OPTIMIZED SCHEDULING AND MAPPING OF LOGARITHM AND ARCTANGENT FUNCTIONS ON TI TMS320C67X PROCESSOR

Mei Yang, Yuke Wang, Jinchu Wang and S.Q. Zheng

Department of Computer Science
Univ. of Texas at Dallas, Richardson, TX 75083
{meiyang, yuke, sizheng}@utdallas.edu

ABSTRACT

DSP processors have gained more importance and popularity in implementing communication systems. Efficient implementation of *logarithm* and *arctangent* functions on DSP processors is necessary for applications such as digital receiver used in modern radar systems and digital communication systems. This paper presents a general scheduling and mapping optimization method based on grain packing to implement the two functions on TI TMS320C67X architecture with multiple parallel function units. Experimental results of our optimized implementation on TMS320C67x have achieved up to 79.5% performance improvement over TI 'C67x library functions. Our optimization method and techniques can also be applied to other DSP processors with parallel execution units.

1. INTRODUCTION

As a critical technology in modern radar systems and software radio systems, digital receiver uses direct IF-sampling and digital signal processing approach to obtain In-phase (I)/Quadrant (Q) signal or Amplitude (A)/Phase (θ) [1]. Figure 1 shows the block diagram of a typical digital receiver. ADC converts the analog IF signal into digital IF stream. The digital IF stream is fed into the IF processor where I/Q values are synthesized. After base-band processing, the A/θ values are obtained.

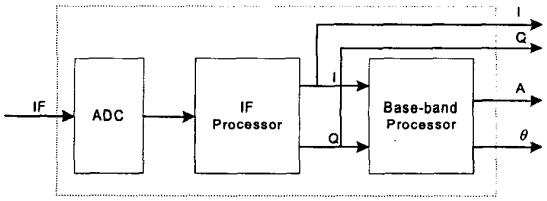

Figure 1. *Block diagram of a typical digital receiver*

The *logarithm* and *arctangent* functions play an important role in the base-band processing to get A/θ, which are computed by equation (1) and (2). The base-band processing can be implemented by a DSP processor.

$$A = 20\log_{10}\sqrt{I^2+Q^2} = 10\log_{10}(I^2+Q^2) \quad (1)$$

$$\theta = arctg(\frac{Q}{I}) \quad (2)$$

With performance of up to 1GFLOPS and a complete set of development tools, TI TMS320C67x ('C67x) offers cost-effective solutions to high-performance floating-point DSP applications [3]. The high performance levels of the TMS320C67x DSP chips are made possible by an innovative architecture, as shown in Figure 2 [3], which is designed to meet variety applications. The 'C67x processor consists of three main parts: CPU (or the core), memory, and peripherals. The CPU has two data paths (A and B), each data path has four functional units (.L, .S, .M and .D) and a register file with 16 32-bit registers. A cross data bus is used for exchanging data between two register files. Each functional unit is controlled by a 32-bit instruction. Thus, the VLIW structure of 'C67x with 256-bit-width instruction allows execution up to 8 32-bit instructions in one clock cycle. 'C67x also provides a large bank of on-chip memory and has a powerful and diverse set of peripherals.

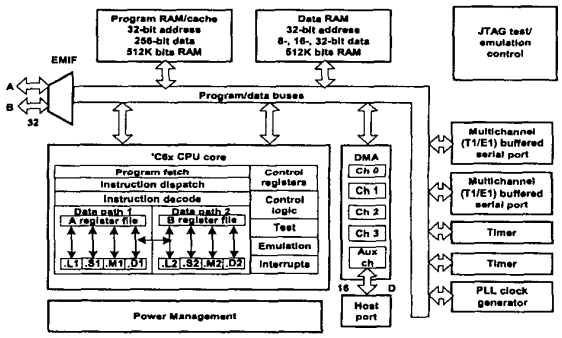

Figure 2. *TMS320C67x block diagram*

Each 'C67x instruction has a functional unit latency of 1 clock cycle followed by variable delay slots, each delay slot corresponding to 1 clock cycle [3]. If instructions are scheduled and mapped in parallel properly, 'C67x hardware resources and delay slots can be fully utilized to speed up computations.

