Adaptive Data Partitioning for Ambient Multimedia*

Xiaoping Hu, Radu Marculescu
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213-3890
{xh, radum} @ece.cmu.edu

ABSTRACT

In the near future, Ambient Intelligence (AmI) will become part of everyday life. Combining feature-rich multimedia with AmI (dubbed Ambient Multimedia for short) has the potential of changing the way we perceive and interact with our environment. One major difficulty, however, in designing Ambient Multimedia Systems (AMS) comes from the strong constraints imposed on system resources by the AmI application requirements. 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 dynamic power management. The potential of the approach is illustrated through a case study of an object tracking application running on a resource-constrained platform.

Categories and Subject Descriptors

C.3 **[Computer Systems Organization]:** Special-purpose and Application-based Systems -- *real-time and embedded system*

General Terms

Design, Algorithms

Keywords

Ambient Intelligence, motion detection, power management

1. INTRODUCTION

Ambient Intelligence (AmI) is the vision that technology will ubiquitously and invisibly be integrated into the natural human environment [1]. Challenges brought up by the AmI lie in the design and manufacturing of the appropriate "microelectronics"; that is, the basic building blocks that need to satisfy the requirements of enhanced performance and mobility, smaller size and weight, and lower cost and energy consumption.

Due to the high demand for functional integration on one hand, and size, weight and battery lifetime constraints on the other hand, the components for ambient multimedia are truly *resource limited devices* (in terms of processing, storage, and energy capabilities) compared to their desktop multimedia counterparts [2][3]. At the same time, real-time applications such as video and audio processing, impose tight constraints in terms of computation, communication and storage capabilities. Consequently, major challenges arise when mapping complex multimedia applications onto resource-constrained systems.

* Research supported by Marco Gigascale Research Center (GSRC)

This paper addresses such design difficulties in the context of implementing an object tracking application onto a resource-limited platform. Object detection and tracking is widely used in multimedia (e.g. video conferencing, digital surveillance systems, etc.). As shown later, the problem of carrying out heavy computations in the presence of limited memory and processing resources can be solved by adaptive data partitioning. 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.

1.1 Contributions and Related Work

Much work in multimedia has been devoted to developing high-performance object-tracking systems [4][5][6]. Our objective in this paper is quite different. Instead of running sophisticated algorithms on powerful chips, we aim at running real-time applications on resource-constrained platforms. By resource-constrained platforms, we mean systems made up of many "weak" components (i.e. components characterized by processing speeds below 100MHz, 100KB or less memory space, and 100mW or less power consumption). Systems with such weak components are important because the basic assumption of AmI is that electronics will become so small that complex systems will be built out of many tiny devices integrated into everyday physical objects. Our contribution in this paper is two fold:

- ***Adaptive data partitioning*:** Regular and irregular video partitioning has been investigated in [7] and [8]. Based on the same idea of reducing the volume of processed data, 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 available processor of the platform for real-time processing. Since different regions of a frame can be processed in parallel, higher frame rates can be achieved with lower energy consumption. Furthermore, the frame partitioning scheme is decided adaptively to meet the changing characteristics of the incoming scenes.
- ***Content-based dynamic power management*:** For real-time video processing, the CPU power consumption depends heavily on the workload imposed by the application and algorithm being executed. Dynamic power management can provide the required services and performance level in a power-efficient way according to the changes in the workload [9]. However, the effectiveness of such approaches depends on the stochastic models that are used to predict the next state of the system based on its recent evolution. In contrast, we propose a power management policy which acts based on the actual video-content and therefore does not involve any prediction. As such, it has much more stable performance and lower overhead.

1.2 Outline of the Paper

The object tracking application is introduced in the next section. Content-based dynamic power management and adaptive data-partitioning techniques are proposed in Sections 3 and 4, respectively.

The experimental results are presented in Section 5. Finally, the conclusions are given in Section 6.

2. REAL-TIME MOVING OBJECT DETECTION AND TRACKING

2.1 System Overview

Our target system is an intelligent surveillance system, consisting of an array of cameras inconspicuously embedded in the environment (e.g. walls of a conference room or office). The system is organized in a hierarchal way at two levels of abstraction: the *node-level* and the *network-level* as shown in Figure1.

