Quantum Computer as Data Parallel Processing Simulation on GPU

Alexander SOIGUINE[1]

Abstract: Geometric Algebra formalism opens the door to developing a theory deeper than conventional quantum mechanics. Generalizations, stemming from implementation of complex numbers as geometrically feasible objects in three dimensions, unambiguous definition of states, observables, measurements, Maxwell equations solution in those terms, bring into reality a kind of physical fields spreading through the whole three-dimensional space and values of the time parameter. The fields can be modified instantly in all points of space and time values, thus eliminating the concept of cause and effect, and perceiving of one-directional time. In the suggested theory all measured observable values get simultaneously available all together, not through looking one by one. In this way quantum computer appeared to be a kind of analog computer keeping and instantly processing information by and on sets of objects possessing an infinite number of degrees of freedom. As practical implementation, the multithread GPUs bearing the CUDA language functionality allow to simultaneously calculate observable measurement values at a number of space/time discrete points only restricted by the GPU threads capacity.

1. Geometric algebra type of analog modeling computer

An analog computer is generally a type of computing device that uses the continuous variation aspect of physical phenomena to model the problem being solved [1]. One special type of physical phenomena to model problems is considered below.

The circular polarized electromagnetic waves following from the solution of Maxwell equations in free space done in geometric algebra terms, [2], [3], are the electromagnetic fields of the form:

$$F = F_0 exp[I_S(\omega t - k \cdot r)] \qquad (1.1)$$

which should be solution of

$$(\partial_t + \nabla)F = 0 \qquad (1.2)$$

Solution of (1.2) must be the sum of a vector (electric field e) and bivector (magnetic field $I_3 h$):

$$F = e + I_3 h$$

with some initial conditions:

$$e + I_3 h|_{t=0,\vec{r}=0} = F_0 = e|_{t=0,\vec{r}=0} + I_3 h|_{t=0,\vec{r}=0} = e_0 + I_3 h_0$$

For a given plane S in (1.1), the solution of three-dimensional Maxwell equation (1.2) has two options:

- $F_+ = (e_0 + I_3 h_0) exp[I_S(\omega t - k_+ \cdot r)]$, with $\hat{k}_+ = I_3 I_S$, $\hat{e}\hat{h}\hat{k}_+ = I_3$, and the triple $\{\hat{e}, \hat{h}, \hat{k}_+\}$ is right hand screw oriented, that's rotation of $\hat{e}$ to $\hat{h}$ by $\pi/2$ gives movement of *right hand screw* in the direction of $k_+ = |k| I_3 I_S$;

[1]Website: https://soiguine.com; E-mail: alex@soiguine.com

- $F_- = (e_0 + I_3 h_0)exp[I_S(\omega t - k_- \cdot r)]$, with $\hat{k}_- = -I_3 I_S$, $\hat{e}\hat{h}\hat{k}_- = -I_3$, and the triple $\{\hat{e}, \hat{h}, \hat{k}_-\}$ is left hand screw oriented, that's rotation of $\hat{e}$ to $\hat{h}$ by $\pi/2$ gives movement of *left hand screw* in the direction of $k_- = -|k| I_3 I_S$ or, equivalently, movement of *right hand screw* in the opposite direction, $-k_-$;

where e_0 and h_0, initial values of e and h, are arbitrary mutually orthogonal vectors of equal length, lying on the plane S. Vectors $k_\pm = \pm|k_\pm| I_3 I_S$ are normal to that plane. The length of the "wave vectors" $|k_\pm|$ is equal to angular frequency ω.

Maxwell equation (1.2) is a linear one. Then any linear combination of F_+ and F_- saving the structure of (1.1) will also be a solution.

