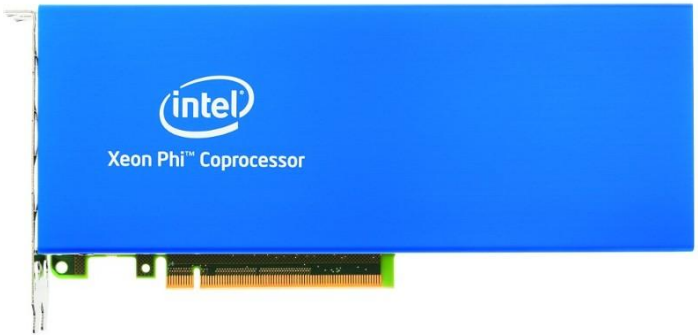
FROM SCALAR & SERIAL TO VECTOR & PARALLEL

Hands-on Lab

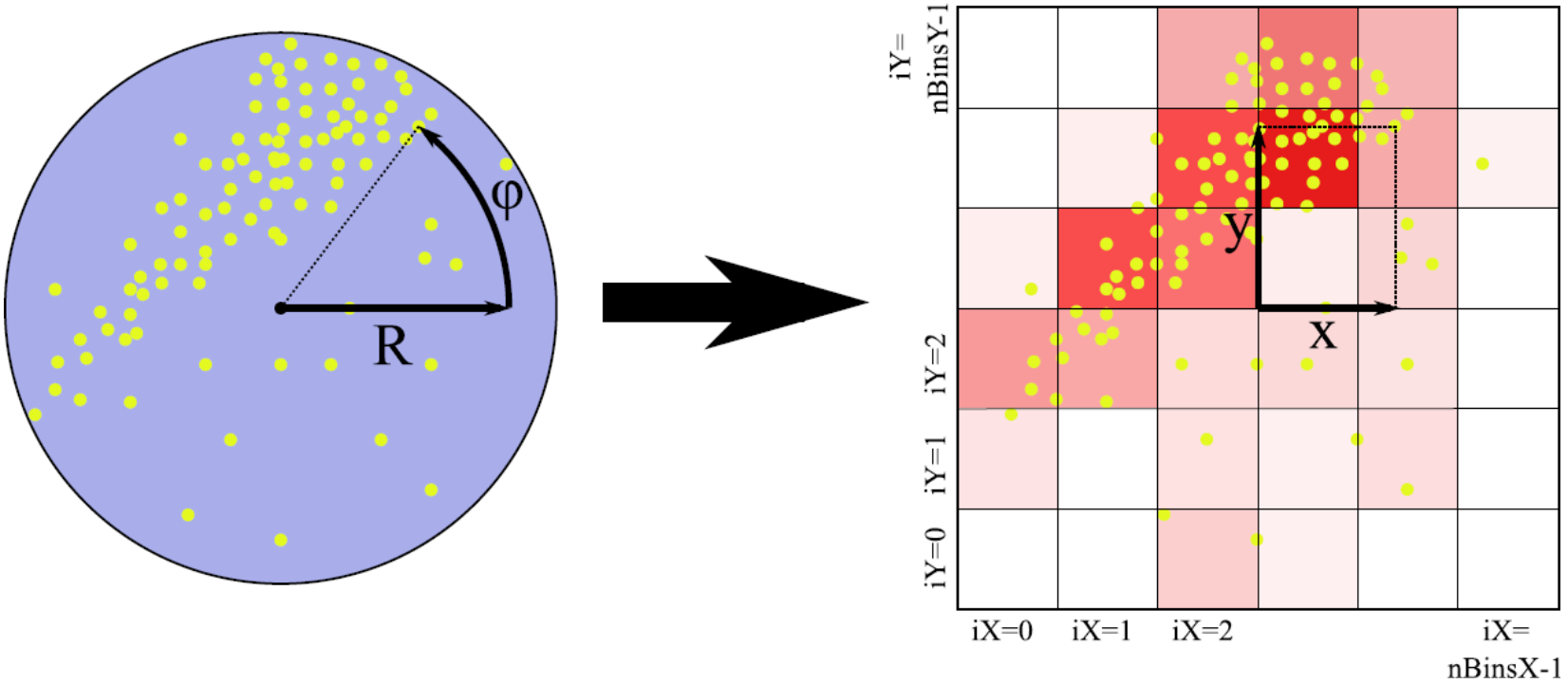
Part 3 of 3

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**

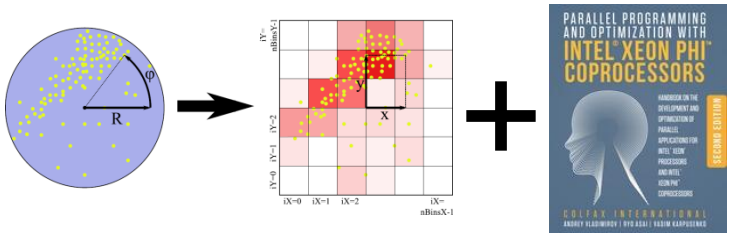


EXAMPLE PROBLEM: BINNING

