Connection Networks for

Output-driven List Multiprocessing

by

Steven D. Johnson

Computer Science Department

Indiana University

Bloomington, Indiana 47405

TECHNICAL REPORT NO. 114

CONNECTION NETWORKS FOR

OUTPUT-DRIVEN LIST MULTIPROCESSING

STEVEN D. JOHNSON

OCTOBER, 1981

Research reported herein was supported (in part) by the National Science Foundation under grant number MCS77-22325.

Connection Networks for Output-driven List Multiprocessing**

Steven D. Johnson*

Abstract

Processor-memory communication is discussed for a multiprocessor whose control is based on call-by-need. Concurrent global memory accesses are arbitrated by a connection network whose design is tailored for list processing. One refinement, called the NEW-sink, reduces the waiting time for a common kind of transaction, the allocation of new list cells. A specific connection scheme, a banyan network, has been simulated to show the effects of design refinements.

CR Categories: 4.32, 6.21.

Keywords: Connection Network, Multiprocessing, call-by-need, list processing.

*Mailing address: Indiana University Computer Science Dept., 101 Lindley Hall, Bloomington, Indiana, 47401.

**Research reported herein was supported (in part) by the National Science Foundation under grant number MCS77-22325.

1.0 INTRODUCTION - A MULTIPROCESS MODEL

The term output-driven refers to a paradigm of computation in which results are produced only when it is determined that they are needed. When confined to the relationship between a function and its arguments, this is known as call-by-need [23]. The only "control structure" is the dispersal of need from its ultimate source, the printer(s) -- which absolutely must produce answers [11]. Of course, these answers are determined by the text of a program. This approach to computation has lead us to view the "data space" as a medium of manifest objects in which are embedded suspensions -- transparent processing entities which converge to needed values.

This heterogeneous relationship between process and data is the foundation for the design of a general purpose list multiprocessor. Most mechanisms for introducing concurrency in programs, especially applicative programs, can be described operationally in terms of suspending constructors. Techniques ranging from collateral argument evaluation [9] to nondeterminisic computations and explicit multiprocessing operators [8,12] have been so described.

Our ultimate goal is to specify the architecture of an extensible computer whose processing resources are transparent to the software being run. "Software independence" is achieved by giving the host system sole responsibility for allocating processors to suspensions. By "computation" we

mean the reduction of a functional expression to normal form. More specifically, we adopt the graph reduction techniques found in Pure LISP [20] for computing results. Processors (also called evaluators) traverse lists, some of which are interpreted as programs. These lists reside in a common store and may contain suspensions. Once a suspension has been accessed, some evaluator computes its value.

The overall view, depicted in Figure 1, is that of a self-activating data space. In any foreseeable implementation technology, the number of stored data objects exceeds the number of processing components by several orders of magnitude; there may be far more suspensions in the data space than evaluators. This report focuses on the problem of sharing the store. The issue of control, which evaluator computes which suspension, is not considered here. Access to the store consists of list processing primitives. Messages have a fixed format about the size of a memory cell; since the intended host language is applicative, unconstrained side effects do not occur [10,14].

The global architecture for the system discussed here is shown in Figure 2. A connection network arbitrates concurrent access to the store by some number of processors. The crucial issue is whether a network can be devised in which processing power can grow without bound. Numerous connection networks have been proposed for multiprocessing. The familiar extremes, the bus architecture and the full crossbar switch

exemplify the fundamental tradeoff between extensibility and complexity in communication networks. The finite bandwidth of any bus imposes an upper limit on the number of components that can be attached to it. The polynomial hardware cost of a crossbar and the algorithmic complexity of its control make it almost impossible to extend. Between the extremes are innumerable "compromise" architectures. A routing network based on a banyan graph structure [22] has been chosen for discussion in this report. It is moderately extensible and has a highly decentralized control algorithm. Within reasonable bounds its hardware cost is acceptable [6], and it is often put forward as a candidate for general purpose multiprocessing [4, 5,15,17].

Although we concentrate on the role of the connection network in the overall design, in doing so we are forced to characterize the behavior of the other components. Assertions about how processors and memories behave are based on other studies [2,16,18], and are used to parameterize a simulation model, documented below. The simulation is part of a more ambitious design effort [19], but in this report it serves primarily as a tool for isolating the qualitative effects of enhancements to the design.

In summary, what follows is a cursory description of communication hardware for the management of data space access in an output-driven, applicative machine. It is also a forum for the discussion of the computational components of such a

machine. The machine's many processors are geared toward list processing. Section 2 establishes some vocabulary for the rest of the report and describes the local behavior of system components. Section 3 describes the architecture of banyan networks. Section 4 discusses the expected global performance of the proposed machine, followed in Section 5 by a description of the observed performance in simulation. Section 6 is a summary.

2.0 DEFINITIONS AND OVERVIEW.

Although "communication" is the main subject of this report, that term is used in a restricted sense, meaning "the exchange of digital information among a number of components." "Component" too is a base term, meaning "some digital computing device," which may be of general or special purpose. At any time, many individual exchanges occur (or are attempted) concurrently.

1. An agent is any component that communicates.
2. A processor is an agent that initiates communication, an "active" or a "computing" component.
3. A memory is an agent that responds to communication, a "passive" or a "storage" component.
4. A switch is an agent that connects processors to memories, a "routing" component.