Summary of Current and Future CyberChem Activities at ISL/NCSA

Volodymyr Kindratenko
Innovative Systems Laboratory (ISL)
National Center for Supercomputing Applications (NCSA)
University of Illinois at Urbana-Champaign (UIUC)

ISL Technology Overview

- **Reconfigurable computing**
  - SRC-6 reconfigurable computer with 2 MAP processors

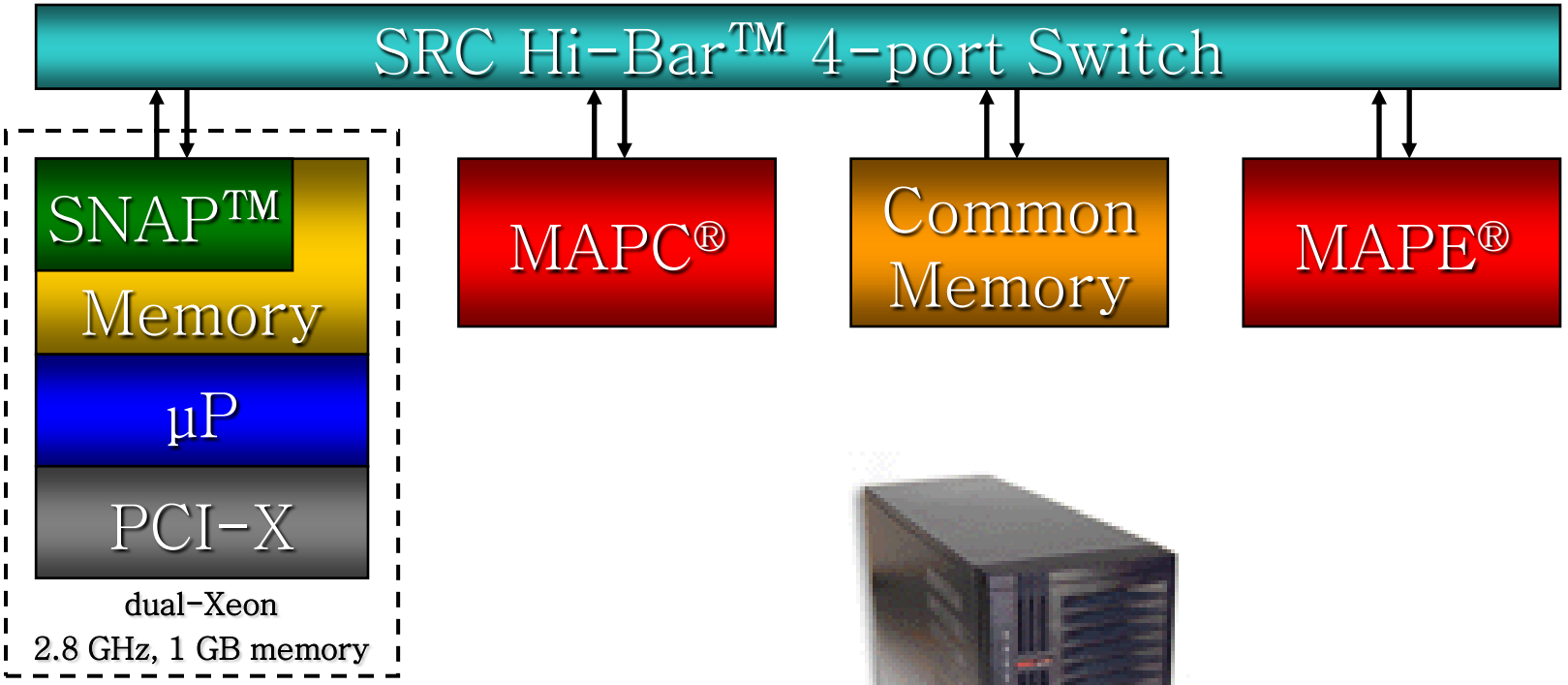
  - SGI Altix 350 with RASC Athena module

