Implementing a Scheme-Based Parallel Processing System

James S. Miller

Received August 1988; Revised March 1989

The Scheme language can be converted into a parallel processing language by adding two new data types (placeholders and weak pairs), two processor synchronization primitives, and a task distribution mechanism. The mechanisms that support task creation, scheduling, and task synchronization are built using these extensions and features already present in the sequential language. Implementing the core of the parallel processing component in Scheme itself provides testbed for a variety of experiments and extensions.

Multischeme, the system resulting from these extensions, supports Halstead's future construct as the simple model for parallelism. By revealing the underlying placeholders on top of which this construct is built, Multischeme supports a variety of additional parallel programming techniques. It supports speculative computation through a simple procedural interface and the automatic garbage collection of tasks. The qlet and qlambda constructs of the QLisp language are also easily implemented in Multischeme, as are the more familiar fork and join constructs of imperative programming.

KEY WORDS: Multischeme; parallel Lisp; implementation; future construct; placeholders.

1. INTRODUCTION

Multischeme is a fully operational parallel-programming system based on the Scheme dialect of Lisp. Like its Lisp ancestors, Multischeme

1 This research was supported in part by the Defense Advanced Research Projects Agency and was monitored by the Office of Naval Research under contract numbers N00014-83-K-0125, N00014-84-K0099, N00014-86-K-0180, and MDA903-84-C-0033. Additional funds and resources were provided by BBN Advanced Computers Inc., and the Hewlett-Packard Corporation. The work was performed as part of the author's dissertation research at the Massachusetts Institute of Technology.

2 Brandeis University, Department of Computer Science, Waltham, Massachusetts 02254.

367
provides a conducive environment for prototyping and testing new linguistic structures and programming methodologies. MultiScheme supports a diverse community of users who have a wide range of interests in parallel programming. MultiScheme's flexible support for system-based experiments in parallel processing has enabled it to serve as a development vehicle for university and industrial research. At the same time, Multi-Scheme is sufficiently robust, and supports a sufficiently wide range of parallel-processing applications, that it has become the base for a commercial product, the Butterfly Lisp System produced by BBN Advanced Computers, Inc.

MultiScheme, in the tradition of the Scheme language, is designed as a "minimalist" system. It provides a small but powerful set of constructs from which a researcher can build layers of language suited directly to a particular application. This paper describes the innermost core of the MultiScheme system, the procedures (written in Scheme) that implement the critical operations of the system. Collectively these procedures are referred to as "the scheduler," although they provide a greater range of services than this name implies.

1.1. Placeholders and the Future Construct

From a simple user's point of view, MultiScheme is just a Scheme system with one important addition: the future construct derived from Halstead's Multilisp. This gives the programmer a way to annotate opportunities for parallelism. The special form future can be wrapped around any expression in the language, and indicates that the enclosed expression is permitted to run in parallel with the surrounding expression. MultiScheme requires the use of a specific construct to express opportunities for parallelism because Scheme permits side-effects, and thus a certain amount of control is desirable. Furthermore, the minimalist approach taken by the Scheme community argues in favor of a programmatic interface to parallelism in order to form a convenient base for experimentation in the design of automated tools for inserting parallelism. Initial exploration into building such tools has been undertaken by Gray and Wang.

As a simple example, the doubly recursive calculation of fibonacci can be conveniently described in MultiScheme:

```
(define (fib n)
  (if (< n 2)
    1
    (+ (future (fib (- n 1)))
        (future (fib (- n 2))))))
```