An Assessment of Multilisp: Lessons from Experience

Robert H. Halstead, Jr.

Received April 1987; Revised June 1987

Multilisp is a parallel programming language derived from the Scheme dialect of Lisp by addition of the future construct. It has been implemented on Concert, a 32-processor shared-memory multiprocessor. A statistics-gathering feature of Concert Multilisp produces parallelism profiles showing the number of processors busy with computing or overhead, as a function of time. Experience gained using parallelism profiles and other measurement tools on several application programs has revealed three basic ways in which future generates concurrency. These ways are illustrated on two example programs: the Lisp mapping function mapcar and the partitioning routine from Quicksort. Experience with Multilisp programming exposes issues relating to side effects, error and exception handling, low-level operations for explicit manipulation of futures and tasks, and speculative computing, which are also discussed. The basic outlines of Multilisp are now fairly clear and have stood the test of being used for several applications, but further language design work is especially needed in the areas of speculative computing and exception handling.

Key Words: Lisp; parallel symbolic computing; futures, performance evaluation.

1. INTRODUCTION

Multilisp is an extended version of Scheme, a lexically scoped dialect of Lisp. The Multilisp execution environment contains the same sorts of data types and primitive operators as Scheme or any Lisp dialect. In Multilisp,
however, many lines of computation, or tasks, can be active simultaneously, manipulating objects in a single shared heap. Multilisp's basic mechanism for generating concurrent tasks is the future construct, described later.

Multilisp is a permissive "playpen" for experimenting with parallel programming. Although future has proven quite versatile as a tool for writing parallel programs, Multilisp does not force its users into a particular programming style. Therefore, Multilisp includes side effects and many other features that are banned from more Puritanical languages. Although Multilisp does not enforce a particular programming style, it does support, through future and the advanced features inherited from Scheme, a reasonable programming environment containing the ingredients necessary for building programs of significant size. Therefore, Multilisp has a dual personality. On the one hand, it directly supports a particular style of parallel programming using future. On the other hand, it is flexible enough to support experimentation with different programming styles. Most of this paper is concerned with the former aspect to Multilisp, but the later sections also address the latter aspect.

Multilisp has been implemented on Concert, a 32-processor shared-memory multiprocessor that has been constructed at the Massachusetts Institute of Technology, M.I.T. Large sections of Concert have been available for use since mid-1985, and smaller sections were available as long ago as 1982. Thus, although the availability of the full-size Concert is relatively recent, there has been ample time to experiment with application programs written in Multilisp. Although many questions remain unanswered, experiments have yielded some insight into how best to use future, and have allowed us to reach some preliminary conclusions about Multilisp's strengths and weaknesses. The goal of this paper is to communicate as much as possible of the insight that has been gained.

We begin with brief overviews of Multilisp and Concert (more extensive descriptions appear elsewhere; see Refs. 3–5) and then illustrate several ways to use future. Next is a discussion of additional Multilisp constructs that help satisfy certain needs of application programs that are not easily met using future alone. Finally, our areas of happiness and dissatisfaction with Multilisp as it stands are summarized, and the directions in which evolution of Multilisp is likely to occur are indicated.

2. MULTILISP BASICS

The expression (future X), where X is an arbitrary Multilisp expression, creates a task to evaluate X and also creates an object known as a future to eventually hold the value of X. When created, the future is in