1 Part A: Tracking Process Activity

1.1 Hardware Information

CPU name	Intel Core i7-3610QM CPU
CPU speed	2.30GHz
L1 Cache Size	32 KB
L2 Cache Size	256 KB
L2 Cache Size	6144 KB
Memory	5995 MB

1.2 Initialization

This experiment was done on my personal computer, that is running Linux Ubuntu 14.04 LTS OS. ALso CPU affinity was set to CPU 0 in order to get accurate readings

1.3 Experimenting with CPU Frequency

In order to measure CPU frequency I caculated multiple sleep calls in a loop so that the ith sleep waits i2 seconds and during that time the number of cycles is measured. Once I got multiple frequency measurements I sorted them using quicksort and then tried to find the pair of CPUs that were the closestclustered and took the highest measurement of the pair.

I ended up getting a CPU Frequency of 2294 MHz, which is pretty close to 2.3 GHz.

1.4 Experiementing with Threshold

I determined threshold through running the experimentation multiple times, first starting at 1000 and increasing the value by 500 until consistent results were produced. I ended up staying with a threshold of 2500.

1.5 Conclusion

Results for this experiment were placed in a file called part_a_output.txt, and was called using run_experiment_A 30

Some measurements below

total inactive time 0.189322 ms

total time 84.064094 ms active time 83.874772 ms average active time 2.89223 ms

By observation, since we have a CPU Frequency of 2.3GHz and average active time of 2.89223 ms than we can say an interrupt occurs about every 6,652,129 cycles from (2.89223 * 1000 * 2300). There are also periods that have short durations, such as less than 1 ms, which suggests it is a little inconsistant with regular timer intervals which I suggest are the ones that happen after every 3 to 4 ms activity. These early interrupts could have been due to processes running in the back such as Google Chrome or Text Editor Sublime. The total time of the experiment was 84.064094 ms and total interrupt time is 0.189322 ms, Therefore the percentage of time lost due to interrupts was 0.023% of the total time.

2 Part A2: Measuring Context Switch Time

2.1 HARDWARE INFO AND SETUP

In this section we measure the context switch time. When two share a CPU, the operating system schedules the processes which results in the CPU switching contexts between the different processes. Our approach was to measure the time between two on a single CPU. To do this, I forked off processes which measure

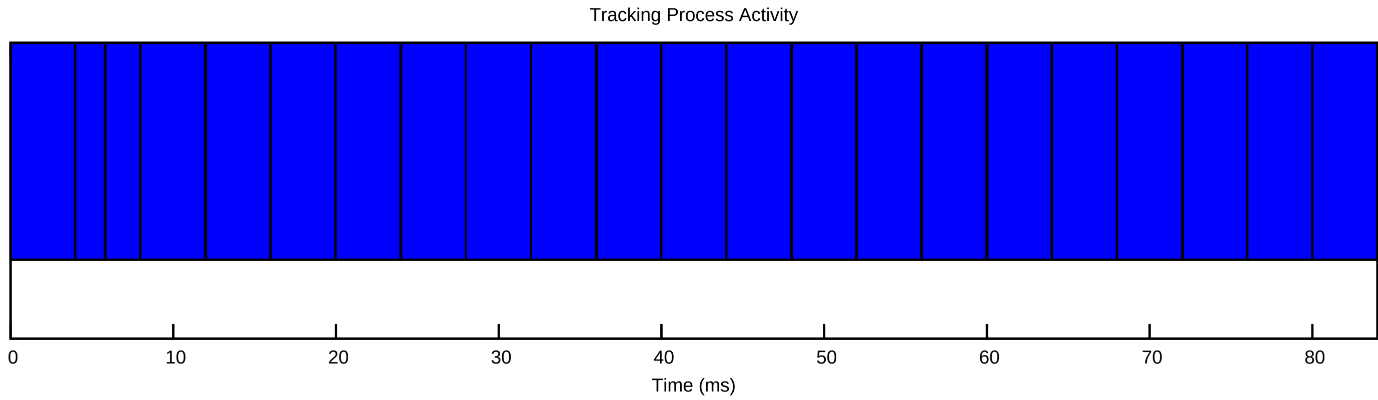

Figure 1: Active Interval(BLUE) Inactive Interval(RED)

their own active/inactive intervals, and find the duration where one child process is scheduled off the CPU and another child process takes over.

2.2 METHOD

In the benchmark program part_a2, I start the counter in the parent process, then spawn off a number of child process which begins measuring their own active/inactive periods in the same way as partA1. The intervals collected is processed in scripts/run_experiment_A2 to extract the context switch periods as well as plot the activity of each processes in part_a2.eps.

