CATEGORY: MACHINE LEARNING & DEEP LEARNING - ML06
POSTER P5301
CONTACT NAME Witold Rudnicki: W.Rudnicki@icm.edu.pl
GPU TECHNOLOGY CONFERENCE

Feature selection @GPU

Witold R. Rudnicki*, Szymon Migacz, Andrzej Sułecki, Łukasz Grad,
Magdalena Zaremba, Paweł Tabaszewski, Antoni Rościszewski
Interdisciplinary Centre for Mathematical and Computational Modelling, University of Warsaw
Pawińskiego 5a, 02-106 Warsaw Poland
*Corresponding author, e-mail: W.Rudnicki@icm.edu.pl

Problem

Datasets in molecular biology have number of features reaching several thousands (gene expression, proteomics) and even millions (SNPs). Identification of the features that truly contribute to the studied phenomena is a key step, necessary for their understanding.

The number of features is so large that only univariate significance tests for each variable are routinely performed, hence any effects that depend on interactions of several features may be overlooked.

Solution

We have developed very fast CUDA based engine that allows for exhaustive multidimensional searches of all combinations of features that may lead to statistically significant relationships between **k-tuple of descriptive variables** and a decision variable, with k = {2,3,4,5}. We present the service based on this engine, that aims at performing feature selection that takes into account joint influence of subsets features in the studied phenomena for data sets described with very large number of features. The first module is devoted to search of epistatic interactions due to structural variations in genomes, nevertheless, the algorithm is easily extensible to more general problems and this work is under development.

Values of the descriptive variables are randomly discretised into m={2,3,4} intervals. Hence for each k-tuple of variables objects are assigned to voxels in k-dimensional subspace spanned on these variables. The number of objects with each decision class is computed for each voxel and the statistical test is performed to check whether such division leads to significant gain of information on the decision.

The procedure is performed multiple times for each k-tuple of variables using different discretisations of variables. The GPU engine is used to count the number of objects of both classes in each voxel for each k-tuple of variables and then compute the value of the appropriate statistics.

Pseudocode (2d case, scalar version)

```
//Build bit vector representations for all variables
BitVec<-BuildVectorRepresentations()
for i = 1 to Nvar
  for j = i+1 to Nvar
     //Compute counters for 0s
     for k1 = 1 to K-1 for k2 = 1 to K-1 Counter_0[k1,k2] = 0;
     for n = 1 to Nobj_0/32 // 32 objects stored in single int,
        for k1 = 1 to K-1
           for k2 = 1 to K-1
              Counter_0[k1,k2] += PopCount(BitVec[n,k1] & BitVec[n,k2])
     //Compute counters for 1s
     for k1 = 1 to K-1 for k2 = 1 to K-1 Counter_1[k1,k2] = 0;
     for n = Nobj_0/32 to Nobj_1/32
     // We assume multiply of 32 objects in each class
        for k1 = 1 to K-1
           for k2 = 1 to K-1
              Counter_1[k1,k2] += PopCount(BitVec[n,k1] & BitVec[n,k2])
        for k1 = 1 to K-1
           locVar1_val0 = locVar1_val1 = locVar2_val0 = locVar2_val1 = 0
           for k2 = 1 to K-1
              locVar1_val0 += Counter_0[k1,k2]
              locVar1_val1 += Counter_1[k1,k2]
              locVar2_val0 += Counter_0[k2,k1]
              locVar2_val1 += Counter_1[k2,k1]
           Counter_0[k1,K] = SumVar0[i,k1]-locVar1_val0
           Counter_1[k1,K] = SumVar1[i,k1]-locVar1_val1
           Counter_0[K,k1] = SumVar0[j,k1]-locVar2_val0
           Counter_1[K,k1] = SumVar1[j,k1]-locVar2_val1
        locVar1_val0 = locVar1_val1 = 0
        for k1 = 1 to K-1
           locVar1_val0 += Counter_0[k1,K]
           locVar1_val1 += Counter_1[k1,K]
        Counter_0[K,K] = SumVar0[i,K]-locVar1_val0
        Counter_1[K,K] = SumVar1[i,K]-locVar1_val1
     ComputeStatistics()
```

