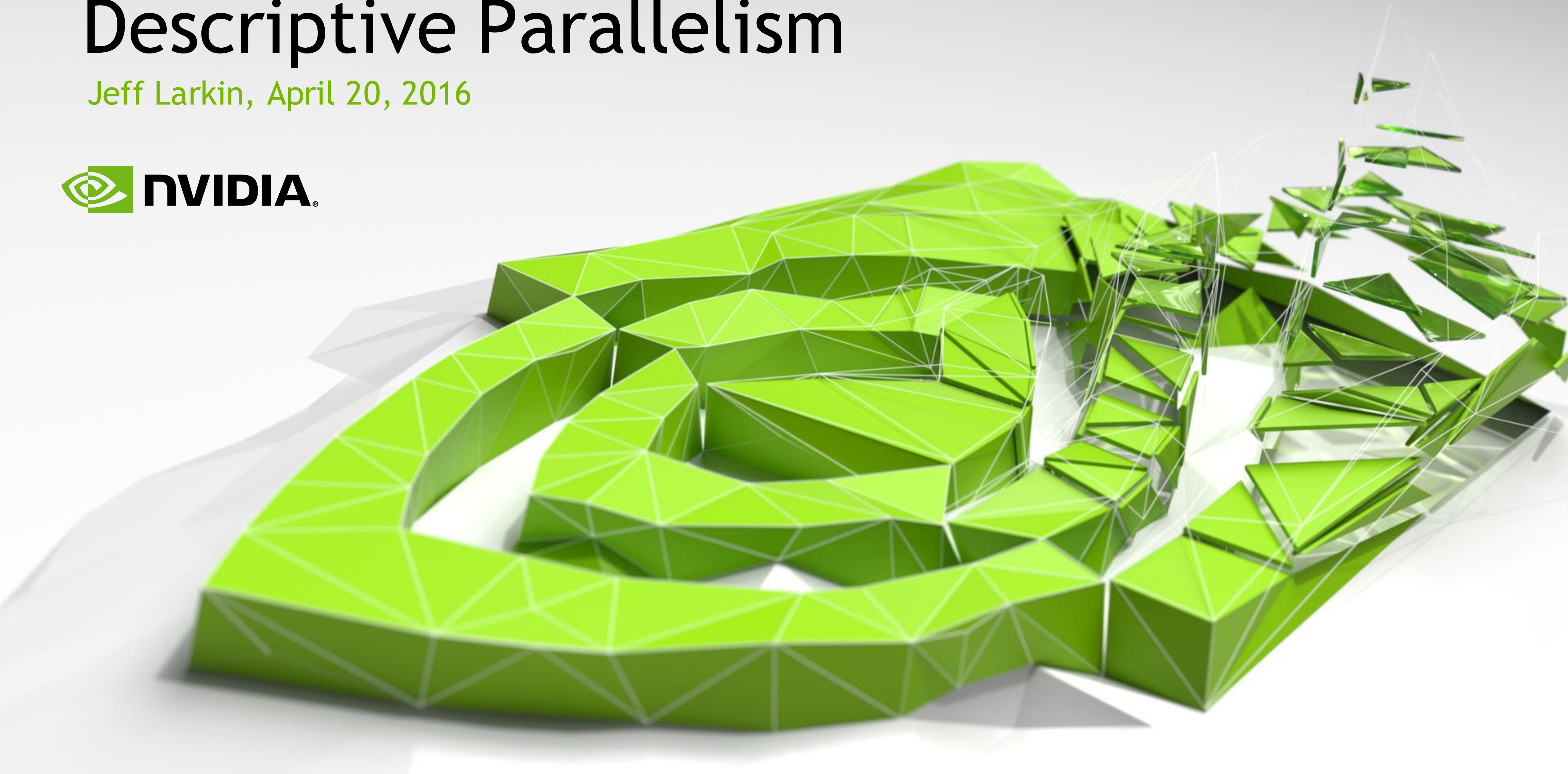
Performance Portability Through Descriptive Parallelism

Jeff Larkin, April 20, 2016



My definition of performance portability

The *same source* code will run *productively* on a variety of *different architectures*.

Two Types of Portability

FUNCTIONAL PORTABILITY

The ability for a single code to run anywhere.

PERFORMANCE PORTABILITY

The ability for a single code to run well anywhere.

Performance Portability is Not

Best == Best

Optimal Everywhere

Optimal Anywhere?

I Assert

Descriptive Parallelism enables performance portable code.

Descriptive or Prescriptive Parallelism

Prescriptive

Programmer explicitly parallelizes the code, compiler obeys

Requires little/no analysis by the compiler

Substantially different architectures require different directives

Fairly consistent behavior between implementations

Descriptive

Compiler parallelizes the code with guidance from the programmer

Compiler must make decisions from available information

Compiler uses information from the programmer and heuristics about the architecture to make decisions

Quality of implementation greatly affects results.

