LOW POWER FIR FILTER REALIZATION WITH DIFFERENTIAL COEFFICIENTS AND INPUT

Tian-Sheuan Chang and Chein-Wei Jen

Dept. of Electronics Engineering, National Chiao-Tung University

1001 Ta-Hsueh Rd, Hsinchu, Taiwan, R.O.C.

ABSTRACT

Most FIR filter realizations use the inputs and coefficients directly to compute the convolution. In this paper, we present a low power and high speed FIR filter designs by using first order difference between inputs and various orders of differences between coefficients. This design first reformulates the FIR operations with the differences in algorithm level. Then, in architecture level, we adopt the DA architecture to exploit the probability distribution such that power consumption can be reduced further. The design is applied to an example FIR filter to quantify the energy savings and speedup. It shows lower power consumption than the previous design with the comparable performance.

I. INTRODUCTION

Recently, due to the popularity of the portable battery-powerd wireless communication systems such as cellular phones, pagers and wireless modems, high performance and low power digital signal processing (DSP) has become increasingly important. One of the commonly used operations in DSP is FIR filters. A N-tap FIR filter with coefficients C_k, input sequence X_j, and output sequence Y_j can be expressed as

$$Y_j = \sum_{k=0}^{N-1} C_k X_{j-k}$$

Conventional realizations of FIR filters use input and coefficients directly, which requires full wordlength of the multiplication and accumulation and thus consumes more power. Realization using differential coefficients (called differential coefficients method (DCM)) has been proposed in [1] to solve above problems. However, their formulation reconstructs original C_kX_{j-k} term by term, which requires many memory accesses. Besides, their formulation only considers coefficients difference. They did not consider the architecture level design to maximize the power saving.

To solve above problems, in this paper, we propose a new algorithm (called differential coefficients/input method (DCIM)) that not only consider differential coefficients but also consider differential inputs. After the algorithm reformulation, we propose an architecture design by using distributed arithmetic (DA) that can group high transition probability LSB input bits at one time so we can skip low transition probability MSB bits more efficiently and save power consumption.

This paper is organized as follows. In Section 2, we first review the DCM algorithm and then we propose our new algorithm formulation. Then in Section 3 we present the DA architecture design based on the DCIM algorithm. In Section 4, we will analyze the power consumption and delay of the proposed design and apply them to an example filter. Finally, we conclude this paper in Section 5.

II. ALGORITHM FORMULATION

The differential coefficients method (DCM) computes the partial product with *m*th-order differences first, and then added the stored previous partial product back. If the *m*th- order differences are defined as

$$\delta^m_{k-m/k} = \delta^{m-1}_{k-m+1/k} - \delta^{m-1}_{k-m/k-1}$$, k=m to (N-1): m=2 to (N-1).

Then the recurrence relation between coefficients using *m*th-order differences is

$$C_k = C_{k-1} + \sum_{i=1}^{m-1} \delta^i_{k-i-1/k-1} + \delta^m_{k-m/k}$$, k= m to (N-1): m=2 to (N-1)

Then, for any two consecutive outputs Y_j and Y_{j+1}, we can rewrite the outputs with first order difference DCM algorithm and obtain

$Y_j=C_0X_j+C_1X_{j-1}+\ldots+C_{N-1}X_{j-N+1}$

$Y_{j+1}=C_0X_{j+1}+C_1X_j+\ldots+C_{N-1}X_{j-N+2}$

$= C_0X_{j+1}+\{(C_1X_j- C_0X_j)+C_0X_j\}+\ldots+\{(C_{N-1}X_{j-N+2}-C_{N-2}X_{j-N+2})+C_{N-2}X_{j-N+2}\}$

$=C_0X_{j+1}+\{(C_1- C_0)\ X_j + C_0X_j\}+\ldots+\{(C_{N-1}-C_{N-2})X_{j-N+2}+C_{N-2}X_{j-N+2}\}$

