LOSP FOR THE INTEL iPSC

William Bricken
August 1987

INTRODUCTION

The Losp Parallel Deduction Engine is an implementation of a new logical formalism for parallel computation. This formalism rests on a single proto-logical concept, the *distinction*. By relying on a single-operator logic and an efficient representation scheme, Losp can execute deductive steps in a highly parallel fashion. This capability has been used for a variety of applications, including inference engine support, theorem proving, expert systems applications, and theoretical studies of boundary mathematics.

THE DESIGN OF LOSP

We transcribe conventional logical expressions into networks of distinctions, mapping each distinction onto a *virtual processor*. Ideally, in a parallel system of sufficiently fine granularity, each distinction is physically implemented by a single processor. Logical structure is maintained by the interconnectivity of the distinction network.

Deduction occurs as each distinction examines its local connectivity and instigates actions to erase or rearrange these connections. The initial distinction network is in the form of the target problem. Local, asynchronous, parallel actions rapidly convert the network into a representation of the solution. Strong parallelism is demonstrated since the global solution to a problem emerges from independent local actions of distinctions.

The Losp formalism and its implementation demonstrate that logical deduction is inherently a parallel process. This parallelism is obscured by conventional representations of logic and is made visually obvious by the network of distinctions. The engine can be used for identification of tautologies, for Boolean minimization, for expert system rulebase validation and compilation, and for program optimization. The network display provides a visual programming language, and illustrates program animation.

THE iPSC LOSP DEMONSTRATION SYSTEM

We have implemented the Losp Parallel Deduction Engine on the Intel iPSC, by constructing it on top of the SOPE distributed AI environment. Each Losp distinction node is encapsulated in a SOPE "system" for purposes of communications and scheduling. Thus each SOPE system provides its Losp distinction node with the environment of an apparent virtual (lightweight) process.

The Losp front-end display has windows that show both visually and computationally the parallel deductive process. We see the message passing activity of the Losp algorithms as they autonomously and cooperatively modify their connectivity. We also see the representation of the network transform itself in parallel as deduction proceeds. Display controls allow modification of the speed of transformation, permit subnetworks to be specified for processing, and allow network reconfiguration. Input and output languages may be selected from logic, LISP, Boolean algebra, and *parens*, the linear notation for distinction networks. The axioms and theorems of boundary logic may also be selectively enabled and disabled.

SOPE: The Systems Oriented Programming Environment

Andrew S. Cromarty
Michael B. Moore
Michael K. Cation

INTRODUCTION

SOPE, the Systems Oriented Programming Environment, is a programming environment for constructing large Artificial Intelligence (AI) software systems, especially those that must operate in a distributed environment and meet severe performance constraints. SOPE is intended to serve as a vehicle for research and development in large distributed systems by supporting multiple programming paradigms and control strategies; it was designed with the goals of providing a high degree of flexibility to the software developer, guaranteed large-system portability across all Common LISP implementations and support for multiple control paradigms.

THE DESIGN OF SOPE

SOPE provides an object-oriented style of programming built upon a Common LISP programming language base. The basic programming unit or "object" in SOPE is the SOPE *system*. Each SOPE system is an independent processing agent with its own local data space and control strategy. Because they are embedded in a taxonomic inheritance graph, SOPE systems inherit "type" and "instance" definitions, and in fact a system may serve as both type and instance. When a system is sent a message, it is "activated" and its top level begins processing until it decides to deactivate itself. Systems can be interrupted between system activations.

A library of initial SOPE systems useful in AI system construction are defined for the user, including blackboards, demons, and inference engines. For example, a blackboard top level has an agenda which consists of systems that the blackboard will activate to accomplish its tasks. The top level can be interrupted between these activations.

Among SOPE's unique features to assist the user are:

-- a *top level* unique to each system, allowing different control mechanisms to be employed on a system-by-system basis

-- types and instances that are represented and treated in a uniform manner

-- communication that may be synchronous or asynchronous

-- uniform message handlers that make no distinction between methods and data objects (in message processing or inheritance)

-- systems that are dynamic and can have structure and inheritance relationships modified at run-time

-- multiple internal representations that support optimization on a system-by-system basis

-- debugging, tracing, and timing tools that provide significantly better support than typical object-oriented programming implementations

THE iPSC IMPLEMENTATION OF SOPE

We have ported a research prototype of the SOPE environment to the iPSC hypercube to pursue studies in distributed object-oriented artificial intelligence software system design and construction.

SOPE systems are implemented as Concurrent Common LISP (CCLisp) structures in the iPSC SOPE implementation. A non-preemptive scheduler of lightweight processes (processes which all have an address space in common) and an emphasis on the message passing paradigm are other important features of this SOPE design. In addition, for the Intel iPSC implementation of SOPE, extensions were added to the SOPE language to allow SOPE objects to be referenced by name between each of the Hypercube processors. This additional address space was implemented using Gold Hill CCLISP "fasl streams" as its primitive communication channel.

Other extensions to SOPE facilitate the use of loosely coupled processes (those which do not share an address space). Locks and semaphores, necessary constructs for synchronization, were added to the language. The resulting research prototype has been used successfully to construct a true distributed implementation of the Losp parallel deductive inference engine.