  - Nallatech H101 PCIXM add-on FPGA board

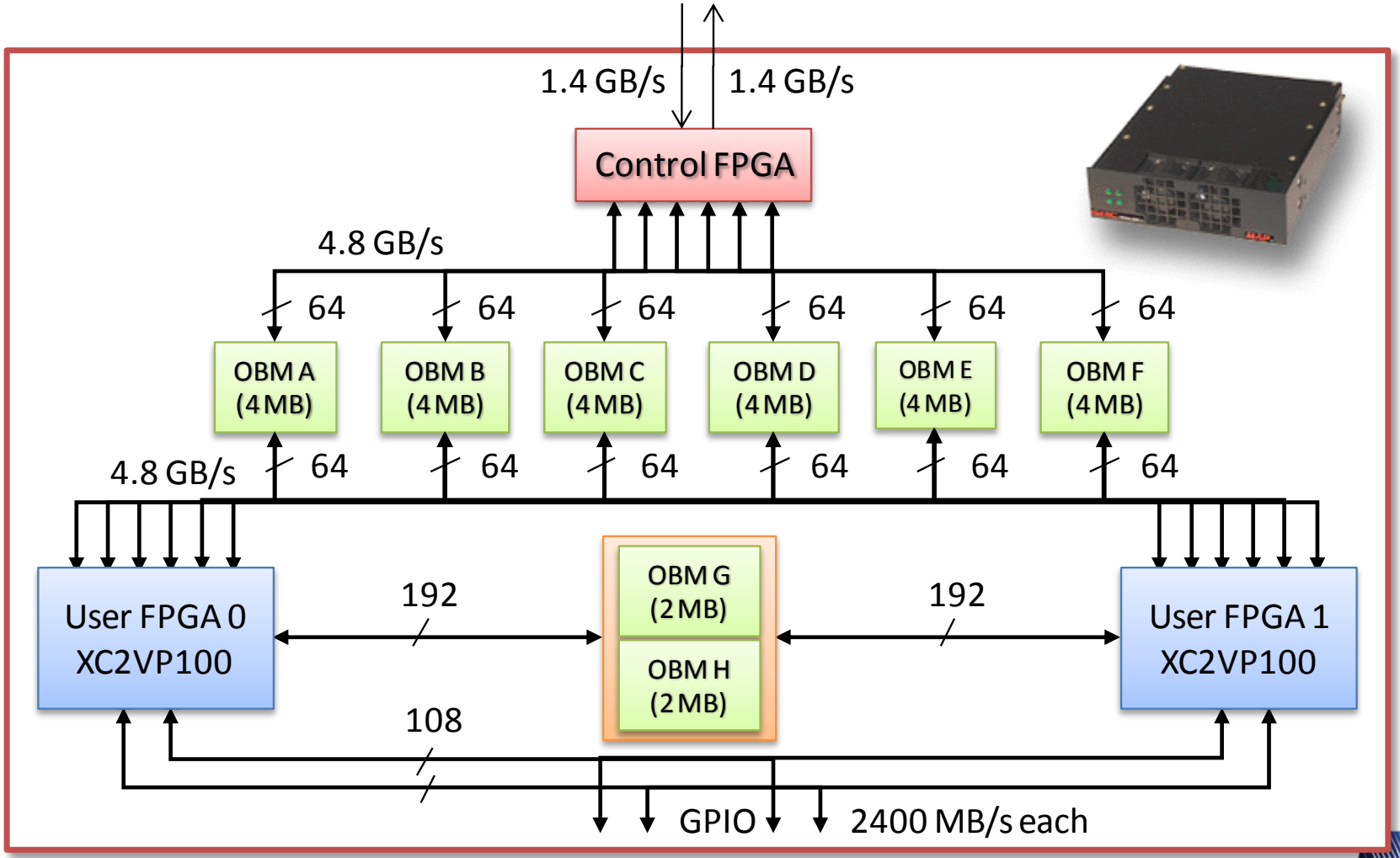
- **Cell/B.E.**
  - IBM Blade Center with 2 Cell/B.E. blades

