Distribution Class Analysis

Hans Moritsch

Vienna University of Technology
Institute of Computer Languages
Argentinierstraße 8
A-1040 Vienna, Austria
hans.moritsch@tuwien.ac.at

Abstract. The compilation of languages for parallel computations which provide constructs to control the dynamic allocation of data structures to processors requires the analysis of the relationship between array references and data distribution functions. Expressions representing classes of distribution functions constitute the basic data flow information. For the interprocedural analysis, they are represented symbolically. A backward phase over the call graph calculates precise procedure summaries, a subsequent forward phase, by propagating calling context, eliminates the symbols therein. The meet operation allows for subsumptions, subclasses of already occurring classes are ignored. If the control flow is affected by the distribution class related to a reference, the data flow information is masked with the decisive condition. The analysis is based on a reduced flow graph containing only the nodes relevant to the analysis outcome.

1 Introduction

Information about the distribution of arrays is essential for a compiler for a data parallel programming language to generate efficient code. Optimization strategies can improve the code considerably, if detailed knowledge about the use of data and work distributions in a program is available. Extensive analysis has to provide this information.

The *distribution* of an array is a function that defines how the elements of the array are partitioned into subsets—often rectangular *segments*—and mapped to processors. This has various implications, most importantly, with respect to the allocation of arrays in the processors' memories, and the processors' responsibilities for preforming calculations with arrays.

The array distribution is specified in connection with the array declaration, after the keyword DIST, for each array dimension as a reference to an intrinsic distribution function such as BLOCK or CYCLIC, or the keyword NONE for "no distribution".[1] For example, a DIST (BLOCK BLOCK) specifies a blockwise distribution for array a in both dimensions, whereas b DIST (BLOCK NONE) defines, that array b is distributed blockwise in the first dimension, and not distributed

[1] NONE for every dimension specifies a replicated array, yet this is seen as a special case of distribution.

in the second dimension i.e., b is is partitioned into blocks of rows. Vice versa, (NONE BLOCK) specifies blocks of columns (see Figure 1).

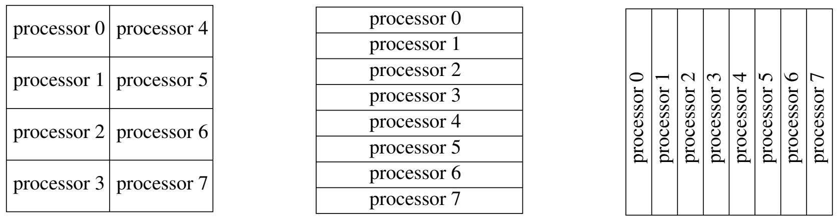

Fig. 1. Segments of a (BLOCK BLOCK), (BLOCK NONE), and (NONE BLOCK) distribution

The distribution of an array may be changed at runtime by means of the DISTRIBUTE statement. Through

DISTRIBUTE b (NONE BLOCK)

b will be *redistributed* to a (NONE BLOCK) distribution.

The current distribution of an array can influence the control flow in the program by means of a *distribution query* in the form of a test for “Identical Distribution Types”.

if a IDT (BLOCK $*$) then ... else ...,

restricts the execution of the then-part to states in which a is distributed blockwise in the first dimension, independent of its distribution in the second dimension. The complementary condition is relevant to the else-part. The wildcard “$*$” is used in the standard manner.

Whenever a distributed array is passed as a parameter in a procedure call, its distribution is passed as well. On procedure return, the actual parameter adopts the distribution of the formal parameter.[2]

The language constructs presented show, in a simplistic syntax, a small subset of the features for distributed arrays in data parallel languages [5, 4]. Primarily, they are supposed to serve as the basis for the presentation of our analysis.

This analysis delivers a range of possible distribution states for each statement of a program composed of several procedures. It includes an intra- and an interprocedural level, connected with each other through a symbolic representation of distribution classes (see Section 4, Section 3).

As a special feature, the analysis deals also with classes of distributions constituted by use of wildcards. This gives reasons for the definition of a *subsumption* relation between more general and more specific classes of distributions.

[2] A different behavior can be achieved through appropriate placements of DISTRIBUTE statements at procedure boundaries.

2 Intraprocedural Data Flow Analysis

2.1 Data Flow Information

A *distribution expression* is a list, equal in length to the number of array dimensions; each element corresponds to an array dimension and is

- an intrinsic distribution function name (such as BLOCK or CYCLIC),
- or NONE,
- or the wildcard "$*$", which stands for any of the above.

A distribution expression specifies a class of distributions, which is called a distribution type [5].

A *distribution state* is a vector of distribution expressions, in which each component is assigned to a declared (distributed) array. Alternatively, a component can be formed from an array identifier, namely of a formal parameter array, which accepts its distribution type from the corresponding actual parameter. The array identifier is in fact a symbol representing the transferred distribution type.

