the accumulator at the same time as the cluster number to which the pixel is assigned. This is accomplished by the pixel shift circuit, which delays the pixel data the number of pipeline stages of the datapath. The number of pipeline stages depends on both the number of clusters and the number of channels due to the tree structure of the comparison and distance calculation in the datapath.

The validity pipeline is a similar structure to the pixel shift circuit. At the beginning of processing, pixel data is requested from memory. Once data is available, the memory indicates that the data is valid, which triggers the datapath and pixel shift circuits to begin operation. The accumulator circuit must be triggered to begin operation at the end of the datapath operation. This is accomplished by delaying the memory data validity signal the appropriate number of clock cycles.

Pixel values are added to the assigned cluster's accumulated values by the accumulator circuit. The three inputs to the accumulator circuit are the enable signal from the validity pipeline, the pixel value from the pixel shift circuit, and the cluster number from the datapath circuit. Each of the K accumulators has associated with it a counter, keeping track of how many pixels have been added. At the end of a complete iteration of the image, the values of all the accumulators and counters contain the information required to compute the new cluster centers. The cluster numbers for all the pixels processed are stored in memory.

5. PARAMETERIZATION

The size of the design depends on the amount and organization of the data that needs to be processed as well as on the number of clusters. The basic parameters that define the design are K, C, B and P. K is the number of clusters, C is the number of channels per pixel, and B is the number of bits per channel. N is the total number of pixels in the image, and P is a summary of the number of pixels used to parameterize the hardware design. P is defined as: $\lceil \log_2(N) \rceil$. For example, an image with 1024×1024 pixels has $N = 1048576$ and $P = 20$. For an MTI IR band with 300×800 pixels, $N = 240000$ and $P = 18$. The parameters to our implementation are summarized in Table 1.

Parameter	Definition	Source
K	number of clusters	Image Analyst
C	number of channels per pixel	Image
B	number of bits per channel	Image
N	total number of pixels	Image
P	$\lceil \log_2(N) \rceil$	Image

Table 1. Parameters for K-means

To parameterize the design, we mapped its growth in terms of these basic parameters. This applies to the four sub-circuits: datapath, pixel shift, validity pipeline, and accumulator, as well as the signals connecting them.

The datapath circuit is pipelined through D stages. D is defined as $2 + \lceil \log_2(C) \rceil + \lceil \log_2(K) \rceil$. The dependence on the number of channels is due to the adder tree that accumulates the distances over channels; the dependence on K is due to the comparator tree to choose the closest cluster. The complexity of the datapath circuit is parameterized by K as well. There are K distances being calculated in parallel and thus K individual pipelines need to be implemented. The bitwidth of the datapath circuit scales with parameters C and B because each of the pipelines for calculating distance depends on the bitwidth B for the size of subtractors, complementers (for absolute value) and adders, as well as C for the size of the addition tree used to sum all the channels.

The depth of the pixel shift circuit is D, the same as that of the datapath. The width of the pixel shift circuit is naturally characterized by the size of the pixel data that ripples through it. Hence, the width scales according to the number of channels C and the the number of bits in each channel B. The validity pipeline is a cascade of 1-bit registers and thus uses only the depth dimension D.

The accumulator circuit must take into consideration the total number of pixels being processed. The size of the circuit is characterized by the number of channels C because each channel of the pixel is accumulated separately and all channels are processed in parallel. The width of the accumulator circuit scales by K because there is one accumulator in the circuit for each cluster. The depth of the circuit, depends on parameters B and P because each register that accumulates one channel of the pixel varies in size with the number of bits in the channel B, and the total number of bits that can potentially be accumulated into the register. The registers of the accumulators are sized for the worst-case scenario in which all the pixels are assigned to one cluster and every channel of every pixel contains the maximum possible value.

6. RESULTS

Our designs are implemented on a Wildstar PCI board from Annapolis Microsystems.[11] The board has three Xilinx Virtex 1000 FPGAs[12] and 40MB of SRAM. The Wildstar is in a 500MHz Pentium III workstation. The k-means implementations we present use one Virtex 1000. There are two implementations. The first, called the RTL design, is the design presented last year.[3] The second is the parameterized version of the design we have presented in this paper. Both implementations were written in VHDL and synthesized with Synplicity's Synplify and Xilinx Alliance tools. The hardware and software used is commercially available.

The RTL design was implemented in VHDL as a set of processes and uses Synplicity design tools[13] plus Xilinx place-and-route tools[14] to generate a bitstream. The parameterized design also makes use of Synplicity design tools, but creates the design out of components built using the Xilinx CoreGen libraries.[15] Xilinx place-and-route tools are once again used to generate the bitstream. Once the bitstream has been created, it is downloaded to the board, resulting in operational hardware.

Both designs are fully pipelined; one pixel is processed every clock cycle. These implementations assume that a single pixel ($C \times B$ bits) can be read in one clock cycle. For the Annapolis Wildstar board, this translates into a constraint of $C \times B \leq 128$ because 128 bits can be accessed from memory in one clock cycle. To simplify the design, we have also constrained the number of classes to being a power of two.

The design presented last year classifies a 1024×1024 pixel image with 10 channels of 12 bit data per channel into 8 clusters. We compare the performance of the RTL design to the performance of the parameterized design in Table 2. We are interested in area and runtime of the two designs, as well as the place-and-route time, which is the time to create the hardware design.

