# ENERGY EFFICIENT IMPLEMENTATION OF LINEAR SYSTEMS ON PROGRAMMABLE PROCESSORS

Mani B. Srivastava
AT&T Bell Laboratories
Murray Hill, NJ

Miodrag Potkonjak
NEC C&C Research Laboratories
Princeton, NJ

**Abstract - This paper describes a behavior transformation based technique to optimize power consumption in linear systems implemented on programmable processors. For the single processor case, our method is based on unfolding by an optimal amount so as to minimize the effective number of operations per sample. For the multiple processor case, the additional increase in throughput due to multiple processors is also traded-off against clock frequency slowdown and voltage reduction. The effectiveness of our techniques and algorithms is demonstrated on numerous real-life examples from DSP, control, and communications.**

## THROUGHPUT AND POWER IN LINEAR SYSTEMS

### Motivation

Linear computations form an important type of computation that is widely used in video and image processing, DSP, control, communications, and many other applications. A large fraction of systems in these application domains are either linear, or have subsystems that are linear over field F, an additive operator $\oplus$, and a multiplicative operator $\otimes$. In recent years two fundamental properties of implementations of linear computations have been identified. The first is that an arbitrary linear computation can be implemented at an arbitrarily high throughput [6]. The second is that there is a fundamental limit on the joint optimization of throughput and latency, the two metrics of speed. Srivastava and Potkonjak [7] showed that it is impossible to reduce latency and increase throughput to an arbitrary level simultaneously.

This paper explores the relationship of throughput with the third, and increasingly important, design metric - power consumption. We seek to find the extent to which power consumption of linear systems can be reduced, both independently and in conjunction with throughput improvement, and to develop techniques for doing so.

To explore the throughput and power relationship in linear systems we take a more thorough and systematic approach than taken previously. First, we consider analytically as well as empirically the effect of several algorithm transformations. Second, we consider implementations not in the form of ASICs with application-specific datapaths as is usually the case, but instead implementations based on single programmable processor and multiple programmable processors. This is important because increasingly programmable processors such as DSP-cores have emerged as the preferred medium of implementation.

0-7803-2612-1/95 $4.00 © 1995 IEEE

## Where does the Power Go, and How to Reduce It?

Prediction of power consumption of a computation at the behavioral level is a difficult task. In CMOS technology there are three sources of power consumption: switching current (dynamic power), short-circuit current, and leakage currents. The switching component not only dominates in most designs. The average power consumption of a CMOS gate due to the switching component is given by:

$$P = \alpha C_L V_{dd}^2 f \qquad \text{(EQ 1)}$$

where $f$ is the system clock frequency, $V_{dd}$ is the supply voltage, $C_L$ is the load capacitance, and $\alpha$ is the switching activity (the probability of a 0→1 transition during a clock cycle). The term $\alpha C_L$ is often lumped into a single parameter called effective switched capacitance. Further, it is well known that the delay through a gate has a monotonically inverse relationship to the supply voltage. Fig. 1 plots the gate delay as a function of voltage normalized to gate delay at 5.0 Volts. Therefore, the maximum rate at which a circuit can be clocked will monotonically decrease as the voltage is reduced.

The above expression suggests several behavior level approaches to reduce power consumption of a computation. First is to shut the system down during periods of inactivity either by shutting off the clock ( $f = 0$ ) or by shutting off the power supply ( $V_{dd} = 0$ ). Second is to reduce the effective switched capacitance $\alpha C_L$ by restructuring the computation, communication, and/or the memory hierarchy, and by changing data encoding and a data formats. The third is to exploit the quadratic dependence of power consumption upon the supply voltage $V_{dd}$ and operate at a reduced voltage while compensating for the resulting loss in circuit speed by techniques that increase the throughput.