The C library of 'C67x provides both single-precision and double-precision floating-point functions of *logarithm* and *arctangent* [4]. Our discussion is focused on single-precision floating-point functions. Table 1 lists the maximum running time (plus the cost of pre-processing and post-processing) with C overhead in clock cycles (CLKs) of base *e/10 logarithm* functions and one/two-argument *arctangent* functions on Code Composer Studio 1.2 (CCS1.2). All optimization options are opened in our testing.

logf(x)	*log10f(x)*	*atanf(x)*	*atan2f(y,x)*
136 CLKs	154 CLKs	266 CLKs	424 CLKs

Table 1. *The maximum running time of TI 'C67x library functions*

The total running time of *log10f* and *atan2f* is 578 CLKs, equivalent to 3.46 μs when the CPU is working at 167 MHz. This computation time is not good enough for some high-speed digital receivers with 1M I/Q input rate or up.

In this paper, we propose a general scheduling and mapping optimization method based on grain packing to implement the two functions on TI TMS320C67x with parallel execution units. Optimization techniques of pre-processing are also discussed in this paper. Experimental results of our optimized implementation on TMS320C67x have achieved up to 79.5% performance improvement over TI 'C67x library functions.

The rest of the paper is organized as the following: section 2 gives analysis of the implementation of *logarithm* and *arctangent* functions in TI 'C67x library; section 3 presents our optimization method and techniques; experimental results and comparisons are discussed in section 4; section 5 concludes the paper.

2. ANALYSIS OF TI 'C67X LIBRARY FUNCTIONS

Generally math functions are implemented by polynomial approximation. The following MacLaurin Series are used to compute *logarithm* and *arctangent* functions [5].

$$ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} \quad \ldots \quad -1 < x \leq 1 \tag{3}$$

$$atan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots \quad |x| < 1. \tag{4}$$

To obtain the desired precision, TI 'C67x library approximate these functions by polynomials of certain sufficient orders [2], [4]. The *logarithm* function, *logf* (natural or base e logarithm) routine uses

$$ln(1+x) = \sum_{i=1}^{8} \{a[i]x^i\} + e(x), \quad 0 \leq x \leq 1, \quad |e(x)| \leq 3 \times 10^{-8} \tag{5}$$

where $e(x)$ or $e(x)/ln(1+x)$ is the error function which has been usually minimized in the min-max (equi-ripple) sense. The error minimization is done by selecting appropriate coefficicents $a[i]$ $(1 \leq i \leq 8)$. Values of $ln(1+x)$ for $1+x$ outside the convergent range are found as:

$$ln(1+x) = ln(f) + n\,ln(2), \quad x > 1, \quad 1 \leq f = \frac{1+x}{2^n} < 2. \tag{6}$$

$log_{10}(x)$ is obtained by multiplication of $ln(x)$ and $1/ln(10)$.

The *arctangent* function, *atanf* (arc or inverse tangent) routine uses the following expansion [2]:

$$atan(x) = x \sum_{i=0}^{8} \{a[2i]x^{2i}\} + xe(x), \quad -1 \leq x \leq 1 \quad and \quad |xe(x)| \leq 2 \times 10^{-8}. \tag{7}$$

Values of $atan(x)$ for x outside the convergent range are obtained by noting the following identity:

$$atan(x) = atan((x-1)/(x+1)) + Pi/4, \quad x > 1 \tag{8}$$

The bilinear conversion $y = (x-1)/(x+1)$ is needed for $x>1$. The case for $x<-1$ is handled by using $|x|$ for all calculations before the final result is adjusted. *atan2f(y,x)* calls *atanf(x)* by passing y/x as the argument.

The implementation of of *logf(x)* and *atanf(x)* [4] is based on expansions (9) and (10) respectively. The computation of both equations involves iterations of multiplication-accumulation calculations (MACs) of $a[i]$ and x^i.

$$ln(1+x) = k_1x + k_2x^2 + k_3x^3 + k_4x^4 + k_5x^5 + k_6x^6 + k_7x^7 + k_8x^8 = x(k_1 + x(k_2 + x(k_3 + x(k_4 + x(k_5 + x(k_6 + x(k_7 + xk_8)))))))) \tag{9}$$

$$atan(x) = x + k_1x^3 + k_2x^5 + k_3x^7 + k_4x^9 + k_5x^{11} + k_6x^{13} + k_7x^{15} + k_8x^{17} = x(x^2(k_1 + x^2(k_2 + x^2(k_3 + x^2(k_4 + x^2(k_5 + x^2(k_6 + x^2(k_7 + x^2k_8)))))))). \tag{10}$$

Taking the example of *atanf* function, Figure 3 shows the C code of equation calculation [4]. The main loop is very simple for C implementation. However, the high data dependency between iterations (the $(i+1)^{th}$ iteration result depends on the i^{th} iteration result) and long delay slots of instructions of TI 'C67x [6] make parallel scheduling very difficult.

```
float x;
{int i;
 float *p=a;
 result=(x^2)*(*p++);
 for (i=8-1; i>0; i--)
        result=(x^2)*(((result)+(*p++)));
}
```