- **GPU**
  - 10-node PC cluster with NVIDIA GeForce 8800 GTX cards

SRC-6 Reconfigurable Computer

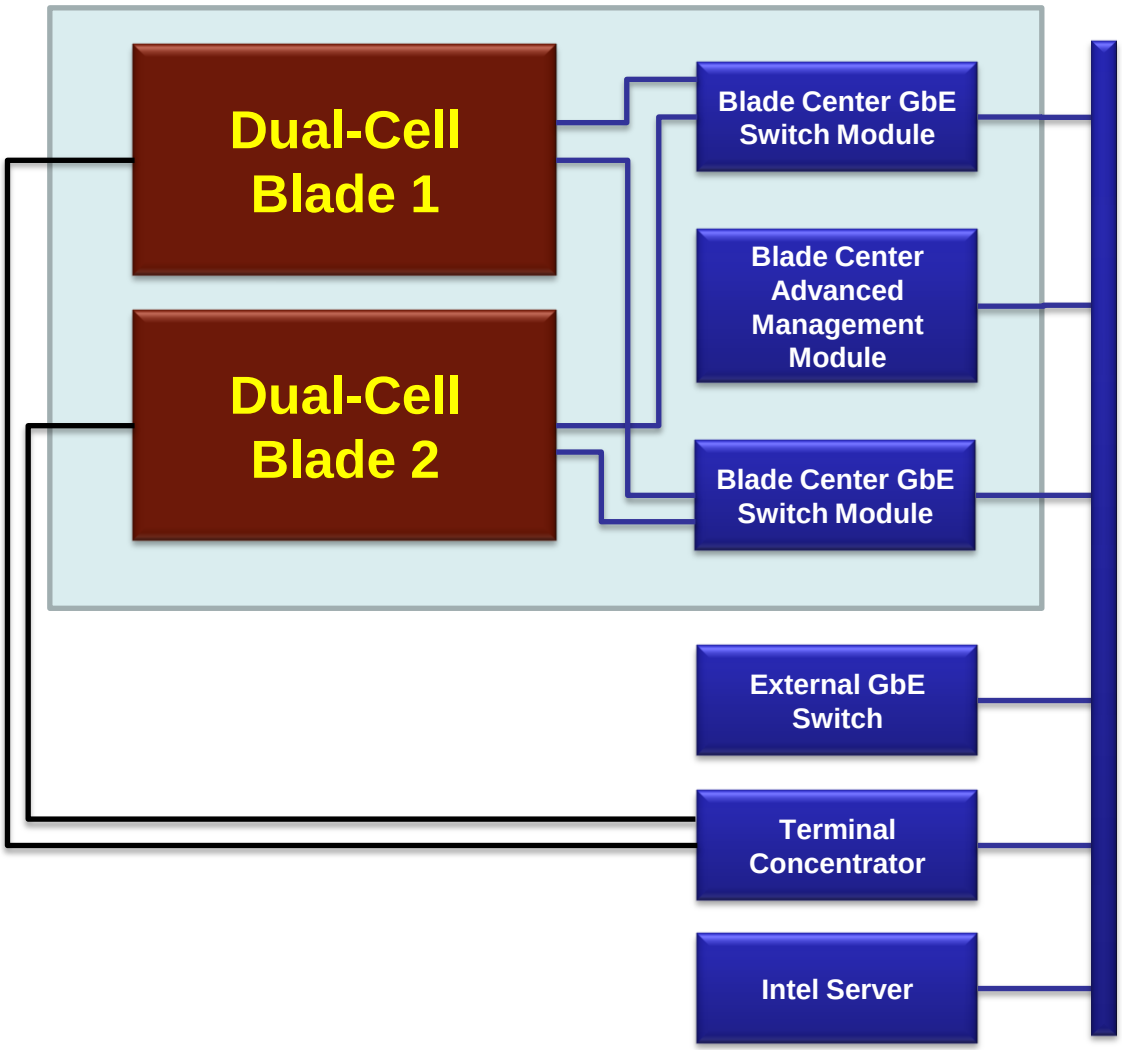


SRC-6 MAP Series E Processor

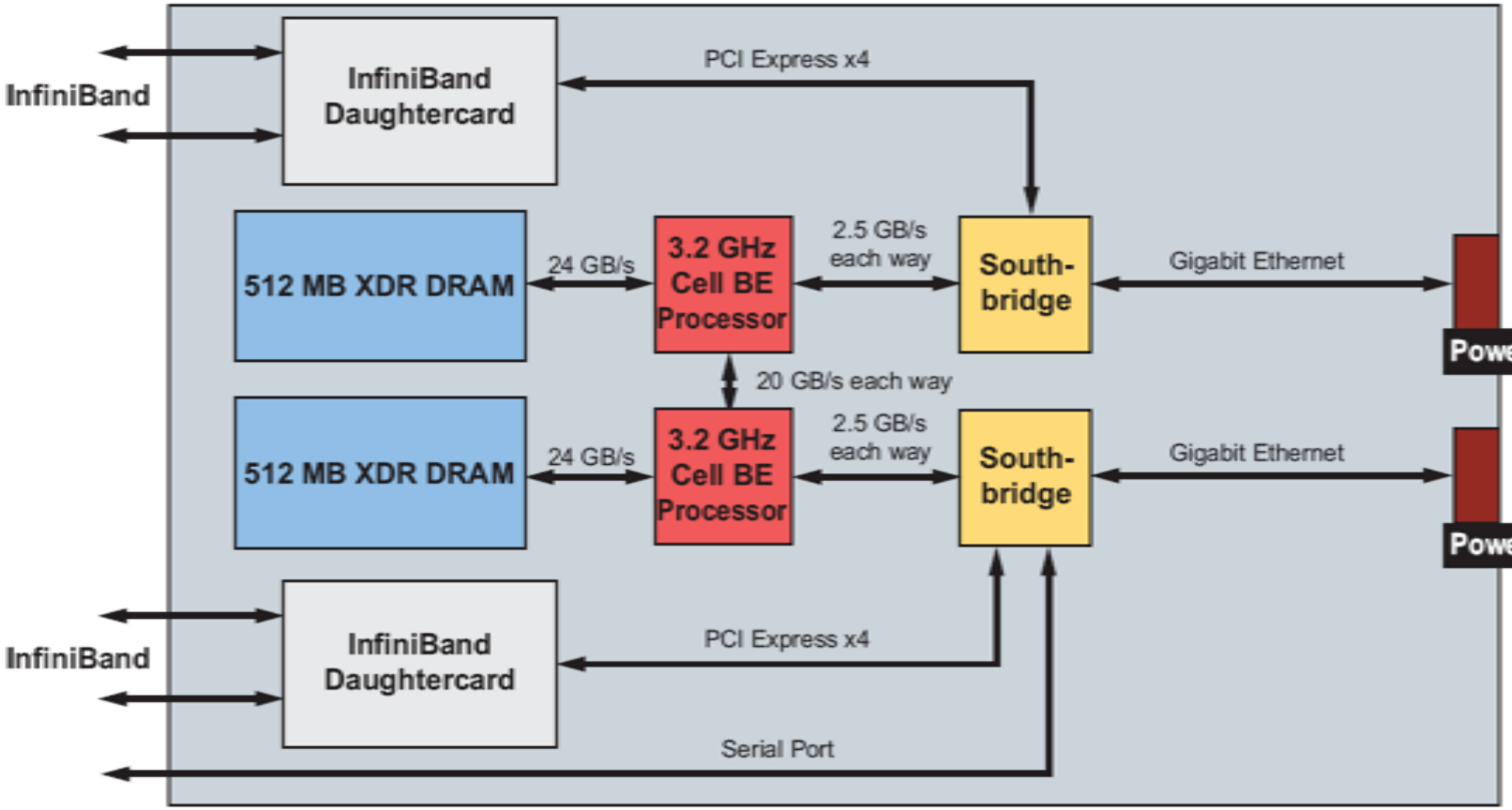


24/10 GFLOPS (SP/DP) peak

IBM Blade Center Chassis



IBM Dual-Cell Blade



410/28 GFLOPS (SP/DP) peak

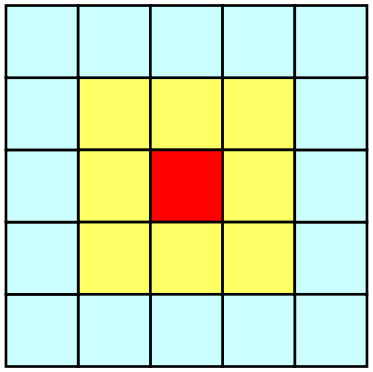
NAMD

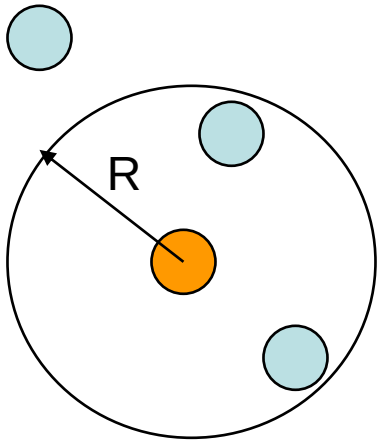
- **NAMD SPEC 2006 kernel ported to SRC-6**
  - Only the non-bonded force field calculations
    - Excluding pair list
  - Using SPFP version of the kernel
    - Kernel execution time is reduced from 9 seconds (2.8 GHz Intel Xeon) to 3 seconds (MAP Series E processor)
  - Main issues
    - Not enough logic elements to implement more than 2 compute engines per chip
    - The MAP C language primitives are not flexible enough to support complex operations

- **NAMD SPEC 2006 kernel ported to Cell/B.E.**
  - Only the non-bonded force field calculations
    - Excluding pair list
  - Using SPFP version of the kernel
    - Kernel execution time is reduced from 8.6 seconds (2.2 GHz AMD Opteron) to 1.1 seconds on one Cell
  - Using DPFP version of the kernel
    - Kernel execution time is reduced from 11.1 seconds (2.2 GHz AMD Opteron) to 2.2 seconds on one Cell
  - Main issue
    - Data manipulation overhead due to the need to vectorize the data for SIMD engine

Optimizations in NAMD

- **Spatial decomposition**
  - for (*k*=0; *k*=M; *k*++)
  - for (*l*=0; *l*=13; *l*++)
- **Cut-off radius**
  - for (*i*=0; *i* < N_k; *i*++)
  - for (*j*=0; *j* < N_l; *j*++)
  - if (distance(*i*,*j*) < cutoff_distance)
- **Newton's Third Law**
  - F(*i*) += f(x_i,x_j)
  - F(*j*) -= f(x_i,x_j)



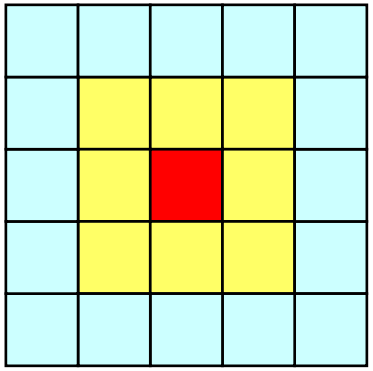


NAMD source code

```
void ComputeList::runComputes(PatchList *patchList)
{
    int i;

    for ( i=0; i<numSelfComputes; ++i ) {
        selfComputes[i].doWork(patchList);
    }

    for ( i=0; i<numPairComputes; ++i ) {
        pairComputes[i].doWork(patchList);
    }
}
```



Potential energy kernel

for each atom *i* in patch p_k

  for each atom *j* in patch p_l

    if atoms *i* and *j* are bonded, compute bonded forces

    otherwise, if atoms *i* and *j* are within the cutoff distance

      add atom *j* to the *i*'s atom pair list

  end

  for each atom *q* in the *i*'s atom pair list

    compute non-bonded forces

  end

end

Simplified potential energy kernel

for each atom *i* in patch p_k

for each atom *j* in patch p_l

if atoms *i* and *j* are within the cutoff distance

compute non-bonded forces

end

end

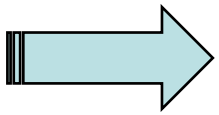
- **The simplifications were necessary because of the overhead due to the non-compute-intensive code (e.g., pair list maintenance)**

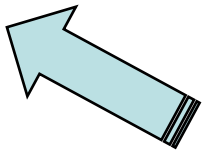
NAMD data storage transformation

- **// original NAMD data storage**

```
struct CompAtom {
  Position position;   // class
  Charge charge;
  unsigned int id : 22;
  unsigned int vdw_type: 10;
};

typedef IntVector Force;
class IntVector {
 public:
  int32 x,y,z;
  inline IntVector(void) { ; }
  inline IntVector(const Vector &v);
};
```