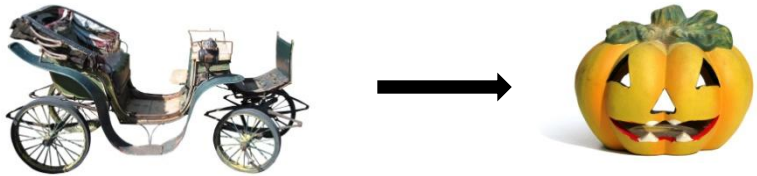


SERVER FOR EXERCISES

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



- Enjoy remote access until midnight



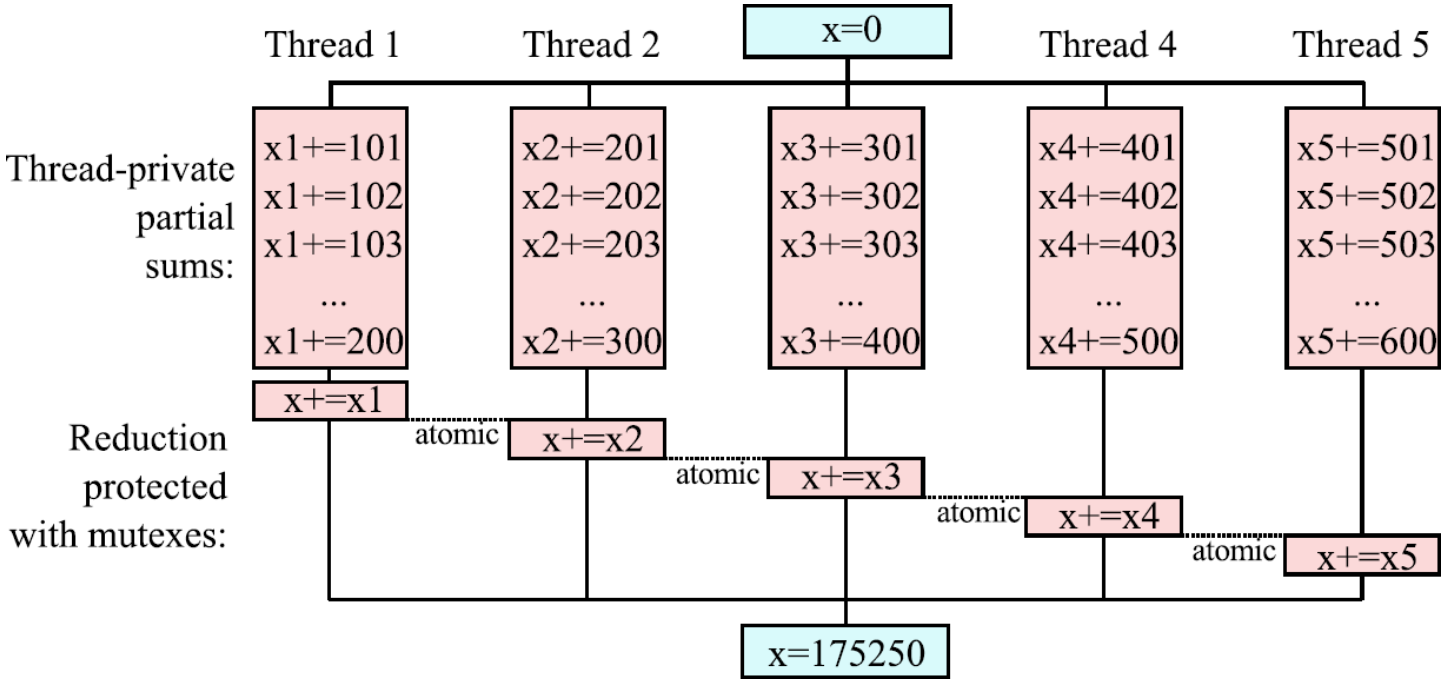
INITIAL APPROACH

```
 // 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]++;
  }
}
```