The DCM algorithm computes $\{(C_1- C_0)\ X_j + C_0X_j\}$, … , $\{(C_{N-1}- C_{N-1})\ X_{j-N+2}+ C_{N-1}X_{j-N+1}\}$, term by term. During the computation of each term, DCM first computes the partial products $(C_k- C_{k-1})\ X_{j-(k-1)}$ and then adds $C_{k-1}X_{j-(k-1)}$ terms back. Each $C_{k-1}X_{j-(k-1)}$ term that has also occurred in computing Y_j was stored in a memory and retrieved when necessary. So only the differential coefficient is used to do the multiplication, and other terms such as $C_{k-1}X_{j-(k-1)}$ are not computed again and just are just added back. By this way, the small quantity because of the difference $(C_k- C_{k-1})$ can save the power consumption to compute the partial products $(C_k- C_{k-1})\ X_{j-(k-1)}$ because we can trade long multiplier with a short one and overheads. However, such computation order is not efficient enough since the operation to add compensated terms $C_{k-1}X_{j-(k-1)}$ back has to be performed for each term computation, which wastes N unnecessary memory access and addition and then consumes memory area and power to store and retrieve them. These compensated terms can be summed together, stored and retrieved once per Y. Besides, the input data X still uses full word length.

Differential input can be introduced to reduce the word length, and thus save power. If the range of the difference between two successive inputs is W_{dx} bits smaller than original input, we may use even shorter multipliers. Such case may occur in speech

systems such as wireless phone system. In such system design, the filter input is often obtained from an analog-to-digital converter (ADC). The analog input (speech signal) is continuous and has only small sharp amplitude change. So the filter input data will be quite close to the neighbor input and their difference will be a small value, which is quite suitable for such differential input design.

The proposed method, which we term the differential coefficient/input method (DCIM) can be formulated as follows. For any consecutive outputs Y_{j-1} , Y_j and Y_{j+1}, we obtain

$Y_{j-1}=C_0X_{j-1}+C_1X_{j-2}+\ldots+C_{N-1}X_{j-N}$

$Y_j=C_0X_j+C_1X_{j-1}+\ldots+C_{N-1}X_{j-N+1}$

$Y_{j+1}=C_0X_{j+1}+C_1X_j+\ldots+C_{N-1}X_{j-N+2}$

Let

$Y_j^1 = Y_j - Y_{j-1} = C_0(X_j - X_{j-1})+C_1(X_{j-1} - X_{j-2})+\ldots+C_{N-1}(X_{j-N+1} - X_{j-N})$

$Y_{j+1}^1 = Y_{j+1} - Y_j = C_0(X_{j+1} - X_j)+C_1(X_j - X_{j-1})+\ldots+C_{N-1}(X_{j-N+2} - X_{j-N+1})$

and define the sum of the first (N-1) partial products of Y_j as

$Y_{j,p} = C_0(X_j - X_{j-1})+C_1(X_{j-1} - X_{j-2})+\ldots +C_{N-2}(X_{j-N+2} - X_{j-N+1})$

Then, reformulate Y_j^1 and Y_{j+1}^1 with DCM, we can express Y_{j+1}^1 as

$$\begin{aligned} Y_{j+1}^1 &= C_0(X_{j+1} - X_j)+ \{(C_1 - C_0)(X_j - X_{j-1}) + C_0(X_j - X_{j-1})\}+\ldots \\ &\quad +\{(C_{N-1} - C_{N-2})(X_{j-N+2} - X_{j-N+1})+ C_{N-2}(X_{j-N+2} - X_{j-N+1})\} \\ &= C_0(X_{j+1} - X_j)+ (C_1 - C_0)(X_j - X_{j-1})+\ldots.+ (C_{N-1} - C_{N-2}) \\ &\quad (X_{j-N+2} - X_{j-N+1})+ \\ &\quad \{C_0(X_j - X_{j-1})+C_1(X_{j-1} - X_{j-2})+\ldots+C_{N-2}(X_{j-N+2} - X_{j-N+1})\} \\ &= C_0(X_{j+1} - X_j)+ (C_1 - C_0)(X_j - X_{j-1})+\ldots.+ (C_{N-1} - C_{N-2}) \\ &\quad (X_{j-N+2} - X_{j-N+1})+ Y_{j,p} \end{aligned}$$

So,

$$\begin{aligned} Y_{j+1} &= Y_{j+1}^1 + Y_j \\ &= C_0(X_{j+1} - X_j)+ (C_1 - C_0)(X_j - X_{j-1})+\ldots.+ (C_{N-1} - C_{N-2}) \\ &\quad (X_{j-N+2} - X_{j-N+1})+ Y_{j,p}+ Y_j \end{aligned}$$

The first N partial products are multiplications between differential coefficients and differential inputs, except the first term that has only one differential operand. Thus, a shorter multiplier than that in DCM can be used. Besides, we stored the summed value of the compensated term $C_0(X_j - X_{j-1})$, …, instead of the individual compensated terms, which will save (N-2) unnecessary memory accesses and additions. The compensated values $Y_{j,p}$ and Y_j can be easily obtained from the previous computation Y_j and stored for the use of this computation Y_{j+1}. Thus, for each output Y, we need only two extra storage ($Y_{j,p}$ and Y_j) and two extra additions.

