

OpenACC Course

Office Hour 3: Expressing Data Locality and Optimizations with OpenACC

Nikolai Sakharnykh, NVIDIA Developer Technologies



Course Syllabus

Oct 1: Introduction to OpenACC

Oct 6: Office Hours

Oct 15: Profiling and Parallelizing with the OpenACC Toolkit

Oct 20: Office Hours

Oct 29: Expressing Data Locality and Optimizations with OpenACC

Nov 3: Office Hours

Nov 12: Advanced OpenACC Techniques

Nov 24: Office Hours

Agenda

Review: Lab 3 Solution & Results

Q&A

Homework & Next Lecture

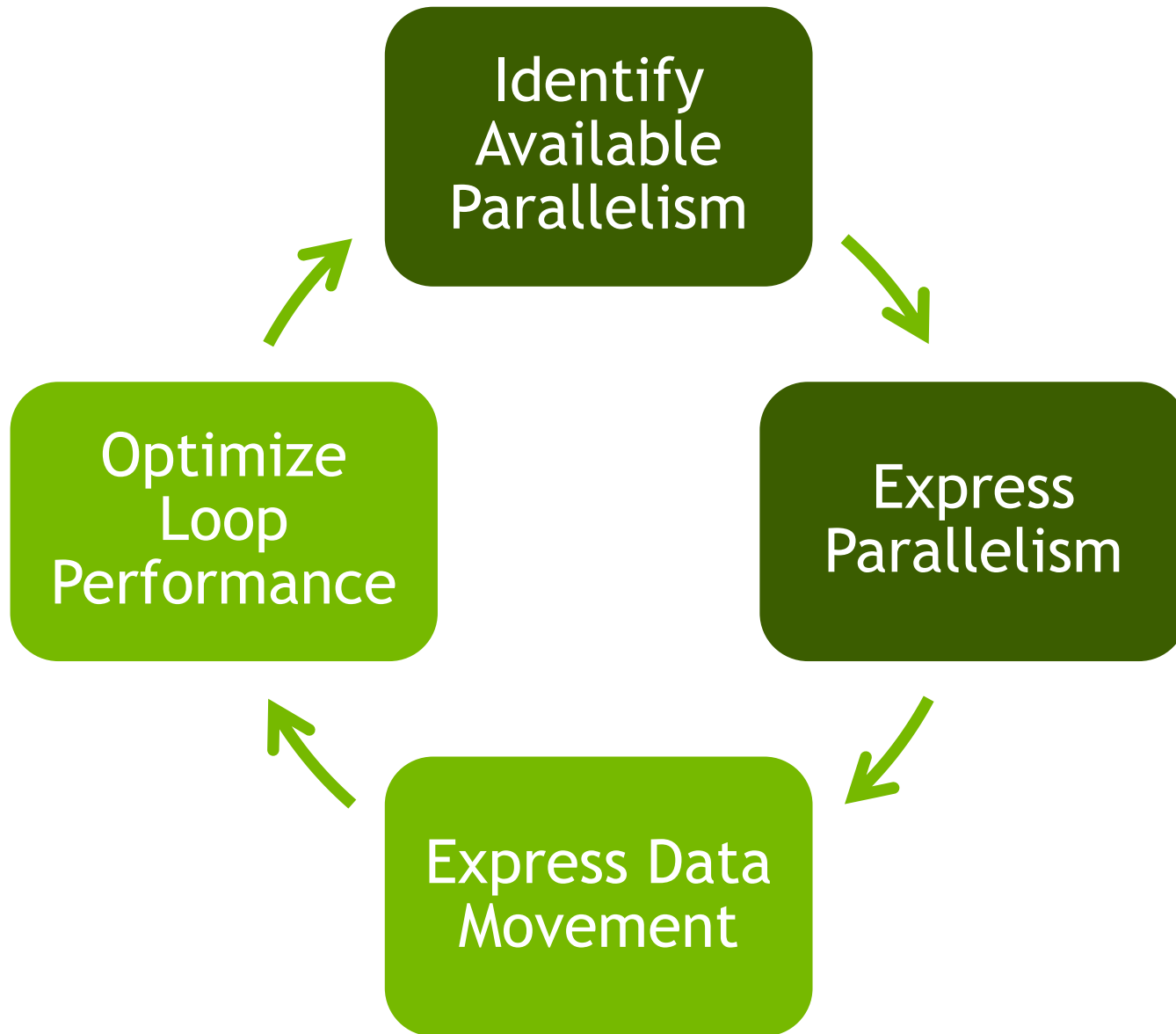
Answered Questions and Recordings

<https://developer.nvidia.com/openacc-course>

Lab 3:

