A Comparison of x86 with Alpha

on Image Manipulation Programs

Abstract

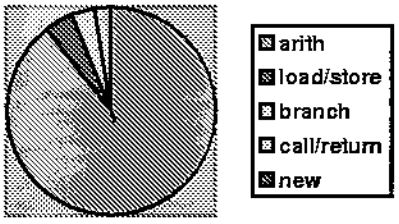
In this paper, we compare Pentium II Xeon with Alpha 21164, based on instruction level behavior on image manipulation programs for the two architectures. We cover statistical information such as instruction distribution, effect of conditional/unconditional branches, basic block sizes, length of uninterrupted code sequences and effectiveness of static/dynamic branch prediction. Overall, the two architectures show similar behavior on broad issues. However, there are certainly some differences in actual numbers obtained for the two experiments, which is expected because the two programs, as well as the two architectures, are developed by different people.

Introduction

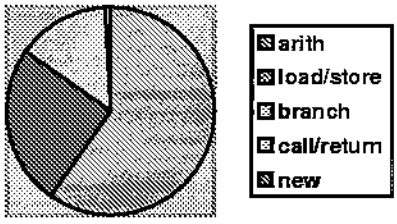
For the first experiment, we used the Visual Etch tool to instrument the Microsoft Photo Editor program on Pentium II Xeon. The second experiment was done with the tool Atom and the behavior of "xv" was analyzed on Alpha 21164. In both these experiments, an image was loaded and several typical image manipulation tools were applied to it. These included resizing, sharpening, converting to "chalk and charcoal", and so on. The instrumentation tools developed for Visual Etch and Atom analyzed these programs and produced data for the respective platforms, which is compared in what follows.

Instruction Mix

Although integer arithmetic and logical instructions dominated both experiments, the actual numbers for the distribution were quite different. The similarity, of course, arises because of the common nature of the two programs – image manipulation -which requires arithmetic/logical instructions the most. The



x86



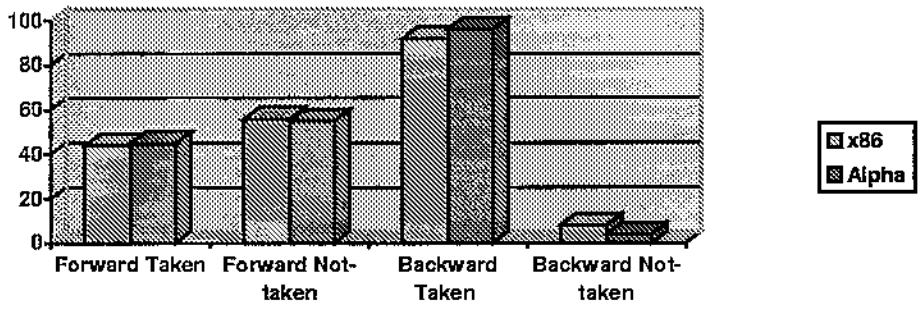
Alpha

differences can possibly be explained by the fact that the two programs were developed by different people, using different compilers and other tools, and involving different architecture specific optimizations.

It is interesting to note that the number of FP operations was negligible compared to integer arithmetic in both the experiments. So both programs use integers for internal representation and processing of images.

Conditional Branches / Static Prediction

The two programs behaved similarly for conditional branches, though Alpha was slightly more consistent in taking backward branches and not taking forward ones. This would mean the static predictor that predicts what we said just now would work better for the Alpha. It is also interesting to note that the numbers for backward branches do indicate loops, but those for forward branches do not necessarily

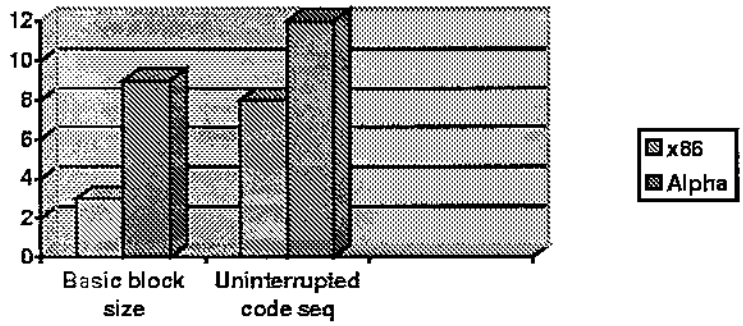


indicate any programming styles (such as forward branches mean exceptional case, otherwise fall through).

As remarked earlier, static prediction works well (and almost the same) for both cases – 72% for x86 and 76% for Alpha. But then, dynamic prediction works better even with small history size and small tables.

Basic Blocks / Uninterrupted Code Sequences

It is interesting to observe that the "xv" execution showed fairly large basic block sizes (~9 instructions) compared to the Photo Editor execution (~3). The same thing also holds for the length of uninterrupted code sequences (12 vs. 8).

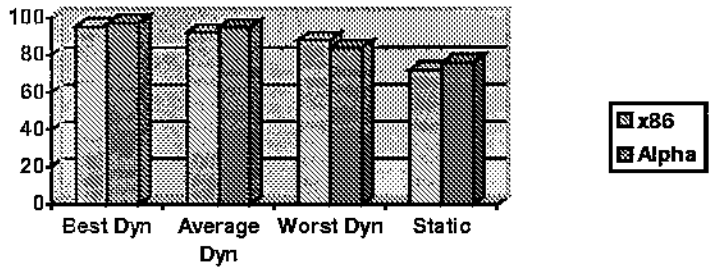


The difference can again be ascribed to different programmers, different tools and different architectures. However, for both experiments, lengths of uninterrupted code sequences are generally much more than (about twice) the actual basic block sizes. This is something that we expect because of jump labels and not-taken branches.

Dynamic Branch Prediction

In the experiments that were conducted, Alpha seemed to have much better performance for the correlated dynamic predictor. For example, a (1024-4-2)-predictor (i.e. 1024 prediction table size, 4 bit history, 2 bit predictor) reached upto 97.1% for the execution of "xv" on Alpha 21164, whereas the best for Photo Editor

on P II Xeon was 95.2% using a (4096-4-2)-predictor. This is definitely contrary to our expectations because x86, with its long pipelines, should be a better performer when it comes to branch prediction – this is the only way it can avoid huge mis-prediction penalties. One possible explanation for the numbers obtained could be that the specific programs chosen had this behavior and this does not hold for the two architectures in general. A second possibility, which we don't preclude, is that there was some error in collecting data for one of the two architectures. A third reason could be that Alpha and x86 actually employ some variations of the technique that we used to predict branches dynamically, and hence, in reality, achieve different success rates.



In spite of these differences, there are certain deductions that are common for the two architectures. First, the dynamic branch predictors perform much better then the simple static scheme, especially on forward branches. Second, some of the cache parameters have very little influence. For instance, for the particular image processing programs, varying the size of the prediction table did not change prediction quality significantly. This is probably due to a large number of loops with fairly few conditional branches. Third, the size of the global history and the number of bits of prediction have the most impact on predictor performance. Changing from two bits to one bit of prediction and from four bits to zero bits of global history can worsen the mis-prediction rate by 3-4%.

Conclusion

The two architectures show different statistics in terms of actual distribution of instructions and sizes of basic blocks / uninterrupted code sequences. However, general trends remain the same – instructions are mostly integer arithmetic, uninterrupted code sequences are much longer than actual basic blocks, and conditional branches work most as predicted statically (at for backward branches, which occur much more often than forward branches). Static prediction works better on x86 than on Alpha whereas it's the other way for dynamic prediction. However, for either architecture, dynamic prediction is much better, even with a small table and few history bits.