2000

A Distributed Spectral-Screening Pct Algorithm

Tiranee Achalakul
Syracuse University, tachalak@syr.edu

Stephen Taylor
Syracuse University, steve@scp.syr.edu

A DISTRIBUTED SPECTRAL-SCREENING PCT ALGORITHM

Tiranee Achalakul
2-106 CST building, Syracuse University
Syracuse, NY 13244
Tel: 315-443-2226, Fax: 315-443-2126 tachalak@syr.edu

Stephen Taylor
2-106 CST building, Syracuse University
Syracuse, NY 13244
Tel: 315-443-2134, steve@scp.syr.edu , www.scp.syr.edu

Abstract

This paper describes a novel distributed algorithm for use in remote-sensing, medical image analysis, and surveillance applications. The algorithm combines spectral-screening classification with the principal component transform (PCT), and human-centered mapping. It fuses a multi- or hyper-spectral image set into a single color composite image that maximizes the impact of spectral variation on the human visual system. The algorithm operates on distributed collections of shared-memory multiprocessors that are connected through high-performance networking. Scenes taken from a standard 210 frame remote-sensing data set, collected with the Hyper-spectral Digital Imagery Collection Experiment (HYDICE) airborne imaging spectrometer, are used to assess the algorithms image quality, performance, and scaling. The algorithm is supported with a predictive analytical model that allows its performance to be assessed for a wide variety of typical variations in use. For example, changes to the number of spectra, image resolution, processor speed, memory size, network bandwidth/latency, and granularity of decomposition. The motivation in building a performance model is to assess the impact of changes in technology and problem size associated with different applications, allowing cost-performance tradeoffs to be assessed.

Keyword: Principal Component Transform, spectral Angle classification, distributed algorithm, performance prediction

1. Introduction

Hyper-spectral image fusion is the process of combining images from different wavelengths to produce a unified color-composite image, removing the need for frame by frame evaluation to extract important information. Image fusion can be accomplished using a wide variety of techniques that include pixel, feature, and decision level algorithms [Hall 1992]. At the pixel level, raw pixels can be fused using image arithmetic, band-ratio methods [Richards and Jia 1998], wavelet transforms [Li et al. 1995)], maximum contrast selection techniques [Peli et al. 1999], and/or the principal/independent component transforms [Gonzalez and Woods1993, Mackiewicz 1993, Lee 1998]. At the feature level, raw images can be transformed into a representation of objects, such as image segments, shapes, or object orientations [Hall 1992, 1997]. Finally, at the decision level, images can be processed individually and an identity declaration used to fuse the results [Hall 1992, 1997]. Most of these fusion techniques have been used on a small number of images where they are said to be particularly effective [Richards and Jia 1998, Hall 1992]. The most notable exception is the Principal Component Transform (PCT) which has been employed in a variety of remote sensing applications. In our research we are particularly interested in fusing a large number of spectra and therefore base our work on the PCT.

The PCT is used to summarize and de-correlate a collection of multi- or hyper-spectral images. It operates by removing redundancy and packing the residual information into a smaller set of images, termed *principal components* [Mackiewicz 1993, Singh 1985&1993]. The first three principal components capture the primary spectral

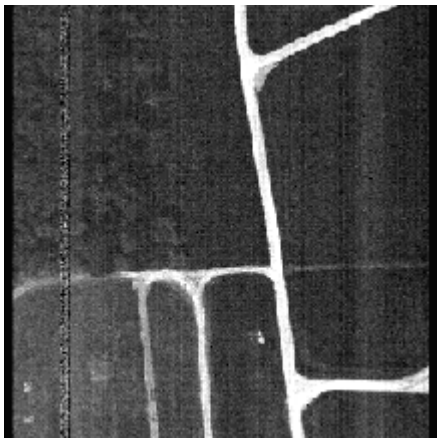
information and are typically used to create a color composite image through an appropriate color-mapping scheme. Unfortunately, in its basic form the algorithm tends to highlight variations that dominate numerically. This has the effect of enhancing the importance of an object that occurs frequently in a scene, for example trees in a forest. As a result, the variations associated with features that occur infrequently, for example a mechanized vehicle in the forest, are lost.

This paper describes and evaluates a novel *distributed spectral-screening PCT algorithm* that extends our previous work on shared-memory multiprocessors to the domain of distributed systems [Achalakul et. al. 1999]. The new algorithm combines the Principal Component Transform (PCT) with spectral angle classification [Kruse et al. 1993] and human-centered color mapping [Boynton 1979, Peterson et al. 1993, Poirson and Wandell, 1993]. Spectral angle classification has the effect of treating aspects of an image that occur frequently with same importance as those that occur infrequently. For example, all trees in a forest would be placed in an equivalence class and considered of equal importance to the class of mechanized vehicles. The human-centered color mapping attempts to match the spatial-spectral content of the output image with the spatial-spectral processing capabilities of the human visual system. This has the effect improving the visual presentation of the data by enhancing important color variations with direct stimulation of the retina.

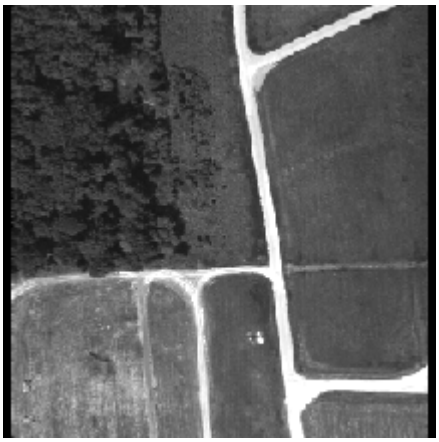
To demonstrate the algorithm, it was applied to a 210-channel hyper-spectral image collected with the Hyper-spectral Digital Imagery Collection Experiment (HYDICE)

sensor, an airborne imaging spectrometer. These images correspond to foliated scenes taken from an altitude of 2000 to 7500 meters at wavelengths between 400nm and 2.5 micron. The scenes contain mechanized vehicles sitting in open fields as well as under camouflage. Figure 1 shows a single hyper-spectral image via a representative sample of frames picked from the 210 spectral bands. Notice that at the 524nm there is an image with significant contrast on the forestry and camouflaged vehicles, however, since this image is hidden in a data set of 210 frames an automated method is required to extract the information without frame-by-frame inspection.

