Final Project Proposal
Allocation of Tasks on Real-Time Multiprocessor Systems using Genetic Algorithms

Oscar Mondragon, and Amanda Minnich
Computer Science Department
University of New Mexico
{omondrag, aminnich}@cs.unm.edu

April 3, 2013

1 Background

Allocation of tasks on real-time multiprocessors systems is a problem with a high computational complexity solution [5]. There are two approaches to solving this problem: the partitioning approach and the global approach. In the partitioning strategy all the instances of a task are allocated to a single processor, while in the global strategy any instance of a task can migrate among processors when required [3]. We will be using the partitioning approach for this project. The multiprocessor maximum utilization bound depends on the allocation algorithm used. Since this allocation problem is NP hard, some approximate solutions have been proposed which include the Worst Fit (WF), First Fit (FF), Next Fit (NF), and Best Fit (BF) heuristics [6].

Allocation is required before tasks can be scheduled in a real time multiprocessor system. Solutions such as Rate Monotonic Scheduling (RM) and Earliest Deadline First Scheduling (EDF) [1] have been proposed. The optimal allocation of tasks varies depending on what kind of scheduling algorithm one wants to use. For this project, we want to use the EDF scheduler. This scheduler uses dynamically calculated priorities as scheduling criteria to choose what task will be scheduled next. These priority values are calculated based on the deadlines of the tasks that are in the run queue. Tasks with earliest deadline will receive higher priorities. A period (T) and a slice (C) are associated to each task. The period is defined as the amount of time the task may receive a CPU allocation, and the slice is the minimum amount of time received for the task during each period [4]. The portion of CPU used for a task is $\frac{C}{T}$. Therefore, the utilization factor of a CPU is given by:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

Where n is the number of tasks allocated on a CPU and $U \leq 1$.

If a maximum utilization (U_{MAX}) for the CPU is given, the equation becomes,

$$U = \sum_{n}^{i=1} \frac{C_i}{T_i} \leq U_{MAX} \leq 1$$

Since we are planning on using this scheduler, our allocation must always satisfy this formula.

Allocation and scheduling have a direct application to Virtual Machine Monitors (VMMs), where virtual cores need to be allocated and scheduled over physical cores. In this problem a variable called the time dilation factor (TDF) also affects allocation. Time dilation allows the perceived availability of resources of a virtual machine to be changed by altering the real time by a factor, called the time dilation factor. This provides for the slowing down or speeding up the passage of time detected by a guest operating system [2]. Slowing down the system has the effect of making the external world appear sped up. For example, if you have a TDF of 10, for 10 seconds of real time you have 1 second in the guest system, so the guest system recieves more events from the external world per unit time. The TDF is defined as $\frac{realtime}{virtualtime}$. During allocation, the following formulas relating to the TDF must always be satisfied:

$$\frac{1}{TDF} \cdot \sum_{n}^{i=1} \frac{C_i}{T_i} \leq U_{MAX} \text{ and } \frac{\sum_{i=1}^{n} F_{VCPUi}}{F_{PCPU}} \leq TDF$$

Where F_{PCPU} is the speed of a physical CPU and F_{VCPUi} is the speed of a virtual core allocated to that physical CPU.

2 Problem

Since optimizing the allocation and scheduling of tasks on real-time multiprocessors systems is quite complex, it seems like the perfect problem for a genetic algorithm. There are a variety of parameters to be optimized, and we need to explore the parameter space, but to do so exhaustively would take exponential time. By encoding a fitness function related to the efficient use of our multiprocessor resources, we can use the genetic algorithm to evolve parameters that (roughly) maximize this value.

3 Methods

Our program is a genetic algorithm that evolves ideal parameters for task allocation on real-time multiprocessors. This program is written in C, so that it easily interfaces with existing virtualization software. We input the number of machines in the population, the number of generations for the algorithm to run, and the number of virtual cores in the virtual machine. We select random values in a fixed range for the maximum utilization value for each physical core and the CPU speed for both the physical and virtual cores. These values are fixed across the different machines in the population: for example in every machine physical core number 1 will have the same max utilization and speed values. Our chromosome consists of a TDF for each virtual machine, slice and period values for every core of every virtual machine, and an allocation matching of a virtual core to a physical core for every virtual core. All of the values evolved must meet the requirements detailed above, so we will check every chromosome for that after each step of evolution. Our fitness function will be

$$fitness = \frac{\sum_{i=1}^{n} \frac{U_{achieved_i}}{U_{max_i}}}{TDF*n}$$

Where,

n = number of physical cores.

$U_{achieved_i}$ = Achieved utilization for physical core i.

U_{max_i} = Maximum utilization for physical core i.

This fitness function measures how close to full each of our physical CPUs are. We plan to always provide enough virtual cores to saturate the available physical cores. The static input parameters for this problem are the number of physical cores, the speed of each physical core, the utilization value of each physical core, the number of virtual cores, and the speed of each virtual core.

Our first step will be to write our basic genetic algorithm. Based on the work we have done on the previous project, we feel that this will not be too difficult. The extra step we will have to add in is checking whether the evolved components meet the requirements described above. Our next step will be to experiment with a variety of mutation rates and types, crossover rates and types, and selection types. We want to generate a table similar to those in Project 2.1. We will also graph the evolving chromosomes to get a feel for the change in parameters over time.

Once we have found the best set of parameters, we plan to do many runs of the algorithm and collect the top performing chromosomes. We then plan to compare them and find the overall top

three. In order to get a feel for how our evolved chromosomes perform in real situations, we would like to test these parameters in the Palacios Virtual Machine Monitor, which is a free software package that Oscar currently works with [5]. This is not planned to be a major component of our project, but we think it would help inform whether our choice of the fitness function made sense.

4 Results

5 Conclusion

References

[1] S. Dall and C. Liu. On a real time scheduling problem. *Operation Research*, 1978.

[2] K. Y. Diwaker Gupta and M. MacNett. To infinity and beyond: Time-warped emulation. proceedings of the 3rd usenix symposium network on networked system design and implementation. *3rd USENIX Symposium network on Networked System Design*, 2006.

[3] J. D. Jose Lopez and D. Garcia. Minimum and maximum utilization bounds for multiprocessor rate monotonic scheduling. *IEEE Trans. on Parallel and Distributed Systems*, 2004.

[4] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 1973.

[5] O. Mondragon and P. Bridges. Emulation of large scale high performance systems through vmms. *9th Annual UNM Computer Science Symposium*, 2013.

[6] O. Pereira and P. Mejia. Edf and rm multiprocessor scheduling algorithms: Survey and performance evaluation. *Queue*, 2005.