

Data Partitioning Techniques for Pervasive Multimedia Platforms

Xiaoping Hu, Umit Y. Ogras, Nicholas H. Zamora, and Radu Marculescu

Department of Electrical and Computer Engineering

Carnegie Mellon University

Pittsburgh, PA 15213-3890

{xh, uogras, nhz, radum} @ece.cmu.edu

Abstract

In this paper, we propose a method for mapping multimedia applications on systems with very limited resources (i.e. memory, computing capability and battery lifetime) by combining adaptive data partitioning with content-based dynamic power management. The potential of the approach is illustrated through a case study of an object tracking application running on a resource constrained system which can be embedded in the environment (e.g. offices, home or conference rooms) to offer significantly more opportunities for ubiquitous information, seamless communication, enhanced security, etc. compared to today's portable or stationary devices. Besides power and performance trade-offs, we also explore the scaling effects on data partitioning and provide insights for possible optimization when designing such systems.

1. Introduction

It is envisioned that 5-10 years from now technology will be ubiquitously and invisibly integrated into the natural and human environment. As such, tiny electronic devices will become part of homes, administrative buildings, airports, vehicles or even human clothes [1]. Due to the high demand in integrating various functions onto a single device and also size, weight, and battery life constraints, these emerging design platforms are far more constrained compared to their desktop multimedia counterparts in terms of processing, storage, and energy capabilities [2]. At the same time, since pervasive multimedia systems must continuously interact with humans, real-time applications such as video and audio processing impose additional demands in terms of computation, communication and storage requirements. Consequently, major design difficulties are expected to arise when resource-demanding multimedia applications need to be mapped onto resource-constrained systems.

This paper addresses such difficulties in the context of implementing a moving object tracking application onto a resource-constrained multimedia platform. As we will show later in the paper, the problem of carrying out heavy computations in the presence of limited memory and battery lifetime, as well as modest processing capabilities, can be solved by adaptive data partitioning and dynamic power management. Indeed, smart data partitioning can help map complex applications onto distributed platforms with limited capabilities. At the same time, a content-based power management scheme can provide significant power savings for the host multi-processor system according to the fluctuations in the workload of the target multimedia application.

Furthermore, we explore the effect of scaling up the size of the data partitioning algorithm on the overall power consumption and provide some insights for possible optimizations.

1.1 Contributions and Related Work

In multimedia domain, many researchers have worked on developing high-performance object-tracking systems. For instance, the authors of [3] present several algorithms for a pan-tilt-zoom camera to automatically track a single moving object. In [4], the authors propose an architecture for real-time object tracking which combines image sensors, FPGAs and memory. An object-tracking algorithm that predicts the object contour using the motion vector information is proposed in [5]. Tracking is achieved by predicting the object boundaries using motion vectors, followed by a contour update, using occlusion/disocclusion detection.

The objective of this paper is quite different. Instead of running sophisticated algorithms on powerful multimedia platforms (e.g. PC-based multimedia), we aim at running real-time multimedia applications on resource-constrained platforms. By resource-constrained platforms, we mean systems made up of many "weak" components; that is, components characterized by processing speeds below 100MHz, memory space up to 100KB, and allowable power consumption below 100mW. Designing complex systems with such weak components is important because future electronic devices will become so small that complex systems built out of many devices integrated into everyday physical objects will become commonplace.

1.2 Outline of the Paper

The real-time object tracking application is introduced in the next section. Data partitioning techniques are introduced in Section 3. A video content-based dynamic power management policy and an adaptive data partitioning algorithm are proposed in Section 4, while the scalability effects are analyzed in Section 5. Experimental results are presented in Section 6 and finally, the conclusion is given in Section 7.

2. Real-time Object Detection and Tracking

Our objective in this paper is to make it possible to run the single object detection and tracking algorithm on a collection of distributed simple microcontrollers powered up by small batteries with limited lifetime. To speed up the algorithm, we process the frames both at pixel and macroblock-level (a macroblock is a

block consisting of 16x16 pixels). As explained later, in order to keep the computational effort low, the operations are only carried out on the necessary macroblocks and related pixels. Figure 1 shows the block-level diagram of the algorithm.

