Dynamo – A Portable Tool for Dynamic Load Balancing on Distributed Memory Multicomputers

Erik Tärnvik

Institute of Information Processing, Department of Computing Science, University of Umeå, S–901 87 Umeå, Sweden, Email: erikt@cs.umu.se

Abstract. Dynamic load balancing is an important technique when developing applications with unpredictable load distribution on distributed memory multicomputers. A tool, Dynamo, that can be used to utilize dynamic load balancing is presented. This tool separates the application from the load balancer and thus makes it possible to easily exchange the load balancer of a given application and experiment with different load balancing strategies.

1 Introduction

The distribution of workload is a basic and important aspect to consider when developing programs on parallel computers. The present paper describes a portable tool that can be used to utilize dynamic load balancing on distributed memory multicomputers (DMM).

There exists problems where the distribution of workload in a DMM can not be predicted. In such cases, the workload should be balanced at run-time. This is known as the dynamic load balancing problem. In the general case, the optimal solution to this problem is known to be NP-complete [2]. However, heuristic methods can be used to implement suboptimal dynamic load balancing strategies.

Problem areas known to exhibit computational characteristics that benefit from dynamic load balancing includes searching problems, e.g. branch-and-bound algorithms [6] used to solve different optimization problem, such as combinatorial optimization or global optimization [1], and problems solved by Monte Carlo type algorithms [4], e.g. particle dynamics.

Section 2 discusses the design of the Dynamo tool and the programming interfaces. In Sect. 3 an example application using Dynamo is presented. Finally, in Sect. 4 we draw some conclusions and give an outline of future work.

2 Dynamo

The Dynamo tool relies heavily on the idea that the application should be separated from the load balancing operations. This functionality separation is necessary if the load balancing mechanism is to be handled by a distributed operating system on future parallel systems. Dynamo addresses the following issues:

- The easy exchange of load balancers allows for testing of load balancing strategies on different types of applications without changes to the load balancer.
A library of load balancers can be built from which the developer can choose and experiment with different load balancers for a given application in order to achieve maximum performance.

An application can be ported to different target architectures provided that the tool is available on the target environment.

Dynamo can be ported. In its present implementation, Dynamo is written in C and built on the PICL (Portable Instrumented Communication Library) communication library of Oak Ridge National Laboratory [3].

Dynamo provides tracing facilities which supports performance evaluation and debugging tools, e.g. visualization and execution playback using Paragraph [5].

Dynamo is not intended to be responsible for the actual load balancing of application programs. Dynamo provides a framework for the development of load balancers. A prototype of Dynamo has been implemented for the Intel iPSC/2 DMM system.

2.1 Tasks

The smallest executable part of a Dynamo program is the task. A task is a part of the solution to a problem, and partitioning a large problem into tasks is the programmers responsibility.

A Dynamo task is defined by its dataset, which is the data associated with the task. The operations on the datasets are not a part of the task, but must be present on each processor in order to support the execution of a task on all processors. The task dataset in itself might be anything the application decides it to be. The only requirement is that at the time of spawning of a task, the dataset occupies a contiguous portion of memory. If this is not the natural way to store a task dataset, the dataset has to be packed and unpacked by the application before communicating with Dynamo. There are two reasons for this requirement:

1. In order to support datasets occupying non-contiguous memory, a set of location attributes (or some similar scheme) would be necessary.
2. If noncontiguous datasets were supported, they would have to be packed into contiguous memory before communication anyhow, in order to minimize communication overhead.

In addition to its dataset, a task has a number of attributes. The type attribute is used to distinguish different task types from each other. The origin attribute is automatically assigned by Dynamo when a task is created and identifies the processor on which the task was originally created. The location attribute is a pointer to the task's dataset. This is used by the application and the load balancer to access the dataset. The size attribute holds the size of the dataset referred to by the location attribute. The combination of location and size gives Dynamo enough knowledge to handle task datasets. The priority attribute is an estimate of how important a certain task is to the solution of a problem. Some load balancer use a priority queue, based on this attribute, for task management. For other applications, the notion of priority might not be applicable. Cost gives an estimate of the processor resources needed to execute the task. The cost attribute is only relative to the other tasks present in the system. This attribute may be used by some load balancers to improve the balancing of the workload in the system.