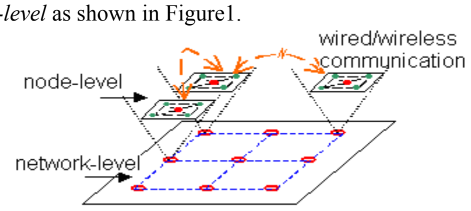

Figure 1. Intelligent surveillance system

At the lowest level, each *node* consists of a video camera and a few simple microcontrollers that process the raw data captured by the camera and cooperate to implement the object-tracking algorithm. Detailed information such as position, direction and velocity of the moving object is collected to assist the network-level management, which consists of transferring the task of object tracking from one node to another as the object moves with respect to the spatial coordinates of the system. As the first step in the overall design methodology, this paper focuses on the design issues that occur at node-level, while future work is left for the overall network design.

2.2 Algorithm Overview

At node-level, the objective is to implement a single object detection and tracking algorithm with a very low computational effort. To speed up the algorithm, the frames are processed at pixel- and macroblock-level (a macroblock is a block consisting of 16x16 pixels). Figure 2 shows the block-level diagram of the algorithm.

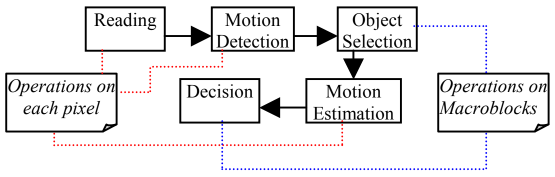

Figure 2. Single object detection and tracking algorithm

We start with reading the image. 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 [11]. 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.

To run the above object-tracking algorithm on a target processor, the available memory should be large enough to store two frames in order to perform the motion detection and estimation steps. For a CIF frame of size 352x288 pixels, more than 75K bytes are needed to store only a single frame with 1 byte for each luminance value. To map such an algorithm to microprocessors with less than 100KB memory is simply impossible. To solve this problem, we propose a data partitioning approach.

3. VIDEO CONTENT-BASED POWER MANAGEMENT SCHEME

To make the object-tracking algorithm based on the coordinated data partitioning scheme power efficient, we first propose a power management scheme using a *spatially sliding window*. Since this approach is based on the actual video data, it differs from previous work by having much less computational overhead and performance degradation; this is because no history-based prediction is involved.

To make the discussion more concrete, consider a cross partitioned frame as in Figure 3, where processing data for each region gets assigned to a separate processor. Also, assume that the moving object is currently residing somewhere in regions 2 and 4. Since the object is far away from regions 1 and 3, processors 1 and 3 can be sent to sleep for a while (and therefore saving power). In order to avoid any performance loss, as well as power and delay penalties due to frequent *shutdown* and *wakeup* actions, we track the moving object with a sliding window having a margin δ as shown in Figure 3.

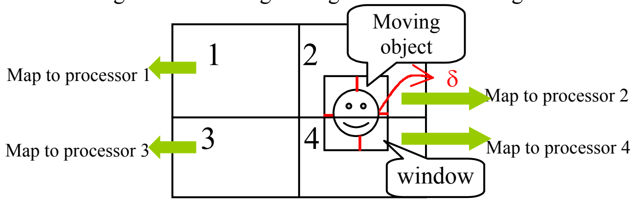

Figure 3. Content-based moving window

The margin δ is set to be at least two times the largest motion estimation range; that is, if the largest range that motion estimation can achieve is *b,* then $\delta \geq 2b$ guarantees that the processor will be wake up at least 1 frame before the object enters the region of interest. The simple power management strategy is summarized in Figure 4.

```
δ=2b; //b is the largest range the motion estimation can achieve;
for each frame{
    Process the frame;
    L= left-most position of the macroblock of the object;
    R=right-most position of the macroblock of the object;
    U=up-most position of the macroblock of the object;
    D=down-most position of the macroblock of the object;
    virtualwindow=Rectangle(L-δ,D-δ,R+δ,U+δ)∩(image area);
    for each partitioned region i{
       if (virturalwindow does not touch region i & processor i is not in standby mode)
          Set processor i to be in standby mode;
       else
         if(virtualwindow touches region i & processor i is in standby mode)
          wake up processor i; } }
```

Figure 4. Content-based power management algorithm

The idea is to partition each video frame into small regions and then assign work only to some processors while sending others to sleep. We should note that different data partitioning strategies can affect significantly the performance of the power management because of the various size and shape of the object. We will address this problem in the next section.

