A Framework for Genetic Algorithms for GPGPUs

The project I am presenting is a generic framework which I have designed and implemented for enhancing the performance of Genetic Algorithms on GPGPU. After making a study of GAs it was noticed that all of them followed a similar cycle of *Fitness – Selection - Crossover – Mutation* phases. Only Fitness evaluation phase is highly compute intensive and is generally the bottleneck of performance. After considering all these important points, the design of the framework was concluded.

The framework is not any algorithm specific. It includes all the phases of a typical GA, Initial Population Generation phase, Fitness Function phase, Selection phase, Crossover phase and Mutation phase. 3 out of these 5 phases have been implemented on the GPU. Initial Population Generation and Selection are done at CPU. The fitness function given by user is used at the GPU. The input to the framework is the general parameters like crossover probability, Mutation prob., Population size, Chromosome size, Gene type (int/char/float/user defined) and a **sequential** fitness function which can be written by any normal programmer. The framework proceeds by first taking all the input arguments and entering the first phase i.e. Population Generation phase. Since GPUs do not have efficient random number generation techniques, this phase is done on CPU. After the required population is generated the Fitness calculation phase is activated. Here, one thread per chromosome in the population is created and the GPU is called to evaluate the fitness function written by user* on the chromosomes in parallel on the GPU. After this phase is completed, the fitness values are copied from the GPU to the CPU and the Selection phase is activated. This phase has been implemented on the CPU as well. The reason being I want to give user the freedom to extract more information after the Fitness phase. If the user wants to add code before or after the Selection phase he can do so. Also if the user wants to change the Selection algorithm he will be able to do so easily. Because the user will be prompted to write only sequential code this phase is done on CPU. Selection phase currently involves filtering out the chromosomes which do not satisfy the minimum fitness value as given input by the user. Chromosomes above that value are preceded to the next phase. Crossover is the next phase. Before crossover, Roulette Wheel Selection is implemented to bring mating parents successively in the population array i.e. on basis of fitness values the potential parents are placed sequentially in the population array. This is done on CPU. As a result all the

parents who will mate are placed next to each other. Now the number of threads created is half the size of existing population. Each thread will pick up two parents (i^{th} thread will work on $2*i^{th}$ and $2*i+1^{th}$ parent) and crossover them at the GPU. The random numbers required for selection of crossover point are passed as an array from the CPU to the GPU. Thus the chromosomes are crossed over at the GPU. Now we have new children and the parents have been destroyed. There is also an option such that the fittest members of every generation always make it to the next generation. After every phase user can add code and work on the population and fitness values. The Mutation phase is executed along with the crossover phase. Since the new born children are at the GPU during the crossover phase, I perform the mutation there itself. Another two arrays of random numbers was sent along, one for deciding the probability and other for choosing mutation point. If the probability is satisfied the mutation is done at the GPU in that phase, thus saving some time and effort. This goes on till the number of generations is completed. Thus both CPU and GPU are used effectively to obtain maximum parallelism in the GA implementation. Memory lookups are also provided in the kernel code*. So user can lookup information in the fitness function and perform calculations. The fitness function signature involves only a pointer to a chromosome and the data which user wants to lookup and the return value SHOULD be a float.

The effort in the project is basically to bring multi-core programming close to layman programmers and they should be able to use the hardware without having to learn the CUDA API. With such an effort the user only has to write sequential functions and the job of parallel execution and result accumulation is done by the framework itself. So end-user is free of the learning curve. Also the test results show a tremendous speedup as compared to MIT Galib (http://lancet.mit.edu/ga/), also the quality of results was comparable and using only sequential fitness functions the GA was run on GPU Tesla T10 processor. Now the search space chosen initially can be huge and still time consumed will be very less.

Note: *Further details can be found in the README

The next step in the process of 'evolution' of this technique is the inclusion of 'user-defined genes' in the framework. In the next version, the user will be able to define his own type of gene as a *struct* but will have to write population generator, crossover and mutation function for such a chromosome. But the fitness calculation and crossover/mutation will be performed on the device in parallel.