2.3 CONCLUSION

As you can see from graphs 2, the length of time each process gets to run before it is forced to switch to another process varies based upon the number of process contending for the CPU. With two child processes there seems to be about 10 context switches within as span of 20 ms each active process lasting almost 3 ms each, however Child 0 seems to be prioritized at certain intervals. I believe this version of Linux is also running a CFS (Completely Fair Scheduler) however it doesn't seem to give each process a fair share of reasources after an interval of about 20 ms, this could be due to the specific Ubuntu OS.

Based on the output of run_experiment_A2, the average context switch time is 2.537 ms, this was due to the irregular periods when child 0 was prioritized for certain consistent periods which was explained above.

3 Part B: Measuring Numa Effects

3.1 Hardware Information

This experiement was done on the wolf server.

CPU name	AMD Opteron Processor 6348
CPU speed	2.80GHz
L1 Cache Size	64 KB
L2 Cache Size	2048 KB
L3 Cache Size	6144 KB
Memory	64408 MB

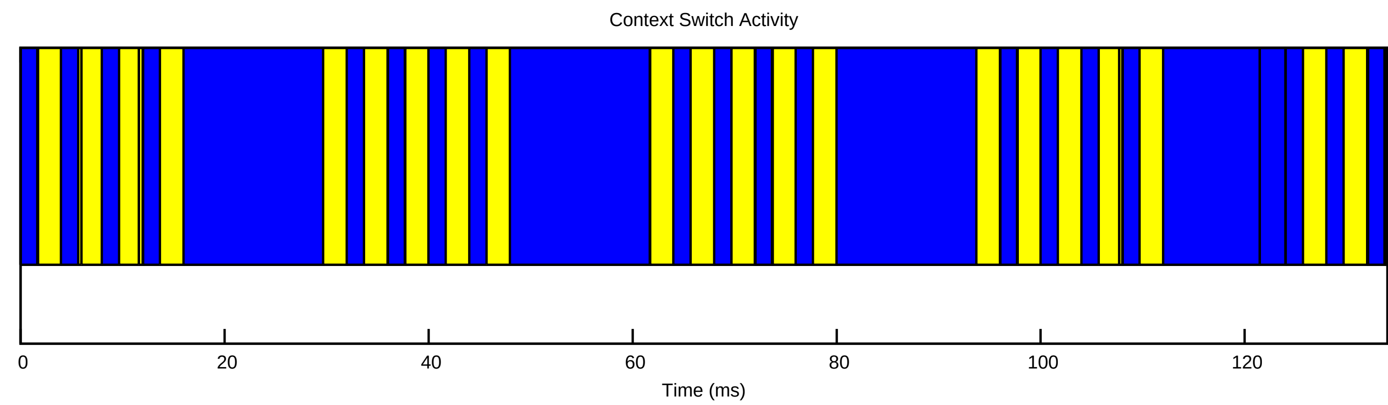

Figure 2: Process activity with two child processes

3.2 Initialization

Using the ïumactl –hardwarec̈ommand, I found that the machine (wolf.cdf) contains 4 nodes
These nodes have about 12 CPU cores each, represented by the data collected below:

Node	CPU
0	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
2	12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23
4	24, 25, 26, 27, 28, 29, 30 ,31, 32, 33, 34, 35
6	36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47

With Node distances:

Node Distances				
Node	0	2	4	6
0	10	16	16	16
2	16	10	16	16
4	16	16	10	16
6	16	16	16	10

3.3 Test Script

The results of our script can be found in part_b_output.txt and the script itself is run_experiment_B

3.4 Observations

From my results file mentioned above by looking at some of the operations from the CPUs below:

CPU	Copy	Scale	Add	Triad
0	5842.4	5819.0	6559.7	6540.0
2	5853.9	5785.0	6429.5	6538.6
14	2970.5	2970.8	3014.5	3014.3
15	2974.2	2971.6	3017.1	3017.2
25	3694.6	3696.8	3939.3	3946.5
26	3687.4	3692.0	3933.4	3951.6
37	4004.9	4011.4	4228.2	4235.7
38	3909.0	3944.8	4136.8	4158.8

The results show that CPU 0 and 2 are the ones with the highest bandwith. This is expected since memory was binded specifically to node 0, and access to local memory will always be faster then remote access. Also based on the table above it seems to be slightly inconsistant with the node distances shown above since CPUs 14 and 15 seem to have the lowest bandwidth, compared to the other 4 CPUs (25, 26, 37, 38) which have relativly closer bandwidth values.

4 Appendix: Benchmark Tool Usage

4.1 run_experiment_A

```
Usage: run_experiment_A <n_intervals>
n_intervals = number of inactive periods for each process to record. Must be >= 0
```

4.2 run_experiment_A2

```
Usage: run_experiment_A2 <n_intervals>
n_intervals = number of inactive periods for each process to record.
```

4.3 run_experiment_B

```
Usage: run_experiment_B
Outputs the result of running STREAM benchmark while binding memory on NODE 0, and varyi
```