User's Manual for Pyramid Emulation on The Connection Machine

Lisa Gottesfeld Brown
Qifan Ju
Cynthia Norman

CUCS-341-88

Department of Computer Science
Columbia University
New York, NY 10027

May 31, 1988

1 Introduction

This manual describes a set of functions for using image pyramids on the Connection Machine. These functions are an extension of *LISP which itself is an extension of COMMON LISP. The functions were designed to work on the Connection Machine 2 of the Northeast Parallel Architectures Center located at Syracuse University.The manual is organized as follows:

- description of pyramids
- naming conventions
- getting started
- global pyramid variables

- pyramid primitives
- horizontal communications
- vertical communications
- pyramid display
- pyramid input/output

2 Description of Pyramids

The routines described herein can be used to write *LISP programs for vision algorithms which use multiresolution image pyramids to structure and manipulate image data. The implementation is described in detail in: Pyramid Algorithms Implementation on the Connection Machine by Hussein Ibrahim; DARPA Image Understanding Workshop 4/88. Basically, by using these routines, the user can program multiresolution algorithms on the Connection Machine so that inter-pyramid communications will be executed efficiently. Examples of typical multiresolution algorithms include the computation of depth from stereo or motion and image registration. Students here at Columbia University and also at Syracuse University are currently working on an implementation of hierarchical stereo correlation using this system. It is important to note that this system deals with pyramids where each node communicates with exactly four children below it, to a single parent above it, and to four neighbors on the same level. This system would probably be inappropriate for emulating pyramids with more general configurations.

3 Naming Conventions

The functions described all use a new data structure called a pyramid (or pmd). This structure is actually composed of two pvars although this is transparent to the user. All the functions are written so that in most cases those dealing with pyramids are analogous to standard *lisp functions which deal with pvars. For example, to create a pmd the function is *defpmd (like

*defvar) and *set-pmd (like *set). In addition, the following conventions are adhered throughout so that properties of functions and variable names are as obvious as possible:

pmd!! All functions which end with *pmd*!! return a pmd.

***–pmd** All functions which begin with an asterisk and end with *pmd* have arguments which are pmds.

pmd– All variables which start **pmd* and end with an asterisk are global pyramid variables (see section on global variables).

These conventions are similar to those for pvars in *LISP. We also adhere to their conventions; namely, functions ending with !! return pvars, functions starting with asterisks use pvars internally.

4 Getting Started

To use the pyramid environment it is necessary that the following two conditions are met:

1. The number of logical processors should equal the number of physical processors. (This will be extended so that the number of logical processors can be any multiple of four times the number of physical processors.)

2. The machine should be configured for 2 dimensions Hence, the number of processors should be of the form 2^2n where n is an integer.

The first condition can be met by attaching to the same number of processors as you configure the machine when *cold-booting. The second condition is met with the :initial-dimensions to *cold-boot. To load the pyramid system, the file "pyramid-emulate" should be loaded and the function pyramid-emulate is then executed. An example session which shows a start-up is shown in the file "pmd-example." Most of the commands issued in this session are also contained in the file, "pmd-init.lisp" which is the initialization file used for testing the system.

5 Global Variables

The following are global variables used by the pyramid system which might also be useful to the user. A variable of particular importance is *pmd-level-number*.

pmd-number-of-levels the number of levels in the pyramid.

pmd-size the size of one side of the base of the pyramid

pmd-self-address the address of each processor in the pyramid. Note: these addresses indicate the location of the physical hypercube connections that connect the processors.

pmd-level-number a pvar which indicates the particular level other than the lowest level which a processor represents. This pvar can be used in a *when to select only the processors on a certain level (above the leaf level).

6 Pyramid Primitives

The following functions are for creating, allocating and setting the values of pyramids and their levels. They form the basis of all pyramid programs. In addition, the *LISP function *let can be used to dynamically create pmds using the function allocate-pmd!!.

```
(*defpmd pmdname &optional pmd-initialization)

(allocate-pmd!!)

(*deallocate-pmd pmd)

(*set-pmd pmd-1 pmd-2)

(*set-level-pmd level pmd-1 pmd-2)
```

7 Horizontal Communications

The following functions execute mesh communications within individual levels of the pyramid. They work according the scheme specified in section 3 of the article "Pyramid Algorithms Implementation on the Connection Machine."

```
(shift-level-pmd!! level direction source-pmd &optional dest-pmd
                   &key border-pmd)
```

This uses the mesh on the specified level to shift the data in source-pmd in the specified direction ('e 'w 'n or 's) and puts the resulting level in the optionally specified dest-pmd and returns it. For example: `(shift-level-pmd!! 1 'e pmd-in pmd-out :border-pmd zero-pmd)` shifts level 1 in pmd-in to the east and stores the result in level 1 of pmd-out which is returned. The border-pmd is used in the same way the border-pvars are used in *lisp commands such as pref-grid!!.

```
(shift-pmd!! direction source-pmd &optional dest-pmd &key border-pmd)
```

This is the same as the above function except all levesl are shifted and stored in dest-pmd and returned.

8 Vertical Communications

The following functions execute top-down communications between levels of the pyramid. They work according the scheme specified in section 4 of the article "Pyramid Algorithms Implementation on the Connection Machine." Top-down communications either transfer (and 'combine) data from 1 or more children up the pyramid or from a single parent to 1 or more of its children. Children are specified as 'a 'b 'c or 'd. When communicating up the pyramid an operation is specified which indicates how the four children are combined before setting the parent value. The operation can be any parallel operation such as +!! or *!!.

```
(send-level-parent-pmd!! level operation source-pmd &optional dest-pmd)
```

```
(send-level-children-pmd!! level source-pmd &optional dest-pmd)

(send-level-child-pmd!! level child source &optional dest-pmd)

(ave-pmd!! level source-pmd &optional dest-pmd)
```

The command ave-pmd!! is used to make a pmd using an image stored in the lowest level and making each successive level above by averaging the four children of each parent. This is a typical example of a function which uses the vertical communication functions to make an image pyramid.

9 Pyramid Display

These routines can be used to print the data stored in a pyramid either as a single level or the entire pyramid.

```
(*display-level-pmd pmd level)

(*display-pmd pmd)
```

10 Pyramid Input/Output

These routines can be used to store a pyramid or pvar into a file for use with standard sequential image processing routines or for reading such a file into a pyramid or pvar for use with this system. Typically a pvar represents an image from which a pyramid can be constructed.

```
(read-pvar!! ''filename'' &optional pvar)

(read-pmd!! ''filename'' &optional pmd)

(write-pvar!! pvar ''filename'')

(write-pmd!! pmd ''filename'')
```