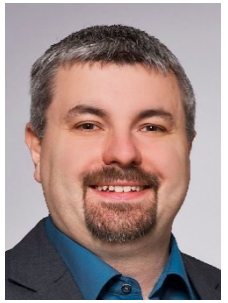
# Programming OpenMP

**Christian Terboven**

**Michael Klemm**

RWTH AACHEN UNIVERSITY

OpenMP



# Agenda (in total 7 Sessions)

- Session 1: OpenMP Introduction
- Session 2: Tasking
- Session 3: Optimization for NUMA and SIMD
- Session 4: What Could Possibly Go Wrong Using OpenMP
- Session 5: Introduction to Offloading with OpenMP
- Session 6: Advanced Offloading Topics
- **Session 7: Miscellaneous Topics**
  - à Review of Session 6, Q&A
  - à Review of Homework Assignments
  - à SIMD, Part 2
  - à Task Affinity
  - à Real World Applications Case Study: NWChem
  - à Hybrid Programming: MPI + OpenMP

# Programming OpenMP

## *Review*

**Christian Terboven**
Michael Klemm

RWTH AACHEN UNIVERSITY
OpenMP

# *Questions?*

# *Jacobi*

# Example solution: Jacobi basic

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

#pragma omp target teams distribute parallel for reduction(max:err)\
          schedule(nonmonotonic:static,1) map(to:A[0:n*m]) map(from:Anew[0:n*m], err)
        for( j = 1; j < n-1; j++) {
          for( i = 1; i < m-1; i++ ) {
            Anew[j *m+ i] = 0.25 * ( A[j     *m+ (i+1)] + A[j     *m+ (i-1)]
                                   +   A[(j-1) *m+ i]     + A[(j+1) *m+ i]);
            err = fmax(err,fabs(Anew[j*m+i]-A[j*m+i]));
          }
        }
        for( j = 1; j < n-1; j++) {
          for( i = 1; i < m-1; i++ ) {
            A[j *m+ i] = Anew[j *m+ i];
          }
        }
        …
        iter++;
    } // end while
```

# Example solution: Jacobi data

```
#pragma omp target data map(to:A[0:n*m]) map(alloc:Anew[0:n*m])
    while ( err > tol && iter < iter_max ) {
        err = 0.0;

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

#pragma omp target teams distribute parallel for schedule(nonmonotonic:static,1)
        for( j = 1; j < n-1; j++) {
            for( i = 1; i < m-1; i++ ) {
                A[j *m+ i] = Anew[j *m+ i];
            }
        }
        …
        iter++;
    } // end while
```

# Example solution: Jacobi unstructure data

```
int main(int argc, char** argv)
{
    …

    initialize(A, Anew, m, n);

    …

    while ( error > tol && iter < iter_max )
    {
        error = calcNext(A, Anew, m, n);
        swap(A, Anew, m, n);

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

        iter++;

    }

    deallocate(A, Anew, m, n);
    return 0;
}
```

# Example solution: Jacobi unstructure data

```
void initialize(double *restrict A, double *restrict Anew, int m, int n)
{
    for(int i = 0; i < m; i++){
        A[i] = 1.0; Anew[i] = 1.0;
    }

    #pragma omp target enter data map(to:A[:m*n],Anew[:m*n])
}

double calcNext(double *restrict A, double *restrict Anew, int m, int n)
{
    double error = 0.0;
    #pragma omp target teams distribute parallel for simd collapse(2) \
    map(to:A[:m*n]) map(from:Anew[:m*n]) map(tofrom:error) reduction(max:error)
    for( int j = 1; j < n-1; j++)
    {
        for( int i = 1; i < m-1; i++ )
        {
            Anew[OFFSET(j, i, m)] = 0.25 * ( A[OFFSET(j, i+1, m)] + A[OFFSET(j, i-1, m)]
                                           + A[OFFSET(j-1, i, m)] + A[OFFSET(j+1, i, m)]);
            error = fmax( error, fabs(Anew[OFFSET(j, i, m)] - A[OFFSET(j, i , m)]));
        }
    }
    return error;
}
```

# Example solution: Jacobi unstructure data

```
void swap(double *restrict A, double *restrict Anew, int m, int n)
{
    #pragma omp target teams distribute parallel for simd collapse(2) \
    map(to:A[:m*n]) map(from:Anew[:m*n])
    for( int j = 1; j < n-1; j++)
    {
        for( int i = 1; i < m-1; i++ )
        {
            A[OFFSET(j, i, m)] = Anew[OFFSET(j, i, m)];
        }
    }
}

void deallocate(double *restrict A, double *restrict Anew, int m, int n)
{
    #pragma omp target exit data map(from:A[:m*n],Anew[:m*n])
    free(A);
    free(Anew);
}
```

# Programming OpenMP

***SIMD, Part 2***

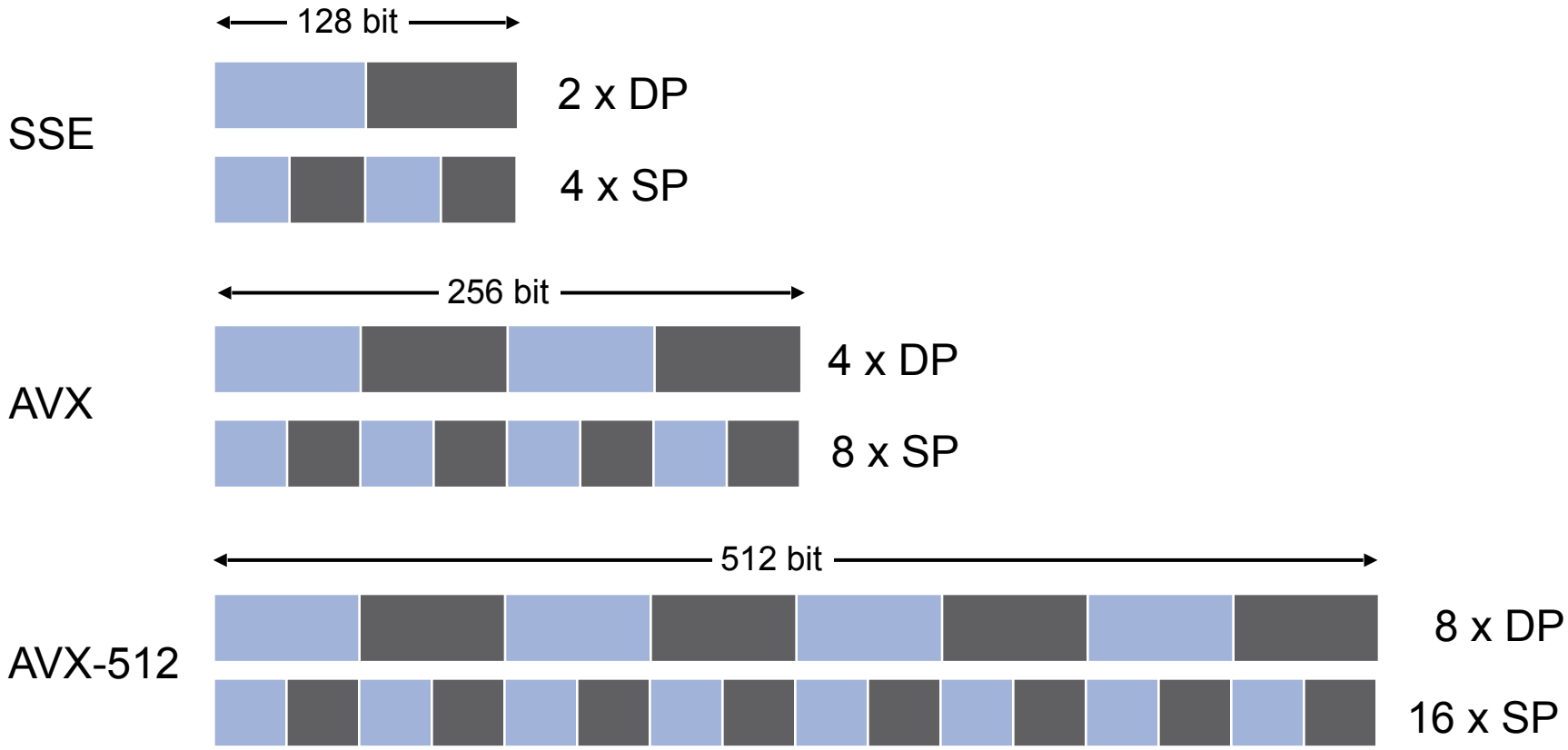
Christian Terboven

**Michael Klemm**

RWTH AACHEN UNIVERSITY

# SIMD on x86 Architectures (Recap)

- Width of SIMD registers has been growing in the past:



# Auto-vectorization (Recap)

- Compilers offer auto-vectorization as an optimization pass
  - à Usually, part of the general loop optimization passes

# Auto-vectorization (Recap)

- Compilers offer auto-vectorization as an optimization pass
  - à Usually, part of the general loop optimization passes
  - à Code analysis detects code properties that inhibit SIMD vectorization

# Auto-vectorization (Recap)

- Compilers offer auto-vectorization as an optimization pass
  - à Usually, part of the general loop optimization passes
  - à Code analysis detects code properties that inhibit SIMD vectorization
  - à Heuristics determine if SIMD execution might be beneficial

# Auto-vectorization (Recap)

- Compilers offer auto-vectorization as an optimization pass
  - à Usually, part of the general loop optimization passes
  - à Code analysis detects code properties that inhibit SIMD vectorization
  - à Heuristics determine if SIMD execution might be beneficial
  - à If all goes well, the compiler will generate SIMD instructions

# Auto-vectorization (Recap)

- Compilers offer auto-vectorization as an optimization pass
  - à Usually, part of the general loop optimization passes
  - à Code analysis detects code properties that inhibit SIMD vectorization
  - à Heuristics determine if SIMD execution might be beneficial
  - à If all goes well, the compiler will generate SIMD instructions

| Example: clang/LLVM | GCC | Intel Compiler |
|---|---|---|
| à -fvectorize | -ftree-vectorize | -vec (enabled w/ -O2) |
| à -Rpass=loop-.* | -ftree-loop-vectorize | -qopt-report=vec |
| à -mprefer-vector-width=<*width*> | -fopt-info-vec-all | |

# Auto-vectorization (Recap)

- Compilers offer auto-vectorization as an optimization pass
  - à Usually, part of the general loop optimization passes
  - à Code analysis detects code properties that inhibit SIMD vectorization
  - à Heuristics determine if SIMD execution might be beneficial
  - à If all goes well, the compiler will generate SIMD instructions

?

- Example: clang/LLVM
  - à -fvectorize
  - à -Rpass=loop-.*
  - à -mprefer-vector-width=<*width*>

GCC
- -ftree-vectorize
- -ftree-loop-vectorize
- -fopt-info-vec-all

Intel Compiler
- -vec (enabled w/ -O2)
- -qopt-report=vec

# In a Time Before OpenMP 4.0

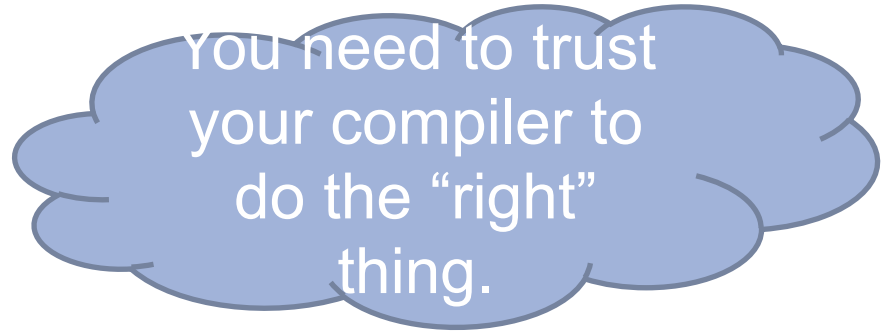
- Support required vendor-specific extensions
  - à Programming models (e.g., Intel® Cilk Plus)
  - à Compiler pragmas (e.g., `#pragma vector`)
  - à Low-level constructs (e.g., `_mm_add_pd()`)