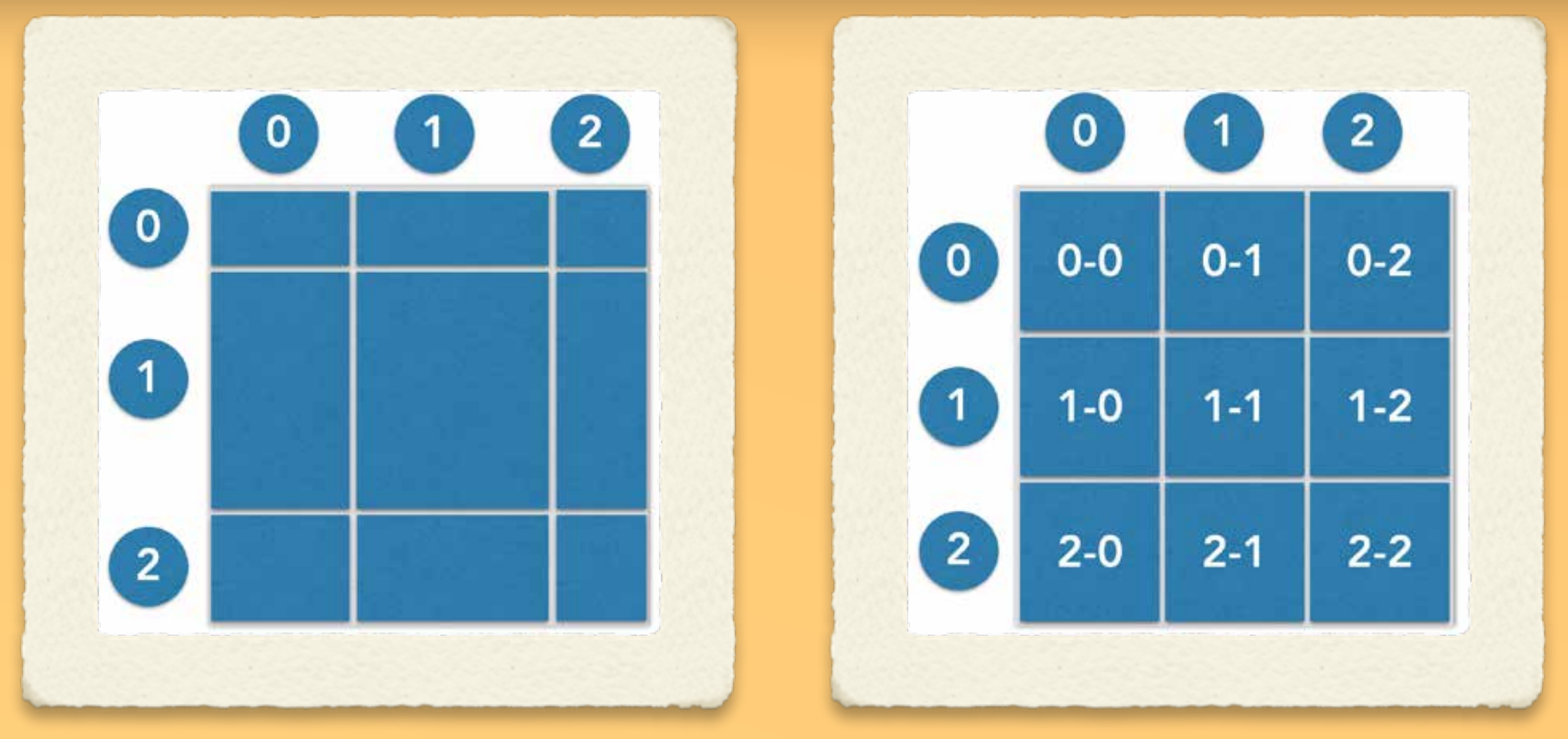


Figure 1. Discretisation of the configuration space for two variables. The statistical tests are performed on GPU using the engine developed for computing multidimensional GWAS analyses.

Computations are performed in two stages. In the first stage the number of objects of each class is computed for each voxel.