<http://bit.ly/nvoacclab3>

Lab 3 Results



Explicit Data Movement: Copy In Matrix

Add data directives to matrix.h

```
void allocate_3d_poisson_matrix(matrix &A, int N) {
    int num_rows=(N+1)*(N+1)*(N+1);
    int nnz=27*num_rows;
    A.num_rows=num_rows;
    A.row_offsets = (unsigned int*) \
        malloc((num_rows+1)*sizeof(unsigned int));
    A.cols = (unsigned int*)malloc(nnz*sizeof(unsigned int));
    A.coefs = (double*)malloc(nnz*sizeof(double));

    // Initialize Matrix

    A.row_offsets[num_rows]=nnz;
    A.nnz=nnz;
    #pragma acc enter data copyin(A)
    #pragma acc enter data \
        copyin(A.row_offsets[:num_rows+1],A.cols[:nnz],A.coefs[:nnz])
}
```

Explicit Data Movement: Vector

Add data directives to vector.h

```
void allocate_vector(vector &v, unsigned int n) {
    v.n=n;
    v.coefs=(double*)malloc(n*sizeof(double));
#pragma acc enter data copyin(v)
#pragma acc enter data create(v.coefs[:n])
}
void free_vector(vector &v) {
    double *vcoefs=v.coefs;
#pragma acc exit data delete(v.coefs)
#pragma acc exit data delete(v)
    free(v.coefs);
}
void initialize_vector(vector &v,double val) {
    for(int i=0;i<v.n;i++)
        v.coefs[i]=val;
#pragma acc update device(v.coefs[:v.n])
}
```


Explicit Data Movement: Present Clause

Add present to all kernels/parallel regions

```
#pragma acc kernels
present(row_offsets,cols,Acoefs,xcoefs,ycoefs)
{
    for(int i=0;i<num_rows;i++) {
        double sum=0;
        int row_start=row_offsets[i];
        int row_end=row_offsets[i+1];
        for(int j=row_start;j<row_end;j++) {
            unsigned int Acol=cols[j];
            double Acoef=Acoefs[j];
            double xcoef=xcoefs[Acol];
            sum+=Acoef*xcoef;
        }
        ycoefs[i]=sum;
    }
}
```

```
#pragma acc kernels present(xcoefs,ycoefs)
{
    for(int i=0;i<n;i++) {
        sum+=xcoefs[i]*ycoefs[i];
    }
}

#pragma acc kernels
present(xcoefs,ycoefs,wcoefs)
{
    for(int i=0;i<n;i++) {
        wcoefs[i] =
            alpha*xcoefs[i]+beta*ycoefs[i];
    }
}
```

Optimize: Optimize Vector & Worker Loops

Add loop optimizations to matvec function (kernels)

