BRIEF DESCRIPTION OF THE PROJECTS :

1. Weaves:

Parallel programing has received new found importance with the advent of multicore systems. Parallel programing models can be broadly classified into two categories – shared and distributed memory. Programs intended for distributed memory applications are implemented as processes. Communication between processes takes place through IPC. Shared memory programs operate under a single address space and are modeled through threads. Most real world applications are neither truly distributed nor truly shared. Execution entities may share parts of the code and data at some point of time during execution and may be completely independent during other times. Neither processes nor threads could model this behavior completely. Weaves attempts to solve this problem by providing arbitrary sharing or selective sharing of code and data through a simple interface which doesn't impose any restrictions on the programmer.

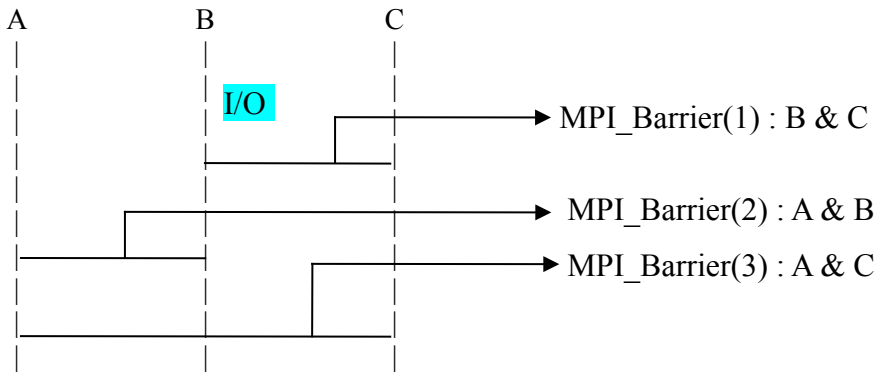
The idea of selective sharing can also be viewed as an idea of data separation. Thread-local storage was designed to suit programing needs where during extended periods of execution threads needed to have independent copies of global variables and thus achieve data separation among them. We use the idea of thread-local storage to provide data separation among threads. Global data is converted to thread local data and API's are exposed through our framework which allows the threads to access each other's thread local storage and thus share data when needed at runtime.

The idea of selective sharing is implemented at the IR (intermediate format) level generated by LLVM (A compiler infrastructure tool from University of Illinois Urbana Champaign). The IR format is compiler independent and avoids architecture level complexities. It exposes API's which help us to convert global data to thread local and instrument variable access with function calls which allow threads to access each other's thread local storage data at runtime.

2.MPI-IO Application Replay:

Evaluation of storage systems is one of the critical needs of scientific community. Various techniques have been devised for evaluation. One of them is by running industry standard I/O benchmarks e.g TPC benchmarks, Postmark, Iozone, etc. Unfortunately these benchmarks are complex and difficult to run.

Replaying I/O traces from a well-known application provides a alternative to these benchmarks. The replayer reads the traces and tries to issue I/O at the same rate as if the original application was being run. The rate at which I/O is issued by each node is dependent upon it's dependencies with other nodes. E.g. Node A may issues a I/O after it receives data from Node B or Node A issues I/O after issuing a barrier along with node B. Thus these calls which cause dependency among the nodes must be correctly determined and accounted for while replaying the trace. Based on our analysis we figured out that all synchronization operations need not be recorded. Only those synchronization operations need to be tracked which are directly co-related to an I/O. The diagram below illustrates the point.



B performs I/O and is involved in a barrier call (1) with C. This barrier needs to be recorded because the amount

of time C spends in the barrier (waiting for B) is dependent upon the time taken by the I/O operation which varies for each file system. The next barrier call (2) which B performs with A also needs to be recorded as amount of time A spends in barrier (waiting for B) is dependent upon the time taken by I/O operation to complete. The last barrier call (3) need not be recorded as both A and C will hit the barrier at the same time and thus they need not wait for each other. This technique helps us to cut down the size of the log file and speed up the replay.