Let's write:

$$\begin{cases} F_+ = (e_0 + I_3 h_0)exp[I_S\omega(t - (I_3 I_S)\cdot r)] = (e_0 + I_3 h_0)exp[I_S\omega t]exp\big[-I_S[(I_3 I_S)\cdot r]\big] \\ F_- = (e_0 + I_3 h_0)exp[I_S\omega(t + (I_3 I_S)\cdot r)] = (e_0 + I_3 h_0)exp[I_S\omega t]exp\big[I_S[(I_3 I_S)\cdot r]\big] \end{cases} \tag{1.3}$$

Then for arbitrary (real[2]) scalars λ and μ:

$$\lambda F_+ + \mu F_- = (e_0 + I_3 h_0)e^{I_S\omega t}\left(\lambda e^{-I_S[(I_3 I_S)\cdot r]} + \mu e^{I_S[(I_3 I_S)\cdot r]}\right) \tag{1.4}$$

is solution of (1.2). The item in the second parenthesis is weighted linear combination of two states (wave functions, g-qubits [4], [5]) with the same phase in the same plane but opposite sense of orientation. The states are strictly coupled, entangled if you prefer, because bivector plane should be the same for both, does not matter what happens with that plane.

Arbitrary linear combination (1.4) can be rewritten as:

$$\lambda e^{I^+_{Plane}\varphi^+} + \mu e^{I^-_{Plane}\varphi^-} \qquad (1.5),$$

with

$$\varphi^\pm = \cos^{-1}\left(\frac{1}{\sqrt{2}}\cos\omega(t \mp [(I_3 I_S)\cdot r])\right),$$

$$I^\pm_{Plane} = I_S\frac{sin\omega(t \mp [(I_3 I_S)\cdot r])}{\sqrt{1 + sin^2\omega(t \mp [(I_3 I_S)\cdot r])}} + I_{B_0}\frac{cos\omega(t \mp [(I_3 I_S)\cdot r])}{\sqrt{1 + sin^2\omega(t \mp [(I_3 I_S)\cdot r])}} + I_{E_0}\frac{sin\omega(t \mp [(I_3 I_S)\cdot r])}{\sqrt{1 + sin^2\omega(t \mp [(I_3 I_S)\cdot r])}}$$

The triple of unit value basis orthonormal bivectors $\{I_S, I_{B_0}, I_{E_0}\}$ is comprised of the I_S bivector, dual to the propagation direction vector; I_{B_0} is dual to initial vector of magnetic field; I_{E_0} is dual to initial vector of electric field. The expression (1.5) is linear combination of two geometric algebra states, g-qubits.

Linear combination of the two equally weighted basic solutions of the Maxwell equation F_+ and F_-, $\lambda F_+ + \mu F_-$ with $\lambda = \mu = 1$ reads:

---

[2] Remember, in the current theory scalars are real ones. "Complex" scalars have no sense.

$$\lambda F_+ + \mu F_-|_{\lambda=\mu=1} = 2\cos\omega[(I_3 I_S)\cdot r]\left(\frac{1}{\sqrt{2}}\cos\omega t + I_S\frac{1}{\sqrt{2}}\sin\omega t + I_{B_0}\frac{1}{\sqrt{2}}\cos\omega t + I_{E_0}\frac{1}{\sqrt{2}}\sin\omega t\right) \quad (1.6)$$

where $\cos\varphi = \frac{1}{\sqrt{2}}\cos\omega t$ and $\sin\varphi = \frac{1}{\sqrt{2}}\sqrt{1+(\sin\omega t)^2}$. It can be written in standard exponential form $\cos\varphi + \sin\varphi\, I_B = e^{I_B\varphi}$.[3]

I will call such kind of g-qubits *spreons* (or sprefields) because they spread over the whole three-dimensional space for all values of time and, particularly, instantly change under Clifford translations over the whole three-dimensional space for all values of time, along with the results of measurement of any observable.