Prescribing vs. Describing

```
while ( error > tol && iter < iter_max )
{
    error = 0.0;

#pragma omp parallel for reduction(max:error)
    for( int j = 1; j < n-1; j++) {
#pragma omp simd
        for( int i = 1; i < m-1; i++ ) {
            Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
                                + A[j-1][i] + A[j+1][i]);
            error = fmax( error, fabs(Anew[j][i] - A[j][i]));
        }
    }

#pragma omp parallel for
    for( int j = 1; j < n-1; j++) {
#pragma omp simd
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
        }
    }

    if(iter++ % 100 == 0) printf("%5d, %0.6f\n", iter, error);
}
```

```
while ( error > tol && iter < iter_max )
{
    error = 0.0;

#pragma acc parallel loop reduction(max:error)
    for( int j = 1; j < n-1; j++) {
#pragma acc loop reduction(max:error)
        for( int i = 1; i < m-1; i++ ) {
            Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
                                + A[j-1][i] + A[j+1][i]);
            error = fmax( error, fabs(Anew[j][i] - A[j][i]));
        }
    }

#pragma acc parallel loop
    for( int j = 1; j < n-1; j++) {
#pragma acc loop
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
        }
    }

    if(iter++ % 100 == 0) printf("%5d, %0.6f\n", iter, error);
}
```

Prescribing vs. Describing

```
while ( error > tol && iter < iter_max )
```

```
while ( error > tol && iter < iter_max )
{
    error = 0.0;
#pragma omp target
{
#pragma omp parallel for reduction(max:error)
    for( int j = 1; j < n-1; j++) {
        for( int i = 1; i < m-1; i++ ) {
            Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
                                + A[j-1][i] + A[j+1][i]);
            error = fmax( error, fabs(Anew[j][i] - A[j][i]));
        }
    }

#pragma omp parallel for
    for( int j = 1; j < n-1; j++) {
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
        }
    }
}
    if(iter++ % 100 == 0) printf("%5d, %0.6f\n", iter, error);
}
```

```
while ( error > tol && iter < iter_max )
{
    error = 0.0;

#pragma acc parallel loop reduction(max:error)
    for( int j = 1; j < n-1; j++) {
#pragma acc loop reduction(max:error)
        for( int i = 1; i < m-1; i++ ) {
            Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
                                + A[j-1][i] + A[j+1][i]);
            error = fmax( error, fabs(Anew[j][i] - A[j][i]));
        }
    }

#pragma acc parallel loop
    for( int j = 1; j < n-1; j++) {
#pragma acc loop
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
        }
    }

    if(iter++ % 100 == 0) printf("%5d, %0.6f\n", iter, error);
}
```

Prescribing vs. Describing

```
while ( error > tol && iter < iter_max )
```

```
while ( error > tol && iter < iter_max )
```

```
#pragma omp target data map(alloc:Anew) map(A)
    while ( error > tol && iter < iter_max )
    {
        error = 0.0;

#pragma omp target teams distribute parallel for
reduction(max:error)
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
                                    + A[j-1][i] + A[j+1][i]);
                error = fmax( error, fabs(Anew[j][i] - A[j][i]));
            }
        }

#pragma omp target teams distribute parallel for
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                A[j][i] = Anew[j][i];
            }
        }

        if(iter++ % 100 == 0) printf("%5d, %0.6f\n", iter, error);
```

```
while ( error > tol && iter < iter_max )
{
    error = 0.0;

#pragma acc parallel loop reduction(max:error)
    for( int j = 1; j < n-1; j++) {
#pragma acc loop reduction(max:error)
        for( int i = 1; i < m-1; i++ ) {
            Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
                                + A[j-1][i] + A[j+1][i]);
            error = fmax( error, fabs(Anew[j][i] - A[j][i]));
        }
    }

#pragma acc parallel loop
    for( int j = 1; j < n-1; j++) {
#pragma acc loop
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
        }
    }

    if(iter++ % 100 == 0) printf("%5d, %0.6f\n", iter, error);
}
```

Prescribing vs. Describing

```
while ( error > tol && iter < iter_max )
```

```
while ( error > tol && iter < iter_max )
```

```
#pragma omp target data map(alloc:Anew) map(A)
```

```
#pragma omp target data map(alloc:Anew) map(A)
```

```
#pragma omp target data map(alloc:Anew) map(A)
    while ( error > tol && iter < iter_max )
    {
        error = 0.0;

#pragma omp target teams distribute
        for( int j = 1; j < n-1; j++)
        {
#pragma omp parallel for reduction(max:error) schedule(static,1)
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
                                    + A[j-1][i] + A[j+1][i]);
                error = fmax( error, fabs(Anew[j][i] - A[j][i]));
            }
        }

#pragma omp target teams distribute
        for( int j = 1; j < n-1; j++)
        {
#pragma omp parallel for schedule(static,1)
            for( int i = 1; i < m-1; i++ )
            {
```

```
while ( error > tol && iter < iter_max )
{
    error = 0.0;

#pragma acc parallel loop reduction(max:error)
    for( int j = 1; j < n-1; j++) {
#pragma acc loop reduction(max:error)
        for( int i = 1; i < m-1; i++ ) {
            Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
                                + A[j-1][i] + A[j+1][i]);
            error = fmax( error, fabs(Anew[j][i] - A[j][i]));
        }
    }

#pragma acc parallel loop
    for( int j = 1; j < n-1; j++) {
#pragma acc loop
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
        }
    }

    if(iter++ % 100 == 0) printf("%5d, %0.6f\n", iter, error);
}
```