While the first technique is not directly relevant to us because we are dealing with computation that is inherently "continuous" in nature as opposed to being event driven, the latter two are of direct relevance. Since that throughput is the sole metric of speed that is important to us, one can combine the latter two approaches to minimize power at the behavioral level in the following fashion: First, apply a behavior transformation to reduce the effective switched capacitance (by reducing the number of operations) or to increase the throughput (by reducing the critical path). Next, in the case of increased throughput, lower the supply voltage just the right amount so as to decrease the clock speed to an extent that the throughput reverts back to what it was before. The net power consumption is reduced if either the effective switched capacitance is reduced at a constant voltage, or if the reduction due to reduced

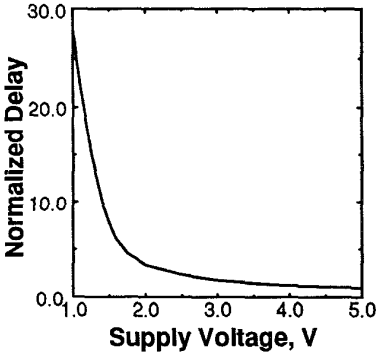


**Figure 1: Normalized Gate Delay (delay @ 5V = 1 unit) vs. Supply Voltage**

voltage and frequency overshadows any increased effective capacitance penalty paid for increase in throughput. Finally, note that at some point further voltage reduction will not be technically feasible because the supply voltage must be greater than threshold voltage in CMOS. When such a point is reached, one can trade-off extra throughput with lower clock frequency or with shutdown, both of which will result in linear reduction.

## Linear Systems

In high-level synthesis terminology, a system is linear if it can be realized by a control-dataflow graph (CDFG) such that at any time instant $n$, the output samples and the next state values are computed by linear combinations of input samples and previous state values. For analysis in this paper we shall use an equivalent representation of the linear computation as a set of discrete-time finite-dimensional state-space equations. Specifically, a $P$ input, $Q$ output, and $R$ state real-valued linear computation can be expressed by the following matrix equations:

$$\begin{aligned} S[n] &= AS[n-1] + BX[n] \\ Y[n] &= CS[n-1] + DX[n] \end{aligned} \qquad \textbf{(EQ 2)}$$

$$n \in \{0, 1, 2, 3, \ldots\} \quad X[n] \in \Re^{P \times 1} \quad S[n] \in \Re^{R \times 1} \quad Y[n] \in \Re^{Q \times 1}$$

$$A \in \Re^{R \times R} \quad B \in \Re^{R \times P} \quad C \in \Re^{Q \times R} \quad D \in \Re^{Q \times P} \quad S[-1] \in \Re^{Rx1}$$

where $X$, $Y$, and $S$ are the input, output, and state vectors respectively, and $A$, $B$, $C$, and $D$ are constant coefficient matrices. Note that the throughput of the system, or the maximum rate at which it can process incoming samples, is decided solely by the critical path of the feedback section corresponding to the term $S[n] = AS[n-1]$. The remaining terms are not in the feedback loop and therefore can be pipelined away.

This is the base or reference case for our analysis, and has following characteristics:

$$\text{\# of muls} = (R+P)(R+Q)$$

$$\text{\# of adds} = (R+P-1)(R+Q)$$

$$\text{feedback critical path} = m + \lceil \log_2(1+R) \rceil$$

and,

$$\text{max throughput} = \frac{1}{\text{feedback critical path}} = \frac{1}{m + \lceil \log_2(1+R) \rceil}$$

For the critical path we assumed that the time for an addition is 1, and the time for a multiplication is $m \geq 1$ which is a reasonable assumption in almost all cases, whether on an ASIC or on a programmable processor. The expressions are obtained by considering the worst case where the coefficient matrices $A$, $B$, $C$, and $D$ are dense matrices with arbitrary non-trivial coefficients (not 0, or 1, or -1). The maximum throughput expression is obtained by organizing the required linear combinations in a maximally fast fashion by first doing the constant-variable multiplications in parallel, and then the additions in a maximally balanced binary tree [6].

## Paper Organization

The remainder of the paper is organized in the following way. In the next subsection the related work is surveyed. The next section introduces the basic idea behind using unfolding to reduce the number of operations per sample This idea is

applied to power reduction for single processor and multiple processor implementations in the third and fourth sections respectively.

### Related Work

The related work can be traced along two lines of research: transformations and low power techniques. Transformations are widely used technique to improve variety of design parameters during high level synthesis [Pot92].

While power optimization was rarely addressed in the past, it has recently attracted much attention with the rapidly growing market of portable products. Interestingly, although it is apparent that the highest potential for power reduction is at higher levels of design process, few efforts for power optimization at the behavioral level has been reported. Chandrakasan et al. [1] demonstrated the effectiveness of transformations for power reduction. Duncan et al. used retiming to reduce power dissipation in DSP designs [4]. Chatterjee and Roy [3] targeted the power reduction in fully hardwired design by minimization switching activity. Wuytack et. al. used trade-offs related to background memory design to achieve power reduction [9].