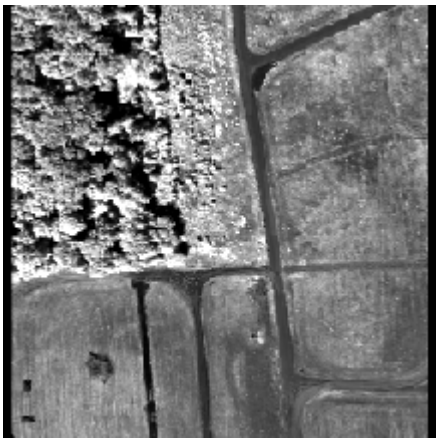
Figure 2 shows the resulting color composite image obtained through the spectral screening PCT. Almost 80% of the variance is pushed into the first principal component and after the first three components there is no significant variance. Thus, it is possible to use only these three bands to generate the final resulting image. Figure 2a demonstrates a standard false color mapping in which the first principal component is mapped to red, the second to green, and the third to blue. Figure 2b shows the alternative human-centered mapping, which maps the first principal component to achromatic, the second to red-green opponency, and the third to blue-yellow opponency. The latter picture, when viewed on a high-quality monitor, shows significantly improved contrast levels. The forested areas show enhanced detail and the camouflaged vehicle in the lower left corner is significantly enhanced against its background. Postprocessing steps can subsequently be applied to detect edges in the image and use structural information to detect and classify the vehicles.



a) 400 nm

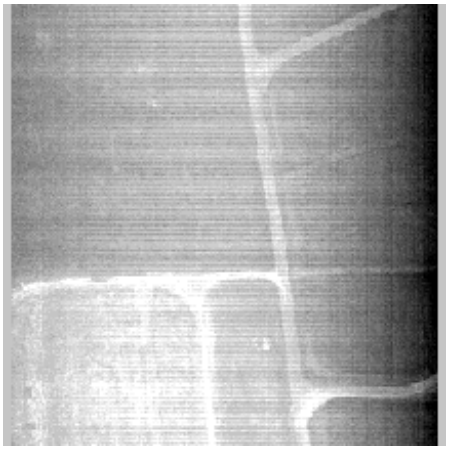


b) 452 nm



c) 524 nm

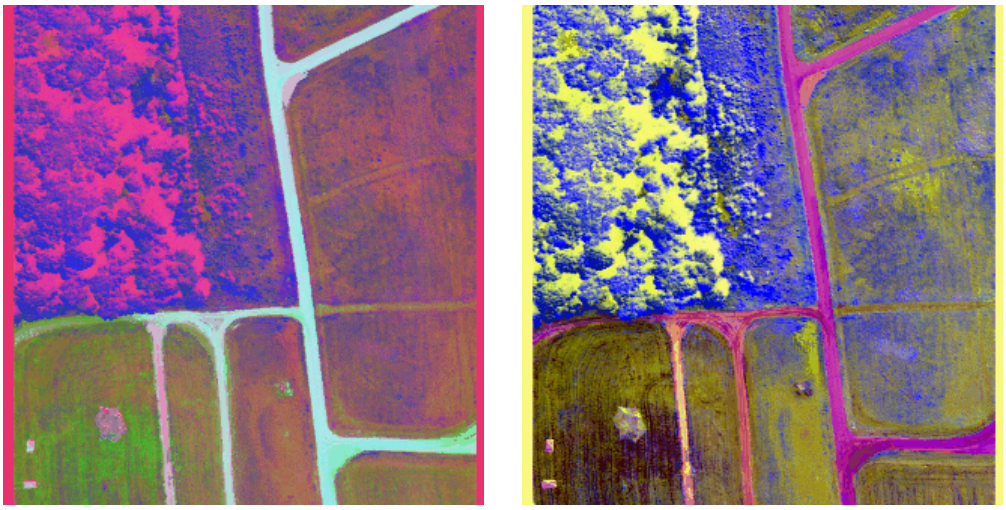
d) 700 nm



e) 997 nm

f) 1998 nm

Figure 1: A set of sample frames from the original hyper-spectral image



a) False color mapping R= pc1, G=pc2, B=pc3.

b) Human-Centered Color mapping method.

Figure 2: Color-Composite Image

Both spectral angle classification and PCT have high computational costs. The spectral angle classification requires the computation of a dotproduct for every pair of pixel vectors in a hyper-spectral image, in the worst case $O(n^2)$ vector operations. Moreover, unlike Fourier, Walsh, or Hadamard transforms, the PCT transformation matrix is not separable, and thus, no high performance uniprocessor algorithm exists [Pardalos et al. 1992]. These performance requirements discourage use of the techniques in real-time applications.

To increase performance we are exploring concurrent algorithms employing low-cost, commercial-off-the-shelf multi-processors connected using high-performance (gigabit) networking. To assess the limitations of the approach an analytical model is presented here that quantifies the expected performance and scalability. The model is validated,

using linear-regression against experimental data that characterizes the gross behavior of the algorithm, in the style of Foster et. al. [Foster 1996]. Performance predictions are made based on reasonable expectations of future technology and typical variations of problem definition, e.g. increases in processor speed, number of processors, network bandwidth, image set size, and image resolution.

2. Concurrent Algorithm

The concurrent algorithm decomposes the three-dimensional cube structure of a multi-spectral image into sub-cubes, as shown in Figure 3, that can be operated on relatively independently. Each sub-cube consists of a set of pixel vectors $x_{ij}=[x_1, x_2, \ldots, x_n]$ similar to the decompositions used in [Palmer et al. 1998]. The allocation of sub-cubes to processors is managed through a variant of the manager/worker technique depicted in Figure 4 [Chandy and Taylor 1992]. This strategy employs a sensor thread that represents the interface to multi-spectral hardware, performs the above decomposition, and distributes sub-cubes to a set of worker threads. Each worker performs relatively independent components of the overall image transformation and associated color mapping techniques. A manager thread coordinates the actions of the workers, gathers partial results from them, assembles the final color composite image, and provides access to display hardware. Although the results in this paper were produced from static multi-spectral files, rather than sensor hardware, the structure of the algorithm can be operated in real-time [Taylor 2000].

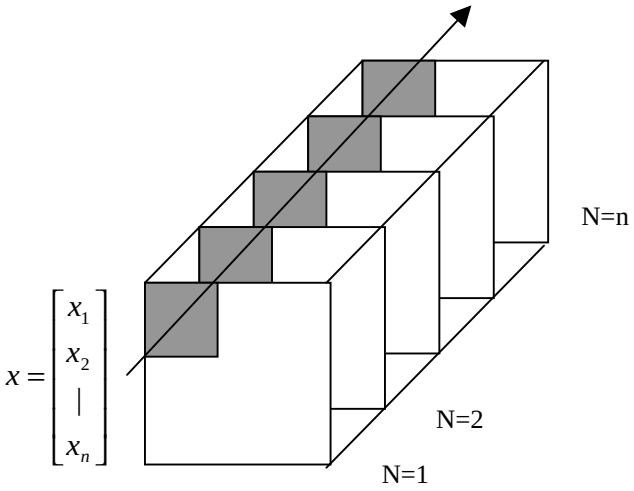


Figure 3: Domain Decomposition.