4. Power-aware Adaptive Data Partitioning

Static data partitioning alone *cannot* guarantee minimum partitioning overhead and best performance for various video clips. For instance, static partitioning is good enough for slowly-changing input videos, when the moving object does *not* change dramatically in terms of size, shape and speed (e.g. news broadcasting). However, if the incoming video is fastly-changing, *adaptive* data partitioning is needed to meet the changes in the workload characteristics. Also, different data partitioning schemes help in power management when the size of object is *comparable* and *matches* the size of one slice in the partitioned frame. For example, it makes sense to use a vertical partitioning scheme when the object is long and narrow since one or

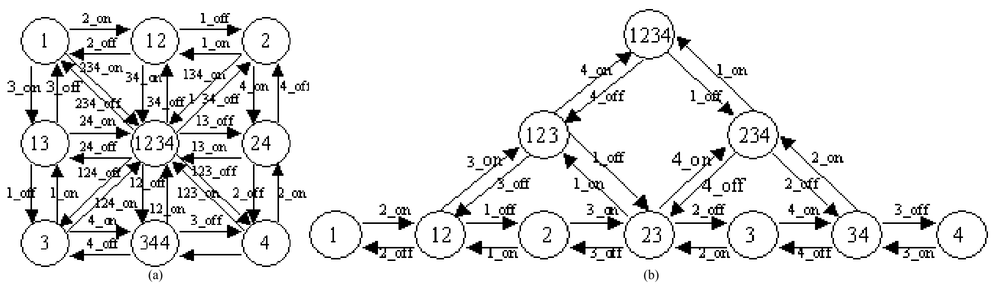

Figure 6 Power modes for cross partitioning (a) and for vertical and horizontal partitioning (b)

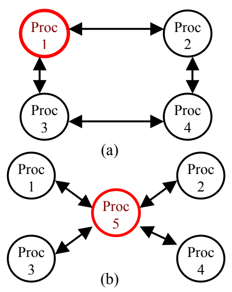

Figure 7. The mesh (a) and star (b) architectures

two processors will be enough to cover the object in this case.

Based on these 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 is summarized in Figure 5.

```
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_θ=average_θ=(object_size)/(on_area)=α/β;
Set N, θl  and θκ ; // N=10, θl  = 0.05 and θκ  = 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, α and β ;
  current_r=L/W; current_θ=α/β;
  if (i!=kN)  {  // k is any integer;
    //Update average_r and average_θ;
    average_r=(current_r + average_r * (ii-1)) / ii;
    average_θ=(current_θ + average_ θ* (ii-1) ) / ii; }
    if (i==kN) {
      average_r=(current_r + average_r * (N-1) ) / N;
      average_θ=(current_θ + average_θ * (N-1) ) / N;
      if (θl ≤ average_θ ≤ θk ) {
         Update the current partitioning mode according to Table 1;
         Set average_r =0 and average_θ=0; } } }
```

Figure 5. Adaptive data partitioning

In order to understand the algorithm in Figure 5, let us define the *object-shape parameter r* to be the ratio of the object's length and width ($r=L/W$) and the *partition-quality parameter* θ=(object_size)/(on_area)=α/β, where the "on area" is the area whose corresponding processors are running normally.

Table 1 Adaptive data partitioning

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

Starting with the default "cross" partitioning, the algorithm switches among the three partitioning modes every *N* frames based on average values of θ and *r* (Table 1 and Figure 5). A ratio less than θ_l means that the object is too tiny compared with the "on" area so the effect of different partitions cannot really be seen. On the other hand, a ratio bigger than θ_k means that 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 a change should be made based on the shape of the object.