Prescribing vs. Describing

```
while ( error > tol && iter < iter_max )
```

```
while ( error > tol && iter < iter_max )
```

```
#pragma omp target data map(alloc:Anew) map(A)
```

```
#pragma omp target data map(alloc:Anew) map(A)
```

```
#pragma omp target data map(alloc:Anew) map(A)
```

```
#pragma omp target data map(alloc:Anew) map(A)
    while ( error > tol && iter < iter_max )
    {
        error = 0.0;

#pragma omp target teams distribute parallel for \
  reduction(max:error) collapse(2)
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
                                    + A[j-1][i] + A[j+1][i]);
                error = fmax( error, fabs(Anew[j][i] - A[j][i]));
            }
        }

#pragma omp target teams distribute parallel for \
  collapse(2)
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
```

```
while ( error > tol && iter < iter_max )
{
    error = 0.0;

#pragma acc parallel loop reduction(max:error)
    for( int j = 1; j < n-1; j++) {
#pragma acc loop reduction(max:error)
        for( int i = 1; i < m-1; i++ ) {
            Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
                                + A[j-1][i] + A[j+1][i]);
            error = fmax( error, fabs(Anew[j][i] - A[j][i]));
        }
    }

#pragma acc parallel loop
    for( int j = 1; j < n-1; j++) {
#pragma acc loop
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
        }
    }

    if(iter++ % 100 == 0) printf("%5d, %0.6f\n", iter, error);
}
```

Prescribing vs. Describing

```
while ( error > tol && iter < iter_max )
```

```
while ( error > tol && iter < iter_max )
```

```
#pragma omp target data map(alloc:Anew) map(A)
```

```
#pragma omp target data map(alloc:Anew) map(A)
```

```
#pragma omp target data map(alloc:Anew) map(A)
```

```
#pragma omp target data map(alloc:Anew) map(A)
    while ( error > tol && iter < iter_max )
    {
        error = 0.0;

#pragma omp target teams distribute
        for( int j = 1; j < n-1; j++)
        {
#pragma omp parallel for reduction(max:error) schedule(static,1)
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
                                    + A[j-1][i] + A[j+1][i]);
                error = fmax( error, fabs(Anew[j][i] - A[j][i]));
            }
        }

#pragma omp target teams distribute
        for( int j = 1; j < n-1; j++)
        {
#pragma omp parallel for schedule(static,1)
            for( int i = 1; i < m-1; i++ )
```

```
while ( error > tol && iter < iter_max )
{
    error = 0.0;

#pragma acc parallel loop reduction(max:error)
    for( int j = 1; j < n-1; j++) {
#pragma acc loop reduction(max:error)
        for( int i = 1; i < m-1; i++ ) {
            Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
                                + A[j-1][i] + A[j+1][i]);
            error = fmax( error, fabs(Anew[j][i] - A[j][i]));
        }
    }

#pragma acc parallel loop
    for( int j = 1; j < n-1; j++) {
#pragma acc loop
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
        }
    }

    if(iter++ % 100 == 0) printf("%5d, %0.6f\n", iter, error);
}
```

Prescribing vs. Describing

```
while ( error > tol && iter < iter_max )
```

```
while ( error > tol && iter < iter_max )
```

```
#pragma omp target data map(alloc:Anew) map(A)
```

```
#pragma omp target data map(alloc:Anew) map(A)
```

```
#pragma omp target data map(alloc:Anew) map(A)
```

```
#pragma omp target data map(alloc:Anew) map(A)
```

```
#pragma omp target data map(alloc:Anew) map(A)
    while ( error > tol && iter < iter_max )
    {
        error = 0.0;

#pragma omp target teams distribute parallel for \
  reduction(max:error) collapse(2) schedule(static,1)
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
                                    + A[j-1][i] + A[j+1][i]);
                error = fmax( error, fabs(Anew[j][i] - A[j][i]));
            }
        }

#pragma omp target teams distribute parallel for \
  collapse(2) schedule(static,1)
        for( int j = 1; j < n-1; j++)
```