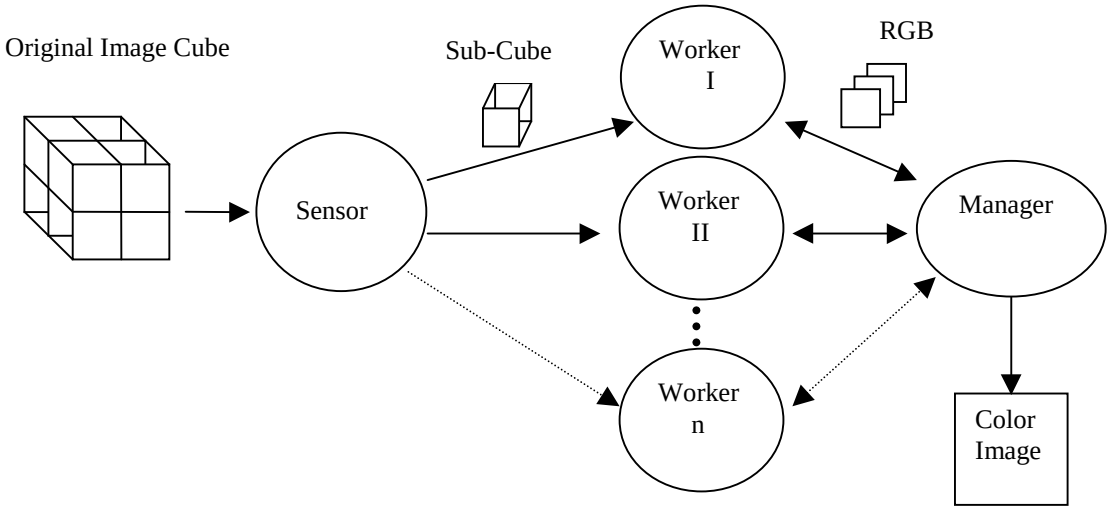


Figure 4: Manager/Worker Communication Model

The main abstract code of the algorithm is shown in Program 1 and is executed at every processor on the network. For example, if there are 3, 8-way multiprocessors, the program is executed 24 times. The sensor, manager, and workers are executed as independent threads with a single thread per processor.

```
main() {
   mp = get_my_multiprocessor_id()
   if(mp == 0) {
         numsubcubes = get_num_subcubes()
         sensor(numsubcubes)
         manager(numsubcubes)
   }
   foreach remaining available processor
         worker()
}
```

Program 1 Communication Structure.

Abstract code for the sensor is shown in Program 2. It repeatedly obtains multi-spectral image cubes from the sensor (1), waits for an appropriate request for work from a worker (2), decomposes the image cube to generate an unassigned sub-cube (3) and sends the sub-cube to the requesting worker (4).

```
sensor() {
   while(sensor device operating) {
         cube = grab_cube()                          /* 1 */
         while(subcubes available) {
               request = recv(aworker)               /* 2 */
               work = generate_subcube(cube)         /* 3 */
               send(aworker, work)                   /* 4 */
         }
   }
}
```

Program 2: Sensor Thread Operation

Each worker thread executes the algorithm shown in Program 3 and maintains a set of sub-cubes (1,4) to operate on. An initial request is sent to the sensor to obtain the first sub-cube (2). After this initial request, the processing of each sub-cube is overlapped with communication of the remaining the next sub-cube from the sensor (3). This represents the primary communication step in the algorithm and corresponds to distributing $1/n^{th}$ of the image cube to each of n-multiprocessors.

```
worker() {
   cubes = {}                                              /* 1 */
   send(request,sensor)                                    /* 2 */
   while(numsubcubes <= numcubes/numworkers) {
         subcube = recv(sensor)                            /* 3 */
         cubes = cubes U subcube                           /* 4 */
         send(request,sensor)                              /* 5 */
         ssubset = spectral_screening(subcube)             /* 6 */
         send(ssubset, manager)                            /* 7 */
   }
   sset = recv(manager)                                    /* 8 */
   substats = statistics(sset)                             /* 9 */
   send(manager, substats)                                 /* 10 */
   [A, m] = recv(manager)                                  /* 11 */
   subcomponents = PCT(A, m, cubes)                        /* 12 */
   subimage = human_centered_mapping(subcomponents)        /* 13 */
   send(subimage, manager)                                 / * 14 */
}
```

Program 3: A Worker Thread

The spectral screening algorithm produces a set of unique spectra. Although each sub-cube contributes to this set through an appropriate abstract operation (6), the set must be accumulated across all sub-cubes. This accumulation is performed through communication with the manager. Each worker sends a prospective subset of the spectra to the manager (7) and overlaps this communication with computation of the next subset.

When all sub-cubes have been processed, the manger transmits the resulting unique set to all workers (8). Typically, the amount of communication in this step is orders of magnitude less than the size of an image cube.

When the spectral screening is completed globally, the algorithm proceeds to compute a set of statistics (mean-vector and covariant-sum) that give a measure of the variation in images at each spectra. Although, once again, the statistics can be largely computed on a per sub-cube basis using an appropriate abstract operation (9), the manager is again involved in assembling the statistics to form a transformation matrix A and mean-vector m (10,11). The communication involved in this step is on the order of n^2 where n is the number of spectra, again typically significantly smaller than the size of the image cube.

With the matrix A and mean-vector m available, the PCT (12) and human-centered mapping (13) can be computed on each sub-cube independently to produce a patch of the final color image. The patches are accumulated at the manager for display (14). Thus, the final communication is only m^2, where m is the size of the image.

Program 4 shows the abstract code of the manager, which serves primarily to synchronize and accumulate partial results from the workers. It is given here for completeness, although it involves no significant numerical technique other than the calculation of the transformation matrix. Note that the method by which a single point of synchronization is typically avoided in a distributed algorithm is through replication and global communication. As will be seen later from the performance model, the organization of a

large number of processors into a significantly smaller number of multiprocessors, connected with Ethernet technology, does not make replication an attractive alternative. We have explored this alternative and found that in practice, it is less efficient than the more simple structure given here for practical problem sizes.

```
manager(numsubcubes) {
   sset = {}
   stats = [][]
   image = [][]
   foreach subcubes of numsubcubes {
         ssubset = recv(aworker)
         sset = sset U ssubset
   }
   foreach worker i
         send(sset, i)
   foreach worker {
         substats = recv(worker)
         stats = merge(stats, substats)
   }
   [cov, m] = stats
   A = eigenvectors(cov)
   foreach worker i
         send([A, m], i)
   foreach worker {
         subimage = recv(worker)
         image = merge(image, subimage)
   }
   display(image)
}
```

Program 4 A Manager Thread

Spectral Angle Classification is a technique that measures the similarity between the spectral signatures of objects in a scene. In a 2-band hyper-spectral space, the similarity

between two signatures can be determined by calculating the angle between the two associated pixel vectors X and Y as shown in Figure 5(a). The spectral signatures can then be separated from one another if there is a sufficient difference in their angles as shown in Figure 5(b).

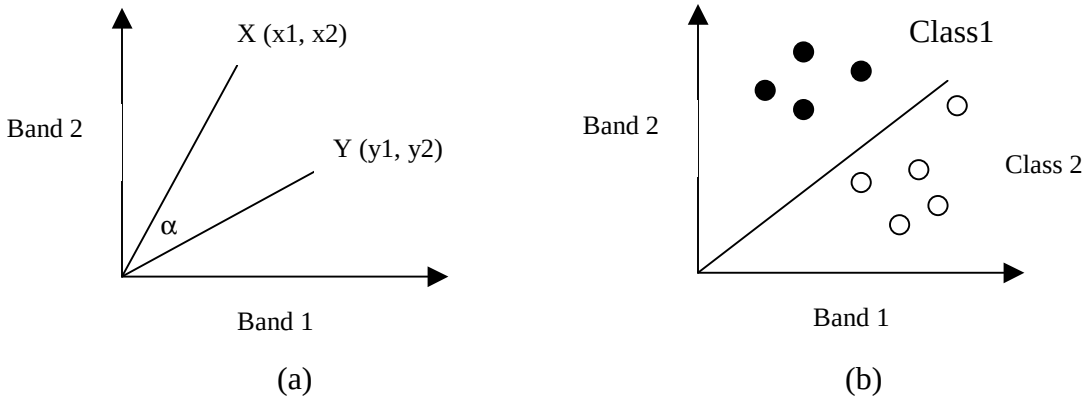


Figure 5: (a) Spectral angle for a two bands image. (b) Classifying spectral space.