The three partitioning modes are switched on the actions decided by the adaptive data partitioning algorithm. The power modes corresponding to each kind of partitioning are shown in Figure 6. In this figure, state *ij* means that processor *i* and processor *j* are on. The power consumption for each partitioning mode can be roughly estimated to be proportional to the number of "on" processors. Assuming that the weight of each state is given by the number of "on" processors (for example, the state 123 has the weight of 3). We can roughly estimate the power consumption for the input clip as $P = P_{unit} \cdot \sum w_i$, where P_{unit} is the average power consumption per processor, w_i is the weight of state when processing the i^{th} frame and $\sum w_i$ is the accumulated weight.

5. EXPERIMENTAL RESULTS

5.1 Basic System Configurations

The adaptive data-partitioning has been applied together with the power management strategy to the application introduced in Section 2. Simulations were run with system-level models built with the Stateflow module in Matlab. The microcontrollers modeled in this case study are Atmel AT91 ARM Thumb microprocessors. These processors have a 256K byte of on-chip SRAM and can work at a clock frequency up to 70MHz. According to the AT91R40008 processor datasheet, the power consumption for different power modes varies between 0.06mW/MHz in standby mode, to 1.03mW/MHz in communication mode.

The system was tested using several CIF video sequences (352x288 pixels) and two architectures (mesh, star), as shown in Figure 7. In the case of the mesh architecture (Figure 7 (a)), the processor 1 is the master processor that takes care of the power management and the adaptive partitioning decision. Consequently, processor 1 is never sent to sleep. In terms of communication, we select 3 to be the intermediate processor to deliver the messages between processor 1 and processor 4. In the case of the star architecture (Figure 7(b)), five processors are used. Processors 1, 2, 3, 4 can all process each of the four partitioned regions, while the processor 5 works as a master node.

In both architectures, a First Come First Served (FCFS) schedule is used to solve the communication conflicts. The total power and time consumption for each frame are measured at the maximum clock frequency (70MHz).

5.2 Power and Performance Results

We have obtained consistent results while testing several CIF video sequences of different lengths ranging from 40 to 180 frames. For clarity reasons, we present in Figures 8, 9 and 10 results for only 40 frames.

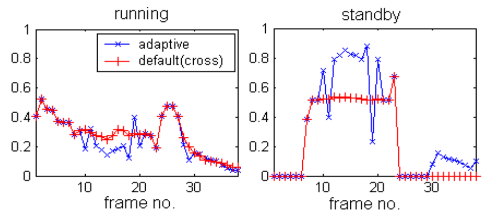

Figure 8. Percentage of time spent in different operation modes for the mesh architecture

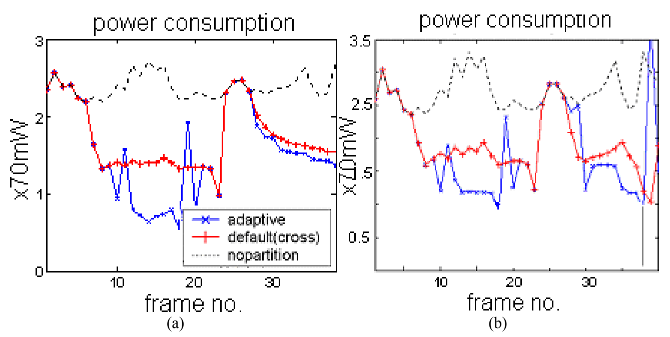

Figure 9. Power consumption for the mesh (a) and star (b) architectures. Results for adaptive partitioning with power management, cross partitioning (fixed) with power management (default) and cross partitioning without power management

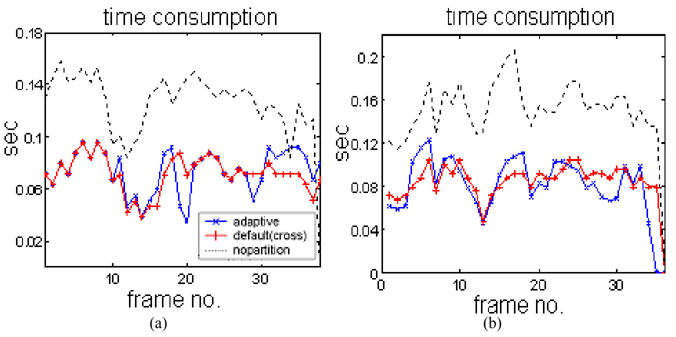

Figure 10. Time consumption on each frame for the mesh (a) and star (b) architectures. Results for adaptive partitioning with power management, cross partitioning (fixed) with power management (default) and no data partitioning