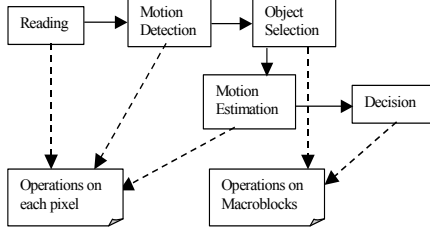


Figure 1 Single object detection and tracking algorithm

The image is first read into the system. Each current pixel is compared with the pixel value from the previous frame, and the SAD (Sum of Absolute Difference) of each macroblock is calculated. The macroblock is detected and marked as a moving macroblock only if the SAD between the two consecutive frames exceeds a preset threshold. The largest group of connected moving macroblocks is selected as the moving object of interest. *Three Step Motion Estimation* is then carried out on each of the macroblocks of the moving object [9]. Decisions related to the position and the velocity of the object are made based on average values of positions and velocities of the macroblocks in the object.

In order to run the above object-tracking algorithm on a target processor, the available memory should be large enough to store two frames to perform the motion detection and estimation steps. Consider a CIF frame with a size of 352x288 pixels; if we only store the luminance value of the frame, more than 75KB are needed to store a single frame with 1 byte for each luminance value. Mapping such an algorithm to microprocessors with 100KB or less memory is simply impossible. To overcome this problem, we propose to use data partitioning as shown next.

3. Data Partitioning

Regular and irregular video partitioning have been investigated in [6] and [7]. In [6] a static global scheduling scheme for mapping computer vision and image processing operations on distributed-memory multiprocessors is developed to reduce the computation and communication cost and optimize parallel time. In [7], two irregular video data partitioning are presented in order to minimize the amount of overhead data.

Although aligned with the same idea of reducing the volume of processed data, our strategy is quite different. For the sake of easy control and communication, we delegate the processing of the basic video frames to multiple microcontrollers in a coordinated fashion. To this end, we allow three regular ways to partition a full video frame: cross, vertical and horizontal partitioning. Using these partitioning schemes, an entire frame can be divided into several regions (or slices), each region being mapped to one processor of the platform for real-time processing. From a performance standpoint, this is essentially a lossless technique where the results of the processing are identical to those that would have been obtained by using a single powerful processor able to process the entire frame at once. Additionally, due to partitioning, different regions of a video frame can be processed in parallel such that higher frame rates can be achieved at lower levels of energy consumption.

Despite the advantages of processing smaller amounts of video data, without careful partitioning, the overhead due to the processing of the necessary neighboring pixels for each processor, and synchronization and communication among microprocessors can be significant. Furthermore, due to the variety of real video sequences, the best partitioning scheme for one video clip may actually be not so good for another one. Given this, we propose next adaptive data partitioning to meet the changing characteristics of the incoming scenes.

4. Power-aware Adaptive Data Partitioning

4.1 Content-based Power Management Scheme

In order to understand the power-aware adaptive data partitioning, first we introduce a video content-based power management scheme. Consider a cross partitioned frame, as shown in Figure 2. Assume that, based on the previously processed frames, the moving object is currently residing somewhere in regions 2 and 4. Since the object is far away from regions 1 and 3, the processors 1 and 3 can be temporarily sent to sleep to save power. In order to avoid any performance loss, as well as power and delay penalties due to frequent shutdown and wake-up actions, we apply a window covering the moving object with some margin δ as shown in Figure 2.

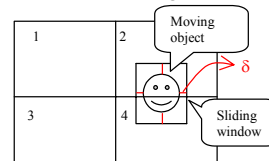


Figure 2 Content-based moving window

The margin δ is set to be at least two times the largest motion estimation range; that is, if the largest range that the motion estimation can achieve is b , then a margin of $\delta \geq 2b$ guarantees that the processor is woke up at least 1 frame before the object enters the region of interest. Accordingly, the overall strategy is simple: for each processed frame, check if the virtual window touches region i ; if not, set processor i to be in standby mode; else wake up the processor i .

