

Super DVS: Enabling the Next Order Energy Reduction through Algorithm and Architecture Parallelization *

Jinfeng Liu, Pai H. Chou, Nader Bagherzadeh
Dept. of Electrical & Computer Engineering
University of California at Irvine
Irvine, CA 92697-2625 USA
{jinfengl, chou, nader}@ece.uci.edu

Abstract

Dynamic voltage scaling (DVS) is a popular approach to power and energy reduction in microprocessors: it not only reduces the average power level, but also increases the processor's energy efficiency (i.e., lower energy per instruction). However, DVS will not be a fruitful technique for further energy reduction based on today's single processor assumption. In fact, it may actually lead to higher energy consumption due to the lack of system-level considerations. We propose to exploit system-level parallelism as an effective means to achieving the next order of energy improvement for the entire system. By using multiple processors to perform the same task, each processor has reduced workload and thus can run at even more energy efficient points beyond the limits of today's best DVS without sacrificing performance. A secondary effect is that the smoother power profile and reduced peak current not only reduce time/power overhead associated with voltage scaling, but also extend battery life. Experimental results on an image processing algorithm show a 60% reduction in energy and 80% in power compared to the best DVS approach.

1 Introduction

Dynamic voltage scaling (DVS) is a well-studied technique for minimizing energy consumption in embedded microprocessors. Many modern processors are designed to operate at different voltage and frequency settings as an effective way to manage power usage. It is advantageous to operate the processor at a lower voltage whenever possible, because energy is proportional to V^2 . In other words, it takes less energy to execute an instruction at a lower voltage, even though the total execution time is longer. DVS techniques exploit slacks in the task set by slowing down the processor just enough without violating the deadline. Many DVS techniques have been proposed to overcome the limits in slowing down the processor, including inter-task DVS, intra-task DVS, slack borrowing, etc.

1.1 Limits of DVS

DVS has been studied extensively for single processors. However, DVS is running against several fundamental limits. Due to variation in workload, it is not always possible to run the processor at the lowest possible constant speed at all times; for example, different types of MPEG frames require a wide range of processing. This not only prevents the processor from operating at the optimal rate, but also forces the processor to pay power and timing overhead associated with voltage scaling. In fact, as processor performance increases, the voltage scaling granularity decreases, and as a result, as much as 30% of the energy could be spent on scaling overhead alone. Unfortunately, most DVS techniques proposed to date do not consider any overhead.

Another limit is DVS's assumption about a single processor, even though many realistic system include peripheral devices controlled by the processor, but this dependency is never modeled. In fact, the power consumption at the system level may be dominated by these peripherals, rather than the processor, and it is crucial to exploit shutdown opportunities (especially communication interface) in peripherals. However, most peripherals are not as power manageable as the processor. As a result, the DVS approach to increasing the processor's power efficiency at the expense of performance can actually keep the dependent peripherals in the same high-power operating mode for a longer period of time. That is, not only is the power saving due to DVS limited to its percentage contribution to the entire system, it can actually result in higher overall energy consumption.

1.2 Beyond DVS limit

To further improve energy efficiency beyond DVS, we break the limits of DVS described above. First, we must improve power efficiency without sacrificing performance; otherwise, we will repeat the pitfall of increasing energy consumed by dependent peripherals. Second, we must decrease the dynamic range of the power of the processor, even though the workload has a wide dynamic range, in order to maximize its energy efficiency (in terms of Joules per instruction). This can be achieved by exploring the *granularity* for voltage scaling, where coarser grain scaling results in lower overhead.

In this paper, we propose to break the DVS limits by exploring system-level pipelining. We improve energy efficiency by applying voltage scaling to each code partition, but make up for the performance loss with parallelism. We make the observation that n processors running at $1/n$ speed will be significantly more energy efficient than a single processor running at the speed determined by today's best DVS techniques. It not only significantly reduces the energy consumed by the processor, but also results in much lower peak power and a smoother power profile, both of which are attractive features for batteries. We also determine the optimal granularity for voltage scaling so as to balance adaptivity with the adaptation cost.

The grand challenge will be to extract parallelism in the application so that it can be mapped onto a multi-processor architecture with a much higher power efficiency as a whole system. Parallelism extraction from a sequential program like C is a very difficult problem in general, but fortunately data regular applications can be described in a variety of data flow models that are amenable to mapping onto architectures with multiple processors. Another challenge is to design the architectural features to enable the processors to compose with each other efficiently.

This paper uses an automatic target recognition (ATR) example to demonstrate the next order of magnitude power/energy saving beyond DVS for data regular applications. We show our parallelized code with shared memory communication, and we discuss

*This research was sponsored by DARPA under contract F33615-00-1-1719

the software run-time support in the form of adjusting references for each pipeline stage.

2 Related Work

Dynamic voltage scaling technique has been studied extensively recently. Researchers have addressed DVS related issues in the following aspects.

Real-time scheduling has been extended to DVS scheduling on variable-voltage processors. A few analytical models have been proposed. Weiser et al. proposed the initial scheduling model in [14], and the aspect of energy minimization was analyzed by Yao et al. and the optimal off-line schedule is given in [15]. Hong et al. proposed an off-line scheduling heuristic for non-preemptive systems in [2] and an on-line algorithm for mixed workload of both sporadic and periodic tasks in [3]. Ishihara et al. analyzed the optimal schedule for a processor that can operate in several discrete voltages in [6] and proposed an integer linear programming solution. Okuma et al. proposed a DVS scheme [7] that always guarantees deadlines for all tasks, although the energy may not be optimal. Shin et al. presented a run-time checking mechanism that can shut down the processor or adjust the processor speed [11] and an algorithm to minimize energy for periodic tasks [12]. Quan et al. improved this technique by finding an optimal schedule for both periodic and sporadic tasks [9].