In this DCIM formulation, we only use first order difference of input. That is because higher order difference of input will have larger range than the range of first order difference due to the continuous property of analog input. However, the higher order difference of coefficients can be easily derived as that in [1]. So we will use the definition of m-th order difference of coefficients in this paper.

III. ARCHITECTURE DESIGN

Fig. 1 shows the architecture design with the DA technique. DA, since its introduction by Pele and Liu[2], has been regarded an efficient bit-serial computational operation to do the filter operations in a single direct step. Without loss of generality, if we express the differential input X_j as bit-level representations in unsigned fraction

$$X_j = \sum_{l=0}^{W_{dx}-1} X_{j,l}\, 2^{-l}$$

we can reformulate the FIR operations as

$$\begin{aligned} Y_j &= \sum_{k=0}^{N-1} C_k X_{j-k} \\ &= \sum_{k=0}^{N-1} C_k \Big(\sum_{l=0}^{W_{dx}-1} X_{j-k,l}\, 2^{-l}\Big) = \sum_{l=0}^{W_{dx}-1} \Big(\sum_{k=0}^{N-1} C_k X_{j-k,l}\Big) 2^{-l} \end{aligned}$$

The terms inside the bracket are precomputed and stored in a memory since all coefficients are constant. The input data is used to address the memory and the result is accumulated to obtain the output. Since all filter coefficients are constant, we can use ROM to store the precomputed partial results and avoid to compute them on line.

As illustrated in Fig. 1 , the ROM address uses the same bit position in all input data. So we can group the high transition probability LSB of all inputs at the same time and separate them with other low transition probability MSB bits. Since MSB bits are often zero, we may skip the memory access and accumulation operation and thus save power. Besides, the ROM realization in DA also offers the low power possibility since the high power multiplication is replaced by just table look up and accumulation.

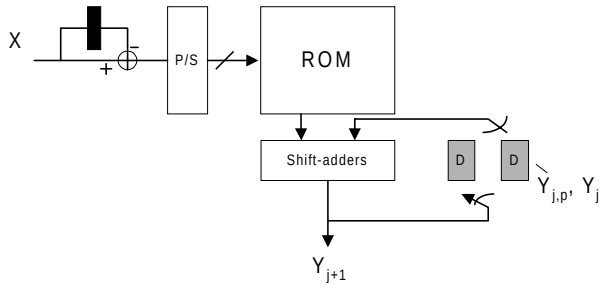


Fig. 1 Architecture design with the DA technique.

Combining with the previous DCIM algorithm, we can obtain the architecture as shown in Fig. 1. First, we subtract the previous input from the current input to obtain the differential input. Then the word-serial bit-parallel input is converted to word-parallel bit-serial output to access the ROM. The ROM table stores the DA result of $Y_{j,p}$ (multiplication result of differential input and differential coefficients). The ROM table result is accumulated by the shift-adders. The compensated value is separated stored in a memory and added back to the output at the final accumulation cycles. The ROM table can be halved by offset binary coding[2] or partitioned into several parts. Readers who interested in these methods can refer the paper by White[2].

IV. COMPUTATIONAL ANALYSIS

The average net computational energy per Y, denoted by E_{NET}, is the sum of multiplication cost (E_{MULT}), data and coefficients storage access cost (E_{MEM}), product terms accumulation costs (E_{ACC}), overhead storage accesses costs(E'_{MEM}), and overhead additions costs (E'_{ADD}). So

$$\{E_{NET}\}_{DCM} = \sum_{\forall i} \{E_i\}_{DCM} + \sum_{\forall j} \{E_j\}_{DCM}$$

$$\{E_{NET}\}_{DCIM} = \sum_{\forall i}\{E_i\}_{DCIM} + \sum_{\forall j}\{E_j\}_{DCIM}$$

$$\{E_{NET}\}_{DCIM,DA} = \sum_{\forall i}\{E_i\}_{DCIM,DA} + \sum_{\forall j}\{E_j\}_{DCIM,DA}$$

$$i \in \{MULT, MEM, ACC\}, j \in \{MEM, ADD\}$$

We have considered both designs with or without DA architecture. For ease of comparison, let the average energy dissipated in a single bit full addition or subtraction be denoted by E_{add}. Let the average energy dissipated per bit in a single bit arithmetic shift of a field be denoted by E_{shift}. Let the magnitude of the m-th order difference between coefficients be W_d^m bits smaller than the original coefficients. For DCM, according to [1], we have the followings

