Two-Electron Integral Evaluation on FPGA, Cell and GPU accelerators

Guochun Shi, Volodymyr Kindratenko (University of Illinois), *Ivan Ufimtsev, Todd Martinez (*Stanford University)

Electron Repulsion Integrals

Electrons in molecules are described by wave functions determined by solving time-independent Schrödinger equation. To do that, a large number of two-electron repulsion integrals over N atom-centered one-electron basis functions needs to be evaluated. There are N^4 integrals to be evaluated. These basis functions are typically a linear combination of primitive atom-centered Gaussian basis functions. The two-electron integrals can be evaluated using integral with the primitive basis functions:

$$(\mu\nu|\lambda\sigma) = \sum_{p=1}^{N_\mu}\sum_{q=1}^{N_\nu}\sum_{r=1}^{N_\lambda}\sum_{s=1}^{N_\sigma} d_{\mu p}\, d_{\nu q}\, d_{\lambda r}\, d_{\sigma s}\, [pq|rs]$$

$$[s_1 s_2|s_3 s_4] = \frac{\pi^3}{AB\sqrt{A+B}} K_{12}(\vec{\mathbf{R}}_{12}) K_{34}(\vec{\mathbf{R}}_{34}) F_0\left(\frac{AB}{A+B}[\vec{\mathbf{R}}_{\mathbf{P}} - \vec{\mathbf{R}}_{\mathbf{Q}}]^2\right)$$

$$A = \alpha_1 + \alpha_2,\ B = \alpha_3 + \alpha_4,\ F_0(t) = \frac{erf(\sqrt{t})}{\sqrt{t}},$$
$$\vec{\mathbf{R}}_{\mathbf{kl}} = \vec{\mathbf{R}}_{\mathbf{k}} - \vec{\mathbf{R}}_{\mathbf{l}},\ \vec{\mathbf{R}}_{\mathbf{P}} = \frac{\alpha_1\vec{\mathbf{R}}_1 + \alpha_2\vec{\mathbf{R}}_2}{A},\ \vec{\mathbf{R}}_{\mathbf{Q}} = \frac{\alpha_3\vec{\mathbf{R}}_3 + \alpha_4\vec{\mathbf{R}}_4}{B},$$
$$K_{ij}(\vec{\mathbf{R}}_{ij}) = exp\left(-\frac{\alpha_i\alpha_j}{\alpha_i + \alpha_j}[\vec{\mathbf{R}}_i - \vec{\mathbf{R}}_j]^2\right)$$

where N_* is the contraction length, d_{**} are contraction coefficients, and [pq|rs] are integrals evaluated over primitive basis functions.
Computation of two-electron repulsion integrals is the most time-consuming operation in direct self-consistent field (SCF) methods in which many millions of electron repulsion integrals are recomputed every SCF iteration.

Reference Implementation

```
compute(Shell *ComputeShell, Real2 *Sprms, Real3 *Coors, int startShell, int stopShell, int totNumShells,
        Real *H_ReductionSum)
{
  for (shell1 = startShell; shell1 < stopShell; shell1++) {
    for (shell2 = shell1; shell2 < totNumShells; shell2++) {
      for (shell3 = shell1; shell3 < totNumShells; shell3++) {
        for (shell4 = shell3; shell4 < totNumShells; shell4++) {
          for (prim1 = 0; prim1 < ComputeShell[shell1].numPrimitives; prim1++) {
            for (prim2 = 0; prim2 < ComputeShell[shell2].numPrimitives; prim2++) {
              for (prim3 = 0; prim3 < ComputeShell[shell3].numPrimitives; prim3++) {
                for (prim4 = 0; prim4 < ComputeShell[shell4].numPrimitives; prim4++)
                {
                  A = Alpha1 + Alpha2;
                  invA = 1 / A;
                  a1a2dA = -Alpha1 * Alpha2 / A;
                  Atomp_x = H_product1D_mod(Alpha1, Atom1_x, Alpha2, Atom2_x, invA);
                  Atomp_y = H_product1D_mod(Alpha1, Atom1_y, Alpha2, Atom2_y, invA);
                  Atomp_z = H_product1D_mod(Alpha1, Atom1_z, Alpha2, Atom2_z, invA);

                  B = Alpha3 + Alpha4;
                  invB = 1 / B;
                  a3a4dB = -Alpha3 * Alpha4 / B;
                  Atomq_x = H_product1D_mod(Alpha3, Atom3_x, Alpha4, Atom4_x, invB);
                  Atomq_y = H_product1D_mod(Alpha3, Atom3_y, Alpha4, Atom4_y, invB);
                  Atomq_z = H_product1D_mod(Alpha3, Atom3_z, Alpha4, Atom4_z, invB);

                  AB = A * B;
                  inv_sqrt = PI / SQRT(AB);
                  rho = AB / (A + B);
                  rpq2 = H_dist2(Atomp_x, Atomp_y, Atomp_z, Atomq_x, Atomq_y, Atomq_z);
                  X = rpq2 * rho;
                  weight = gamma0(X);

                  I1 = H_Computel_mod(Atom1_x, Atom2_x, Atom3_x, Atom4_x, a1a2dA, a3a4dB, inv_sqrt);
                  I2 = H_Computel_mod(Atom1_y, Atom2_y, Atom3_y, Atom4_y, a1a2dA, a3a4dB, inv_sqrt);
                  I3 = H_Computel_mod(Atom1_z, Atom2_z, Atom3_z, Atom4_z, a1a2dA, a3a4dB, inv_sqrt);

                  H_ReductionSum[numElements] +=
                    SQRT(F_PI * rho) * I1*I2*I3*weight * Coeff1*Coeff2*Coeff3*Coeff4;
          } } } }
          numElements++;
  } } } }
}
```

```
float gamma0(float T)
{
  if (T < 25.0f) {
    T *= 1.714081e+01;
    index = float2int(T);
    f0 = table[index][0];
    f1 = table[index][1];
    f2 = table[index][2];
    f3 = table[index][3];
    f4 = table[index][4];
    return f0 + T*(f1 + T*(f2 + T*(f3 + T*f4)));
  }
  else {
    return 8.862269e-01;
  }
}
```