2. CUDA GPU simulation of the analog modeling computer

Measurement is by definition the result of action of operator, namely state, wave function, written in the form of g-qubit $(\alpha + I_S\beta)$ [5], on an observable C:

$$(\alpha - I_S\beta)C(\alpha + I_S\beta) = (\widetilde{\alpha + I_S}\beta)C(\alpha + I_S\beta)$$

Take as first example the case of observable as a vector expanded in $\{I_3 I_S, I_3 I_{B_0}, I_3 I_{E_0}\} \equiv \{e_1, e_2, e_3\}$, basis vectors dual to bivectors $\{I_S, I_{B_0}, I_{E_0}\}$:

$$C = c_1 e_1 + c_2 e_2 + c_3 e_3$$

Measure it with wave function (1.6):

$$4cos^2\omega[(I_3 I_S)\cdot r]\left[\left(\frac{1}{\sqrt{2}}\cos\omega t - I_S\frac{1}{\sqrt{2}}\sin\omega t - I_{B_0}\frac{1}{\sqrt{2}}\cos\omega t - I_{E_0}\frac{1}{\sqrt{2}}\sin\omega t\right)\right] C\left[\left(\frac{1}{\sqrt{2}}\cos\omega t + I_S\frac{1}{\sqrt{2}}\sin\omega t + I_{B_0}\frac{1}{\sqrt{2}}\cos\omega t + \frac{1}{\sqrt{2}}I_{E_0}\sin\omega t\right)\right] = 2cos^2\omega[(I_3 I_S)\cdot r]\left[\left(\cos\omega t - I_S\sin\omega t - I_{B_0}\cos\omega t - I_{E_0}\sin\omega t\right)\right]\left(c_1 I_3 I_S + c_2 I_3 I_{B_0} + c_3 I_3 I_{E_0}\right)\left[\left(\cos\omega t + I_S\sin\omega t + I_{B_0}\cos\omega t + I_{E_0}\sin\omega t\right)\right]$$

The result is:

$$4cos^2\omega[(I_3 I_S)\cdot r][c_3 e_1 + (c_1\sin 2\omega t + c_2\cos 2\omega t)e_2 + (c_2\sin 2\omega t - c_1\cos 2\omega t)e_3] \quad (2.1)$$

Geometrically that means that the measured vector is rotated by $\frac{\pi}{2}$ in the I_{B_0} plane, such that the $c_3 e_3$ component becomes orthogonal to plane I_S and remains unchanged. Two other vector components became orthogonal to I_{B_0} and $I_{E_0}$and continue rotating in I_S with angular velocity $2\omega t$. The factor $4cos^2\omega[(I_3 I_S)\cdot r]$ defines the dependency of that transformed vector values through all points of the three-dimensional space.

Similar example of measurement is that of the g-qubit type observable $C_0 + C_1 B_1 + C_2 B_2 + C_3 B_3$ (actually Hopf fibration) by a state $\alpha + \beta_1 B_1 + \beta_2 B_2 + \beta_3 B_3$ [4]:

[3] Good to remember that the two basic solutions F_+ and F_- differ only by the sign of $I_3 I_S$, which is caused by orientation of I_S that in its turn defines if the triple $\{\hat{E}, \hat{H}, \pm I_3 I_S\}$ is right-hand screw or left-hand screw oriented.

$$C_0 + C_1B_1 + C_2B_2 + C_3B_3 \xrightarrow{\alpha+\beta_1B_1+\beta_2B_2+\beta_3B_3} C_0 + \left(C_1[(\alpha^2+\beta_1^2)-(\beta_2^2+\beta_3^2)] + 2C_2(\beta_1\beta_2-\alpha\beta_3) + 2C_3(\alpha\beta_2+\beta_1\beta_3)\right)B_1 + \left(2C_1(\alpha\beta_3+\beta_1\beta_2) + C_2[(\alpha^2+\beta_2^2)-(\beta_1^2+\beta_3^2)] + 2C_3(\beta_2\beta_3-\alpha\beta_1)\right)B_2 + (2C_1(\beta_1\beta_3-\alpha\beta_2) + 2C_2(\alpha\beta_1+\beta_2\beta_3) + C_3[(\alpha^2+\beta_3^2)-(\beta_1^2+\beta_2^2)])B_3$$

with:

$B_1 = I_S$, $B_2 = I_{B_0}$, $B_3 = I_{E_0}$, $\alpha = 2\cos\omega[(I_3I_S)\cdot r]\frac{1}{\sqrt{2}}\cos\omega t$, $\beta_1 = 2\cos\omega[(I_3I_S)\cdot r]\frac{1}{\sqrt{2}}\sin\omega t$, $\beta_2 = 2\cos\omega[(I_3I_S)\cdot r]\frac{1}{\sqrt{2}}\cos\omega t$, $\beta_3 = 2\cos\omega[(I_3I_S)\cdot r]\frac{1}{\sqrt{2}}\sin\omega t$

gives a G_3^+ element spreading through the three-dimensional space for all values of time parameter t:

$$4cos^2\omega[(I_3I_S)\cdot r]\left[C_0 + C_3I_S + (C_1\sin 2\omega t + C_3\cos 2\omega t)I_{B_0} + (C_2\sin 2\omega t - C_1\cos 2\omega t)I_{E_0}\right] \quad (2.2)$$

The current approach transcends common quantum computing schemes since the latter are principally based on qubit entanglement (whatever it is) and thus have tough problems of creating large sets of entangled qubits. In the current scheme any test observable can be placed anywhere into continuum of the (t, r) dependent values of the spreon state. The above formulas (2.1) and (2.2) give the results of measurements simultaneously for all points (t, r).

The sprefield hardware requires special implementation as a photonic/laser device that does not exist yet. Instead, we have a very convenient equivalent simulation scheme where the amount of simultaneously available space/time points of observable measured values is only restricted by the overall available Nvidia GPU number of threads.

Consider the case of measuring a vector when the three parallel calculated measured by the sprefield vector components $4cos^2\omega[(I_3I_S)\cdot r]c_3$, $4cos^2\omega[(I_3I_S)\cdot r](c_1\sin 2\omega t + c_2\cos 2\omega t)$ and $4cos^2\omega[(I_3I_S)\cdot r](c_2\sin 2\omega t - c_1\cos 2\omega t)$ are processed using the following fragments of CUDA code:

```
uint width = 512, height = 512;                //optional
dim3 blockSize(16, 16);                        //optional

__global__ void quantKernel(float3* output, int dimx, int dimy, int dimz, float t)
{

  float c1 = 1.0;                              //components of observable vector
  float c2 = 1.0;
  float c3 = 1.0;

  float omega = 12560000.0;                    // possible angular velocity in the sprefield

  float tstep = 1.0f;

 float factor = 0.0;

  int qidx = threadIdx.x + blockIdx.x * blockDim.x;
  int qidy = threadIdx.y + blockIdx.y * blockDim.y;
  int qidz = threadIdx.z + blockIdx.z * blockDim.z;
```

```
  size_t oidx = qidx + qidy*dimx + qidz*dimx*dimy;

 output[oidx][0] = oidx*tstep;
 factor =4*(cosf(omega * output[oidx][0])) * (cosf(omega * output[oidx][0]));
 output[oidx][0] += factor * c3;
 output[oidx][1] = oidx*tstep;
 output[oidx][1] += factor * (c1 sin(2 * omega * t) + c2 cos(2 * omega * t));
 output[oidx][2] = oidx*tstep;
 output[oidx][2] += factor * (c2 sin(2 * omega * t) -c1 cos(2 * omega * t));
}

template<typename T>
void init(char * devPtr, size_t pitch, int width, int height, int depth)
{
   size_t qPitch = pitch * height;
   int v = 0;
   for (int z = 0; z < depth; ++z) {
      char * slice = devPtr + z * qPitch;;
      for (int y = 0; y < height; ++y) {
         T * row = (T *)(slice + y * pitch);
         for (int x = 0; x < width; ++x) {
            row[x] = T(v++);
         }
      }
   }

int keyboard(unsigned char key)
{
   switch (key)
   {
      case (27) :
                  cudaFree(d_output);
                  free(h_output);
                  return 1;
                  break;
            default:
            break;

   }
}

int main(void)
{
   VolumeType *h_volumeMem;
   unsigned char key;
   __device__ float g_fAnim = 0.0;

   float3* h_output = (float3*)malloc(size * sizeof(float3));     //array for measured vector
                                                                   //observable values
```

```
 float3* d_output = NULL;

  checkCudaErrors(
   cudaMalloc((void **)&d_output, width * height * sizeof(float3)));
  checkCudaErrors(cudaMemset(d_output, 0, width * height * sizeof(float3)));

  cudaExtent volumeSizeBytes = make_cudaExtent(width, SIZE_Y, SIZE_Z);
  cudaPitchedPtr d_volumeMem;
  checkCudaErrors (cudaMalloc3D(&d_volumeMem, volumeSizeBytes));

  size_t size = d_volumeMem.pitch * SIZE_Y * SIZE_Z;
  h_volumeMem = (VolumeType *)malloc(size);
  init<VolumeType>((char *)h_volumeMem, d_volumeMem.pitch, SIZE_X, SIZE_Y, SIZE_Z);
  checkCudaErrors (cudaMemcpy(d_volumeMem.ptr, h_volumeMem, size,
cudaMemcpyHostToDevice));

  cudaArray * d_volumeArray;
  cudaChannelFormatDesc channelDesc = cudaCreateChannelDesc<VolumeType>();
  cudaExtent volumeSize = make_cudaExtent(SIZE_X, SIZE_Y, SIZE_Z);
  checkCudaErrors ( cudaMalloc3DArray(&d_volumeArray, &channelDesc, volumeSize) );

  cudaMemcpy3DParms copyParams = {0};
  copyParams.srcPtr = d_volumeMem;
  copyParams.dstArray = d_volumeArray;
  copyParams.extent = volumeSize;
  copyParams.kind = cudaMemcpyDeviceToDevice;
  checkCudaErrors ( cudaMemcpy3D(&copyParams) );
  while(1)
 {
      g_fAnim += 0.01f;
      quantKernel<<<1,dim3(4,4,4)>>>(d_output,4,4,4, g_fAnim);
      cudaError_t error = cudaGetLastError();

      checkCudaErrors (cudaMemcpy(h_output, d_output, osize, cudaMemcpyDeviceToHost));
      if (keyboard(key)==1)
           exit(error);
 }

  return error;
}
```

