International Journal of Intelligent Information Systems
2014; 3(6-1): 10-14
Published online October 17, 2014 (http://www.sciencepublishinggroup.com/j/ijiis)
doi: 10.11648/j.ijiis.s.2014030601.12
ISSN: 2328-7675 (Print); ISSN: 2328-7683 (Online)

Parallel image processing using algorithmic skeletons

Sare Eslami Khorami

Islamic Azad University South Tehran Branch, Tehran, Iran

Email address:
sare.eslami@gmail.com

To cite this article:
Sare Eslami Khorami. Parallel Image Processing Using Algorithmic Skeletons. *International Journal of Intelligent Information Systems.* Special Issue: Research and Practices in Information Systems and Technologies in Developing Countries. Vol. 3, No. 6-1, 2014, pp. 10-14.
doi: 10.11648/j.ijiis.s.2014030601.12

Abstract: In the last few decades, image processing has achieved significant theoretical and practical progress. It has been so fast that image processing can be easily traced in several disciplines and industries. At present, various methods have been proposed to implement image processing. The present paper aims to present a technique for image processing which utilizes design and analysis of parallel algorithms. It employs a new approach called "algorithmic skeletons" which is composed of a set of programming templates; hence facilitating the programmers' work.

Keywords: Image Processing, Algorithmic Skeletons, Face Detection and Recognition

1. Introduction

Image processing has two major sections: image enhancement and machine vision. Image enhancement includes methods such as using fading filters and increasing contrast to enhance image visibility, picture thickness, and to make sure they are correctly displayed in the target environment (such as a printer or a monitor)[15]. Machine vision involves methods to understand the content of images to be used in area such as robotics and axis images [17]. Parallel processing has accelerated processing such as computing speed of computer systems [1]. As image processing is capable of parallelism, this paper addresses image processing by using parallelism.

2. Algorithmic Skeletons

Algorithmic skeletons are general models for parallel programming [1]. It provides a programming language which is not only simple and independent of machine architecture but also highly efficient. Skeletons are algorithmic patterns used in parallel programming. They are usually integrated with a host language and they are considered the only source of parallelism in that language. For example, mapping skeletons run a function on all items in a list in parallel. FARM skeletons implements master-slave in parallel or D&C represents divide and conquer parallelism [16]. A task is recursively sub-divided until a condition is met, then the sub-task is executed and results are merged while the recursion is unwound. The most important feature of a skeleton is its generality that is the ability to be used in different applications [1]. In most algorithmic skeletons, there are a set of functions that has to be defined by the user. Once defined by the user, these functions are compiled into a specific location in the skeleton and then they are executed after a pre-processing. Usually every skeleton has a performance model which is presented as a mathematical formula and the user is able to predict the performance time by compiling the related parameters in this model.

3. Classification and Skeletonization of Image Operations

Image processing operations can be classified as low-level, intermediate-level and high-level (Table 1); Based on this classification, it is possible to define a skeleton library for image operations.

Table 1. *Image operations.*

Image operations	Source	Output
Low-level	Image	Image
Intermediate-level	Image	Object/vector-data
High-level	Object/vector-data	Object/vector-data

3.1. Low-Level Image Operations

Low-level image processing operations use the values of image pixels to modify individual pixels in an image. They can be divided into point-to point, neighborhood-to-point and global-to-point operations [5]. Point-to-point operations depend only on the values of the corresponding pixels from the input image and the parallelization is simple. Neighborhood operations produce an image in which the output pixels depend on a group of neighboring pixels around the corresponding pixel from the input image. Operations like smoothing, sharpening, filtering, noise reduction and edge detection are highly parallelizable. Global operations depend on all the pixels of the input image, like Discrete Fourier Transform (DFT) and they are also parallelizable.

3.2. Intermediate-Level Image Operations

Intermediate-level image processing operations work on images and output other data structures, such as detected objects (e.g., faces) or statistics, thereby reducing the amount of information. Operations such as Hough transform [10] (to find a line in an image), center-of-gravity calculation[11], labeling an object[12], are examples of intermediate-level image operations. They are more limited from the aspect of data parallelism when compared to low level operations. They can be defined as image to-object operations.

3.3. High-Level Image Operations

High-level image processing operations work on vector data or objects in the image and return other vector data or objects. They usually have irregular access patterns and thus are difficult to run data parallel. They can be divided into object-to-object or object-to-point operations. Position estimation [13] and object recognition theory [14] are examples of this category.