4.2 Power-aware Adaptive Data Partitioning

It is obvious that the static data partitioning cannot guarantee a minimum partitioning overhead and best performance for various video clips. For instance, for big or tiny objects in the frames, different data partitioning schemes will not make a big difference with respect to the overall performance of the above power management policy. This is because different data partitioning schemes affect the power management results when the size of object is comparable to the size of one region (slice) in the partitioned region. It is also better to have the shape in the partitioned region match the shape of the object. For example, it is better to use a vertical partitioning scheme when the object is tall and narrow since one or two processors are usually enough to cover the moving object in this case.

Based on these two observations, we propose an adaptive data-partitioning algorithm that can work together with the above power management scheme to provide significant power savings. The algorithm and the update strategy for the current partitioning mode are summarized in Figure 3.

In order to understand the algorithm in Figure 3, let us define the object-shape character r to be the ratio of the object's length and width ($r=L/W$) and the partition-quality character $\theta=(\text{object_size})/(\text{on_area})=\alpha/\beta$, where the "on area" is the area whose corresponding processors are running. The parameter N is the interval of checking the data partitioning quality in order to avoid too frequent switchings among the partitioning modes.

Table 1 Adaptive data partitioning

Current partitioning	Cross	Vertical	Horizontal
$r_{\text{average}} \geq 1$	Vertical	Cross	Vertical
$r_{\text{average}} < 1$	Horizontal	Horizontal	Cross

```

Set partition_mode = cross and turn on all the processors;
Process 1st frame & carry out the window-based power management;
current_r=average_r=L/W;
current_theta=average_theta=(object_size)/(on_area)=alpha/beta;
Set N, theta_l and theta_k; //N=10, theta_l = 0.05 and theta_k = 0.25 in our experiments;
for each frame i{
    ii=mod(i,N);
    Process frame i, carry out the window-based power management for frame i and
    update the value of L, W, alpha and beta;
    current_r=L/W; current_theta=alpha/beta;
    if (i!=kN) { //k is any integer;
        //Update average_r and average_theta;
        average_r=(current_r + average_r * (ii-1)) / ii;
        average_theta=(current_theta + average_theta * (ii-1)) / ii; }
    if (i==kN) {
        average_r=(current_r + average_r * (N-1)) / N;
        average_theta=(current_theta + average_theta * (N-1)) / N;
        if (theta_l <= average_theta <= theta_k) {
            Update the current partitioning mode according to Table 1;
            Set average_r = 0 and average_theta = 0;
        } } }

```

Figure 3 Power-aware adaptive data partitioning

Starting with the default "cross" partitioning, the algorithm switches among the three partitioning modes every N frames (according to the entries in Table 1) based on average values of θ and r . A ratio less than θ_l means that the object is too small compared with the "on" area so the effect of different partitions cannot really be seen. On the other hand, a ratio more than θ_k means that the current partition is good enough so that there is small amount of "wasted area" and most of the "on" area is covered by the object. θ between these two bounds means the current partition is not optimal and changes should be made based on the shape of the object.

5. Scaling Effects

An interesting issue to study is the scaling effects on the overall system performance. Since the total power dissipation is mainly determined by the power consumption during communication and running modes, let's approximate the power consumption as:

$$P_{\text{total}} = C_{\text{com}} \cdot p_{\text{com}} + C_{\text{run}} \cdot p_{\text{run}} + P_{\text{tran}} + \Delta \quad (1)$$

where Δ is a constant approximating the power consumption in standby and idling modes, p_{com} and p_{run} are the average percentage of time spent in communication and running modes, C_{com} and C_{run} are power consumption values in these two operation modes, P_{tran} is the power consumed when transitioning among different operation modes. Obviously, p_{com} , p_{run} and P_{tran} depend on the number of slices used to divide the original

video frame and also on the characteristics of the video stream itself. So the power consumption can be written as in equation (2) below:

$$P_{\text{total}}(n, \text{image}) = C_{\text{com}} \cdot p_{\text{com}}(n, \text{image}) + C_{\text{run}} \cdot p_{\text{run}}(n, \text{image}) + P_{\text{tran}}(n, \text{image}) + \Delta \quad (2)$$

where n is the number of slices used to divide the image data.

