A Characterization of Branch Behavior in DSP Applications

Lucian Codrescu, Erich Plondke
Motorola, Semiconductor Products Sector, DSP Platforms
7700 W. Parmer Lane
Austin, TX 78726
{lucian.codrescu, erich}@motorola.com

ABSTRACT

This paper presents an analysis of the control-flow behavior of modern DSP applications. Historically, DSP applications have been characterized as being loop-dominated with very little non-loop branch activity. Further, these loops are typically characterized as inner loops with large iteration counts that can be determined before the loop starts. DSP processors are generally organized around this philosophy. They typically contain special hardware to accelerate inner loop branches (zero-overhead hardware loop support), but they suffer cycle penalties when general-purpose software branches or non-standard loops are encountered.

This paper aims to test these historical assumptions. The control-flow attributes of a set of modern production-quality DSP software is analyzed. First, the overall contribution of branches to performance is shown. The branches are then characterized by type: loop, unconditional, conditional, procedure call and return, and indirect branch. For loop branches, the important attributes are the average iteration count, the average loop length, and when the iteration count can be determined. Conditional branches are analyzed by showing a histogram distribution of taken percentage. Procedure call and return is analyzed by studying the average number of consecutive returns before a call.

Taken together, the data presented in this paper helps to illuminate the changing direction of DSP applications. The DSP architect can use this data to help guide future designs.

1. INTRODUCTION

DSP architectures strive for the highest possible performance on important applications while meeting stringent cost constraints. Pipelined processors can sustain substantial cycle degradation due to control-flow (branches) if the hardware is not properly optimized to the application behavior. Because of cost constraints, it is important to target the dominant branch behavior in a cost-effective manner.

Execution time in DSP applications has historically been dominated by a small number of inner-loops. These loops are characterized as being short, having no branches inside the loop, and having iteration counts that can be determined before the loop starts. The zero-overhead hardware loop structure present in most DSPs today is an excellent, cost-effective solution for this type of branch behavior.

The industry trend in signal processing is towards more complex applications written in C and away from small kernels written in assembly. As the applications become bigger and more complex, it is important to understand if the historical philosophy on DSP control flow behavior still holds true. If the control-flow behaviors are changing, then new DSP processors need new mechanisms to address these changing behaviors. Towards that goal, this paper presents a detailed analysis of the branch behavior in current, complex DSP applications.

This software includes EFR, AMR, G729, MPEG4, JPEG, SPIP, and MP3. Two benchmarks from the EMBC networking suite are included for comparison. The software is written in C for the StarCore SC140 DSP. Optimizations include the use of intrinsic functions, hand restructuring of C (unrolling, loop merging, etc), and hand assembly optimization of a few key kernels. Given these realistic, real-world applications, data on control flow behavior is collected.

First, the general importance of control flow is determined by calculating how frequently branches interrupt the instruction stream. These branches are then categorized into loop, conditional, procedure, and indirect. Loop branches are further broken down into loops with known iteration counts and those with unknown counts. Then, for each branch category, the important execution characteristics are studied.

For loops, a distribution of the average length and iteration counts is shown. For conditional branches, a distribution of taken percentage is shown, and for procedure call and return, data is collected on the average call number of consecutive returns without a call.

Branch behavior is principally a characteristic of the application. The data presented demonstrates that historical perceptions of DSP control flow are accurate for many applications. For example, in the speech coders considered, 80% of the branches are loop tails. However, these characteristics change dramatically as the DSP is used to perform other tasks. For the multi-media applications, this fraction drops to 50%, and it drops even further with networking applications. The non-loop branches studied are predominately either strongly biased conditional branches or procedure call/return branches.

2. METHODOLOGY

The applications were written in C and optimized for the Starcore SC140 DSP[3]. This DSP features a 4MAC/2AGU Variable-Length-Execution-Set (VLES) architecture. A highly optimizing compiler converts C code into explicitly parallel assembly code. The compiler used is the MetroWerks StarCore compiler version 2.0. In some cases, key routines where hand-coded in assembly.

