University of Texas at Dallas

Project2: Multi-Threading program testing report

Sujin Lee, Saghar Abdi

SE3377.005

Prof. Sridhar Alagar

20 Apr. 2024

Time table

- **Regular Testing Result Data**

Testing case	Time takes for number of threads (s)						
(size)	**1**	**4**	**16**	**32**	**64**	**128**	**256**
P2_tc0	1.904433	0.551279	0.178505	0.109347	0.110841	0.087701	0.10978
P2_tc1	3.774598	1.063416	0.30639	0.220286	0.178694	0.152689	0.176705
P2_tc2	7.440606	2.087063	0.588301	0.367002	0.302278	0.295152	0.293574
P2_tc3	15.87356	4.20025	1.102479	0.722275	0.580216	0.568717	0.587663
P2_tc4	29.81414	8.686733	2.367861	1.401907	1.104108	1.086232	1.074762
avg	11.7614672	3.3177482	0.9087072	0.5641634	0.4552274	0.4380982	0.4484968

**tested in CS3 machine & data is average of 5 tests for each number of threads.*

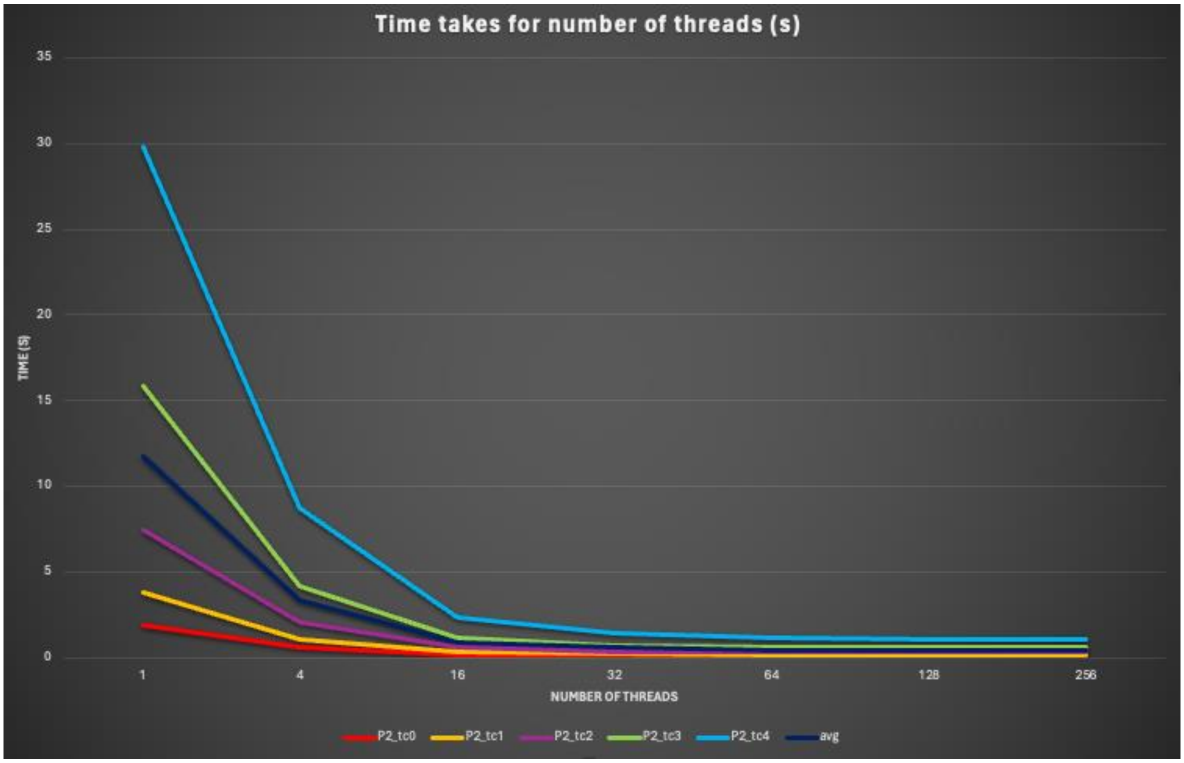
- **Speed Up Testing Result Data**

Testing case	Speedup = t(single thread) / Time takes for number of threads (s)						
(size)	**1**	**4**	**16**	**32**	**64**	**128**	**256**
P2_tc0	1	3.454572	10.6687936	17.4164175	17.1816656	21.715066	17.3477227
P2_tc1	1	3.54950273	12.3195861	17.1349882	21.1232498	24.7208247	21.3610141
P2_tc2	1	3.56510848	12.6476175	20.2740203	24.6151093	25.209404	25.3449079
P2_tc3	1	3.77919291	14.3980566	21.9771624	27.3580098	27.9111667	27.0113228
P2_tc4	1	3.43214693	12.5911715	21.2668487	27.0029236	27.4473078	27.740229
avg	1	3.54501502	12.9430769	20.8476254	25.8364659	26.8466458	26.2241942