A *masked distribution state* is a distribution state with a distribution mask, a *distribution mask* is a set of pairs (array identifier, distribution expression). A pair may be labeled with a negation "$\neg$". The same array identifier can occur in more than one pairs, thus an arbitrary number, including zero, of distribution expressions can be assigned to an array. It is also possible, that no array occurs, i.e., the mask is the empty set. A distribution state without a (non-empty) mask is called *immediate*.

The data flow information set consists of *sets* of masked distribution states.

Example 1. The masked distribution state

$$\{a\,(\textsc{block}\ *), \neg\, c\,(\textsc{block}\ \textsc{none})\} \cdot [^{a:}(\textsc{block}\ \textsc{cyclic})\ ^{b:}\hat{a}\ ^{c:}(*\ \textsc{block})]$$

at a program point denotes the following.[3] If, on procedure entry, array a is BLOCK distributed in the first dimension, irrespective of its distribution in the second dimension, and array c is not (BLOCK NONE) distributed, then a is (CYCLIC BLOCK) distributed at that point, and array b has the same distribution type as a on procedure entry, and array c is BLOCK distributed in the second dimension, whereas its distribution in the first dimension is not specified and can be any of BLOCK, CYCLIC, NONE.[4]

A distribution expression is called *generic* in the presence of one or more wildcards.[5] A distribution expression which is not generic is called *terminal.* A distribution state is called generic, if it contains at least one generic distribution expression, otherwise it is called terminal.

[3] A pair (a, δ) in a mask is written as $a\,\delta$, and an array identifier component as $\hat{a}$.

[4] "a is δ distributed", "a has a δ distribution", and "a has the distribution type δ" have the same meaning.

[5] The genericity is an extension of the definition in [5].

2.2 Subsumption

Based on genericity, we define a relation on distribution expressions. A distribution expression δ is said to *subsume* a distribution expression δ', $\delta \succeq \delta'$, iff δ and δ' are equal in length, and δ contains $n \geq 0$ wildcards, and δ' contains at most n wildcards, and for each wildcard in δ' there is a wildcard in δ at the same position (i.e., for the same array dimension). By this definition a distribution expression subsumes itself. In contrast to this, a distribution expression δ is said to *properly subsume* a distribution expression δ', $d \succ \delta'$, iff $d \succeq \delta'$ and $\delta' \not\succeq d$. Note that $d \succ \delta'$ implies $d \neq \delta'$ and δ being generic.

Let $\tau(\delta)$ denote the set of terminal distribution expressions represented by δ. Iff δ subsumes δ', $\tau(\delta')$ is a subset of $\tau(\delta)$.

A distribution state s is said to subsume a distribution state s', $s \succeq s'$, iff s and s' are equal in length, and every component of s formed from a distribution expression subsumes the corresponding component of s', and every component formed from an array identifier is equal to the corresponding component of s'.

A distribution mask m specifies a predicate $S(m)$ on the distribution state of the formal parameter arrays on procedure entry. A distribution masks m is said to subsume a distribution mask m', $m \succeq m'$, iff m' represents the same or a stronger condition than m, i.e., iff $S(m') \subseteq S(m)$.

A masked distribution state $w = m \cdot s$ is said to subsume a masked distribution state $w' = m' \cdot s'$, $w \succeq w'$, iff $m = m'$ and $s \succeq s'$, or $m \succeq m'$ and $s = s'$, or both $m \succeq m'$ and $s \succeq s'$ hold, i.e., at least one of its parts subsumes the corresponding part of w'.

Sets of masked distribution states, which do not contain elements properly subsumed by other elements, are called *subsumption free*. They constitute the elements of the data flow information set L. Let W denote the set of masked distribution states. Then $L = \mathcal{P}(W)$, and for all $\ell \in L$, $w, w' \in W$

$$w \in \ell \wedge w \succ w' \rightarrow w' \notin \ell.$$

2.3 Meet Operation

The meet operation is a modified set union operation such that the result does not contain elements which are properly subsumed by other elements, the *subsumption free union* $\tilde{\cup}$. Let $\ell_1, \ell_2 \in L$. Then

$$\ell_1 \,\tilde{\cup}\, \ell_2 := \ell_1 \cup \ell_2 - \{w' \in W \mid \exists w \in \ell_1 \cup \ell_2 \wedge w \succ w'\}.$$

Based on $\tilde{\cup}$ a partial order on L can be defined,

$$\ell_1 \leq \ell_2 \Leftrightarrow \ell_1 \tilde{\cup} \ell_2 = \ell_1.$$

$\leq$ is the modified superset relation $\tilde{\supseteq}$,

$$\ell_1 \tilde{\supseteq} \ell_2 \Leftrightarrow \forall w' \in \ell_2 \,[\, w' \in \ell_1 \vee \exists w \in \ell_1 \ (w \succ w')\,].$$

2.4 Transfer Functions

The distribution state can change due to (i) redistributions, (ii) distribution queries, and (iii) procedure calls. For the pertaining types of flow graph nodes the transfer functions are given; for all other types the transfer function is the identity function.