Figure 3. *The main loop of atanf function in TI 'C67x library*

Figure 4 shows the compiled assembly code of each iteration of the C main loop of *atanf* on Code Composer Studio1.2. The latency of each iteration is 8 CLKs. The total number of clock cycles of the equation calculation is 68. The maximum total running time (plus the cost of pre-processing and post-processing) with C overhead of *atanf* function is 266 CLKs. The maximum running time of *logf, log10f,* and *atan2f* is 136, 154, and 424 CLKs respectively. All clock cycle numbers are reported by CCS1.2.

```
1. LDW      *p_to_coef++, coef_I     ;delay 4 slots
2. MPYSP    f^2, sum, result         ;delay 3 slots
3. NOP3
4. ADDSP    result, coef_i, sum      ;delay 3 slots
5. NOP      2
```

Figure 4. *The assembly code for each iteration of atanf function in TI 'C67x library*

The efficiency of these functions cannot satisfy the requirement of some high-speed digital receivers. Unfortunately, the performance of these functions cannot be improved using all the optimization options of the parallelizing compiler CCS1.2. Although the compiler optimization techniques (such as software pipelining [7], [8], [9]) are suitable for applications with a large number of independent variables, such as FFT, FIR, IIR, etc., they do not work well for applications with dependent variables such as *logf* and *atanf*. In the following, we will introduce our optimization method and techniques for implementing *logarithm* and *arctangent* functions.

3. OPTIMIZED IMPLEMENTATION OF LOGARITHM AND ARCTANGENT FUNCTIONS

3.1. Algorithm Optimization

For the same computational problem, different algorithms result in different data dependency structures. When mapped to a

particular architecture, different algorithms may lead to different performance. Our goal of optimization is to improve the parallelism of computations by finding an algorithm suitable for TMS320C67x architecture. To achieve this goal, we need to reduce data dependency and utilize delay slots of instructions. Our approach consists of the following steps: (1) Find a computing process that is equivalent to the polynomial expansion and with minimum data dependency, (2) Construct fine-grained program graph, (3) Grain packing to produce coarse-grained program graph, (4) Scheduling and mapping coarse-grained nodes to appropriate function units of 'C67x so that delay slots can be fully utilized.

Taking the example of *atanf*, equation (10) can be derived as the following equation:

$$atan(x) = x + k_1x^3 + k_2x^5 + k_3x^7 + k_4x^9 + k_5x^{11} + k_6x^{13} + k_7x^{15} + k_8x^{17}$$
$$= x + x^3(k_1 + k_2x^2) + x^7(k_3 + k_4x^2) + x^{11}(k_5 + k_6x^2) + x^{15}(k_7 + k_8x^2). \quad (11)$$

The fine-grain data dependency graph [11] is thus obtained in Figure 5, which is different from traditional fine-grain program graph [10]. Here each node denotes an instruction, L, M and + represents the instruction for load, multiplication and addition respectively. Inside each node is the destination operand. Each edge represents the data dependency of next node (edge head) to previous node (edge tail). The number on each edge gives the instruction delay slots. There are 31 nodes in this graph.

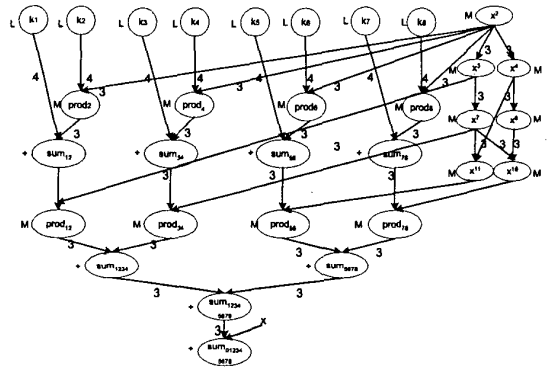

Figure 5. *The fine-grain data dependency graph of equation (11)*

Then we use grain packing [10] to optimize parallel scheduling. Instead of improving performance by reducing inter-processor communications in [10], grain packing technique is used here to fully utilize functional units of TI 'C67x and delay slots of its instructions. Figure 6 presents the grain packing graph which groups 31 small nodes into 16 larger nodes (named as $A_1 \ldots A_9$, $B_1 \ldots B_7$). We observe that operations of the nodes at the same level can be executed in parallel; nodes on different levels should be scheduled according to data dependency; and computation in each node should be done sequentially and at the same side of functional units. Computation of x^{4i+3} in node A_9 can be scheduled in the delay slots of other nodes.