For a given video stream, as we divide each frame into more slices (to increase the value of n), p_{com} will also increase accordingly because more communication among the processors is needed in order to perform the required video processing. On the contrary, p_{run} is reduced by increasing the value of n because each slice processing would involve less data.

In summary, the overall effect of scaling the system up depends on the characteristics of incoming video frames and the value of their "costs" (i.e. power consumption in these two modes), C_{com} and C_{run} . We can easily get the surface of $P(n, \text{image})$ according to equation (2); from these surfaces, the optimal scale of the system can be derived accordingly.

6. Experimental Setup and Results

6.1 Basic System Configuration

The above adaptive data-partitioning scheme with power management has been applied to the object tracking application in Section 2. Simulations were run with system-level models built with Stateflow module in Matlab. The processors modeled in this case are Atmel AT91 ARM Thumb microprocessors, which can work at a clock frequency between 0Hz to 70MHz with a 256KB of on-chip SRAM. The power consumption for different operation modes is listed in Table 2. Compared to powerful microprocessors used in desktops or laptops, these processors are, indeed, resource-constrained devices.

Table 2 Operation modes of AT91R40008 processor

Mode	Conditions	Power consumption [mW/MHz]
Normal	Running	0.73
	Communication	1.03
	Idling	0.20
Standby		0.06

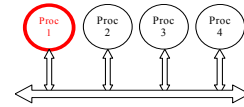


Figure 4 Four-processor bus architecture

The system was tested using several CIF video sequences (352x288 pixels) with the bus architecture in Figure 4. The four processors work on the four partitioned regions shown in Figure 2, with the processor1 acting as master that takes care of the power management and adaptive data partitioning decisions. As such, processor 1 is never sent to sleep. The master processor can talk to all the other processors over the bus, using a First Come First Served (FCFS) schedule to solve the communication conflicts. The total power and time consumption for each frame is measured at maximum clock frequency (70MHz).

6.2 Power and Performance Results

We have obtained consistent results while testing several CIF video sequences of different lengths ranging from 40 to 180

frames. In order to show clearly the performance as a function of time, we present detailed results only for 40 frames (Figure 5).

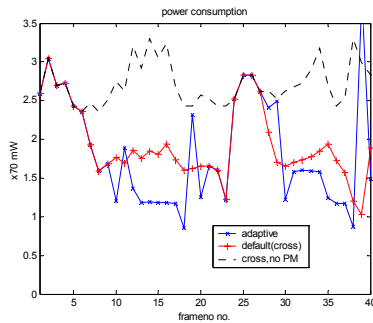


Figure 5 Total power consumption per frame for the bus architecture in Fig.4. Results for adaptive partitioning with power management, cross partitioning w/ power management (default) and cross partitioning w/o power management

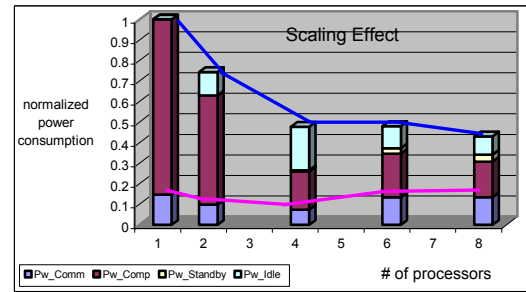
The total power consumption of the four processors using the adaptive data partitioning and power management scheme is compared in Figure 5 with that of a static data partitioning with and without power management. As shown during the period from the 11th to the 19th frame, by using the window-based power management, about 40% of power consumption is saved. Furthermore, another 30% of the power consumption can be saved compared to static data partitioning by using the power-aware adaptive data partitioning. The overhead of adaptive data partitioning can be seen when the system changes its partitioning mode (the peaks at the 10th and the 20th frame). By adjusting the frequency (parameter N in Fig.3) with which we check the partitioning quality, we can avoid frequent switchings of the partitioning mode so that the cost can be kept under an acceptable level.