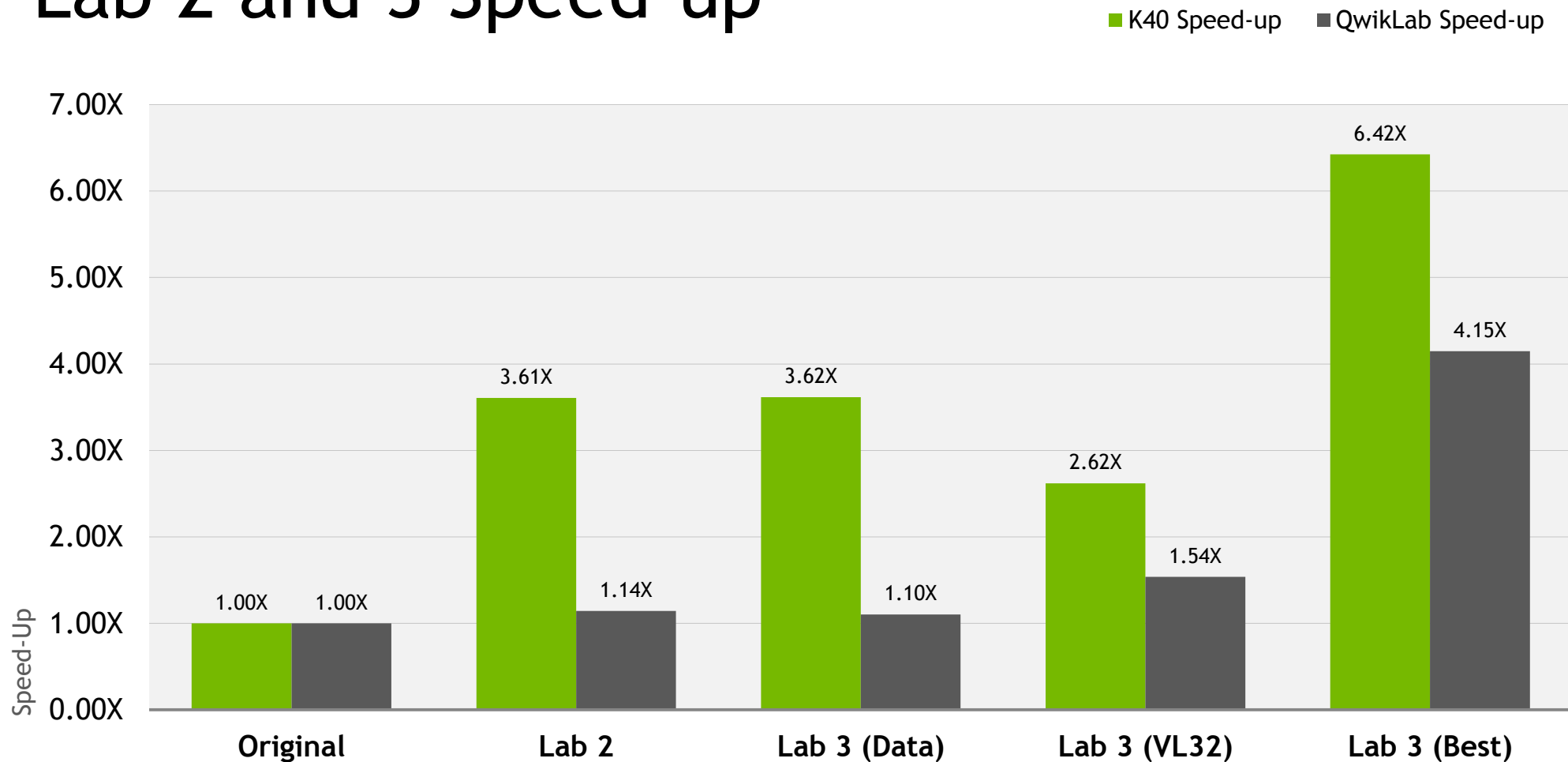
```
#pragma acc kernels present(row_offsets,cols,Acoefs,xcoefs,ycoefs)
{
#pragma acc loop device_type(nvidia) gang worker(4)
    for(int i=0;i<num_rows;i++) {
        double sum=0;
        int row_start=row_offsets[i];
        int row_end=row_offsets[i+1];
#pragma acc loop device_type(nvidia) vector(32)
        for(int j=row_start;j<row_end;j++) {
            unsigned int Acol=cols[j];
            double Acoef=Acoefs[j];
            double xcoef=xcoefs[Acol];
            sum+=Acoef*xcoef;
        }
        ycoefs[i]=sum;
    }
}
```