3.4. Skeletons for Image Operations

It is possible to use the data-parallelism paradigm with the master-slave approach for low level, intermediate-level and high-level image processing operations. A master processor is selected for splitting and distributing the data to the slaves. The master can also process a part of the image (data). Each slave processes its received part of the image (data) then, the master gathers and assembles the image (data) back. Based on the above observation, we identify a number of skeletons for parallel processing of low-level, intermediate-level and high-level image processing operations. They are named according Eto the type of the operator. Headers of some skeletons are shown in code 1.

```
//skeleton for point to point operations
void PixelToPixelOp(E_IMG *in, E_IMG
*out,void(*op)());
//skeleton for neighborhood to point operations
void NeighborToPixelOp(E_IMG *in, E_IMG *out,
E_WIN *win,void(*op)());
//skeleton for global to point operations
void GlobalToPixelOp(E_IMG *in, E_IMG *out,
void(*op)());
//skeleton for image to object operations
void ImageToObject(E_IMG *in, E_OBJ *out,
void(*op)());
//skeleton for object to object operations
void ObjectToObject(E_OBJ *in, E_OBJ *out,
void(*op)());
//skeleton for object to point/value operations
void ObjectToPoint(E_OBJ *in, E_Point *out,
void(*op)());
```

Code 1. Skeleton library

Each skeleton can be executed on a set of processors. From this set of processors, a host processor is selected to split and distribute the image to the other processors. The other processors from the set receive a part of the image and the image operation which should be applied to it. Then the computation takes place and the result is sent back to the host processor. The programmer of the image processing application should only select the skeleton from the library and returns the appropriate operation as a parameter.

4. Face Detection and Recognition

For recognizing a face from an image, first, it is necessary to separate it from the image, and then, it should be recognized from a data base of known faces. So, the face recognition process can be divided in two parts:

4.1. Face Detection

For detecting faces, we have proposed an algorithm [3] by searching for the presence of skin tone colored pixels or groups of pixels. We have used the YUV color domain, because it separates the luminance (Y) from the true color (UV). In the RGB color space, the components represent not only color but also luminance, which varies from one situation to another (due to the fact that changing light causes the reliability to be decreased). By using the YUV color domain, not only the detection has become more reliable but also the skin-tone identification has become easier, because the skin tone can now be indicated in a 2−dimensional space. By measuring the UV values of human skin-tone, the skin-tone region has been identified as a rectangle in the UV spectrum (Figure 1) and every non-skin color out of the ”skin box” is seen as non face (Figure 2).

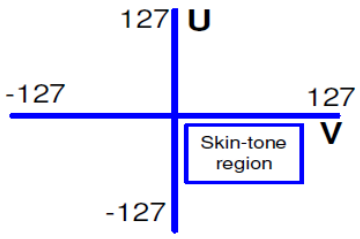

Figure 1. *Skin region in UV spectrum.*

Figure 2. *Skin-tone result.*

The face (skin) detection part separates the skin-tone from the image and sends only the luminance and the coordinate of the skin to the recognition part. It should be mentioned that the recognition part distinguishes faces from other parts of the body such as hands and feet that have the same skin color. Furthermore, if an image is coded in the RGB color space, it is first converted to YUV.

4.2. Face Recognition

The next step of the process is the recognition part. Through this process, an area of skin, detected in the previous step, is identified with respect to a face database. For this purpose, a Radial Basis Function (RBF) neural network is used [6]. The reason for using an RBF neural network is its ability to cluster similar images before classifying them [4]. An RBF neural network structure is demonstrated in Figure 3.

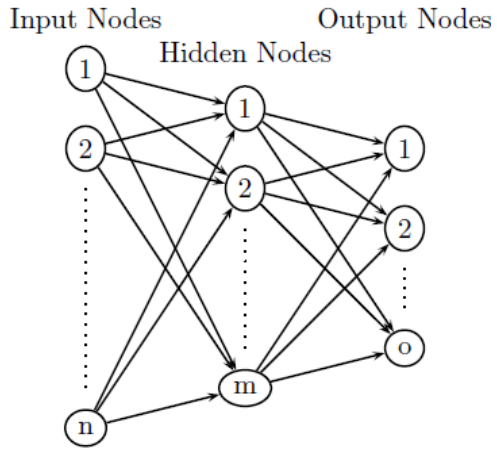

Figure 3. *Architecture of RBF neural network*

Its architecture is similar to that of a traditional three layer feed forward neural network. The input layer of this network is a set of n units, which accepts the elements of an n-dimensional input feature vector. (Here, the RBF neural network input is the face which is gained from the face detection part. Since it is normalized to a 64 * 72 pixel face, it follows that n = 4608.) The input units are completely connected to the hidden layer with m hidden nodes. Connections between the input and the hidden layers have fixed unit weights and, consequently, it is not necessary to train them. The purpose of the hidden layer is to cluster the data and decrease its dimensionality. The RBF hidden nodes are also completely connected to the output layer. The number of the outputs depends on the number of people to be recognized (o equals the number of persons plus one according to Figure).

The output layer provides the response to the activation pattern applied to the input layer. The change from the input space to the RBF hidden unit space is nonlinear, whereas the change from the RBF hidden unit space to the output space is linear.

For the recognition part, a skin area should be fed to the neural network input. Subsequently, the output should be calculated for each person from the database. The network node has one output node for each person from the database and the maximum value between the output nodes is considered to be the recognized person. For distinguishing a face from other parts of the body and from noise, we have preserved one of the outputs of the neural network.

5. Skeletonizing

This section shows how it is possible to skeletonize image processing applications via a skeletons library. According to Section 4, face recognition can be divided into two main tasks:

Detecting skin in the image, which can be further dived into two parts:

Finding the skin-tone in the image; we can map this part of the program as low-level image processing operations, because the input of this part is an image and the output is also an image.

Separating the skin-tones from the image as objects, and determining the coordinates of each of these skin-tones. So we map this part as intermediate level image processing operations, because the input is an image and the output is a set of objects (faces).

Sending each of the skin-tones (faces) to the neural network for identification, according to the faces which are in the data base. We map this part as high-level image processing operations, because the input is an object and the output is the number of the recognized person.

Code 2 shows the C-code of face recognition.

```
/*find skin tone*/
for (y=1; y < HEIGHT-1; y++){
for (x=1; x < WIDTH-1; x++){
/* convert color */
Convertcolor(R[x][y],G[x][y],B[x][y],
U[x][y],V[x][y]);
/* find skin-tone */
if( MIN_U<U[x][y]&&U[x][y]< MAX_U &&
MIN_V<V[x][y]&&V[x][y]< MAX_V)
out[x][y] = 1;
else out[x][y] = 0;        }
}
/*label the image*/
for (y=1; y < HEIGHT-1; y++)
```

```
for (x=1; x < WIDTH-1; x++)
if(out[x][y])
label(label[x][y]);
/*Neural network*/
for (h=0;h< HIDDEN_NODE; h++){
out_hidden[h]=0;
for (i=0;i< INPUTE_NODE; i++){
out_hidden[h] += input[i]*i2h_weight[h][i];
}
out_hidden[h] = ActiveFunc(out_hidden[h]);
}
for (o=0;o< OUTPUT_NODE; o++){
out_rbf[o]= 0;
for (h=0;h< HIDDEN_NODE; h++){
out_rbf[o] += out_hidden[h]*h2o_weight[o][h];
}
}
person = 0;
max = 0;
for (o=0;o< OUTPUT_NODE; o++){
if( out_rbf[o] > max){
max= out_rbf[o];
person = o;
}
}
```

Code 2: Face recognition

The main parts of the program are the parts which are inside the loops and they have the same operations for each pixel in an image or for each object (face). For being able to bring data parallelism into the program, we use skeletons as mentioned in Code 1. The code can be divided into the following tasks:

- Convert color: Since in our setup input is in RGB, for detecting the skin tone in the UV domain, the values of U and V should be calculated for each pixel.
- Binarization: For each pixel, it should be checked whether it is within the skin-tone box or not (see Figure 3).
- Labeling: For separating the faces from the image, the same label should be assigned to pixels which are nearby in the skin-tone.
- Neural network: The neural network for recognizing the objects which are detected in the previous part.

The main function of the skeletonized code is shown in Code 3 where (the first three tasks are mapped onto the first three skeletons; and the neural network is mapped onto the second three skeletons).

```
PixelToPixelOp(RGB, UV, &yc2ycbcr);
PixelToPixelOp(UV, skin, &Binarization);
ImageToObject(skin, obj, &labeling);
for( i= 0; i < num_object; i++){
ObjectToObject(obj,hidden, &NeuralNet_hiddennode);
ObjectToObject(hidden,out, &NeuralNet_outputnode);
ObjectToValue(out, person, &Find_max);
}
```

Code 3: Main function of skeleton code for face recognition

6. Evaluation and Discussion

We have implemented the skeletons library for the IMAP-board. Each implemented skeleton follows a standard template: first, the control processor reads the image or data from the external memory; Then, it distributes the data between the PEs; After that, it sends the determined operations (instructions) from the skeletons to the PEs; Finally, it gathers the result from the PEs and writes it in the external memory.

Table 2 shows the execution time for each skeleton in the program. The image size that we have used is 256 * 240 pixels and the neural network that we have used has 4608 input nodes, 15 hidden nodes and 6 (5 persons + 1 noise) output nodes.

Table 2. *execution time.*

Skeleton	Time(ms)
PixelToPixelOp(RGB, UV, &yc2ycbcr)	1.9
PixelToPixelOp(UV, skin, &Binarization)	1.75
ImageToObject(skin, obj, &labeling)	1.58
ObjectToObject(obj, hidden, &NeuralNet _hiddennode)	2.1
ObjectToObject (hidden, out, &NeuralNet_outputnode)	0.567

We have also implemented a manually optimized version of face recognition (without using the skeletons) on the IMAP-board. The difference is that each skeleton reads the image (object) from the external memory, distributes it between the PEs, and again stores the results in this external memory, whereas in the optimal solution it is not always necessary to read and write data from/to the external memory. We have measured the execution time for reading, distributing and gathering an image (256 * 240 pixels) for each skeleton and it is 0.16ms. The average time for running each skeleton is 1.58ms (Table 2).