```
   #pragma omp parallel for
#pragma vector always
#pragma ivdep
   for (int i = 0; i < N; i++) {
     a[i] = b[i] + ...;
   }
```

# In a Time Before OpenMP 4.0

- Support required vendor-specific extensions
  - à Programming models (e.g., Intel® Cilk Plus)
  - à Compiler pragmas (e.g., `#pragma vector`)
  - à Low-level constructs (e.g., `_mm_add_pd()`)

```
   #pragma omp parallel for
#pragma vector always
#pragma ivdep
   for (int i = 0; i < N; i++) {
     a[i] = b[i] + ...;
   }
```



# SIMD Loop Construct

- Vectorize a loop nest
  - à Cut loop into chunks that fit a SIMD vector register
  - à No parallelization of the loop body

- Syntax (C/C++)

```
#pragma omp simd [clause[[,] clause],…]
for-loops
```

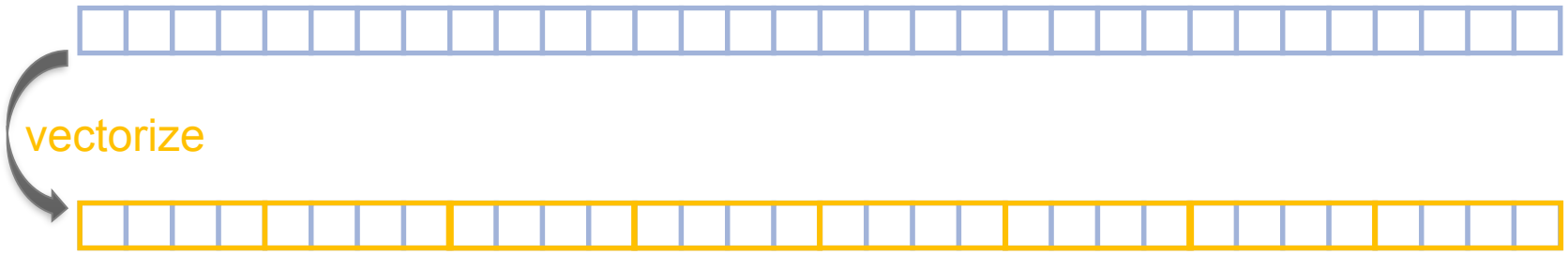
- Syntax (Fortran)

```
!$omp simd [clause[[,] clause],…]
do-loops
[!$omp end simd]
```

# Example

# Example

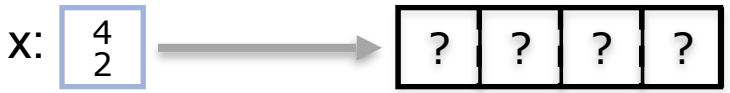
```
float sprod(float *a, float *b, int n) {
  float sum = 0.0f;
#pragma omp simd reduction(+:sum)
  for (int k=0; k<n; k++)
    sum += a[k] * b[k];
  return sum;
}
```



# Data Sharing Clauses

# Data Sharing Clauses

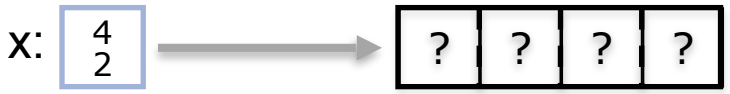
- `private(var-list)`:
  Uninitialized vectors for variables in *var-list*



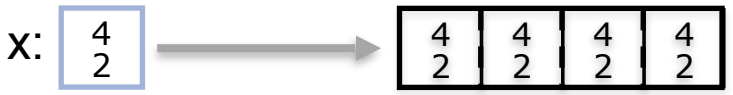

- `firstprivate(var-list)`:
  Initialized vectors for variables in *var-list*

# Data Sharing Clauses

- `private(var-list):`
  Uninitialized vectors for variables in *var-list*




- `firstprivate(var-list):`
  Initialized vectors for variables in *var-list*




- `reduction(op:var-list):`
  Create private variables for *var-list* and apply reduction operator *op* at the end of the construct

# SIMD Loop Clauses

- `safelen (length)`
  - à Maximum number of iterations that can run concurrently without breaking a dependence
  - à In practice, maximum vector length

# SIMD Loop Clauses

- `safelen (length)`
  - à Maximum number of iterations that can run concurrently without breaking a dependence
  - à In practice, maximum vector length
- `linear (list[:linear-step])`
  - à The variable's value is in relationship with the iteration number
    - à $x_i = x_{orig} + i * \text{linear-step}$

# SIMD Loop Clauses

- `safelen (length)`
  - à Maximum number of iterations that can run concurrently without breaking a dependence
  - à In practice, maximum vector length
- `linear (list[:linear-step])`
  - à The variable's value is in relationship with the iteration number
    - à $x_i = x_{orig} + i * \text{linear-step}$
- `aligned (list[:alignment])`
  - à Specifies that the list items have a given alignment
  - à Default is alignment for the architecture

# SIMD Loop Clauses

- `safelen (length)`
  - à Maximum number of iterations that can run concurrently without breaking a dependence
  - à In practice, maximum vector length
- `linear (list[:linear-step])`
  - à The variable's value is in relationship with the iteration number
    - à $x_i = x_{orig} + i * \text{linear-step}$
- `aligned (list[:alignment])`
  - à Specifies that the list items have a given alignment
  - à Default is alignment for the architecture
- `collapse (n)`

# SIMD Worksharing Construct

- Parallelize and vectorize a loop nest
  - à Distribute a loop's iteration space across a thread team
  - à Subdivide loop chunks to fit a SIMD vector register

- Syntax (C/C++)