TI 'C67x has eight functional units operate in parallel, but each floating-point instruction can only use 2 functional units, such as LDW (.D1 and .D2), MPYSP (.M1 and .M2), and ADDSP (.L1 and .L2). Due to this limitation, the maximum paralleled operations of loads, multiplications, or additions in one clock cycle are 2. Therefore we get the optimized parallel scheduling of *atanf* in Figure 7. Each column represents the specified functional unit, each cell shows what computation is assigned to the functional unit at each clock cycle, the shaded cells are delay slots of the previous instruction. The total number of clock cycles of equation calculation is only 29, which reduces 57.4% compared to 68.

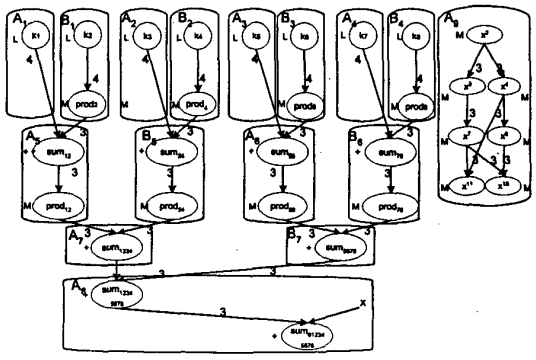

Figure 6. *The grain-packing graph of equation (11)*

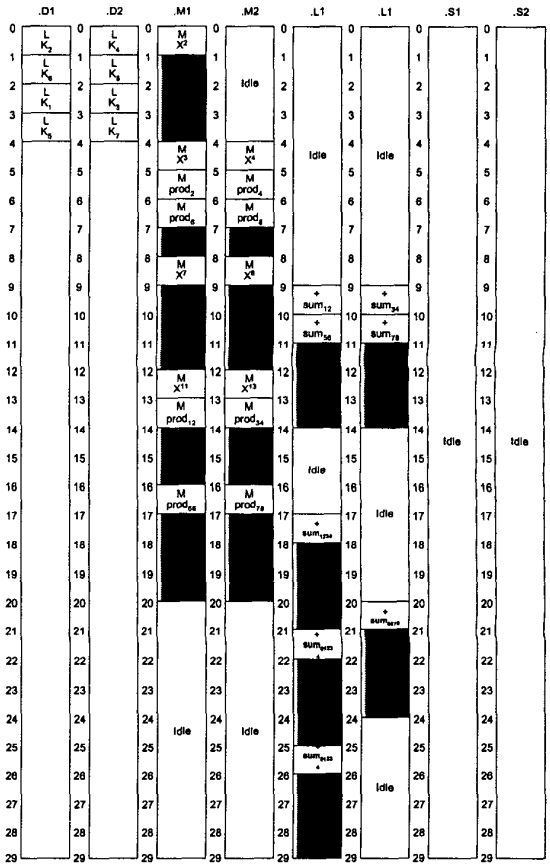

Figure 7. *Parallel scheduling of calculation of equation (11)*
Notes: $prod_{2i+2}$ represents $(k_{2i+2}x^2)$, $prod_{(2i+1)\,(2i+2)}$ represents $x^{4i+3}(k_{2i+1}+k_{2i+2}x^2)$, $0 \le i \le 3$; $sum_{(2i+1)\,(2i+2)}$ represents $(k_{2i+1}+k_{2i+2}x^2)$, and so on, $0 \le i \le 3$.

Similarly, equation (9) can be transformed as equation (12). The total number of clock cycles of equation calculation of *logf* is 25.

$$\begin{aligned} ln(1+x) &= k_1x + k_2x^2 + k_3x^3 + k_4x^4 + k_5x^5 + k_6x^6 + k_7x^7 + k_8x^8 \\ &= x(k_1 + k_2x) + x^3(k_3 + k_4x) + x^5(k_5 + k_6x) + x^7(k_7 + k_8x) \end{aligned} \quad (12)$$

This general optimization method is also applicable to other polynomial approximation math functions.

3.2 Optimization of Pre-processing

The pre-processing (processing must be done before equation calculation) is also important for the code efficiency. Our pre-processing optimization techniques include the implementation of division using RCPSP instruction and efficient separation of the exponent and the mantissa.