More realistic DVS models have been proposed to consider design issues, e.g. DVS overhead and the presence of the scheduler. Hong et al. presented a synthesis design flow for variable-voltage processor cores [4]. The impact of DVS overhead is studied with a few scheduling schemes. It turns out that the analytical models may not be yield optimal solutions when overhead is taken into account. Burd et al. presented an implementation scheme for a microprocessor with DVS capability [1]. In [8] Pering et al. simulated different DVS scheduling algorithms and considered the presence of the scheduler as a part of the system, and the DVS overhead is also examined.

Some researchers have applied DVS to embedded applications, where the processor only deals with one or a few specific tasks. Shin et al. proposed an intra-task DVS scheme [10] that tries to maximally utilize the slack time within one task, as opposed to inter-task DVS scheme that borrows slack time from finished tasks. Im et al. proposed an inter-task DVS scheme for multi-media applications in [5] based on the observation that the slack time cannot be utilized when no task is available. The solution is to buffer the tasks such that the slack time can be used when the buffer is not empty.

3 Motivating Example: ATR

We use an automatic target recognition (ATR) algorithm [13] as our motivating example. Its block diagram is shown in Fig. 1. The algorithm is described in Fig. 2. It takes a sample image as the input. The TARGET DETECTION module detects M targets on the original image. For each target, a region of interest (ROI, which is a smaller image surrounding the target on the original image) $roi0$ is extracted and Fourier transformed by FFT module to space-frequency domain. The transformed $roi1$ is multiplied by a few predefined templates, then Fourier transformed inversely by IFFT module back to space domain. For each template, the resulting $roi2$ has a specific attribute Val that indicates the distance of the target. If it is larger than a threshold value, $roi2$ is fed to the COMPUTE DISTANCE module, an image processing routine to calculate the distance.

The timing requirement to the ATR algorithm is to sustain a frame rate, the number of images processed per unit time. Its inverse is called *frame delay*, which is the maximum delay time to process each frame. The goal is to reduce the energy consumption of the processor for a given frame delay.

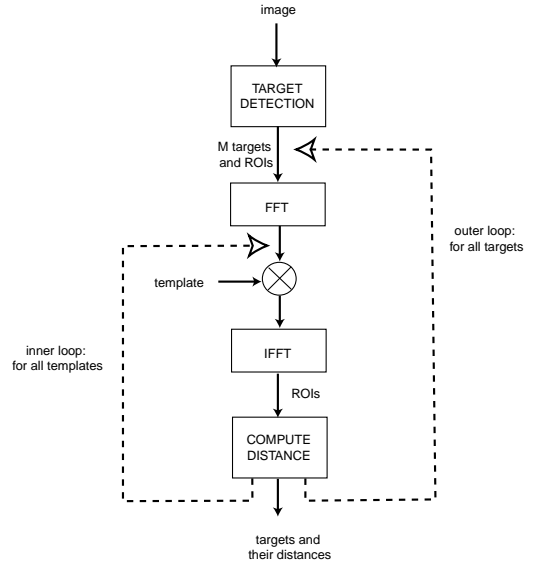


Figure 1: Block diagram of the ATR algorithm

```

ATR(image im, array TEMPLATE, float Valth)
  TARGET := TARGET DETECTION(im)
  A: for each (target ∈ TARGET) loop
    roi0 := getRoi(im, target)
    roi1 := FFT(roi0)
    for each (template ∈ TEMPLATE) loop
      roi2 = IFFT(roi1 × template)
    B: if roi2.Val > Valth then
      COMPUTE DISTANCE(roi2)
    end if
  end loop
end loop
format data and return computed distance

```

Figure 2: The ATR algorithm

One key characteristic of the algorithm is that its execution delay falls in a diverse range with regard to M , the number of targets detected by TARGET DETECTION module, since FFT and IFFT are quite computation-intensive. In addition, the *if* statement at label B also affects the distribution of the execution delay due to another computation-intensive module COMPUTE DISTANCE.

Among the current DVS techniques, *intra-task DVS* [10] is most suitable for this problem. By applying this technique, static timing analysis is performed at compile time to insert additional annotations at labels A and B to compute remaining the worst-case workload (in worst-case instruction count, *WCIC*). In the beginning, the remaining *WCIC* is the *WCIC* of the whole algorithm. At A, the remaining *WCIC* is computed based on number of targets M . At B, the outcome of the conditional branch will modify *WCIC* by subtracting the instruction count (*IC*) of COMPUTE DISTANCE module. The *ICs* of other modules are fixed for fixed-size images, e.g., 128×128 . Therefore, A and B are the only places where the remaining *WCIC* could decrease.

During the execution of the algorithm at run-time, when a new frame arrives, the processor speed is set based on the frame delay and the *WCIC* of the whole algorithm. As the program execution reaches A or B, the remaining *WCIC* is updated and the processor adjusts its frequency and voltage based on the remaining time and *WCIC*.