```
#pragma omp for simd [clause[[,] clause],…]
for-loops
```

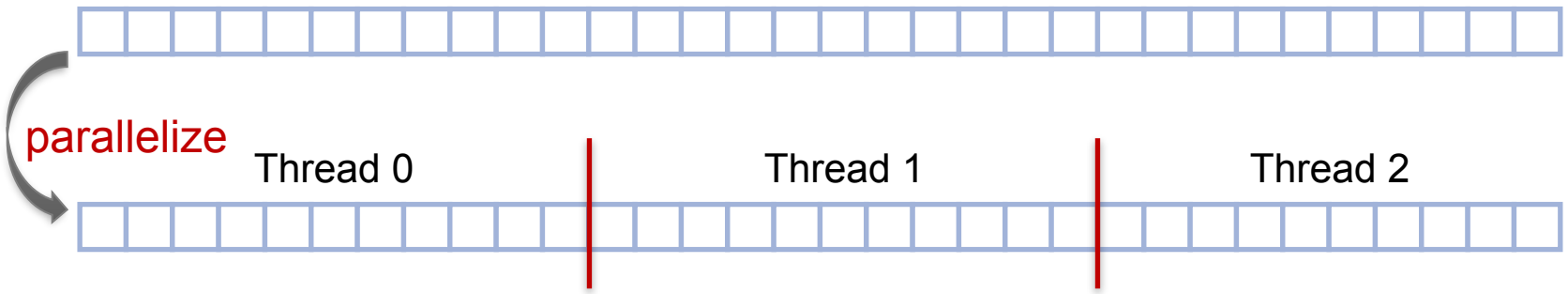
- Syntax (Fortran)

```
!$omp do simd [clause[[,] clause],…]
do-loops
[!$omp end do simd [nowait]]
```

# Example

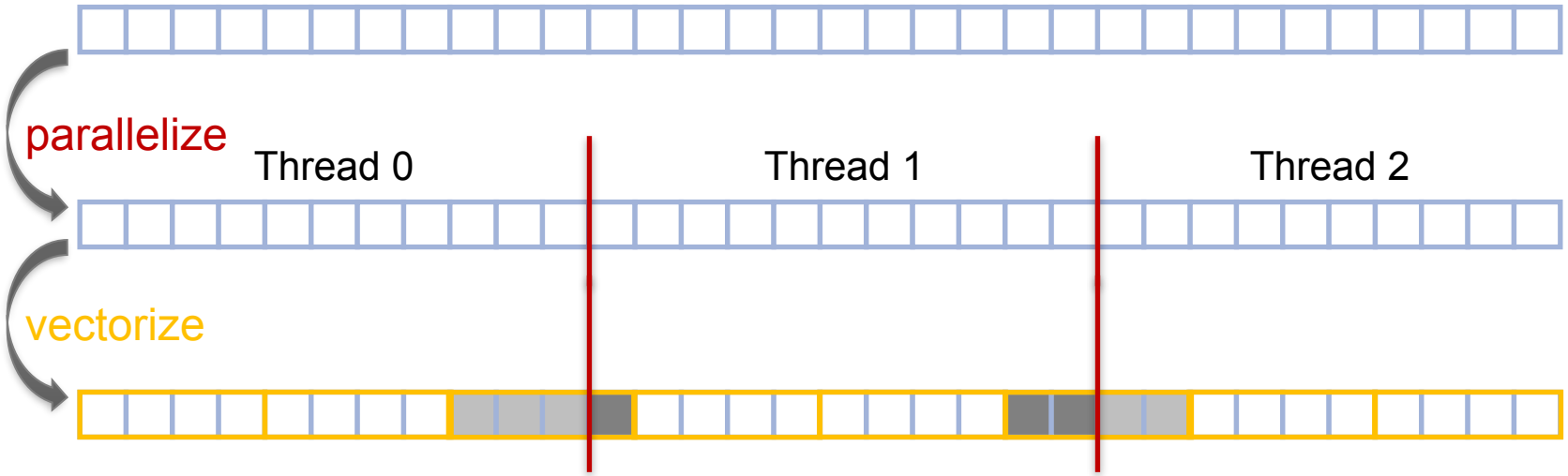
# Example

```
float sprod(float *a, float *b, int n) {
  float sum = 0.0f;
#pragma omp for simd reduction(+:sum)
  for (int k=0; k<n; k++)
    sum += a[k] * b[k];
  return sum;
}
```



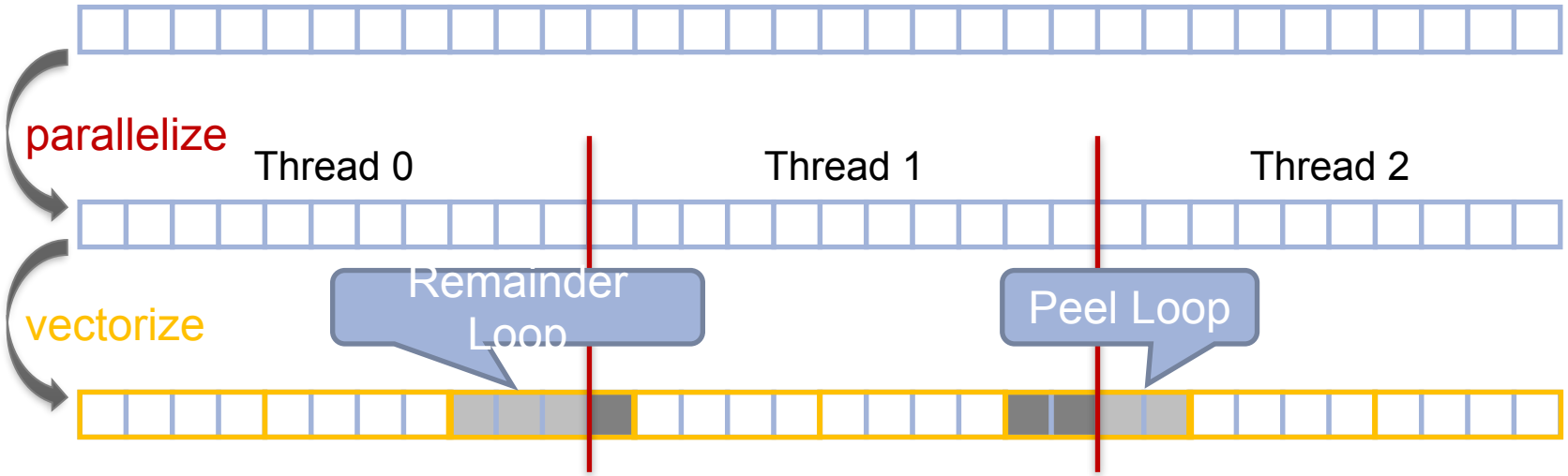
# Example

```
float sprod(float *a, float *b, int n) {
  float sum = 0.0f;
#pragma omp for simd reduction(+:sum)
  for (int k=0; k<n; k++)
    sum += a[k] * b[k];
  return sum;
}
```



# Example

```
float sprod(float *a, float *b, int n) {
  float sum = 0.0f;
#pragma omp for simd reduction(+:sum)
  for (int k=0; k<n; k++)
    sum += a[k] * b[k];
  return sum;
}
```



# Be Careful What You Wish For…

- You should choose chunk sizes that are multiples of the SIMD length
  - à Remainder loops are not triggered
  - à Likely better performance

# Be Careful What You Wish For…

```
float sprod(float *a, float *b, int n) {
  float sum = 0.0f;
#pragma omp for simd reduction(+:sum) \
                     schedule(static, 5)
  for (int k=0; k<n; k++)
    sum += a[k] * b[k];
  return sum;
}
```

- You should choose chunk sizes that are multiples of the SIMD length
  - à Remainder loops are not triggered
  - à Likely better performance

# Be Careful What You Wish For…

