Parallel Process Distribution  
of Robotic Software Applications

Nicholas Anthony Degani  
Advisor: Arvind Krishnamurthy  
February 1, 2003

1 Proposal

Parallel processing is a constantly changing field, as new hardware creates new paradigms for study. At the birth of the field, manufacturers created specialized hardware pre-wired to execute processes in parallel, and thus the field largely focused on hardware design and implementation. As the field matured, more venues have opened themselves to those interested in parallel architectures; researchers have connected processors directly to each other with shared memory resources, the cluster model of networked IBM clones has risen in popularity, and even the inter-connected channels of the World Wide Web have become a subject of interest to students of parallel processing in the form of distributed computing systems.

One area of parallel processing of particular interest is the distribution of interrelated programs across a pre-defined network, and their interactions. In other words, if a programmer has a set of related processes he’d like to run on a machine, how can he best speed up the output of these processes by using a parallel processors? What would be the optimal number of processors to use in such a machine, and how should they be connected? How should the processes be distributed across the processors? Also, how would any of these solutions change if fundamental parameters relating to the interrelated processes changed? For example, if the amount of information communicated from Process A to Process B doubled (and so on down the line), how would these solutions change?

I propose to investigate a mathematical model based on graph theory to explore these questions, and to create an algorithm which could automatically generate solutions given certain parameters. The model would be initially based on current graph theories that seek to bifurcate a graph into equal parts (i.e. the weight of all the nodes on each half is roughly equal) while minimizing the weight of the bifurcated edges. However, such a model excludes a key component of the system: edges with weights dependent on some independent variable. Thus I’ll seek to modify these algorithms to accommodate an independent random variable with a simple probability distribution function, thus equating the expected load across processors and minimizing the expected communications across the network.

To ground this project in reality, I will attempt to incorporate my theoretical findings with a real-life implementation. Professor Scassellati has developed a set of real-time robotic software applications that attempt to use a parallel machine architecture to speed processing. The parallel network architecture consists five sets of four processors; within each set all four processors are interlinked, while each processor than directly connects to a switch which links between sets. Within the architecture, several physical processes are directly
connected to certain processors requiring the associated input/output processes to be run on those processors alone. For example, the visual input from one of the robot’s “eyes” is connected to one processor, while the input for a motor is connected to a different processor. Generally what happens is a input is received to the parallel network through an input process (like visualization), which then passes data to secondary process (like motion detection), which then feeds data to an output process (like motorized movement). While such an example seems straight-forward and simple, an accurate “artificial intelligence” for the robot would probably require the joint operation of dozens of processes, all of which would be distributed across the parallel network.

While it would be fantastic to develop a working program to directly attack the problem found in Prof. Scassellati’s project, that will only be a secondary goal. The primary deliverable will be a paper examining a mathematical model of the problems presented above, along with an algorithm to plot the optimal distribution of processes. A secondary deliverable could include modifications to various programs used in the Professor’s project, or a program to determine optimal process allocation for the specifics of his project.