In the following, let $s[a]$ denote the component of a distribution state s assigned to array a (the "a-component"), $w.m$ denote the mask part of a masked distribution state $w = m \cdot s$, $w.s$ denote the state part of w, and $\ell \in L$ denote the incoming data flow information at a node.

2.4.1 Redistribution

The effect of the redistribution

$$\text{DISTRIBUTE } a\ \delta,$$

where a is an array and δ is a distribution expression, is defined by the transfer function $f^{\mathrm{D}} : L \to L$ of a distribute node (a node representing a DISTRIBUTE statement). It sets in all elements of ℓ the a-component to δ. I.e., $f^{\mathrm{D}}(\ell)$ executes, $\forall\, w \in \ell$,

$$w.s[a] \leftarrow d.$$

In case of a statement

$$\text{DISTRIBUTE } a = b,$$

where both a and b are arrays, it sets the a-component to the distribution expression, or the array identifier, which is currently assigned to array b. Thus $f^{\mathrm{D}}(\ell)$ executes, $\forall\, w \in \ell$,

$$w.s[a] \leftarrow w.s[b].$$

Example 2. The redistribution

$$\text{DISTRIBUTE } a\ (\text{BLOCK CYCLIC})$$

in the state

$$[{}^{a:}(\text{BLOCK BLOCK}),\ {}^{b:}(\text{BLOCK NONE})]$$

yields

$$[{}^{a:}(\text{BLOCK CYCLIC}),\ {}^{b:}(\text{BLOCK NONE})],$$

whereas

$$\text{DISTRIBUTE } a = b,$$

in the same state, yields

$$[{}^{a:}(\text{BLOCK NONE}),\ {}^{b:}(\text{BLOCK NONE})].$$

2.4.2 Distribution Query A node n in the flow graph representing a distribution query

$$\text{if } a \text{ IDT } \delta \text{ then } \dots \text{ else } \dots,$$

where a is an array and δ is a distribution expression, has two successor nodes, corresponding to the two possible outcomes. For processing distribution queries, prior to the analysis, two *assertion nodes*, n^{true} and n^{false}, are added to the flow graph between n and its successors.[6] The effect of the query is specified through the transfer function $f^{\mathrm{A}} : L \to L$ of an assertion node; the transfer function of n itself is the identity function.[7]

We have to distinguish, whether the a-component of a distribution state in ℓ, $w.s[a]$, is (i) a distribution expression η, or (ii) an array identifier $\hat{u}$.

In the first case, for n^{true}, the query is evaluated through determining the intersection $\delta \cap \eta$; for n^{false}, the intersection $\neg\delta \cap \eta$ is calculated, respectively. If the result is empty, the entire distribution state is removed from ℓ, otherwise the a-component is set to the intersection. So, $f^{\mathrm{A}}(\ell)$ for n^{true} executes, $\forall\, w \in \ell$,

$$w.s[a] \leftarrow \delta \cap w.s[a].$$

In the second case, as the query depends on $\hat{u}$, it cannot be evaluated. Its effect is expressed symbolically, through adding the pair $(\hat{u}, \delta)$ for n^{true} (the pair $(\hat{u}, \neg\delta)$ for n^{false}, respectively), to the mask. Thus $f^{\mathrm{A}}(\ell)$ for n^{true} executes, $\forall\, w \in \ell$,

$$w.m \leftarrow w.m \cup (w.s[a], \delta).$$

$f^{\mathrm{A}}(\ell)$ for n^{false} is specified analogously.

Example 3. The effect of the query

$$a \text{ IDT } (\text{BLOCK } *)$$

on the state (in the then-part)

$$[^{a:}(* \text{ BLOCK}),\ ^{b:}(\text{BLOCK NONE})]$$

is, with $(\text{BLOCK } *) \cap (* \text{ BLOCK}) = (\text{BLOCK BLOCK})$,

$$[^{a:}(\text{BLOCK BLOCK}),\ ^{b:}(\text{BLOCK NONE})].$$

In contrast, for the state

$$[^{a:}\hat{a}\ ^{b:}(\text{BLOCK NONE})],$$

f^{A} yields

$$\{a\,(\text{BLOCK } *),\} \cdot [^{a:}\hat{a}\ ^{b:}(\text{BLOCK NONE})].$$

[6] For queries without else-part, only what is said about n^{true} applies.
[7] However, f^{A} knows both a and δ.

2.4.3 Procedure Call A procedure call can, by some redistribution in the called procedure, change the distribution type of a parameter array. Further, the effect can, through a distribution query, depend on the current (i.e., before the call) distribution type of the same, or a different, actual parameter.

Transfer functions for procedure call and return nodes are involved in handling a procedure call. We assume only one return node in a procedure's flow graph.