```
float sprod(float *a, float *b, int n) {
  float sum = 0.0f;
#pragma omp for simd reduction(+:sum) \
                     schedule(static, 5)
  for (int k=0; k<n; k++)
    sum += a[k] * b[k];
  return sum;
}
```

- You should choose chunk sizes that are multiples of the SIMD length
  - à Remainder loops are not triggered
  - à Likely better performance
- In the above example …
  - à and AVX2, the code will only execute the remainder loop!
  - à and SSE, the code will have one iteration in the SIMD loop plus one in the remainder loop!

# OpenMP 4.5 Simplifies SIMD Chunks

- Chooses chunk sizes that are multiples of the SIMD length
  - à First and last chunk may be slightly different to fix alignment and to handle loops that are not exact multiples of SIMD width
  - à Remainder loops are not triggered
  - à Likely better performance

# OpenMP 4.5 Simplifies SIMD Chunks

```
float sprod(float *a, float *b, int n) {
  float sum = 0.0f;
#pragma omp for simd reduction(+:sum) \
                     schedule(simd: static, 5)
  for (int k=0; k<n; k++)
    sum += a[k] * b[k];
  return sum;
}
```

- Chooses chunk sizes that are multiples of the SIMD length
  - à First and last chunk may be slightly different to fix alignment and to handle loops that are not exact multiples of SIMD width
  - à Remainder loops are not triggered
  - à Likely better performance

# SIMD Function Vectorization

# SIMD Function Vectorization

```
float min(float a, float b) {
    return a < b ? a : b;
}


float distsq(float x, float y) {
    return (x - y) * (x - y);
}

void example() {
#pragma omp parallel for simd
    for (i=0; i<N; i++) {
        d[i] = min(distsq(a[i], b[i]), c[i]);
}   }
```

# SIMD Function Vectorization

- Declare one or more functions to be compiled for calls from a SIMD-parallel loop

- Syntax (C/C++):

```
#pragma omp declare simd [clause[[,] clause],…]
[#pragma omp declare simd [clause[[,] clause],…]]
[…]
function-definition-or-declaration
```

- Syntax (Fortran):

```
!$omp declare simd (proc-name-list)
```

# SIMD Function Vectorization

# SIMD Function Vectorization

```
#pragma omp declare simd
float min(float a, float b) {
    return a < b ? a : b;
}

#pragma omp declare simd
float distsq(float x, float y) {
    return (x - y) * (x - y);
}

void example() {
#pragma omp parallel for simd
    for (i=0; i<N; i++) {
        d[i] = min(distsq(a[i], b[i]), c[i]);
}   }
```

# SIMD Function Vectorization

```
#pragma omp declare simd
float min(float a, float b) {
    return a < b ? a : b;
}

#pragma omp declare simd
float distsq(float x, float y) {
    return (x - y) * (x - y);
}

void example() {
#pragma omp parallel for simd
    for (i=0; i<N; i++) {
        d[i] = min(distsq(a[i], b[i]), c[i]);
}   }
```

```
_ZGVZN16vv_min(%zmm0, %zmm1):
    vminps %zmm1, %zmm0, %zmm0
    ret
```

```
_ZGVZN16vv_distsq(%zmm0, %zmm1):
    vsubps %zmm0, %zmm1, %zmm2
    vmulps %zmm2, %zmm2, %zmm0
    ret
```

```
vmovups (%r14,%r12,4), %zmm0
vmovups (%r13,%r12,4), %zmm1
call _ZGVZN16vv_distsq
vmovups (%rbx,%r12,4), %zmm1
call _ZGVZN16vv_min
```

# SIMD Function Vectorization

- `simdlen (`*`length`*`)`
  - à generate function to support a given vector length

# SIMD Function Vectorization

- `simdlen (length)`
  - à generate function to support a given vector length
- `uniform (argument-list)`
  - à argument has a constant value between the iterations of a given loop

# SIMD Function Vectorization

- `simdlen (length)`
  - à generate function to support a given vector length
- `uniform (argument-list)`
  - à argument has a constant value between the iterations of a given loop
- `inbranch`
  - à function always called from inside an if statement
- `notinbranch`
  - à function never called from inside an if statement

# SIMD Function Vectorization

- `simdlen (length)`
  - à generate function to support a given vector length
- `uniform (argument-list)`
  - à argument has a constant value between the iterations of a given loop
- `inbranch`
  - à function always called from inside an if statement
- `notinbranch`
  - à function never called from inside an if statement
- `linear (argument-list[:linear-step])`
- `aligned (argument-list[:alignment])`

# inbranch & notinbranch

```
#pragma omp declare simd inbranch
float do_stuff(float x) {
    /* do something */
    return x * 2.0;
}

void example() {
#pragma omp simd
    for (int i = 0; i < N; i++)
        if (a[i] < 0.0)
            b[i] = do_stuff(a[i]);
}
```

# inbranch & notinbranch

```
#pragma omp declare simd inbranch
float do_stuff(float x) {
    /* do something */
    return x * 2.0;
}

void example() {
#pragma omp simd
    for (int i = 0; i < N; i++)
        if (a[i] < 0.0)
            b[i] = do_stuff(a[i]);
}
```

```
vec8 do_stuff_v(vec8 x, mask m) {
    /* do something */
    vmulpd x{m}, 2.0, tmp
    return tmp;
}
```

# inbranch & notinbranch

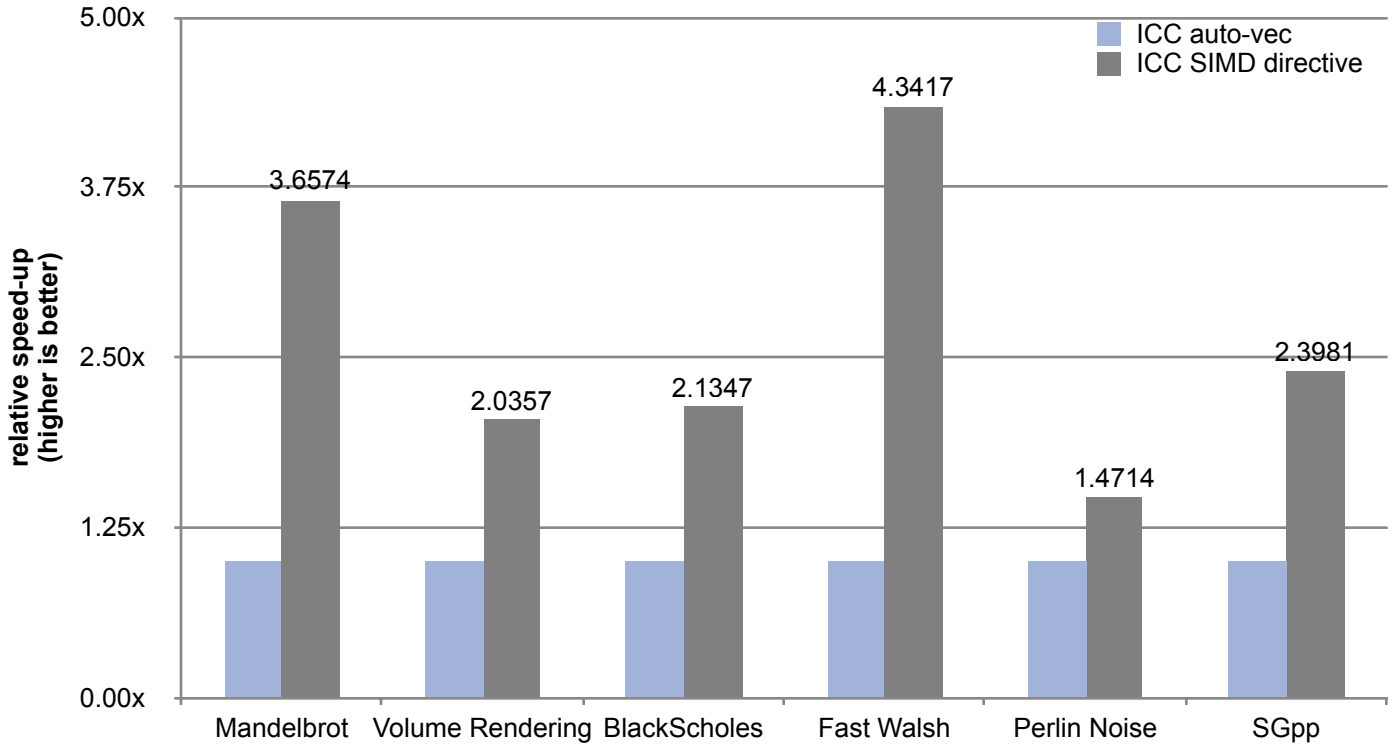
```
#pragma omp declare simd inbranch
float do_stuff(float x) {
    /* do something */
    return x * 2.0;
}

void example() {
#pragma omp simd
    for (int i = 0; i < N; i++)
        if (a[i] < 0.0)
            b[i] = do_stuff(a[i]);
}
```

