Automatic Vectorization of Communications for Data-Parallel Programs

Cécile Germain\textsuperscript{1} and Franck Delaplace\textsuperscript{2}

\textsuperscript{1} LRI-CNRS Université Paris-Sud
\textsuperscript{2} Université d'Evry

Abstract. Optimizing communication is a key issue in compiling data-parallel languages for distributed memory architectures. We examine here the cyclic distribution, and we derive symbolic expressions for communication sets under the only assumption that the initial parallel loop is defined by affine expressions of the indices. This technique relies on unimodular changes of basis. Analysis of the properties of communications leads to a tiling of the local memory addresses that provides maximal message vectorization.

1 Introduction

Static analysis of data-parallel programs, for the generation of distributed code, has been proposed by many authors, for instance [5] [6] [8] [10] [15]. Static analysis aims to improve performance over run-time resolution [3] which includes a lot of pure overhead in form of guards and tests. Many static compilation schemes have been considered; they differ in important points such as interleaving computation and communication as in [6], or having identical management of local and non-local data such as in [8]. However, they all use three basic sets: \textit{Compute}(s) is the part of the index set which is local to processor \textit{s}; \textit{Send}(s) (resp \textit{Received}(s)) is the part of a distributed array that has to be sent (resp received) by processor \textit{s} when owner computes rule is applied. The central problem of static analysis is to define these sets at compile-time, and in an efficient form.

Two major costs have to be considered for the code generation scheme: the computing cost, and the communication cost. The computing cost is all the overhead required to compute local indices, and, when a communication occurs, to compute the parameters of the communication, the destination processors and the local addresses. As pointed out by [6], naive resolution leads to a symbolic form involving integer divides for each forwarded data, which may be as inefficient as run-time resolution. The communication cost depends on the volume and number of communications. For a data-parallel program, the volume, i.e. the number of data to send to a remote processor, cannot be modified, because it is fixed by the placement function (e.g. ALIGN and DISTRIBUTE directives). At the code generation level, optimization is only directed towards the number of communications, by aggregating all data that are to be sent to the same processor. Although this may seem a very specialized problem, the overwhelming part of startup in message cost makes this optimization a major component of performance, as shown in [15].
To be amenable to static analysis, the references must be affine functions of the parallel loop indices, a reference being an access or alignment function, and the loop bounds must be defined by affine inequalities. These assumptions are the weakest possible. Under these assumptions, deriving efficient closed forms of the previous sets for the most general block-cyclic distribution is an open problem. [8] gives a general compiling scheme under the weakest assumptions, but provides closed forms only when indices are independent: for instance, $T[j, i]$, but not $T[2i + j, i - j]$. [5] uses a finite state machine approach, allowing optimal memory utilization, but restricts references to array sections and uses integer divides. [10] solves the same problem with a virtualization method. Other special cases have been solved, for unit strides in [15], for one-dimensional arrays in [6] and [14].

In this paper, we derive closed forms providing an efficient code generation scheme, under the weakest assumptions, when the parallel arrays are cyclically distributed. Next part formally states the problem and discusses the relationship with the problem of scanning integer polyhedra. Part three analyzes the conditions for message vectorization and proposes an explicit closed form achieving maximal vectorization; part four details the SPMD code and its optimizations, and presents some examples.

2 General Compilation Scheme

2.1 Problem Statement

We consider nested parallel loops, with given alignment and the cyclic distribution, such as described in High Performance Fortran (HPF); we restrict our analysis to the static subset of HPF where arrays are aligned once, at compile-time, and all index functions are affine; moreover, the index set must be described by affine inequations. The generic loop nest is:

\[
\text{forall } i \text{ in } C \\
X(Bi + b) = f(Y(A_1i + a_1), Z(A_2i + a_2), \ldots) \\
\text{end forall}
\]

where $B$, $A_1$ and $A_2$ are integer matrices, and $b$, $a_1$ and $a_2$ are integer vectors.

Some notations must be defined, associated with the cyclic distribution: let $p_1, p_2, \ldots, p_n$ be the extents of the PROCESSOR target of the distribution, $p$ be the vector with coordinates $p_i$, $P$ the diagonal matrix with coefficients $p_i$, and $P$ the processor set, i.e. $P = \prod_{i=0}^{n}[0, p_i-1]$. Template element $j$ is laid on processor $s$ such that $s_i \equiv j_i \mod p_i$ for all $i = 1 \ldots n$. In the following, the coordinates subscripts are elided, and scalar operations are extended to vector ones by coordinates. Hence, array element $j$ defines a set of spatial coordinates $s$ and a set of memory coordinates $t$ by euclidean division: $j = Pt + s$, with $0 \leq s < p$. For any $s$ in $P$, $Z^s_n$ is the set of integer vectors congruent with $s$ modulo $p$.

Distributed code for the previous loop can be generated at compile-time if \text{Compute}(s), \text{Send}(s) and \text{Receive}(s)$ can be described for each processor $s$ in a