Extending this concept from two bands to n-bands, the calculation of the spectral angle can be performed by the following equation that operates on two n-dimensinal pixel vectors.

$$\alpha(x,y) = \cos^{-1}\left(\frac{x \bullet y}{\|x\| \bullet \|y\|}\right) = \cos^{-1}\left(\frac{\sum_{i=1}^{n} x_i y_i}{\left(\sum_{n=1}^{n} x_i^{\,2} \sum_{n=1}^{n} y_i^{\,2}\right)^{1/2}}\right)$$

Program 5 shows the abstract code for the spectral screening process. For a given spectral angle threshold, α_{thr}, a set of unique spectral signatures is formed by calculating the spectral angle between all the pixel vectors in a hyper-spectral image using the above

equation. The unique signature set is initially empty. Each pixel vector is compared to all of the vectors in the unique set by calculating the associated spectral (2). If all the angles exceed the threshold (3), the pixel vector is added into a set (4); otherwise it is discarded. On completion of the process, a unique set of spectral signatures is determined in which the spectral angle between every pair of pixel vectors is greater than the threshold, α_{thr}. This unique set is then used, instead of the entire collection of pixel vectors in the hyper-spectral image, in the spectral de-correlation process. By adding this screening method, we are assured a variation that dominates numerically (backgound) in the original hyper-spectral image, will not dominate the resulting image; small objects in the scene will have an equal chance of being pushed into the foremost principal components.

```
spectral_screening(subcube)
{
   S = {}                                          /* 1*/
   for each v_i in a subcube {
         for all v_j in S {
                  α(i, j) = cos^-1(i • j / ||i|| • ||j||)   /* 2 */
                  if(all (α(i, j)) > α_thr )               /* 3 */
                     S = S U {v_i}                         /* 4 */
         }
   }
}
```

Principal Component Transform treats each source image as a matrix and forms the associated covariance matrix, which characterizes variations in image contrast. The covariance matrix is then used to form, through a linear transformation, a collection of

principal components that effectively summarize the variations across all spectra. The output components carry enough spectral frequency information to reconstruct the original multi-spectral images. The components are rank ordered by the magnitude of their variances (eigenvalues); therefore, most of the spectral contrast is pushed forward to the first few components. The linear transformation thereby permits identification of information that might not be apparent in any single image, or simple linear combination of images that are selected empirically. The PCT algorithm can be divided into two parts that calculate the transformation matrix A, and subsequently transform the data. Consider the pixel vector of the form

$$x = \begin{bmatrix} x_1 \\ x_2 \\ | \\ x_n \end{bmatrix}$$

The mean vector can be defined as

$$m_x = \frac{1}{K}\sum_{k=1}^{K} x_k$$

where K is the number of pixels in an original image set.

The covariance matrix of the n-spectral band image can then be calculated as follows:

$$C_x = \frac{1}{K}\sum_{k=1}^{K} x_k x_k^T - m_x m_x^T$$

Because C_x is real and symmetric, finding a set of n orthonormal eigenvectors is always possible [Noble 1969]. The transformation matrix, A can then be formed by lining the sorted eigenvectors calculated from the covariance matrix in each row. The fist row of

matrix *A* is the eigenvector corresponding to the largest eigenvalue, and the last row is the eigenvector corresponding to the smallest eigenvalue. Program 6 shows the abstract code for the PCT algorithm in which the multi-spectral image, *I*, is transformed into a set of principal components, *PC*.

```
statistics(sset)
{
        m = 0;
        for  all pixel i in sset
                m = m + i;
        m = m / k;       // where k = number of vectors in sset
        cov = 0;
        for all pixel i in uset {
                C_i = I_i I_i^T − mm^T ;
                cov = cov + C_i;
        }
        substats = [cov, m]
}

eigenvectors(stats)
{
        eigvector, eigvalue  = find_eigvector(stats);
        eigvector  =  sort(eigvector, eigvalue)
        A = [eig_1 | eig_2 | … | eig_n]
}

PCT(A, m, cubes)
{
        for all pixel vector V_i in cubes
                PC_i = A(V_i – m);
}
```

Program 6: Principal Component Transform

Human-Centered Color Mapping assigns the first three Principal Components, which have the maximum variance, to a standard representation of the human color space and

subsequently converts this representation to RGB values than can be used to drive a color display. A large number of color spaces have been proposed in the literature of color vision [Boynton 1979]. In this paper, we choose to work with the luminance/chrominance model, or YOZ model, favored by Peterson et al. 1993. The response of the three cones in the human visual system can be transformed into a Luminance band (Y) and two color-opponent bands: red-green (O), and blue-yellow (Z). The information bandwidth of the human color channels is unequal. The spatial frequency bandwidth of the Luminance channel is much greater than the color opponent channels [Poirson and Wandell, 1993]. This suggest that mapping the first Principal Component into the luminance channel and the second and third Principal Components into chromatic channels of the visual system will provide an efficient utilization of the human visual bandwidth.

The YOZ color space is derived from the standard chromacity coordinates termed XYZ, developed by Commission Internationale de l' Eclairage (CIE) in 1931 using the following empirically derived transform given in Peterson et al. 1993:

$$[YOZ] = [XYZ]\begin{bmatrix} 0 & 0.47 & 0 \\ 1 & -0.37 & 0 \\ 0 & -0.10 & 1 \end{bmatrix}$$

The luminance channel Y is just the CIE Y-channel and the blue-yellow opponent channel Z is just the CIE Z-channel. The red-green opponent channel (O) is given by the equation, O = 0.47X – 0.37Y – 0.10Z.

To obtain the appropriate mapping from YOZ to RGB, we follow the work Boynton. The color space-mapping matrix, k, is derived from the measured spectral power distribution of the display (i.e. intensity at each wavelength) and an empirical color matching function as follows [Boynton 1979]:

$$k = [T\ P]^{-1}$$

The color matching function T is an n-by-3 matrix where each column is determined by having human observers match their color primaries to spectral test lights at different wavelength. Matrix P is a 3-by-n matrix representing the measured spectral power distribution of the primaries. In our experiments, we have used the YOZ color matching functions for matrix T and the spectral power distribution of a typical RGB monitor for matrix P. The normalization of matrix k is shown below:

$$k = \begin{bmatrix} 0.4387 & 0.4972 & 0.0641 \\ 0.4972 & -0.1403 & -0.0795 \\ -0.1355 & 0.0116 & 0.4972 \end{bmatrix}$$

Differential YOZ input values are used because a negative O value indicates green color. The final equation for YOZ to RGB mapping can thus be stated as follows:

$$[RGB] = (128 + (k\_final * ([YOZ] - 128))) / 256$$

3. Predictive Model

Recall that the motivation in building a performance model is to assess the impact of changes in technology and problem size associated with different applications, allowing cost-performance tradeoffs to be assessed. Many performance-modeling techniques have been presented in the literature for analyzing the performance of concurrent algorithms. Some of the most interesting include statistical, simulation, analytical, and benchmarking models. Each model has its own advantages and suits a specific type of application [Fahringer 1996]. In our work we are primarily concerned with predicting the performance scaling characteristics on a variety of architectures. We therefore choose to analyze the Concurrent Spectral-Screening PCT algorithm by forming an analytical model based on weighting factors that are calibrated experimentally [Foster 1996, Rieffel 1998]. This method uses a linear equation to describe the gross behavior of the algorithm executed on a multi-processor. It allows parallel speedup on a given machine to be predicted and provides the ability to assess crucial concurrent performance bottlenecks. It is also possible to estimate the number of processors needed to complete the task, given some particular time restriction.