The applications were then simulated using a custom instrumented version of the Starcore simulator. The simulator was modified to collect more detailed information on branch behavior. The applications used are summarized in the following table.

Table 1. Applications

Application	Description
EMBC_ospf	Djikstra's shortest path algorithm
EMC_routelookup	Lookup using a Patricia Tree
AMR_{enc/dec}	Adaptive Multi-Rate Speech Coder
ERF_{enc/dec}	Enhanced Full-Rate Speech Coder/Decoder
G729_{enc/dec}	G729 Speech Coder/Decoder
JPEG_{enc/dec}	JPEG image encoder/decoder
SPIP	Image Processing Application
MPEG2_dec	MPEG Video
MPEG4_dec	MPEG Video Decoder w/ H.263 Support
MP3	MPEG layer 3 audio decoder

3. Branch Frequency and Type Analysis

First, data is collected on how frequently branches interrupt the instruction stream. Simulations were run to collect the average number of VLES between branch instructions. A VLES is an execution-set and typically takes 1 cycle.

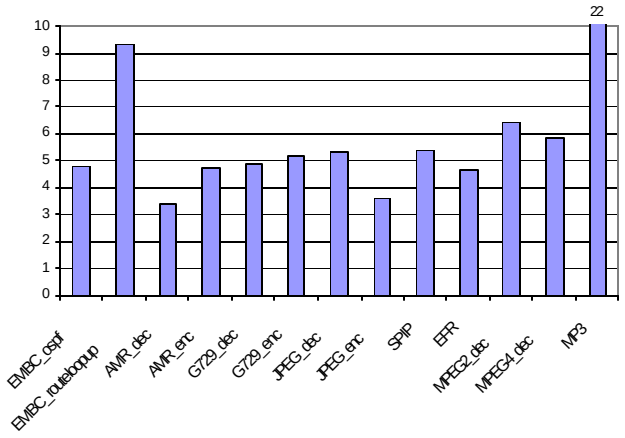

Figure 1 Avg VLES Between Attempted Branch Instructions

Data on the average number of VLES between attempted branch instructions in shown in Figure 1. Attempted branches are all change-of-flow instruction, taken or not taken. For most applications, branches occur every 3 to 6 cycles.

While this is quite frequent, typically taken branches are more important as they potentially interrupt the flow of instructions through the pipeline. To understand how frequently taken branches occur, Figure 2 shows the distance between taken branch instructions.

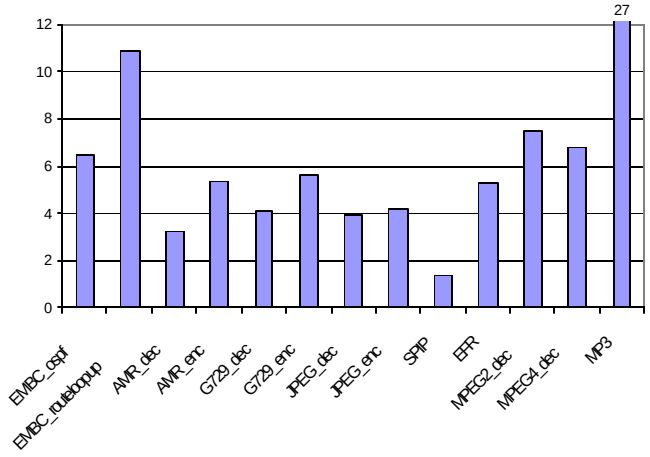

Figure 2 Avg. VLES Distance between taken branches

Branches occur quite frequently in these applications. This implies that the cycle penalty for branches is a first-order performance effect. Only one of the applications, MP3 has a much longer branch-to-branch interval of 27 VLES. In this application, the main computation loop was extensively hand-unrolled.

Next, a distribution of the branch type is collected. The types are conditional branch, indirect branch, loop with unknown trip count, loop with known trip count, subroutine call and return, and unconditional branch. The results are shown in Figure 3. It can be seen that the traditional loop with known trip count is a

substantial portion of the total branches. However, there are two important points to notice. First, non-loop branches represent a non-negligable portion. For the codecs, around 20% of the branches are non-loop. Second, the portion of non-loop branches is highly dependent on the application. Media benchmarks tend to have more non-loop branches. Over half the branches in MPEG4 and MP3 are not the traditional loop branch. This behavior is even more pronounced in the EMBC ospf benchmark, where only 25% of the branches are loop branches.

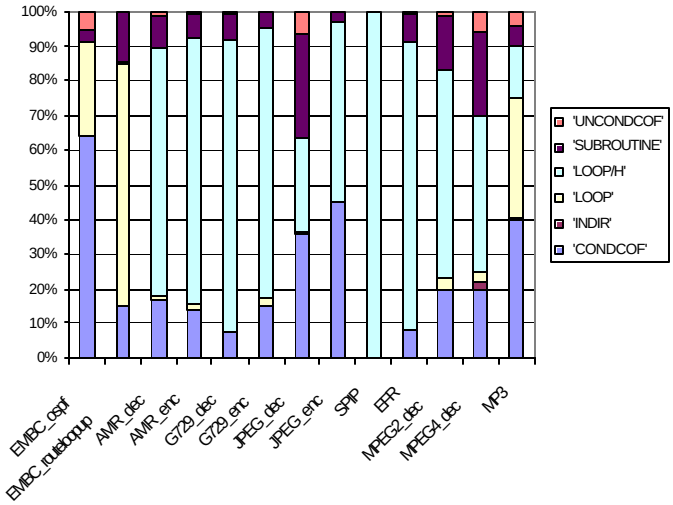

Figure 3 Branch Type Distribution

3.1 Loop Analysis

For loops, the important characteristics are the average length, the average iteration count, and whether or not the trip count (iteration count) is known before the loop starts. Data on the loop length is shown in Figure 4. Loops tend to be short. In most applications, over 90% of the loops are less than 8 VLES in length. Fully 20% of the loops are only 1 or 2 VLES in length. (Note that the category 0 is the range (0-1], while the category 1 is (1-2], etc. Therefore, the 0 category represents the number of loops of length 1.) Two of the benchmarks, MP3 and MPEG2, feature longer loops. In MP3 this is due to loop unrolling. In MPEG2, this is due to code structure which emphasizes large outer *while* loops rather than tight inner loops.

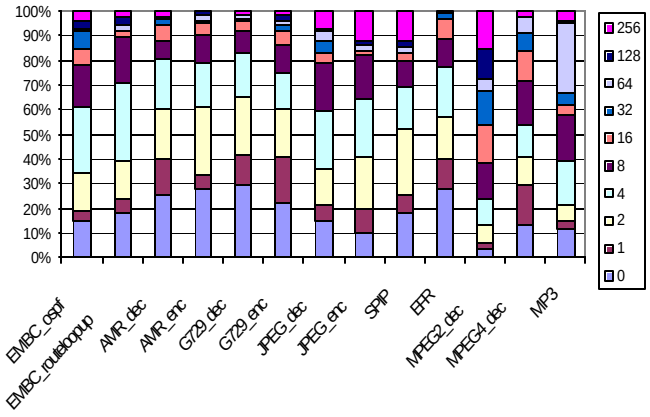

Figure 4 Loop Length Distribution

The next graph, Figure 5, shows the distribution of loop iteration count for the applications. Many of the applications feature loops with small trip counts. In 70% of the loops in these applications, the loop iteration count is less than 16.

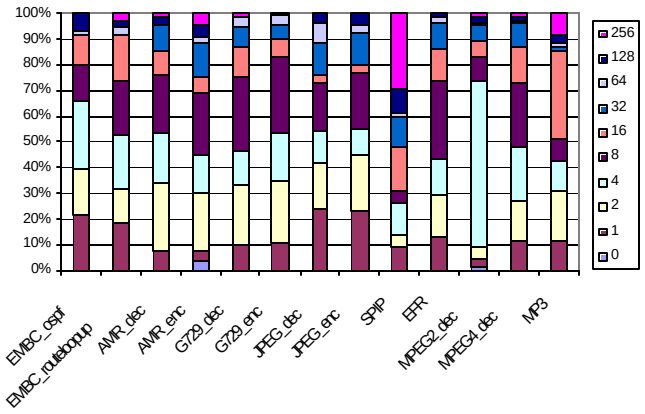

Figure 5 Loop Trip Count Distribution

Next, the determinability of the loop trip count is studied. Figure 6 shows the percentage of loops for which the trip count could be determined before the loop executed. Here, a very clear distinction is seen between DSP-type applications and non-DSP applications. For the codecs and multi-media applications, close to 100% of the loops have known trip counts. In the EMBC networking benchmarks, nearly none of the loops have known trip counts. These are generally while loops where the exit condition depends on data computed inside the loop.

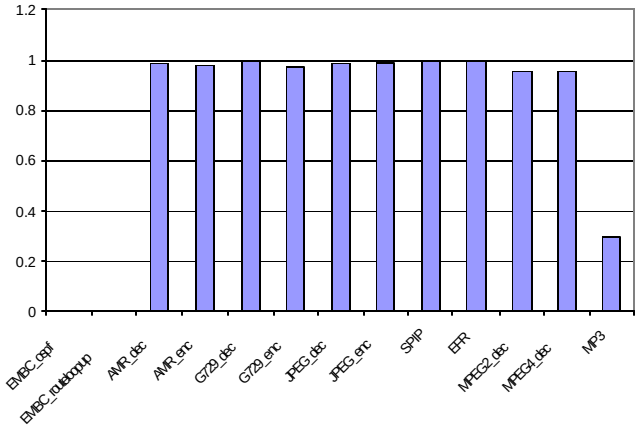

Figure 6 Percentage of Loops with Known Trip Counts

3.2 Non-loop branches

For the non-loop branches, Figure 7 shows a distribution of taken percentage. The data shows that most of the branches are strongly biased. Over 60% of the dynamically executing non-loop branches are taken between 95-100% of the time. And more than 20% of the branches are taken less than 10% of the time. This behavior is consistent with a broad range of applications and is the basic reason why branch prediction mechanisms are effective [1]. The most cost effective way to exploit this behavior is through code layout optimization [2].

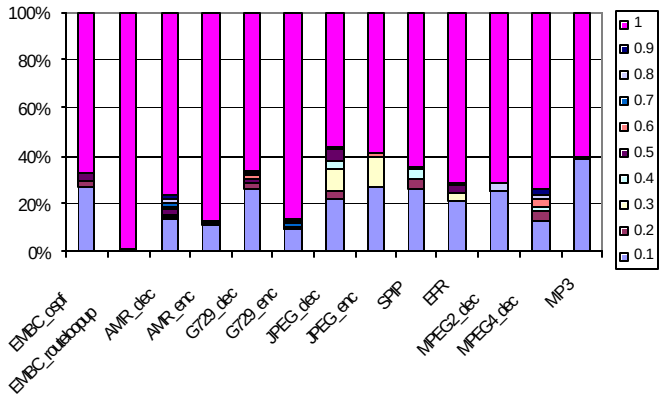

Figure 7 Non-loop Taken Branch Distribution

Another category of branch instruction is procedure call and return. Depending on the application, these overheads can be important. For example, 30% of the branches in jpeg decoder were procedure calls or returns. Techniques to accelerate procedure call and return often rely on Return Address Stack (RAS) mechanisms [5]. It is important to know how many consecutive returns are encountered before a call instruction, as this metric indicates the necessary depth of the return address stack. Figure 8 shows that this metric is usually around 1. In particular, jpeg decoder needs only a single entry return address stack.

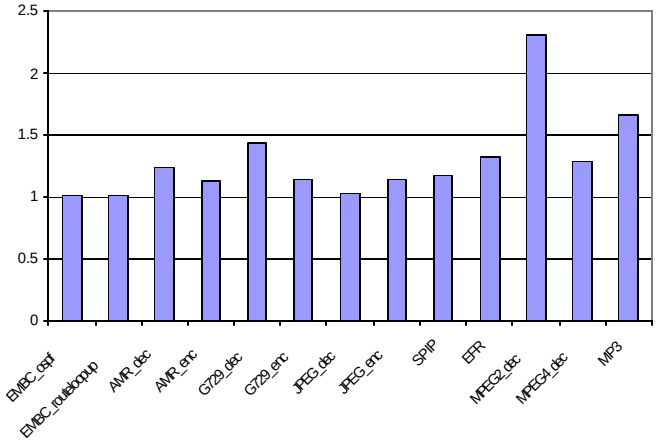

Figure 8 Consecutive Returns before Call

The final category of branch instruction is indirect (or computed) branch. For the applications studies, indirect branches contributed a negligible amount towards the total number of dynamically executed branch instructions. For this reason, no further data was collected on indirect branches.

4. SUMMARY

This work studies a set of realistic production-quality application software written for the SC140 DSP. The branch behavior is considered in detail. Data collected demonstrates a number of trends in DSP software.

For many of the applications, loops represent 80% of the branches. While this is a substantial amount, an important observation is that this amount is declining as DSPs take on tasks outside of their traditional roles. Several of the multi-media applications had only 50% of the branches involved in loop tails, while for the networking applications, this percentage is down to 25%.

The loop bodies and trip counts tend to be small, usually less than 8 VLES and 16 iterations, respectively. Further, for the vast majority of these loops, the trip count can be determined before the loop executes. However, these loop characteristics change depending on the nature of the application. The EMBC networking benchmarks shown have radically different behavior. They emphasize longer outer loops with long but unknown trip counts.

The non-loop branches studied are very strongly biased. Over 90% of these branches are either taken greater than 90% of the time or taken less than 10% of the time. Procedure call depths tend to shallow, with the average number of consecutive *returns* before a *call* close to 1. Finally, indirect branches are shown to have a negligible impact on cycles.

5. CONCLUSIONS

DSP processors must include cost-effective and high-performance support for control flow. The most important category is loops with small bodies, and small and known trip

counts. The traditional zero-overhead hardware loop is an excellent choice. Machines that rely on other mechanisms, such as long delay slots [4], are at a disadvantage in code size and/or performance.

Conditional branches are an important fraction of the DSP software workload (more than 20%), and this fraction is growing. However, these branches are generally very strongly biased. DSPs such as the SC140 with short branch latencies together with software tools that perform intelligent block layout can essentially mitigate most control flow cycle penalties.

6. REFERENCES

[1] M. Evers, "Improving Branch Prediction by Understanding Branch Behavior," PhD thesis, University of Michigan, Ann Arbor, 2000

[2] K. Pettis and R.C. Hansen, "Profile guided code positioning," in *Proc. ACM SIGGPLAN Conf. Programming Language Design and Implementation*, June 1990, pp. 16-27.

[3] StarCore DSP Technology , *SC140 DSP Core Reference Manual*, June 2000.

[4] Texas Instruments Incorporated, *TMS320C6000 CPU and Instruction Set Reference Guide*, March 1999.

[5] C.F. Webb. Subroutine call/return stack. *IBM Tech. Disc. Bulletin*, 30(11), April 1988.