In this stage the counters for each voxel can be computed separately for each class, what is facilitated by sorting objects by decision variable. Discretised value of each variable is coded as bit-vector. For example when variable is discretised to three values; the corresponding representation is
0 -> {1,0,0}, 1 -> {0,1,0}, 2 -> {0,0,1} , see Figure 1.
For each object the location of the particular voxel is checked using AND operator on corresponding bit-vectors and counters for all voxels are updated. The result is non-zero for precisely one voxel. One should note that it takes m^k operations. It may seem wasteful at first, however, thanks to this representation two optimisations are possible:

- bits corresponding to each value are stored in bit-vectors, each bit-vector corresponds to all objects, hence one can use bit-wise operation and perform the operations for 32 objects in parallel,
- bit vectors are very compact representation, only two bits per object are required.

Moreover, not all counters need to be computed. One can precompute the number of objects corresponding to each discrete value of each variable, and then m^{k-1} counters need to be computed directly from data, remaining values can be computed using counters and known values for single variables.

Once all counters are computed for both classes the statistics are computed.

CUDA Implementation

The engine is currently implemented for 2-dimensional problems with 2, 3 and 4 values per variable. Each CUDA thread calculates counters and statistics for one pair of features, counters are stored in registers.

Pseudocode for CUDA version

```
unsigned counter[2][2] = {0};
for n = 0 to Nobj_0/32
   if (n % 4 == 0)
      __syncthreads()
      prefetch bitVectors to shared memory
      __syncthreads()
   for k1 = 1  to K-1
      for k2 = 1 to K-1
         counter[k1,k2] += PopCount( shBitVec[n%4, k1, threadIdx.x] &
shBitVec[n%4, k2, threadIdx.y])
ReverseBits() // revers bits in each element of "counter" array
for n = 0 to Nobj_1/32
   if (n % 4 == 0)
      __syncthreads()
      prefetch bitVectors to shared memory
      __syncthreads()
   for k1 = 1  to K-1
      for k2 = 1 to K-1
         counter[k1,k2] += PopCount( shBitVec[n%4, k1, threadIdx.x] &
shBitVec[n%4, k2, threadIdx.y])
RecoverOriginalCounters()
// recover  original values in counter[2][2] array
// and copy to counter_0[2][2] and counter_1[2][2] arrays
PopulateFullCounterArray()
//Populate full counter_0[3][3] and counter_1[3][3] arrays
CalculateStatistics()
```