Speedup and Efficiency. The basic notations used in performance measurement are *speedup (sp)* and *efficiency (e)* [Pardalos 1992, Foster 1996]. Speedup is defined as the ratio of the time required by the concurrent algorithm to complete the task using one processor to the time required when P processors are used. If P is the number of processors, Ts is the time used to solve the problem sequentially, and T_o represents the sum of the overhead of each processor, speedup can be defined as

$$sp = \frac{T_s}{\frac{T_s + T_o}{P}}$$

or

$$sp = \frac{T_s P}{T_s + T_o}$$

The ideal speedup (or maximum speedup obtainable) is *P*. The parallel efficiency can also be calculated by measuring the processor utilization in solving a problem. The efficiency, e, is defined as the ratio of useful work to the total work, or the ratio of the sequential time to the product of the parallel time and number of processors:

$$e = \frac{sp}{P} = \frac{1}{1 + \frac{T_o}{T_s}}$$

Concurrent Analytical Model. The total time for concurrent execution in each processor, T_{conc} is the sum of computation time, communication costs and idle time in each processor.

$$T_{conc} = T_{comp} + T_{comm} + T_{idle}$$

The average computation required in each processor, T_{comp} is equal to the time used to solve the problem sequentially, T_s, divided by number processors in the system, *P*.

$$T_{comp} = \frac{T_s}{P}$$

Idle time occurs in only the fastest computers. The total execution time, T_t can then be defined as the sum of computation and communication time of the slowest processor and the time used to compute sequential steps in the algorithm, T_{sq}.

$$T_t = T_{comp} + T_{comm} + T_{sq}$$

and the efficiency of the algorithm can thus be modeled as follows:

$$e = \frac{sp}{P} = \frac{T_s}{T_{conc} P} = \frac{T_{comp}}{T_{conc}} = \frac{T_{comp}}{T_{comp} + T_{comm}} = \frac{1}{1 + T_{comm} / T_{comp}}$$

Communication Model. To a first order, communication costs can be divided into two parts: the time used to transfer messages into the interconnection network, and the time used for messages to travel through the network. The former cost depends on the speed of communication hardware and software of each processor. The latter cost depends on how processors are connected. In our experiments, we are primarily interested in low-cost, high-performance local area networks based on switched-Ethernet, 100BaseT and Gigabit. The communication time T_{comm} can be modeled as followed

$$T_{comm} = T_o + T_p$$

where T_o is the message overhead and T_p is the transport time.

The message overhead includes communication latency and the time used for synchronization. The transport time includes the time used to format and transfer

messages. The transport time is the product of message size (in bytes) and network throughput, T_w (transport time per byte).

In our experiments, modern high-performance network switches were used to connect multiprocessors. With this technology, several multiprocessors can send and receive messages without compromising the network throughput. Thus, assuming the total data of size N is to be divided evenly among P Processors, the communication can be described in the following equation:

$$T_{comm} = T_o + T_w \frac{N}{p}$$

Computation Model. To develop the computation model, we need to be able to determine the computational complexity of each step in a concurrent algorithm. The complexity of a step is taken to be the time used to complete the step as a function of the problem size [Cormen 1990] and is expressed using weighting factors C_1 through C_8 that represent the relative importance of each step. Recall that the computation time, T_{comp} is defined as

$$T_{comp} = \frac{T_s}{P}$$

In the concurrent algorithm, the original hyper- or multi-spectral image cube is decomposed into a set of sub-cubes where each sub-cube is distributed to a worker. The sequential time, T_s, can then be, defined as follows:

$$T_s = kT_b$$

Where k is the number of sub-cubes and T_b is the time use to compute one sub-cube.

Let *m* be width and height of each sub-cube in the hyper-spectral image, *n* be the number of spectral band, *s* be the number of unique spectra per sub-cube, and *p* be the number of processors. Considering each component of the algorithm in turn:

1. **Spectral Screening:** The computation associated with this step involves a calculation taken over all pixel vectors concurrently, m^2 at each worker. Each computation (the arccosine of dotproduct of pixel vectors pair) involves the calculation between a new vector (of size n) and all vectors in the unique set (s). Thus the time required, T_1, is:

$$T_1 = C_1 m^2 sn$$

2. **Merge Unique Sets:** This step is computed sequentially at the manager. The computation involves an angle calculation associated with each pixel vector (of size n) in p-1 sets, where each set contains s pixel vectors. The time required, T_2, is:

$$T_2 = C_2 (p-1) sn$$

3. **Mean vector:** This step involves taking an average of the pixel values in a unique spectral set at the manager. The number of operations is related to the number of unique spectra (s) and the number of frame (n). The time required, T_3, is:

$$T_3 = C_3 sn$$

4. **Covariance sum**: The computation associated with the covariance sum is performed over the pixels in a unique set of size s at the worker. Each computation on a pixel involves matrix multiplication (complexity of n^2). The time required, T_4, is:

$$T_4 = C_4 n^2 s$$

5. **Covariance Matrix:** This computation involves forming the matrix sum of the matrices returned from the previous steps at the manager. There are p matrices of size nxn. The time required, T_5, is:

$$T_5 = C_5 n^2 p$$

6. **Transformation matrix:** The time used in this step is dominated by the time used to calculate eigenvectors at the manager. The time required, T_6, is:

$$T_6 = C_6 n^3$$

7. **Transformation of the Data:** The computation in this step is performed over the pixels in an image of size m^2. Each computation on a pixel involves matrix multiplication with the complexity of n^2, at the worker. The time required, T_7, is:

$$T_7 = C_7 n^2 m^2$$

8. **Color mapping:** This step of the algorithm involves linear transformation of the first three principal components in achromatic, red-green, and blue-yellow opponency at the worker. The time required, T_8, is directly proportional to the size of sub-cube:

$$T_8 = C_8 m^2$$

The total time to compute one sub-cube, T_b, is thus $T_1 + T_3 + T_4 + T_7 + T_8$. The total time for sequential computation T_{sq}, is $T_2 + T_5 + T_6$. The total execution time for an n-band image cube of size *mxmxp*, can then be defined as:

$$T_t = T_{comp} + T_{comm} + T_{sq} = \frac{kT_b}{p} + T_o + T_w \frac{km^2 n}{p} + T_{sq}$$

The performance model can thus be described as:

$$T_t = \frac{k}{p}(C_1 m^2 sn + C_2 sn + C_3 n^2 s + C_4 n^2 m^2 + C_5 m^2) +$$

$$C_6(p-1)sn + C_7 n^2 p + C_8 n^3 + C_9 T_w \frac{km^2 n}{p} + T_o$$

The parallel efficiency can also be predicted with:

$$e = \frac{1}{1 + T_{comm} / T_{comp}} = \left[1 + \frac{T_o + C_9 T_w km^2 n / p}{\frac{k}{p}(C_1 m^2 n + C_2 sn + C_3 n^2 s + C_4 n^2 m^2 + C_5 m^2} \right]^{-1}$$

Model parameters: The analytical model developed in the previous section describes the performance of the concurrent algorithm in terms of the number of spectra, the image size, and network bandwidth. To calibrate the model and assess the relative importance of each phase of the algorithm, it is necessary to assign values to the weighting factors C_1 through C_8. In addition, we add two values, T_o to represent the synchronization overhead, and C_9 to represent any additional computation required to format data for a communication device.

