Lowering HPF Procedure Interface to a Canonical Representation*

Jan Borowiec**, Arthur Veen***

Abstract. Handling the procedure interface in an HPF compiler is complex due to the many possible combinations of Fortran 90/HPF properties of an actual array argument and its associated dummy argument. This paper describes an algorithm that reduces this complexity by mapping all the combinations of properties to a small set of canonical Internal Representations. These internal representations as well as the necessary run-time descriptors are also described. The algorithm has been implemented in the commercial HPF compiler produced by the PREPARE project.

1 Introduction

In the recently completed PREPARE project [9] a commercial HPF 2.0 compiler ([6], [7]) has been implemented. In contrast to other HPF compilers that generate C or FORTRAN code as intermediate representation, this compiler uses an especially tailored Internal Representation (IR).

To make the PREPARE compiler easily retargetable, it is embedded in the CoSy family, a set of compilers for several source languages as well as target processors in which complicated components, such as optimizers, can be reused in many different compilers [1]. The extent to which components can be reused depends greatly on the way source language constructs are lowered, i.e. mapped to constructs of the IR. Since the array mapping facilities are unique to HPF, an extended version of the IR of the CoSy family had to be defined which we call the PIR (Prepare Internal Representation). The PIR and the CoSy IR have been carefully designed to contain only a minimal set of canonical constructs. We call such constructs canonical not only because they serve to represent similar constructs in different source languages, but also because most source languages contain a variety of redundant components that can all be lowered to the same PIR construct. Especially a large language as Fortran 90 contains such redundancy. To keep the development and maintenance of the HPF compiler manageable and to maximize the reuse of optimizers and efficient code generators, it was essential to remove as much of this redundancy as possible.

* This work was supported by the Commission of the European Communities under Esprit Project 6516 PREPARE.

** GMD FIRST Research Institute For Computer Architecture and Software Technology, Rudower Chaussee 5, 12489 Berlin, Germany; email: borowiec@first.gmd.de

*** Parallel Computing, Amsterdam, email: arthur@parcom.nl
The lowering of HPF is performed in two steps (see Fig. 1.). First a conventional front-end translates the source program into an abstract syntax tree called the HIR (High level IR). This HIR is specific for HPF and is described in detail in [3]. The second step of the lowering is performed by the component called the HPT, the HIR-to-PIR-Transformer.

HPT's output is rooted by a PIR program unit which contains (refers to) a list of types, a list of globals declarations, and a list of procedure units. A procedure unit contains a list of locals declarations and a body which is a list of basic blocks. In turn, a basic block entails a list of statements. Thus, stating below that HPT generates e.g. a statement, we mean that an appropriate statement subtree is created and appended to the statement list of the currently created basic block of the currently compiled procedure, etc.

A previous paper [4] gave details about the part of the HPT that performs the lowering of array assignment statements, WHERE and FORALL statements and constructs. It was shown that all these HPF components can be transformed to just one powerful canonical PIR construct: the parallel array assignment.

Another major part of the HPT is the handling of the array mapping information (ALIGN, DISTRIBUTE, etc.). The current paper reveals some of this by describing the canonical PIR constructs that represent mapping information (section 2). Most of this paper (sections 3 and 4) focuses on the most complex part of this, the handling of mapped arrays at procedure call boundaries.

The mapping information represented in the PIR is used by later components...