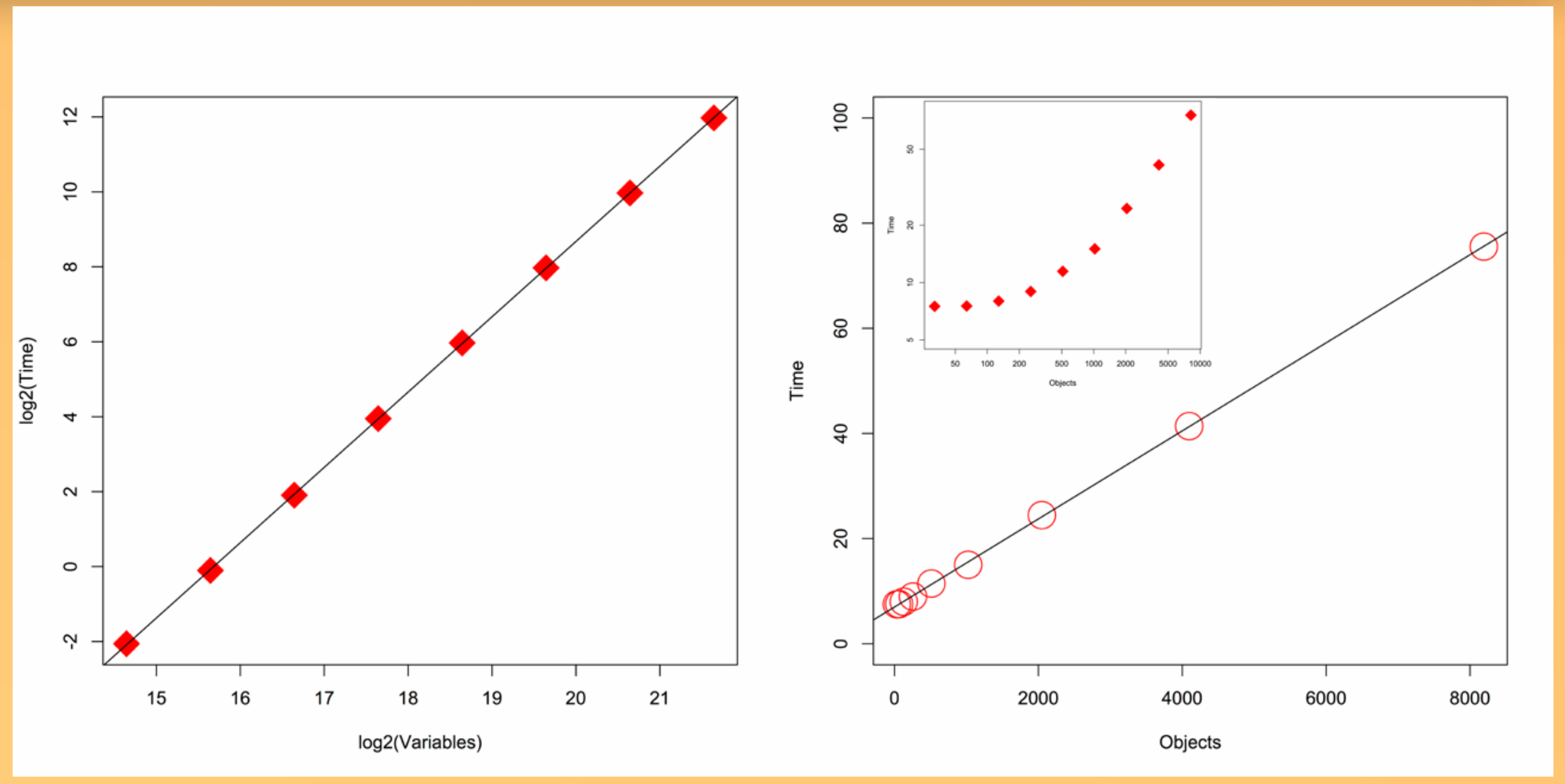


Figure 2. Scaling of the task with increasing number of variables (left) and increasing number of objects (right). Inset on the right panel displays the same data in logarithmic scale. Lines of linear fit to data are displayed in both panels. In the left panel is $r^2=1$, in the right panel $r^2=0.9999$.

Default size of CUDA block is 16x16 threads. In general case for one iteration over objects (i.e. 32 consecutive objects stored as bit vectors) we need to read 2*(16+16) integers from global memory, let us notice that for example the threads with the same value of threadIdx.x require the same bit vector corresponding to a specific feature. Therefore to reuse bit vectors and reduce the required bandwidth to global memory we are using shared memory as a block-level cache.

Performance

In the case of 2-dimensional problem with three values per variable, corresponding to the analysis of epistasis in SNP data, computing counters and computation of statistics takes roughly the same time when number of objects in the dataset is about 1000.

The scaling is nearly perfectly quadratic with increasing number of variables and linear with the increasing number of objects, see Figure 2. For 1024 objects and 204 800 variables total computation time takes 15.5 seconds on GeForce Titan Black, when single statistics is computed. Computations for 1024 objects and 3 276 800 take 4018 s.