Design	RTL	Parameterized
Area in slices	9420	8884
Percent area used	76 %	72 %
Total gate equivalents	406,536	433,474
Max Frequency	63.78Mhz	63.07Mhz
Place-and-route time	3:32:56	1:04:49

Table 2. Comparison of RTL and Parameterized Designs

In our results, area is measured by the number of logic slices used on the FPGA. The Virtex 1000 chips have a total of 12288 slices. Typically, some slices are used to route signals instead of as logic. Another measure of area is the number of "gate equivalents" used to implement the design. A gate equivalent is a two-input logic gate; this is a standard measure of area used in digital hardware design. The measure of speed used is the maximum clock frequency. These numbers are generated by the Xilinx place-and-route tools.

The two designs generated with these competing design styles have comparable area and speed. The parameterized design has more gate equivalents packed into fewer Virtex slices; we believe this is due to the fact that the components used in the parameterized design make more efficient use of hardware resources. The parameterized design uses designed and optimized modules provided by the Xilinx CoreGen tools; the RTL design requires the synthesis tool to generate these modules.

The largest difference between the two designs is the ease of generating new implementations using the parameterized methodology, and the fact that the parameterized design has a place-and-route time more than three times faster than the RTL design without a loss of quality in the results. To illustrate the ease of changing parameters, we have implemented k-means clustering on the Wildstar board for the combinations of parameters shown in Table 3.

K	C	B	P
16	5	10	20
8	14	8	20
8	20	6	20
8	10	12	20

Table 3. Combinations of Parameters for which K-means Designs have been Generated

Each new design took a few minutes of setting the parameters plus approximately an hour to place-and-route. Changing the RTL design to generate a new design takes several days of changes as well as three and a half hours of place-and-route time. In addition, it is much easier to introduce errors when changing the RTL code versus the parameterized code.

While these results reflect the use of one Virtex 1000 chip on a Wildstar board, in reality we parallelize the implementation by using two Virtex chips with each processing half the image. This change affects the host code only, and the speedup is the same for both the RTL code and the parameterized code.

7. CONCLUSIONS

In order to accommodate real data sets, we have implemented parameterized k-means in reconfigurable hardware. Our implementation is parameterized by the number of clusters K, the the number of channels, the number of bits per channel and the number of pixels in the image. The performance of the parameterized k-means implementation is very close in area and speed to the non-parameterized version. The main advantage, in addition to the parameterization, is that it generates a hardware solution three times faster. The reason for this speedup is the use of predesigned macros for components such as adders and comparators.

Our current approach is constrained by the memory bandwidth to the Virtex chip on the Wildstar board. We process one pixel each clock cycle, and fully pipeline the processing. In the future, we plan to investigate adding pipelining to our parameterized implementation. In other words, the structure of the pipeline would change depending on the number of clock cycles required to fetch the input data. In addition, we are investigating applying this parameterized approach in other domains, including parameterized floating point modules for reconfigurable hardware.

ACKNOWLEDGMENTS

We would like to thank Annapolis Microsystems for their support, and Synplicity and Xilinx Corporations for their donations to the Rapid Prototyping Laboratory at Northeastern University. We would like to thank researchers at Los Alamos and at Northeastern University who contributed to this project. This research was funded by the US Department of Defense and Department of Energy.

REFERENCES

1. G. B. Coleman and H. C. Andrews, "Image segmentation by clustering," *Proceedings of the IEEE* **67**(5), pp. 773–785, 1979.
2. A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Computing Surveys* **31**, pp. 264–323, 1999.
3. J. Theiler, M. Leeser, M. Estlick, and J. J. Szymanski, "Design issues for hardware implementation of an algorithm for segmenting hyperspectral imagery," in *Image Spectrometry VI*, *Proc. SPIE* **4132**, pp. 99–106, 2000.
4. J. R. Jensen, *Introductory Digital Image Processing: A Remote Sensing Perspective*, Prentice Hall, 1996. Second Edition.
5. R. A. Schowengerdt, *Remote Sensing: Models and Methods for Image Processing*, Academic Press, 1997. Second Edition.
6. Los Alamos National Laboratory, 2000. `http://nis-www.lanl.gov/nis-projects/mti/`.
7. NASA, 2000. `http://geo.arc.nasa.gov/esdstaff/health/sensor/cfsensor.html`.
8. NASA, 2000. `http://landsat.gsfc.nasa.gov/`.
9. Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Communications* **COM-28**, pp. 84–95, 1980.
10. M. E. Leeser, J. Theiler, M. Estlick, N. V. Kitaryeva, and J. J. Szymanski, "Effect of data truncation in an implementation of pixel clustering on a custom computing machine," in *Reconfigurable Technology for Computing and Applications II*, *Proc. SPIE* **4212**, pp. 80–89, 2000.
11. Annapolis Microsystems, Inc., 2001. `http://www.annapmicro.com`.
12. Xilinx Corporation, 2000. `http://www.xilinx.com/products/virtex.htm`.
13. Synplicity Corporation, 2000. `http://www.synplicity.com`.
14. Xilinx Corporation, 2000. `http://www.xilinx.com/products/software/software.htm`.
15. Xilinx Corporation, 2001. `http://www.xilinx.com/products/logicore/coregen/index.htm`.