CROSS-LAYER ADAPTIVE VIDEO CODING TO REDUCE ENERGY ON GENERAL-PURPOSE PROCESSORS

Daniel Grobe Sachs,[†] *Sarita V. Adve,*[‡] *and Douglas L. Jones*[†]

[†]Coordinated Science Laboratory, [‡]Department of Computer Science
University of Illinois at Urbana-Champaign
grace@cs.uiuc.edu

ABSTRACT

Traditionally, video encoders have been designed assuming that the more redundancy is removed, the better the encoder. However, on current laptops, reducing the compression efficiency of the video encoder by reducing the number of instructions used to perform compression can actually reduce the total energy used to encode and transmit a sequence. The correct balance between computation and compression efficiency may change dynamically, motivating adaptive encoders. At the same time, recent general-purpose processors also employ energy-driven adaptations. For best gains, the adaptations in the hardware and application layers must be coordinated. From a system design viewpoint, this coordination must happen through minimal, well-defined interfaces.

This paper develops (1) an adaptive video encoder for general-purpose processors that trades computational complexity for compression efficiency to minimize total system energy, and (2) a method for determining the best configuration for such an encoder when running on a processor that is also adaptive. Our adaptive processor employs recent energy saving techniques of dynamic voltage and frequency scaling and architectural adaptation. Using a detailed simulator, we show that our cross-layer adaptive application algorithm reduces energy significantly, when employed on a fixed or adaptive processor.

1. INTRODUCTION

Until recently, most video was encoded by large, fixed installations with no particular computational power or energy limitations, leading to the inevitable optimization of video encoders to achieve the best possible compression of the video. More recently, as desktop video encoding has become practical, some "short-cuts" (such as the computationally-constrained motion search introduced in [1]) have been used to reduce computational load or achieve real-time encoding on smaller systems. While reducing computation can reduce total system energy, the emphasis has been on creating techniques that sacrifice as little compression efficiency as possible while achieving a particular performance goal.

However, in many cases the key resource we want to conserve is *energy*, not bandwidth. Assuming we disallow increased distortion, we can only reduce the number of bits encoding the sequence by expending additional energy searching for and removing additional redundancy from the video stream. This only reduces the total energy consumed if we use less energy performing additional computation than we would have used transmitting the extra bits. This suggests that by adaptively controlling "short-cuts" that trade CPU time for bit rate, we can achieve energy savings by assuring we only perform compression that saves enough bits to make the extra computation worthwhile.

Furthermore, the relative cost of computation and network bandwidth can vary over time, as many modern general-purpose processors allow the energy per instruction to be varied through adaptive mechanisms. In many cases, increasing the system performance — allowing more instructions to be executed in the same amount of wall-clock time — increases the amount of energy used per instruction. This not only increases the energy savings achievable by reducing computation, but also means that the optimal set of "short-cuts" depends not only on the video stream being encoded, but also on other demands being made on the processor.

To optimize system-wide energy, we created an adaptive video coding algorithm, an adaptation control algorithm that picks the best configuration of the video encoder, and a framework to coordinate the adaptations in the hardware and application layers. This cross-layer framework exchanges enough information between the processor and application to ensure the adaptations in the two layers are coordinated; at the same time, each layer controls its own adaptation process without the need to expose it to the other layer.

This work was done as part of the broader Illinois GRACE (Global Resource Adaptation through CoopEration) project [2], which aims to develop adaptive systems in which all layers (including the hardware, operating system, network, and applications) coordinate to optimize multimedia quality of service and resource usage. In this paper, we focus on processor and application adaptation, using a simplistic model of transmit energy consumption and assuming adequate network bandwidth. We also assume a single application running at a time. Other work in the GRACE project is considering more detailed network models and resource management across multiple applications [3].

2. PROCESSOR ADAPTATION FOR ENERGY

We consider a modern superscalar general-purpose processor employing out-of-order issue, a conventional cache/memory hierarchy, and hardware adaptation techniques to reduce energy consumption. Recall that:

$$\text{Execution Time} = \frac{\text{Instruction Count}}{\text{IPC} \times f} \quad (1)$$

$$\text{Energy} = C_{eff} V^2 \times f \times \text{Execution Time} \quad (2)$$

$$= C_{eff} V^2 \frac{\text{Instruction Count}}{\text{IPC}} \quad (3)$$

where C_{eff} is the effective capacitance of the processor, V is the voltage, and f is the clock frequency. IPC is the average number of instructions executed per cycle, which can be greater than one for superscalar processors.

