1. Introduction

PC is an extension of C. The original motivation was to simplify the writing of message–passing code for multiprocessors with distributed memory. The objectives were two–fold: first to make it easier to write programs for parallel machines; second to insure that exchange of data among different processors is done correctly. The language provides an efficient approach to a unified address space for distributed memory processors as well as a programming model for general parallel computing.

Single Code Programming

PC makes the important assumption that each processor knows the names of the variables in all processors. To this end we require, in the current implementation, that all processors run the same PC code. For acronym addicts, our programming model is Single Code Multiple Datastream or SCMD. Thus the programming style is similar to SIMD languages (e.g., *-Lisp on the Connection Machine) used for SIMD programs. However, SCMD programs need not proceed in lock–step, operation by operation, as would occur on SIMD machines.

deterministic versus *chaotic*

In implementing a language such as PC, one must decide whether to assume that everyone knows what everyone else is doing, or not. For example, if you know in advance that a neighbor may want to borrow a cup of sugar, you can put one by the back door when you have a spare moment and then go on about your business. Otherwise, you would have to wait at home indefinitely for the knock at the back door. Moreover, you can make sure that there is sugar available; otherwise the operation of dipping the cup in the sugar bin may have undesired results. We refer to a parallel programming environment in which the implementor assumes that we know in advance what others will require of us a *deterministic* environment. An environment in which we must constantly be prepared for arbitrary demands on our resources we call a *chaotic* environment. For simplicity, we have limited our implementation of a compiler for PC to the deterministic case. There is nothing (that we know of) in PC that prevents implementation of a compiler in the chaotic case, but it is clear that the demands on the operating system and node architecture are much greater in order to do so efficiently.

How to Use PC

In the Cosmic Environment

In the current implementation, the compiler for PC produces C code. Compilation for the sun network is done via `pcc -C`. The compiler is in the directory `/usr/local/bin`. For example, if `codename.pc` is a PC code, then

```
/usr/local/bin/pcc -C  codename.pc
```

produces a C code called `codename.c`. This file contains message–passing C code that can be compiled for the Cosmic Environment using the C compiler:

```
/usr/local/cube/ccgh  -o codename codename.c  -litocube -ltrk -lcube
```

with the usual additional options available. The compiler `ccgh` is located in the directory `/usr/local/cube` so this directory must be included in your path if you do not want to use the full path every time. This compiler will produce an executable named `codename.gh`.

To run on the Suns in the Cosmic Environment, you must first get some resources. For example,

```
getcube ghost 2n
```

will acquire two Suns on which to execute your code(s). The executable file `a.out` resulting from the above compile can be executed via

```
spawn codename  -1
```

on the Suns.

You should not have either of the following in your PC code:

```
#include <stdio.h>
#include <malloc.h>
```

Inclusion of either of them can cause unpredictable results.

When you are done running your code you should release the ghost cube by saying `freecube`.