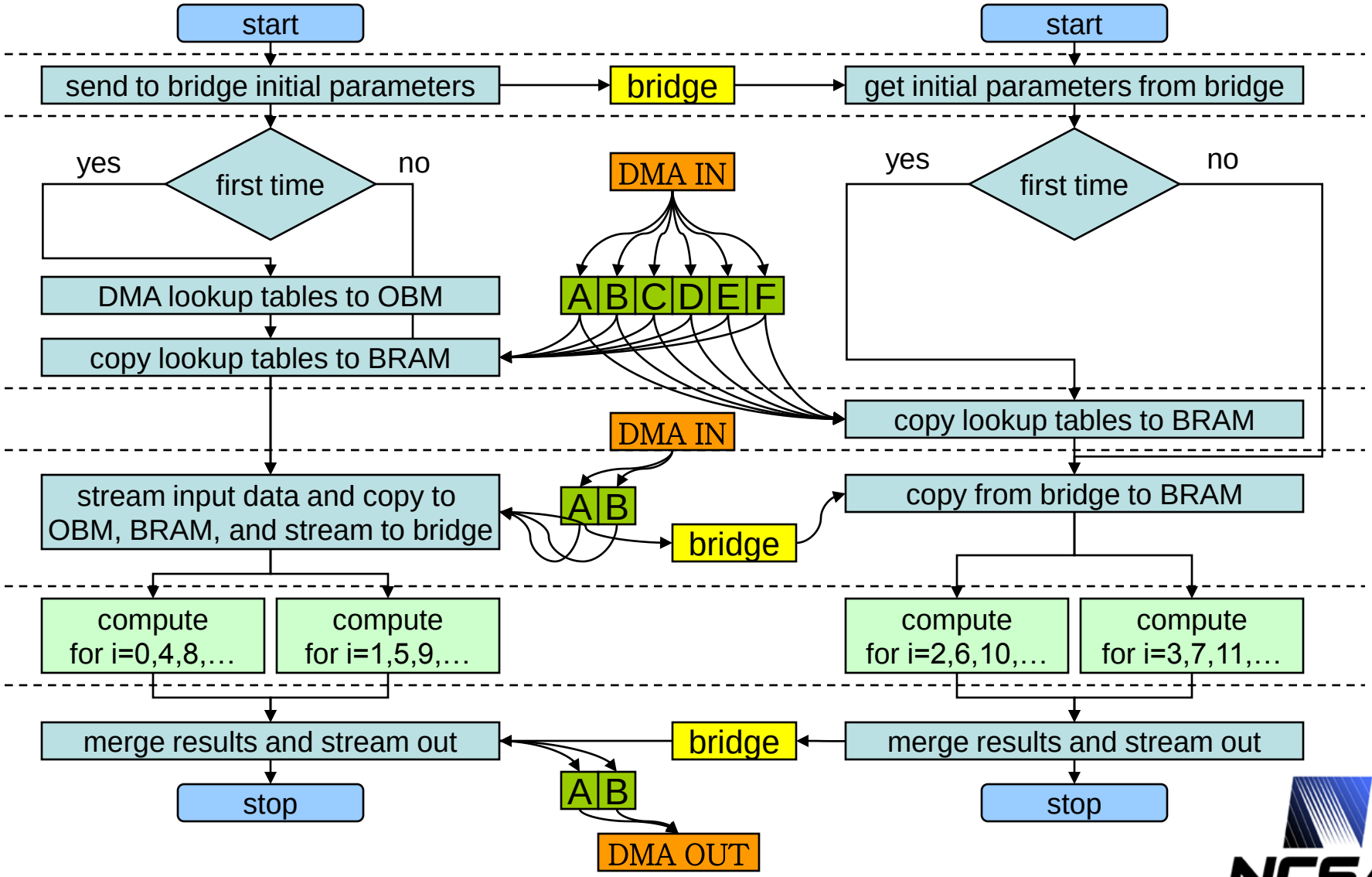


- **// MAP data storage (in BRAM)**

```
struct FPGA_input_data {
    float p_x;
    float p_y;
    float p_z;
    float p_charge;
    int p_atomVdwType;
    int f_x;
    int f_y;
    int f_z;
};

struct FPGA_output_data {
    int nothing;
    int f_x;
    int f_y;
    int f_z;
};
```

NAMD kernel MAP E implementation



NAMD kernel MAP E implementation

- **Primary chip**

Device Utilization Summary:

Number of BUFGMUXs	1 out of 16	6%
Number of External IOBs	832 out of 1164	71%
Number of LOCed IOBs	832 out of 832	100%
Number of MULT18X18s	131 out of 444	29%
Number of RAMB16s	258 out of 444	58%
Number of SLICEs	44094 out of 44096	99%

Timing analysis: Actual: 9.964ns

- **Secondary chip**

Device Utilization Summary:

Number of BUFGMUXs	1 out of 16	6%
Number of External IOBs	745 out of 1164	64%
Number of LOCed IOBs	745 out of 745	100%
Number of MULT18X18s	134 out of 444	30%
Number of RAMB16s	258 out of 444	58%
Number of SLICEs	40427 out of 44096	91%

Timing analysis: Actual: 9.971ns

Execution time ~3.07 seconds (measured on CPU)

~0.15 seconds due to data DMA in/out and (measured on MAP)

~0.84 seconds due to MAP function call overhead

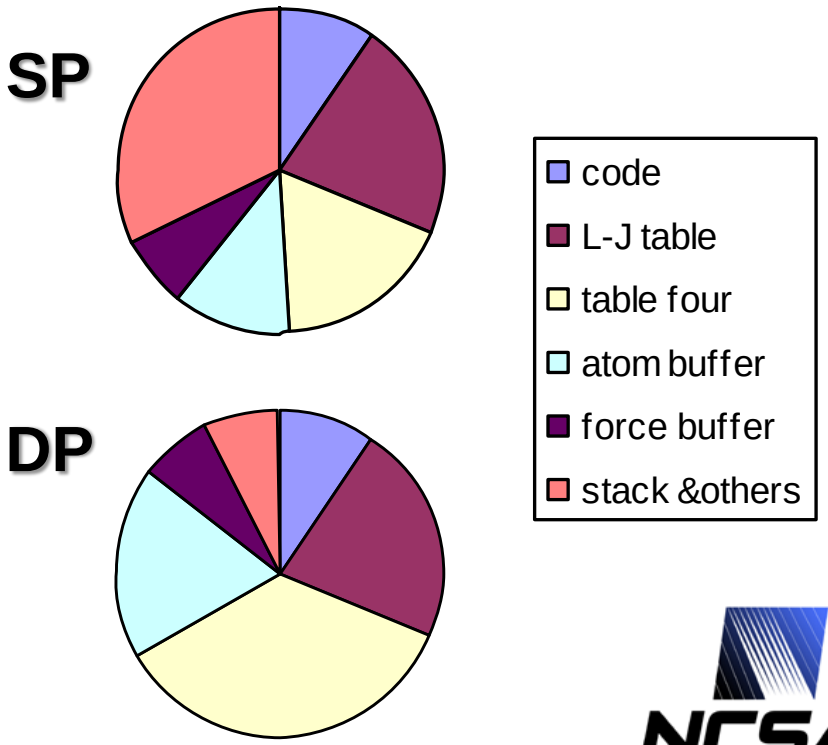
~2.08 seconds due to actual calculations (measured on MAP)

which is 3x speedup

NAMD data storage in Cell SPE

LS usage	Code	L-J table	Table four	Atom buffer	Force buffer	Stack & others
SP (KB)	25	55	45	30	18	83
DP(KB)	25	55	91	48	18	19