Return Node The purpose of the transfer function $f^{\mathrm{R}} : L \to L$ for a return node of a procedure p is to summarize the effect of a call of p on the formal parameter arrays' distribution types. Local arrays in p do not affect the analysis of the calling procedure, hence f^{R} shrinks ℓ to information about formal parameter arrays. $f^{\mathrm{R}}(\ell)$ executes, $\forall\, w \in \ell$,

$$w.s \leftarrow \Pi_{F_p}(w.s),$$

where Π_{F_p} denotes the projection onto the set F_p of formal parameter arrays of p. Components of s assigned to local arrays, if they exist, are removed. Since only formal parameter arrays can occur in a mask, f^{R} has no effect on the latter.

After completion of the analysis of a procedure, R_p, the set of *return states*, identical with the result of f^{R}, describes the effect of a call.

Call Node The processing of a procedure call requires the previous analysis of the called procedure. The transfer function of the call node $f^{\mathrm{C}} : L \to L$ is based on R_p and has to interpret the parameter transfer. Masks in distribution states in ℓ, i.e., at the calling site, remain unaffected by f^{C}. In the following, let A denote the set of actual parameters of a call of p, and $\varphi_p : A \to F_p$ denote the mapping to corresponding formal parameters.[8]

Immediate Return States At first we consider distribution states in R_p without masks. $f^{\mathrm{C}}(\ell)$ executes, for $\forall\, w \in \ell, r \in R_p$, acting on a temporary copy w' of w,

$$\forall\, a \in A \colon t \leftarrow r.s[\varphi(a)],$$

where t is the component of r corresponding to the actual parameter a. If t is a distribution expression, then the component's new value is ready, so

$$w'.s[a] \leftarrow t.$$

Otherwise, t is an identifier of a formal parameter array. The corresponding actual parameter is $\varphi^{-1}(t)$, and the respective component at the call site will become the new value

$$w'.s[a] \leftarrow w.s[\varphi^{-1}(t)],$$

regardless of whether it is a distribution expression or an array identifier. In the latter case, it refers to a formal parameter of the calling procedure (currently analyzed) and thus represents a distribution type passed to it from *its* caller.

[8] We consider only array parameters.

Components of w' assigned to other arrays than actual parameters of the call remain unmodified. The ultimate result of f_p^{C} is formed from the subsumption free union of all temporary states w' built as described,

$$f_p^{\mathrm{C}}(\ell) = \widetilde{\bigcup}_{w \in \ell, r \in R_p} w'.$$

Example 4. The analysis of the procedure $p(x, y, z)$ reveals the return state

$$r = [{}^{x:}(\textsc{block block}),\ {}^{y:}\hat{z},\ {}^{z:}\hat{x}].$$

The call $p(a, b, c)$ in state

$$w = [{}^{a:}(\textsc{block cyclic}),\ {}^{b:}(\textsc{block none}),\ {}^{c:}\hat{b},\ {}^{d:}(*\ \textsc{block})]$$

produces, by
$r.s[\varphi(a)] = r.s[x] = (\textsc{block block})$,
$r.s[\varphi(b)] = r.s[y] = \hat{z}$, $w.s[\varphi^{-1}(z)] = w.s[c] = \hat{b}$,
$r.s[\varphi(c)] = r.s[z] = \hat{x}$, $w.s[\varphi^{-1}(x)] = w.s[a] = (\textsc{block cyclic})$,
the state

$$w' = [{}^{a:}(\textsc{block block}),\ {}^{b:}\hat{b},\ {}^{c:}(\textsc{block cyclic}),\ {}^{d:}(*\ \textsc{block})].$$

Example 5. The analysis of the procedure $q(x, y)$ reveals the return state

$$r = [{}^{x:}(\textsc{block cyclic}),\ {}^{y:}\hat{y}].$$

The call $q(a, b)$ in the state

$$w = [{}^{a:}(\textsc{block block}),\ {}^{b:}(\textsc{block none})]$$

produces, by
$r.s[\varphi(a)] = r.s[x] = (\textsc{block cyclic})$,
$r.s[\varphi(b)] = r.s[y] = \hat{y}$, $w.s[\varphi^{-1}(y)] = w.s[b] = (\textsc{block none})$,
the state

$$w' = [{}^{a:}(\textsc{block cyclic}),\ {}^{b:}(\textsc{block none})].$$

In contrast, the call $q(a, b)$ in the masked state

$$w = \{a\,(\textsc{block}\ *)\} \cdot [{}^{a:}(\textsc{block block}),\ {}^{b:}(\textsc{block none})]$$

produces, retaining the mask,

$$w' = \{a\,(\textsc{block}\ *)\} \cdot [{}^{a:}(\textsc{block cyclic}),\ {}^{b:}(\textsc{block none})].$$

Masked Return States A mask in a return state can be evaluated to the extent in which it does not relate (taking into account the formal–actual parameter mapping) to distribution types transferred already to the calling procedure.

