Optimizing the performance of N-Version independent programs

M.Malhotra
Department of Computing
Imperial College
London SW7 2BZ
Email: m.malhotra@doc.ic.ac.uk

Abstract - The purpose of this research is to optimize the performance of an N-Version computer system by altering the performance of processors. The paper proposes a critical computer that can execute independent programs consisting of three versions and a voter. Using a simulation tool and four sets of differing tasks it determines the effect on task execution time of altering the speed of individual processors. Results show that increasing processor speed does not result in all tasks taking less time to execute, as a matter of fact some tasks take longer to execute. Similarly reducing processor speed results in some tasks taking less time to execute. Results show that is possible to reduce power consumption by reducing the speed of a processor that executes voters. It is also possible to reduce power consumption and the execution time of particular tasks by reducing the speed of a processor executing a version. Results show it is possible to predict if a task will take longer or less time to execute due to a change in processor speed, but it is not possible to quantify by how much.

I Introduction

Computer systems are increasingly being used to perform critical tasks. At the same time processors are becoming faster and can perform more tasks in the same time period as older processors. Therefore critical computer systems can now be used to control a larger number of instruments and devices rather than just a few. To control a large number of instruments a computer system will have to perform a large number of independent tasks. The purpose of this research is to investigate techniques of optimizing the execution of a proposed critical computer system that performs independent tasks.

Microprocessors are usually available in a number of different packages, clock frequencies and features. The cost and performance of these processors varies. Normally higher performance microprocessors cost more and consume more power, while lower performance microprocessors cost less and consume less power. One technique of reducing cost and power consumption is to use lower performance microprocessors or to alter the clock frequency of a processor. Reducing the clock frequency of a microprocessor generally results in lower power consumption. The technique has been used on a number of spacecrafts the author has worked on, such as the Cassini Magnetometer computer [1]. The time taken to execute tasks in a critical computer system is important and is often a measure of how frequently an instrument is sampled or monitored. The purpose of this research is to investigate the effect of altering processor speed on the execution time of tasks in a particular architecture.

Past and existing critical computer systems have used a technique called N-Version programming to tolerate faults [2][3][4][5]. N-Version programming consists of one or more versions of the same program and a voter. The voter inputs results from the different versions and then outputs a result based on a majority vote. The technique of N-Version programming relies on differences between versions to tolerate faults. It is possible to create the differing versions using a number of techniques, such as by writing the same program in different languages.

The technique of N-Version programming has been the subject of a number of independent studies. However, the technique requires further investigation because most critical computer systems use redundancy to tolerate faults. To reduce the risk of common faults designers often introduce differences between redundant units. Therefore, there will be differences between tasks executed on different redundant units.

II. Proposed Architecture

In the experiment that follows each task will consist of three differing versions and a voter. Figure 1 shows the proposed computer architecture that will execute tasks consisting of three versions and a voter [6].

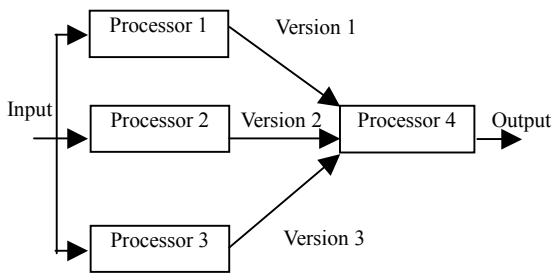


Figure 1.Architecture of Three-Version system

In Figure 1 Processor 1, Processor 2 and Processor 3 each receive the same input and execute version 1, version 2 and version 3 of each task respectively. These three processors output a result to Processor 4. Processor 4 executes the voter

of each task and outputs a result based on a majority vote. The computer system will execute a maximum of one thousand independent tasks.

The processors will execute independent tasks in pre-emptive multi-tasking environment. The four processors execute independent tasks in a round-robin manner allocating each task equal processing time. As soon as a task completes it is rescheduled. Therefore, tasks are being continuously executed and even if one task takes too long it will not prevent the execution of other tasks.