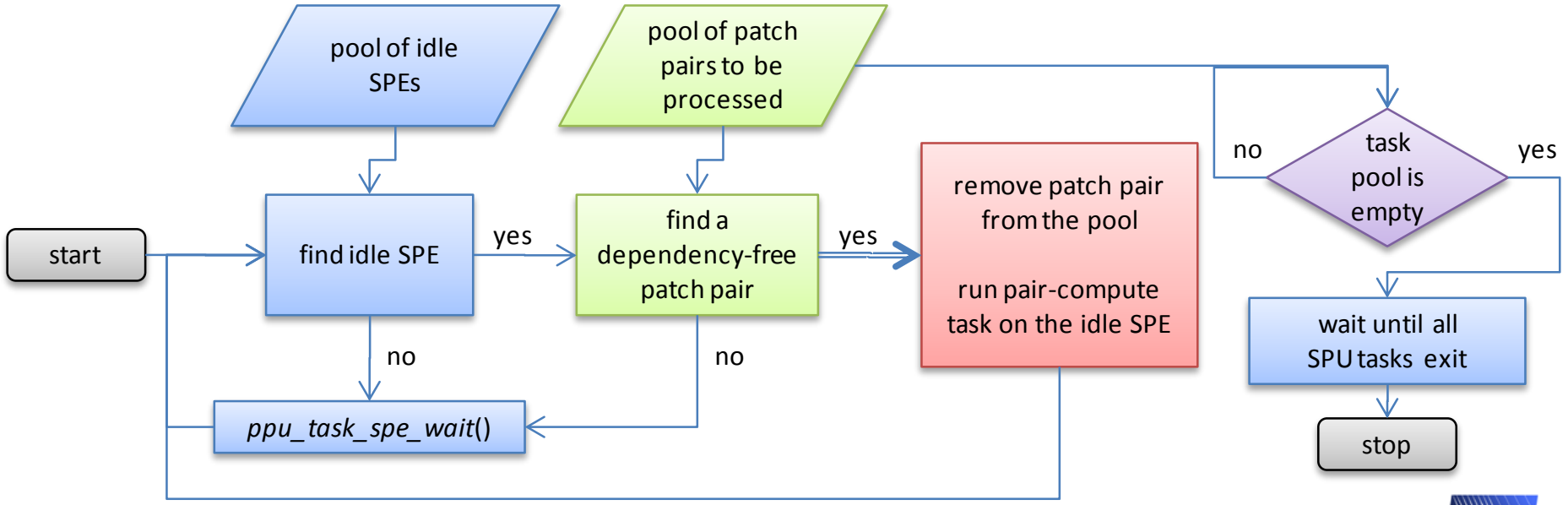
- **No data transformation needed**
- **Entire patch can be loaded into SPEs**
- **DPFP implementation can barely fit**



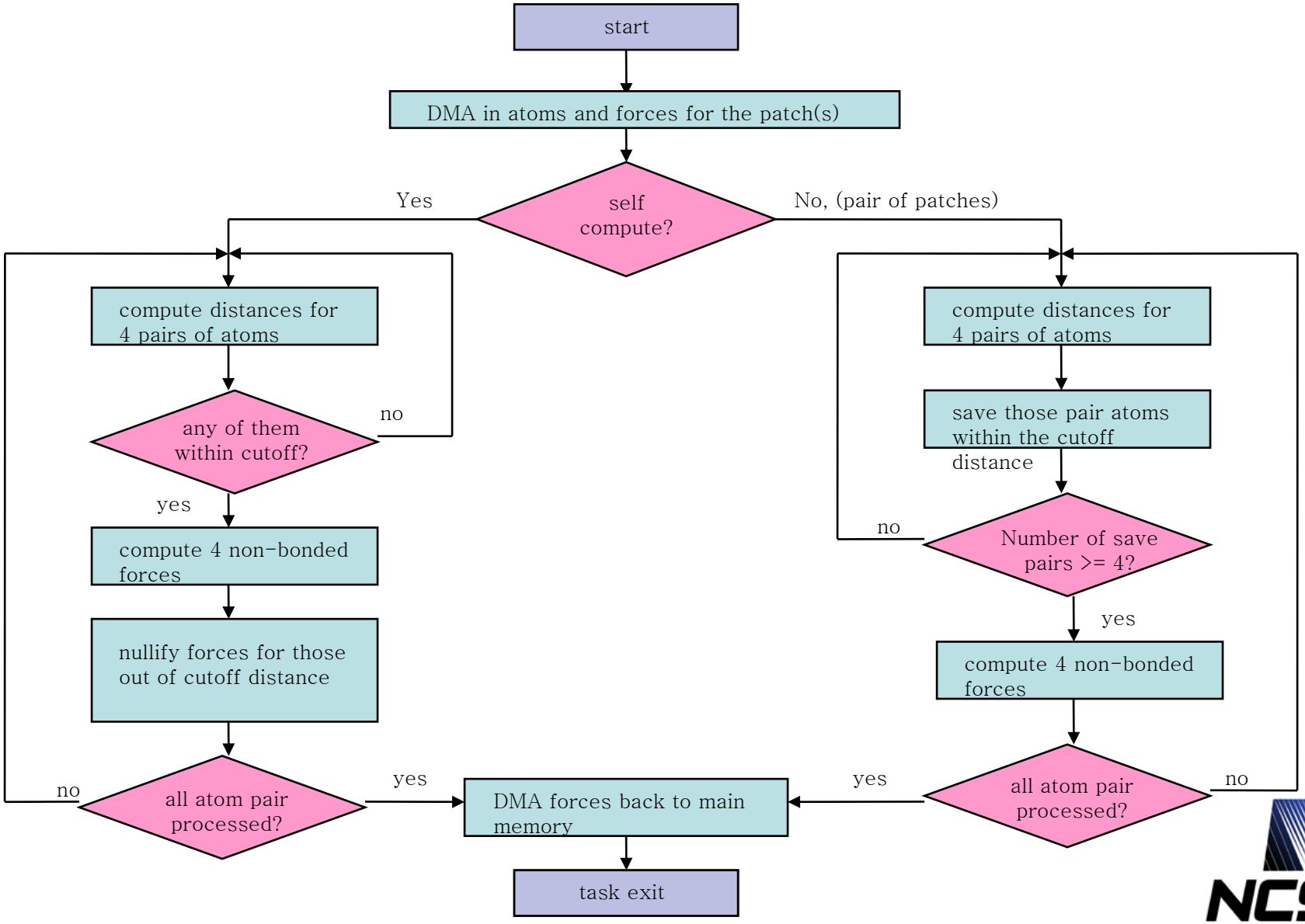
NCSA

NAMD Kernel on Cell/B.E.