It is notable that the voltage at point B could scaled made many

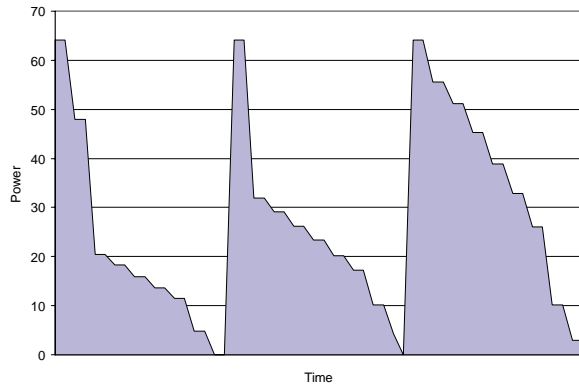


Figure 3: Power profile of intra-task DVS

times for each image frame, since B is located inside the inner loop. In our current analytical study, we set the maximum number of targets to 7, so there might be 0-7 targets in each frame. The number of matching templates is 3. As a result, B could be reached up to 21 times for one frame. Therefore, to process each frame, the processor may change its voltage/frequency up to 23 times, including the initial maximum speed setting and the speed drop after A.

A typical power profile for intra-task DVS is shown in Fig. 3. It always starts with a high power spike at the beginning of each time slot for one frame, and then steps down once at A, followed by a few more drops each time when the *if* condition evaluates to *FALSE* at B.

Intra-task DVS appears to be the right solution for reducing processor energy for application-specific algorithms. However, there are still some issues that need to be addressed for more energy efficient designs. For example, there is a high power spike at the beginning of each time slot, because the initial processor speed is always set at a high speed for the worst case. In most cases the worst-case instruction count *WCIC* is not a good measure of the typical workload of the application. Therefore, what is seen in the power profile chart is a few sharp slumps after the initial spike at peak power. By the time when one frame is about to be finished, the processor normally operates at a very low power level; and it must switch to high power for the next frame in anticipation for the worst-case workload.

A power profile in such a pattern has several undesirable properties. High peak power is known to have many negative effects, e.g. it shortens the battery life. In addition, frequent switching between high-power mode and low-power mode will incur an overhead on both time and energy when the processor is changing its supply voltage and frequency. Although most DVS studies do not consider this overhead, it can be non-trivial when the voltage change is significant. Finally, the power profile with high jitters is not very energy efficient compared to a smooth one.

One potential solution to this problem is to extract the parallelism in the algorithm and distribute the workload into multiple processors. Successful parallelization can reduce both power and energy by running only a part of the code on each processor at a slower speed. Even though more processors are involved, the overall energy and power consumption will still be reduced significantly. However, it is generally difficult to parallelize serial algorithms. In addition, parallel algorithms require sophisticated support for synchronization, communication (e.g. cache coherence), etc., which may consume even more energy than the energy saved by parallelization.

Another solution is to find one or more “average” speeds to execute the algorithm such that the power profile is smoothed by lowering the height of the power spike; meanwhile, the energy will

be also reduced. In an “ideal” solution, the minimum energy is achieved by running the processor at a constant speed such that the task is finished just before the end of the allocated time slot. However, the value of this optimal speed must be determined at the beginning to process the task, but it cannot be known until the last instruction (actually the last branch instruction) of the algorithm is completed. Thus, such an “ideal” solution is generally not possible. A partial solution is to partition the algorithm into several segments such that an average speed can be found for each segment. The energy spent within the segment is reduced; and it also helps to lower the power spike.

4 Super DVS: Energy Efficiency through Parallelism

We present our new *super DVS* technique in Section 4.1 to improve the energy efficiency of the ATR algorithm. It explores parallelism of the algorithm by first partitioning the code, and then pipelining the execution of all partitions. This allows to distribute the workload to multiple processors running at slower speeds. For each processor, we apply DVS and derive the optimal constant speed that minimizes energy and peak power at the same time. The new technique is called *super DVS* in the sense that DVS is applied to each small partition of the code. Parallel execution normally needs special treatment for communication. We propose a simple data handling technique in Section 4.2 by managing FIFO buffers between processors. In Section 4.3, we propose an additional scheme that processes multiple frames together to further reduce energy consumption.

4.1 Super DVS

We first partition the algorithm into a few independent segments such that the average speed can be found for each segment. Then, the partitioned code can be pipelined in a multi-processor architecture. Finally, DVS is applied to each processor to achieve energy and power reduction.

4.1.1 Partitioning: finding a constant speed for each partition

With intra-task DVS, the code of the whole algorithm is divided into segments that are executed at different speeds. It is sometimes preferable to find a constant speed to reduce energy consumption. Although such a goal is normally difficult for the entire algorithm, finding the average speed for a portion of the algorithm is possible. We observe that such possibility is largely dependent on the way in which the algorithm is composed.

When intra-task DVS is applied to the ATR algorithm, due to the frequency changes at A and B, many different speeds will be applied to run the algorithm. Even the same code segment can be executed at different speeds. For example, if there are 5 targets detected for a given frame, module FFT will be executed 5 times, possibly at different speeds. However, because there is no loop-carried dependency across different iterations of the outer loop (as well as the inner loop), we can reorder the execution of the loops to execute those 5 instances of FFT together at a single speed. Similarly, the potential constant speeds can also be applied to IFFT and COMPUTE DISTANCE.

The ATR algorithm must be reconstructed to apply this technique. In the two-level nested loop, the iterations on both inner loop and the outer loop are executed to completion. Although this is the natural way to exercise loops, we have to partition the loop and reorder the sequence to access FFT, IFFT and COMPUTE DISTANCE for finding the average speed for each segment.

