CSO-ALGORITHMS course

Multiprocessor Scheduling algorithm for maximum CPU utilization.

ABSTRACT:

Scheduling processes efficiently in a multiprocessor environment using the best optimization algorithm is very much important to maximize the CPU utilization. Multiprocessor scheduling can be done using several algorithms like binary search, dynamic programming, average distribution of tasks, etc. But we concentrated on the idle cycles generated at the synchronization points. We used the subset sum algorithm which helps in freeing up a few CPUs, which can then be used for other processes. Thereby decreasing the waiting time of other processes. We concentrated on reducing the idle cycles generated at the synchronization points.

INTRODUCTION

In multiprocessor computing, unlike a single CPU, the operating system has access to more than one processor and sometimes even several nodes. In single CPU systems, process scheduling is done by the operating system using several algorithms like FCFS, Round-robin, SJF etc.. All of them aim towards decreasing the waiting time of the processes that enter the waiting queue.

In embedded systems, it is common to use more than one processor. More than one process running at a time for a few applications have issues like data inconsistency, interdependence, dependence on the same resources or their instances. And if processes belonging to the same application are distributed on different nodes or CPUs, inter-communication required will further consume more time and cause traffic. So we focus on reducing the idle CPU cycles and pack the tasks efficiently to free up the CPUs.

RELATED WORK

Multiprocessor scheduling is more complex compared to single processor scheduling because of the possibility of heterogeneity between the processors. When the processors are identical i.e. HOMOGENEOUS, in terms of their functionality, any processor available can be used to run any process in the queue. But when the processes to be executed have different loads, then task packing becomes a deciding factor for the maximum CPU utilization.

One of the ways to deal with this without chaos is by Asymmetric Multiprocessing wherein the master server is responsible for scheduling decisions, The other way is symmetric multiprocessing, in which the CPUs follow self scheduling .

Several algorithms are being come up with, for achieving best performance through constraint optimization problems. One such is Genetic algorithm to solve the scheduling problem of multiprocessors that minimizes the makespan [1]. Most currently existing optimal real-time multiprocessor scheduling algorithms follow the fairness rule, in which all tasks are forced to make progress in their executions proportional to their utilization, to ensure the optimality of the algorithm[2].

Further improvements to this is an efficient real-time multiprocessor scheduling algorithm in which the fairness rule is completely relaxed and a semi-greedy algorithm is introduced. [3]. Priority-Driven Scheduling algorithms are also being developed for the real time.[4]

Focusing on non-preemptive tasks, recent work used bin-packing strategies such as first fit or best-fit and non-preemptive Earliest Deadline First (npEDF) scheduling method to apply to each processor for better results.[5]

OUR APPROACH

To improve the throughput of the system, we implemented a task scheduling algorithm which uses a subset algorithm to divide the processes among the CPUs.

Process with a maximum amount of load is considered to be the maximum boundary and the remaining processes whose loads sum up to this maximum boundary are scheduled into a single processor. Thereby freeing idle CPU cycles.

$$\text{maximum boundary} = \sum t[i] \leq t\_max \text{ where } i \text{ goes from } 0 \text{ to } k.$$

The task reallocation solution for the packing is carried out by the Subset Sum algorithm, a particular case of the Knapsack problem[6]

ALGORITHM

In order to free CPUs, we will try to oversubscribe one or more CPUs with two or more tasks, so that original CPU(s) where these tasks were running can be free. we define t[i] as the running time of task i between two synchronization points

then,

$$t\_max \geq \sum t[i] \text{ , } i \geq 2 \text{ and } i \text{ goes to } n\text{-}1.$$

Given a set of n bins, the CPUs (all of the same size, which in turn corresponds to the runtime of the slowest task in a parallel session t_max) as well as a set of n items, the Tasks (of size t[i] , where all t[i] is smaller or equal to the size of the bins t_max), the algorithm determines a distribution of the items into the bins so that the number of still empty bins is as large as possible. A necessity for the algorithm to be successful in finding a distribution with one or more still empty bins is that two or more items fit in one bin. As stated in Eq. (4), the size of at least two items must be smaller than the size of the bin.[7]

IMPLEMENTATION DETAILS

We implemented our code in C language and used Quicksort to sort the data structure which holds the load of the processes to be scheduled. And further used the subset algorithm to find subsets which sum up to the maximum boundary and scheduled them in different CPUs accordingly.

The support function which calls the subsetsum function is implemented as:

```
void supportfunction(int P[],int L[],int m,int n)
{
	int maxsum=P[n-1];
	L[0]=maxsum;
	int index=1;
	int idea[n];
	for(int i=0;i<n;i++)
		idea[i]=0;
	idea[n-1]=1;
	subsetsum(P,L,m,n,n-1,index,idea,maxsum);
}
```

The combinationutil function is used to find subsets that sum up to the maximum boundary and update the scheduling:-

```
void subsetsum(int P[],int L[],int m,int n,int r,int index,int idea[],int maxsum)
{
	if(r-1==0)
		return;

	int data[r];
	for(int i=0;i<r;i++)
	{
		data[i]=-1;
	}
	sum(P,L,index,data,r,maxsum);
}
```

