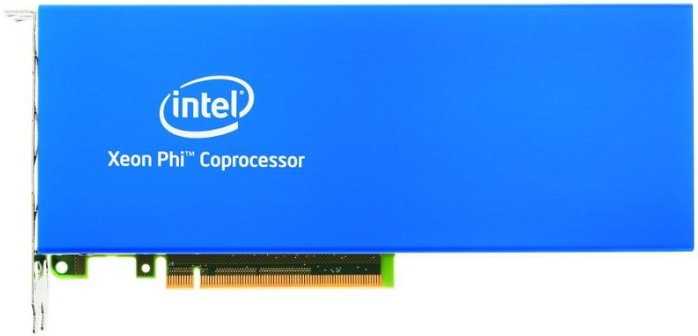
FROM SCALAR & SERIAL TO VECTOR & PARALLEL

Hands-on Lab

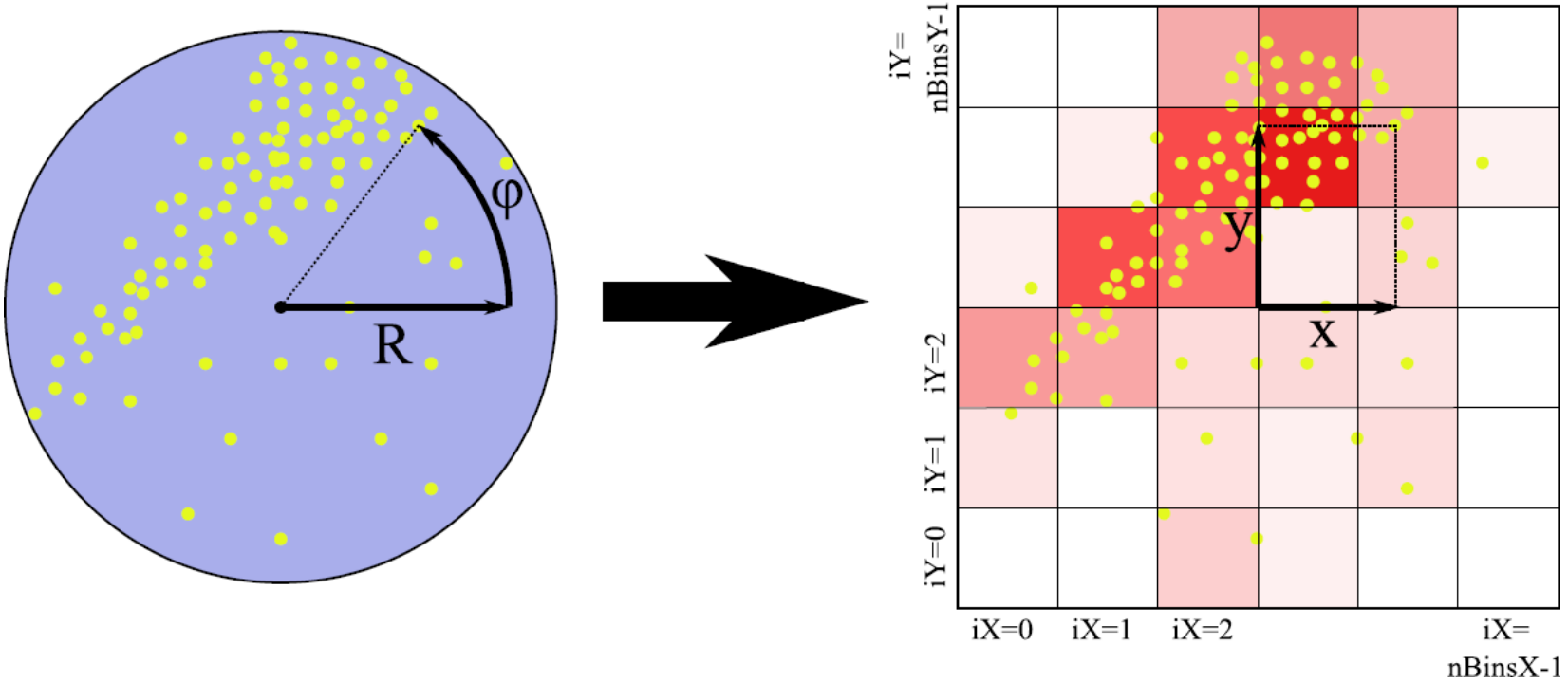
Part 2 of 3

GOALS

- Part 1: see multi-threading on Intel architecture in action
- **Part 2: learn to vectorize & future-proof vectorization**
- Part 3: experiment with what it takes to make the memory subsystem happy