Figure 8 shows the average percentage of time spent in normal running and standby mode with adaptive data partitioning and with static (i.e. non-adaptive) cross partitioning for the mesh architecture in Figure 7(a). As shown, due to the adaptive partitioning, the percentage of time spent in standby mode increases, while the percentage of time spent in running mode decreases during the period from 11th to 19th frame. This means that the system dynamically switches to a less power-consumption configuration by adaptively changing data partitioning mode. Figure 9 shows in detail the effects of such adaptiveness.

For the mesh architecture (Figure 9(a)), during the period from the 11th to the 19th frame, by using the window-based power management with static data partitioning, about 50% of power consumption is saved. Furthermore, another 40% of the power consumption can be saved compared to static data partitioning by using the adaptive data partitioning. The overhead incurred by the adaptive data partitioning can be seen when the system changes its partitioning mode (see the peaks at the 10th and the 20th frame). By tuning the interval of checking the partitioning quality (i.e. the parameter *N* in Figure 5), we can avoid frequent switchings of the partitioning mode such that the cost can be kept under an acceptable level. The total power consumption per frame of the five processor star architecture shows (qualitatively speaking) similar results (Figure 9(b)). Similar results have been obtained for the bus architecture [12].

Finally, as shown in Figure 10, due to the static data partitioning, the system with no partitioning is speeded up by 50%, on average, while adaptive data partitioning does not degrade much of the performance in time.

6. CONCLUSION AND FUTURE WORK

In this paper, we have proposed adaptive data partitioning and content-based power management schemes that can be combined and used for video applications running on resource constrained systems that implement AmI applications. Future work will focus on designing the system at the network-level. In that case, instead of being tightly connected, as is the case at node level, the system components are loosely coupled. Consequently, new communication and power management policies need to be designed in order to exploit the distributed characteristic at that level.

REFERENCES

[1] E.Aarts, R.Roovers, "*IC Design Challenges for Ambient Intelligence*", Proc. DATE, 2003.

[2] F.Voekhorst, "*Ambient Intelligence, the Next Paradigm for Consumer Electronics*", Proc. ISSCC 2002.

[3] M.Lindwer, et al, "*Ambient Intelligence Visions and Achievements: Linking Abstract Ideas to Real-World Concepts*", Proc. DATE, 2003, *pp.*10-15.

[4] 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*", Proc. ICME 2002, Vol 1, *pp.* 789-792.

[5] T.Hamamoto, S.Nagao, "*Real-time Objects Tracking By Using Smart Image Sensor And FPGA*", IEEE Intl. Conf. on Image Processing, 2002, *pp.*441-444.

[6] K.Hariharakrishnam, D.Schonfeld, P.Raffy, F.Yassa, "*Object Tracking Using Adaptive Block Matching*", Proc. ICME 2003, Vol 3, *pp.* 65-68.

[7] 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.

[8] D.Altilar, Y.Paker, "*Minimum Overhead Data Partitioning Algorithms for Parallel Video Processing*", 12th Intl. Conf. on Domain Decomposition Methods, 2001.

[9] L.Benini, A.Bogliolo, G.Micheli, "*A Survey of Design Techniques for System-level Dynamic Power Management*", IEEE Transactions on VLSI Systems, Vol 8, No. 3, June, 2000. *pp.*299-316.

[10] A.Acquaviva, L.Benini, B.Ricco, "*An Adaptive Algorithm For Low-power Streaming Multimedia Processing*", Proc. DATE, March, 2001, *pp.*273-279.

[11] 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.

[12] X.Hu and R.Marculescu, "*Power-aware Data Partitioning for Ambient Multimedia*", Technical Report, ECE Department, Carnegie Mellon University, March 2004. Available at http://www.ece.cmu.edu/~sld/pubs