QUALITY BASED COMPUTE-RESOURCE ALLOCATION IN REAL-TIME SIGNAL PROCESSING

Joseph Yeh and John Wawrzynek

University of California at Berkeley, Department of EECS, Berkeley, CA 94720
jyeh,johnw@eecs.berkeley.edu

ABSTRACT

We present a novel method for controlling the complexity of real-time signal processing computational tasks, in order to make sure that a total quality metric for all the signal processing tasks is maximized. The method makes decisions about how much compute power is allocated to each task through past observations of the input and output data of each task. We present preliminary results from filtering applications that demonstrate the ability of the system to maximize the total quality of a large number of tasks under a real-time computational constraint.

1. INTRODUCTION

Although the performance of computing devices continues to grow, new multimedia applications still arise to challenge the capabilities of these devices, particularly media compression algorithms such as H.264 and MPEG-4. At the same time, recent technological and market trends are pushing devices, particularly portable wireless communications devices, to handle a larger number of computational tasks while consuming minimal power. Many of these computational tasks have real time constraints, and therefore cannot afford to be delayed. We present a framework in which such computational tasks can be performed at different levels of quality, with higher levels of quality requiring more computational resources in terms of percentage of CPU time in scalar architectures and/or functional units in highly parallel architectures, where the levels are set by a evaluator which takes into account the computational constraint of the system, and the characteristics of the input and output signals.

This framework needs to be robust to changes in the nature of the input signals and the overall computational constraints. Therefore, it does not rely on any *a priori* estimates of quality for a given implementation. Instead, it relies on observations of task inputs and task outputs to both establish the current tradeoffs between computational costs and quality and allocate computational resources among different tasks appropriately. Furthermore, the framework itself is computationally efficient.

Much work, by Goel [1] and Chandrasekharan [2] in particular, has been done to minimize power consumption

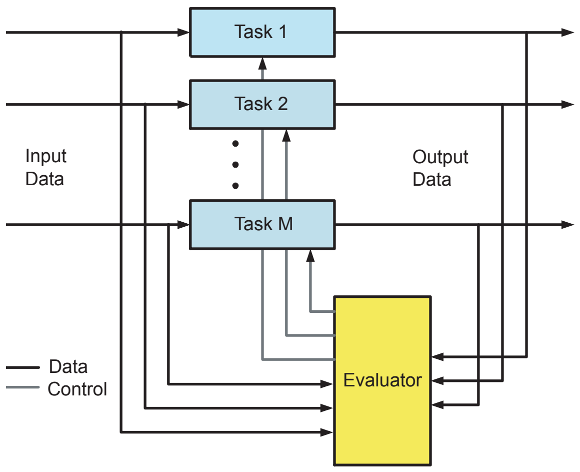


Fig. 1. System Diagram

or instruction count for a given level of quality (equivalent of signal processing performance in [1]). We investigate a *dual* problem of maximizing quality for a given computational speed, corresponding to a constant level of power consumption in a conventional computer architecture. In our formulation, there is no need for any extra hardware overhead to switch power on or off to processing units, making it more versatile and architecture-independent.

The next section describes the mathematical framework and terminology used in our model. Section 3 then introduces an example application for the system and section 4 presents some preliminary results. Section 5 presents possibilities for future directions and Section 6 concludes the paper.

2. FRAMEWORK

The basic block diagram of the system is shown in Figure 1. The computational device is running M *tasks* simultaneously. Each of the tasks can be run at a certain *implementation level*, and each implementation level of each task has a *computational cost* of $c_m(l)$, where m is the task number and l is a non-negative integer corresponding to the implementation level of the task. The tasks are ordered so that $c_m(l)$ increases with l. In a particular time period, the total cost $\sum_{i=1}^{M} c_m(l)$ of the selected implementations of all tasks cannot exceed C, the total amount of computation available. While the device is running the tasks, the *tradeoff mechanism* maintains state variables $b_m(l)$, which are estimates of the *benefit* offered by each particular implementation level of each task. This benefit represents an abstract quantity whose meaning is dependent on our overall objective in allocating computational resources.

In a given time period, implementation levels l_m have been chosen, and the inputs and outputs of each task are available to the tradeoff mechanism. For each time period, for each task m, we make I_m observations of the input $x_m[n]$ and O_m observations of the output $y_m[n]$. After all observations are made for the time period, we assess the quality of each task through a function $Q_m(x_m[n], y_m[n])$.