PARALLEL REDUCTION

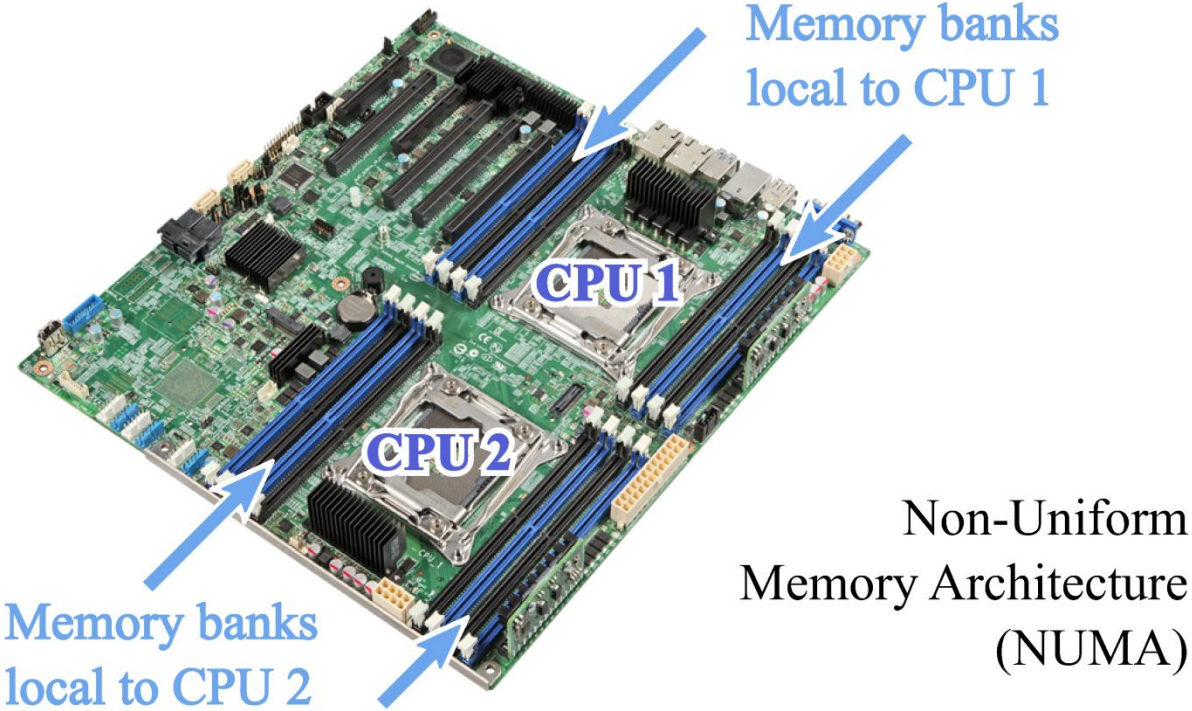


Key: using thread-private containers for partial sums; mutexes only after the parallel loop