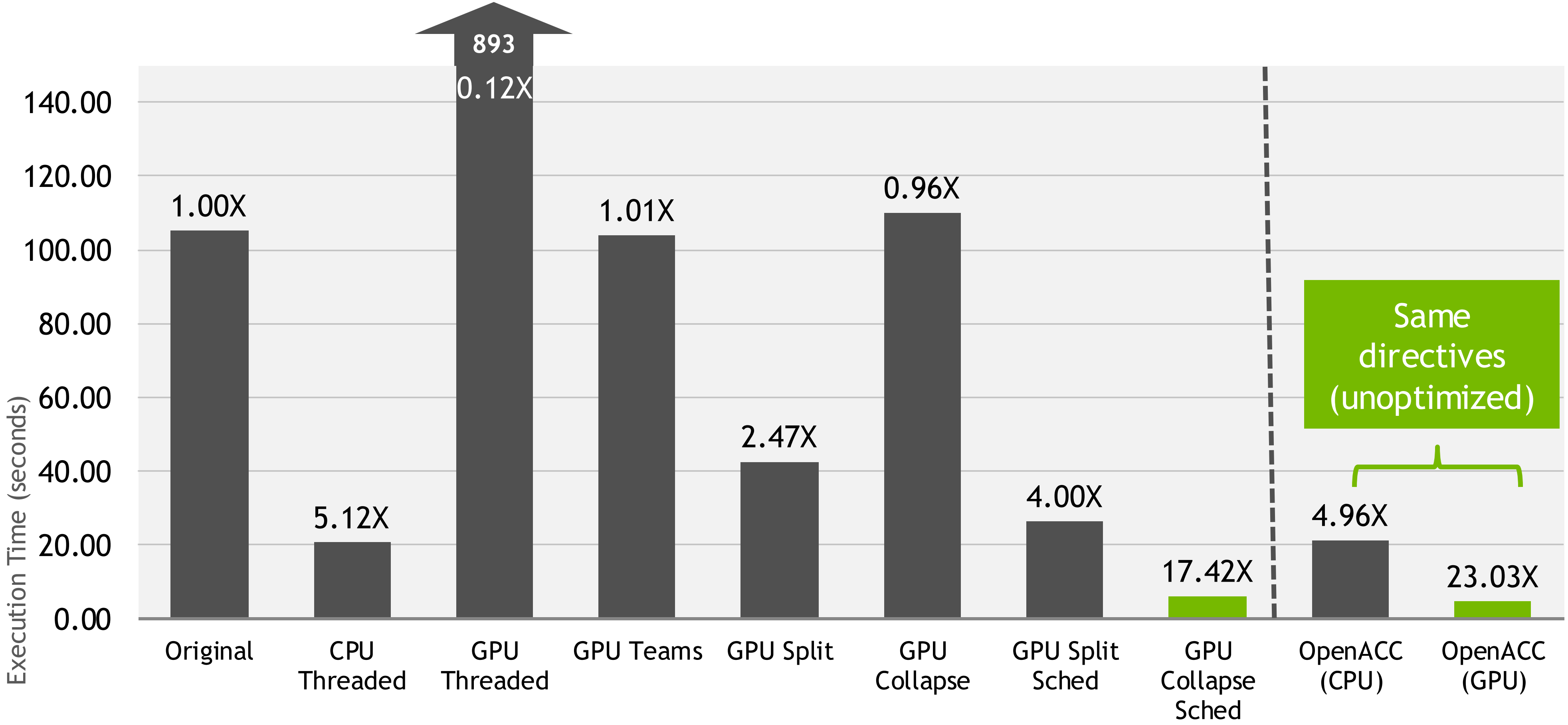
```
while ( error > tol && iter < iter_max )
{
    error = 0.0;

#pragma acc parallel loop reduction(max:error)
    for( int j = 1; j < n-1; j++) {
#pragma acc loop reduction(max:error)
        for( int i = 1; i < m-1; i++ ) {
            Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
                                + A[j-1][i] + A[j+1][i]);
            error = fmax( error, fabs(Anew[j][i] - A[j][i]));
        }
    }

#pragma acc parallel loop
    for( int j = 1; j < n-1; j++) {
#pragma acc loop
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
        }
    }

    if(iter++ % 100 == 0) printf("%5d, %0.6f\n", iter, error);
}
```

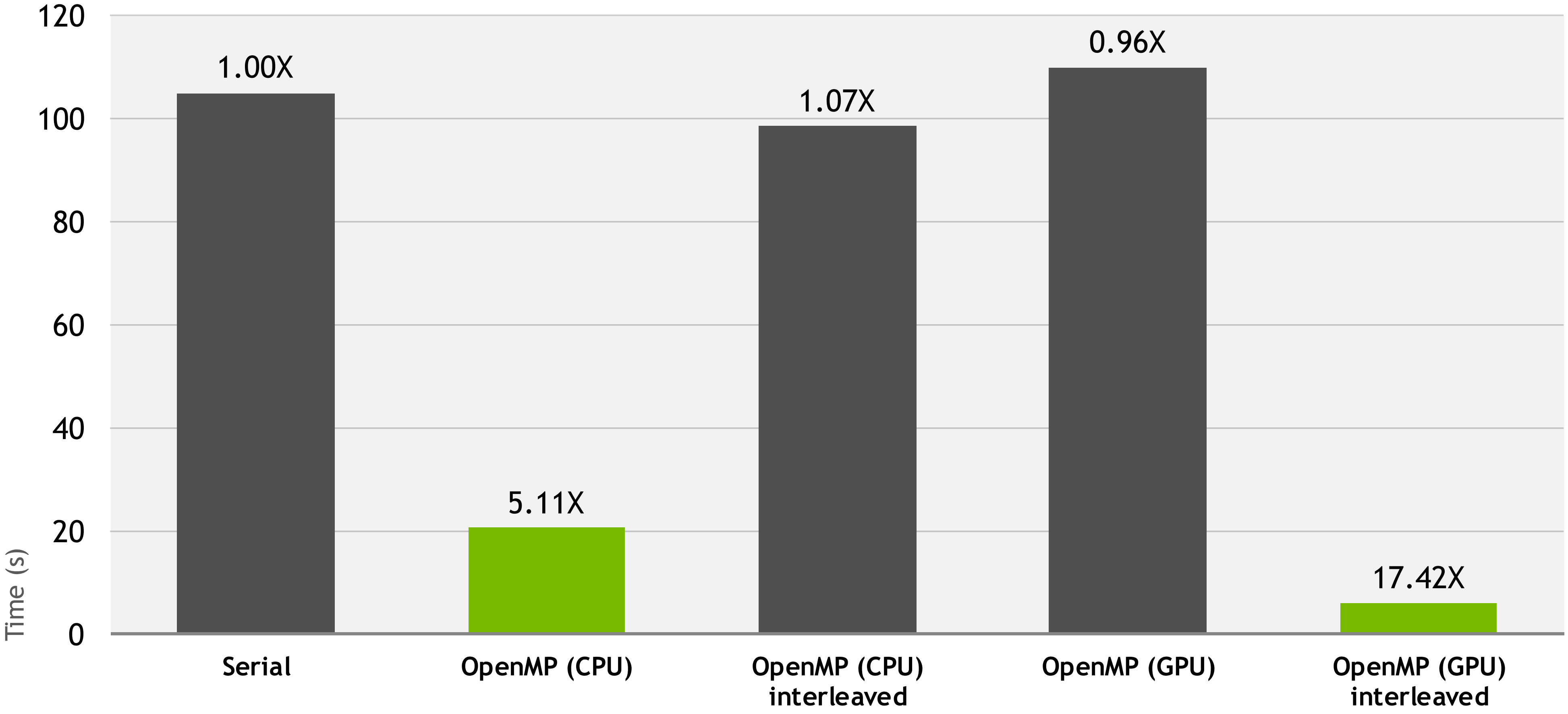
Execution Time (Smaller is Better)