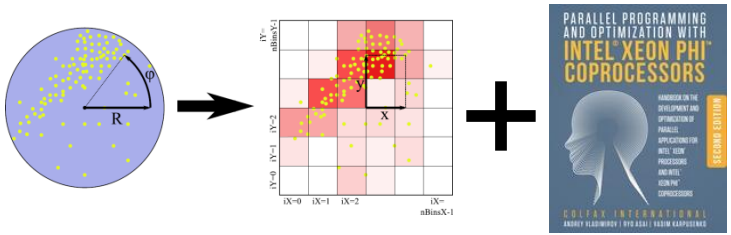


EXAMPLE PROBLEM: BINNING

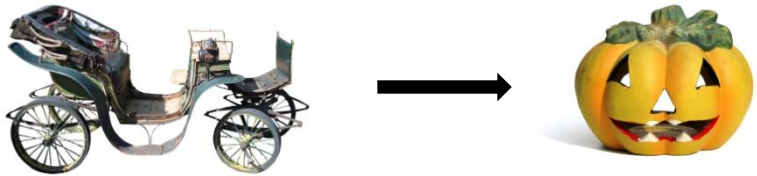


SERVER FOR EXERCISES

- Instructions at **uni.colfax-intl.com/cdt**
- Find code of lab + exercises from **xeonphi.com/book**