Optimize: Optimize Vector & Worker Loops

Add loop optimizations to matvec function (parallel loop)

```
#pragma acc parallel loop present(row_offsets,cols,Acoefs,xcoefs,ycoefs) \  
    device_type(nvidia) gang worker vector_length(32) num_workers(4)  
for(int i=0;i<num_rows;i++) {  
    double sum=0;  
    int row_start=row_offsets[i];  
    int row_end=row_offsets[i+1];  
#pragma acc loop reduction(+:sum) device_type(nvidia) vector  
    for(int j=row_start;j<row_end;j++) {  
        unsigned int Acol=cols[j];  
        double Acoef=Acoefs[j];  
        double xcoef=xcoefs[Acol];  
        sum+=Acoef*xcoef;  
    }  
    ycoefs[i]=sum;  
}
```

Lab 2 and 3 Speed-up



OpenACC Multicore Speed-up

OpenACC was not designed for GPUs, it was designed for any parallel architecture.

PGI 15.10 officially releases support for targeting OpenACC directives on multi-core CPUs.

Rebuild the code with: `-ta=multicore`.

- The compiler does not currently support two targets in the same executable.
- CPU-specific loop optimizations can use `device_type(host)`
- Multicore target currently targets 1 gang per CPU core without vectorization.

OpenACC Multicore Speed-up

```
pgc++ -fast -ta=multicore -Minfo=accel main.cpp -o cg
```

```
dot(const vector &, const vector &):
```

```
    6, include "vector_functions.h"
```

```
    14, Loop is parallelizable
```

```
        Generating Multicore code
```

```
    14, #pragma acc loop gang
```

```
waxpby(double, const vector &, double, const vector &, const vector &):
```

```
    6, include "vector_functions.h"
```

```
    29, Loop is parallelizable
```

```
        Generating Multicore code
```

```
    29, #pragma acc loop gang
```

```
matvec(const matrix &, const vector &, const vector &):
```

```
    8, include "matrix_functions.h"
```

```
    17, Loop is parallelizable
```

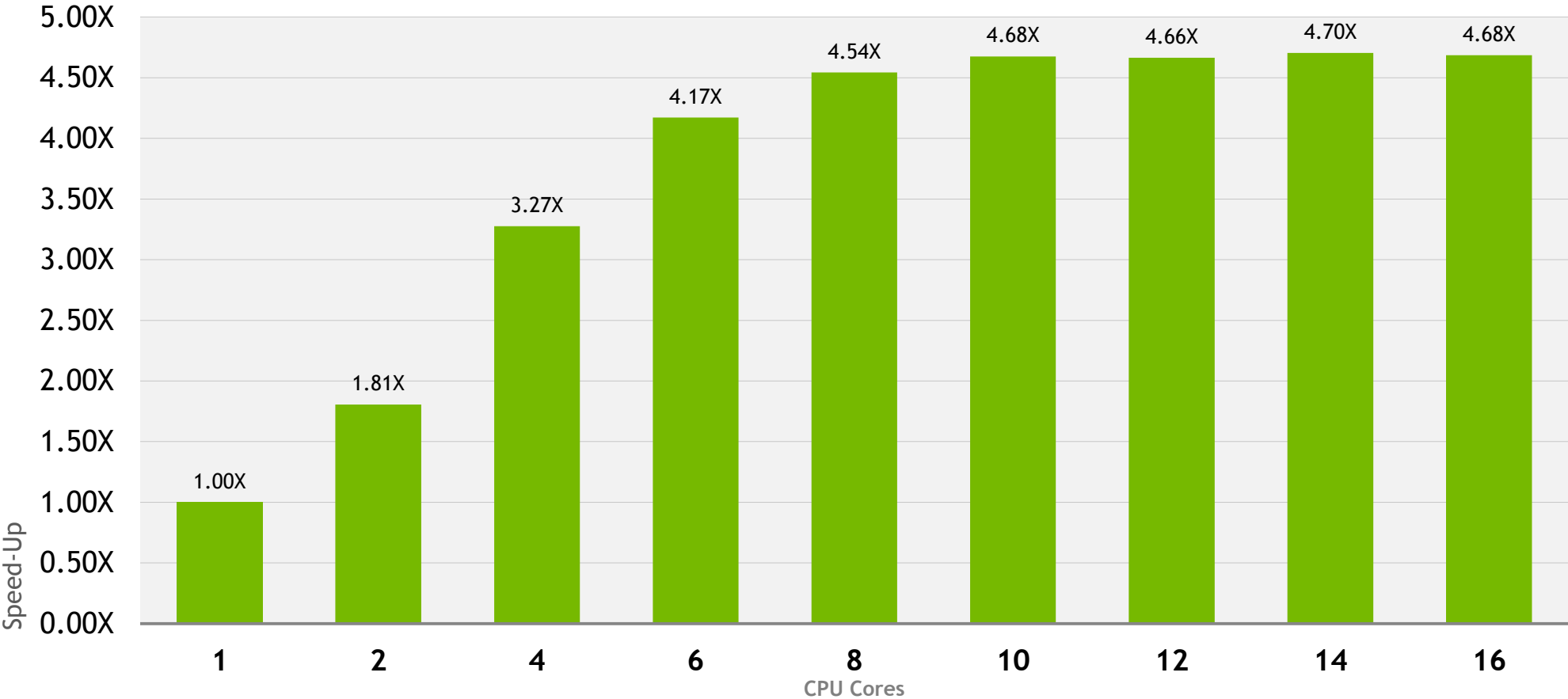
```
        Generating Multicore code
```

```
    17, #pragma acc loop gang
```

```
    22, Loop is parallelizable
```