Two techniques used by CPUs to reduce energy are: reducing V and reducing C_{eff}/IPC. To reduce V, processors use dynamic voltage scaling (DVS) [4]. This reduces the processor frequency,

This work is supported in part by the National Science Foundation under grants NSF CCR-025638, EIA-0072043, CCR-0096126, and CCR-0209198. Sarita Adve is also supported by an Alfred P. Sloan Research Fellowship.

which enables reducing the voltage. Voltage reduction is linear with frequency for a large range; therefore, this results in quadratic savings of energy, but with increased execution time.

To affect C_{eff}/IPC, hardware adaptation adapts the processor architecture, typically by switching off certain parts of the chip, such as functional units, issue logic, or parts of the cache [6]. Such adaptations reduce the effective capacitance of the processor, reducing the amount of energy used for each cycle. While architecture adaptations may also reduce IPC, thereby increasing execution time, if the processor resources being switched off are relatively underutilized, energy savings can be achieved.

A key to effectively using hardware adaptation is the adaptation control algorithm, which determines when and what to adapt. We build on the frame-granularity algorithm for *non-adaptive* applications in [6]. Before the start of each frame, the algorithm chooses the hardware configuration (i.e., the frequency/voltage and architecture) that can complete the processing of the frame within the deadline with the lowest energy. To do this, the algorithm must predict the execution time and energy consumption for each hardware configuration for the next frame. This is done using the following observations for multimedia applications from [5]:

- For a given (non-adaptive) application and a given frame type (I, P, B), the IPC stays roughly constant across all frames of that type. This is because the IPC is determined by the nature of work done in the frame, which stays the same. Further, IPC is roughly independent of frequency, since these applications are compute intensive. Thus, the IPC for each frame type for a given hardware configuration can be pre-determined by profiling a frame of that type on the corresponding architecture (at any frequency).

- The number of instructions (and hence execution time) in frames of the same type may vary. This change is relatively slow; therefore, recent history can be used to predict the instruction count for the next frame.

Using information about the IPC and instruction count, the execution time for a frame can be predicted for any hardware configuration. Analogous observations can be applied for predicting energy consumption [6]. Specifically, the average *energy per instruction* or *EPI* is roughly the same for all frames of the same type for a given hardware configuration and non-adaptive application, and can be determined by profiling. The energy for a given hardware configuration and frame can thus be determined using this EPI and the predicted instruction count. At the beginning of each frame, the algorithm therefore has enough information to choose the hardware configuration with the lowest energy consumption that meets the application's deadline.

Application adaptation affects the above algorithm in two ways. First, the IPC and EPI now hold constant only for the frames that use the same application configuration. We therefore need to profile each combination of hardware architecture and application configuration to determine IPCs and EPIs, and the hardware adaptor must know which application configuration will be used for the next frame. Second, the change in instruction count for two successive frames of the same type is no longer predictable using simple history based schemes since these frames could employ different application configurations. It is also not possible to use a history predictor for just the subset of frames using the same application configurations since two such successive frames could be separated by many other frames (employing other application configurations) and the properties of the stream could have changed significantly in that time. We therefore need other methods to determine the instruction count, discussed in Section 3.2.

3. APPLICATION ADAPTATION FOR ENERGY

3.1. Adaptive video encoder

The encoder used in this work is based on the TMN (Test Model Near-Term) 1.7 encoder,[1] which encodes standards-compliant H.263 streams. We modify the encoder to trade off computational complexity against the number of bits output by providing mechanisms to vary the compression efficiency of the encoder.

Because the above TMN encoder uses a full search to find motion vectors, we replaced the motion search with a fast search similar to the logarithmic motion search technique [7]. This reduced motion search to a total of approximately 25% of the CPU time, and the DCT and IDCT together take up approximately 30% of the CPU time. We use this version as our baseline fixed application.

To allow the complexity of the encoder to vary, we selectively drop motion search and DCT computation based on adaptive thresholds. For motion search, at each step, we compare the SAD (sum of absolute difference) with an externally specified threshold, terminating the motion search if, after any step, the candidate motion vector has a SAD of less than the threshold. For DCTs, we only DCT-transform the 8x8 block of coefficients if the sum of the absolute values of the coefficients exceeds a threshold. To do this, we must deviate from the H.263 specification by adding an extra bit before each 8x8 block of coefficients is transmitted, indicating whether or not a DCT was performed on that block.

Because the motion search and DCT thresholds can take on a large number of values, we choose a set of four thresholds across the range for each. This gives a total of 16 application configurations.