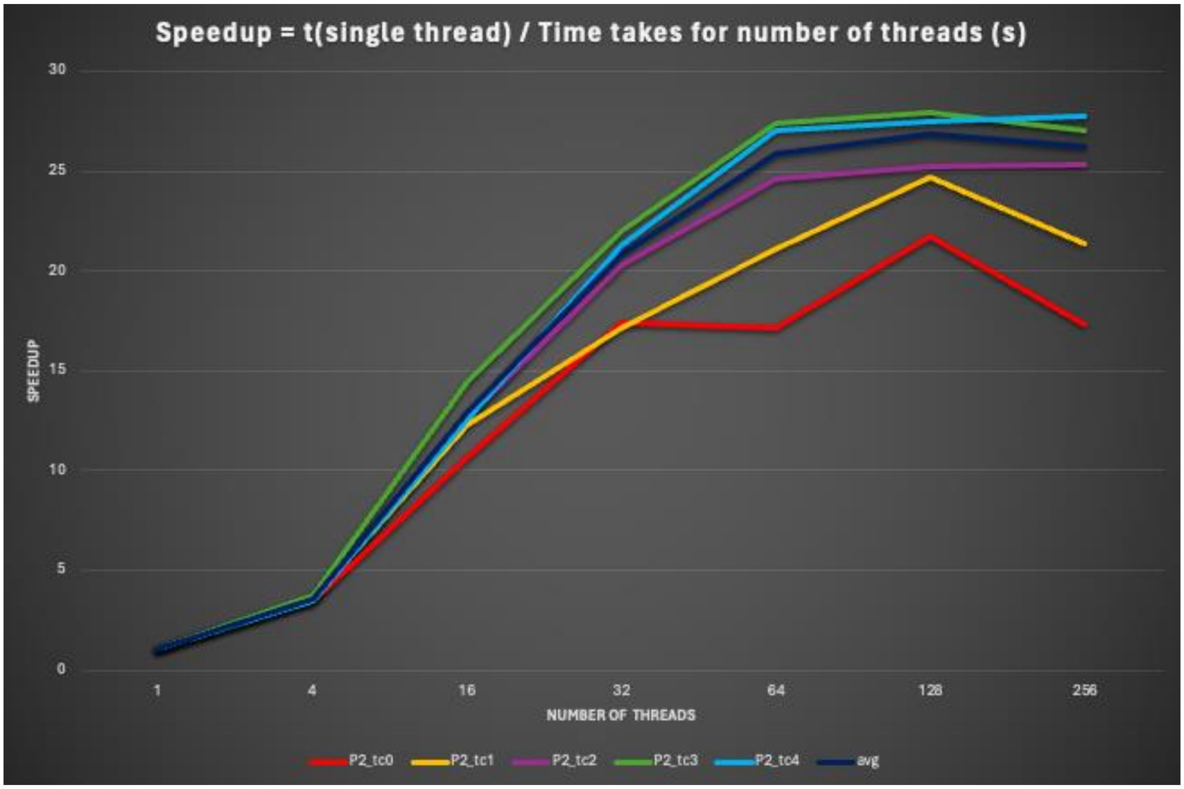
**tested in CS3 machine & data is average of 5 tests for each number of threads.*

Visual Aid

- Regular Testing Result Graph



- Speed Up Testing Result Data



Sujin Lee, Saghar Abdi

Testing Result (Conclusion)

Using various test cases provided, this program records the time taken to do a specified task on a single thread and multiple threads. These values are then calculated and compared to evaluate the speedup and efficiency. The test was compiled and run on a CS3 machine with 48 CPUs, and each can run two parallel threads. As we know, the speedup is calculated by the time a single thread takes to do a task divided by the amount of time taken to do the same task with multiple threads. The ideal expectation is that as we increase the number of threads, we see a decrease in the time taken to complete a task, and an increase in speedup, which is a linear speedup. However, we need to consider other factors that could affect the speedup pattern, such as thread management overhead and CPU limits.