Same directives (unoptimized)

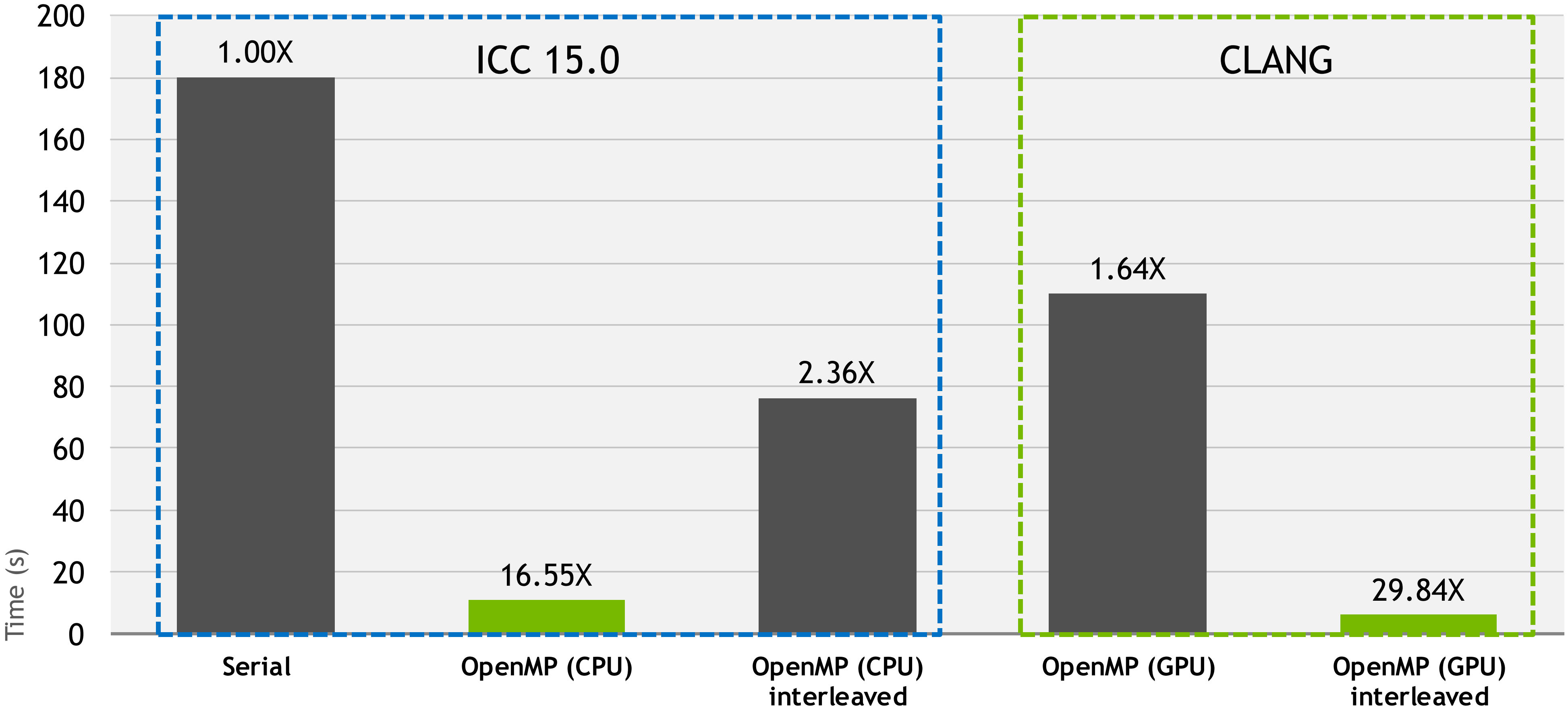
NVIDIA Tesla K40, Intel Xeon E5-2690 v2 @3.00GHz - CLANG pre-3.8, PGI 16.3