We reconstruct the algorithm and partition the two-level nested loop into three stages. In *STAGE1*, the algorithm finishes FFT for all M ROIs. Then, IFFT for M ROIs with different templates is performed in *STAGE2*. Finally, those K ROIs whose target distances

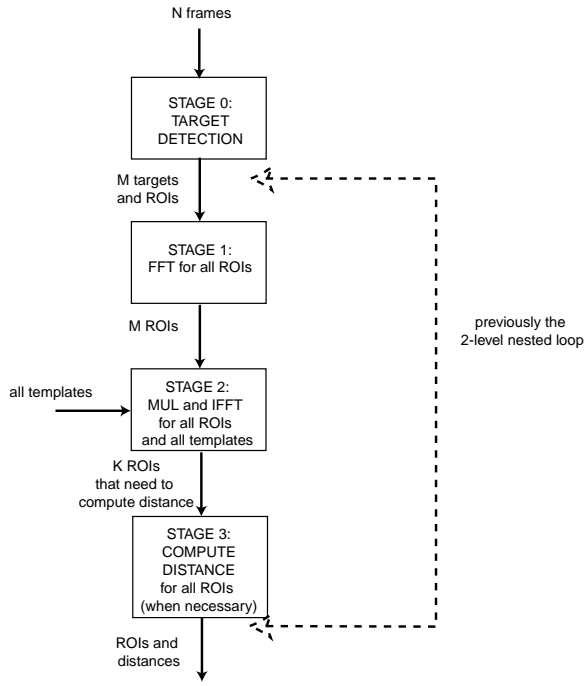


Figure 4: Partition the nested loop into stages

need to be calculated are processed together in STAGE3. The modified block diagram is shown in Fig. 4. At the beginning, M targets are detected in STAGE0. Thereafter, the ROIs for these targets flow through the three following stages.

After the code transformation, the entire algorithm can be executed at four constant speeds (actually three speeds: STAGE2 and STAGE3 can be executed in the same speed since there is no conditional branch between them) for four stages. Special treatment must be applied to STAGE3 to achieve a constant speed. The *if* condition can be resolved at STAGE2 after IFFT by attaching some flags to each ROI indicating whether computing distance is needed. Therefore, the remaining workload $WCIC$ of STAGE3 ($WCIC(stage3) = K \times IC(COMPUTEDISTANCE)$) is known before it is executed such that the average speed can be calculated at the beginning of STAGE3.

Through analysis, we observed this code partitioning and transformation technique can moderately reduce energy by 5% - 15%. The reduction largely depends on the input data set and the size of each partition. The detailed discussion is outside the scope of this paper. We still present the sketch of this technique because it naturally leads to a parallel version of the algorithm.

4.1.2 Parallelization through pipelining

Parallelization on the algorithm can be helpful for power management. For example, if an algorithm can be parallelized into two processors with each processing half workload and running at half speed, an approximately $4 \times$ energy reduction will be achieved even after one new processor is added. However, in general it is quite difficult to parallelize an algorithm that has already been implemented in a serial style.

In the ATR algorithm, we observed that there is no data dependency between different image frames. Therefore, the code partition in Fig. 4 is ready to be executed in parallel, by pipelining the execution of all stages on different processors. After pipelining the algorithm, each stage itself will have the execution time of one entire frame delay, while all four stages together consume one frame

```

STAGE0(time delay, image im, buf ROI0)
  setProcessorSpeed(delay/WCIC(STAGE0))
  TARGET := TARGET DETECTION(im)
  M := 0
  for each (target ∈ TARGETS) loop
    ENQUEUE(ROI0, getRoi(im,target))
    M := M + 1
  end loop
  return M

STAGE1(time delay, buf ROI0, buf ROI1, int M)
  setProcessorSpeed(delay/(M * WCIC(L1)))
  for i := 1, M loop L1
    roi := DEQUEUE(ROI0)
    ENQUEUE(ROI1, FFT(roi))
  L1: end loop
  return M

STAGE2(time delay, buf ROI1, buf ROI2, int M,
  array TEMPLATE, float Valth)
  setProcessorSpeed(delay/(M * size(TEMPLATE) * WCIC(L2)))
  K = 0
  for i = 1, M loop
    roi0 := DEQUEUE(ROI1)
    for each (template ∈ TEMPLATEs) loop L2
      roi1 = IFFT(roi0 × template)
      if roi1.Val > Valth then
        K := K + 1
        ENQUEUE(ROI2, roi1)
      end if L2end loop
    end loop
  end loop
  return K

STAGE3(time delay, buf ROI2, buf ROI3, int K)
  setProcessorSpeed(delay/(K * WCIC(L3)))
  for i = 1, K loop L3
    roi := DEQUEUE(ROI2)
    roi.distance := COMPUTE DISTANCE(roi)
    ENQUEUE(ROI3, roi)
  L3: end loop
  return K

```

Figure 5: Parallel algorithms for super DVS

delay previously. As a result, more energy saving is available by slowing down the speeds of all processors. Ideally, if all four stages are perfectly balanced, that is, they have the same workload, the optimal energy reduction could be expected. However, this is generally not possible. For example, the workload of STAGE2 is about three times as much as that of STAGE1, since there will be three IFFTs for each FFT. Also, the workload of STAGE0 is fixed; while in other stages the workload will vary.

4.1.3 Super DVS: reducing energy in parallel

The final step is to apply DVS to each pipelined processor to improve energy efficiency. We call the new technique super DVS since it applies DVS to a smaller partition of the code. In Section 4.1.1 we already constructed the code partition for each stage such that the code can be executed at a constant speed. In Section 4.1.2 we extended the time slot for each stage as one frame delay for all processors. Therefore, we can find an optimal speed for each stage that maximally utilizes the extended time slot.

Fig. 5 describes the parallel super DVS algorithms for four pipeline stages, respectively. At the beginning of each stage, the processor speed is set according to the $WCIC$ to be executed, which is always known ahead of time. Therefore, each processor can set a near-optimal speed such that the execution of the code completes just in one frame delay.