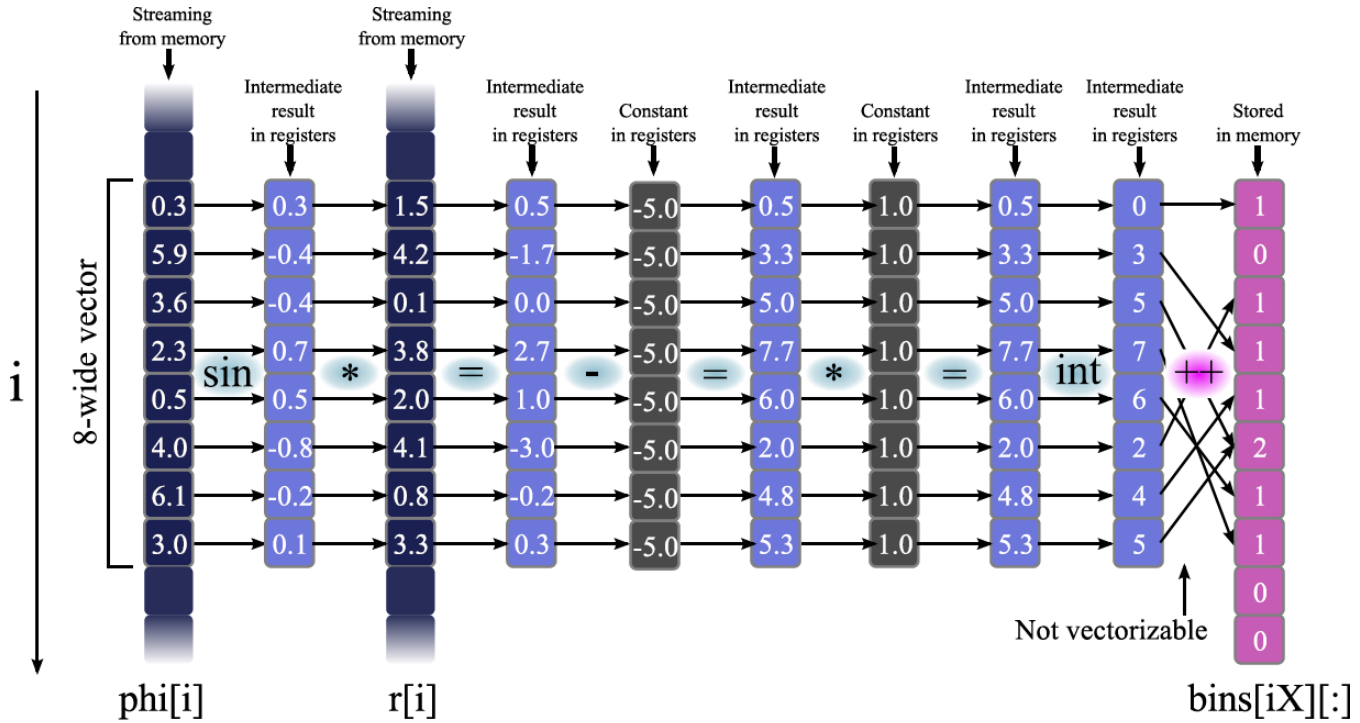
- Enjoy remote access until midnight



INITIAL APPROACH

```
 // Reference implementation: scalar, serial code without optimization
void BinParticlesReference(
        const InputDataType  & inputData, BinsType & outputBins) {
  // Loop through all particle coordinates
  for (int i = 0; i < inputData.numDataPoints; i++) {
    // Transforming from cylindrical to Cartesian coordinates:
    const FTYPE x = inputData.r[i]*COS(inputData.phi[i]);
    const FTYPE y = inputData.r[i]*SIN(inputData.phi[i]);

    // Calculating the bin numbers for these coordinates:
    const int iX = int((x - xMin)*binsPerUnitX);
    const int iY = int((y - yMin)*binsPerUnitY);

    // Incrementing the appropriate bin in the counter:
    outputBins[iX][iY]++;
  }
}
```

VECTORIZATION OPPORTUNITY



STRIP-MINING TO THE RESCUE

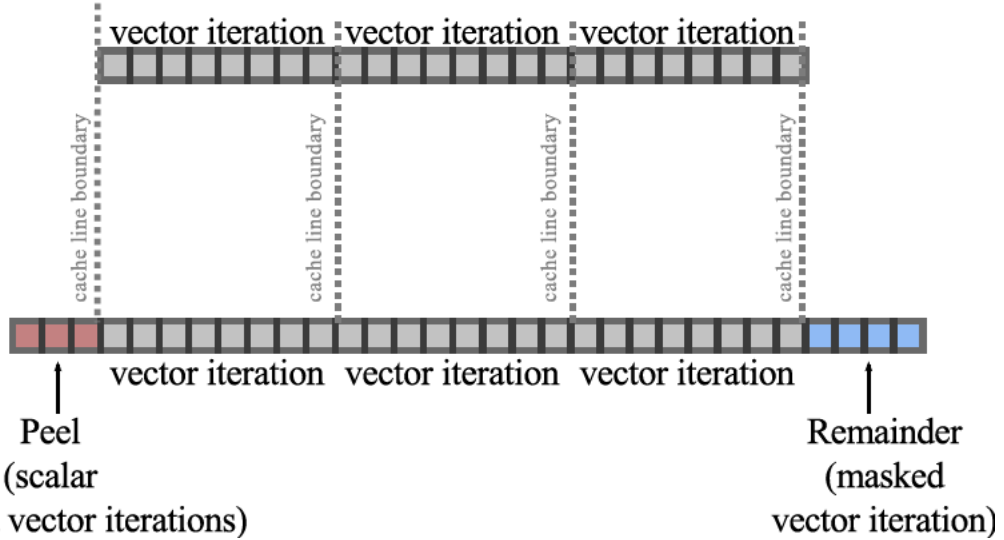
```
const int STRIP_WIDTH = 16;
for (int ii = 0; ii < inputData.numDataPoints; ii += STRIP_WIDTH) {

  int iX[STRIP_WIDTH], iY[STRIP_WIDTH];
  const FTYPE* r   = &(inputData.r[ii]);
  const FTYPE* phi = &(inputData.phi[ii]);

  for (int c = 0; c < STRIP_WIDTH; c++) { // Vector loop
    const FTYPE x = r[c]*COS(phi[c]); // Transforming from cylindrical
    const FTYPE y = r[c]*SIN(phi[c]); // to Cartesian coordinates
    iX[c] = int((x - xMin)*binsPerUnitX); // Calculating the bin numbers
    iY[c] = int((y - yMin)*binsPerUnitY); // for these coordinates
  }

  for (int c = 0; c < STRIP_WIDTH; c++) // Scalar loop
    threadPrivateBins[iX[c]][iY[c]]++;
}
```