```
vec8 do_stuff_v(vec8 x, mask m) {
    /* do something */
    vmulpd x{m}, 2.0, tmp
    return tmp;
}
```

```
for (int i = 0; i < N; i+=8) {
    vcmp_lt &a[i], 0.0, mask
    b[i] = do_stuff_v(&a[i], mask);
}
```

# SIMD Constructs & Performance




M.Klemm, A.Duran, X.Tian, H.Saito, D.Caballero, and X.Martorell. Extending OpenMP with Vector Constructs for Modern Multicore SIMD Architectures. In Proc. of the Intl. Workshop on OpenMP, pages 59-72, Rome, Italy, June 2012. LNCS 7312.

# Programming OpenMP

***NUMA***

**Christian Terboven**
Michael Klemm

RWTH AACHEN UNIVERSITY
OpenMP

# Improving Tasking Performance: Task Affinity

# Motivation

- Techniques for process binding & thread pinning available
  - à OpenMP thread level: `OMP_PLACES & OMP_PROC_BIND`
  - à OS functionality: `taskset -c`

OpenMP Tasking:

- In general: Tasks may be executed by any thread in the team
  - à Missing task-to-data affinity may have detrimental effect on performance

OpenMP 5.0:

- `affinity` clause to express affinity to data

# `affinity` clause

- New clause: `#pragma omp task affinity (list)`
  - à Hint to the runtime to execute task closely to physical data location
  - à Clear separation between dependencies and affinity

- Expectations:
  - à Improve data locality / reduce remote memory accesses
  - à Decrease runtime variability

- Still expect task stealing
  - à In particular, if a thread is under-utilized

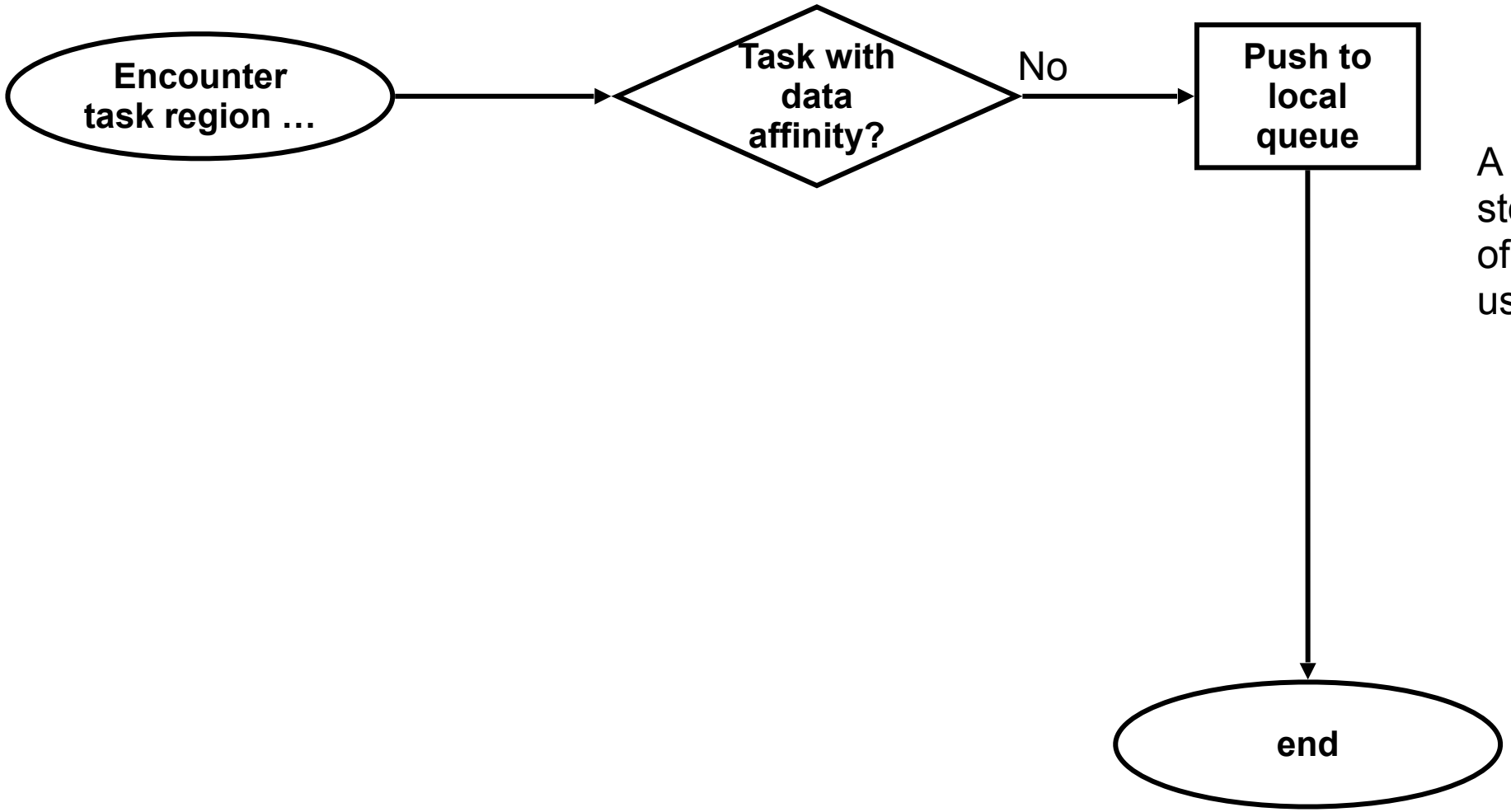
# Code Example

- Excerpt from task-parallel STREAM

