Computer Architecture

ECGR 5181

Spring 2013

Project 3

Akshay Deshpande

800807594

adeshpa8@uncc.edu

**Statement:**

Write a C program for FFT. Each element of the matrix is a 32-bit integer. Ensure that your program can find FT for variable input size n. Measure response time and CPU execution time. Consider different

input sizes. What do you conclude from these measurements? Present your results in an appropriate form.

**Observations:**

FFT program has been run for different input sizes. To compile my C program on Linux terminal I used following command: ***gcc –o exe project3.c –lm.*** To measure response time and CPU execution time I have used 'time' command on my Linux machine. This command gives out three measurements-

* **Real** is wall clock time - time from start to finish of the call. This is all elapsed time including time slices used by other processes and time the process spends blocked (for example if it is waiting for I/O to complete).
* **User** is the amount of CPU time spent in user-mode code (outside the kernel) *within* the process. This is only actual CPU time used in executing the process. Other processes and time the process spends blocked do not count towards this figure.
* **Sys** is the amount of CPU time spent in the kernel within the process. This means executing CPU time spent in system calls *within the kernel,* as opposed to library code, which is still running in user-space. Like 'user', this is only CPU time used by the process. See below for a brief description of kernel mode (also known as 'supervisor' mode) and the system call mechanism.

User + Sys will tell you how much actual CPU time your process used. Note that this is across all CPUs, so if the process has multiple threads it could potentially exceed the wall clock time reported by Real. Note that in the output these figures include the User and Sys time of all child processes as well, although the underlying system calls return the statistics for the process and its children separately.

**Readings:**

$ time ./exe 128

real 0m0.002s  
user 0m0.000s  
sys 0m0.000s

$ time ./exe 256

real 0m0.002s  
user 0m0.000s  
sys 0m0.000s

$ time ./exe 512

real 0m0.002s  
user 0m0.000s  
sys 0m0.000s

$ time ./exe 1024

real 0m0.002s  
user 0m0.000s  
sys 0m0.000s

$ time ./exe 2048

real 0m0.002s  
user 0m0.000s  
sys 0m0.000s

$ time ./exe 4096

real 0m0.003s  
user 0m0.004s  
sys 0m0.000s

$ time ./exe 8192

real 0m0.004s  
user 0m0.004s  
sys 0m0.000s

$ time ./exe 16384

real 0m0.007s  
user 0m0.004s  
sys 0m0.004s

$ time ./exe 32768

real 0m0.012s  
user 0m0.014s  
sys 0m0.000s

$ time ./exe 65536

real 0m0.019s  
user 0m0.012s  
sys 0m0.004s

$ time ./exe 131072

real 0m0.038s  
user 0m0.024s  
sys 0m0.004s

$ time ./exe 262144

real 0m0.104s  
user 0m0.096s  
sys 0m0.004s

$ time ./exe 524288

real 0m0.220s  
user 0m0.208s  
sys 0m0.008s

**Result Table:**

|  |  |  |
| --- | --- | --- |
| **Data Size (N)** | **Response time (milliseconds)** | **CPU Execution time (milliseconds)** |
| 128 | 2 | 0 |
| 256 | 2 | 0 |
| 512 | 2 | 0 |
| 1024 | 2 | 0 |
| 2048 | 2 | 0 |
| 4096 | 3 | 4 |
| 8192 | 4 | 4 |
| 16384 | 7 | 8 |
| 32768 | 12 | 14 |
| 65536 | 19 | 16 |
| 131072 | 38 | 28 |
| 262144 | 104 | 100 |
| 524288 | 220 | 216 |

**Graphs:**

**Conclusion:**

* As the data size (N) doubles the Response time increases (approximately doubles).
* CPU execution time increases too as the data size increases but slower than the increase in response time.
* If the user is allowed to provide the value of data size(N) via scanf command, then the response time gets added with the time needed to input a value by user. In my FFT code, I have used an argument from Linux command line by which I can avoid use of scanf command.

Hence, the response time is quite less.

* Interpretation of the readings from 'time' command
* real < user: The process is CPU bound and takes advantage of parallel execution on multiple cores/CPUs.
* real ≈ user: The process is CPU bound and takes no advantage of parallel execution.
* real > user: The process is I/O bound. Execution on multiple cores would be of little to no advantage.