Laws of Data Parallel Assignment

J.P. Wray

Department of Computer Science
The Queen's University of Belfast
Belfast BT7 1NN, UK
jp.wray@qub.ac.uk

Abstract. A set of laws for data parallel assignment is outlined. The laws illustrate the mathematical tractability of this programming construct and provide a means of correctly transforming a complex assignment into a sequence of simpler assignments which may then be interpreted on a variety of parallel architectures.

1 Introduction

One means of exploiting the potential of supercomputers for the efficient execution of scientific programs is to specify a set of "fine grained" order independent operations which may be applied to a data structure. The concept of applying order independent updates to data structures [1, 8, 7, 12, 10] may be viewed as an extension of multiple assignment [3, 4]. Data parallel assignment [9, 10] captures the concept of independence of a set of operations; it may be executed using any combination (parallel or sequential) of its atomic constituents (individual updates) and, consequently, is suitable for implementation on a wide range of parallel architectures.

Previous work on data parallel assignment [9, 10, 13] has resulted in the development of complementary denotational and axiomatic definitions of its semantics. These definitions exhibit mathematical regularity and facilitate the development and verification of correct data parallel programs. In further work a formal array processor based operational semantics of the construct has been specified [15] and data parallel assignment has been compared with apparently similar constructs in Fortran 90 and High Performance Fortran [11].

In this paper a set of laws for data parallel assignment is sketched. The motivation for studying the algebraic properties of data parallel assignment stems from the observation [5] that

> ... conventional programs are mathematical expressions, and are subject to a set of laws as rich and elegant as those of any other branch of mathematics, engineering, or natural science.

That is, it is possible to define laws which allow a programmer to manipulate and reason about programs in much the same way that mathematicians reason about, for example, logic and calculus. These manipulations may be carried out without reference to underlying domains or assertions. In particular, algebraic

laws may be used to define correctness-preserving transformations that can facilitate the interpretation of data parallel assignment on various parallel computer architectures. Furthermore, another consistent complementary description of the construct will provide more information about its mathematical regularity—as observed by Hoare and Lauer [6], a good intuitive criterion for such regularity of a language is that it is easy to describe formally *in more than one way.*

Section 2 provides a brief overview of data parallel assignment. The laws of data parallel assignment are outlined in Section 3. A structural normal form for data parallel assignment, the validity of which is established using some of the laws, is proposed in Section 4.

2 Data Parallel Assignment

Data parallel assignment is a means of specifying a set of *independent* updates that are to be applied to a fine grained data structure (an array). For example, if a, b, and c are $n \times n$ matrices, $\{(1,1),\ldots,(n,n)\}$ denotes the set of index pairs (i,j) such that $1 \leq i,j \leq n$, and $\{(k,l) \in S \mid P(k,l)\}$ denotes the set of index pairs which belong to S and satisfy the predicate P, then the data parallel assignment

$$\forall(i,j) \in \{(1,1),\ldots,(n,n)\}.a(i,j) := b(i,j) + c(i,j)$$

assigns to a the sum of b and c, and

$$\forall(i,j) \in \{(k,l) \in \{(1,1),\ldots,(n,n)\} \mid k = l\}.a(i,j) := 1$$

assigns the value "1" to each element in the diagonal of a.

The examples above involve scalars and structures of uniform dimension. A structure of one dimension may be assigned values from a structure of a different dimension using a data parallel assignment incorporating an *index function*, i.e., a mapping from the indices of the structure on the left-hand side of the assignment to the indices of a structure on the right-hand side. Data parallel assignments may also involve conditional expressions. In general, a data parallel assignment statement has the form

$$\forall ilist \in S.id(ilist) := exp$$

where *ilist* is a list of index variables, S is a set expression defining a set of actual indices (integer tuples), *id* is an array variable, and *exp* is an expression which may contain *ilist* as well as index functions. The effect of the assignment is to "simultaneously" assign, for all indices in the set S, the values of *exp* to the corresponding elements of *id*. Full details of the the syntax and denotational semantics of data parallel assignment may be found in [13].

3 The Laws

Each law is an equivalence of the form "$A1 \equiv A2$", where $A1$ and $A2$ are data parallel assignment schemas, and signifies that $A1$ may be rewritten as $A2$, and vice-versa, whilst preserving meaning. The laws fall into three main categories:

1. Substitution laws, which define how bound variables may be renamed whilst preserving meaning;
2. Laws derived from standard set and function laws such as the associative and distributive laws for set union;
3. Laws for simplifying assignments, for example, laws which define absorption of assignments and laws which define the removal of conditional expressions from assignments.

Lack of space precludes listing the laws in full here. For complete details see [14]. The following is an example of a law in the third of the above categories:

$$
\begin{aligned}
& a \notin free(S, B, e2) \models \\
& \forall i \in S.a(i) := If\ B\ Then\ e1\ Else\ e2 \\
& \equiv \forall i \in \{j \in S \mid B(j/i)\}.a(i) := e1; \forall i \in \{j \in S \mid \neg B(j/i)\}.a(i) := e2
\end{aligned}
$$

This law defines how a data parallel assignment involving a conditional expression may be replaced by a pair of unconditional assignments. It is valid only if the array variable a does not occur free in any of the sub-expressions S, B, or $e2$. The set expressions in each of the two unconditional assignments could then be further simplified through the application of substitution and set laws.

The validity of each of the laws can be established [14] using a straightforward application of the denotational definitions. One application of the laws is in establishing the equivalence of a data parallel assignment and a sequence of such assignments having a special form—structural normal form—discussed in the next section.

