Abstract

This paper summarizes work conducted at NPAC, Syracuse, in the ARPA-funded research project PCRC (Parallel Compiler Runtime Consortium). We describe the architecture, functionality and implementation of a public domain PCRC runtime library, and an associated HPF compiler that uses this library. Experimental data show that on some typical applications the compiler gives performance comparable with, and sometimes better than, commercial HPF compilers. An infrastructure constructed during the compiler development is also described.

As currently implemented the core NPAC PCRC runtime library is a class library consisting of about 20,000 lines of C++ code. Three groups of functions are provided: distributed data management, data movement routines, and computational functions corresponding to HPF intrinsic functions. Uniform compiler- (or user-) specified decomposition of data and computation is facilitated by three important classes of object: the *distributed array descriptor* (DAD), the distributed *range*, and the process *group*. The DAD captures all array distributions allowed in HPF 1.0. The *Range* represents an integral interval distributed over some process dimension, corresponding to either a distributed array dimension or a triplet index in a section specification or `FORALL` statement. The *Group* captures certain runtime dynamics of a distributed array resulting from rank-reduced sectioning and procedure argument passing. Ranges and groups occur as subcomponents of the DAD.

Data movement routines include `pcrc_write_halo()`, which supports utilization of ghost regions for efficient processing of *shift* communications, and `pcrc_remap()`, which supports arbitrary remapping of regularly distributed arrays. General *gather* and *scatter* operations are provided for irregular data access. All HPF intrinsics (except, currently, the sorting operations) are supported.

The HPF compiler serves as a testing ground for effective HPF compilation techniques and also as an exemplar for use of the PCRC runtime. A novel communication detection algorithm was devised to distinguish communication requirements occurring in an assignment between two distributed arrays. This identifies three disjoint cases: *no communication*, *shift communication*, and *remap communication*. The algorithm accounts for the combined effect of alignment, distribution, and sectioning of an array, and produces better results than the commercial compilers we compared with. A performance study on a Laplace solver shows our compiler delivers results comparable to commercial compilers when the arrays are simply distributed, and our compiler gives better results when the arrays have strided alignments to their templates.

The uniform approach of using the DAD as the mechanism for argument passing between node program and runtime routines, as well as between caller and callee in node program, means we are able to handle the `INHERIT` directive effectively.

Along with the above work, a general infrastructure for high performance compiler construction is being constructed. Sharing a common abstract syntax tree specification, this infrastructure now consists of an HPF 2.0 front-end and a Java front-end, together with a class library modified and extended from Sage++. This includes up to 80,000 lines of C and C++. The infrastructure has been tested on seven UNIX platforms.