Importance of Loop Scheduling



NVIDIA Tesla K40, Intel Xeon E5-2690 v2 @3.00GHz -CLANG pre-3.8, PGI 16.3

Importance of Loop Scheduling



NVIDIA Tesla K40, Intel Xeon E5-2690 v2 @ 3.00GHz - Intel C Compiler 15.0, CLANG pre-3.8, PGI 16.3

Why Can’t OpenMP Do This? (1)

Default schedule is implementation defined

```
#pragma omp target teams distribute parallel for \
  reduction(max:error) ~~collapse(2) schedule(static,1)~~
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
                                    + A[j-1][i] + A[j+1][i]);
                error = fmax( error, fabs(Anew[j][i] - A[j][i]));
            }
        }

#pragma omp target teams distribute parallel for \
  ~~collapse(2) schedule(static,1)~~
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                A[j][i] = Anew[j][i];
            }
        }
```

- The compiler can freely specify the default schedule, but not add the collapse
- Wrong loop for coalescing on the GPU

Why Can’t OpenMP Do This? (2)

Default schedule is implementation defined

```
#pragma omp target teams distribute
        for( int j = 1; j < n-1; j++)
        {
#pragma omp parallel for reduction(max:error) ~~schedule(static,1)~~
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
                                    + A[j-1][i] + A[j+1][i]);
                error = fmax( error, fabs(Anew[j][i] - A[j][i]));
            }
        }

#pragma omp target teams distribute
        for( int j = 1; j < n-1; j++)
        {
#pragma omp parallel for ~~schedule(static,1)~~
            for( int i = 1; i < m-1; i++ )
            {
                A[j][i] = Anew[j][i];
            }
        }
```

- Right loop for coalescing on the GPU
- Makes no sense to distribute to teams on the CPU

Why Can’t OpenMP Do This? (3)

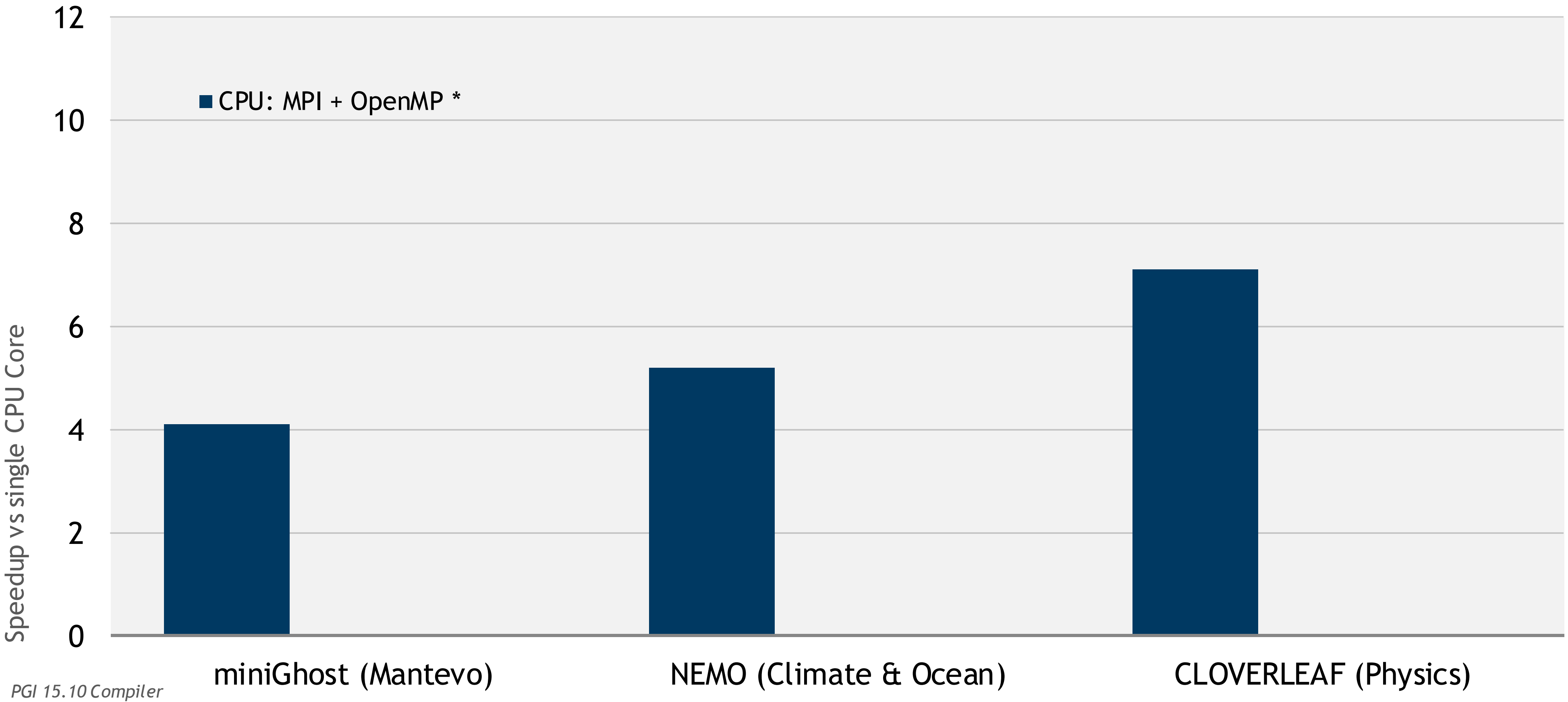
Default schedule is implementation defined

```
#pragma omp target teams distribute parallel for \
  reduction(max:error) collapse(2) simd
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
                                    + A[j-1][i] + A[j+1][i]);
                error = fmax( error, fabs(Anew[j][i] - A[j][i]));
            }
        }

#pragma omp target teams distribute parallel for collapse(2) simd
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                A[j][i] = Anew[j][i];
            }
        }
```