Consequently, the execution time for sending and gathering an image, takes 11% of skeleton execution time (0.16/(1.58−0.16) = .11). Note that in general, it is not necessary to send/collect the entire image to/from a skeleton (For instance, for the neural network, it's only necessary to send the skin-tone region). From these measurements, we may deduce that in general the execution time for skeletonized code is in the order of 10% worse than the execution time of an optimized program (on the IMAP-board and assuming that similar types of skeletons are used in the application).

For the face recognition case study, the skeletonized code takes 8.21ms and the optimized code takes 7.8ms, which is an overhead of approximately 5%. Based on this initial experience, we expect that skeletonization can be used as a very convenient programming and implementation method which does not result in an excessive execution time overhead. It relieves the programmer from many tedious low level implementation and parallelization details.

References

[1] H. Gonz´alez-V´elez, M. Leyton, "A Survey of Algorithmic Skeleton Frameworks: High-Level Structured Parallel Programming Enablers," in Research Monographs in Parallel and Distributed Computing. MIT Press, 2008.

[2] Aldinucci, M.; Danelutto, M.; Antoniu, G.; Jan, M. "Fault-Tolerant Data Sharing for High-level Grid: A Hierarchical Storage Architecture". Achievements in European Research on Grid Systems, 2008.

[3] Wang, Q., Wu, J. Long, C. Li, B, "P-FAD: Real-time face detection scheme on embedded smart cameras ," in Distributed Smart Cameras (ICDSC), 2012 Sixth International Conference,2012.

[4] Y. Hu and J. Hwang, Handbook of neural network signal processing. CRC Press, 2002.

[5] C. H. Chu, E. J.Delp, L.H. Jamieson,H. J. Siegel, F. J.Weil, and A. B. Whinston, "A model for an intelligent operating system for executing image understanding tasks on a reconfigurable parallel architecture," Journal of Parallel and Distributed Computing, vol. 6, pp. 598–662, June 1998.

[6] J. Haddadnia, K. Faez, and P. Moallem,"Human face recognition with moment invariants based on shape information," in Proceedings of the International Conference on Information Systems, Analysis and Syn-thesis, vol. 20, (Orlando, Florida USA), International Institute of Informatics and Systemics(ISAS), 2001.

[7] Mario Leyton, Jose M. Piquer. "Skandium: Multi-core Programming with algorithmic skeletons", IEEE Euro-micro PDP 2010.

[8] G. Yaikhom, M. Cole, S. Gilmore, and J. Hillston. "A structural approach for modelling performance of systems using skeletons." Electronic Notes in Theoretical Computer Science, 190(3):167–183,2007.

[9] N. Zhang, Y. Chen, W. Jian-Li," Image parallel processing based on GPU ," Advanced Computer Control (ICACC), 2010 2nd International Conference, March 2010.

[10] S. Eghtesadi, M. Sandler, "Implementation of the Hough transform for intermediate-level vision on a transputer network", Journal of Parallel and Distributed Computing, Volume 13, Issue 3, Pages 212–218, April 1989.

[11] R. Boynton, "Measuring weight and all three axes of the center of gravity of a rocket motor without having to re-position the motor", presentation at the 61st Annual Conference of the Society of Allied Weight Engineers Virginia Beach, Virginia May 20-22, 2002.

[12] Z. Fang , X. Li , "A parallel processing approach to image object labeling problems", CSC '87 Proceedings of the 15th annual conference on Computer Science, Page 423, New York, 1987.

[13] C.Papamanthou, F. Preparata, R.Tamassia, "Algorithms for Location Estimation Based on RSSI Sampling", Springer-Verlag Berlin Heidelberg, 2008.

[14] S. Bohlhalter, C.Fretz, B. Weder, "Hierarchical versus parallel processing in tactile object recognition: a behavioural-neuroanatomical study of aperceptive tactile agnosia", Brain ,2002.

[15] P. Jonker and W. Caarls, "Application driven design of embedded real-time image processing," in Proceedings of ACIVS 2003 (Advanced Concepts for Intelligent Vision Systems), (Gent, Belgium), 2003.

[16] J. darlingtons, Y. Guo, H.W. To, J. Yang, "Functional Skeletons for Parallel coordination", proceeding of 1st EuroPar Conference, Stokholm, Sweden, pp. 55-66, Agust 1995.

[17] R. Jones, "Machine vision applications", science direct, Volume 1, Issue 4, 1991, Pages 439–446.