For every actual parameter, the distribution expression δ of every pair in the mask, in which the formal parameter corresponding to the actual parameter a occurs, will be compared with the a-component of the distribution state w

$$\forall\, a \in A\colon \forall\, (\varphi(a), \delta) \in m\colon t \leftarrow w.s[a].$$

If t is a distribution expression η, the pair is evaluated through computing the intersection $\delta \cap \eta$. Only if the result is non-empty for all pairs, w' will be produced as described, and added to the result of f_p^C.

If t is an array identifier $\hat{u}$, the pair $(\hat{u}, \delta)$ is added to the mask,

$$w'.m \leftarrow w'.m \cup (w.s[a], \delta).$$

This is equivalent to the handling of queries in f^A. However, here the condition arises from a query in the called, not in the currently analyzed, procedure.

Example 6. The analysis of the procedure $q'(x, y)$ reveals the masked return state

$$r = \{x\,(\textsc{block}\ *)\} \cdot [^{x:}(\textsc{block cyclic}),\ ^{y:}\hat{y}].$$

The call $q'(a, b)$ in the state

$$w = [^{a:}(\textsc{block block}),\ ^{b:}(\textsc{block none})]$$

evaluates $(\varphi(a), (\textsc{block}\ *)) \in r.m$, $w.s[a] = (\textsc{block block})$,
$(\textsc{block block}) \cap (\textsc{block}\ *) = (\textsc{block block}) \neq \varnothing$.

As the intersection is non-empty, f^C produces (see Example 5) the state

$$w' = [^{a:}(\textsc{block cyclic}),\ ^{b:}(\textsc{block none})].$$

In contrast, the call $q'(a, b)$ in the state

$$w = [^{a:}(\textsc{cyclic block}),\ ^{b:}(\textsc{block none})]$$

evaluates
$w.s[a] = (\textsc{cyclic block})$,
$(\textsc{cyclic block}) \cap (\textsc{block}\ *) = \varnothing$,
hence nothing will be produced.

Example 7. The call $q'(a, b)$ in the state

$$w = [^{a:}\hat{a}\ ^{b:}(\textsc{block none})]$$

evaluates $w.s[a] = \hat{a}$. The intersection $\hat{a} \cap (\textsc{block}\ *)$ is unfeasible, hence the result (see Example 6) will be equipped with an equivalent mask,

$$w' = \{a\,(\textsc{block}\ *)\} \cdot [^{a:}(\textsc{block cyclic})\ ^{b:}(\textsc{block none})].$$

If $q'(a, b)$ is called in the masked state

$$w = \{b\,(*\ \textsc{block})\} \cdot [^{a:}\,\hat{a}\ ^{b:}(\textsc{block none})],$$

the mask will be expanded,

$$w' = \{a\,(\textsc{block}\ *), b\,(*\ \textsc{block})\} \cdot [^{a:}(\textsc{block cyclic})\ ^{b:}(\textsc{block none})].$$

2.5 Flow Graph Reduction

Most often, the nodes of the flow graph actually having an effect on the analysis represent a very small fraction of all the nodes. It is obvious that excluding the irrelevant rest (nodes with identity transfer functions) from the analysis can significantly improve its performance. The flow graph can be reduced to the minimum extent necessary. By inserting an edge from an irrelevant successor k of a relevant node n_1 to every relevant node n_2 that can be reached from k along a path $(k, k_1, \ldots, k_l, n_2)$ of irrelevant nodes $k_i, i \geq 1$, the k_i can be eliminated. Subsequent to the analysis of a procedure, the data flow information at the reduced flow graph's node k is propagated to the nodes $(k_1, \ldots, k_l)$ in the original flow graph.

3 Symbolic Interprocedural Analysis

3.1 Backward Pass

In a (first) *backward* pass, the MASKED_DISTRIBUTION_STATES data flow analysis (see Section 2) is performed for each procedure, in reverse topological order along the call graph. Unknown distribution types handed over from calling procedures are represented by means of symbols (identifiers of formal parameters arrays, see Section 2.1). The result of the analysis of a procedure is expressed—without loss of precision—as a procedure summary (set of return states, see Section 2.4.3) using these symbols.

A called procedure is always analyzed in advance of the calling one(s), so it is possible to treat a call by interpretation of the procedure summary; there is no need to analyze the called procedure specifically for different actual parameter distribution types, or different call sites; every procedure needs to be analyzed only once.

3.2 Forward Pass

Information obtained in the analysis of the main procedure does not depend on calling context and therefore does not refer to symbols. Consequently, it does not contain masks, i.e., the result of the analysis of the main procedure is built from immediate states only. In a (second) *forward* pass, this information is gradually propagated into all called procedures where it allows for resolving the symbols, and thus eliminating the masks (see Algorithm 1).

For each procedure, in topological order along the call graph, and for all procedure calls therein, and for all—now immediate—states at such a call site (call node), the actual parameter distribution types are mapped to the formal parameters. In the called procedure, the—now known—distribution types of the formal parameters can be substituted for the symbols, hence all masks can be evaluated (cf. Section 2.4.3). A non-empty intersection for all pairs in a mask must appear at least once (over all states at all call sites), otherwise the state will eventually be removed.