Chandrakasan et. al [2] and Tiwari et al [8] did pioneering work in power minimization and optimization when programmable platforms are targeted.

## UNFOLDING-DRIVEN POWER REDUCTION VIA MINIMIZATION OF NUMBER OF OPERATIONS

As mentioned earlier, reduction in number of operations (and thereby in effective switched capacitance) and the trading-off of reduced voltage with increased throughput obtained via behavior-level techniques are two basic approaches to behavior level power reduction. Unfolding, factoring, and common sub-expression elimination are all behavior level transformations that can increase throughput or reduce number of operations in linear computations, and are therefore crucial to power reduction of linear systems.

Unfolding together with block processing or with look-ahead has been shown by various researchers [5] to be effective in obtaining arbitrary improvement in throughput. Essentially, several consecutive input samples are processed together as a batch to produce a batch of output samples together with the input state value for the next batch. In the original non-unfolded computation $X[n]$ and $S[n-1]$ are used to compute $Y[n]$ and $S[n]$. In a system that has been unfolded $i$ times, a batch of $i+1$ input samples $X[n] \ldots X[n+i]$ together with $S[n-1]$ is used to compute a batch of output samples $Y[n] \ldots Y[n+i]$ and the next state $S[n+i]$. The basic computation executed by a linear system that has been unfolded $i$ times can be represented by the following state-space equations:

$$\begin{aligned} S[n+i] &= A^{i+1}S[n-1] + A^{i}BX[n] + A^{i-1}BX[n+1] + \ldots + BX[n+i] \\ Y[n] &= CS[n-1] + DX[n] \\ Y[n+1] &= CAS[n-1] + CBX[n] + DX[n+1] \\ &\ldots \\ Y[n+i] &= CA^{i}S[n-1] + CA^{i-1}BX[n] + CA^{i-2}BX[n+1] \ldots + CBX[n+i-1] + DX[n+i] \end{aligned} \quad \text{(EQ 3)}$$

Note that these equations process $i+1$ data samples for each execution. For the $i$

times unfolded system we get the following characteristics:

$$\#(*, i) = \#\text{ of multiplications } = R^2 + (i+1)PR + (i+1)QR + \frac{(i+1)(i+2)}{2}PQ$$

$$\#(+, i) = \#\text{ of additionss } = R^2 + (i+1)PR + (i+1)QR + \frac{(i+1)(i+2)}{2}PQ - R - (i+1)Q$$

$$\text{CP}(i) = \text{feedback critiical path for } i \text{ times unfolded system } = m + \lceil \log_2(1+R) \rceil$$

$$\text{MaxThroughput}(i) = \text{max throughput for } i \text{ times unfolded system } = \frac{(i+1)}{m + \lceil \log_2(1+R) \rceil}$$

As expected, when $i = 0$ the above equations reduce to those for the non-unfolded case presented earlier. Further, the maximum achievable throughput is arbitrarily increased as the amount of unfolding increases because the feedback critical path remains the same while more samples are processed.

An interesting observation is that the effective number of operations per input sample is lower in the unfolded case when the amount of unfolding $i$ is less than a certain threshold. In particular,

$$\frac{\#(*, i)}{i+1} - \#(*, 0) = \#\text{ of muls per sample with unfolding } - \#\text{ of muls per sample without unfolding}$$

$$= -\frac{i}{i+1}R^2 + \frac{i}{2}PQ \qquad \textbf{(EQ 4)}$$

$$< 0 \qquad \text{for } i < \left(\frac{2R^2}{PQ} - 1\right)$$

and,

$$\frac{\#(+, i)}{i+1} - \#(*, 0) = \#\text{ of adds per sample with unfolding } - \#\text{ of adds per sample without unfolding}$$

$$= -\frac{i}{i+1}R(R-1) + \frac{i}{2}PQ \qquad \textbf{(EQ 5)}$$

$$< 0 \qquad \text{for } i < \left(\frac{2R(R-1)}{PQ} - 1\right)$$

It should also be noted that above expressions for differences in numbers of * and + per sample achieve minimum at certain $i$ below the shown thresholds - in other words, as one unfolds, the number of operations per sample at first decreases to reach a minimum and then begins to rise.

The above lead to the following strategies for low power implementations on single and multiple processors.