In our experiment, two different network technologies are used: 100BaseT and gigabit networking. On the gigabit network, the time used to transfer one byte through the network T_w was measured at 0.002 microsecond. On a 100 baseT network the T_w was measured at 0.008 microsecond.

A naïve method to calibrate C_1 through $C_{8,}$ C_9, and T_o is to run ten experiments, obtain the total time for each in terms of known values for n, m s, k, p, and T_w, and solve the resulting equations simultaneously. Unfortunately, this approach was found to be inadequate because the behavior cannot be accurately represented by a linear combination of the variables. Instead, we utilize linear regression [Hogg 1989] and apply the least-square fitting method with the data acquired from experiments to designate values for the weighting factors. The least-square tries to fit a curve as closely as possible to a set of points on a plane. Our model is a linear equation of the form, $y = a_0 + a_1 x_1 + a_2 x_2 + ... + a_n x_n$. To apply the least square fitting method, the following equation is used:

$$v = (M^T M)^{-1} M^T y$$

where

$$y = \begin{bmatrix} y_1 \\ y_2 \\ : \\ y_n \end{bmatrix}, M = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^m \\ 1 & x_2 & x_2^2 & x_2^m \\ : & : & : & : \\ 1 & x_n & x_n^2 & x_n^m \end{bmatrix}, v = \begin{bmatrix} a_0 \\ a_2 \\ : \\ a_m \end{bmatrix}$$

and where m=9 represents the number of weighting factors and n=25 is the number of experiments executed to resolve these factors.

The associated experiments were performed on three Intel multiprocessors, each running Windows NT. Each multiprocessor has 8 processors running at 550Mhz each. The experiments varied the size of the source image, the number of processors, the number of spectral bands, the network connection, and the granularity of the decomposition. After approximately 25 experiments no significant variations in the value of the weighting factors were obtained and the final values are listed below.

$$T_o = 8.8756,\ C_1 = 7.2833e\text{-}009,\ C_2 = -6.2733e\text{-}005,\ C_3 = -5.2628e\text{-}007,$$

$$C_4 = 4.1329e\text{-}008,\ C_5 = -4.8906e\text{-}005,\ C_6 = 1.6035e\text{-}005,\ C_7 = -1.6350e\text{-}005$$

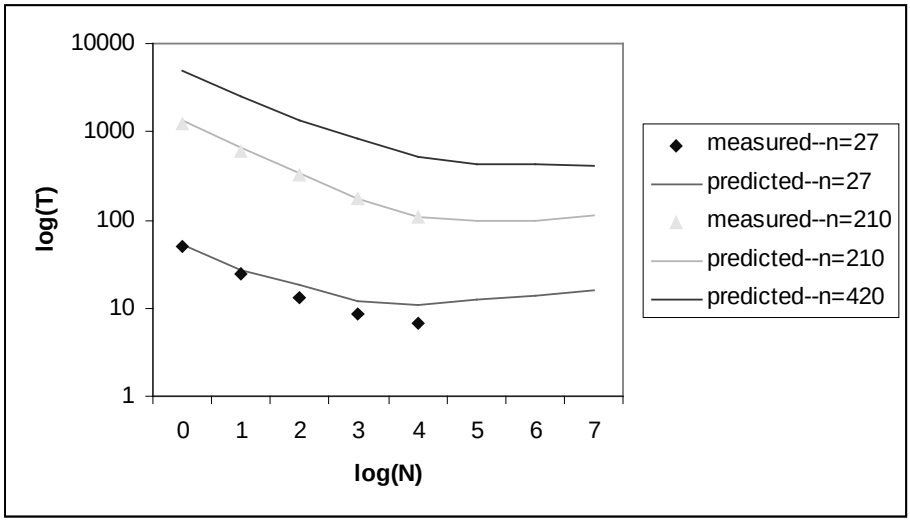
$$C_8 = 8.0959e\text{-}006,\ C_9 = 15.8635$$

4. Performance Result

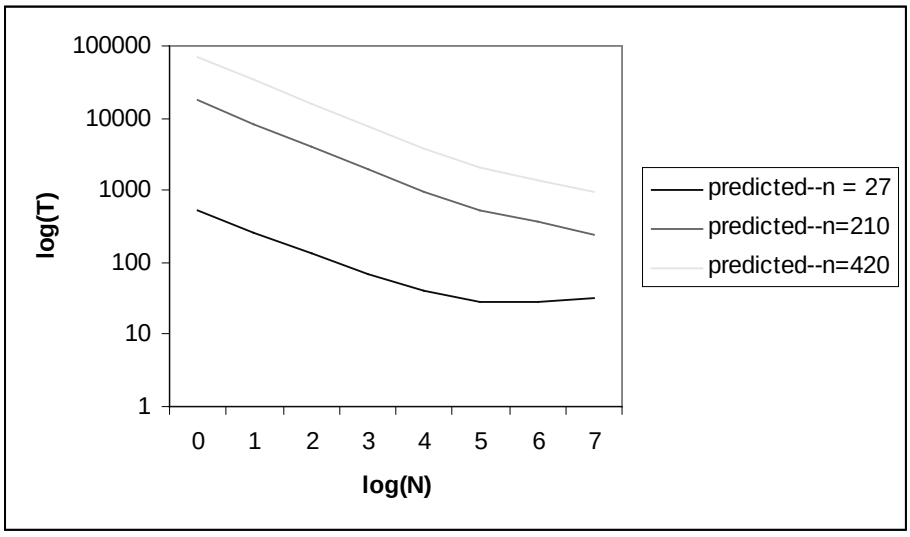
In this section we study the algorithms scaling properties for all of the primary variations of interest, comparing measured and predicted performance results. The results are a small but representative sampling of a much more broad range of experiments that we have conducted to validate the model.

Variations in Problem Size. There are two application specific properties associated with problem size: the *number of spectra n* and the *image resolution m*. The performance of the concurrent algorithm was measured on the gigabit network with 24 processors

arranged in three multiprocessors, as described in the previous section. Figures 6a and 6b plot the measured and predicted execution time as a function of the number of processor *p*, where possible, experiments were based on the HYDICE data set, with 320x1280 resolution and up to 210 spectra. Each plot shows the impact of variations in the number of spectra, image resolution is varied between the plots.



a) Problem size: 320x1280



b) Problem size: 2048x2048

Figure 6: Varying Problem Size

For large problems, e.g. 2048x2048x420, the algorithm performs within 20% of linear speedup using 64 processors, the efficiency drops below 0.75 at 96 processors, and below 0.9 at 48 processors. For medium sized problems, e.g. 320x1280x210 spectra, the algorithm performs within 20% of linear at 16 processors, the efficiency drops below 0.75 at 16 processors, and below 0.9 at 8 processors. For small problem sizes, e.g. 320x1280x27, the algorithm performs within 15% of linear speedup using 8 processors, the efficiency drops below 0.75 at 8 processors, and below 0.9 at 4 processors.