These observations are then used to adjust the implementation levels of the tasks. Currently, we use a two phase method to do this adjustment. First, based upon the quality assessments Q_m, we adjust benefit estimates $b_m(l)$ for the implementation levels of each task. Then, we solve a *multiple-choice knapsack problem* to determine which implementation levels l_m are chosen for the next time period. More specifically, we try to choose l_m for all tasks in order to solve the following optimization problem:

$$\text{maximize} \quad \sum_{i=1}^{M} b_i(l_i) \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{M} c_i(l_i) \leq C \tag{2}$$

Although this is a NP-hard integer linear programming problem, this particular optimization problem has been well-investigated, and will not pose a significant burden on the computing platform. Techniques by Pisinger [3] were able to solve problems with 100 tasks and 100 implementation levels for each task in 0.33 seconds on a 66 Mhz machine; common desktop machines today have clock speeds more than ten times higher. Even if the machine performance does not scale as much, it will be feasible to solve the problem at least once per second.

In developing this system, we first consider systems in which all tasks are similar and Q_m is the same function for all m, as in the example presented below in Section 3, where we define a specific kind of Q_m and method for modifying $b_m(l)$. We plan to eventually extend the system to consider different tasks and quality metrics.

3. EXAMPLE

3.1. Setup

We demonstrate our system's applicability by considering a simple example where all tasks are FIR filters, and different implementation levels involve different approximations to an ideal filter. In this example, the computational resource being allocated is the number of shift-add combinations that can occur to produce one output observation from each of the filters. All the FIR filters are approximations to whitening filters matched to AR (auto-regressive) processes. Each of the AR processes are generated by passing white noise through a monic all-pole filter $H_{im}(e^{j\omega})$ (for convenience, all the poles are located inside the unit circle), so each task involves filtering a different AR process generated with a different filter. We choose this particular application of removing a signal's redundancy because a big portion of audio and video compression revolves around exploiting the redundancy inherent in natural phenomena.

For each task, the highest level implementation corresponds to the ideal whitening filter $H_m(e^{j\omega}) = 1/H_{im}(e^{j\omega})$, with a time domain impulse response $h_m[k]$. We implement the filter as a cascade of second order (three tap) subfilters $h_{ma}[k]$, with $H_m(e^{j\omega}) = \prod_a H_{ma}(e^{j\omega})$, in order to maximize the overall mathematical precision for a given wordlength [4]. We then break down the individual taps in the subfilters into CSD (canonical signed digit) form [5], representing them as sums and differences of powers of two; each power of two term will represent a shift-add operation required to implement that tap. We then generate lower level implementations for each filter (constraining all implementations to have a first tap $h_m[0]$ of one by constraining all subfilters $h_{ma}[0]$ to have first tap of one) by selecting powers of two to delete from the subfilter taps.

We perform this selection by calculating for each subfilter T_{ma}, the total powers of two in $h_{ma}[1]$ and $h_{ma}[2]$ combined. For $t = 1$ through T_{ma}, we determine the best t powers of two to delete by exhaustively searching over all possible combinations of t powers of two and selecting one that minimizes an error function. After this is done for all subfilters, we determine for $u = 1$ through T_m, where $T_m = \sum_a T_{ma}$, the best u powers of two to delete from all the filters collectively. Finally, for every filter, the implementations are ordered based upon how many powers of two they contain. The more powers of two the filter has, the more it approximates the ideal filter.

3.2. Experimental Parameters

We expect the variance of the filter output to be at a minimum when the filter is at the highest level implementation. The more the transfer function of the filter deviates from the ideal, the greater the variance of the filter output will be. Therefore, we define the quality of the filters to be the negative of the estimated sample variance of the filter outputs; $Q_m = -(\sum_{n=1}^{N} y_m^2[n])$ for a given time period. In this example, N is equal to both I_m and O_m for all tasks m. When utilizing the Q_m to modify the $b_m(l)$ functions, we normalize them by a factor α to increase our confidence in them as quality estimates.

For this example, we wish to have the overall objective of achieving the same quality from all the tasks, or to have the Q_m as close to each other as possible. At the start of our simulation, all of the $b_m(l)$ are initialized to zero, and we modify them as such:

- Find task numbers r and s such that $Q_r \geq Q_m$ and $Q_s \leq Q_m$ for all tasks m.
- Calculate $d = (Q_r - Q_s)/\alpha$
- Increase $b_s(l)$ by $\lfloor d \rfloor$, for all $l > l'$ where l' is the current implementation level for task s.