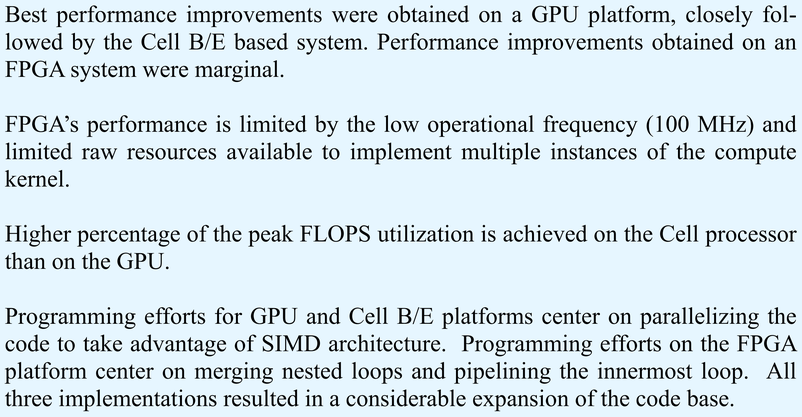
```
inline Real H_product1D_mod(Real alpha_a, Real coor_a,
                            Real alpha_b,
                            Real coor_b,
                            Real inv)
{
  return (alpha_a * coor_a + alpha_b * coor_b) * inv;
}
```

```
inline Real H_dist2(Real Atom1_x, Real Atom1_y, Real Atom1_z,
                    Real Atom2_x, Real  Atom2_y, Real Atom2_z)
{
  const Real dx = Atom1_x - Atom2_x;
  const Real dy = Atom1_y - Atom2_y;
  const Real dz = Atom1_z - Atom2_z;
  return dx * dx + dy * dy + dz * dz;
}
```

```
inline Real H_Computel_mod(Real coor1,
                           Real coor2,
                           Real coor3,
                           Real coor4,
                           Real a1a2dA, Real a3a4dB,
                           Real inv_sqrt)
{
  const Real dx1 = coor2 - coor1;
  const Real dx2 = coor4 - coor3;
  const Real preexp = a1a2dA * dx1 * dx1 + a3a4dB * dx2 * dx2;
  return inv_sqrt * EXP(preexp);
}
```

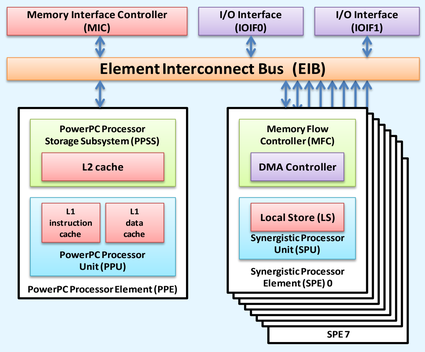
GPU Implementation

Mapping to GPU	description	grainulatiry	Load balance	Reduction overhead
1B1Cl	One thread block <-> one contracted integral	Intemediate	intemediate	Communication within thread block; intermediate overhead
1T1Cl	One thread <-> one contracted integral	Coare-grained	Not balanced	No communication; no overhead
1T1Pl	One thread <-> one primitive integral	Fine-grained	Perfectly balanced	Communication with different thread blocks; large overhead



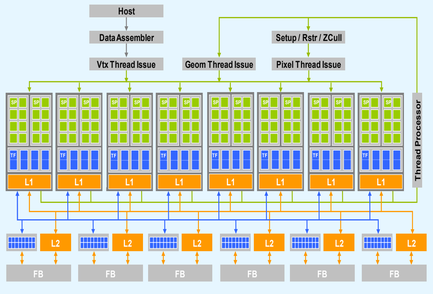
Systems

Cell processor



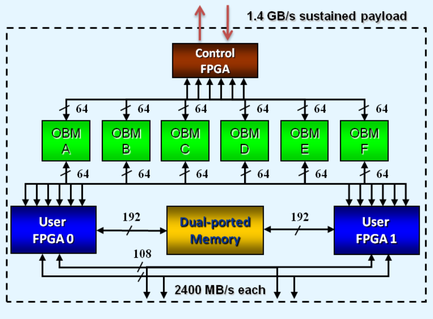
GFLOPS	200
Bandwidth between host memory and PE memory (GB/s)	25.6
Local memory to PE bandwidth (GB/s)	
Frequency (GHz)	3.2
# of processing elements	8