MULTI-THREADED VERSION

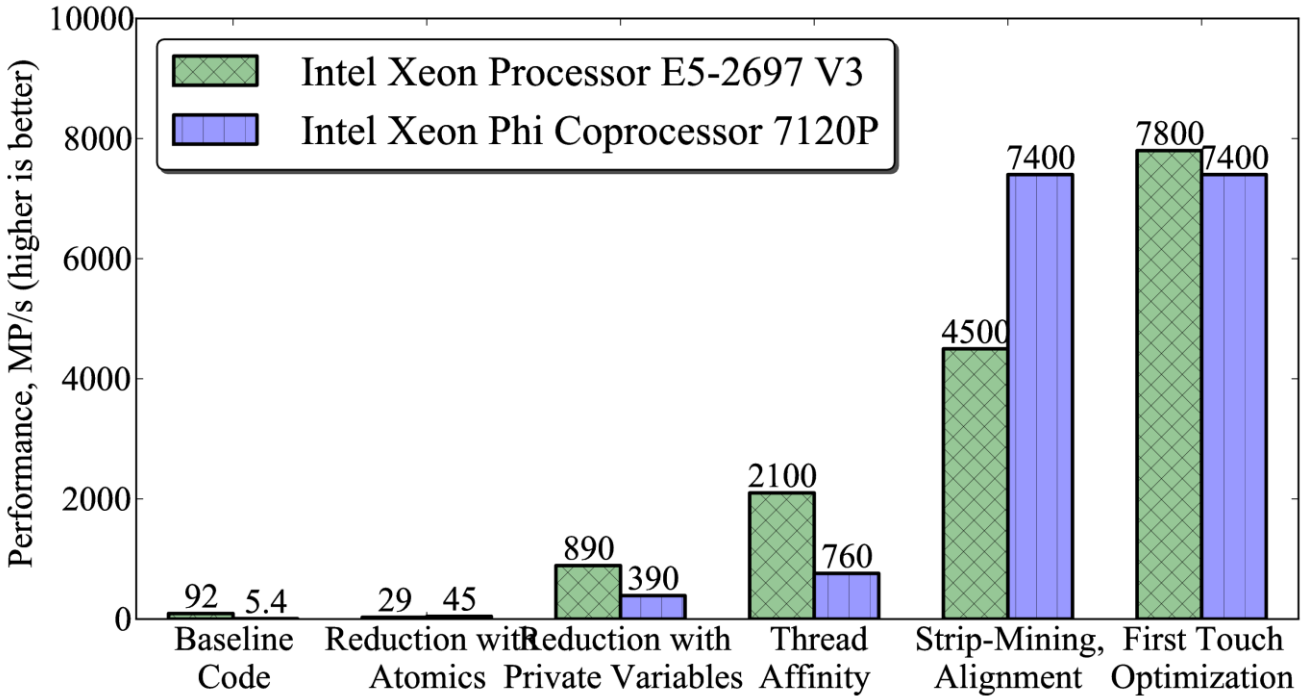
```
void BinParticles_2(const InputDataType& inputData, BinsType& outputBins){
#pragma omp parallel
  { BinsType threadPrivateBins; // Declare thread-private containers
    for (int i = 0; i < nBinsX; i++)
      for (int j = 0; j < nBinsY; j++)
        threadPrivateBins[i][j] = 0;
#pragma omp for
    for (int i = 0; i < inputData.numDataPoints; i++) {
      // …transforming from cylindrical to Cartesian coordinates:
      // …calculating the bin numbers for these coordinates:
      // Incrementing the appropriate bin in the thread-private counter:
      threadPrivateBins[iX][iY]++;
    }
    for(int i = 0; i < nBinsX; i++) // Reduction outside the parallel loop
      for(int j = 0; j < nBinsY; j++)
#pragma omp atomic
        outputBins[i][j] += threadPrivateBins[i][j];
  } }
```