FINE-TUNING VECTORIZATION

```
for (i = 0; i < n; i++)  A[i] = ...
```

Code Path 1:
data aligned from iteration 0,
n is multiple of vector length

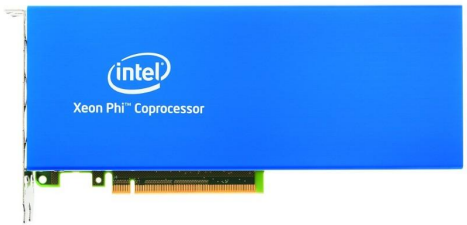
Code Path 2:
data aligned from iteration 3,
n is not a multiple of vector length

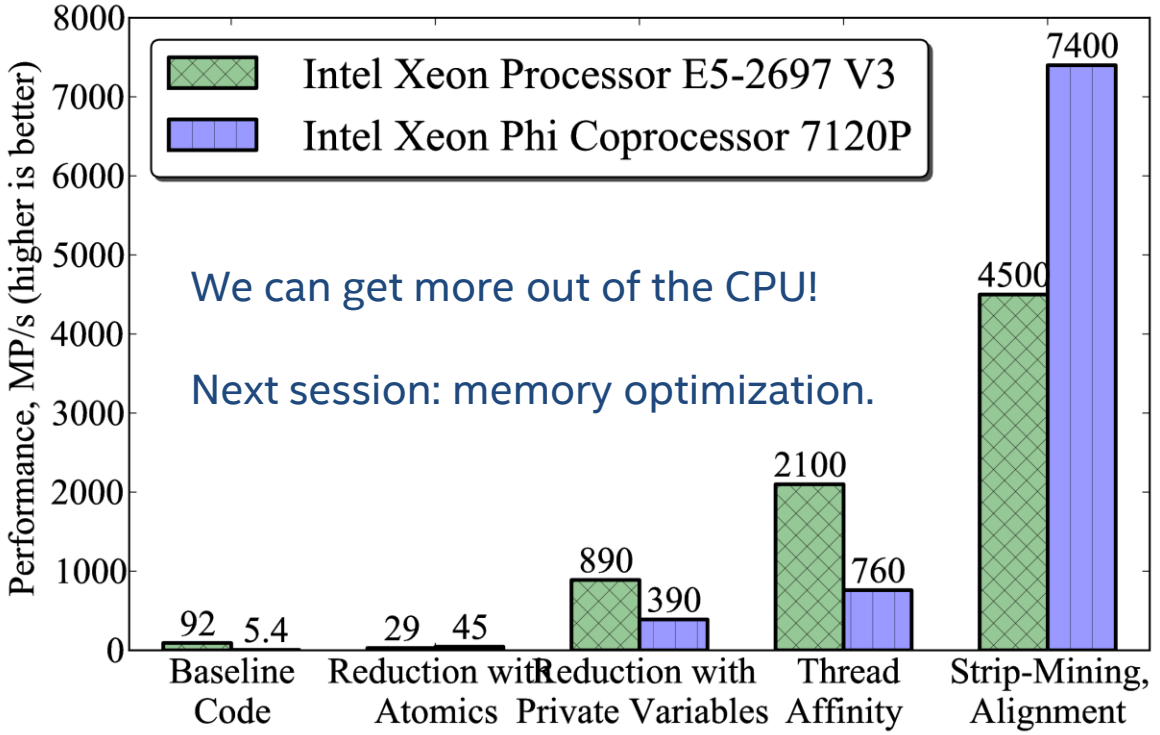


BETTER VECTORIZATION

```
// Allocating data on a 64-byte aligned memory heap address
rawData.r   = (FTYPE*) _mm_malloc(sizeof(FTYPE)*n, 64);
rawData.phi = (FTYPE*) _mm_malloc(sizeof(FTYPE)*n, 64);

// Later in the code:
for (int ii = 0; ii < inputData.numDataPoints; ii += STRIP_WIDTH) {

   int iX[STRIP_WIDTH] __attribute__((aligned(64))); // Aligned allocation
   int iY[STRIP_WIDTH] __attribute__((aligned(64))); // on the stack

   // …

   // Compiler hint: we promise alignment, no need for peeling
#pragma vector aligned
   for (int c = 0; c < STRIP_WIDTH; c++) {
     // …
}
```

PERFORMANCE RESULTS





NEXT SESSION

- Part 3: making the memory subsystem happy

By the way:
download this tutorial
colfaxresearch.com