## POWER REDUCTION FOR SINGLE PROCESSOR

In the case of a single programmable processor the throughput that is achieved is solely decided by the number of operations. It follows that the throughput is maximized by using the value of $i$ that minimizes the total number of instructions. If one assumes that + and * are the basic processor instructions (they need not take the same number of cycles), it can be shown that the optimum value of unfolding $i_{opt}$ is one of the following two values:

$$i_{opt} = \left\lfloor \sqrt{\frac{(2R-1)R}{PQ}} - 1 \right\rfloor \quad \text{or} \quad \left\lceil \sqrt{\frac{(2R-1)R}{PQ}} - 1 \right\rceil$$

depending on whichever leads to a smaller value of $i_{opt}\left(PQ - \frac{R(2R-1)}{i_{opt}+1}\right)$. If both lead to same value of $i_{opt}\left(PQ - \frac{R(2R-1)}{i_{opt}+1}\right)$, we pick the smaller $i_{opt}$ so as to save on coefficient memory because larger unfolding leads to more coefficients.

We can now obtain the following expression for maximum improvement in throughput:

$$S_{\max} = \text{max improvement in single processor throughput by unfolding}$$
$$= (i_{opt}+1)\frac{\#(*,0)+\#(+,0)}{\#(*,i_{opt})+\#(+,i_{opt})}$$

Finally, the processor voltage can be reduced so that the clock frequency $f$ is reduced by a factor of $S_{\max}$. This leads to a reduction in processor power because in the expression for power $P = \alpha C_L V_{dd}^2 f$ the terms $V_{dd}^2$ and $f$ are reduced whereas the other two terms remain constant. It must be mentioned that we are implicitly assuming that the processor power consumption is dominant compared to coefficient and data memory power consumption, an assumption that is true in most CPU-memory systems as found in DSP and control processing systems.

As an example, consider a hypothetical linear computation with $P = 1$ input, $Q = 1$ output, $R = 12$ states. Then, from the approach above one can show that $i_{opt} = 16$ which leads to $S_{\max} = 4.075$. One can therefore reduce the voltage such that the clock frequency is reduced by a factor of 4.00 so that the throughput reverts back to the original throughput. If the initial voltage is 3.0V, then from Fig. 1 it follows that the voltage reduction to 1.5V will result in the desired clock slowdown. The processor operating at 1.5V and computing the equations that have been unfolded 16 times will has same throughput as the processor operating at 3.0V and computing the initial non-unfolded equations. However, in the unfolded case one obtains a power reduction of $\left(\frac{3.0}{1.5}\right)^2 \times \frac{1}{(1/4)}$, or a factor of 16 over the initial power consumption. If the initial voltage was 5.0V, then the improvement is by a factor 27.7.

The above result is based on analysis that assumed that the coefficient matrices $A$, $B$, $C$, and $D$ are dense matrices with arbitrary non-trivial coefficients (not 0, or 1, or -1). While this is certainly true of linear systems that are found in process controllers, it is often not the case with filters found in DSP applications where these matrices often tend to be sparse and have coefficients that are trivial. Unfortunately, it is not possible to come with analytical expressions for the non-dense case.

However, we have empirically found that unfolding helps in reducing the number of operations and the power even in such cases - although by smaller factors. The only problem is that the optimum level of unfolding and the number of operations can no longer be found by merely evaluating closed-form formulas. We therefore use the following heuristic to find the desired level of unfolding in the non-dense

cases: first pick the best performing level of unfolding from amongst all values of $i$ from 0 through $i_{opt}$, the optimum value analytically predicted for the dense case. If the best level turns out to be $i_{opt}$, then we continue to unfold further as long as the number of operations continues to decline.

In case unfolding results in such a large increase in throughput (reduction in number of operations) that even after reducing the voltage to the minimum feasible (about 1V in the technology that we used) the new system has higher throughput than the original, then one can obtain a further reduction in power by operating the processor at an even lower frequency (or, equivalently, by shutting the processor for part of the time). This, however, did not happen for any of our examples.