FIRST-TOUCH ALLOCATION

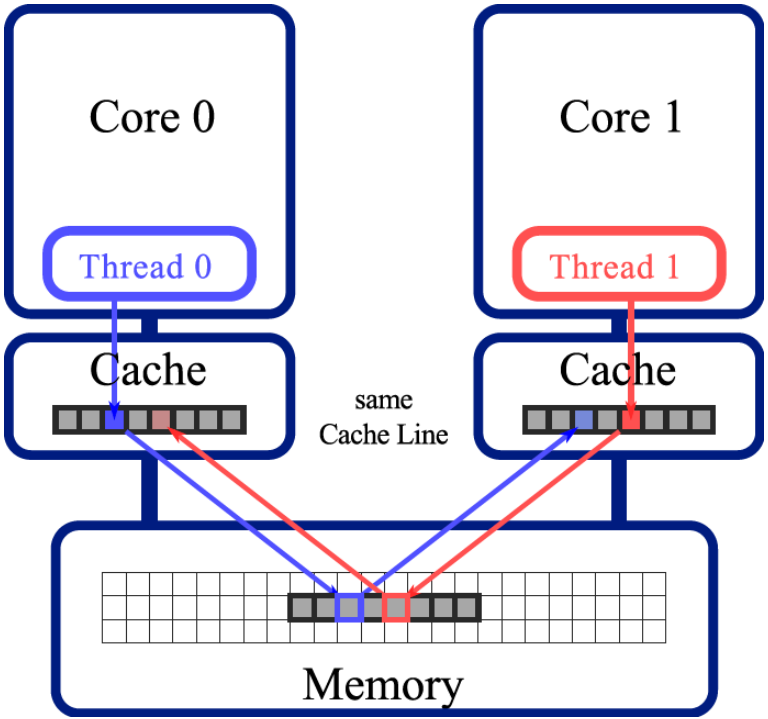


Non-Uniform
Memory Architecture
(NUMA)

RESULT OF MEMORY TRAFFIC TUNING



FALSE SHARING



GLOBAL CONTAINERS

```
// Using a global container in
// threads-first layout
int nThrds=omp_get_max_threads();

// Instead of storing scalars in
// each bin, store an array with
// values for each thread
int glBins[nBinsX][nBinsY][nThrds];

#pragma omp parallel
{
  int iThd = omp_get_thread_num();
  // … later, memory access:
  gllBins[iX[c]][iY[c]][iThrd]++;
}
```

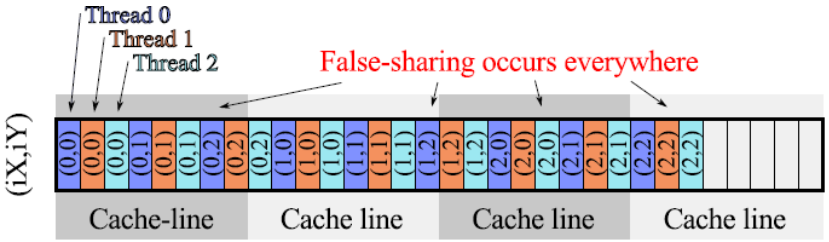
```
// Using a global container in
// threads-last layout
int nThrds=omp_get_max_threads();

// Instead of storing scalars in
// each bin, store an array with
// values for each thread
int glBins[nThrds][nBinsX][nBinsY];

#pragma omp parallel
{
  int iThd = omp_get_thread_num();
  // … later, memory access:
  gllBins[iThrd][iX[c]][iY[c]] ++;
}
```

FALSE SHARING

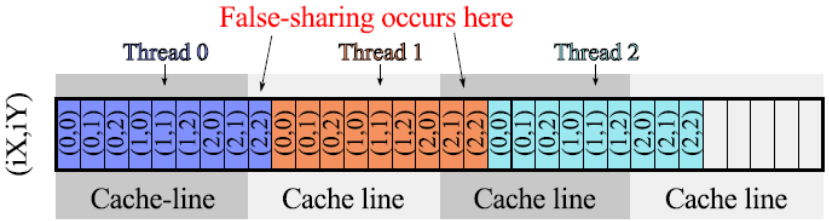
Case #1:
global container
threads-first layout