III Task Execution Time

Four sets of tasks, with different execution times, have been created for purpose of experiment [7]. The execution time of versions were based on those performed by Cassini Magnetometer computer [1][7]. Simultaneously, experiments performed by Bagley have also shown that execution times of versions will differ [9]. These four sets of tasks will be referred to as Taskset 1, Taskset 2, Taskset 3 and Taskset 4. Each task set contains execution time of three different versions for one thousand tasks. A simulation tool called NVP has been created to simulate the execution of tasks and determine task execution time. The NVP tool inputs task execution times from Taskset 1, Taskset 2, Taskset 3 or Taskset 4 and then perform a simulation.

The NVP tool performs simulations with logarithmically increasing number of tasks. The NVP tool performs simulations over three task ranges. First, the tool simulates execution of one to ten tasks increasing in steps of one task. Second, the tool simulates execution of ten to one hundred tasks increasing in steps of ten tasks. Finally, the tool simulates execution of one hundred to one thousand tasks increasing in steps of one hundred tasks. The aim is to observe any trends over the three task ranges and to study the effect of increasing number of tasks. Therefore, it is necessary to perform several simulations with different number of tasks rather than only for a fixed number of tasks.

In each simulation the NVP tool records the time taken to execute the slowest task (i.e. the task that takes longest to execute), the fastest task (i.e. task that takes least amount of time to execute) and a particular task, which in this case is task one. The time taken to execute the fastest task is referred to as Min Task, while the time taken to execute the slowest task is referred to as Max Task. The execution time of all tasks will be between Max Task and Min Task and this enables a study on all the tasks in a simulation.

Over each task range the gradient of a best-fit line is calculated. Figure 2 shows the execution time of task one with increasing number of tasks and the line of best fit. The gradient of best-fit line is a measure over ten different simulations of varying number of tasks, rather than for a fixed number of tasks. Once the gradient of best-fit line is determined the difference between estimated time taken (i.e. gradient of line of best-fit multiplied by number of tasks) and simulated time taken is recorded for each simulation. The difference between expected and measured execution time is used to calculate the standard deviation. This value gives an indication of degree of best-fit line and level of noise in the system.

Figure 2 Time taken to execute Task One

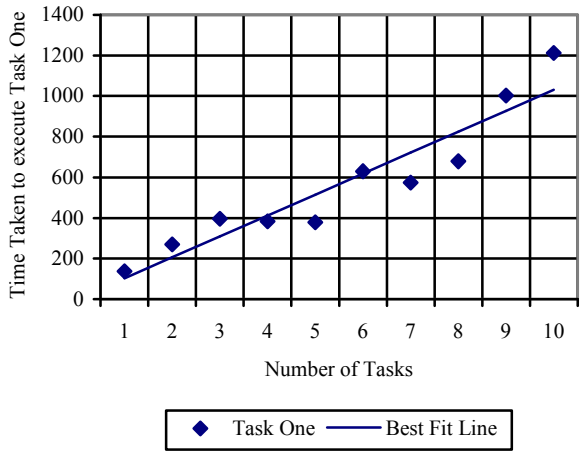


Table 1 shows the result from simulations where the speed of all processors is identical. The column Data is the task set used in a simulation, while No. of tasks is the task range. In each simulation the time taken to execute slowest task, fastest task and task one is recorded. In Table 1 Min-Grad is the gradient of line of best fit for the fastest task. Similarly, Max-Grad and Task 1-Grad are the gradient of best-fit line for slowest task and task one respectively. In Table 1 Min-Stdev, Max-Stdev and Task 1-Stdev are the standard deviation of data between line of best fit and measured execution time.

Table 1 Simulation result

Taskset	No. of Tasks	Min Grad	Min Stdev	Max Grad	Max Stdev	Task1 Grad	Task1 Stdev
Taskset1	1-10	53	45	123	35	103	59
	10-100	29	112	117	154	73	178
	100-1000	16	553	106	1316	81	1922
Taskset2	1-10	48	47.7	98	12	97	21
	10-100	31	107	103	160	78	189
	100-1000	14	1038	103	928	78	676
Taskset3	1-10	47	49	137	17	105	36
	10-100	31	53	119	149	93	103
	100-1000	13	1478	98	878	73	801
Taskset4	1-10	44	17	97	34	47	16
	10-100	31	68	97	199	41	127
	100-1000	6	739	98	976	44	406

IV Reducing the speed of a Processor

This experiment aims to determine the effect of reducing processor speed. In this experiment the speed of Processor 4 and Processor 1 will be reduced by fifty percent. The figure of 50% was chosen in order to make the results easy to interpret. The NVP tool will simulate this effect by increasing the effect on processor clock. So the slice time will be as before

however, the processor clock will increase by double this amount. This experiment consists of two simulations.