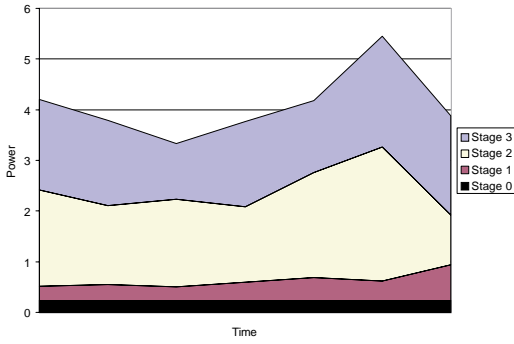


Figure 6: Power profile of super DVS

1. For STAGE0, the workload is set to $WCIC(STAGE0)$. The actual $IC(STAGE0)$ is only slightly different from $WCIC(STAGE0)$ since the module TARGET DETECTION is most computation intensive, while the instructions in the small loop to handle buffers are trivial. Therefore, when the speed of its processor is decided by the frame delay $delay$ and $WCIC(STAGE0)$, the time slot with a duration $delay$ is almost fully utilized. The speed to execute STAGE0 fixed for all frames if the same $delay$ is given. Stage0 computes the number of targets in variable M and put M ROIs into buffer $ROI0$.
2. For STAGE1, the processor speed is set according to M , which is the number of ROIs in buffer $ROI0$, $delay$, and the workload in loop L1 $WCIC(L1)$. This is because loop L1 will be executed M times during one frame delay, and the $WCIC$ of this loop can be analyzed statically. STAGE1 generates M ROIs in space-frequency domain after FFT, and puts them into buffer $ROI1$.
3. In STAGE2, the processor speed is set based on M , $delay$, $WCIC(L2)$ and $size(TEMPLATE)$, since STAGE2 will execute loop L2 in $M * size(TEMPLATE)$ times. Although there is an *if* block inside loop L2, the variance is trivial compared with computation-intensive IFFT. Therefore, using $WCIC(L2)$ to compute the average speed is almost optimal. This stage produces K ROIs to buffer $ROI2$, based on whether it is necessary to compute their distances.
4. Finally, in STAGE3, the processor speed is decided by K , $delay$ and $WCIC(L3)$. K ROIs are extracted from the input buffer $ROI2$ and distances are computed accordingly in loop L3. The results are queued to an output buffer $ROI3$.

The power profile for super DVS is shown in Fig. 6. During each period of one frame delay, each processor is running at a constant speed. Therefore, the overall power profile of all four processors is also constant during each period. Super DVS produces a much smoother power profile compared to intra-task DVS by eliminating the high peak spike. Significant savings on both power and energy are achieved by running the new parallel algorithms.

4.2 Implementation related issues: buffer management

In this section we discuss a few implementation related issues. As the parallel algorithm shown in Fig. 5, the four parallel processors communicate via FIFO buffers containing data to be produced and consumed. Our assumption is a shared memory organization, shown in Fig. 7.

In this memory organization, each stage produces new ROIs and appends them to the tail of its output buffer, which is the same as

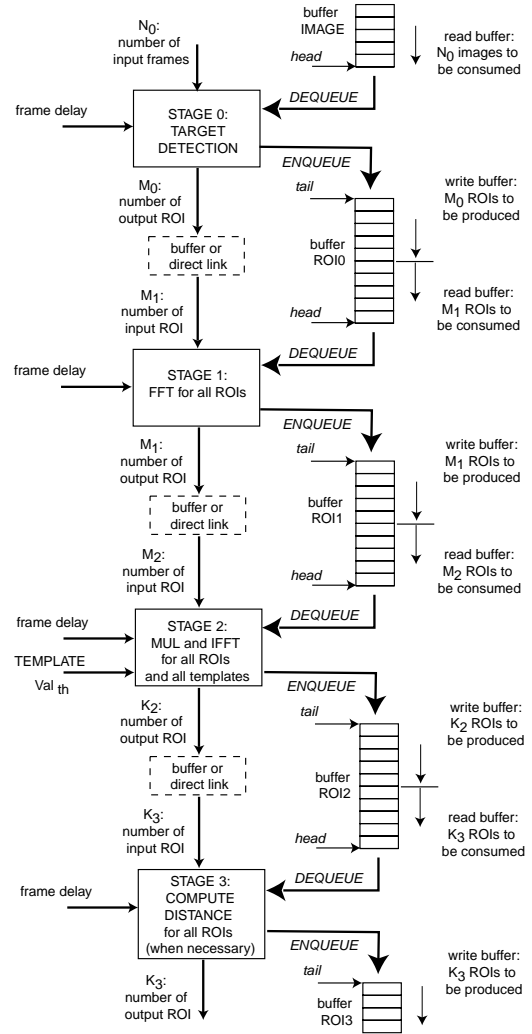


Figure 7: Pipelined processors with shared memory buffers

```

STAGE0(time delay, buf IMAGE, buf ROI0, int N)
setProcessorSpeed(delay/(N * WCIC(L0)))
M := 0
for i := 1..N loop
  im = DEQUEUE(IMAGE)
  TARGET := TARGET DETECTION(im)
  for each (target ∈ TARGET) loop L0
    ENQUEUE(ROI0, getRoi(im, target))
  M := M + 1
L0: end loop
end loop
return M

```

Figure 8: Modified STAGE0 to process N frames at a time

the input buffer of the next stage. Because the buffer size to be consumed next is passed to the consumer stage as the boundary of the input buffer, when the consumer stage fetches data from the head of the buffer, it will not access the data that is currently being generated by the producer. The variables to indicate workload M and K can be directly passed through to the next processor, or they can be organized in some other buffers in the same manner. The variable *delay* is a constant or it can be set by external events, e.g. a user interface. It also can be accessed in separate FIFO buffers to avoid simultaneous access.