In general, the performance drop from linear speedup decreases as the problem size increases. The dominant issue is problem size. For small problems, there is not sufficient computation to gain an impact from a large number of processors – there is simply not enough work to keep the processors busy. As a result, the performance gain begins to drop off as the number of processors increase. Note that Step 6 of the algorithm which involves sequential code to compute the eigenvectors of the covariance matrix, is not a significant factor in overall performance (5%). Hence, there were no extensive efforts to optimize this step through concurrent execution. The complexity of the eigenvector calculation is related to the number of spectra *n* used in the problem. Although the eigenvector algorithm has a complexity of $O(n^3)$, the time used does not dominant with typical problem sizes. This is because the performance of Steps 1, 3, 4, and 7 are also related to the number of spectral bands; these steps dominate Step 6 as the number of spectra increases.

At small problem sizes, since there is not a large enough computation to warrant large numbers of processors, an alternative approach to concurrency would be more beneficial for real-time applications: to *multi-process in time rather than space*. This alternative is the current focus of our research efforts in extending the work in this paper.

The experiments demonstrate that the accuracy of the predictive model is within 10% for large problem sizes. For a small problem sizes the predicted time can be as much as 25% off of the measured time, but the general trend is correct. The error in the model is likely to be an artifact of the regression method coupled with additional operations, such as buffer management, that are not yet accurately reflected in the model.

Variations in Processor Speed. Figure 7 plots the measured and predicted performance of the algorithm for the medium sized HYDICE data set with 275MHz, 550MHz processors and 1.1GHz processors. With a small number of processors, the performance gained is almost double when the processor speed is doubled. When a large number of processors are used, the performance gain is reduced from linear by 10% at 128 processors. This is due to the computation/communication ratio. The communication time was measured at 5% of the computation time at 1 processor. The overhead increases to 15% when 128 processors are used.

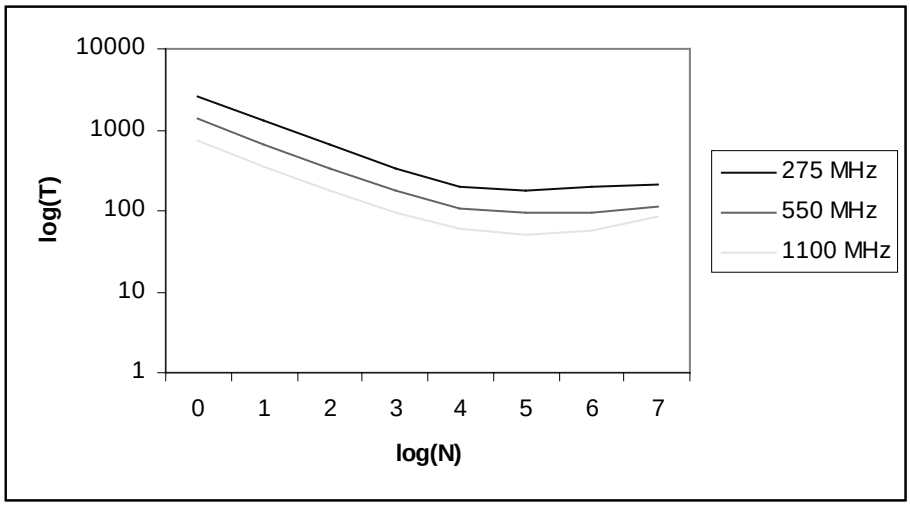
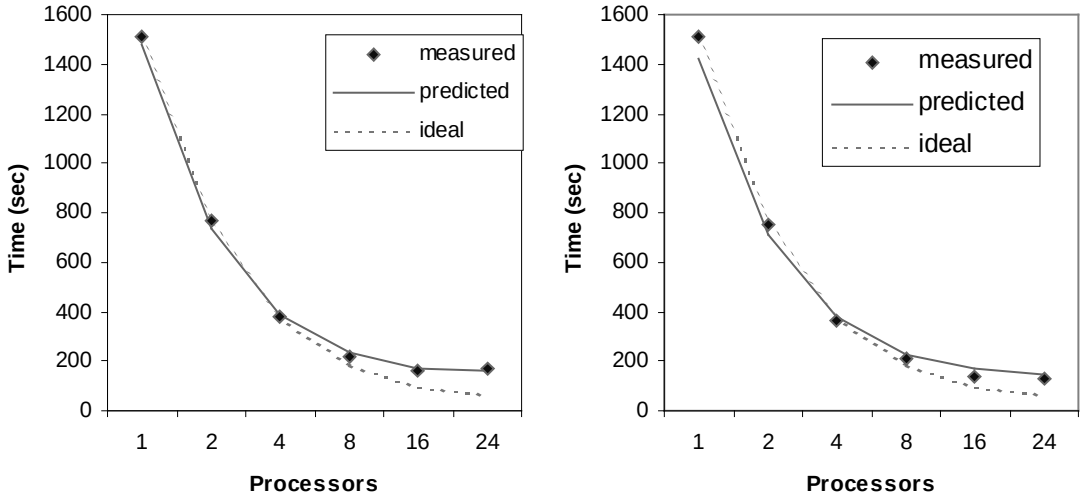


Figure 7: Varying Processor Speed

Variations in Network Performance. Figure 8 plots the measured and predicted performance of the algorithm on the medium sized HYDICE data set, with two different networking technologies: 100BaseT and Gigabit Ethernet. In general, a 15% performance improvement is gained when a gigabit network is used instead of 100 BaseT. Notice that the model is more accurate for the 100BaseT experiments. This is largely due to unpredictable contention in the gigabit networking. Although the average throughput was measured at 350 Megabit/sec, the actual speeds realized in experiments varies considerably. Using 100BaseT connection, the processors have no problem keeping up with the network speed so the actual performance is closer to the average throughput.



a) 320x1280x210, 100 BaseT b) 320x1280x210, gigabit

Figure 8: Varying Network Performance

Variations in Granularity. The granularity of problem decomposition is the ratio of computation to communication. Increasing the granularity should reduce the overhead of communication, but conversely limit scalability [Chandy and Taylor 1992]. In this algorithm the size of a sub-cube allocated to each worker for processing provides a mechanism to control granularity. Figure 9 examines performance using four different decompositions. The results show that dividing an image cube into a considerably larger number of sub-cubes than the number of processors (e.g. 3 times the number of processors) improves performance. The performance improvements stem from the ability of the algorithm to *overlap* computation and communication, thus reducing communication overhead and increasing overall throughput. When the granularity is too fine, the computation on each sub-cube becomes too small, and communication overhead

dominates. With a problem size of 320x1280 at 1 processor, the standard deviation is 7% off of the mean. With 24 processors, the standard deviation calculated at 3% off of the mean. This shows that the granularity of decomposition has more effect when a smaller number of processors are used. In this experiment, the performance difference is up to 4% when the decomposition was more than n=48 sub-cubes. This indicates that, for this problem size, using more than 24 computers will not buy substantial performance improvement. The general effect is more pronounced in larger image sets. With the problem size of 2048x2048 at 1 processor the standard deviation is 8% off of the mean and 7% at 24 processors. This indicates that with this problem size, the image cube can be further divided into finer granularity.