Looking at the regular testing result data, it is clear that the time taken by threads to complete a task decreases up to 32, and in some cases, 64 threads. Beyond that point, the decrease slows down and is very insignificant. The slope of each test case in this table gives us a clear vision of this process.

Looking at the speedup data, we can observe that as we increase the number of threads (up to 32), there is a significant increase in the speedup. However, starting from 64 threads, we notice that there is little to no increase, and in some test cases, there’s a decrease in the speedup. The reduction is more significant for test cases one and two, as seen in the graphs.

From the above information, we can conclude that – for this program – as we increase the number of threads, our expectations are met up to approximately 64 threads. Beyond that point, we see a nonlinear increase and decrease (depending on the type of information we are looking for). The ideal number of threads based on this program is about 64, which may vary based on input, overhead management, and other factors that might affect the program.

Sujin Lee, Saghar Abdi

Functional and Non-Functional Requirements

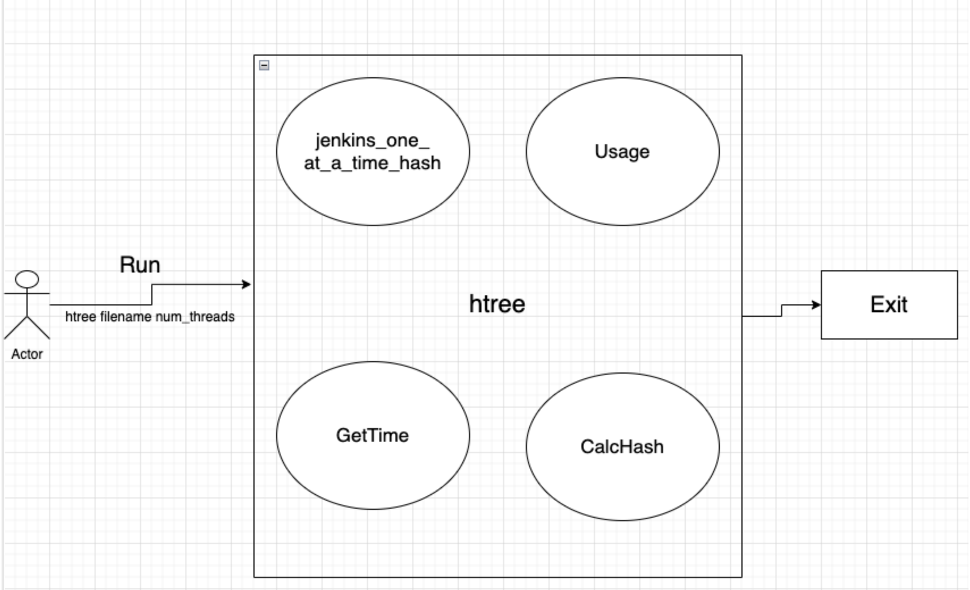
FR	Description	Priority
FR1	The Customized Multi Thread Program must use pthread(POSIX) to utilize multithreading to improve performance.	1
FR2	The Customized Multi Thread Program must use mmap() to allocate and read memory mapping to efficiently read the file	2
FR3	The Customized Multi Thread Program must handle errors correctly and print error message to show related error to the user.	5
FR4	The Customized Multi Thread Program must calculate the hash value of a given file using the Jenkins one-at-a-time hash algorithm.	3
FR5	The Customized Multi Thread Program must time while program is running and print it when program ends (to compare how numThread affects to the run time).	4
FR6	The Customized Multi Thread Program must follow the output format provided in hash_tc# file	6
NFR1	The Customized Multi Thread Program must be accountable to the users in order to ensure that any user who wants to run this program can see an appropriate error message if an error occurs.	2 (FR3)

Method definition (Function Prototype)

name	parameter	return type	content
jenkins_one_at_a_time_hash	Key, Len	uint32_t	This function was provided with the project, and detailed information can be found at https://en.wikipedia.org/wiki/Jenkins_hash_function.
Usage	string	void	This function takes a strings as a parameter and is meant to execute an error message when user enters wrong command to execute the object file.
GetTime	none	double	This function implementation is from trat.c to store runtime of the program, and later we will use it to compare benefit of number of threads.

CalcHash	*arg	void	This function calculate hash value using multithreading using pthread(POSIX).
			Also, this function will serve as start routine when each thread is created.
			`int`
			`pthread_create(pthread_t* thread,`
			`const pthread_attr_t* attr,`
			`void* (*start_routine)(void*),`
			`void* arg);`

Usecase Diagram


jenkins_one_
at_a_time_hash
Usage
Run
htree filename num_threads
Actor
htree
Exit
GetTime
CalcHash