- Compile can always choose 1 team, making this behave as a `parallel for`
- Maybe the compiler can treat SIMD as a coalescing hint?
- Is this the new best practice?

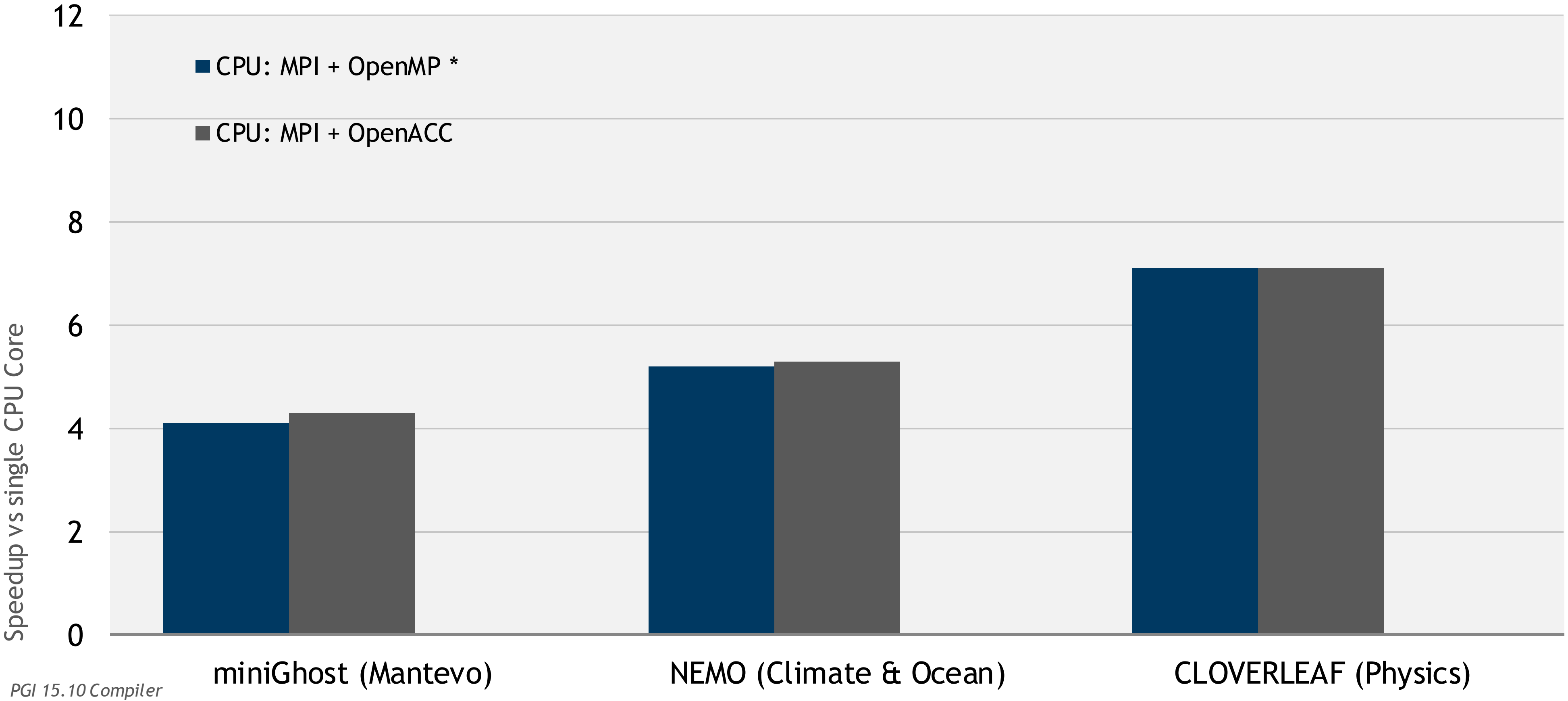
Performance Portability Results



PGI 15.10 Compiler
miniGhost: CPU: Intel Xeon E5-2698 v3, 2 sockets, 32-cores total, GPU: Tesla K80 (single GPU)
NEMO: Each socket CPU: Intel Xeon E5--2698 v3, 16 cores; GPU: NVIDIA K80 both GPUs
CLOVERLEAF: CPU: Dual socket Intel Xeon CPUE5-2690 v2, 20 cores total, GPU: Tesla K80 both GPUs

