RTOS Scalability Experiment

Testing the parallel scalability of running multiple Impulse C tasks concurrently within the same Xilkernel environment

Introduction

This document discusses the scalability experiments that were performed to robustness of Xilkernel when adding and communicating with different IP cores generated by Impulse C to the same project. The hardware computation was based on the simple bubblesort algorithm, which is performed many times within the FPGA before displaying the actual sort data.

Based on our experiments we empirically noticed that the performance of the Impulse C processes (which are wrapped by POSIX threads) does not scale very well when more than one more and more IP cores are added to the system. Our reasoning for this is discussed in the concluding section of this document. First, we discuss the algorithm that was partitioned into hardware and how it interfaces with the software running on the Power PC.

Hardware Algorithm

In order to generate a hardware process that consumed a great deal of time we chose to implement the simple bubble sort algorithm. The difference is that, after every run through the algorithm, the sort order for the data collection is changed, so we are guaranteed that each sort will encounter the worst case time complexity of **$O(n^2)$**. The implementation for this hardware process can be seen below, which constantly sorts a data collection of 32 elements (i.e. STREAMDEPTH = 32).

```
// Sort the data using the bubblesort algorithm
for (counter = 0; counter < NUM_LOOPS; counter++)
{
	for (index = 0; index < STREAMDEPTH; index++)
	{
		for (innerIndex = 0; innerIndex < STREAMDEPTH - 1; innerIndex++)
		{
			if (counter % 2 == 0)
			{
				if (samples[innerIndex] > samples[innerIndex + 1])
				{
					nSample = samples[innerIndex + 1];
					samples[innerIndex + 1] = samples[innerIndex];
					samples[innerIndex] = nSample;
				}
			}
			else
			{
				if (samples[innerIndex] < samples[innerIndex + 1])
				{
					nSample = samples[innerIndex + 1];
					samples[innerIndex + 1] = samples[innerIndex];
					samples[innerIndex] = nSample;
				}
			}
```

```
                }
        }
}
```

Figure 1: Hardware source code (where NUM_LOOPS = 10000)

The input data to this function is streamed using a FIFO queue that is 32 elements deep and 32 bits wide, essentially resulting in the transfer of 32 integers from the PowerPC processor to the FPGA fabric using the PLB. Similarly, the output sort data from this function is streamed to the PowerPC across the PLB using a FIFO that is 32 elements deep and 32 bits wide.

Experiment Metrics

To test whether or not the Xilkernel was robust enough to handle increasingly large amounts of IP cores, we simply synthesized more instances of this bubble sort process and spawned more producer and consumer threads to interface with them. Our primary measurement was the amount of computation time (in terms of clock cycles) for each thread to run from start to finish, which means the data was properly generated, transferred to the hardware process to be sorted, and then returned for display. The source code that implemented this timing is shown below.

```
xil_printf("Starting BubbleSort.\r\n");

TIMER_INIT(timer);
START_TIME(timer, startCycles);

/* Insert code for the threads here */
pthread_create(&pid1, NULL, (void*)Producer, input1);
pthread_create(&cid1, NULL, (void*)Consumer, output1);
pthread_create(&pid2, NULL, (void*)Producer, input2);
pthread_create(&cid2, NULL, (void*)Consumer, output2);
pthread_create(&pid3, NULL, (void*)Producer, input3);
pthread_create(&cid3, NULL, (void*)Consumer, output3);
pthread_create(&pid4, NULL, (void*)Producer, input4);
pthread_create(&cid4, NULL, (void*)Consumer, output4);

/* Join on the threads before they exit */
pthread_join(pid1, NULL);
pthread_join(cid1, NULL);
pthread_join(pid2, NULL);
pthread_join(cid2, NULL);
pthread_join(pid3, NULL);
pthread_join(cid3, NULL);
pthread_join(pid4, NULL);
pthread_join(cid4, NULL);

END_TIME(timer, endCycles);

xil_printf("Total time: %10d\n", endCycles - startCycles);

xil_printf("Ending BubbleSort.\r\n");
```

Figure 2: Xilkernel profiling code

The results from our experiments are shown in Table 1 below.

Table 1: RTOS performance measurements

Test ID	Core instances	Compute time (cycles)
1	1	136529691
2	3	174566501
3	4	221190315
4	5	Non-responsive (halt)

We noticed that the computation time seemed to increase at a somewhat linear rate before the entire system became non-responsive with a total of five bubblesort cores in the FPGA fabric. The growth of this computation time is shown below in Figure 3.

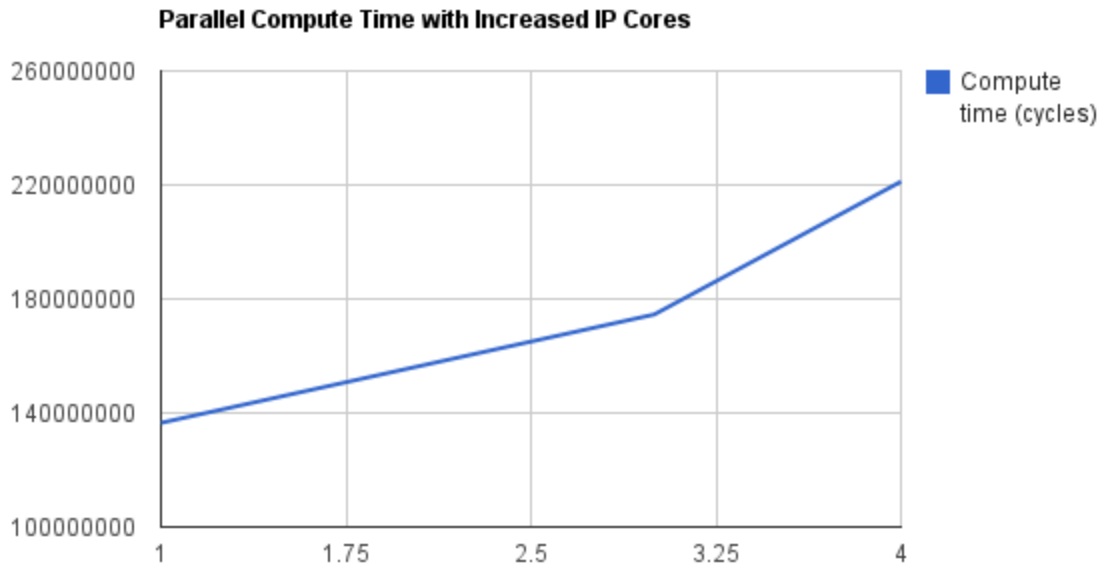

Figure 3: Scalability test results with up to 4 concurrent Impulse C IP cores

Concluding Remarks

Despite our original hypothesis that the computation time would increase at a logarithmic rate (i.e. less than linear), it has been shown that the computation time increases linearly as more IP cores and associated threads are added to the system. To the best of our knowledge, this effect is a direct result of the following:

1. The PLB interface that is used to transmit data between the PowerPC and FPGA fabric has a constrained bandwidth (usually 32 bits), meaning that only one thread could control the PLB for data transfer at a time. This results in sequential data transfer across the bus, which in turn means that the hardware processes do not receive data to process concurrently. Therefore, this delay in streaming data to and from the FPGA due to the PLB bandwidth constraints likely has an impact on the lifetime of the threads.
2. Due to the simple nature of the Xilkernel, it is possible that the thread management logic isn't tuned for high performance applications. The overhead in managing multiple threads running concurrently might be larger than a more mature RTOS such as QNX.