The net effect of these modifications is that, by changing the thresholds, we can vary both bit rate and the number of instructions by approximately a factor of two. Because the adaptations increase the rate of the resulting sequence, energy savings are upper-bounded by the amount of CPU energy that can be saved. By enabling the processor to run at a lower frequency, reducing the amount of computation by $1/2$ allows for the total amount of energy used by the CPU to be reduced to $1/8$ the original value. This permits a total energy savings of up to approximately $7/8$ the original energy in cases where the CPU energy is dominant.

3.2. Adaptation Control Algorithm

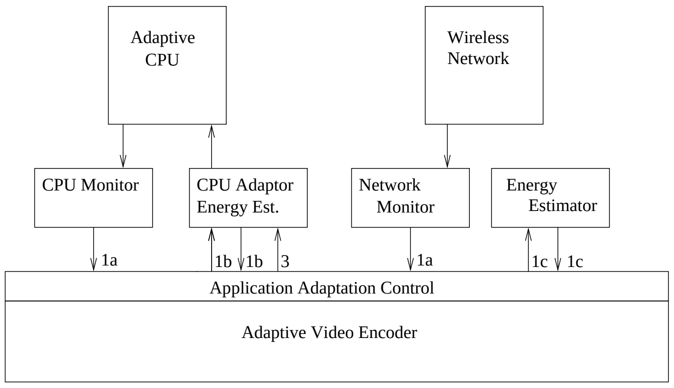
Like the hardware adaptation, we perform the adaptation of the video encoder on a frame-by-frame basis by choosing a set of thresholds for the encoder for each frame. The goal for the adaptation mechanism is to find the configuration (the set of thresholds) for the video encoder that minimizes the total energy consumption of the system. For this purpose, our algorithm estimates the total energy consumption for each application configuration as follows.

Overview. The total system energy is the sum of the CPU and network energy. To determine the network energy for an application configuration, our network model (Section 4) simply requires a prediction of the number of bytes that will be generated for the next frame. The predictors used are described below.

The CPU energy for a given application configuration depends on the hardware configuration that will be chosen for the next frame. This hardware configuration, however, itself depends on the application configuration chosen for the next frame. Thus, there is a close coupling between the hardware and application adaptations. From a system design viewpoint, we would like this coupling to be reduced to the minimal, allowing the hardware and application to create independent adaptation processes to the extent possible.

Figure 1 illustrates our proposed algorithm and interface. To determine the CPU energy estimate for a specific application configuration, the application layer simply queries the hardware layer,

[1] Available from http://www.xs4all.nl/˜roalt/h263.html.



```
Application Adaptation Algorithm:

1. For each application configuration:
   a. Get instruction and byte count for previous frame
      from CPU and network monitors, and estimate for
      next frame.
   b. Send estimated instruction count to CPU adaptor.
      Get estimated CPU energy.
   c. Send estimated byte count to network monitor.
      Get estimated network energy.
   d. Sum estimated CPU, net energy to find total
      energy estimate.

2. Choose lowest−energy configuration.

3. Send instruction−count to CPU adaptor,
   and execute frame.
```

Fig. 1. Application adaptation algorithm and cross-layer interfaces

providing it with the minimal information needed to determine the lowest energy hardware configuration for the specified application configuration. The minimum information needed by the hardware layer is (i) the EPI of the application configuration for each possible hardware configuration, and (ii) the instruction count for the application configuration for the next frame (based on the algorithm in Section 2). The EPI values are pre-determined through profiling as described earlier. The instruction count information is communicated explicitly by the application layer, estimated through a set of predictors described next. Using this information, the hardware layer determines the least energy hardware configuration for the corresponding application configuration and returns back the energy estimate to the application layer.

Instruction count and network byte predictors. We predict instruction count and byte count for the next frame using a set of linear predictors. For every transition between the previous frame's application configuration and the next frame's application configuration, the linear predictors map the previous frame's instruction and byte count to an estimate for the next frame. These predictors are estimated in the least-squares sense from sample data generated by encoding a representative sequence repeatedly, with each frame choosing randomly between the encoder's different application configurations. For each instruction-count prediction, a fixed leeway is added (set to ensure that no more than approximately 5% of deadlines are missed).

The complete system. The complete system requires an offline generation of the linear predictors for the application, and an initial on-line profiling phase where the CPU adaptor determines the IPC and EPI values for each combination of possible hardware and application configurations. In the rest of the run, before each frame, we do the following (illustrated by Figure 1). The application layer determines the actual instruction count and network bytes in the last frame from the CPU and network monitors. It uses these to generate predictions for the instruction count and network bytes for the next frame for each application configuration. It sends to the CPU layer the instruction count estimate for each application configuration and gets back corresponding energy estimates. The application layer similarly obtains the network energy estimate by sending the network byte count to the network layer. The application layer then sums the two to determine the total energy for each application configuration and picks the lowest energy configuration. It indicates this choice to the hardware layer, which then picks the corresponding lowest energy hardware configuration. A key property of our system is that the hardware adaptation process itself appears like a black box to the application and vice versa.

