Improving FFTW Benchmark to Measure Multi-core Processor Performance

William J. Pilaud
Curtiss Wright Controls Embedded Computing
bpilaud@curtisswright.com

The challenge for the embedded industry is to find an impartial method of comparison between different processor technologies, architectures, and system topologies for COTS based digital signal processing (DSP) boards. Often, most benchmark systems specialize in one type of use case that might simulate real applications but may lead to invalid conclusions or opinions for system selection. Increasingly, the embedded industry is investigating multi-core processors as a platform for DSP. DSP performance is a difficult thing to benchmark as processor companies change processor technology from single to two or more cores, cache architectures, memory connection strategies and chip-to-chip interconnects. This paper will describe the investigation of an Intel processor and propose a modest addition to Fastest Fourier Transform in the West (FFTW) benchmark to highlight these architecture changes.

1. Introduction

Matteo Frigo and Steven G. Johnson of MIT created and released FFTW on March 24, 1997, which provided a unique digital signal processing benchmark for a remarkable number of general-purpose processors. FFTW is a math library used to compute Discrete Fourier Transforms (DFTs)1. The FFTW library continues to evolve as processor, network and software technology change by adding new features, compliers, operating systems support, and library features. FFTW has benchmark software called benchFFT. This benchmark shows speed results of many processors. However, all of these benchmarks results are the performance of a single core or thread and do not show the true capability of multi-core processors. Today, most general-purpose processor companies offer multi-core processors, therefore a change of the benchFFT that would show the performance improvement of these architectures that would be beneficial to DSP integrators.

2. Understanding benchFFT

3.

FFTW's benchFFT calculates mflops as

$$mflops = \frac{5\ x\ N\ x\ log_2(N)}{(time\ for\ one\ FFT\ in\ \mu s)}$$

By examining this calculation, as FFT size increases then given some fixed amount of time, processing time (FLOPS) would have to increase. Therefore, the more FLOPs the lower the latency time if processing time is significant to system design. DSP integrators can use this benchFFT FLOP time to estimate processor need for their applications and estimate latency. However, depending on processor architecture, memory connect and even processor to processor interconnect this latency is very hard to predict. Figure 1 shows a single precision complex power of two out-of-place benchFFT for the FFTW3 algorithm for a 2.4 GigaHertz Intel Pentium 4.2.

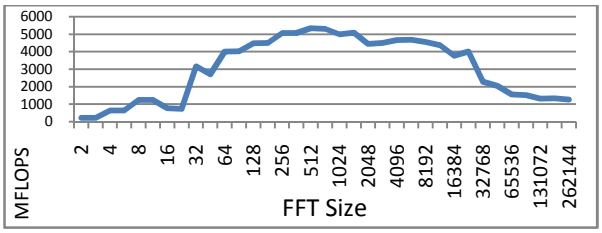

Figure 1: benchFFT for 2.4G P4 using FFTW3

This bell shaped curve for single precision complex even numbered FFTs is typical for all processors on the FFTW benchmark website. As the size of the FFT increases to a certain size, the FFT latency gets smaller or FLOPs increase. At some FFT size, the hardware, algorithm, and data size reach an optimum and then quickly, as the FFT increases further in size the performance drops because the processor has to wait for data to move from slower memory sub-systems to processor memory (cache). Therefore, with very large FFT sizes the latency is directly proportional to non-cache memory speed.

Some people have speculated that the reason for high latency on the smaller FFT is due to the overhead of library function calls or some sort operating system overhead, but this is not true.

The reason why FFTW shows lower than theoretical maximum is that the FFTW algorithm is not just doing floating-point operations. Depending on FFT size, FFTW is spending most of the processing time, moving data from one variable to another. On a 2-point FFT, 98.4% of the processing time is used for data movement and other operating system overhead with less than 2% doing floating point math. At optimal sizes between 256 and 2048 points, the processor peaks to nearly 40% of theoretical maximum floating point capability. Also, for the smaller FFT sizes the FFTW designers elected to use loops. If the FFTW designers had unrolled the loops then the FFT would improve from 2% to 10% floating point utilization for some of the smaller FFT sizes. However, this "un-rolling the loops method" for calculating FFTs is very prohibitive when the FFT size becomes larger because the FFT code would grow exponentially with this method and would quickly perform worse than loop based

algorithms. FFTW does not really benchmark the processors actual floating point performance; it really shows the optimal floating point problem size to processor system data movement capability.