In the first simulation the speed of processor 4 is reduced by fifty percent. Processor 4 executes the voter of each task. If the versions are not ready for voting then the voter executes in one unit because it simply has to check if all the versions have been completed. If the versions are ready for voting then the voter executes in five units. Voting only begins after all the versions have been executed. The voter spends most of its time waiting for version to complete. Therefore, reducing the speed of processor 4 may have a negligible effect on the time it takes to execute tasks.

In the second simulation the speed of processor 1 is reduced by fifty percent. Processor 1 executes version one of each task. Therefore, reducing processor 1 speed will mean that version one of each task will take longer to execute. As a result reducing processor speed will cause tasks to take longer to execute.

V Reducing Processor 4 Speed

Figure 3 is a graph of percentage change in the gradient of Min-Grad (i.e. the fastest task in the data). This shows that Min-Grad generally changes by less than 5%. This change is within the margin of error for the simulation tool (i.e. +/-5%). The only exception to this is with Taskset 2 and over the range one hundred to one thousand tasks. Over this range Min-Grad decreases by 7.7%. This decrease is accompanied with a fifty percent decrease in standard deviation. The decrease is not typical of the other tasks sets and may be attributed to noise. So this change may be attributed largely to noise. Therefore, reducing the speed of processor 4 has a negligible effect on Min-Grad.

Figure 4 Percentage change in Max-Grad

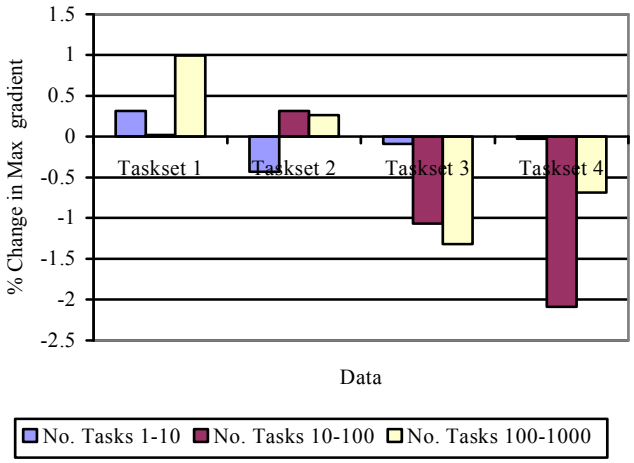


Figure 4 is a graph of percentage change in gradient of the slowest task in the data. This graph also shows that Max-Grad changes less than 5%. Therefore, this technique has negligible effect on the slowest task.

Figure 4 Percentage change in Max-Grad

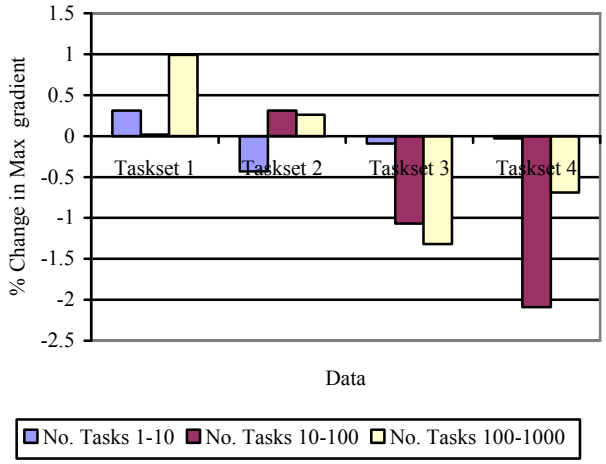
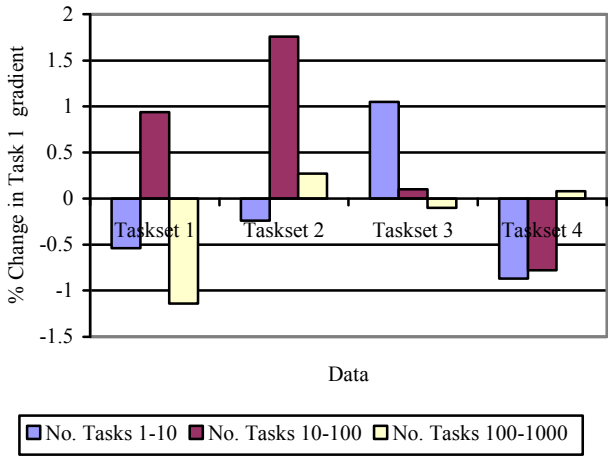