This buffer management scheme significantly simplifies the communication between parallel processors. Fig. 7 shows that, at any moment during the execution of this multi-processor system, any data in ROI buffers is accessed exclusively by at most one processor. Simultaneous accesses to the same data from two or more processors will never happen. (If variables $M, K, delay$ are also organized by separate buffers, there is no simultaneous access to these variables as well.) This suggests the data access is simplified to an extent that the processor does not need to acquire and release locks to access critical sections, since there is no critical section at all. For the same reasons, the cache coherence protocol is not necessary to implement such a system, if each write to the buffers will always commit to the shared memory. In fact, in many embedded DSP applications, the processor may not have caches.

By this simple buffer managing technique, we can avoid some sophisticated data handling mechanisms in general-purpose multi-processor systems, since they are not necessary in this specific application. The implementation of such a system is also easier than a general-purpose multi-processor system.

Although this parallel solution can reduce the energy of the processors, it may increase the energy in the memory system. For example, the memory must be designed to be multi-ported, which will increase the energy consumption. As many DVS studies do not consider the impact of the memory system, we also focus on the energy reduction to the processors.

DVS control is made very easy for this multi-processor system. Since all processors are working on totally independent data sets, the DVS control of each processor is independent to each other, as far as they all finish in allocated time slots with the same duration *delay*. In the shared memory organization, the four running processors with different speeds pose some challenges to the memory designer. The memory system must accommodate simultaneous accesses from processors at different speeds.

One alternative solution is to use some processors that can directly pass its produced data to the consuming processor. Each processor has built-in input and output buffers to hold its local data set. The entire content of the buffer can be read/written from/to other processors. If applicable, this type of processor is an appropriate choice to implement ATR algorithm with super DVS technique. A sketch of directly linked data communication is shown in Fig. 9

4.3 Coarser granularity: processing multiple frames together

Our super DVS scheme reduces the energy and power of the processors by introducing a 3-frame delay for each frame. If more delay is allowed, we have an additional scheme that can save even more energy.

We have discussed previously that it is preferred to find an average processor speed at a coarser granularity (more workload). We have already applied this technique in Section 4.1.1. We observed that different frames will be processed at different speeds in STAGE1, STAGE2 and STAGE3 (STAGE0 always has the same speed), if we can find out the average speeds to process multiple frames, we can expect more energy reduction.

Such an addition is a straight-forward extension to the existing scheme. The only change is that STAGE0 will now fetch N frames

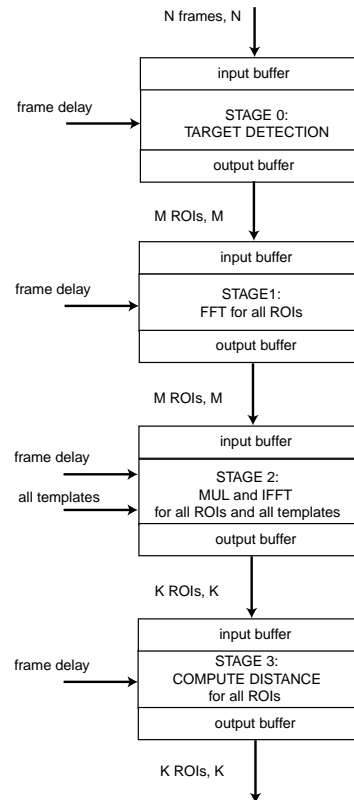


Figure 9: Pipelined ATR with directly linked data connection

from its input buffer, which contains multiple frames; and the *delay* will be changed accordingly by N times for N frames. (Both N and updated *delay* are set externally.) The delay for each frame is now ranging from $3N$ to $4N - 1$ frame delay. The modified algorithm for STAGE0 is shown in Fig. 8. No change to the other stages is necessary.

In this paper we present two types of techniques: I. parallelization (Section 4.1.2), and II. averaging the processor speed at a coarser grain (Section 4.1.1, 4.3). Our new super DVS technique is a combination of these two techniques, plus DVS. It is notable that these new techniques and the existing power management techniques, including DVS (intra-task, and inter-task), static voltage scaling (SVS), shutting down idle components, and etc., are orthogonal techniques such that they can be applied either individually, or in combination. For example, if another parallel partition is available without finding an average speed for each processor, power and energy savings are still available. If DVS is not allowed, SVS can be applied to pipelined processors.

We will present some analytical results on energy reduction of super DVS technique in next section.

5 Analytical Results on Energy Reduction

In this section we present an analytical study to the new super DVS technique. We compare the results with the best-known existing technique, which is intra-task DVS.

5.1 An empirical processor model

We assume an abstract processor model for our analysis. A parameterized voltage-scalable processor is shown in Table 1. The peak power value is normalized to 100 to simplify the comparison. We

Parameter	Description	Value
V _{max}	the maximum supply voltage	3.3V
V _{th}	the threshold voltage	0.8V
F _{max}	the maximum frequency	1GHz
P _{max}	the maximum power consumption	normalized to 100 when N _{sw} = 1
T _{max}	the time overhead to switch between on and off	50 μs, 50k cycles at full speed
V, F, P	current voltage, frequency and power	to be calculated

Table 1: An abstract model of a voltage-scalable processor

Segment	WCIC (1000 instruction)	N _{sw}
L0	400	1
L1	170	1
L2	194	1
L3	377	1

(a) parameters of code segments

Series	Numbe of Frames	Numbe of targets per frame	Frame delay
A	200	0 - 2 (light workload)	16.7ms
B	200	5 - 7 (heavy workload)	16.7ms
C	200	random	16.7ms

(b) parameters of input data series

Table 2: Parameters of the code and input data

also assume all the four processors running pipelined ATR algorithms with super DVS are the same type of the processor as the one which is running the serial algorithm with intra-task DVS, although in practical concerns they will probably be different types of processors.