- **Job dispatcher on the PPU**
  - Distributes per-patch calculations between 8(16) SPUs

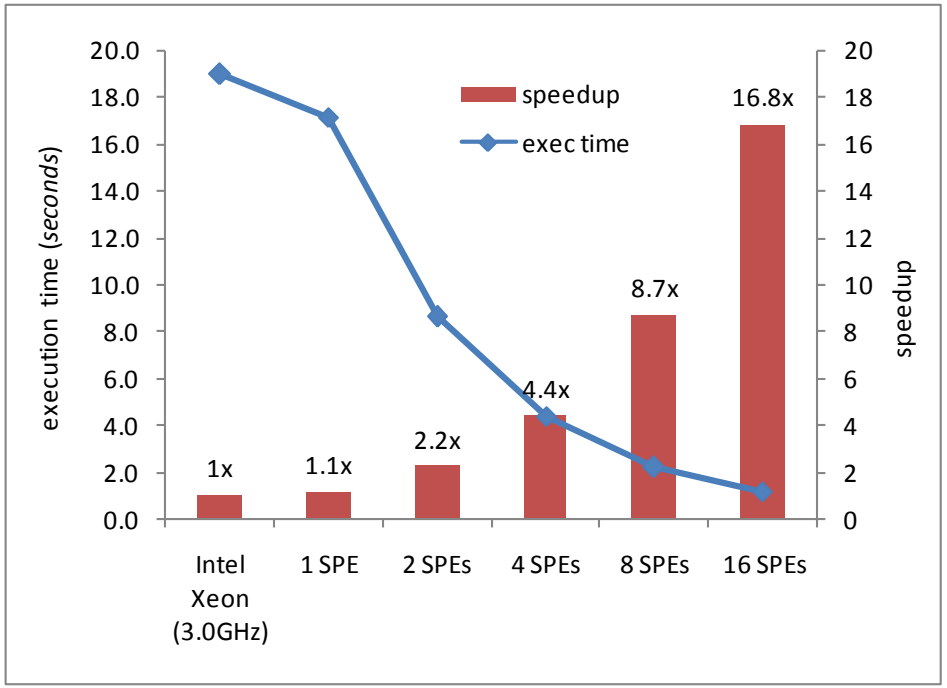


SPE Kernel Implementation



NAMD Cell/B.E. performance

- **SPFP version of the kernel**
  - 1.1 seconds on one Cell
- **DPFP version of the kernel**
  - 2.2 seconds on one Cell
- **Linear scaling**



NCSA

PyQuante

- **Two electron integral evaluation using Rys polynomials**
  - polynomial code from GAMESS

Coulomb interaction between 4 contracted Gaussians

```
static double contr_coulomb(int lena,double *aexps,double *acoefs,double *anorms,
                            double xa,double ya,double za,int la,int ma,int na,
                            int lenb,double *bexps,double *bcoefs,double *bnorms,
                            double xb,double yb,double zb,int lb,int mb,int nb,
                            int lenc,double *cexps,double *ccoefs,double *cnorms,
                            double xc,double yc,double zc,int lc,int mc,int nc,
                            int lend,double *dexps,double *dcoefs,double *dnorms,
                            double xd,double yd,double zd,int ld,int md,int nd){
  double val = 0.;
  for (int i=0; i<lena; i++)
    for (int j=0; j<lenb; j++)
      for (int k=0; k<lenc; k++)
        for (int l=0; l<lend; l++)
          val += acoefs[i]*bcoefs[j]*ccoefs[k]*dcoefs[l] *
            coulomb_repulsion(xa,ya,za,anorms[i],la,ma,na,aexps[i],
                              xb,yb,zb,bnorms[j],lb,mb,nb,bexps[j],
                              xc,yc,zc,cnorms[k],lc,mc,nc,cexps[k],
                              xd,yd,zd,dnorms[l],ld,md,nd,dexps[l]);
  return val;
}
```

Coulomb Repulsion

```
static double coulomb_repulsion(…) {
  int norder = (la+ma+na+lb+nb+mb+lc+mc+nc+ld+md+nd)/2 + 1;
  double A = alphaa+alphab;
  double B = alphac+alphad;
  double rho = A*B/(A+B);
  double xp = product_center_1D(alphaa,xa,alphab,xb);
  double yp = product_center_1D(alphaa,ya,alphab,yb);
  double zp = product_center_1D(alphaa,za,alphab,zb);
  double xq = product_center_1D(alphac,xc,alphad,xd);
  double yq = product_center_1D(alphac,yc,alphad,yd);
  double zq = product_center_1D(alphac,zc,alphad,zd);
  double rpq2 = dist2(xp,yp,zp,xq,yq,zq);
  double X = rpq2*rho;
  Roots(norder,X);  /* Puts correct roots/weights in "common" */
  double sum = 0.;
  for (int i=0; i<norder; i++) {
    double t = roots[i];
    double Ix = Int1d(t,la,lb,lc,ld,xa,xb,xc,xd,alphaa,alphab,alphac,alphad);
    double Iy = Int1d(t,ma,mb,mc,md,ya,yb,yc,yd,alphaa,alphab,alphac,alphad);
    double Iz = Int1d(t,na,nb,nc,nd,za,zb,zc,zd,alphaa,alphab,alphac,alphad);
    sum += Ix*Iy*Iz*weights[i]; /* ABD eq 5 & 9 */
  }
  return 2*sqrt(rho/M_PI)*norma*normb*normc*normd*sum; /* ABD eq 5 & 9 */
}
```

```
static double product_center_1D(double alphaa,
double xa, double alphab, double xb) {
  return (alphaa*xa+alphab*xb)/(alphaa+alphab);}
```

```
static double dist2(double x1, double y1, double z1, double
x2, double y2, double z2) {
return (x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)+(z1-z2)*(z1-z2);}
```

Polynomial roots evaluation

- **static void Root123(int n, double X)**
  - 475 DPFP multiplications
  - Over 30 conditional branches
- **static void Root4(double X)**
  - 518 DPFP multiplications
  - 10 conditional branches
- **static void Root5(double X)**
  - 786 DPFP multiplications
  - 10 conditional branches
- **Multiple sqrt, exp, division…**

- **FPGAs do not have enough logic to implement this code “as is”**

Polynomial roots evaluation

- **However, via code restructuring it may be possible to collapse multiple explicit expressions into a single "generic" expression.**

- **Current code:**
  - RT1 = (((-2.43758528330205E-02*X+2.07301567989771E+00)*X - 6.45964225381113E+01)*X+7.14160088655470E+02)*E + R15/(X-R15);
  - WW5 = (((((((((((((7.28996979748849E-19*Y-1.26518146195173E-17) *Y+1.886145834486E-16)*Y-2.876728287383E-15)*Y+ 4.114588668138E-14)*Y-5.44436631413933E-13)*Y+ 6.64976446790959E-12)*Y-7.44560069974940E-11)*Y+ 7.57553198166848E-10)*Y-6.92956101109829E-09)*Y+ 5.62222859033624E-08)*Y-3.97500114084351E-07)*Y+ 2.39039126138140E-06)*Y-1.18023950002105E-05)*Y+ 4.52254031046244E-05)*Y-1.21113782150370E-04)*Y+1.75013126731224E-04;

786 DPFP multipliers

- **"Generic" replacement:**
  - IN=(cond)? X : Y;
  - C=(cond1) ? C1 : (cond2) ? C2 : …
  - RES=((((((((((((((C[0]*IN+C[1])*IN+C[2])*IN+…)

~150 DPFP multipliers

```
static double Int1d(double t, int ix, int jx, int kx, int lx,
          double xi, double xj, double xk,double xl,
          double alphai, double alphaj, double alphak, double alphal) {
  Recur(t, ix, jx, kx, lx, xi, xj, xk, xl, alphai, alphaj, alphak, alphal);
  double Ix = Shift(ix, jx, kx, lx, xi-xj, xk-xl);
  return Ix;
}

static double Shift(int i, int j, int k, int l, double xij, double xkl) {
  double ijkl = 0;
  for (int m=0; m<l+1; m++) {
     double ijm0 = 0;
     for (int n=0; n<j+1; n++)
        ijm0 += binomial(j,n)*pow(xij,j-n)*G[n+i][m+k];
     ijkl += binomial(l,m)*pow(xkl,l-m)*ijm0; /* I(i,j,k,l)<-I(i,j,m,0) */
  }
  return ijkl;
}
```

```
static int binomial(int a, int b) { return fact(a)/(fact(b)*fact(a-b));}

static int fact(int n) {  if (n <= 1) return 1;return n*fact(n-1);}
```

```
static void Recur(double t, int i, int j, int k, int l, double xi, double xj, double xk, double xl,
                  double alphai, double alphaj, double alphak, double alphal) {
  int n = i+j,  m = k+l;
  double A = alphai+alphaj, B = alphak+alphal;
  double Px = (alphai*xi+alphaj*xj)/A, Qx = (alphak*xk+alphal*xl)/B;

  RecurFactorsGamess(t,A,B,Px,Qx,xi,xk);

  G[0][0] = M_PI*exp(-alphai*alphaj*pow(xi-xj,2)/(alphai+alphaj) -alphak*alphal*pow(xk-xl,2)/(alphak+alphal))/sqrt(A*B);

  if (n > 0) G[1][0] = C*G[0][0];  /* ABD eq 15 */
  if (m > 0) G[0][1] = Cp*G[0][0]; /* ABD eq 16 */

  for (int a=2; a<n+1; a++) G[a][0] = B1*(a-1)*G[a-2][0] + C*G[a-1][0];
  for (int b=2; b<m+1; b++) G[0][b] = B1p*(b-1)*G[0][b-2] + Cp*G[0][b-1];

  if ((m==0) || (n==0)) return;

  for (a=1; a<n+1; a++) {
    G[a][1] = a*B00*G[a-1][0] + Cp*G[a][0];
    for (b=2; b<m+1; b++)
      G[a][b] = B1p*(b-1)*G[a][b-2] + a*B00*G[a-1][b-1] + Cp*G[a][b-1];
  }
  return;
}
```

```
static void RecurFactorsGamess(double t, double A,
  double B, double Px, ouble Qx, double xi, double xk)
{
   double fff = t/(A+B)/(1+t);
   B00 = 0.5*fff;
   B1 = 1/(2*A*(1+t)) + 0.5*fff;
   B1p = 1/(2*B*(1+t)) + 0.5*fff;
   C = (Px-xi)/(1+t) + (B*(Qx-xi)+A*(Px-xi))*fff;
   Cp = (Qx-xk)/(1+t) + (B*(Qx-xk)+A*(Px-xk))*fff;
   return;
}
```