Figure 5 is a graph of the percentage change in the gradient of task one. The change in gradient of task one is less than the margin of error for the simulation tool. Therefore, this technique has a negligible effect on the execution time of tasks. This suggests that the speed of processor 4 can be reduced without increasing the time it takes to execute tasks

Figure 5 Percentage change in Task 1-Grad



VI Reducing Processor 1 Speed

Figure 6 shows the percentage decrease in Min-Grad when the speed of processor one is reduced by fifty percent. This graph shows that Min-Grad decreases between 30% and 50%. The only exception to this is with Taskset 4. The fastest task of Taskset 4 data is a special case and executes within one time slice, as a result its execution time varies with time. These results suggest that reducing processor 1 speed causes the fastest task to take less time to execute.

Figure 6 Percentage change in Min-Grad

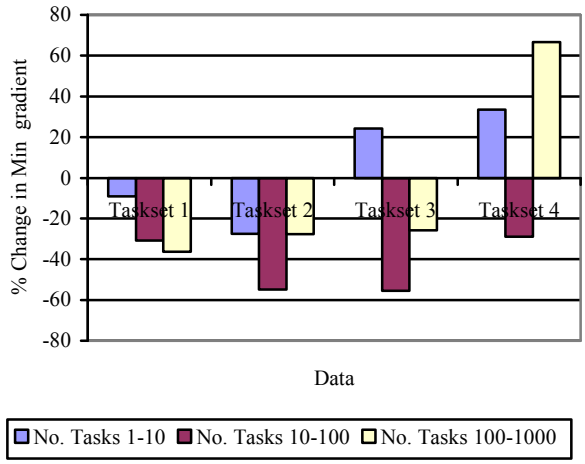
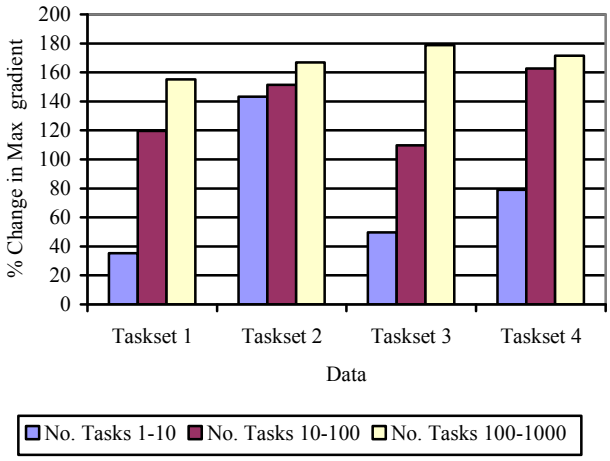


Figure 7 shows the percentage increase in Max-Grad when the speed of processor one is reduced by fifty percent. This graph shows that Max-Grad increases between 110% and 170%. The increase in gradient is common to all data types and this suggests that the result is independent of data. This means that reducing processor one speed causes the slowest task to take longer to execute.

Figure 7 Percentage change in Max-Grad



Results suggest that if the speed of processor 1 is reduced then the following occurs. Faster tasks will take less time to execute, whereas slower tasks will take longer to execute. If the speed of processor one is reduced then this will have two effects. First, version one of each task will take longer to execute because processor one is taking longer to process tasks. Second, version two and version three of each task will take less time to execute. This is because tasks with version one as the slowest will wait longer for it to complete. As a result these tasks will require less processing time from processor two and processor three. So version two and version three of each task will take less time to execute.