** NEMO run used all-MPI*

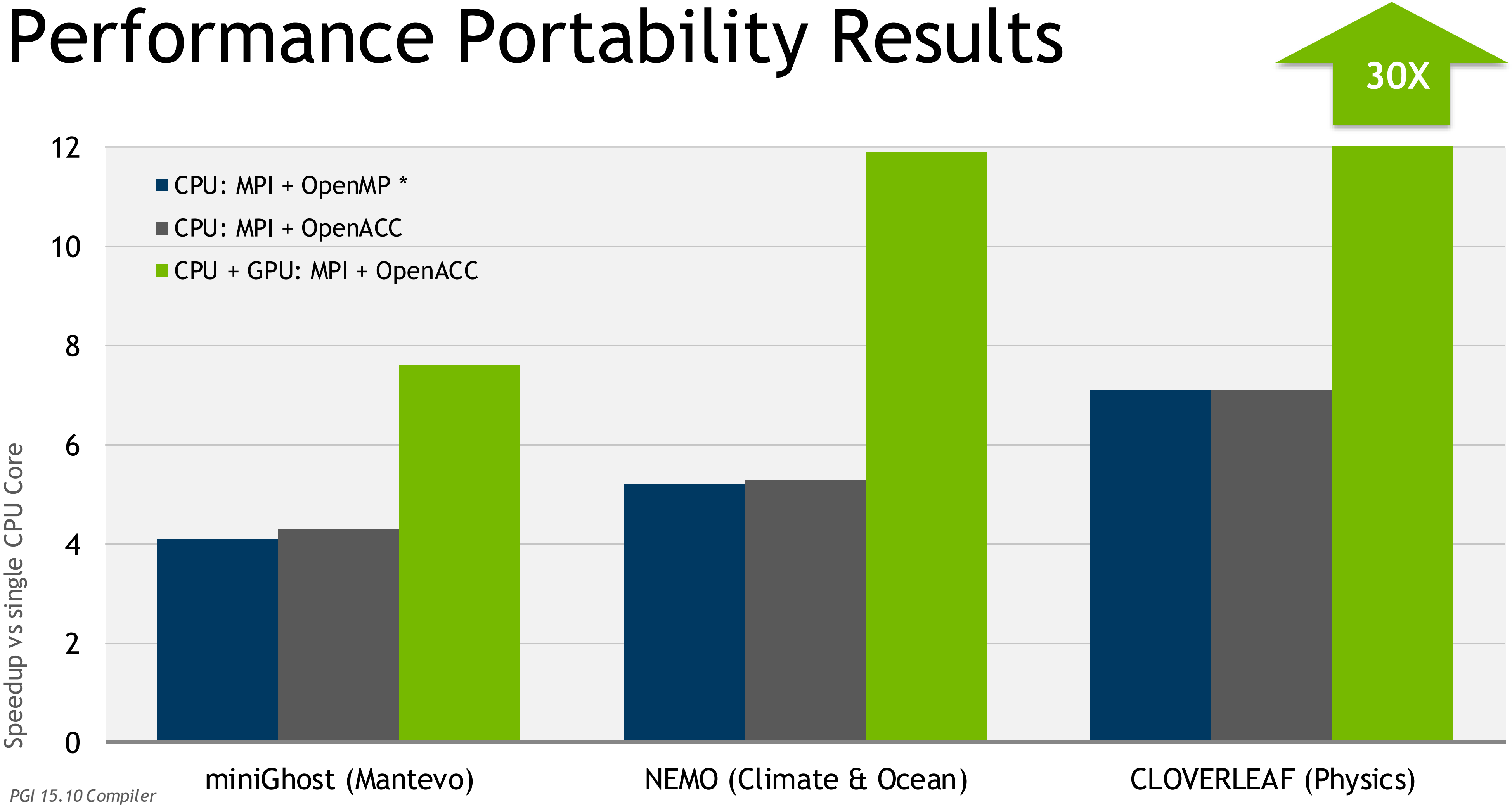
Performance Portability Results



PGI 15.10 Compiler
miniGhost: CPU: Intel Xeon E5-2698 v3, 2 sockets, 32-cores total, GPU: Tesla K80 (single GPU)
NEMO: Each socket CPU: Intel Xeon E5--2698 v3, 16 cores; GPU: NVIDIA K80 both GPUs
CLOVERLEAF: CPU: Dual socket Intel Xeon CPUE5-2690 v2, 20 cores total, GPU: Tesla K80 both GPUs

** NEMO run used all-MPI*

Performance Portability Results



PGI 15.10 Compiler
miniGhost: CPU: Intel Xeon E5-2698 v3, 2 sockets, 32-cores total, GPU: Tesla K80 (single GPU)
NEMO: Each socket CPU: Intel Xeon E5--2698 v3, 16 cores; GPU: NVIDIA K80 both GPUs
CLOVERLEAF: CPU: Dual socket Intel Xeon CPUE5-2690 v2, 20 cores total, GPU: Tesla K80 both GPUs

** NEMO run used all-MPI*

Challenge to the Community

OpenMP should additionally adopt a descriptive programming style

It is possible to be performance portable by being descriptive first and prescriptive as necessary

The community must adopt new best practices (parallel for isn’t enough any more)