The remaining states represent the result of the whole analysis.

Algorithm 1. INTERPROCEDURAL DISTRIBUTION CLASS ANALYSIS

```
{backward pass:}
for each procedure $p$ in reverse topological order do
    solve MASKED_DISTRIBUTION_STATES on $p$'s flow graph
endfor

{forward pass:}
for each procedure $p$ in topological order do
    if $p \neq$ MAIN then
        remove masked states with untagged masks in $p$
        remove masks in $p$
    endif
    {all states in $p$ are immediate, propagate them into called procedures:}
    for each call of a procedure $q$ do
        for each state $s$ at call site do
            {evaluate masks in $q$:}
            for each mask $m$ in $q$ do
                if all pairs in $m$ yield non-empty then tag mask
            endfor
        endfor
    endfor
endfor
```

Figures 2 to 4 show the complete analysis of a program consisting of a main procedure which calls a procedure p, which in turn calls a procedure q.

4 Conclusion

In this paper, we presented an interprocedural data flow analysis which determines for every program statement the set of possible states of array distributions, considering classes of distributions. It deals with dynamic redistribution and the impact of distribution queries. The data flow information has a complex structure and supports subsumption relationships through wildcards.

The data flow analysis of procedure calls avoids approximations; the result is equivalent to the inline expansion of the calls. This is achieved by representing in the data flow information the formal parameters' properties as symbols, and by carrying along the conditions involving formal parameters' properties. In the second pass the symbols and conditions are resolved.

In our opinion, the basic principle behind is very general and can be employed for other kinds of data flow information and transfer functions than for the actual analysis as well.

References

1. B. Chapman, P. Mehrotra, H. Moritsch, and H. Zima. Dynamic Data Distributions in Vienna Fortran. *In Proceedings of the Supercomputing '93 Conference.* November 1993, Portland, Oregon.
2. B. Chapman, H. Moritsch, and H. Zima. Dynamically Distributed Arrays: Specification of the Compilation Method. Deliverable D1Z-2, CEI Project PACT - Programming Environments, Algorithms, Applications, Compilers, and Tools for Parallel Computation, April 1994.
3. B. Chapman, H. Moritsch, and H. Zima. The Implementation of Dynamic Data Distributions in the Vienna Fortran Compilation System: Language, Compile- and Run-Time Support. Deliverable D5.1f, ESPRIT III Project PPPE - Portable Parallel Programming Environments, July 1995.
4. High Performance Fortran Forum. *High Performance Fortran Language Specification, Version 2.0*, January 1997.
5. H. Zima, P. Brezany, B. Chapman, P. Mehrotra, and A. Schwald. Vienna Fortran – A Language Specification. ICASE Internal Report 21, ICASE, Hampton VA, 1992.
6. H. Zima and B. Chapman. *Supercompilers for Parallel and Vector Computers.* ACM Press Frontier Series, Addison-Wesley, 1990.