4. EXPERIMENTAL METHODOLOGY

We used the architecture simulator RSIM [8] for our simulations. We model a superscalar, out-of-order processor, supporting four architectural configurations varying in instruction window size and number of functional units [6]. The processor can also scale voltage and frequency, over a continuous frequency range of 100MHz to 1GHz. To estimate energy use, RSIM incorporates Princeton's Wattch [9] power-modeling package.

We use a fixed energy-per-byte value to account for the energy cost of transmitting the encoded information from the wireless media transmitter to its destination. For an unloaded 802.11-style network, the combination of a per-byte and per-packet energy cost is reasonably accurate [10]; for convenience we assume that each frame is transmitted in a single packet, although many frames exceed the 802.11b MTU of 2400 bytes.

We used two standard MPEG-4 test sequences at QCIF (176x144) resolution, *Carphone* (382 frames) and *Foreman* (400 frames). To generate the offline predictors for instruction and byte count, we used the *Carphone* sequence, and applied these predictors to both sequences. For estimates of IPC and EPI, we used on-line profiling using the first 85 frames for both sequences. To allow the profiling to complete and the encoder to settle, we do not count the energy used for the first 100 frames of each sequence.

We set the deadline at 33.333 ms per frame (30fps). This allows the adaptive processors to run relatively slowly; depending on the configuration of the adaptive encoder, frequencies in the range of 100MHz to 250MHz are required. Two CPUs were evaluated: an "expensive" one that consumes approximately 40W at 1GHz in the most aggressive configuration, and an "inexpensive" one that is otherwise identical but uses exactly one tenth the energy for every operation. The network uses $2 \times 10^{-6} J$ per byte transmitted [10]. The quantization step size of the encoder was fixed at $Q = 4$.

The average PSNR of the encoded stream across the 16 configurations tested remained within 1dB of the PSNR achieved by a non-adaptive (H.263-compliant) encoder. The number of deadline misses never exceeded 4%, and was generally much lower.

We did not account for adaptation overhead since it is incurred only once per frame and is expected to be small relative to the frame size [6].

5. RESULTS

Our simulations show that our adaptive application provides significant energy savings over both fixed and adaptive hardware. Furthermore, even with imperfect predictions, we can realize most of the achievable energy savings.

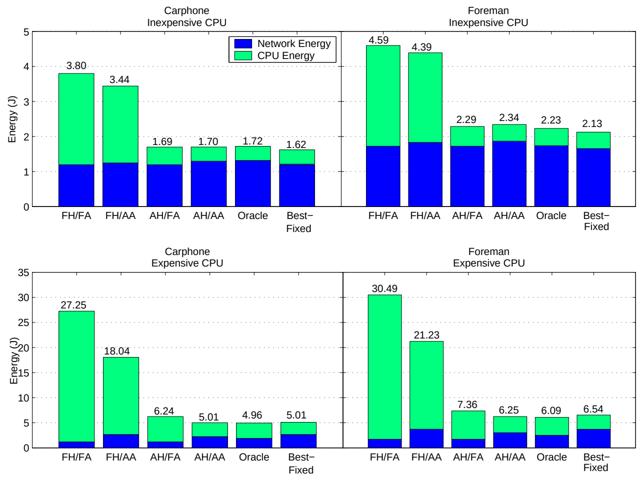


Fig. 2. Energy comparison for fixed and adaptive systems

Figure 2 shows the energy required to encode the *Carphone* and *Foreman* streams for both the inexpensive and expensive CPUs, under six hardware/application scenarios. In each chart, the first 4 bars show all the combinations of fixed and adaptive hardware and application respectively (e.g., FH/FA denotes fixed hardware running the baseline application that does the full amount of work, FH/AA denotes fixed hardware and adaptive application). To understand the limitations of our predictors, we also ran the adaptive hardware/adaptive software case with an "oracle" predictor (denoted "Oracle"). This predictor provides exact values for energy and instruction count for each application configuration by encoding the next frame using each possible application configuration. It allows a locally-optimal selection of the application configuration to be made. However, because the configuration selected for each frame affects all future frames, the configuration chosen by the oracle may not be globally optimal. Finally, we also tested a system where the adaptive hardware ran with a fixed application configuration where the thresholds are chosen to minimize energy across the specific encoded sequence, denoted "Best-Fixed."