4. Multi-core support to benchFFT

Benchmarking the performance of a Multi-core processor is not the same as multiplying the results of one core by the number of cores on the processor. Multi-core processors typically have architectures that *share* cache and external memory, that will affect benchmark performance. For example, two separate 1 GHz processors could run some large FFT algorithms faster than a 1 GHz dual-core processor. To complicate matters, general-purpose processors manufacturers are always adding math acceleration instructions to the processor core. The chip designers have added mathematical functions, widened inner processor pipelines, increased math register size or added specialized cache structures to improve math calculation speed. Simply multiplying the single core benchFFT results by the number of cores will not result in the true performance of the multi-core processor. The only way to find out how fast a processor can run FFTs is to benchmark all of the cores in the processor simultaneously.

Change the benchmark routine by running multiple instances of the same benchmark with processor affinity, the resulting processor performance is a different from the predicted performance. The taskset command In LINUX controls processor affinity. For example, here is a snippet of the benchmark script used in benchFFT.

```
if test "$speed" = yes; then
   time="`$program $useropt --report-benchmark
            --time-min $time_min –speed $problem | tail -1`" || wait
```

This script generates fftw3 results like:

```
fftw3 scof 2 302.86 3.3018589e-08 0.002147
fftw3 scof 4 834.58 4.7928065e-08 0.002011
```

By modifying the benchmark script to add the command taskset, each core will now run the benchmark independently and dump the results simultaneously (the example here is a 4 core processor).

```
taskset -c 3 $program $useropt –report-benchmark --time-min $time_min
--speed $problem > a1.out &
taskset -c 2 $program $useropt --report-benchmark --time-min $time_min
--speed $problem > a2.out &
taskset -c 1 $program $useropt --report-benchmark --time-min $time_min
--speed $problem > a3.out &
taskset -c 0 $program $useropt --report-benchmark --time-min $time_min
--speed $problem > a4.out &
```

Now that each core is running the benchmark, the results would look something like this:

```
fftw3 scof 2 297.62 3.36e-08 0.000422
fftw3 scof 2 296.74 3.37e-08 0.000471
fftw3 scof 2 302.11 3.31e-08 0.000428
fftw3 scof 2 294.99 3.39e-08 0.00047
```

By running a script or a program that adds the floating-point figures together and averaging the setup time and accuracy, the resultant processor performance of a quad core processor would look like this:

```
fftw3 scof 2 1191.5 3.3575e-08 0.000448
```

Comparing the single core benchmark with its FLOPs results multiplied by four and the actual benchmark running in all cores in parallel, we can see that the prediction differs in the actual (see figure 2).

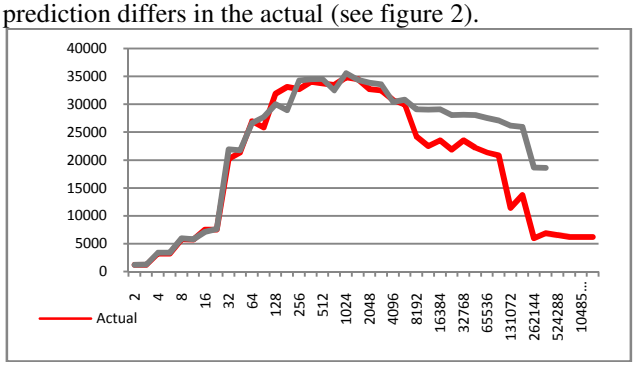

Figure 2: Actual Quad Core FFT performance

Interestingly, except for some of the small calculations the actual FFT performance of the quad processor does not actually match predicted performance. In fact, some FFT sizes the system seems to perform better than expected. However, as soon as the FFT size and algorithm size require the processor to use the slower external memory (non-cache) the processor performance quickly drops to some factor related to the external memory bandwidth.

5. Conclusion

Multi-core processors will increase their adoption in the embedded computer market. Multi-core processor manufacturers are aggressively optimizing the power to performance of these processors. In effect, multi-core chip providers are optimizing size, weight and power for the signal processor applications.

Therefore, a scalable benchmark applicable to multi-core DSP performance could be a method of predicting what and when multi-core DSP boards can perform an DSP application.

---

1 "*FFTW FAQ-Section 1 Introduction and General Information*", retrieved from FFTW: http://www.fftw.org/faq/section1.html#whatisfftw, April 29, 2009.

2 “2.4 GHz Pentium 4, GNU compilers”, http://www.fftw.org/speed/Pentium4-2.4GHz-gcc/