Parallel languages and Compiler

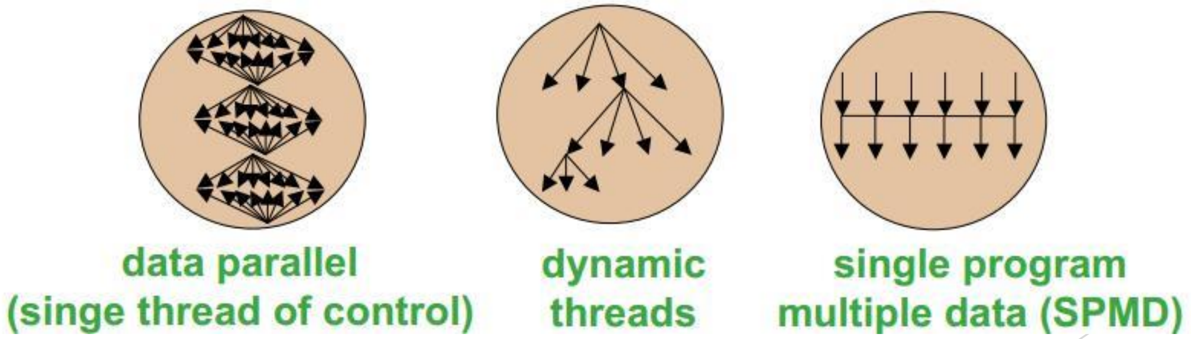
Dependence Analysis of Data Arrays

Parallel/Concurrent Languages

A concurrent language is defined as one which uses the concept of simultaneously executing processes or threads of execution as a means of structuring a program. A parallel language is able to express programs that are executable on more than one processor. Both types are listed as concurrency is a useful tool in expressing parallelism, but it is not necessary. In both cases, the features must be part of the language syntax and not an extension such as a library.

Picking a languages to use in order to write Parallel Programs includes a deeper thought, and a couple on initial questions. For example, different programs hide the machine in different ways. Hiding improves programmability (productivity), portability, while exposing gives programmers control to improve performance.

There are two components to picking a language- control and data (communication/sharing).



Dependence Analysis of Data Arrays

Parallelization or locality optimizations frequently reorder the operations executed in the original program. As with all optimizations, operations can be reordered only if the reordering does not change the program's output.

Two accesses, whether read or write, are clearly independent (can be reordered) if they refer to two different locations. In addition, read operations do not change the memory state and therefore are also independent.

We say that two accesses are data dependent if they refer to the same memory location and at least one of them is a write operation. To be sure that the modified program does the same as the original, the relative execution ordering between every pair of data-dependent operations in the original program must be preserved in the new program.

There are three flavors of data dependence:

- True dependence, where a write is followed by a read of the same location.
- Anti-dependence, where a read is followed by a write to the same location.
- Output dependence, which is two writes to the same location.

Definition of Data Dependence of Array Accesses

Let us consider two static accesses to the same array in possibly different loops. The first is represented by access function and bounds T = (F, f, B, b) and is in a d-deep loop nest; the second is represented by T' = (F ' , f , B ' , b ') and is in a rf'-deep loop nest.
These accesses are data dependent if
At least one of them is a write reference and There exist vectors i in Zd and i' in Zd' such that

(a) Bi > 0,

(b) B'i' > 0, and

(c) Fi f = F'i' f.

Since a static access normally embodies many dynamic accesses, it is also meaningful to ask if its dynamic accesses may refer to the same memory location.
To search for dependencies between instances of the same static access, we assume T — T' and augment the definition above with the additional constraint that i ^ i' to rule out the trivial solution.

Example:

Consider the following 1-deep loop nest:

```
for (i = 1; i < 10; i ) {
Z[i] = Z[i-1];
}
```

This loop has two accesses: Z[i - 1] and Z[i}; the first is a read reference and the second a write.

To find all the data dependences in this program, we need to check if the write reference shares dependence with itself and with the read reference:

1. Data dependence between Z[i - 1] and Z[i]. Except for the first iteration, each iteration reads the value written in the previous iteration. Mathematically, we know that there is a dependence because there exist integers i and %' such that

1 < i < 10, 1 < i' < 10, and i - 1 = i'.

There are nine solutions to the above system of constraints: (i = 2, %' - 1), (i = 3,z' = 2), and so forth.

2. Data dependence between Z[i] and itse//. It is easy to see that different iterations in the loop write to different locations; that is, there are no data dependencies among the instances of the write reference Z[i]. Mathematically, we know that there does not exist a dependence because there do not exist integers i and i' satisfying

1 < i < 10, 1 < i' < 10, i = i', and i ^ %'.

Notice that the third condition, i = i', comes from the requirement that Z[i] and Z[i'] are the same memory location The contradictory fourth condition, i ^ %', comes from the requirement that the dependence be nontrivial between different dynamic accesses.

It is not necessary to consider data dependences between the read reference Z[I -1] and itself because any two read accesses are independent.