Compiler generates 1 gang per CPU core and ignores `device_type(nvidia)` loop optimizations.

OpenACC Multi-core Speed-up



PGI 15.9 Compiler (-fast optimization), Intel Xeon CPU E5-2698 v3 @ 2.30GHz (Haswell)

Q&A

Questions from the last class

Q1: What does "unstructured" mean in the context of unstructured data movement?

Q2: Unstructured data directives, is it coming from OpenMP, or a new concept?

Q3: When we copy in the structure which contains embedded pointers that are pointing to host memory, OpenACC will automatically fix up those pointers to refer to the device copies?

Q4: A gang would be similar to a block in CUDA, a worker to a thread?

Q5: In which file can I see the code generated for the GPU after the code is compiled? Is necessary to ask for it using a compiler flag?

Next Steps & Homework

Homework Reminder

This week's homework will build upon the previous lab by applying the two steps discussed today: Express Data Movement and Optimize Loops.

Go to <http://bit.ly/nvoacclab3> from your web browser to take the free lab, or download the code from <https://github.com/NVIDIA-OpenACC-Course/nvidia-openacc-course-sources/> to do the lab on your own machine.

If you have not already completed the previous labs, it's highly recommended that you do <http://bit.ly/nvoacclab2> first.

Course Syllabus

Oct 1: Introduction to OpenACC

Oct 6: Office Hours

Oct 15: Profiling and Parallelizing with the OpenACC Toolkit

Oct 20: Office Hours

Oct 29: Expressing Data Locality and Optimizations with OpenACC

Nov 3: Office Hours

Nov 12: Advanced OpenACC Techniques

Nov 24: Office Hours

OpenACC Kernels Matvec

```
matvec(const matrix &, const vector &, const vector &):  
    8, include "matrix_functions.h"  
        15, Generating copyout(ycoefs[:num_rows])  
            Generating  
copyin(xcoefs[:],Acoefs[:],cols[:],row_offsets[:num_rows+1])  
        16, Loop is parallelizable  
            Accelerator kernel generated  
            Generating Tesla code  
                16, #pragma acc loop gang, vector(128) /* blockIdx.x  
threadIdx.x */  
                20, Loop is parallelizable
```

OpenACC and OpenMP

Number of Cores	OpenACC (s)	OpenMP(s)
1	38.689294	36.354030
2	21.616645	20.052727
4	11.856071	11.085337
6	9.370762	8.963102
8	8.594175	8.357504
10	8.322460	8.229950
12	8.253005	8.262911
14	8.249902	8.222371
16	8.253441	8.219130

- ▶ OpenACC column uses the same, *acc kernels* code as the GPU and multicore target.
- ▶ OpenMP column uses *omp parallel for* on the outer loop.
- ▶ Same CPU used for both

Multicore & GPU Speed-Up