In the case of the "inexpensive" CPU, network energy is dominant; therefore, the adaptive application gives only small benefits. Focusing on the systems with adaptive hardware, the difference between the fixed application doing the full work, the adaptive application using the linear predictors, and the adaptive application using the oracle predictor is only approximately 1% for *Carphone*, and 3% for *Foreman*. However, the fixed application with the best threshold (with a modest motion search threshold and the DCT threshold disabled) outperformed both of the adaptive application systems (AH/AS and Oracle) by a small amount – 3% to 5%.

However, in the case of the "expensive" CPU – which is much closer to the energy consumption of current general-purpose processors – the value of the hardware/application co-adaption becomes clear. In this case, the network energy and the CPU energy both contribute significantly to the total energy, so a meaningful trade-off can be made between these subsystems. We now observe that the addition of application adaptation saves significant amounts of energy. On the adaptive hardware, the addition of application adaptation saves an additional 20% or so more energy for *Carphone*, and around 15% more for *Foreman*. Comparing AH/AS with Best-Fixed, we also observe that the flexibility to choose different configurations on a frame-by-frame basis provides an additional energy savings of approximately 5% for our adaptive system when encoding *Foreman* (which consists of an initial "talking head" segment followed by a more complex "pan-and-zoom" segment).

Once again, we observe that using the predictions for the instruction and byte count instead of the actual values reported by the "oracle" predictor incurs an energy penalty of only approximately 1% for *Carphone*, and 3% for *Foreman*.

6. CONCLUSIONS

We have shown that application adaptation can provide significant energy reductions over both fixed and adaptive hardware. Furthermore, we have achieved joint hardware-application adaptation while preserving the logical separation between layers. Our application does not know what hardware configuration will be used and vice versa. By keeping the hardware and application adaptation processes logically separate, we can simplify the end-to-end adaptation process. Currently, this work accepts the amount of CPU time allocated to the adaptive encoder as an input from a higher-level resource manager. Future work will address the design of this resource manager, which will allow this decoupled approach to be extended to multiple simultaneously-executing applications. Other future work includes more accurate modeling of the network environment.

7. REFERENCES

[1] V. Bhaskaran et al. "Motion estimation using a computation-constrained criterion." *Proc. 13th International Conference on Digital Signal Processing (DSP '97)*, Vol. 1, pp. 229-232, Santorini, Greece, 1997.

[2] S. V. Adve, et al. "The Illinois GRACE project: Global Resource Adaptation through CoopEration." *Proc. Workshop on Self-Healing, Adaptive, and Self-Managed Systems (SHAMAN)*, June 2002.

[3] W. Yuan, K. Nahrstedt, S. Adve, D. Jones, and R. Kravets. "Design and Evaluation of A Cross-Layer Adaptation Framework for Mobile Multimedia Systems." *Proc. SPIE/ACM Multimedia Computing and Networking Conference (MMCN'03)*, Santa Clara, CA, January 2003, 1-13.

[4] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen, "A dynamic voltage scaled microprocessor system," *IEEE Int. Solid-State Circuits Conf.*, Nov. 2000.

[5] C. J. Hughes, P. Kaul, S. V. Adve, R. Jain, C. Park, and J. Srinivasan. "Variability in the Execution of Multimedia Applications and Implications for Architecture." *Proc. 28th Intl. Symp. on Computer Architecture*, 2001, pp. 254-265.

[6] C. J. Hughes, J. Srinivasan, and S. V. Adve. "Saving Energy with Architectural and Frequency Adaptations for Multimedia Applications." *Proc. 34th Intl. Symposium on Microarchitecture (MICRO-34)*, Dec. 2001, pp. 250-261.

[7] J. R. Jain and A. K. Jain, "Displacement measurement and its application in interframe image coding," *IEEE Trans. Communications*, vol. 29, no. 12, Dec. 1981, 1799-1808.

[8] C. J. Hughes, V. S. Pai, P. Ranganathan, and S. V. Adve, "RSIM: Simulating Shared-Memory Multiprocessors with ILP Processors," *IEEE Computer*, vol. 3, no. 2, Feb. 2002.

[9] D. Brooks, V. Tiwari, and M. Martonosi. "WATTCH: A Framework for Architectural-Level Power Analysis and Optimizations." *Proc. 27th Intl. Symp. on Computer Architecture*, 2000.

[10] L. Feeney, and M. Nilsson. "Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment." *Proc. IEEE INFOCOM*, 2001.

[11] A. Buyuktosunoglu, D. Albonesi, S. Schuster, and D. Brooks. "A Circuit Level Implementation of an Adaptive Issue Queue for Power-Aware Microprocessors." *Proc. 11th Great Lakes Symposium on VLSI*, West Lafayette, IN, 2001.