Data-Centric Locality in Chapel

Ben Harshbarger (Chapel Implementer, Cray Inc.)

The Chapel programming language is designed with performance in mind, but the implementation has yet to deliver in practice for distributed memory programs. A significant cause of overhead is the insertion of communication idioms by the compiler. The compiler tends to be overly conservative with these idioms, and defaults to assuming that data is remote in order to ensure correctness. When the compiler incorrectly assumes that a variable is remote no actual communication occurs, but the generated idioms result in overhead and often prevent the back-end C compiler from optimizing. The work presented in this talk is a first step towards improving this situation.

In the Cray Chapel implementation the 'local' statement can be used to assert that no communication will occur within a block of code. The compiler uses this information to eliminate most of the overhead inside the block. This statement is a valuable tool used to improve performance, especially in the implementation of array distributions.

While useful, the 'local' statement is somewhat heavy-handed and lacks precision. The statement cannot be used to define locality for individual variables, arguments, or fields in aggregate types. It is also challenging to define its semantics when optimizations are present.

Class variables are a key case in which we use the 'local' statement to eliminate overhead. Class variables introduce communication overhead due to the fact that a class reference may point to local or remote data. Even if a user's code does not use classes directly, Chapels' arrays and domains are implemented using classes and are likely to be used indirectly. This leads to our first data-centric locality assertion - the ability to specify that a class variable is local.

For the 1.11 release of Cray Chapel, the "local field" pragma was introduced as a first step towards this functionality. This pragma allows developers to specify that a field in an aggregate type will share the same locale as the containing object. The overall effect is that class designers have more control over distributed performance. The performance benefits from this work are promising, and indicate that this direction should be valuable in the effort to improve distributed program performance.

Ongoing work in this area involves the development of a more robust language-level construct to provide more flexibility to users than a pragma allows for. This talk will present an overview of this work where the semantics, implementation, and performance of this construct will be discussed.