programming language. It also allows multiple views of the same program with different features highlighted and even the use of graphics in the presentation.

Once the code has been optimized, the code generator produces an executable source program in some concrete language, such as C. This source program is linked with the runtime system and executed on a given data set. The semantics of persistent data are build in to UHF; the compiler can examine the contents of a data file to help guide optimizations on a particular program. For example, some matrix optimizations depend on the size of an array (such as when and when not to tile); but the size of the array may be stored in the input file and not part of the program. In our system, the compiler could examine the input file and extract the size of the array and replace the variable with a constant. This could then allow more aggressive optimizations to be applied when generating a specific executable for the given input.

2 Internal Format

The UHF internal format is the core of our system. It uses a RISC-like model adapted from SUIF [2] and Omniware [1]. This model was used for its ability to map easily to modern RISC processors and the success of these systems in providing portability as well as performance. Operations are performed on scalars which are translated to local variables in the target system. These scalars can be thought of as an unlimited supply of registers. The compiler on the target platform is expected to use aggressive register scheduling specific to the host processor.

We extend this simple model with special operations to access arrays and records. An array represents a homogeneous container with indexed access and a record represents a heterogeneous container with associative access. Access is done via load and store operations. Finally, simple control flow instructions are adapted from the control flow instructions of SUIF. Scalars are stored by value in registers whereas arrays and records are considered to be references. In the code generator, a reference is attached before the contents are used and released after they are no longer needed. References passed as arguments into functions are assumed to already be attached. More details can be found in [4].

3 Code Generation

The current runtime system assumes a simple ordering of data (row-wise) on disk and manages the movement of data between disk and main memory. When a complex data layout is encountered, it is necessary to construct wrappers to translate a request into a disk transfer or cache hit. This code is placed into the extended Comanche runtime library. The compiler is only concerned with arrays and records and when to map these into memory.

3.1 Standard Transformations

Several standard transformations taken from high performance compiling are essential to obtaining acceptable performance in our system. The principal research drawn from is locality improvement. A detailed analysis of these transformation and when they can be applied can be found in [3].
Loop interchange will reorder loops with the goal of having the inner most loop index appear in the unit stride dimension of each array reference (rightmost in C and leftmost in Fortran).

Loop fusion is used to avoid extra traversals of large data structures. Two loops are fused into one loop sometimes with the insertion of conditional statements to normalize the number of iterations.

Tiling or iteration space blocking changes the iteration order to access matrices in submatrix order. This is often used when it is not possible to align each of the inner most loops with the unit stride dimension such as in the case of matrix multiplication and transpose.

Strip mining is a technique that can be applied to any loop where the loop is broken into two loops: an outer loop that iterates over blocks and the inner loop that iterates within the blocks. This is used to move code outside of the inner loop so that it will not be executed too frequently.

3.2 Attach and Release

The attach and release directives combined with the controlled access to memory allow aliases to be controlled. If aliases are not restricted, we cannot guarantee correct execution without indirecting each and every every single access to memory. Specifically, references to the same remote data element must resolve to the same cached element.

As illustration consider the program in Figure 2. We will assume that the array, A, cannot be held completely in local memory and must treated as a remote memory access. f1, f2, and f3 are deliberately left ambiguous in order to illustrate the problems with the general case.

We begin by transforming the program to isolate individual array accesses (Figure 3). This is in accord with the UHF internal format. A layer of indirection is imposed on each access to the array so that the data are moved between remote and local memory (Figure 4). Note that this code uses a minimal amount of storage: five doubles, a pointer, and a remote store record. The read and write statements have the same semantics as those found in the C standard I/O libraries.

However, each access to read and write invokes a call on the runtime system which can take hundreds of cycles to complete. It is clear that inclusion of runtime system calls inside of inner loops will severely degrade performance. Instead, we might wish to write code to read entire rows in a single call (Figure 5). Unfortunately, this is incorrect if any two of f1(i), f2(i), and f3(i) are equal or overlap since we will be reading and writing different copies of the same remote data. We can determine overlap using data dependence analysis but these techniques require that index expressions be affine functions of the index variables; we are not able to solve the general case that is presented.

Our solution introduces two new instructions: attach and release. These instructions tell the runtime that the region is to be mapped into memory and that an address to the cached data is to be returned. The new version is presented in Figure 6. The runtime will return the same pointer if one of the ambiguous functions is equal to any of the others. Simple reference counting can be used to keep track of how many outstanding release operations are left.

Further enhancements can be made such as adding a read only flag for wa2 and wa3. The compiler can determine this by examination of the code which shows that wa2 and wa3 are not