We use the following equations to compute parameter P, F, V .

$$F = C_f \frac{(V - V_{th})^2}{V} \quad (1)$$

$$P = C_p N_{sw} V^2 F \quad (2)$$

C_f and C_p are processor dependent constants. They can be computed by applying maximum values of V, F, P . N_{sw} is a factor indicating the number of switching activities per cycle, it depends on the program.

5.2 Properties of the algorithm and data set

Table 2(a) gives the instruction count of the loops in each stage. These numbers are used in both intra-task DVS and super DVS to update the workload. The four code segments do not show a large variance in power consumption, indicating that the factor N_{sw} is about the same for all segments. For simplicity we set them all to 1. We assume the frame rate is 60 frames per second. Therefore the frame delay is 16.7ms.

We apply three series of input images. The first series A has very few targets in each frame, ranging from 0-2. That is, the workload of the algorithm is far less than the worst case. The second series B has a heavy workload with 5-7 targets per frame. In set C, the number of targets is random. The input data sets are summarized in Table 2(b)

5.3 Power and energy reduction by super DVS

Table 3 shows both energy and (peak) power reduction of super DVS compared with intra-task DVS. We also vary the number of frames per image group to examine the effect of an increased granularity by averaging the larger workload among multiple frames. In all three input series, super DVS exhibits a 60% energy reduction; and the peak power can be reduced by as much as 80%-90%. We make following observations from the analytical results.

1. Intra-task DVS is not quite “low-power” in the sense that its peak power is always the same (the high-power spike) regardless of the workload, because it always starts from the worst case. On the other hand, super DVS adapts both energy and power to the workload very well.

Technique	Energy		Peak Power	
	value *	% reduction	value *	% reduction
Intra-task DVS	11.7		63.7	
Super DVS (N = 1 frame)	4.53	61.3%	3.17	95.0%
Super DVS (N = 2 frames)	4.45	62.0%	3.17	95.0%
Super DVS (N = 4 frames)	4.38	62.6%	2.89	95.5%
Super DVS (N = 8 frames)	4.34	62.9%	2.10	96.7%

Input image data: series A (light workload)

Technique	Energy		Peak Power	
	value *	% reduction	value *	% reduction
Intra-task DVS	84.4		63.7	
Super DVS (N = 1 frame)	33.4	60.4%	14.9	76.6%
Super DVS (N = 2 frames)	33.2	60.7%	12.8	80.0%
Super DVS (N = 4 frames)	33.1	60.8%	11.3	82.3%
Super DVS (N = 8 frames)	33.1	60.8%	10.7	83.2%

Input image data: series B (heavy workload)

Technique	Energy		Peak Power	
	value *	% reduction	value *	% reduction
Intra-task DVS	49.5		63.7	
Super DVS (N = 1 frame)	19.7	60.2%	18.8	70.5%
Super DVS (N = 2 frames)	17.9	63.8%	11.9	81.3%
Super DVS (N = 4 frames)	17.0	65.7%	10.9	82.9%
Super DVS (N = 8 frames)	16.4	66.9%	9.46	85.1%

Input image data: series C (random workload)

* energy and power values are normalized

Table 3: Energy and power saving achieved by super DVS

2. Super DVS can significantly reduce both power and energy compared with intra-task DVS. The percentage reduction to peak power is more than the saving on energy. This is because super DVS produces a more smoothed power profile, while the power profile of intra-task DVS always has high peaks and sharp jitters.
3. When the workload does not vary too much (series A and B), processing multiple frames together may not yield extra gains. This is due to the fact that the average speed in multiple frames is not quite different from speeds in individual frames. While in series C, where the workload varies in a diverse range, additional savings can be achieved at a coarser granularity.

5.4 The impact of DVS overhead

Not every DVS study has considered the DVS overhead, the overhead to change the voltage of the processor. Mostly they assume it is either free, that is, the frequency and voltage of the processor can be changed in zero time and zero energy, or the overhead is trivial. These assumptions are generally not true.

We propose an abstract model for an analytical study to see how much the overhead can impact the energy cost of DVS techniques. Our model is simple: we assume a parameter T_{max} , which is the maximum time overhead when the processor is switched between off ($F = 0$) and full speed F_{max} . T_{max} is a processor dependent constant. It indicates how quickly the processor can switch from one voltage/frequency setting to another. We assume $T_{max} = 50\mu s$, equivalent to 50,000 cycles at maximum clock speed.

For a frequency/voltage change from setting V_1/F_1 to V_2/F_2 , the time overhead is scaled by the frequency difference F_2 and F_1 , that is,

$$T_{V_1/F_1 \rightarrow V_2/F_2} = T_{max} \frac{|F_2 - F_1|}{F_{max}} \quad (3)$$

For simplicity, we also assume during the switching time, the average power overhead is the average of P_1 and P_2 , which refer to the power consumption before and after the change. The energy overhead can be computed accordingly. In reality, the energy spent

DVS granularity \ Techniques	Coarse-grain 100% workload 100% frame delay	Fine-grain 50% workload 50% frame delay	Very-fine-grain 10% workload 10% frame delay
Intra-task DVS	3.87%	7.52%	29.5%
Super DVS (N = 1 frame)	0.002%	0.004%	0.02%
Super DVS (N = 2 frames)	0.001%	0.001%	0.01%
Super DVS (N = 4 frames)	<0.001%	0.001%	0.005%
Super DVS (N = 8 frames)	<0.001%	<0.001%	0.003%

Table 4: Energy overhead vs. different DVS granularity

on DVS may be even higher. Some other models of the overhead to change processor voltage can be found in [1, 4].