6.3 Scalability Issues

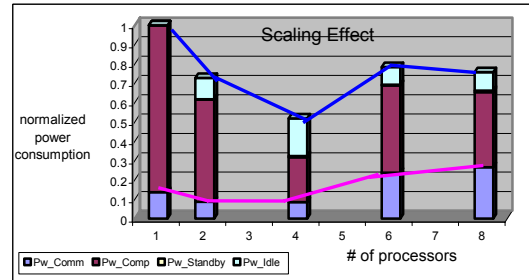
The scaling effects have been discussed in Section 5. Now we present the effects by measuring the power consumption of individual processors with different values of n (n equals 1, 2, 4, 6, and 8) using various input image sequences. Instead of presenting the surface of $P(n, image)$, we show the $P(n)$ curves when the images contain mid-size and large moving objects. In Figures 6, the power consumption is normalized with respect to the power consumption value obtained when using a single processor (without data partitioning). For clarity, power consumption on different modes (idling, standby, running, and communication) is shown with different colors.

As we can see in Figure 6, the power savings become less obvious as the system scales up. This is caused by the increase in the amount of power consumption during communication because more information needs to be exchanged between the cooperating processors as the same amount of data is partitioned into more pieces. Also, larger moving objects make p_{com} and p_{run} more sensitive to n . As shown in Figure 6(a), for medium size objects, the effect of increasing the number of slices n is not significant so the total power consumption of the individual processor keeps reducing when n is increasing. However, the penalty of increasing n becomes obvious as the size of the object becomes larger

(Figure6(b)). The total power consumption increases as we increase the number of slices used for partitioning from 4 to 6.



(a)



(b)

Figure 6 Normalized power consumptions vs. number of processors for the bus architecture (a) object size is medium, between 1/20 and 1/4 of the image size (b) object size is large, more than 1/4 of the image size

7. Conclusions

In this paper we have proposed adaptive data partitioning and content-based power management schemes, which can be combined and used for video applications running on resource constrained systems for pervasive multimedia platforms. Future work will concentrate on building a network of such systems with the objective of monitoring a larger area of interest.

References

- [1] M.Lindwer, et al, "Ambient Intelligence Visions and Achievements: Linking Abstract Ideas to Real-World Concepts", DATE, 2003, pp.10-15.
- [2] E.Aarts, R.Roovers, "IC Design Challenges for Ambient Intelligence", DATE, 2003, pp.2-7.
- [3] Y.W.Huang, B.Y.Hsieh, S.Y. Chien, L.G. Chen, "Simple and Effective Algorithm for Automatic Tracking of a Single Object Using a Pan-tilt-zoom Camera", ICME 2002, Vol 1, pp. 789-792.
- [4] T.Hamamoto, S.Nagao, "Real-time Objects Tracking By Using Smart Image Sensor And FPGA", ICIP, 2002, pp.441-444.
- [5] K.Hariharakrishnam, D.Schonfeld, P.Raffy, F.Yassa, "Object Tracking Using Adaptive Block Matching", ICME 2003, Vol 3, pp. 65-68.
- [6] C.Lee, Y.Wang, T.Yang, "Static Global Scheduling for Optimal Computer Vision and Image Processing Operations on Distributed-Memory Multiprocessors", Technical Reports, 1994, CS. Dept. UC Santa Barbara
- [7] D.Altılar, Y.Paker, "Minimum Overhead Data Partitioning Algorithms for Parallel Video Processing", 12th Inter. Conf. on Domain Decomposition Methods, 2001.
- [8] A.Acquaviva, L.Benini, B.Ricco, "An Adaptive Algorithm For Low-power Streaming Multimedia Processing", DATE, March, 2001, pp.273-279.
- [9] R.Li, B.Zeng, M.Liou, "A new three-step search algorithm for block motion estimation", IEEE Trans. CASVT, Vol. 4, No. 4, Aug. 1994, pp.438-442.