```
1  #pragma omp task \
2      shared(a, b, c, scalar) \
3      firstprivate(tmp_idx_start, tmp_idx_end) \
4      affinity( a[tmp_idx_start] )
5  {
6    int i;
7    for(i = tmp_idx_start; i <= tmp_idx_end; i++)
8      a[i] = b[i] + scalar * c[i];
9  }
```

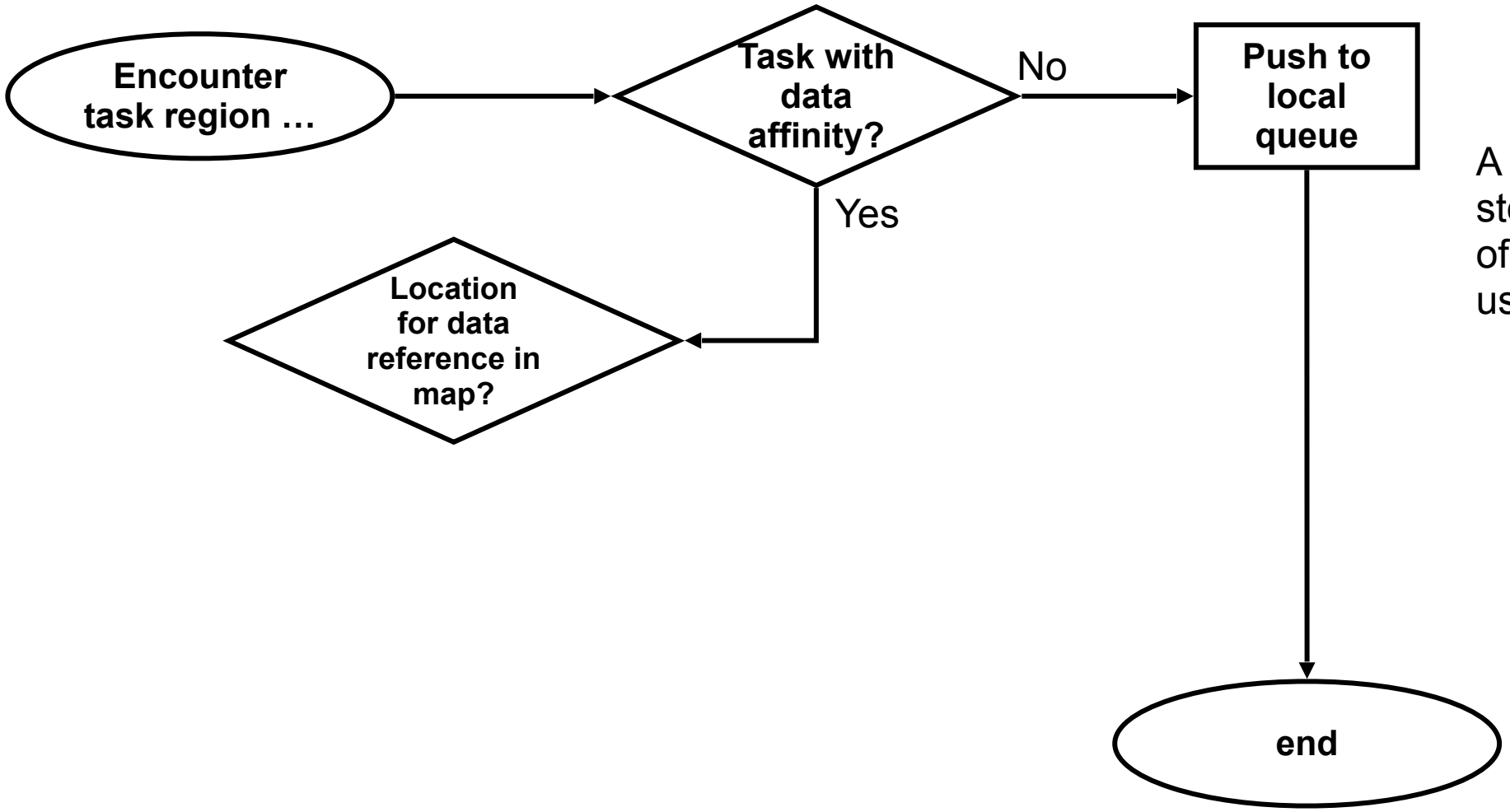
- à Loops have been blocked manually (see `tmp_idx_start/end`)
- à Assumption: initialization and computation have same blocking and same affinity

# Selected LLVM implementation details



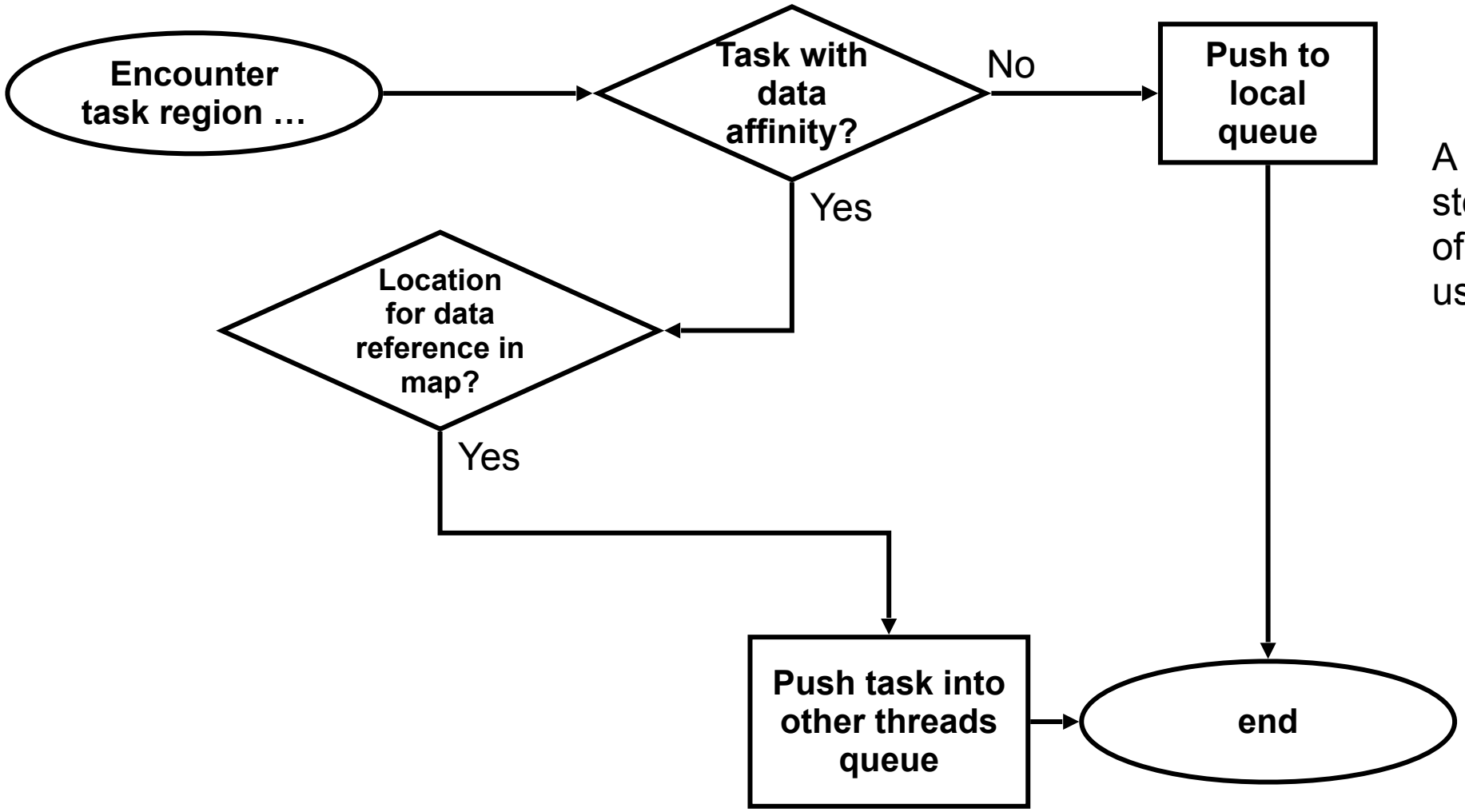

A map is introduced to store location information of data that was previously used

# Selected LLVM implementation details



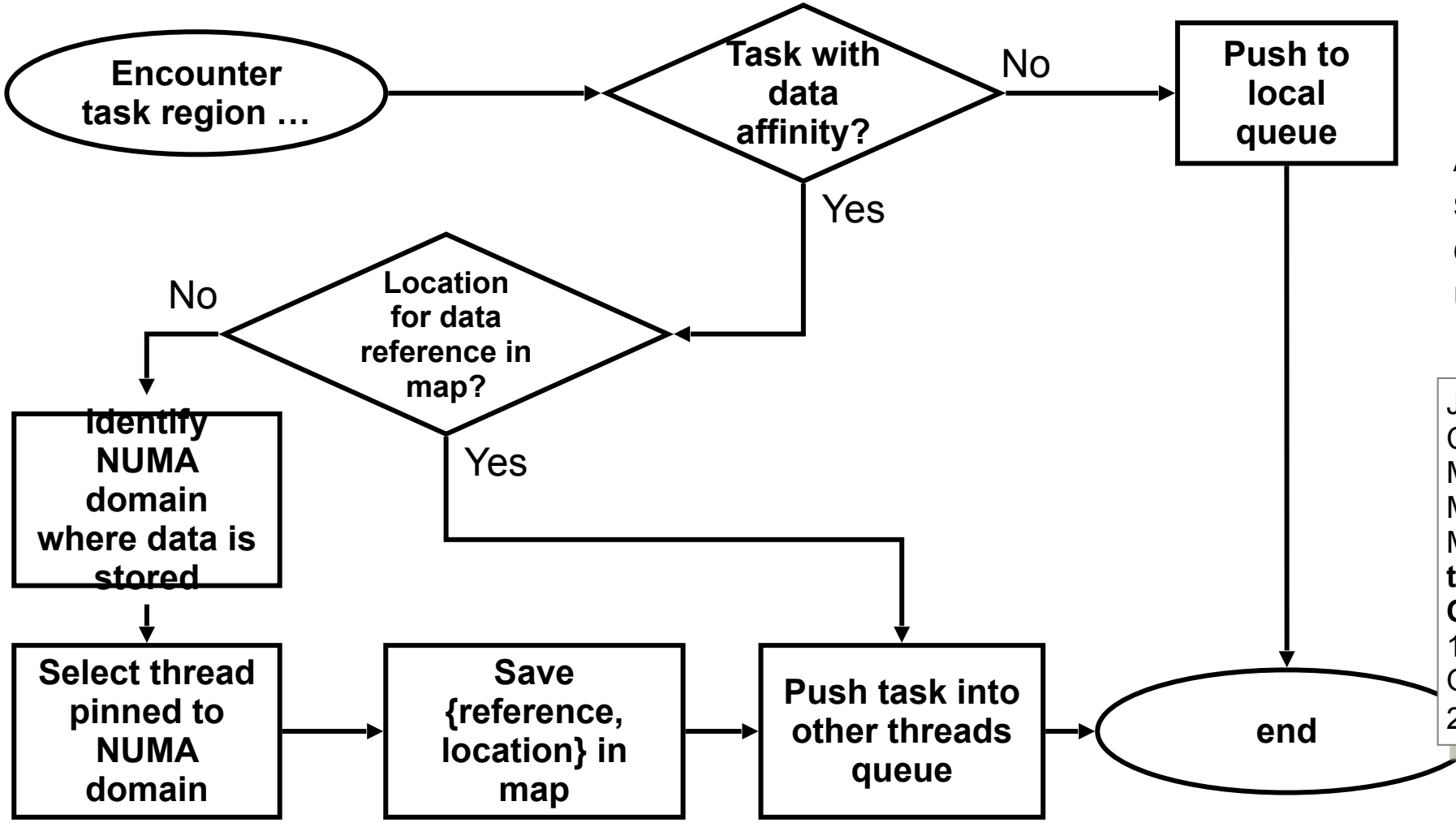

A map is introduced to store location information of data that was previously used

# Selected LLVM implementation details




A map is introduced to store location information of data that was previously used

# Selected LLVM implementation details

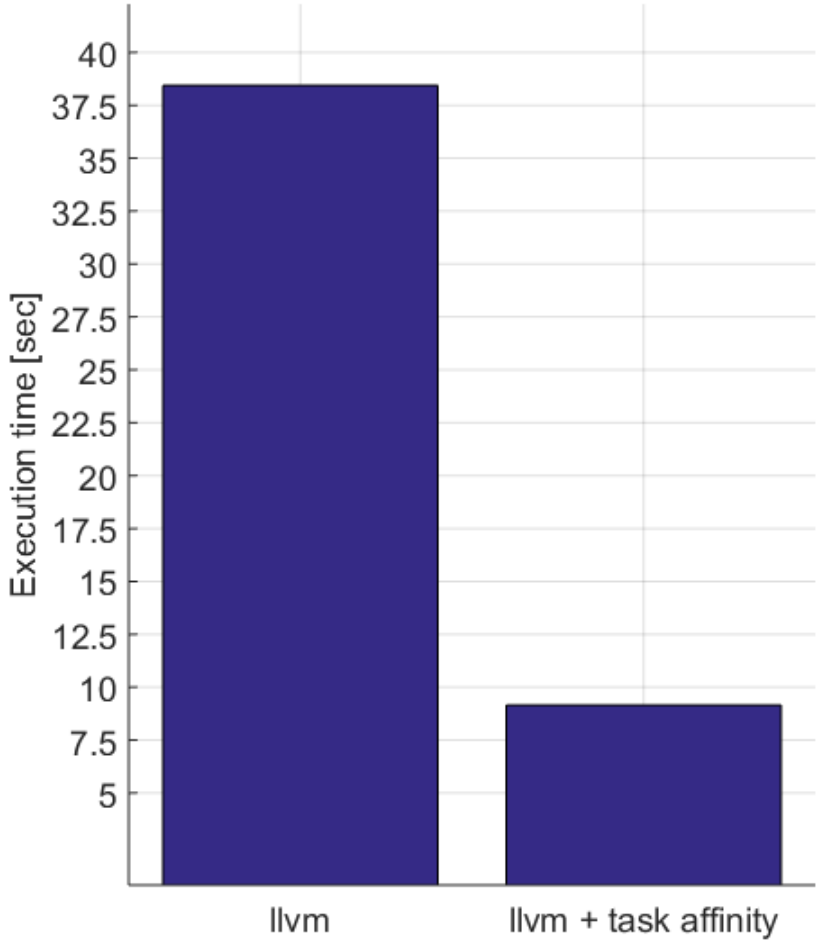



A map is introduced to store location information of data that was previously used