$\{E_{mult}\}_{DCM}=N((W_x+1)(W_c-W_d^m-1)/2*E_{add}+(W_x+W_c-W_d^m)(W_c-W_d^m-1)*E_{shift})$

$\{E_{MEM}\}_{DCM}=N(W_x + W_c-W_d^m)E_{mem}$

$\{E_{ACC}\}_{DCM}=(N-1)(W_c+ W_x +\lceil\log_2 N\rceil)E_{add}$

$\{E'_{MEM}\}_{DCM}=2mN(W_c+W_x) E_{mem}$

$\{E'_{ADD}\}_{DCM}=mN(W_c+W_x) E_{add}$

As shown in the formula, the overhead of DCM $\{E'_{MEM}\}_{DCM}$ and $\{E'_{ADD}\}_{DCM}$ is proportional to order of difference and tap numbers, which will quickly offset the energy savings by the differential coefficients.

Similar formula can be obtained for DCIM without DA architecture since both DCM and DCIM without DA use shift and add operation for multiplications.

$\{E_{mult}\}_{DCIM}=N((W_x-W_{dx}+1)(W_c-W_d^m-1)/2*E_{add}+(W_x-W_{dx}+W_c-W_d^m)(W_c-W_d^m-1)*E_{shift})$

$\{E_{MEM}\}_{DCIM}=N(W_x -W_{dx} + W_c-W_d^m)E_{mem}$

$\{E_{ACC}\}_{DCIM}=(N-1)(W_c+ W_x-W_{dx} +\lceil\log_2 N\rceil)E_{add}$

$\{E'_{MEM}\}_{DCIM}=2m(W_c+W_x+\lceil\log_2 N\rceil)E_{mem}+2(W_c+W_x+\lceil\log_2 N\rceil)E_{mem}+2W_xE_{mem}$

$\{E'_{ADD}\}_{DCIM}=m(W_c+W_x+\lceil\log_2 N\rceil)E_{add}+(W_c+W_x+\lceil\log_2 N\rceil)E_{add}+W_xE_{add}$

For DCIM with DA architecture, the multiplication cost will be the cost of memory table look up. So

$\{E_{mult}\}_{DCIM,DA}=(W_x-W_{dx})(W_c-W_d^m + W_x-W_{dx} +\lceil\log_2 N\rceil)E_{MEM}$

As to the data and coefficient storage cost, since the DA architecture has distributed the coefficients in the ROM, only the differential input data access cost has to be counted.

$\{E_{MEM}\}_{DCIM,DA}=N(W_x-W_{dx})E_{mem}$

The product term accumulation power is the power consumed by the shift-adder in the architecture.

$\{E_{ACC}\}_{DCIM,DA}=(W_x-W_{dx})(W_c-W_d^m+W_x-W_{dx}+\lceil\log_2 N\rceil)(E_{add}+E_{shift})$

The overhead required by DCIM is the compensated Y_j and $\{Y_{j,p,n}:n=1:m\}$ and increases as the order of differences between coefficients used increases. The overhead for m-th order difference between coefficients is (m+1) storage and (m+1) extra additions. The overhead cost (read and write) is

$\{E'_{MEM}\}_{DCIM,DA}=2m(W_c+W_x+\lceil\log_2 N\rceil)E_{mem}+2(W_c+W_x+\lceil\log_2 N\rceil) E_{mem}+2W_xE_{mem}$

$\{E'_{ADD}\}_{DCIM,DA}=m(W_c+W_x+\lceil\log_2 N\rceil)E_{add}+(W_c+W_x+\lceil\log_2 N\rceil) E_{add}+W_xE_{add}$

So the average net energy savings, denoted by S_{NET}, is

$S_{NET1}=(\{E_{NET}\}_{DCM}-\{E_{NET}\}_{DCIM})/\{E_{NET}\}_{DCM}$

The storage models used in this paper are the same as that in [1]. For square root model, $\{E_{MEM}\}=K_1\sqrt{\{S_{MEM}\}}$. For logarithm model, $\{E_{MEM}\}=K_2\log_2\{S_{MEM}\}$. For linear model, $\{E_{MEM}\}=K_3\{S_{MEM}\}$. The memory sizes $\{S_{MEM}\}$ for the two methods are