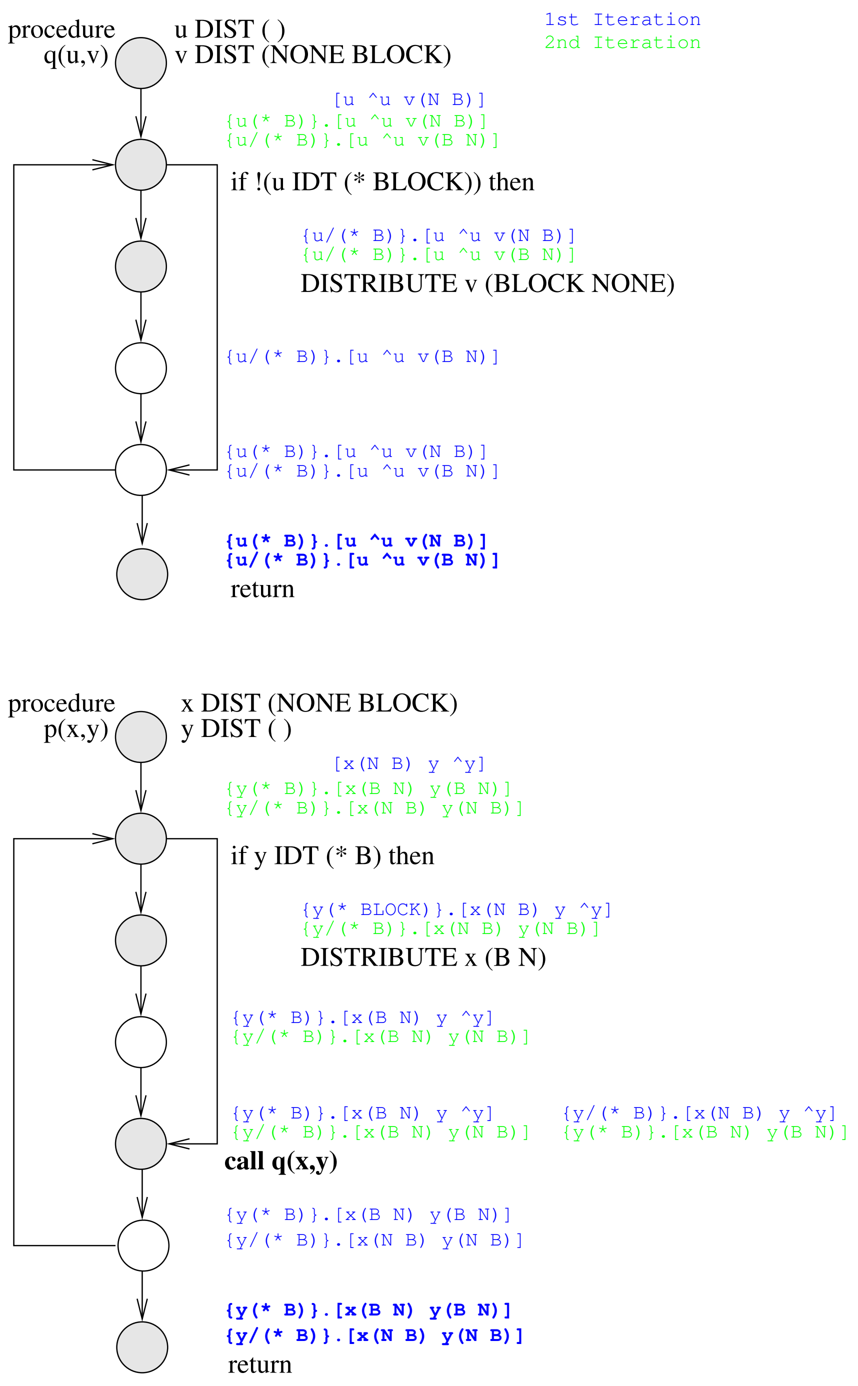

Fig. 2. Solution of MASKED_DISTRIBUTION_STATES for the procedures q and p (pass 1)

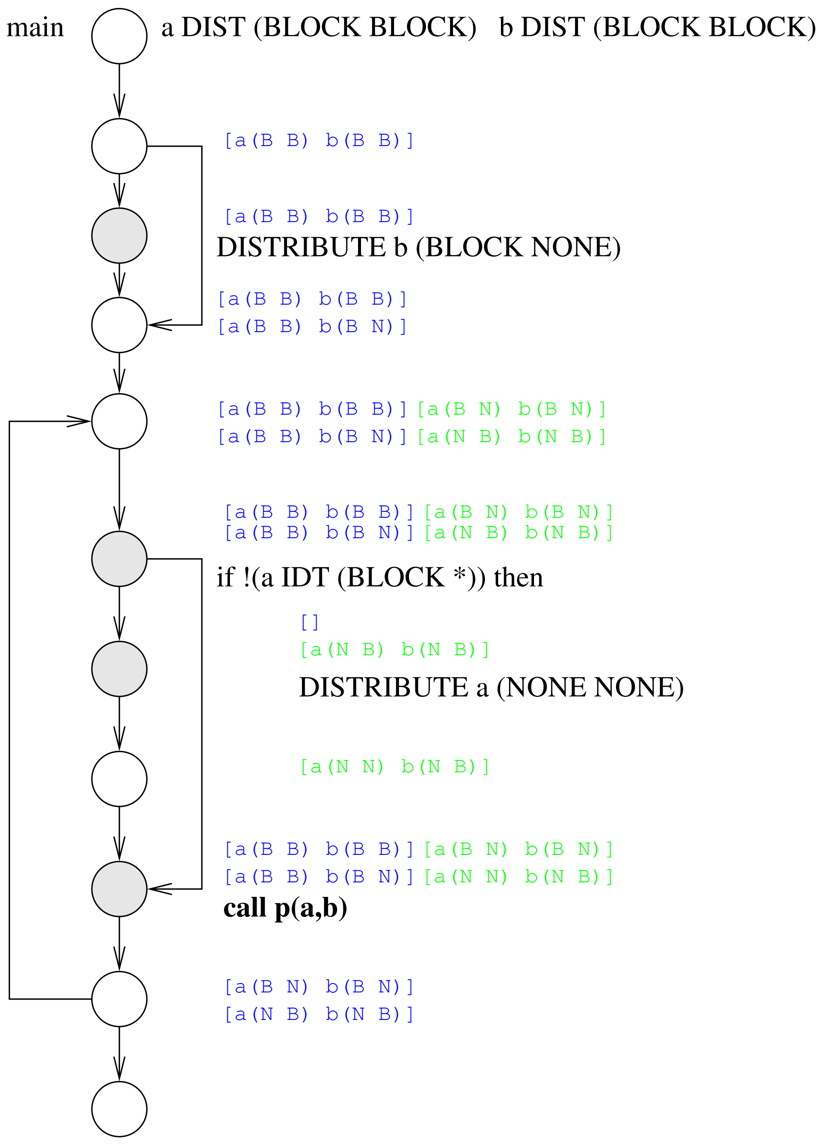
main
a DIST (BLOCK BLOCK)   b DIST (BLOCK BLOCK)
[a(B B) b(B B)]
[a(B B) b(B B)]
DISTRIBUTE b (BLOCK NONE)
[a(B B) b(B B)]
[a(B B) b(B N)]
[a(B B) b(B B)] [a(B N) b(B N)]
[a(B B) b(B N)] [a(N B) b(N B)]
[a(B B) b(B B)] [a(B N) b(B N)]
[a(B B) b(B N)] [a(N B) b(N B)]
if !(a IDT (BLOCK *)) then
[]
[a(N B) b(N B)]
DISTRIBUTE a (NONE NONE)
[a(N N) b(N B)]
[a(B B) b(B B)] [a(B N) b(B N)]
[a(B B) b(B N)] [a(N N) b(N B)]
call p(a,b)
[a(B N) b(B N)]
[a(N B) b(N B)]