In Table 1 we give the empirically obtained numbers for the actual coefficients from several real-life examples, mainly linear controllers and filters, as well as analytically predicted numbers for dense coefficient matrices with same values of $P$, $Q$, and $R$ as in the real-life examples. The average processor power reduction was x2.2 for 3.0V initial, and x2.7 for 5.0V initial. It must be emphasized that this power reduction is being obtained with no processor area penalty. While the coefficient memory requirements do increase, memories come in sizes that are powers of 2 - therefore in most cases there will be no impact on memory size either.

| | | | For Dense Coefficient Matrices | | | | | | | Using Coefficients from Real Examples | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Initial | | After Optimization | | | | | | Initial | | After Optimization | | | | |
| P | Q | R | # Op | V | i | # Ops | V | Frq Red | Pwr Red | Design | # Op | V | i | # Ops | V | Frq Red | Pwr Red |
| 1 | 1 | 3 | 28 | 3<br>5 | 3 | 19.75 | 2.4<br>3.6 | 1.42 | 2.2<br>2.7 | mat | 28 | 3<br>5 | 3 | 19.75 | 2.4<br>3.6 | 1.42 | 2.2<br>2.7 |
| 1 | 1 | 5 | 66 | 3<br>5 | 6 | 33.43 | 2.0<br>2.8 | 1.97 | 4.4<br>6.3 | lin5 | 66 | 3<br>5 | 6 | 33.43 | 2.0<br>2.8 | 1.97 | 4.4<br>6.3 |
| 1 | 1 | 5 | 66 | 3<br>5 | 6 | 33.43 | 2.0<br>2.8 | 1.97 | 4.4<br>6.3 | wdf5 | 32 | 3<br>5 | 3 | 27.25 | 2.7<br>4.2 | 1.17 | 1.4<br>1.7 |
| 1 | 1 | 8 | 153 | 3<br>5 | 10 | 53.91 | 1.7<br>2.2 | 2.84 | 8.8<br>14.7 | iir8 | 34 | 3<br>5 | 0 | 34 | 3.0<br>5.0 | 1 | 1<br>1 |
| 1 | 1 | 10 | 231 | 3<br>5 | 13 | 67.57 | 1.6<br>2.0 | 3.42 | 12.0<br>21.4 | iir10 | 85 | 3<br>5 | 9 | 59.8 | 2.4<br>3.6 | 1.42 | 2.2<br>2.7 |
| 1 | 1 | 12 | 325 | 3<br>5 | 16 | 81.23 | 1.5<br>1.9 | 4.00 | 16.0<br>27.7 | iir12 | 114 | 3.<br>5 | 11 | 71.83 | 2.2<br>3.3 | 1.59 | 3.0<br>3.7 |
| 2 | 1 | 5 | 78 | 3<br>5 | 4 | 50 | 2.3<br>3.3 | 1.56 | 2.7<br>3.6 | dist | 48 | 3<br>5 | 3 | 47.25 | 3.0<br>5.0 | 1.02 | 1<br>1 |

**Table 1: Power Reduction in a Single Programmable Processor using Unfolding-Driven Voltage-Throughput Trade-off**

Interestingly, note that even if voltage reduction is not an option due to hardware and technology constraints, the increased throughput can be traded-off against reduced clock frequency for a linear reduction in power consumption. Instead of clock frequency reduction, one may also use shutdown by stopping the clock or making supply voltage zero for part of the sample period. This strategy gives an average power reduction of x1.37 (27%) over all our examples.

## IMPLEMENTATION ON MULTIPLE PROCESSORS

Potentially more savings can be obtained using our approach if one considers implementations that are not restricted to a single processor. By using more than one processors the throughput achieved by the implementation can be reduced compared to the single processor case, and by using enough processors the maximum possible throughput (decided by the critical path through the feedback portion of the linear computation) can be achieved. The extra throughput thus obtained can be used for further throughput-voltage trade-off as long as the power reduction from this compensates for the power increase due to more processors.

As an example, consider the same hypothetical linear computation with $P = 1$ input, $Q = 1$ output, $R = 12$ states, and dense coefficient matrices that we considered for the single processor case. Previously we had shown that the number of operations per sample is minimized when the linear computation is unfolded for $i_{opt} = 16$ times, and that the maximum throughput achieved by a single processor relative to original non-unfolded implementation is $S_{\max}(1) = (i_{opt}+1)\frac{\#(*,0)+\#(+,0)}{\#(*,i_{opt})+\#(+,i_{opt})} = 4$. Now, if a second processor is added, the throughput will increase by x2 (ignoring communication costs), and at the same time power consumption will increase by x2 due to the addition of the second processor. Now, one can reduce the voltage such that the clock frequency of both the processors is reduced by $S_{\max}(1) = 2x4 = 8$. If the initial voltage was 3.0V, then this reduced voltage (from Fig. 1) is given by 1.27V. Therefore, the 16-unfolded two-processor 1.27V implementation will have a power reduction of $\left(\frac{3.0}{1.27}\right)^2 \times \frac{1}{1/8} \times \frac{1}{2} = 22.3$ relative to an non-unfolded 3.0V single-processor implementation. This improvement of about x22 is greater than the x16 obtained if one were to restrict to a single processor only.