The execution time of tasks in a pre-emptive multitasking environment varies and is difficult to predict. This makes the study of independent N-Version programs more complicated. To assist in solving this problem each task is assigned a value that is representative of its execution time, and will be used to predict the time taken to execute a task within a group of independent tasks. This value will be referred to as the relative execution time of a task. Tasks with a similar execution time in a sequential environment should behave in a similar manner and the relative execution time will be used to compare tasks.

Figure 8 is a graph of percentage change in Task 1-Grad when the speed of processor 1 is reduced by fifty percent. The effect on Task 1-Grad is best explained in relation to the relative execution time of task one, see Table 2. This shows that version one of task one in Taskset 1 is the fastest version (i.e. 2 units) and version three is the slowest (i.e. 98 units). If the speed of processor 1 is reduced by fifty percent then version one will probably take twice as long to execute. However, this should not affect the time it takes to execute the task because version three is considerably slower. Version three of this task will take less time to execute because of the reason mentioned earlier. Therefore, this task should take less time to execute. This is shown in Figure 8 as a decrease in the gradient of task one (Taskset 1 data). The gradient of task one in Taskset 1 decreases between 65% and 84%.

Figure 8 Percentage change in Task 1-Grad

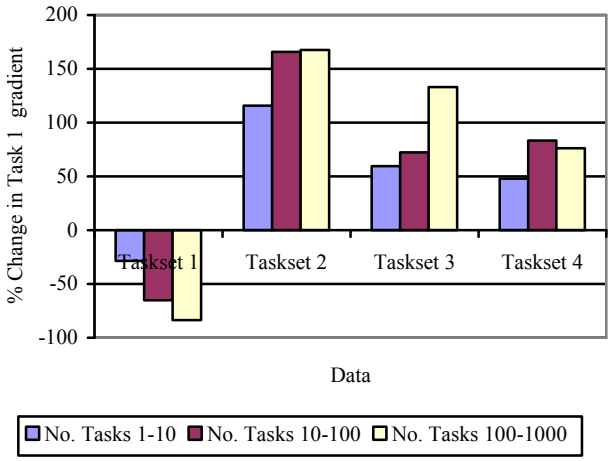


Table 2 Task one Relative execution time

Data	Execution Time of Task 1		
	Version 1	Version 2	Version 3
Taskset 1	2	54	98
Taskset 2	78	46	92
Taskset 3	64	78	44
Taskset 4	28	11	47

The gradient of task one in Taskset 2, Taskset 3 and Taskset 4 increase between 72% and 167% in Figure 8. Table

2 shows that if version one of these tasks takes twice as long to execute then this will become the slowest version of task one. For example, in Table 2 the relative execution time of task one of Taskset 2 data is 78 units, 48 units and 92 units for version one, version two and version three respectively. If version one takes twice as long to execute then its relative execution time will effectively increase to 156 units. Therefore, version one will become the slowest version. As a result this task will take longer to execute when the speed of processor one is reduced by fifty percent.

If version one of task one in Taskset 2 takes twice as long to execute compared to before then its relative execution time will be 156 units. Therefore the slowest execution time of this task will increase from 92 to 156, this is an increase of 70%. However, the results in Figure 8 indicate that the gradient of task one of Taskset 2 increases by 165%. Similarly the relative execution time of task one in Taskset 3 increases by 64%, whereas Figure 8 shows that the gradient of task one in Taskset 3 increases by 72% and 133%. The relative execution time of task one in Taskset 4 increases by 19%, whereas the gradient increases by 83% and 76%. Therefore, the change in gradient is not proportional to the change in relative execution time. This means that it is difficult predict how much faster or slower a task will execute without detailed simulation.

VI Increasing the speed of a processor

Figure 9 shows percentage change in Min-Grad when the speed of processor one is increased fifty percent. The graph shows that Min-Grad decreases when processor one speed is increased. This means that the fastest task is now taking less time to execute. The percentage decrease in gradient is different for each data set, suggesting that the decrease in gradient is dependent on data.