Jannis Klinkenberg, Philipp Samfass, Christian Terboven, Alejandro Duran, Michael Klemm, Xavier Teruel, Sergi Mateo, Stephen L. Olivier, and Matthias S. Müller. **Assessing Task-to-Data Affinity in the LLVM OpenMP Runtime**. Proceedings of the 14th International Workshop on OpenMP, IWOMP 2018. September 26-28, 2018, Barcelona, Spain.

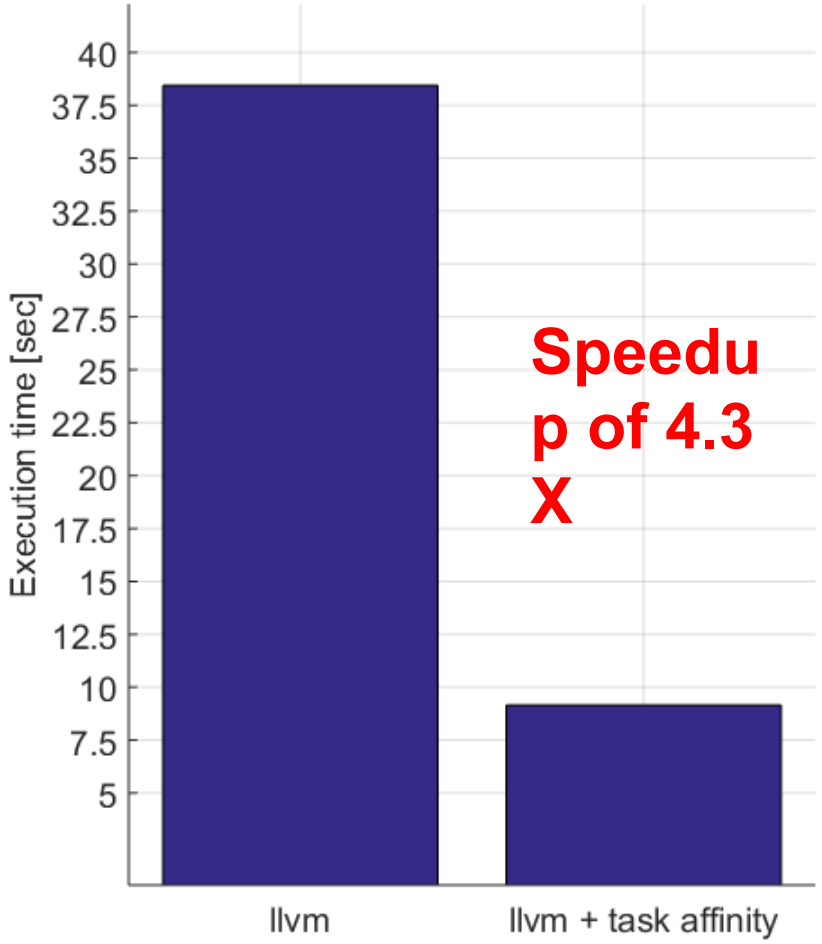
# Evaluation

## Program runtime
## Median of 10 runs



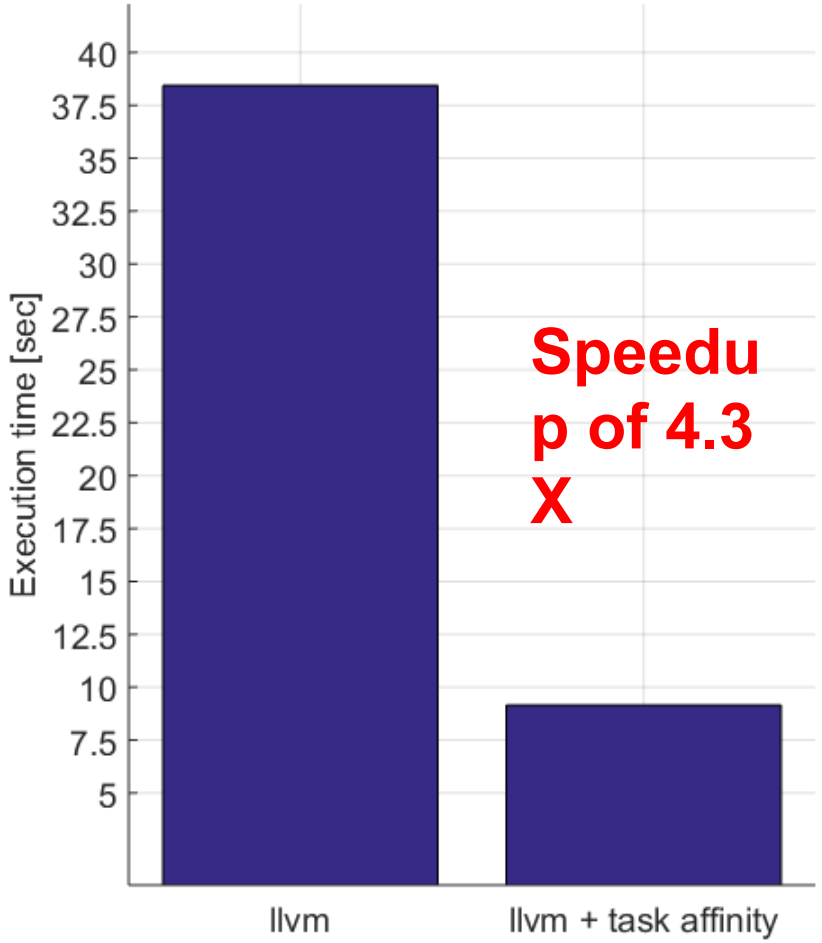
# Evaluation

**Program runtime**
**Median of 10 runs**



# Evaluation

**Program runtime**
**Median of 10 runs**

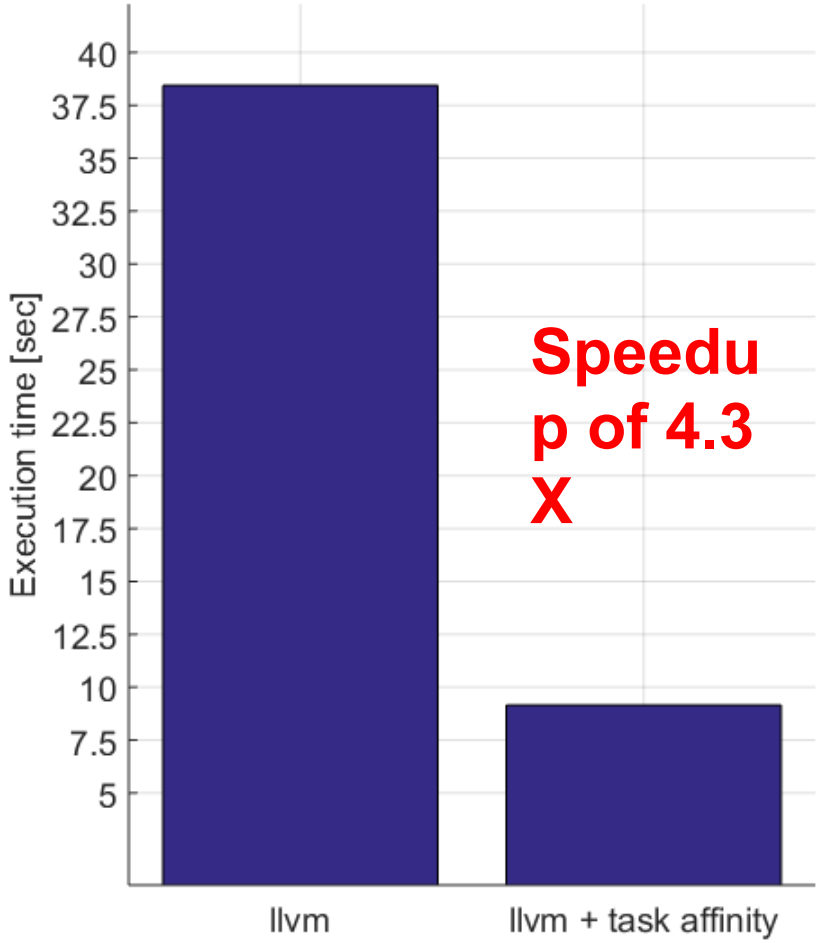



**Speedup of 4.3X**

**LIKWID: reduction of remote data volume from 69% to 13%**

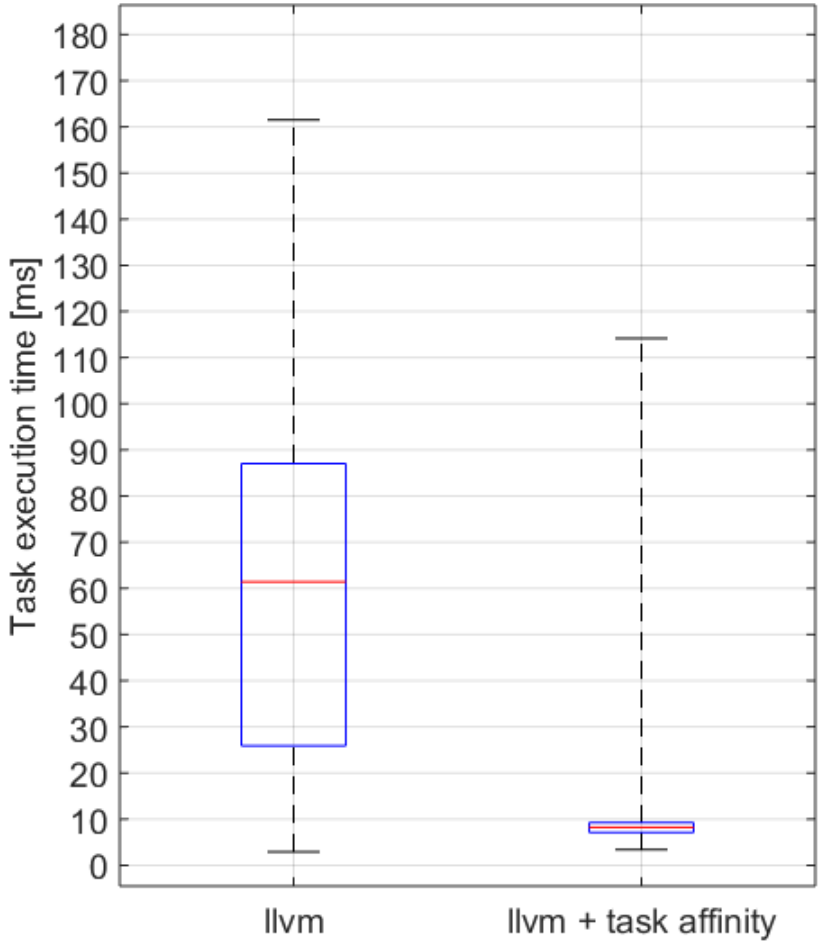
# Evaluation

## Program runtime
## Median of 10 runs




**Speedup of 4.3X**

## Distribution of single task execution times




**LIKWID: reduction of remote data volume from 69% to 13%**

# Summary

- Requirement for this feature: thread affinity enabled

- The `affinity` clause helps, if
  - à tasks access data heavily
  - à single task creator scenario, or task not created with data affinity
  - à high load imbalance among the tasks

- Different from thread binding: task stealing is absolutely allowed

OpenMP