4 A Structural Normal Form

The definition of an operational interpretation of data parallel assignment is simplified if the number of possible syntactic forms which needs to be considered is reduced. This can be achieved by defining a structural form into which any particular data parallel assignment (involving arithmetic operators of maximum arity 2) may be transformed without changing its meaning:

Definition 1. A data parallel assignment of the form

$$\forall i \in S.a(i) := L\ OP\ NL$$

where

1. **L** is an expression in which the only term (if any) involving free tuple variables is **a(i)**;
2. **NL** is an expression in which the only term (if any) involving free tuple variables is **b(F(i))**, where b is an array variable and **F** is an index function;
3. **OP** is a binary operator

is in **structural normal form.**

Data parallel assignment is interpreted on a model distributed array processor in [15]. To this end, the structural normal form ensures that each "atomic" assignment involves at most one "non-local" data element (that which occurs in the sub-expressions NL).

It is necessary to prove that any data parallel assignment can be transformed into an equivalent sequence of structural normal form assignments. The strict denotational equivalence of two sequences of data parallel assignments is inappropriate for this purpose since, for example, the final values of temporary variables (such as "dummy" variables used in swapping the values of two program variables) are not significant from the programmer's point of view. Therefore the notion of "relative equivalence", as defined below, is used.

Definition 2. Let **SS1** and **SS2** be finite sequences of data parallel assignments, and let $a \in Id$. We say that **SS1** and **SS2** are **equivalent with respect to** a iff

$$\forall \sigma \in St. \forall i \in iddomain(\sigma, a). \mathcal{C}[SS1]\sigma(a, i) = \mathcal{C}[SS2]\sigma(a, i)$$

where C is the semantic function for commands, and $iddomain(\sigma, a)$ denotes the index range of the array a in state σ.

Informally, $SS1$ and $SS2$ are equivalent with respect to a iff they always effect the same transformation on a, regardless of their effect on other program variables. The following theorem states that the structural normal form defined above possesses the required properties.

Theorem 3. *For any data parallel assignment* **A** *to a variable* **a**, *there exists a finite sequence of structural normal form data parallel assignments which is equivalent, with respect to* **a**, *to* **A**.

Proof. By induction on the structure of A, appealing to the laws discussed earlier. Details are omitted. □

5 Conclusion

Data parallelism continues to be a significant feature of parallel computing. There is a need for a model of data parallel computation that is tractable and efficiently implementable. It has been shown [11] that correctness proofs of programs based on data parallel assignment are more tractable than their equivalents expressed in Fortran 90 and HPF; it remains to be shown how data parallel assignment may be efficiently implemented on a range of parallel architectures.

One possible approach is to derive an efficient implementation of a program expressed using data parallel assignment through the automated application of a sequence of correctness preserving transformations. This approach has been successfully applied in the derivation of efficient parallel implementations from functional specifications [2]. The laws presented in this paper provide the basis for many of the necessary transformation rules. It would, however, be necessary to define a set of special purpose transformations for carrying out optimizations for particular architectures. Alternatively, it may be possible to transform programs expressed using data parallel assignment into equivalent Fortran 90 or HPF versions and thereby exploit the sophisticated compilers that have been developed for these languages. This is a topic for further investigation.

References

1. K.M. Chandy and J. Misra. *Parallel Program Design: A Foundation.* Addison-Wesley, 1988.
2. M. Clint, S. Fitzpatrick, T.J. Harmer, P.L. Kilpatrick, and J.M. Boyle. A family of data-parallel derviations. In W. Gentzsch and U. Harms, editors, *Proceedings of High Performance Computing and Networking, Volume II, LNCS 797*, pages 457–462. Springer-Verlag, 1994.
3. E.W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.
4. D. Gries. *The Science of Programming.* Prentice-Hall International, 1981.
5. C.A.R. Hoare, I.J. Hayes, He Jifeng, C.C. Morgan, A.W. Roscoe, J.W. Sanders, I.H. Sorensen, J.M. Spivey, and B.A. Sufrin. Laws of programming. *Communications of the ACM*, 30(8):672–686, 1987.
6. C.A.R. Hoare and P.E. Lauer. Consistent and complementary formal theories of the semantics of programming languages. *Acta Informatica*, 3:135–153, 1974.
7. M. Metcalf and J. Reid. *Fortran 90 explained.* Oxford University Press, 1990.
8. R.H. Perrott. A language for array and vector processors. *ACM Transactions on Programming Languages and Systems*, 2:266–287, 1979.
9. A. Stewart. SIMD language design using prescriptive semantics. *BIT*, 28:639–650, 1988.
10. A. Stewart. An axiomatic treatment of SIMD assignment. *BIT*, 30:70–82, 1990.
11. A. Stewart. Reasoning about data-parallel array assignment. *Journal of Parallel and Distributed Computing*, 27:79–85, 1995.
12. P.J.L. Wallis. Some primitives for the portable programming of array and vector processors. *BIT*, 21:436–448, 1981.
13. J.P. Wray. *The Semantics of Synchronised Assignment.* PhD thesis, The Queen's University of Belfast, July 1992.
14. J.P. Wray. Algebraic laws and a normal form for data parallel assignment. Technical report, Department of Computer Science, The Queen's University of Belfast, May 1996.
15. J.P. Wray and A. Stewart. Correct translation of data parallel assignment onto array processors. *Formal Aspects of Computing*, 6:417–439, 1994.