Figure 9 Percentage change in Min-Grad

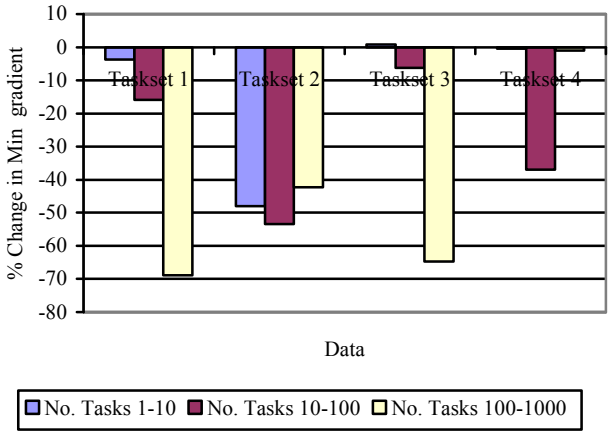
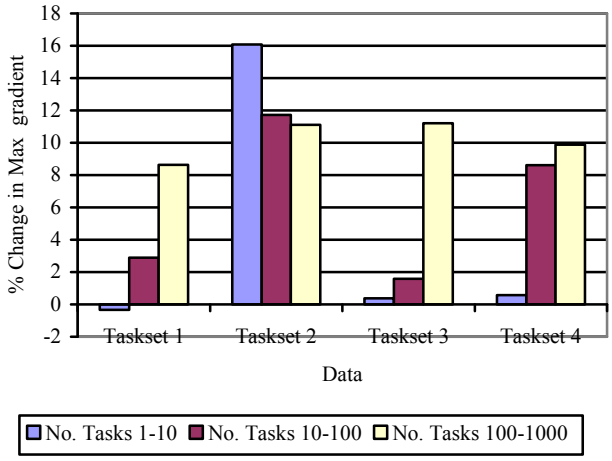


Figure 10 is a graph of percentage change in Max-Grad. This shows that Max-Grad increases between 2% and 12%

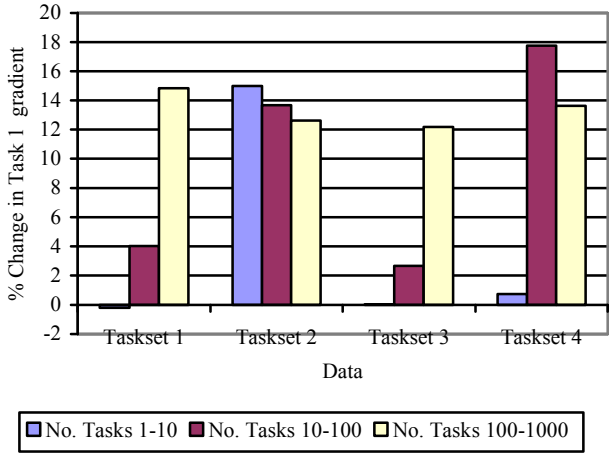
Figure 10 Percentage change in Max-Grad



when the speed of processor one is doubled. This means that in general the slowest task is now taking longer to execute.

In Taskset 1 data version one of task one has the shortest relative execution time compared to version two and version three. This means that even if version one takes less time to execute the voter will have to wait version two and version three to complete. This suggests that increasing the speed of processor one will have no effect on this task. However, the results in Figure 11 show that in general task one is taking longer to execute. Since version one is taking less time to

Figure 11 Percentage change in Task 1-Grad



execute it can only mean that version two and version three are taking longer to execute.

Results suggest that if processor one speed is doubled then the effect is as follows. First, version one of each task will take less time to execute because processor one executes version one. Second, version two and version three of each task will take longer to execute. This is because tasks whose slowest version is one are taking less time to execute.

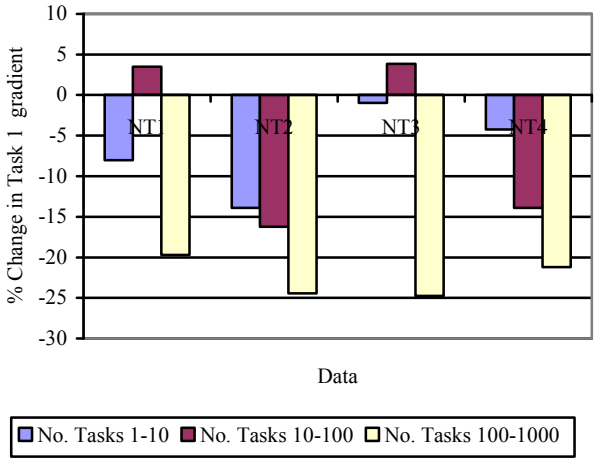
Therefore, version two and version three of these tasks are restarting more frequently and so using more time of processor two and processor three. This means that the effect of increasing processor speed is dependent on which processor executes each of the versions.