In general, the situation is more complex when adding processors. First, addition of processors causes at least a linear increase in switched capacitance, and hence power, for a given voltage and clock frequency. The increase in switched capacitance may be super-linear due to inter-processor communication hardware. Second, the speed-up is not linear, and begins to saturate due to inter-processor communication overhead. Even if the inter-processor communication cost is ignored, the computation cannot be speeded up more than that allowed by the critical path of the feedback section. Finally, the voltage cannot be reduced below a certain point.

The following analytic approach explores the unfolding-driven power-throughput trade-off in implementations using multiple processors. The simplifying assumptions are (i) inter-processor communication does not cost any time, (ii) effective switched capacitance $\alpha C_L$ increases linearly with the number of processor $N$, (iii) voltage cannot be reduced below 1V, and (iv) both addition and multiplication instructions take one clock cycle (i.e. $m = 1$ ).

The first step is to unfold the linear computation to the optimum level $i = i_{opt}$ where the number of operations (instructions) per sample is minimized. The second step is to increase the number of processors to $N$. Let $S_{\max}(N, i)$ be the maximum improvement in throughput achieved by $N$ processors on an $i$ times unfolded linear computation compared to a single processor on the original non-unfolded computation. The third step is then to slow-down each of the $N$ processor by a factor of $S_{\max}(N, i)$ - this is done by reducing the voltage just the right amount so as to decrease the clock frequency (increase the gate delay) by $S_{\max}(N, i)$. Let $V(d)$ be the voltage at which the value of gate delay relative to the initial implementation is $d$, with $V(1)$ typically being 3.0V or 5.0V. Then, the power of the new $N$ processor implementation relative to the original non-unfolded implementation is:

$$Power(N) = \left(\frac{V(1)}{V(S_{\max}(N, i_{opt}))}\right)^2 \times \frac{S_{\max}(N, i_{opt})}{N}$$

The task is to find the optimum value $N = N_{opt}$ where the above expression is minimized. The crucial missing piece of the puzzle is an estimate of $S_{\max}(N, i)$, the maximum improvement in throughput achieved by $N$ processors on an $i$ times unfolded linear computation. It can be shown via some intricate algebraic manipulation that under our simplifying assumptions,

$$S_{\max}(N, i) = \frac{(i+1)\ (\#(*, 0) + \#(+, 0))}{\left(\frac{\#(*, i) + \#(+, i)}{N}\right)} = NS_{\max}(1, i) \qquad \text{for } (N \le R)$$

$$S_{\max}(N, i) = \frac{(i+1)\ (\#(*, 0) + \#(+, 0))}{\max\left(\left\lceil \log_2\left(1 + \frac{N}{R}\right)\right\rceil + \left\lceil \frac{R + 1 - 2^{\lceil \log_2(1 + N/R)\rceil}}{N/R} \right\rceil, \frac{\#(*, i) + \#(+, i)}{N}\right)}$$
$$\text{for } (R < N < R\langle R + 1\rangle)$$

and,

$$S_{\max}(N, i) = \frac{(i+1)\ (\#(*, 0) + \#(+, 0))}{\max\left((1 + \lceil \log_2(1 + R)\rceil), \frac{\#(*, i) + \#(+, i)}{N}\right)} \qquad \text{for } (N \ge R\,(R+1))$$

The speed-up due to multiple processors is linear for $N \le R$, which will allow a linear decrease in frequency, and therefore power, and thus offset the linear increase in power due to increase in number of processors. Therefore, one can always add up to $R$ processors and get a reduction in power due to the reduction in the voltage term. In other words, the optimum number of processors is at least $R$.