$\{S_{MEM}\}_{DCM}=N[(m+1)(W_c+W_x)- W_d^m]$

$\{S_{MEM}\}_{DCIM}=N(W_c-W_d^m+W_x-W_{dx})+2m(W_c+W_x+\lceil\log_2 N\rceil)+2(W_c+W_x+\lceil\log_2 N\rceil)+2W_x$

$\{S_{MEM}\}_{DCIM,DA}=N(W_x-W_{dx})+3*(W_c+W_x+\lceil\log_2 N\rceil)+(W_c-W_d^m+W_x-W_{dx}+\lceil\log_2 N\rceil)2^N$

For large N, the ROM size will be impractical large. One solution is to partition long taps number into smaller pieces. For fair comparison, we will use the low power library data in [1]. In that library, $E_{shift}=200$, and $E_{add}=170$.

Fig. 2 and Fig. 3 shows the energy savings S_{NET1} for a 26-tap Hamming windows. For fair comparison, we use the same parameters as in [1]. The W_{dx} is assumed to be 2 for conservative estimation. W_d^m will be 2m for 26-tap filters. As shown in the figures, DCIM is superior than DCM due to the savings of unnecessary compensated value accesses. DCIM with DA has worse performance at linear and square root memory model due to the large memory size, but it gives large energy savings at the logarithmic memory model. So low word length and high order of differences design is suitable for DCIM without DA. The delays of the DCM and DCIM without DA is comparable since we use add and shift to replace multiplications in both designs. Throughput of DCIM with DA will be higher than that of DCM since we use the precomputed data and avoid computations.

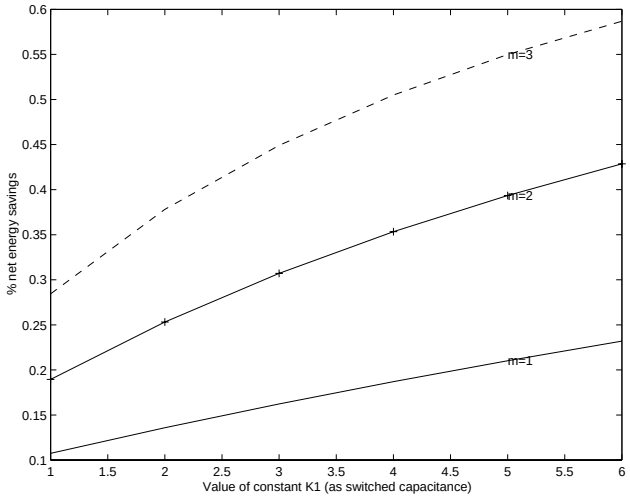
V. CONCLUSION

This paper presents low power FIR realization by using both first order difference between input and various order differences between coefficients. For an example filter, DCIM shows greater energy savings than DCM due to the savings of unnecessary compensated value operations. DCIM with DA architecture can be applied to short filters for large energy savings. For long filters, we can directly apply DCIM without DA to avoid the exponential growth of DA ROM table.

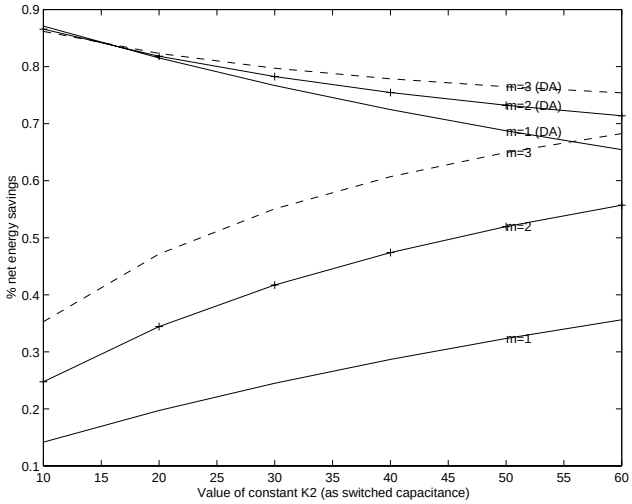
REFERENCES

[1] N Sankarayya, K. Roy, and D. Bhattacharya, "Algorithms for Low Power and high speed FIR filter realization using differential coefficients," IEEE Trans. CAS, vol. 44. No. 6.Pp.488-497, 1997.