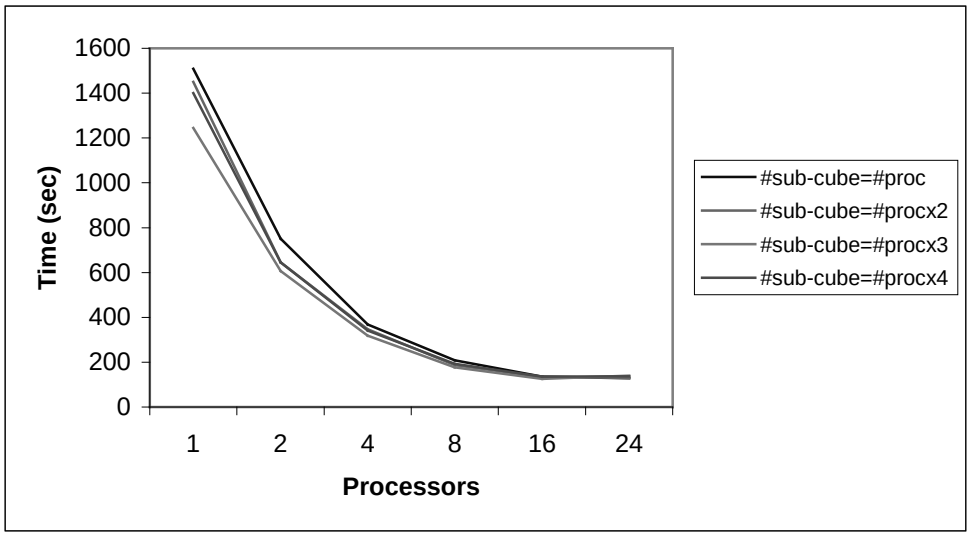


Figure 9: Varying Granularity Decomposition

Although quantifying the performance of the algorithm, the primary result from this set of experiments is that the presented model can be used to provide a first order analytical method for assessing the impact of changes in technology and problem size. This allows

a system to be designed that trades off system cost for performance on a particular application. Using the model a wide range of practical design questions can be answered, for example:

- *For a given fixed cost, what performance can be expected from the algorithm?*
- *How fast will the algorithm operate if the processor speed doubles?*
- *What network speed will realize my cost – performance objectives?*
- *What granularity will maximize the performance on a particular system configuration?*

5. Conclusion

This paper has described a Concurrent spectral-screening PCT algorithm and its associated analytical model for performance prediction. The algorithm has been applied to a typical remote sensing application for camouflage detection. The analytical model was validated against a large set of experimental data. Given a problem size and a time constraint, the model can be used to estimate the number of processors needed to achieve the required performance. In the near future, COTS-multiprocessors with 16 processors or more, where each processor runs at 1000 MHz, will be available. Using a network of 8 of these machines (128 processors), the remote sensing problem size of 210 frames of 1024 by 1024 pixels can be solved 414.05 seconds. We are currently developing a real-time multi-spectral camera system for use in low-altitude Ariel photography. This system provides a stream in 12 spectra. With the emerging technology we could expect one 16-ways multiprocessors machine to process an image cube with 12 frames in

approximately 0.1 sec at 1024x1024 resolution. We expect a network of such machines to enable real-time image fusion for surveillance applications.

6. References

1. Achalakul T., Haaland P., Taylor S., "MathWeb: A Concurrent Image Analysis Tool Suite for Multi-spectral Data Fusion", Sensor Fusion: Architectures, Algorithms, and Applications III, Orlando, FL, Vol. 3719, pp 351-358, April 1999.
2. Boynton T. M., *Human Color Vision*, Rinehart, and Winston, New York, 1979.
3. Chandy L. M., *Taylor S., An Introduction to Parallel Programming*, Jones and Bartlett publishers, Boston, 1992.
4. Cormen T. H., Leiserson C. E., Rivest R. L., *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.
5. Fahringer T., *Automatic Performance Prediction of Parallel Programs,* Kluwer Academic Publishers, Boston/London, 1996.
6. Foster I., Gropp W., and Stevens R.. "The Parallel Scalability of the Spectral Transform Method", Monthly Weather Review, 1992.
7. Gonzalez R. C., Woods R. E., *Digital Image Processing*, Addison-Wesley Publishing Company, Inc., 1993.
8. Hall D.L, "An Introduction to Multisensor Data Fusion," Proceedings of The IEEE, Vol. 85, No. 1, January 1997, pp. 6-23.
9. Hall D.L., *Mathematical Techniques in Multisensor Data Fusion*, Boston, MA:Artech House, 1992.

10. Hogg R. V., Tanis E. A., *Probability and Statistical Inference*, Macmillan Publishing Company, New York, 1989.
11. Kruse F. A., Lefkoff A. B., Boardman J. W., Heidebrecht K. B., Shapiro A. T., Barloon P. J., and Goetz F. H., "The spectral Image Processing System (SIPS) – Interactive Visualization and Analysis of Imaging Spectrometer Data", Remote Sensing Environment vol 44, 1993, pp 145-163.
12. Lee T., *Independent Component Analysis: Theory and Applications*, kluwer Academic Publishers, Boston, 1998.Li H., Manjunath B. A., Mitra S. K., "Multisensor Image Fusion Using the Wavelet Transform," Graphical Models and Image Processing, Vol. 57, No. 3, May 1995, pp. 235-245.
13. Li H., Manjunath B. A., and Mitra S. K., Multisensor Image Fusion Using the Wavelet Transform, Graphical Models and Image Processing, 57, (1995), 235-245.
14. Mackiewicz A. and Ratajczak W., "Principal Components Analysis (PCA)", Computers & Geosciences, vol.19, 1993, pp303-342.
15. Noble B., Apply Linear Algebra, Prentice-Hall, Englewood Cliffs, NJ, 1969.
16. Palmer M., Totty B., Taylor S., "Ray Casting on shared-Memory Architectures: Efficient Exploitation of the Memory Hierarchy", IEEE Concurrency, Vol 6, No. 1, pp 20-36, 1998.
17. Pardalos P. M., Phillips A. T., Rosen J. B., *Topics in Parallel Computing in Mathematical Programming*, Science Press, New York/Beijing, 1992.
18. Peli T., Peli E., Ellis K., Stahl R., "Multi-Spectral Image Fusion for Visual Display", SPIE vol. 3719 Sensor Fusion: Architectures, Algorithms, and Applications III, 1999, pp359-368.

19. Peterson H. A., Ahumada A. J., and Watson A. B., “An Improved Detection Model for DCT Coefficient Quantization”, SPIE, vol. 1913, 1993, pp. 191-201.
20. Poirson A. B., Wandell B. A., “Appearance of Colored Patterns: Pattern-color Separability”, Journal of the Optical Society of America, vol. 10, 1993, pp 2458-2470.
21. Richards J. A., and Jai X., *Remote Sensing Digital Image Analysis: An Introduction*, New York, NY: Springer, 1998.
22. Rieffel M., *Performance Modeling for Concurrent Particle Simulations*, Ph.D. Thesis, California Institute of Technology, 1998
23. Singh A. and Eklundh L., “A Comparative Analysis of Standardised and Unstandardised Principal Components Analysis in Remote Sensing”, International Journal of Remote Sensing, vol. 14, 1993, pp1359-1370.
24. Singh A. and Harrison A., “Standardized Principal Components”, International Journal of Remote Sensing, vol. 6, 1985, pp 883-896.
25. Taylor S., Achalakul T., Lee J., Lhee K., Robila S., “Resilient Remote Sensing”, National Symposium on Sensor and Data Fusion, San Antonio, TX, June 2000.