NVIDIA 8800 GTX GPU



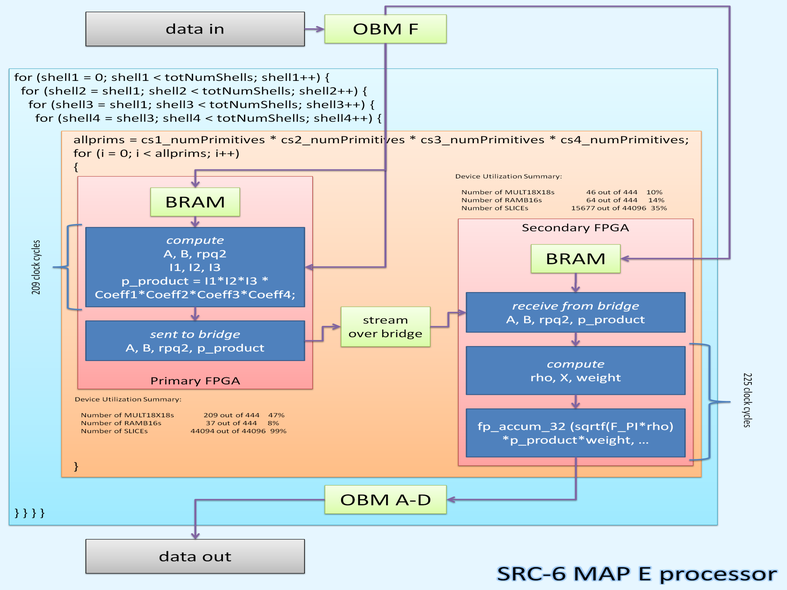
GFLOPS	345.6
Bandwidth between host memory and PE memory (GB/s)	2.0
Local memory to PE bandwidth (GB/s)	86.4
Frequency (GHz)	1.35
# of processing elements	128

SRC-6 MAP-E

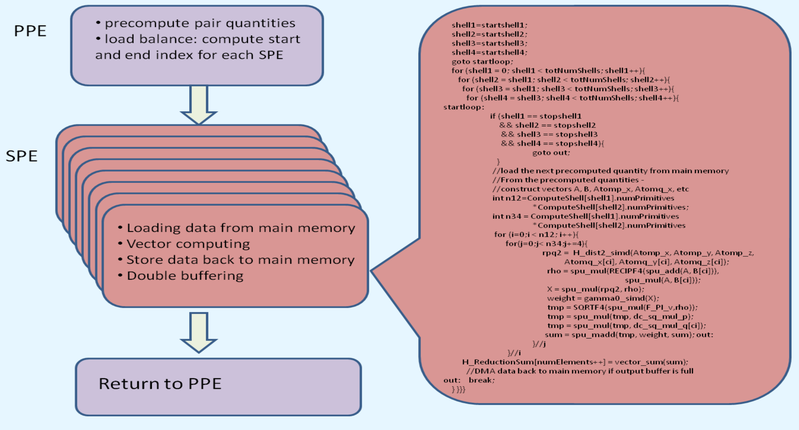


GFLOPS	24
Bandwidth between host memory and PE memory (GB/s)	1.4
Local memory to PE bandwidth (GB/s)	4.8
Frequency (GHz)	0.1
# of processing elements	

SRC-6 Implementation



Cell/B.E. Implementation



Results

Evaluation of two-electron repulsion integrals for a 64 H atom system using STO-6G basis set

Accelerator type		Kernel run time	CPU pre-caculation	GPU-CPU transfer	Overall runtime	Overall speedup+
FPGA (SRC-6)					42.85	2.6
GPU (8800 GTX)	1B1CI	1.608			1.632	68
	1T1CI	1.099			1.123	100
	1T1PI	2.863	0.012	0.012	2.88	39
Cell BE		1.25	0.089	0*	1.34	84

** On Cell B/E double buffering makes the communication hidden within computation. +The results are compared with the CPU code running on a 2.33 GHz Intel Xeon, compiled with icc. The CPU code runs in 112.55 seconds.*

Conclusions

Best performance improvements were obtained on a GPU platform, closely followed by the Cell B/E based system. Performance improvements obtained on an FPGA system were marginal.

FPGA's performance is limited by the low operational frequency (100 MHz) and limited raw resources available to implement multiple instances of the compute kernel.

Higher percentage of the peak FLOPS utilization is achieved on the Cell processor than on the GPU.

Programming efforts for GPU and Cell B/E platforms center on parallelizing the code to take advantage of SIMD architecture. Programming efforts on the FPGA platform center on merging nested loops and pipelining the innermost loop. All three implementations resulted in a considerable expansion of the code base.

LELAND STANFORD JUNIOR UNIVERSITY 1891
NCSA
1867
ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN