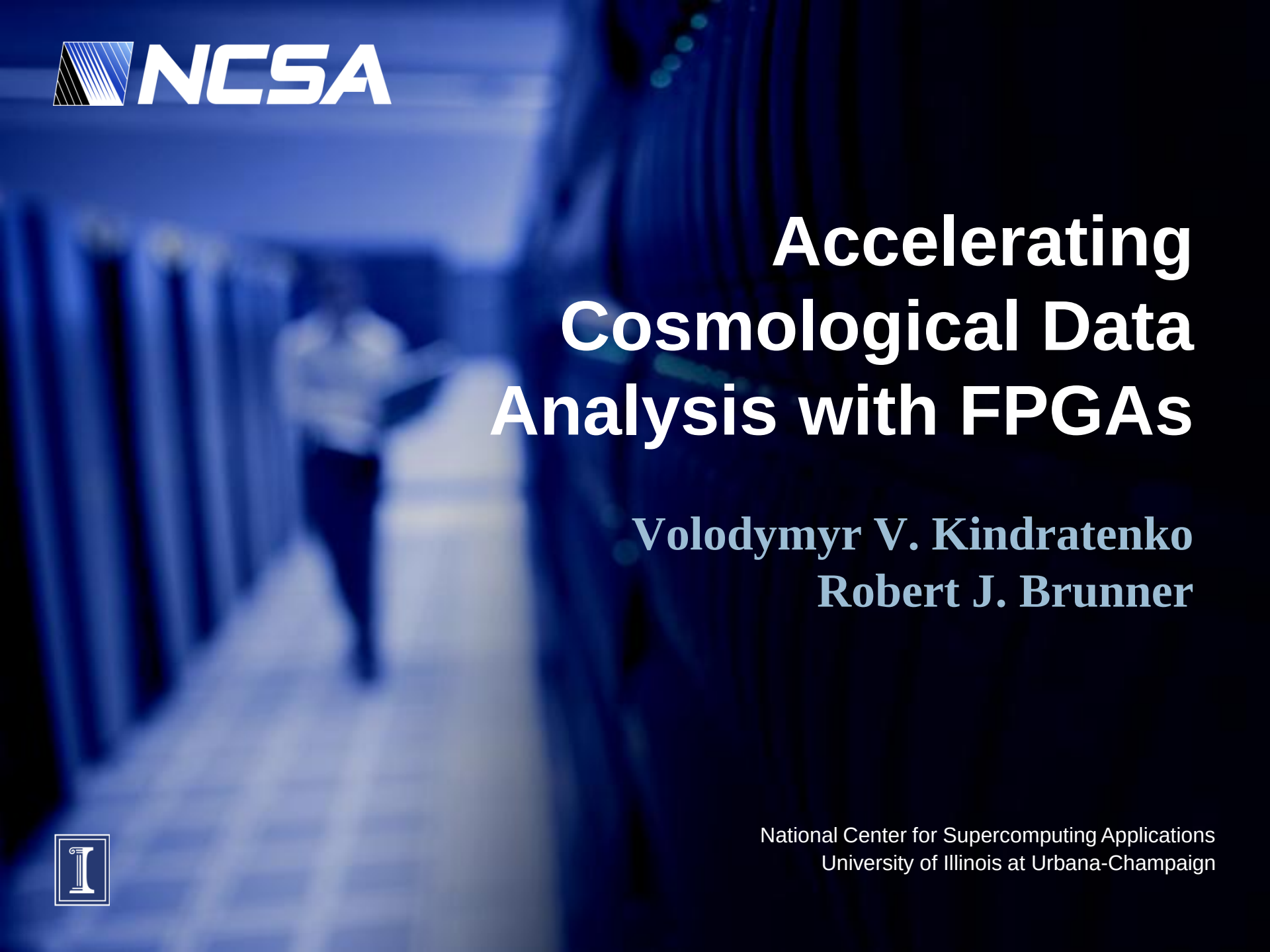


Accelerating Cosmological Data Analysis with FPGAs

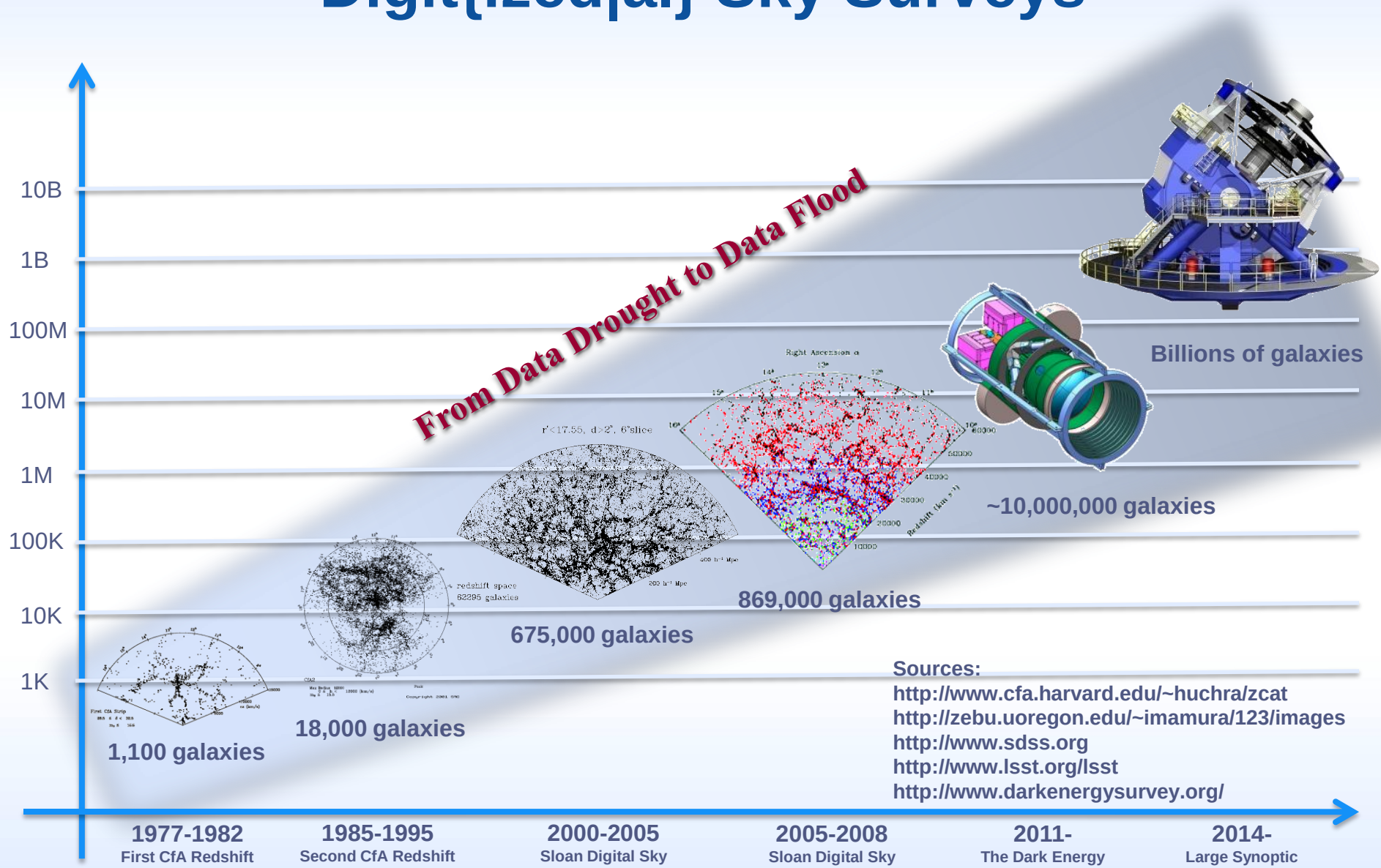
Volodymyr V. Kindratenko
Robert J. Brunner

National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign

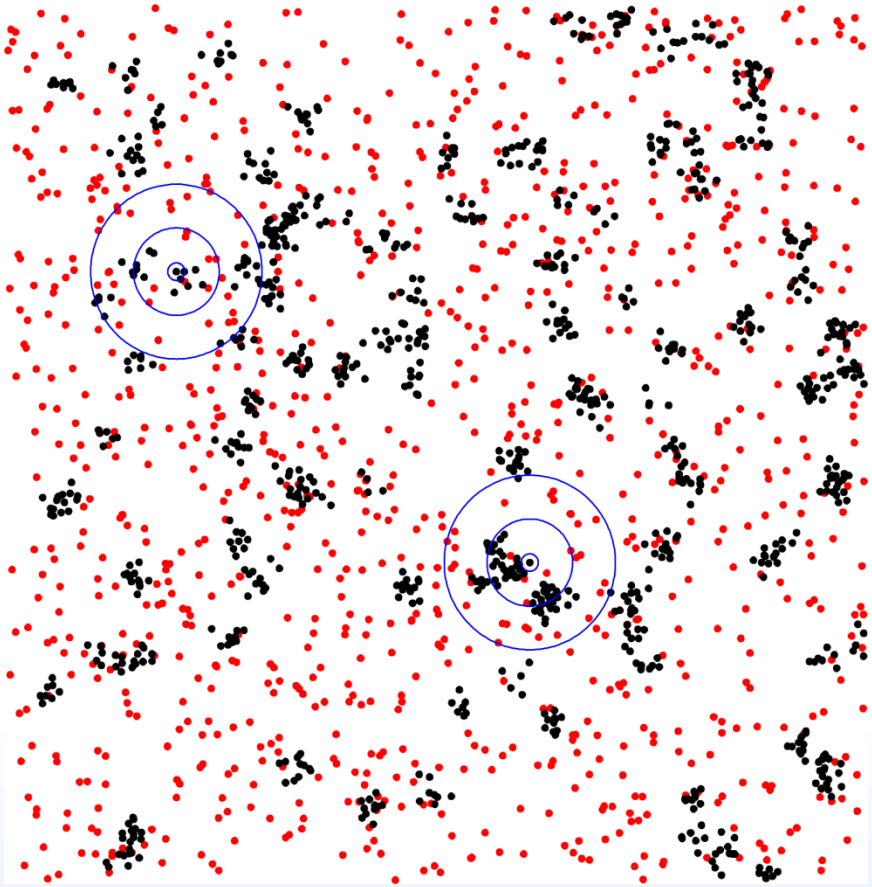
Presentation Outline

- **Motivation**
  - Digital sky surveys
- **Angular Correlation function**
  - Concept
  - Algorithm
- **Nallatech H101 implementation**
  - Suitability of FPGAs
  - Kernel implementation
- **Results**
- **Comparison with other systems**

Digit{ized|al} Sky Surveys



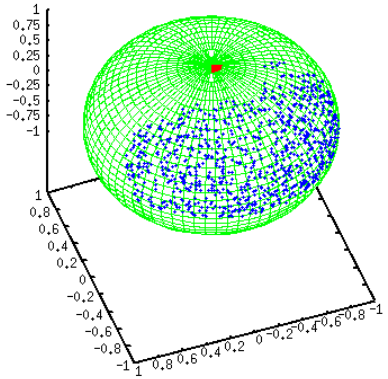
Example Analysis: Angular Correlation



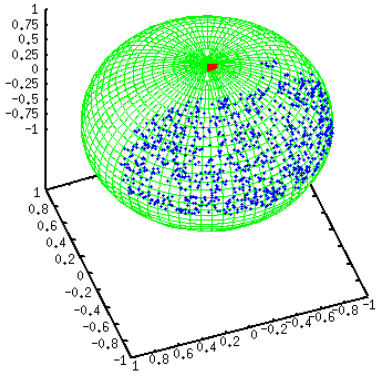
- **Two-point angular correlation function (TPACF), ω(θ), describes the frequency distribution of angular separations θ between celestial objects in the interval (θ, θ + δθ)**
- **Red (random data) are, on average, randomly distributed, black (observed data) are clustered**
  - random points: ω(θ)=0
  - observed points: ω(θ)>0
- **TPACF can vary as a function of angular distance, θ (blue circles)**
  - random: ω(θ)=0 on all scales
  - observed: ω(θ) is larger on smaller scales

Two Point Angular Correlation Function

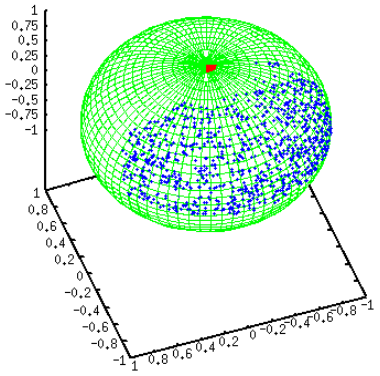
observed data



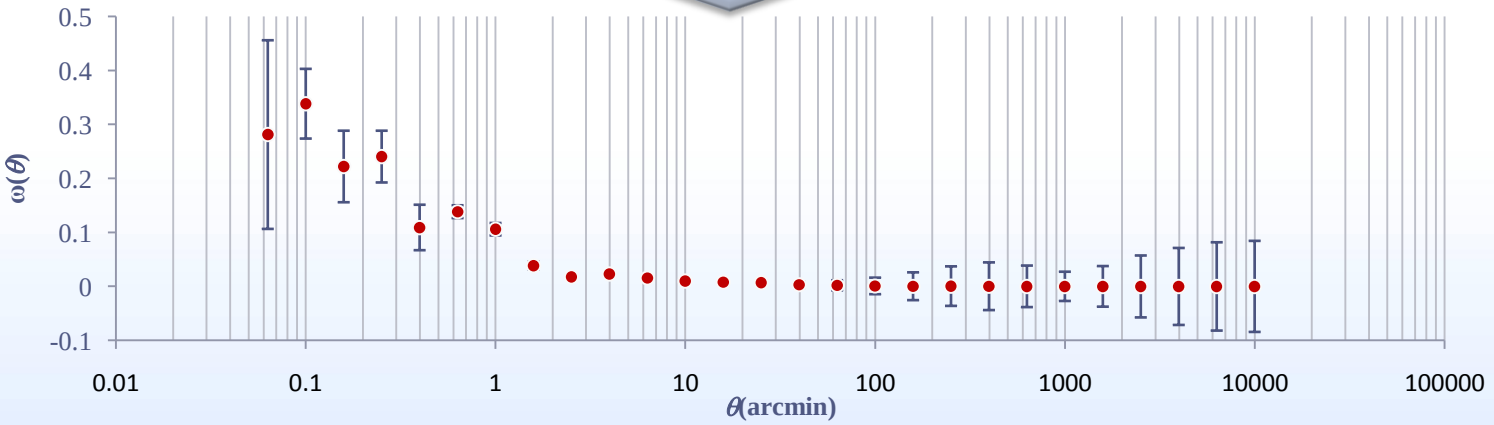
random dataset 1



random dataset n_R



$$\omega(\theta) = \frac{n_R \cdot DD(\theta) - 2\sum_{i=0}^{n_R-1} DR_i(\theta)}{\sum_{i=0}^{n_R-1} RR_i(\theta)} + 1$$



TPACF Algorithm

- **Modified Landy & Szalay estimator**

$$\omega(\theta) = \frac{n_R \cdot DD(\theta) - 2\sum_{i=0}^{n_R-1} DR_i(\theta)}{\sum_{i=0}^{n_R-1} RR_i(\theta)} + 1$$

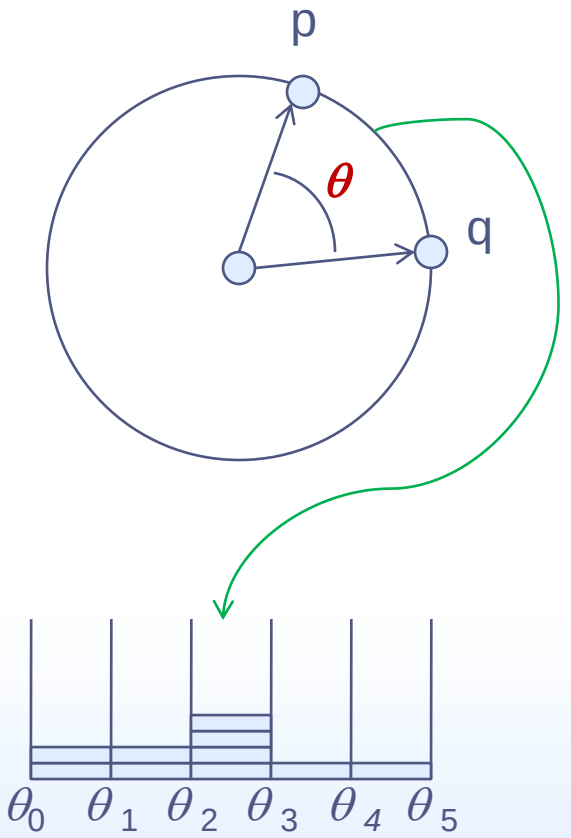
- **Angular distance**

$$\theta = \arccos(\ p \cdot q) = \arccos(\ x_p x_q + y_p y_q + z_p z_q)$$

- **Bin edges**

$$binedge_j = \cos(10^{\log_{10}\theta_{\min} + j/k}),\ j = 0,\ldots,M$$

- **Computing DD/DR/RR counts**



Bin Count Kernels

***Algorithm 1**: DD and RR bin counts*

Input:
1) *N* points x_1, .., x_N
2) for each x_i, jackknife sample number
3) *M* bins $[\theta_0, \theta_1), [\theta_1, \theta_2), .., [\theta_{M-1}, \theta_M)$

Output: bin counts

```
1  for i=1,..., N-1 do
2     for j=i+1,..., N do
3        l←binmap(θ(x_i,x_j), θ)
4        for k=1,..., K do
5           if s_i ≠ k do B_{k,l}← B_{k,l} +1
```

***Algorithm 2**: DR bin counts*

Input:
1) 2 sets of *N* points x_1, .., x_N and y_1, .., y_N
2) for each x_i, jackknife sample number
3) *M* bins $[\theta_0, \theta_1), [\theta_1, \theta_2), .., [\theta_{M-1}, \theta_M)$

Output: bin counts

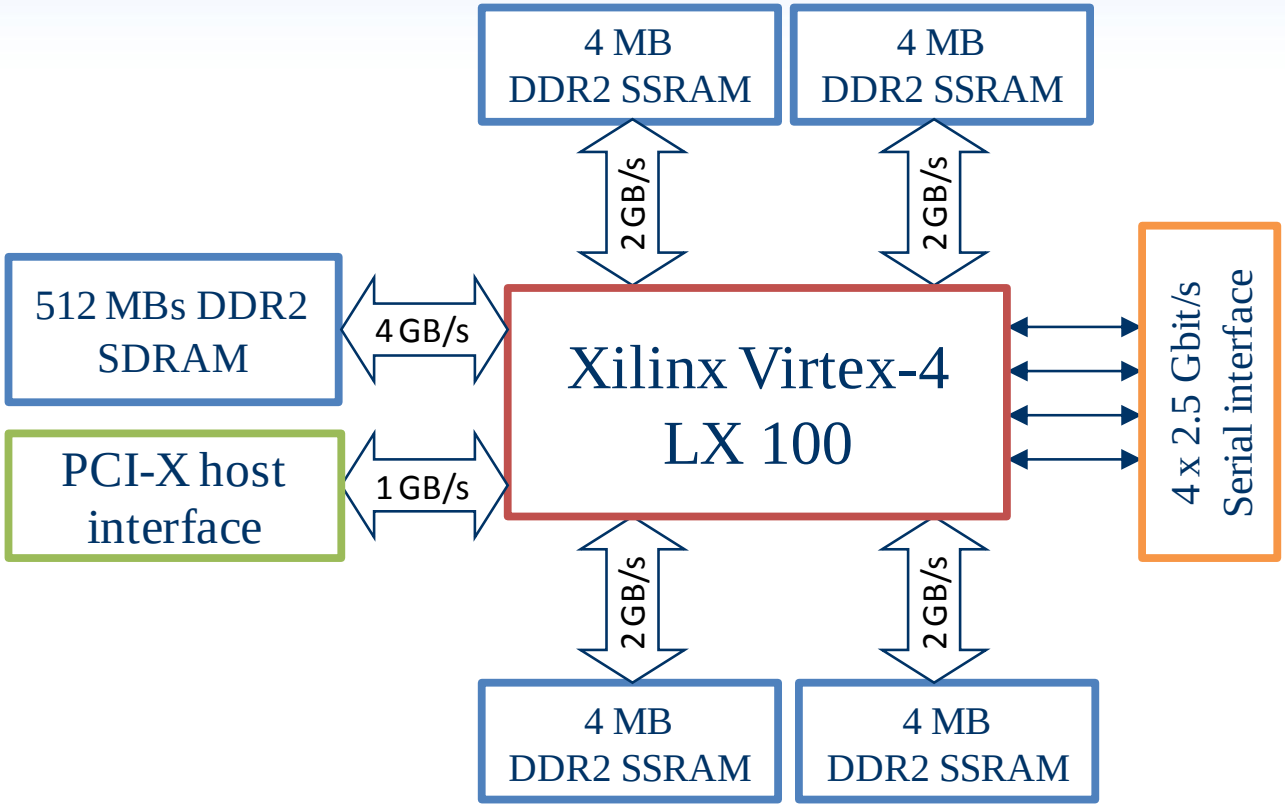
```
1  for i=1,..., N do
2     for j= 1,..., N do
3        l←binmap(θ(x_i,y_j), θ)
4        for k=1,..., K do
5           if s_i ≠ k do B_{k,l}← B_{k,l} +1
```

Error Estimation via Jackknife Re-sampling

- **Conceptually**
  - Divide observed dataset into subsamples
  - For each such subsample
    - Remove it from the dataset
    - Compute correlation using the remaining samples
  - Calculate variance of computed correlation values
- **In practice**
  - Label each observed object with the subsample number
  - Update only those histograms of angular separations, $DD(\theta)$ and $DR(\theta)$, that do not belong to the given subsample

$$\textbf{if } s_i \neq k \textbf{ do } B_{k,l} \leftarrow B_{k,l} + 1$$

Target Hardware: Nallatech H101



SP performance (GFLOPS)	20
Host bandwidth (GB/s)	1.0
Local memory bandwidth (GB/s)	8 & 4
Frequency (MHz)	100

RC Amenability Test (RAT)

- **Goal: Estimate algorithm performance and resources usage before a single line of code is written**
- **RAT sub-tests**
  - Throughput
  - Numerical precision
  - Resources

B. Holland, K. Nagarajan, C. Conger, A. Jacobs, and A. George, “RAT: A Methodology for Predicting Performance in Application Design Migration to FPGAs”, in Proc. *High-Performance Reconfigurable Computing Technologies & Applications Workshop*, November, 2007.

RAT: Throughput

		Alg1	Alg2
Application Dataset Parameters			
$N_{elements}$, input	elements	32,768	65,536
$N_{bytes/element}$, input	bytes/element	32	
$N_{elements}$, output	elements	320	
$N_{bytes/element}$, output	bytes/element	8	
Nallatech H101 Communication Parameters			
$throughput_{ideal}$	MB/s	1,000	
α_{write}	$0<\alpha<1$	0.147	
α_{read}	$0<\alpha<1$	0.001	
Kernel computation parameters			
$N_{ops/element}$	ops/element	~163,840	327,680
$throughput_{proc}$	ops/cycle	10	
f_{clock}	MHz	100	
Software Implementation Parameters			
t_{soft}	sec	17.8	40.5
N_{iter}	iterations	1	

RAT: Throughput (Cont.)

$$T_{read} = (N_{elements} \times N_{bytes/element}) / (\alpha_{write} \times throughput_{ideal})$$

$$T_{write} = (N_{elements} \times N_{bytes/element}) / (\alpha_{read} \times throughput_{ideal})$$

$$T_{comm} = T_{read} + T_{write}$$

$$T_{comp} = (N_{elements} \times N_{ops/element}) / (f_{clock} \times throughput_{proc})$$

$$t_{RCSB} = N_{iter} \times (T_{comm} + T_{comp})$$

$$speedup = t_{soft} / t_{RCSB}$$

$$util_{commSB} = T_{comm} / (T_{comm} + T_{comp})$$

$$util_{compSB} = T_{comp} / (T_{comm} + T_{comp})$$

	Alg1	Alg2
t_{comm} (sec)	0.01	0.017
t_{comp} (sec)	5.369	10.737
$util_{commSB}$	0.0019	0.0016
$util_{compSB}$	0.9981	0.9984
t_{RCSB} (sec)	5.379	10.754
speedup	**3.3×**	**3.8×**

RAT: Numerical Precision

- **Bin Boundaries**
  - From 0.01 to 10000 arcmin
  - Bin boundaries (in dot product space):
    - 0.999999999995769
    - 0.999999999989373
    - 0.999999999973305
    - 0.999999999932946
    - 0.999999999831569
    - 0.999999999576920
    - 0.999999998937272
    - 0.999999997330547
    - 0.999999993294638
    - …
  - Only 12 digits after the decimal point are important
    - 41-bit fixed-point will do it
- **Single-precision floating-point (SPFP) is not sufficient to implement calculations below 1 arcmin**
- **Double-precision floating-point (DPFP) is an overkill**
- **41-bit fixed point is just right, but this data type is not supported in DIME-C**

RAT: Resources

- **Memory**
  - 2 datasets of 32,768 32-byte values = 2 MB of memory
  - 320 8-byte elements to store bin counts ~2.5 KB
- **Hardware multipliers**
  - 9 XtremeDSP slices per DPFP multiplier → 27 slices per dot product
- **Random logic elements**
  - 693 random logic slices per DPFP multiplication; 821 random logic slices per DPFP addition → 3,721 slices per dot product
  - 256 slices to store 32 bin boundaries
  - 77 slices per DPFP comparison operator → 2,310 slices per fully unrolled binary search tree for 32 elements
  - …
  - **Slice count estimate to implement major parts: 6,747**

Implementation Steps

- **Start with a reference C implementation**
- **Build DIMEtalk network**
  - DIMEtalk network editor
- **Write kernel in DIME-C**
  - DIME-C IDE
- **Compile/synthesize/implement the design**
  - DIME-C C to VHDL Function Generator
  - DIMEtalk System Design
  - Xilinx ISE
- **Write host interface using DIMEtalk API**
  - ICC

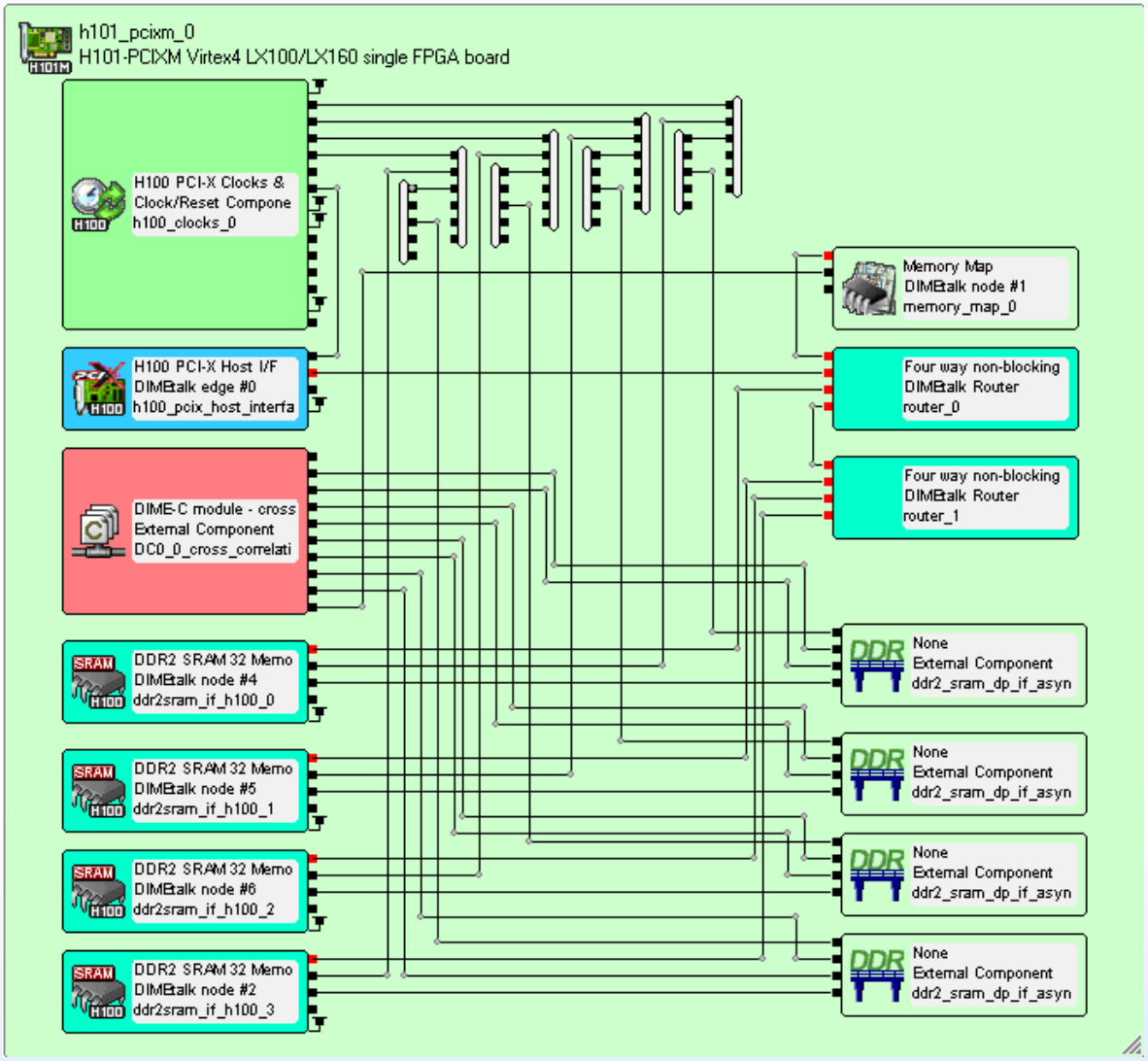
Reference Implementation

```
// pre-compute bin boundaries, binb

loadObservedData(data);
computeDD(data, npd, data, npd, 1,binb, nbins, njks, DD);

for (i = 0; i < random_count; i++) // loop through random data files
{
      loadRandomData(random[i]);
      computeRR(random[i], npr[i], random[i], npr[i], 1, binb, nbins, njks, RRS);
      computeDR(data, npd, random[i], npr[i], 0, binb, nbins, njks, DRS);
}

// compute w
for (k = 0; k < nbins; k++)  {
      ω[k] = (random_count * 2*DD[k] - DRS[k]) / RRS[k] + 1.0;
}
```

Reference Implementation (Cont.)

```
void compute{DD|DR|RR}(struct cartesian *data1, int n1, struct cartesian *data2, int n2,
                       int doSelf, int nbins, double *binb, int njk, long long **data_bins)
{
   if (doSelf) { n2 = n1; data2 = data1; }            // setup pointers for Self-compute

   for (i = 0; i < ((doSelf) ? n1-1 : n1); i++) {     // loop over points in the first set
      double xi = data1[i].x;                         // get point from first dataset
      double yi = data1[i].y;
      double zi = data1[i].z;
      int jk = data1[i].jk;

      for (j = ((doSelf) ? i+1 : 0); j < n2; j++) {   // loop in second dataset
         double dot = xi * data2[j].x + yi * data2[j].y + zi * data2[j].z;   // dot product

         int indx, k = nbins;                         // find bin it belongs to
         if (dot >= binb[0]) indx = 0;                // eliminate those outside the range
         else {  while (dot > binb[k]) k--; indx = k+1; }   // sequential search

         for (l = 0; l < njk; l++)
            if (l != jk) data_bins[l][indx] += 1;     // update all but jk bins
      }
   }
}
```

DIMEtalk Network



Kernel Written in DIME-C

```
01  #define NBINS 32
02  #define NJK 10
03  #define N1 32768
04  #define N2 32768
05  #define N1pN2 65536
06
07  void cross_correlation (sram double X1[N1pN2], sram double X2[N1pN2], sram double Y1[N1pN2],
08     sram double Y2[N1pN2], sram double Z1[N1pN2], sram double Z2[N1pN2], sram double JK[N1pN2],
09     sram double BINV[N1pN2], int n1, int n2, int nb, int njk, int doSelf)
10  {
11    int i, j, k, jk, offset, indx, nb1, nb2, i_end, j_start, val;
12    double x1, y1, z1, x2, y2, z2, dot;
13    double binb00, binb01, binb02, binb03, binb04, binb05, binb06, binb07, binb08, binb09, binb10, binb11;
14    double binb12, binb13, binb14, binb15, binb16, binb17, binb18, binb19, binb20, binb21, binb22, binb23;
15    double binb24, binb25, binb26, binb27, binb28, binb29, binb30;    // NBINS
16    int binv1_00[NBINS], binv2_00[NBINS], binv3_00[NBINS], binv4_00[NBINS];  // NJK
…
```

Kernel Written in DIME-C (Cont.)

```
28
29   nb1 = nb + 1;  nb2 = nb + 2;    // make a local copy of bin boundaries

30   for (i = 0; i < nb1; i++) {  // copy bin boundaries
31      binb00 = binb01; binb01 = binb02; binb02 = binb03; binb03 = binb04; binb04 = binb05; binb05 = binb06;
32      binb06 = binb07; binb07 = binb08; binb08 = binb09; binb09 = binb10; binb10 = binb11; binb11 = binb12;
33      binb12 = binb13; binb13 = binb14; binb14 = binb15; binb15 = binb16; binb16 = binb17; binb17 = binb18;
34      binb18 = binb19; binb19 = binb20; binb20 = binb21; binb21 = binb22; binb22 = binb23; binb23 = binb24;
35      binb24 = binb25; binb25 = binb26; binb26 = binb27; binb27 = binb28; binb28 = binb29; binb29 = binb30;
36      binb30 = BINV[n1 + i];
37   }

38   for (i = 0; i < nb2; i++) {  // clear BRAM for bin counts
39      binv1_00[i] = 0; binv2_00[i] = 0; binv3_00[i] = 0; binv4_00[i] = 0;
…
50   }

51   if (doSelf) i_end = n1-1; else i_end = n1;
52
```

Kernel Written in DIME-C (Cont.)

```
53  for (i = 0; i < i_end; i++) {        // loop thru one dataset

54      x1 = X1[i];         y1 = Y1[i];         z1 = Z1[i];         jk = (int)JK[i];
55      if (doSelf) j_start = i+1; else j_start = 0;

56      for (j = j_start; j < n2; j++) {        // loop thru second dataset

57        if (doSelf) offset = j; else offset = n1 + j;

58        x2 = X2[offset];         y2 = Y2[offset];         z2 = Z2[offset];
59        dot = x1 * x2 + y1 * y2 + z1 * z2;            // dot product

60        // unrolled binary search
61        if (dot < binb15) { if (dot < binb23) { if (dot < binb27) { if (dot < binb29) { if (dot < binb30) indx = 31;
62        else indx = 30; } else { if (dot < binb28) indx = 29; else indx = 28; } } else { if (dot < binb25)  {
63        if (dot < binb26) indx = 27;  else indx = 26; } else { if (dot < binb24) indx = 25; else indx = 24; } } }
64        else { if (dot < binb19) { if (dot < binb21) { if (dot < binb22) indx = 23; else indx = 22; } else {
65        if (dot < binb20) indx = 21; else indx = 20; } } else { if (dot < binb17) { if (dot < binb18) indx = 19;
66        else indx = 18; } else { if (dot < binb16) indx = 17; else indx = 16; } } } }
67        else { if (dot < binb07) { if (dot < binb11) { if (dot < binb13) { if (dot < binb14) indx = 15; else indx = 14; }
68        else { if (dot < binb12) indx = 13; else indx = 12; } } else { if (dot < binb09) { if (dot < binb10) indx = 11;
69        else indx = 10; } else { if (dot < binb08) indx = 9; else indx = 8; } } } else { if (dot < binb03) { if (dot < binb05) {
70        if (dot < binb06) indx = 7; else indx = 6; } else { if (dot < binb04) indx = 5; else indx = 4; } } else {
71        if (dot < binb01) { if (dot < binb02) indx = 3; else indx = 2; } else { if (dot < binb00) indx = 1; else indx = 0; }}}}
…
```

Kernel Written in DIME-C (Cont.)

```
72
73        bin_bank = j % 4;  // update bin values
74        if (jk != 0) val0 = 1; else val0 = 0;            // update corresponding bin for jk=0
75        if (bin_bank == 0) binv1_00[indx] = binv1_00[indx] + val0;
76        else if (bin_bank == 1) binv2_00[indx] = binv2_00[indx] + val0;
77        else if (bin_bank == 2) binv3_00[indx] = binv3_00[indx] + val0;
76        else binv4_00[indx] = binv4_00[indx] + val0;
…
127     }
128  }
129
130  for (i = 0; i < nb2; i++)     {    // copy results back to SRAM
131        BINV[i] = (double)((binv1_00[i] + binv2_00[i]) + (binv3_00[i] + binv4_00[i]));
132        BINV[nb2+i] = (double)((binv1_01[i] + binv2_01[i]) + (binv3_01[i] + binv4_01[i]));
133        BINV[2*nb2+i] = (double)((binv1_02[i] + binv2_02[i]) + (binv3_02[i] + binv4_02[i]));
134        BINV[3*nb2+i] = (double)((binv1_03[i] + binv2_03[i]) + (binv3_03[i] + binv4_03[i]));
135        BINV[4*nb2+i] = (double)((binv1_04[i] + binv2_04[i]) + (binv3_04[i] + binv4_04[i]));
136        BINV[5*nb2+i] = (double)((binv1_05[i] + binv2_05[i]) + (binv3_05[i] + binv4_05[i]));
137        BINV[6*nb2+i] = (double)((binv1_06[i] + binv2_06[i]) + (binv3_06[i] + binv4_06[i]));
138        BINV[7*nb2+i] = (double)((binv1_07[i] + binv2_07[i]) + (binv3_07[i] + binv4_07[i]));
139        BINV[8*nb2+i] = (double)((binv1_08[i] + binv2_08[i]) + (binv3_08[i] + binv4_08[i]));
140        BINV[9*nb2+i] = (double)((binv1_09[i] + binv2_09[i]) + (binv3_09[i] + binv4_09[i]));
141        BINV[10*nb2+i] = (double)((binv1_10[i] + binv2_10[i]) + (binv3_10[i] + binv4_10[i]));
142  }
143 }
```

Results: Resources Utilization

	Estimated	Actual	Total available
DSP48s	27	36	96
RAMB16s		79	240
Slices	6,747	23,893	49,152

Conclusions:

1. Based on the algorithm description only, reliable resource utilization estimate is not possible
   - FPGA/board limitations/features are not fully considered
   - DIME-C adds overhead
   - DIMEtalk network ~10,000 slices
2. RAT at best gives an indication if the design might not fit

Results: Data Transfer Time

Actual

	Alg1	Alg2
Data in (bytes)	1,048,824	1,835,266
Data in (sec)	0.0048	0.0067
Data out (bytes)	2,816	2,816
Data out (sec)	0.000057	0.000046

Predicted

t_{comm} (sec)	0.01	0.017

Lesson:

1. Run your own micro-benchmarks to probe the sustained I/O bandwidth

Results: Kernel Execution Time

kernel	CPU	FPGA (actual)		FPGA (predicted)	
		time	speedup	time	speedup
DD (sec)	17.8	5.436	3.3×	5.379	**3.3×**
RR (sec)	10.8	5.436	2.0×		
DR (sec)	40.5	10.816	3.8×	10.754	**3.8×**

Conclusion:

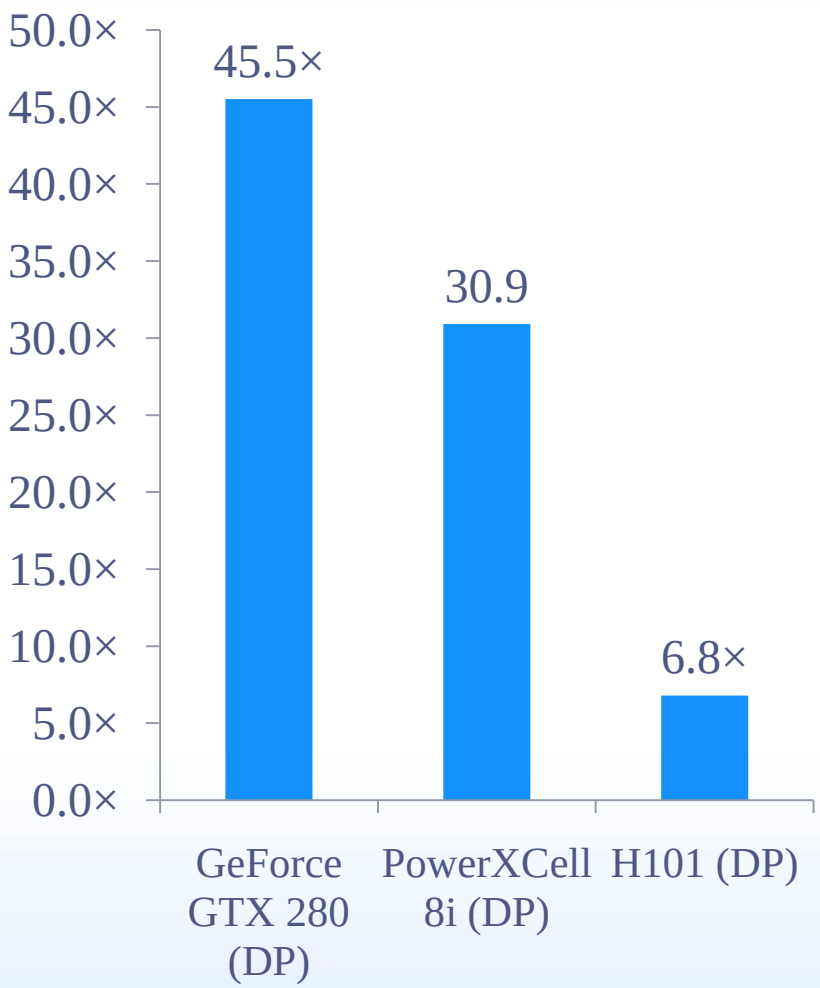
1. RAT can be used to accurately predict kernel execution time as long as one can implement the kernel in the same way as he thought it could be done when applying RAT throughput test

Results: a 97K Dataset

kernel	CPU	FPGA design 1		FPGA design 2	
		Time	speedup	time	speedup
DD, *sec*	153.5	47.3	3.3×	23.8	6.5×
RR, *sec*	103.3	47.3	2.2×	23.8	4.3×
DR, *sec*	375.8	94.6	4×	47.5	7.9×
total, *sec*	632.6	189.2	3.3×	95.1	**6.7×**

"FPGA design 2" consists of 2 compute kernels implemented on the FPGA

Speedup vs. 2.4 GHz AMD Operon



- **An unfair comparison**
  - V4 LX100 is a 90 nm chip (and not the largest in the family)
  - GTX-280 and PowerXCell 8i are 65 nm chips
  - DIME-C/DIMEtalk consumes additional resources and limits design frequency to 100 MHz
    - The price to pay for using a high-level language…

Acknowledgements

- **Funding**
  - NSF STCI (OCI 08-10563)
  - NASA AISR (NNG06GH15G)
- **Technical support**
  - Nallatech
  - NCSA Innovative Systems Lab's Jeremy Enos

Source code at
http://www.ncsa.uiuc.edu/~kindr/projects/hpca/