In the task sets (see Table 2) version one was never the slowest version of task one. As a result task one of all Tasksets took longer to execute, i.e. the gradient of task one increased in Figure 11. Therefore, the experiment was repeated with version one of task one as its slowest version. Task one of Taskset 1, Taskset 2, Taskset 3 and Taskset 4 data was replaced with the same task. In this task version one, version two and version three execute in 90 units, 60 units and 40 units respectively. The new data is referred to as NT1, NT2, NT3 and NT4 and is identical to Taskset 1, Taskset 2, Taskset 3 and Taskset 4 data with the exception of task one.

When executing NT1, NT2, NT3 and NT4 the voter will wait for version one to complete because it is the slowest version (i.e. 90 units). If the speed of processor one is increased fifty percent then version one should execute in half the time because the next slowest version is version two (i.e. 60 units). The relative execution time will decrease from 90 units to 60 units. Therefore task one relative execution time will decrease by 33%.

Figure 12 shows a graph of percentage change in Task 1-Grad when processor one speed is increased fifty percent. Over the one hundred to one thousand task range task one gradient decreases between 20% and 25%, this is much less than the expected 33%. This occurs because although version one is taking less time to execute, version two and version three are now taking longer to execute

Figure 12 Percentage change in Task 1-Grad



VII Conclusion

In order to design future critical computer systems it will be necessary to simulate the effect of optimizing techniques. The effect of these optimizing techniques on task execution time is not easily predictable. For example, results showed that reducing the speed of processor 4 had a negligible effect on task execution time. This means that to save power processor 4 could be replaced with a slower and less powerful processor. If at the same processor 1, processor 2 and processor 3 were replaced with faster processors, power consumption should remain as before but tasks should take less time to execute.

Altering the speed of a processor that executes one of the versions has a more significant effect. Increasing the speed of a processor that executes a version does not result in all tasks taking less time to execute, as a matter of fact results showed some tasks took longer to execute. Similarly reducing the speed of a processor that executes a version does not result in all tasks taking longer to execute. The effect of altering processor speed on a task is dependent on how long it takes to execute each of its versions and how the versions are distribution on processors. Therefore, to simultaneously effect all tasks it is necessary to increase or decrease the speed all processors executing a version rather than a single processor. Alternatively results show it is possible to reduce the speed of a processor such that certain tasks take longer to execute, while causing other tasks to take less time to execute.

If the relative execution time of each version is known then it is possible to predict whether a task will take more or less time to execute. However, results also showed that the change in relative execution time is not proportional to the change in gradient. Therefore, it is difficult to predict precisely how long a task will take to execute if the processor speed is increased or reduced with out simulation. Therefore, changing the speed of a processor may not have the desired effect on execution time of a task.

VIII References

[1] http://cassini.jpl.nasa.gov
[2] Avizienis A, N-Version approach to fault-tolerant software, IEEE Trans. Software Eng., Vol. SE-11 No. 12, Dec 1985, pp1491-1501.
[3] Briere D, Traverse P, Airbus A320/A330/A340 Electrical Flight Controls A Family of Fault Tolerant Systems, FTCS-23, Toulouse, France, June 1993, pp616-623.
[4] Yeh Y.C., Dependability of the 777 Primary Flight Control Systems, DCCA 1995, University of Illinois at Urbana-Champaign, September 27-29, pp1-13.
[5] Bretz E A, By-wire cars turn the corner, Spectrum, Vol.38,No 4, April 2001, pp68-73.
[6] Malhotra M, Investigating The Execution of Independent Multi-Version Programs, 2002 PRDC, Tsukuba, Japan, December 16-18 2002.
[7] Malhotra M, DASC 2002, Indianapolis, Indiana, U.S.A., October 2003.
[8] http://www.sp.ph.ic.ac.uk/cassini/
[9] http://dada.perl.it/shootout/