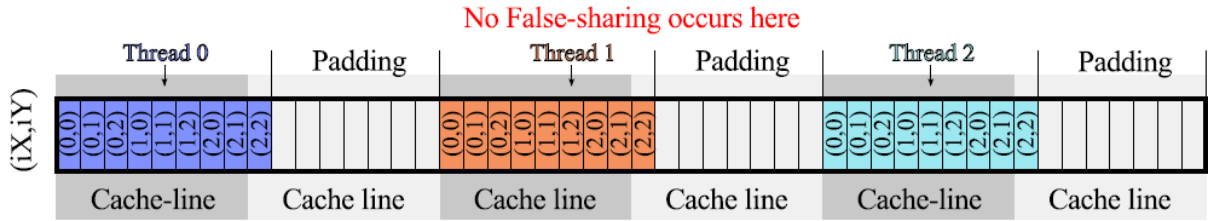
Bins

(0,0)	(1,0)	(2,0)
(0,1)	(1,1)	(2,1)
(0,2)	(1,2)	(2,2)

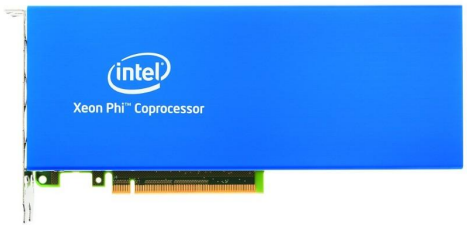
Case #2:
global container
threads-last layout

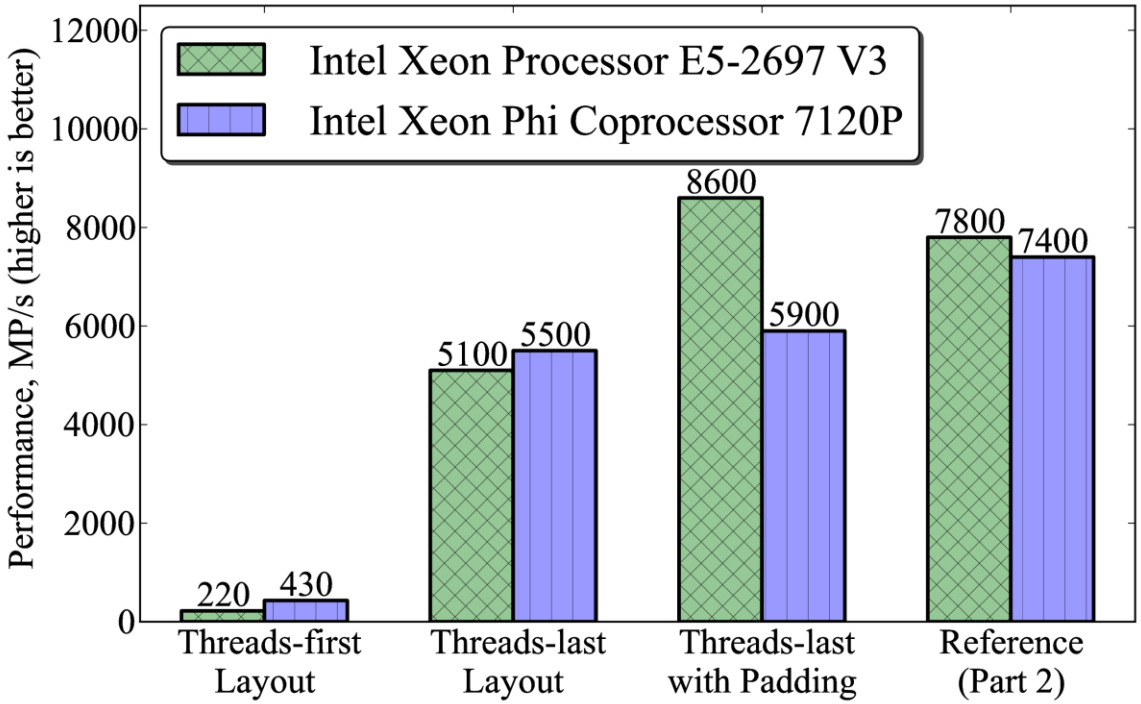


Case #3:
global container
threads-last layout
with padding



FALSE SHARING AND PERFORMANCE





WHAT NEXT?

- Download this tutorial ⟶
- Visit us at **colfaxresearch.com** for more educational materials
- For example, learn how to optimize memory traffic in a compute-bound application with loop tiling: **colfaxresearch.com/?p=13** (blast from the past)