Baldwin's thoughts on parallel programming

The quotes and ideas in this report are taken from the paper written by Douglas Baldwin and entitled "Why we can't program multiprocessors the way we're trying to do it now." The paper challenges the suitability of both imperative and declarative languages to parallel programming. The points he makes are often bold and somewhat controversial. Baldwin analyses how well different languages deal with data dependencies, data parallelism, granularity and generality.

Data dependencies

- "imperative languages are inherently sequential, being very deeply founded on the assumption that any two statements will be executed in some definite order relative to one another."
- "data dependencies are very difficult to detect in imperative languages. One of the reasons for this is that side effects have no locality."
- "the analysis [of data dependency] may be harder in object-oriented or modular languages because it has to cross supposedly inviolable object or module boundaries"
- "The mathematical formalisms on which these [declarative] languages are based generally disallow side-effects. Thus a variable's value is defined in exactly one place, and is never changed after being defined."

Data parallelism

- "In imperative languages, potentially data parallel computations are usually written as a loop over the appropriate data structures. This is one of the hardest forms in which to detect parallelism automatically."
- "the most general way of expressing a data parallel computation in a declarative language is through a recursive definition of some sort. Unfortunately, in all current declarative languages these recursive definitions introduce apparent data dependencies involving parameters to or results from the recursion."
- "Of all declarative languages, logic languages are best suited to describing data parallelism ... data parallelism is implicit."

Granularity

- "effectively using multiprocessors requires fairly coarse-grained parallelism in programs."
- "In many parallel programming systems the 'natural' granularity is too fine for use on multiprocessors. In some cases granularity can be coarsened by collecting several fine-grained processes into a single aggregate.

This approach necessarily serializes the fine-grained operations within each aggregate."

- "Reducing communication while retaining enough processes to provide significant parallelism requires that communication patterns in the original program be sparse. Unfortunately, fine-grained parallel programming systems do not always lead to sparse communication patterns."

Generality

- "A final requirement for languages that will support general-purpose programming is that they really be general purpose ... it occasionally happens that generality is achieved at the expense of easy parallelization."
- "Logic languages are a case in point ... many common operations cannot be implemented efficiently as logic programs ... arithmetic, input and output, etc."

Summary

- "software technology for parallel programming is in sad shape."
- "Two deep flaws ... the use of side-effect based model of computation ... imperative languages are ill suited for parallel programming ... the second problem is an inability to express data parallelism without using iteration or recursion ... particularly severe in declarative languages."

With the above points in mind we can analyse what definitive programming has to offer in terms of describing data dependencies, data parallelism, granularity and generality. In doing so we can get some idea of how suited definitive programming is for parallel programming.

Data dependencies A definitive language is like a declarative language in this respect. The mathematical formalism of actions disallow side-effects. The result of this is that actions may be performed in any order.

Data parallelism A definitive language is like a declarative language in this respect also. Data parallelism is implicit. Any number of actions whose guards are true may perform their redefinitions in parallel. This is similar to logic languages.

Granularity Parallelism tends to be fine-grained in definitive programs. Some way of allocating actions/agents and state to processors has to be found to make an efficient implementation on a multi-processor system.

Generality Definitive programming is general-purpose. Common operations, such as input-output and arithmetic, are accommodated within the paradigm without introducing side-effects.