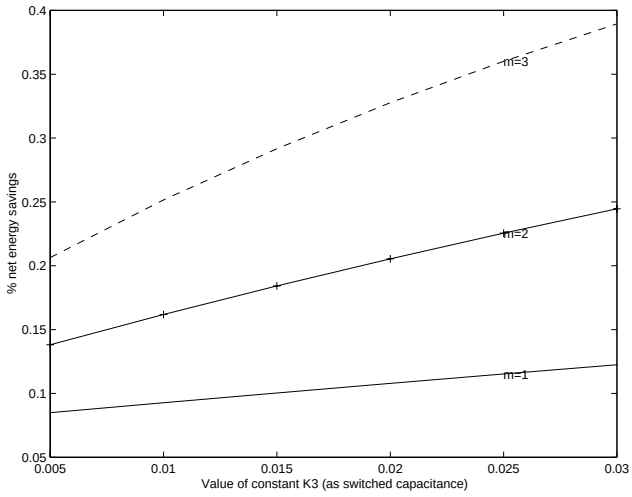
[2] S.A White, "Applications of distributed arithmetic to digital signal processing: a tutorial review," IEEE ASSP. July. 1989.



(a) Square root memory model.

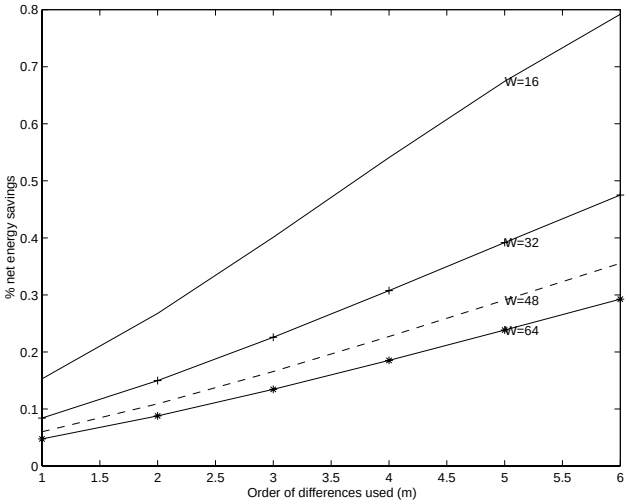


(b) Logarithmic memory model.

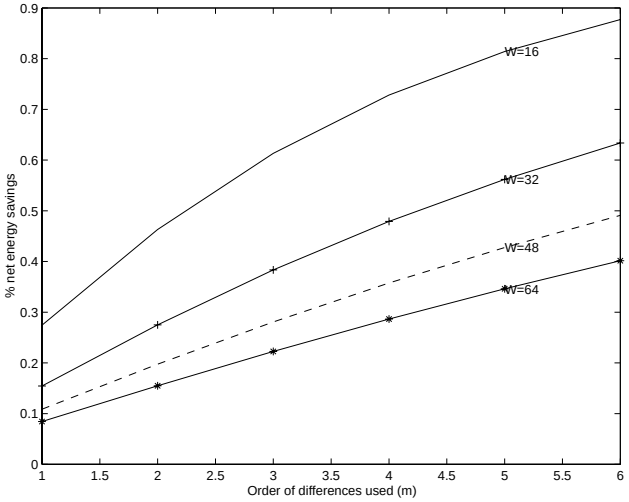


(c) Linear memory model.

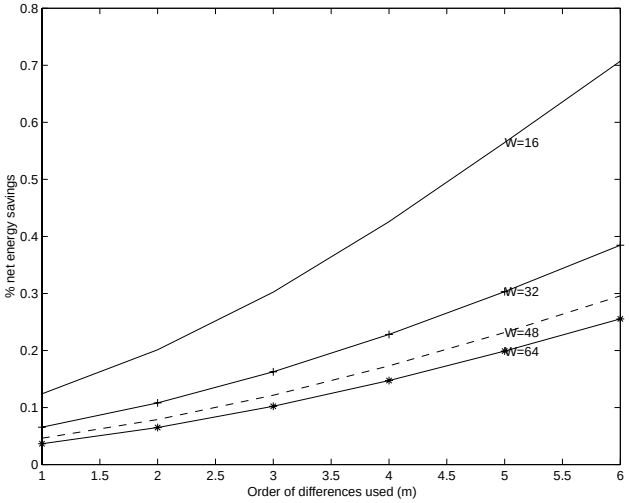
Fig. 2 Energy Savings as a function of the memory access constant.



(a) Square root memory model (K1=1.0).



(b) Logarithmic memory model (K2=20).



(c) Linear memory model (K3=0.005).

Fig. 3 Energy savings as a function of the order of differences used.