The intuition behind this scheme is that we want to give task s, the worst performing task more computational resources so that it may possibly perform better. By increasing the benefit levels of its higher implementation levels, we will select the higher implementation level if we can penalize other tasks without losing more than $\lfloor d \rfloor$ total estimated benefit. We floor the quality difference d in order to keep the knapsack problem integer.

4. NUMERICAL RESULTS

We have simulated a system with 20 different filters, requiring from six to twelve non-trivial (not equal to one) taps selected offline at random. At their highest (exact) implementation level, the filters take between 27 and 59 shift-add combinations per output sample, and the whole system requires 889 shift-add combinations to run each task at the highest implementation level. Figure 2 shows results from two experiments, one where our system has a constraint of 450 shift-add combinations, and one where our system has a constraint of 500 shift-add combinations. In both experiments, we generate $N = 40000$ samples per time period from each of the AR processes based upon filtering streams of white noise with sample variance of one using the inverse filters $H_{im}(e^{j\omega})$. The normalization factor α used in calculating the Q_m is set to 2000. All of our arithmetic is done in a fixed-point format with ten bits of precision beyond the decimal point.

We have plotted both the mean and standard deviation of the Q_m versus the time period number. In the experiment with 450 shift-add combinations, the standard deviation of the Q_m averages around 10^3 after an initial period of 120

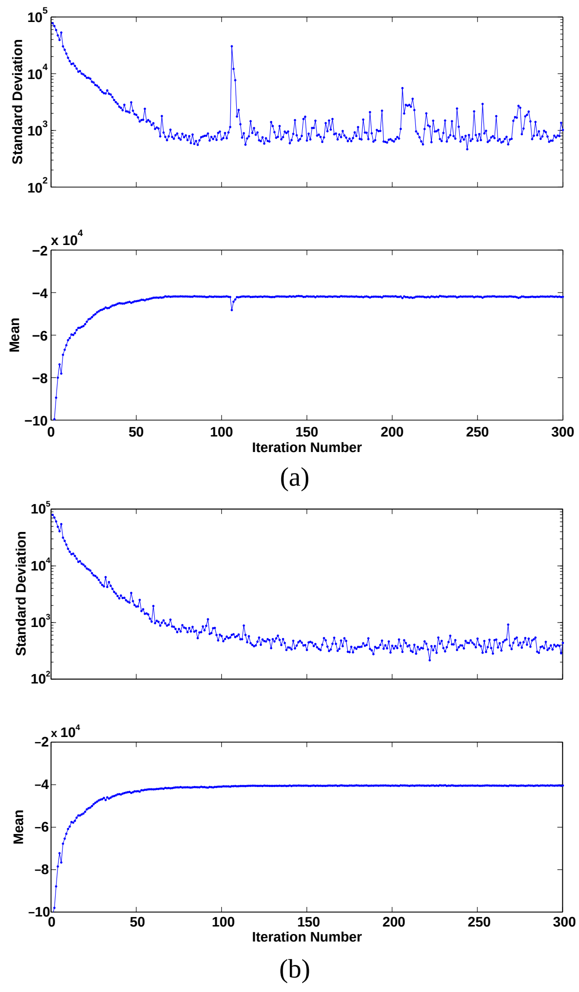
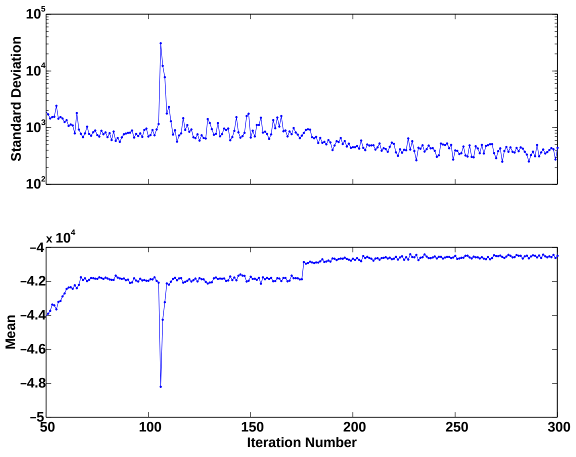