While the above expressions are true only for dense coefficient matrices, the observation that the speed-up is linear for $N \le R$ is valid even for real-life non-dense coefficient matrices in a slightly modified form: the speed-up will be *at-least* linear (under our assumption of zero communication cost). We exploit this fact, and conservatively use $N = R$ processors to get at least a linear increase in throughput (on top of what $i_{opt}$ level unfolding alone gives) and trade this increased throughput with a voltage reduction to slow down the clock by an equivalent amount. Table 2 shows the resulting power reduction for our suite of examples.

| Design | P | Q | R | Init. V | Unfolding + Single Processor (Table 1) | | | Unfolding + Multiple Processors | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | V | Frq Red | PwrRed | # of Procs | V | Frq Red | PwrRed |
| mat | 1 | 1 | 3 | 3.0<br>5.0 | 2.4<br>3.6 | 1.42 | 2.2<br>2.7 | 3 | 1.5<br>1.8 | 4.26 | 5.7<br>10.9 |
| lin5 | 1 | 1 | 5 | 3.0<br>5.0 | 2.0<br>2.8 | 1.97 | 4.4<br>6.3 | 5 | 1.2<br>1.4 | 9.85 | 12.3<br>25.1 |
| iir5/wdf5 | 1 | 1 | 5 | 3.0<br>5.0 | 2.7<br>4.2 | 1.17 | 1.4<br>1.7 | 5 | 1.4<br>1.6 | 5.85 | 5.4<br>11.4 |
| iir8 | 1 | 1 | 8 | 3.0<br>5.0 | 3.0<br>5.0 | 1 | 1<br>1 | 8 | 1.3<br>1.5 | 8 | 5.3<br>11.1 |
| iir10 | 1 | 1 | 10 | 3.0<br>5.0 | 2.4<br>3.6 | 1.42 | 2.2<br>2.7 | 10 | 1.1<br>1.3 | 14.2 | 10.6<br>21.0 |
| iir12 | 1 | 1 | 12 | 3.0<br>5.0 | 2.2<br>3.3 | 1.59 | 3.0<br>3.7 | 9<br>12 | 1.1<br>1.2 | 14.3<br>19.1 | 11.8<br>27.6 |
| dist | 2 | 1 | 5 | 3.0<br>5.0 | 3.0<br>5.0 | 1.02 | 1<br>1 | 5 | 1.4<br>1.7 | 5.1 | 4.4<br>8.8 |

**Table 2:** Power Reduction with Unfolding and Multiple Processors

# CONCLUSION

We introduced a new approach for power minimization in linear computations using transformations. The method relies on unfolding the linear computation to an optimal level so as to minimize the effective number of addition and multiplication operations that are needed to process each sample. The increase in throughput gained via this reduction in number of operations is then traded-off against clock frequency reduction and supply voltage reduction, thus leading to power reduction. The resulting power reduction is substantial both for the single processor case and for the multiple processor case.

# REFERENCES


[1] A.P. Chandrakasan, et al. "Hyper-LP: A Design System for Power Minimization using Architectural Transformations", *ICCAD-92*, pp. 300-303, 1992.

[2]] A.P. Chandrakasan, M.B. Srivastava, R.W. Brodersen, "Energy Efficient Programmable Computation", *VLSI Design Conference*, pp. 261-264, 1994.

[3] A. Chatterjee, R.K. Roy, "Synthesis of Low Power DSP Circuits Using Activity Metrics", Proc. of *VLSI Design Conference*, pp. 265-270, 1994.

[4] P. Duncan, S. Swamy, R. Jain, "Low-Power DSP Circuit Design Using Retimed Maximally Parallel Architectures", *Symposium on Integrated Systems*, pp. 266-275, 1993.

[5] K. K. Parhi, D. Messerschmitt, "Pipeline interleaving and parallelism in recursive Filters, Parts 1", *IEEE Trans. on ASSP*, Vol. 37, pp. 1099-1117, 1989.

[6] M. Potkonjak, J. Rabaey, "Maximally Fast and Arbitrarily Fast Implementation of Linear Computations", *ICCAD-92*, pp. 304-308. 1992.

[7] M. B. Srivastava, M. Potkonjak: "Transforming Linear Systems for Joint Latency and Throughput Optimization", *EDAC-94 European Design Automation Conference*, pp. 267-271, 1994.

[8] V. Tiwari, S. Malik, A. Wolfe, "Power Analysis of Embedded Software: A first Step towards Software Power minimization, *ICCAD-94*, pp. 384-390, 1994.

[9] S. Wuytack, et al., "Global communication and memory optimizing transformations for low power systems", *Intnl. Workshop on Power Design*, 1994.