Here we will introduce the implementation of division using RCPSP. Taking the example of *atan2f(y,x)*, it calls *atanf(x)* by passing $\frac{y}{x}$ as the argument. In *atanf(x)*, if $|x| > 1$, a bilinear conversion must be done as $\frac{|x|-1}{|x|+1}$ [2], [4]. Totally two divisions are involved in the pre-processing of *atan2f*. In TI 'C67x, division is implemented by multiplying the numerator to the reciprocal of the denominator [4]. The reciprocal is approximated by Newton-Raphson algorithm [6]to extend the precision of the mantissa. Equation (13) gives the principle of Newton-Raphson algorithm, *c* is the input variable, an initial estimate *x[0]* is needed. After each iteration, the precision is doubled.

$$x[i+1] = x[i](2 - cx[i]) \quad (13)$$

In our implementation, we first examine $|y|$ and $|x|$, if $|y| > |x|$, do $\frac{|y|-|x|}{|y|+|x|}$ directly. Thus only one division is needed in the pre-processing of *atan2f*. We use instruction RCPSP [6] to get the reciprocal with 8-bit precision. A 2-iteration Newton-Raphson approximations can guarantee 23-bit precision. In this way the clock cycles needed for pre-processing of *atan2f* is reduced from 201 to 38. The pre-processing of *atanf* ($|x|>1$) can also be optimized similarly.

4. EXPERIMENTAL RESULTS

To evaluate the effectiveness of our proposed optimization method, experiments of *logf, log10f, atanf,* and *atan2f* are conducted on TMS320C67x. All assembly programs are written and debugged on Code Composer Studio1.2. Test programs of our optimized functions and TI 'C67x library functions are profiled and number of clock cycles for these functions with C overhead are measured by CCS1.2. The maximum running time (all pre-processing and post-processing is counted) in CLKs of our optimized implementation of the four functions vs. TI 'C67x library functions are given in Table 2.

Noticeably, the improvement of the optimized assembly code is over 67% and up to 79.5%. The total number of clock cycles for the *log10f* and *atan2f* now is 134, which is only 0.80 μs when CPU works at 167 MHz.

Function	Optimized Function (CLKs)	TI 'C67x Library function (CLKs)	Improvement Percent
logf	44	136	67.6%
log10f	47	154	69.5%
atanf	79	266	70.3%
atan2f	87	424	79.5%

Table 2. *Comparison of our optimized implementation vs. TI 'C67x library function in terms of total clock cycles*

5. CONCLUSIONS

DSP processors have gained more importance and popularity in recent years for a wide variety of applications. Real-time applications such as digital receivers require efficient implementation of *logarithm* and *arctangent* functions. This paper presents a general scheduling and mapping optimization method based on grain packing to implement the two functions on TI TMS320C67x architecture with multiple parallel function units. Optimization techniques of pre-processing are also discussed in this paper. Experimental results of our optimized implementation have achieved significant performance improvement over TI 'C67x library functions. Our optimization method and techniques are applicable to other polynomial approximation math functions and other DSP processors with parallel execution units.

REFERENCES

[1] Fakatselis, J., Chester,D.B., "Subsampling digital IF receiver implementations", *Southcon/96. Conference Record*, 1996, pp. 92-97.

[2] Texas Instruments, *A Collection of Functions for the TMS320C30*, 1990.

[3] Texas Instruments, *TMS320C62x/C67x Technical Brief*, 1998.

[4] Texas Instruments, rts.src, TMS320C62x/C67x library functions package with Code Composer Studio1.2.

[5] Engineering Fundamentals website, http://www.efunda.com/math/taylor_series/taylor_series.cfm

[6] Texas Instruments, *TMS320C62x/C67x CPU and Instruction Set Reference Guide*, 1998.

[7] Ray Simar Jr., "DSP Architectures, Algorithms, and Code-generation: Fission or Fussion?", *Proc. IEEE Workshop on Signal Processing Systems*, 1997, pp. 50-59.

[8] Yin-Tsung Hwang, Ying-Chou Chuang, "High Performance Code Generation For VLIW Digital Signal Processors", *Signal Processing Systems, 2000. SiPS 2000. 2000 IEEE Workshop on, 2000*, pp. 683 -692

[9] Yin-Tsung Hwang and Jer-sho Hwang, "Simulated Evolution Based Parallel Code Generation for Programmable DSP Processors", *Journal of Information Science & Engineering*, vol. 14, No. 1, 1998, pp. 138-165.

[10] Kai Hwang, *Advanced Computer Architectures*, McGraw-Hill, 1993.

[11] Texas Instruments, *TMS320C62x/C67x Programmer's Guide*, 1998.