The GPU CUDA case of simulation of measurement of a g-qubit type of observable only differs in quantKernel GPU executed function:

```
__global__ void quantKernel(float4* output, int dimx, int dimy, int dimz, float t)
{

  float c1 = 1.0
  float c2 = 1.0;
  float c3 = 1.0;
```

```
float omega = 12560000.0;                          // variant of angular velocity in the sprefield

float tstep = 1.0f;

float factor = 0.0;

  int qidx = threadIdx.x + blockIdx.x * blockDim.x;
  int qidy = threadIdx.y + blockIdx.y * blockDim.y;
  int qidz = threadIdx.z + blockIdx.z * blockDim.z;

  size_t oidx = qidx + qidy*dimx + qidz*dimx*dimy;

 output[oidx][0] = oidx*tstep;
 factor =4*(cosf(omega * output[oidx][0])) * (cosf(omega * output[oidx][0]));
 output[oidx][0] += factor * c3;
 output[oidx][1] = oidx*tstep;
 output[oidx][1] += factor * (c1 sin(2 * omega * t) + c2 cos(2 * omega * t));
 output[oidx][2] = oidx*tstep;
 output[oidx][2] += factor * (c2 sin(2 * omega * t) -c1 cos(2 * omega * t));
 output[oidx][3] = factor;
}
```

More flexibility in measurements can be achieved by scattering of the sprefield wave function before applying to observables.

Arbitrary Clifford translation $e^{I_{B_C}\gamma} = \cos\gamma + \sin\gamma\left(\gamma_1 I_S + \gamma_2 I_{B_0} + \gamma_3 I_{E_0}\right)$ acting on spreons (1.6) gives:

$$2\cos\omega[(I_3 I_S)\cdot r]\Big[\frac{1}{\sqrt{2}}(\cos\gamma\cos\omega t - \gamma_1\sin\gamma\sin\omega t - \gamma_2\sin\gamma\cos\omega t - \gamma_3\sin\gamma\sin\omega t) +$$
$$\frac{1}{\sqrt{2}}(\cos\gamma\sin\omega t + \gamma_1\sin\gamma\cos\omega t - \gamma_2\sin\gamma\sin\omega t + \gamma_3\sin\gamma\cos\omega t)I_S +$$
$$\frac{1}{\sqrt{2}}(\cos\gamma\cos\omega t + \gamma_1\sin\gamma\sin\omega t + \gamma_2\sin\gamma\cos\omega t - \gamma_3\sin\gamma\sin\omega t)I_{B_0} +$$
$$\frac{1}{\sqrt{2}}(\cos\gamma\sin\omega t - \gamma_1\sin\gamma\cos\omega t + \gamma_2\sin\gamma\sin\omega t + \gamma_3\sin\gamma\cos\omega t)I_{E_0}\Big] \qquad (2.3)$$

This result is defined for all values of t and r, in other words the result of Clifford translation instantly spreads through the whole three-dimensions for all values of time.

The instant of time when the Clifford translation was applied makes no difference for the state (2.3) because it is simultaneously redefined for all values of t. The values of measurements $O\left(C_0, C_1, C_2, C_3, I_S, I_{B_0}, I_{E_0}, \gamma, \gamma_1, \gamma_2, \gamma_3, \omega, t, r\right)$ also get instantly changed for all values of time of measurement, even if the Clifford translation was applied later than the measurement. That is an obvious demonstration that the suggested theory allows indefinite event casual order. In that way the very notion of the concept of cause and effect, ordered by time value increasing, disappears.

Since general result of measurement when Clifford translation takes place in an arbitrary plane is pretty complicated, I am only giving the result for the special case $\gamma_1 = 1$ and $\gamma_2 = \gamma_3 = 0$ (Clifford translation acts in plane I_S). The result is:

$$O\left(C_0, C_1, C_2, C_3, I_S, I_{B_0}, I_{E_0}, \gamma, \gamma_1, \gamma_2, \gamma_3, \omega, t, r\right)_{\gamma_1=1, \gamma_2=\gamma_3=0}$$
$$= 4cos^2\omega[(I_3 I_S) \cdot r]\big[C_0 + (C_2 \, sin\, 2\gamma + C_3 \, cos\, 2\gamma) I_S$$
$$+ \big(C_1 \, sin\, 2\omega t + sin\, 2\gamma \, cos\, 2\omega t\, (C_2 + C_3)\big) I_{B_0}$$
$$+ (-C_1 \, cos\, 2\omega t + sin\, 2\gamma \, sin\, 2\omega t\, (C_2 - C_3)) I_{E_0}\big]$$

The only component of measurement, namely the one lying in plane I_S, does not change with time. The I_{B_0} and I_{E_0} components do depend on the time of measurement being modified forward and backward in time if Clifford translation is applied. Clifford translation modifies measurement results of the past and the future.

3. Conclusions

In the suggested theory all measured observable values get available all together, not through looking one by one. In this way quantum computer appeared to be a kind of analog computer keeping and instantly processing information by and on sets of objects possessing an infinite number of degrees of freedom. The multithread GPUs bearing the CUDA language functionality allow to simultaneously calculate observable measurement values at a number of space/time discrete points, forward and backward in time, the number only restricted by the GPU threads capacity. That eliminates the tough hardware problem of creating huge and stable arrays of qubits, the base of quantum computing in conventional approaches.

References

[1] B. Ulmann, Analog Computing, Oldenbourg: De Gruyter , 2022.

[2] A. Soiguine, "Instantaneous Spreading of the g-Qubit Fields," *Journal of Applied Mathematics and Physics,* vol. 7, pp. 591-602, 2019.

[3] A. Soiguine, "Scattering of Geometric Algebra Wave Functions and Collapse in Measurements," *Journal of Applied Matrhematics and Physics,* vol. 8, pp. 1838-1844, 2020.

[4] A. Soiguine, Geometric Phase in Geometric Algebra Qubit Formalism, Saarbrucken: LAMBERT Academic Publishing, 2015.

[5] A. Soiguine, The Geometric Algebra Lift of Qubits and Beyond, LAMBERT Academic Publishing, 2020.