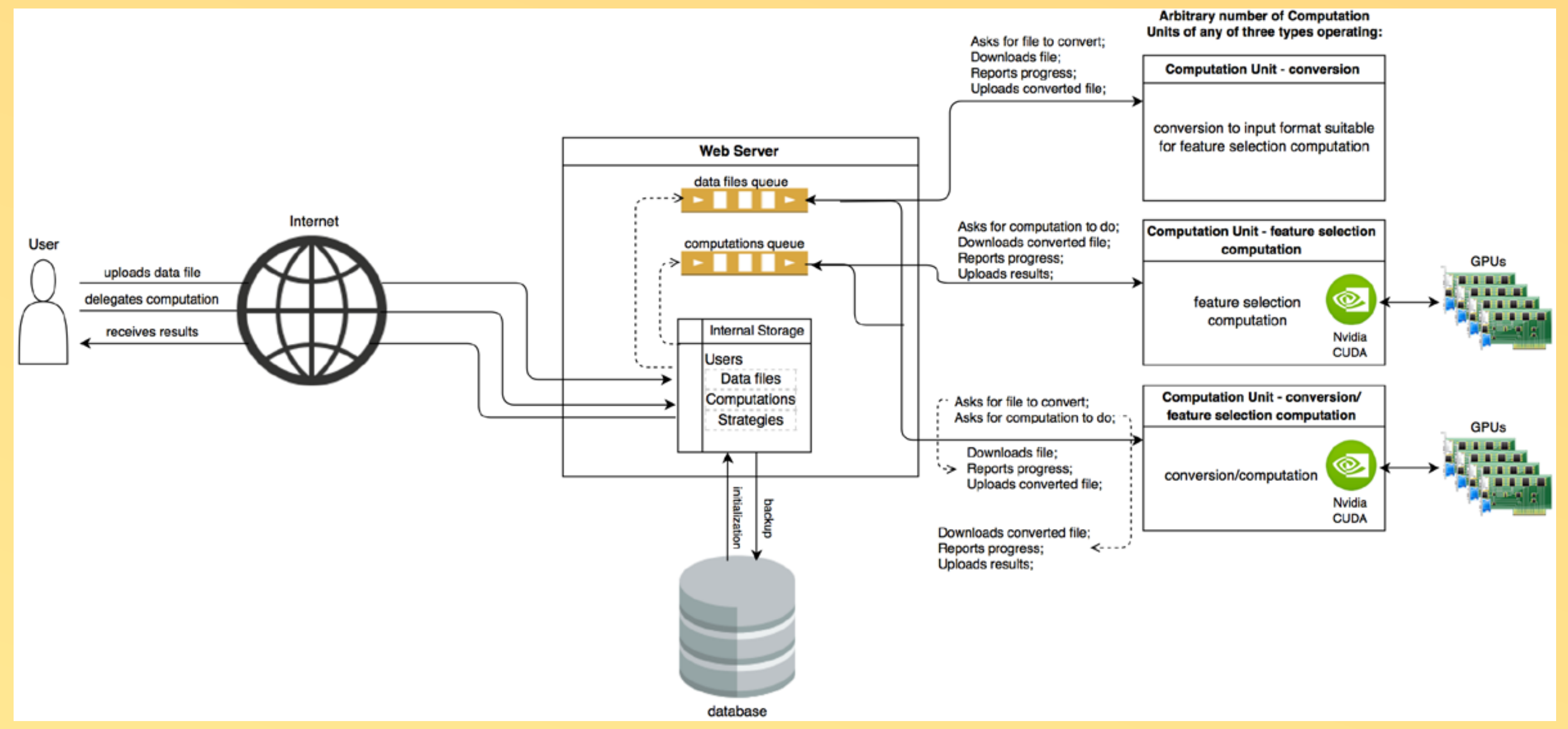
The GPU engine performance is 1.35 $*10^9$ model evaluations per sec.

With such performance it is feasible to compute 2-dimensional GWAS analysis for 10 mln SNPs on a single GPU card. Current implementation is capable of using multiple GPUs on the same workstation, the GPUs need not to be identical. Tests have been performed for all combinations of GeForce 480 GTX, Tesla K20 and GeForce Titan Black.

Web Server *(FeatureSelector.icm.edu.pl)*

Server is available (starting from March 1 2015) for analysis of Genome Wide Associated Studies, subsequent modules will be added at later times. The web server is developed using GO language. It serves needs of two classes of users. Occasional users who can submit their data without registration, are limited to datasets not larger than 2 GB and the results are retained for 48 hours. The space available for registered users is limited to 20GB and is permanent. The server consists two layers - web server and and computational engines. The Web server is responsible for interactions with users, storage and transfers of data, and managing computations. Computations are performed on remote servers.

Figure 3. Schematic representation of the Feature Selection Server.


User
uploads data file
delegates computation
receives results
Internet
Web Server
data files queue
computations queue
Internal Storage
Users
Data files
Computations
Strategies
initialization
backup
database
Arbitrary number of Computation Units of any of three types operating:
Asks for file to convert; Downloads file; Reports progress; Uploads converted file;
Computation Unit - conversion
conversion to input format suitable for feature selection computation
Asks for computation to do; Downloads converted file; Reports progress; Uploads results;
Computation Unit - feature selection computation
feature selection computation
Nvidia CUDA
GPUs
Asks for file to convert; Asks for computation to do;
Downloads file; Reports progress; Uploads converted file;
Downloads converted file; Reports progress; Uploads results;
Computation Unit - conversion/ feature selection computation
conversion/computation
Nvidia CUDA
GPUs