The sum function is used to find maximum numbers of tasks which can be given to a single CPU:

```
int sum(int P[],int L[],int index,int data[],int r,int
maxsum)
{
	int n=r,c=0;
	while(c!=n)
	{
		for(int i=r-1;i>=0;i--)
		{
			if(data[i]!=0)
			{
			int s=P[i];
			c++;
			for(int j=i-1;j>=0;j--)
			{
				if(s+P[j]<=maxsum && data[j]!=0)
				{
					s+=P[j];
					data[j]=0;
					c++;
				}
			}
			if(s<=maxsum)
			{
				L[index++]=s;
				data[i]=0;
			}
			}
		}
	}
	printf("%d \n",index);
}
```

RESULTS AND EVALUATION

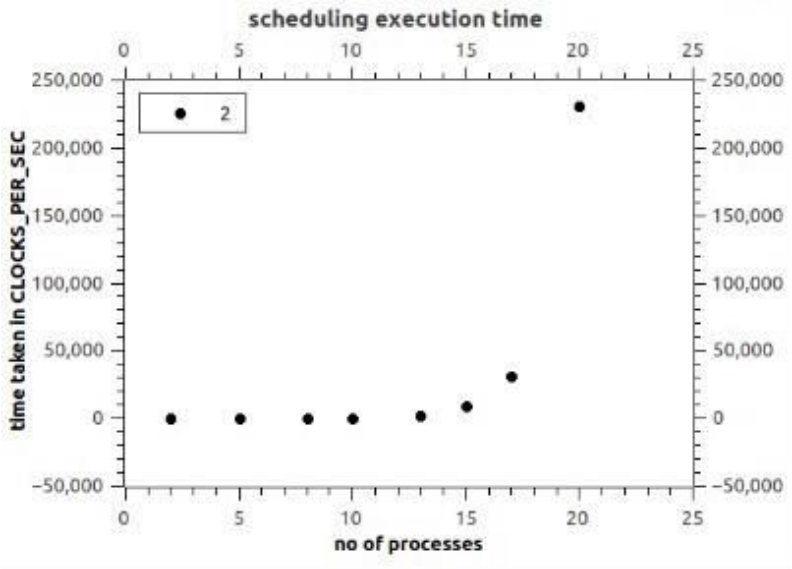
We evaluated our application on Qtiplot to obtain the execution times per clock cycle and behaviour of algorithm with respect to different number of processes

EXECUTION FRAMEWORK

Data is collected by running the code on a system based on Intel(R) Core(TM) processors, on a linux operating system. With CPU @ 2.20 GHZ with 1TB main memory. This collected data is used for performance analysis on the Qtiplot application.

EVALUATION RESULTS:

Very sharp increase in the execution time is observed when the number of processes approaches 20. This behaviour can be accounted to the complexity of the subset sum algorithm on which our algorithm is based.



CONCLUSION AND FUTURE WORK

In this work, we provide our scheduling algorithm for the distribution of tasks as per their loads on the multiple CPUs. Subset sum problem we have used here is NP-complete, so the solution that can be obtained in polynomial time may not be an optimal one. In future works, we may come up with the optimal solution to apply to our scheduling mechanism and get the benefits. Our code performs scheduling for multiprocessors. For better simulations of the algorithm, development of the code for different nodes is to be done.

ACKNOWLEDGEMENT

We would like to thank our Prof Dr. Ravi Shankar who gave us the opportunity to do this wonderful project. We would like to express thanks of gratitude to our tutor Navin Mani Upadhyay for guidance, suggestions, instructions and support in various phases of completion of our Project. We also acknowledge the support of our classmates who have helped us to improve our project.

REFERENCES

[1]E. S. H. Hou, N. Ansari and Hong Ren, "A genetic algorithm for multiprocessor scheduling," in IEEE Transactions on Parallel and Distributed Systems, vol. 5, no. 2, pp. 113-120, Feb. 1994.

[2] Miklos Ajtai, James Aspnes, Moni Naor, Yuval Rabani, Leonard J Schulman, Orli Waarts, "Fairness in Scheduling, Journal of Algorithms",Volume 29, Issue 2,1998,

[3]Alhussian, Hitham & Zakaria, Nordin & Hussin, Fawnizu Azmadi. (2014). An Efficient Real-Time Multiprocessor Scheduling Algorithm. Journal of Convergence Information Technology.

[4] Goossens, J., Funk, S. & Baruah, S. Priority-Driven Scheduling of Periodic Task Systems on Multiprocessors. *Real-Time Systems* 25, 187–205 (2003). https://doi.org/10.1023/A:1025120124771

[5]A hierarchical multiprocessor scheduling framework for DSP applications," Proceedings of the IEEE Asilomar Conference on Signals, Systems and Computers, 122-126 vol.1, Pacific Grove, California, November, 1995.

[6]E. Horowitz, S. Sahni, Computing partitions with applications to the

knapsack problem, J. ACM 21 (2) (1974) 277–292

[7] Gladys Utrera, Montse Farreras, Jordi Fornes. “Task Packing: Efficient task scheduling in unbalanced parallel programs to maximize CPU utilization”, Journal of Parallel and Distributed Computing 134 (2019) 37–49.