We present the impact of voltage-changing overhead to intra-task DVS and super DVS in Table 4. Data series A is applied for this study. The first column of the results shows that intra-task DVS spends up to 4% of energy on changing voltage, which may not be considered as “trivial”; while such overhead for super DVS is less than trivial. We could try to increase T_{max} and see how sharply the overhead grows on intra-task DVS. However, because it may not be meaningful to have very large values of T_{max} for recent processors, we consider the equivalent cases by working on a smaller data set while fixing T_{max} . This refers to fine-grained DVS, in which cases the voltage changes are made quite frequently such that the DVS overhead is comparable to the slack time being saved by DVS.

In the ATR algorithm, suppose we want the algorithm to work on images with smaller size or less pixels. Therefore, the $WCIC$ of the program segments in Table 2 will all decrease. We assume they will decrease by the same factor, approximately. On the other hand, since the algorithm is dealing with less amount of data, its frame delay should be also decreased accordingly such that the performance of the algorithm is sustained (it is still processing the same amount of data in unit time). For example, if the image size is reduced by half (50% pixels), all the $WCIC$ in Table 2 should reduce by a factor of 50%, while the frame delay also reduces to its half. This is a reasonable assumption for fine-grained DVS.

In Table 4 these cases are studied to examine the impact of overhead when DVS is applied in finer granularities. It indicates that intra-task DVS may not be scalable very well to finer granularities with the sharply growing energy overhead. For example, in the third column, intra-task DVS will spend 30% of the energy on changing voltage. This is because intra-task DVS potentially have more frequent and more dramatic voltage changes. Especially, during each frame the processor must be switched from low-power to high-power once with larger time and energy overhead; and this overhead will be even more expensive in fine-grained cases. In contrast, the DVS overhead still remains trivial for super DVS. Super DVS incurs four voltage changes per frame, and these changes are quite smooth with small overhead. The overhead can be further amortized by processing multiple frames together where only four voltage changes are applied to N frames. As a result, intra-task DVS (and potentially inter-task DVS) may not be scalable to finer-grained applications, and will suffer more from the overhead on slowly-switched processors. On the other hand, super DVS will be still applicable to finer-grained applications or on processors that switch between different voltage/frequency settings slowly.

6 Conclusion

Increasing the level of parallelism is known to be beneficial for power management. However, this area is often overlooked in recent studies due to the difficulty to parallelize serial algorithms. In this paper we present a new technique that effectively parallelizes an image processing algorithm such that it could be pipelined in a four-processor system. The energy efficiency is improved by slowing down each processor; meanwhile the performance requirement

is compensated by increased parallelism. We propose a super DVS technique that combines the parallel approach with DVS. We successfully discover an optimal voltage setting for each processor to minimize both energy and peak power. The new technique will enable the existing DVS techniques to further reduce energy by the next order of magnitude. It also reduces the DVS overhead and is proven to be more scalable to finer-grained applications, where the overhead to change the processor’s voltage setting is very costly. We believe our new technique is generally applicable to a large class of signal processing applications with regular data access patterns. The work proposed in this paper represents one of the core CAD tools in a larger design framework for energy efficient embedded systems.

References

- [1] T. Burd and R. Brodersen. Design issues for dynamic voltage scaling. In *Proc. International Symposium on Low Power Electronics and Design*, pages 9–14, July 2000.
- [2] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. Srivastavas. Power optimization of variable voltage core-based systems. In *Proc. Design Automation Conference*, pages 176–181, June 1998.
- [3] I. Hong, M. Potkonjak, and M. B. Srivastava. On-line scheduling of hard real-time tasks on variable voltage processor. In *Proc. International Conference on Computer-Aided Design*, pages 653–656, November 1998.
- [4] I. Hong, G. Qu, M. Potkonjak, and M. Srivastavas. Synthesis techniques for low-power hard real-time systems on variable voltage processors. In *Proc. IEEE Real-Time Systems Symposium*, pages 178–187, December 1998.
- [5] C. Im, H. Kim, and S. Ha. Dynamic voltage scaling technique for low-power multimedia applications using buffers. In *Proc. International Symposium on Low Power Electronics and Design*, August 2001.
- [6] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proc. International Symposium on Low Power Electronics and Design*, pages 197–202, August 1998.
- [7] T. Okuma, T. Ishihara, and H. Yasuura. Real-time task scheduling for a variable voltage processor. In *Proc. International Symposium on System Synthesis*, pages 24–29, November 1999.
- [8] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proc. International Symposium on Low Power Electronics and Design*, pages 76–81, August 1998.
- [9] G. Quan and X. S. Hu. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *Proc. Design Automation Conference*, pages 828–835, June 2001.
- [10] D. Shin, J. Kim, and S. Lee. Low-energy intra-task voltage scheduling using static timing analysis. In *Proc. Design Automation Conference*, pages 438–443, June 2001.
- [11] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Proc. Design Automation Conference*, pages 134–139, June 1999.
- [12] Y. Shin, K. Choi, and T. Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *Proc. International Conference on Computer-Aided Design*, pages 365–368, November 2000.
- [13] R. Sims. Signal to clutter measurement and atr performance. In *Proc. of the SPIE - The International Society for Optical Engineering*, volume 3371, pages 13–17, April 1998.
- [14] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *USENIX Symposium on Operating Systems Design and Implementation*, pages 13–23, 1994.
- [15] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *IEEE Annual Foundations of Computer Science*, pages 374–382, 1995.