Fig. 3. Solution for the main procedure

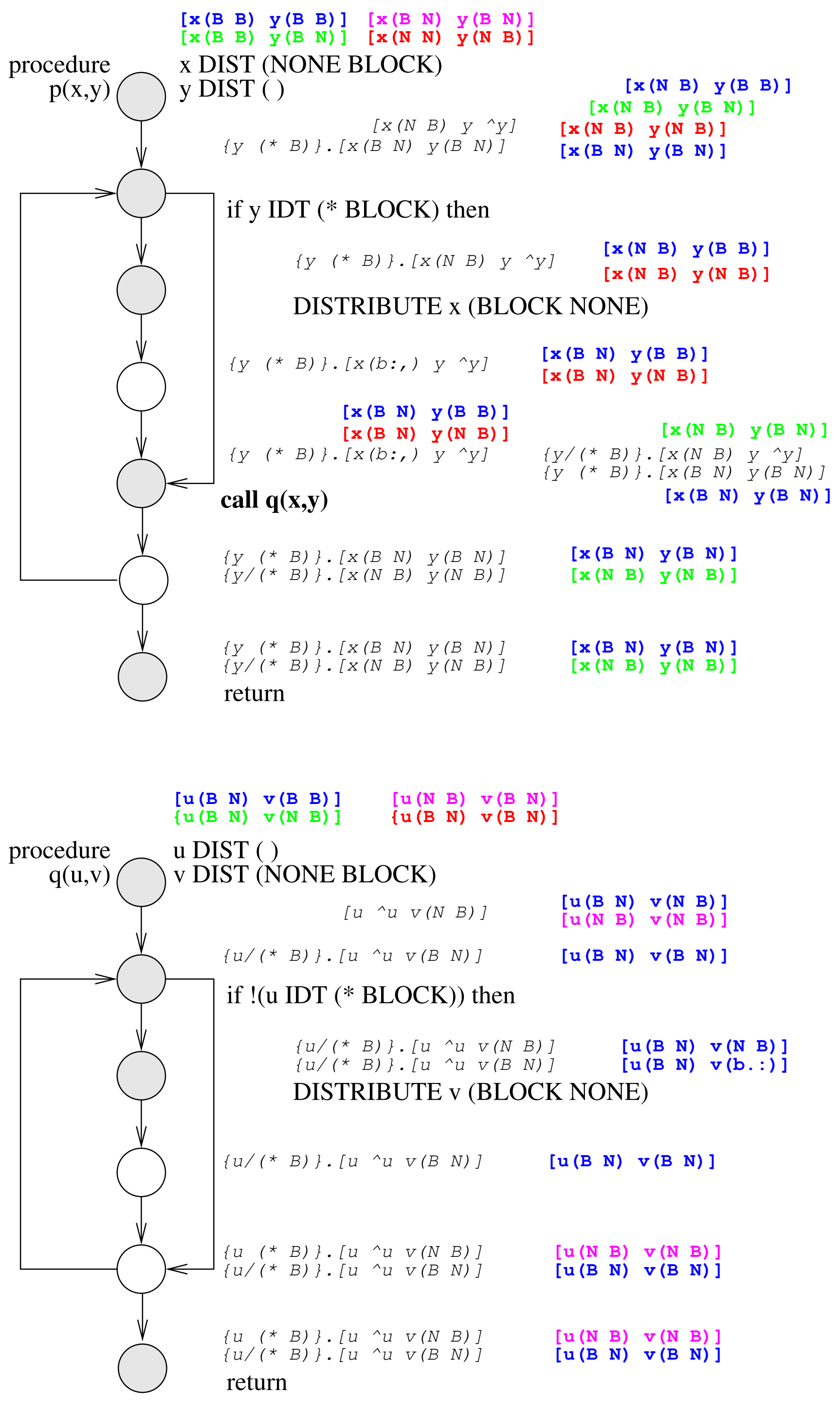

Fig. 4. Result for p and q after the down-propagation of states (pass 2)