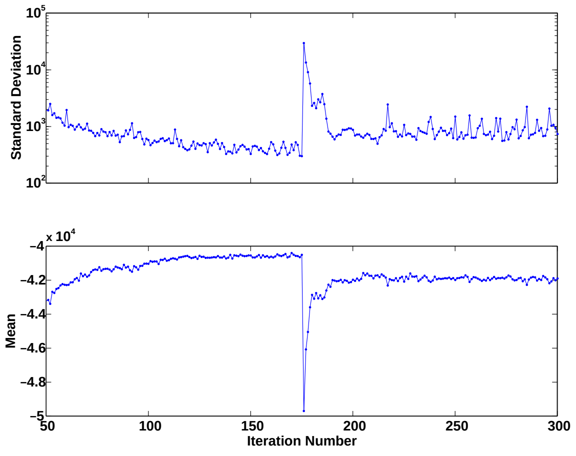
Fig. 2. Mean and standard deviation of Q_m in simulations of 20 filters with computational constraint of (a) 450 and (b) 500 shift-add combinations

iterations. In the experiment with 500 shift-add combinations, the standard deviation of the Q_m is much smaller, averaging around 400 after an initial period of 120 iterations. This seems to imply that the more computational resources there are, the better our heuristic works in driving the Q_m to be equal, although the standard deviation averages an order of magnitude less than the mean in both experiments.

To demonstrate how well the system has learned the resource-quality tradeoffs, we perform more simulations in which the overall computational constraint is changed at iteration 175. The results are shown in Figure 3. We first perform a simulation where we increase the constraint from 450 to 500 shift-add combinations, resulting in an increase in total quality; the mean of the Q_m increases, while the standard deviation settles. We then perform another simulation where we decrease the constraint from 500 to 450 shift-add combinations, resulting in a decrease in the mean



(a)



(b)

Fig. 3. Mean and standard deviation of Q_m in simulations of 20 filters with change of constraint at iteration 175 from (a) 450 to 500 and (b) 500 to 450 shift-add combinations

of the Q_m. The results show that the system is quick to adapt. In the case where the constraint is increased, the system settles immediately into a new average quality level. In the case where the constraint is decreased, the system takes about ten iterations to settle into a new average quality level.

5. FUTURE DIRECTIONS

Our eventual plan is to apply our framework to more complicated problems, for instance, ones where the nature of the input signals changes from throughout time and the Q_m metric needs to depend on both the input and output observations instead of just output observations as in the above example. One possible example where this might occur is lowpass filtering (as in [1][2]) where the input signal might have more or less signal energy in the stopband, causing different Q_m to be measured for the same implementation level.

We would also like to extend the framework to situations where the tasks are completely different in nature, such as the audio compression versus the video compression for the same media stream. Another one of our eventual goals is to use this framework on tasks which are interconnected, *i.e.* the output of one task is the input of another; an example of this would be the motion estimation and the transform coding in a MC-DCT hybrid video coder, where the prediction error from the motion estimation is the input of the DCT block transform. Both these tasks can be performed at different quality levels, with the net effect of producing different quality video at the decoder for the same encoded bitrate.

6. CONCLUSION

We have presented a framework for running multiple real-time tasks where the tasks themselves admit multiple implementation levels of different quality. In this framework, the implementation levels are changed based upon observation of the input and output data of the tasks. Using this framework, we have constructed and simulated an example system where multiple filters are running simultaneously to demonstrate its ability to adapt to different computational constraints.

Although the example is simple, and the actual computation overhead is small, we hope to extend this work to tasks and applications that are challenging to current architectures. Examples of such tasks would be computational kernels such as segmentation for VOP coding in MPEG-4 and transform coefficient prediction in H.264.

7. REFERENCES

[1] M. Goel and N. R. Shanbhag, “Low-power digital signal processing via dynamic algorithm transformations (DAT),” in *Asilomar Conference on Signals, Systems and Computers*, November 1998.

[2] S. Nawab, A. Oppenheim, A. Chandrakasan, J. Winograd, and J. Ludwig, “Approximate signal processing,” *J. VLSI Signal Processing Systems*, vol. 15, no. 1/2, January 1997.

[3] David Pisinger, *Algorithms for Knapsack Problems*, Ph.D. thesis, University of Copenhagen, 1995.

[4] A. Oppenheim and R. Schafer, *Discrete Time Signal Processing*, Prentice-Hall, 1989.

[5] J. Coleman and A. Yardakul, “Fractions in the canonical-signed-digit number system,” in *Conf. on Information Sciences and Systems*, March 2001.