# Case Study: NWChem TCE CCSD(T)

TCE: Tensor Contraction Engine

CCSD(T): Coupled-Cluster with Single, Double, and perturbative Triple replacements

# NWChem

- Computational chemistry software package
  - Quantum chemistry
  - Molecular dynamics
- Designed for large-scale supercomputers
- Developed at the EMSL at PNNL
  - EMSL: Environmental Molecular Sciences Laboratory
  - PNNL: Pacific Northwest National Lab
- URL: http://www.nwchem-sw.org

# Finding Offload Candidates

- Requirements for offload candidates
  - Compute-intensive code regions (kernels)
  - Highly parallel
  - Compute scaling stronger than data transfer, e.g., compute $O(n^3)$ vs. data size $O(n^2)$

# Example Kernel (1 of 27 in total)

```
      subroutine sd_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,
     1                 h7d,triplesx,t2sub,v2sub)
c     Declarations omitted.
      double precision triplesx(h3d*h2d,h1d,p6d,p5d,p4d)
      double precision t2sub(h7d,p4d,p5d,h1d)
      double precision v2sub(h3d*h2d,p6d,h7d)
!$omp target „presence?(triplesx,t2sub,v2sub)"
!$omp teams distribute parallel do private(p4,p5,p6,h2,h3,h1,h7)
      do p4=1,p4d
      do p5=1,p5d
      do p6=1,p6d
      do h1=1,h1d
      do h7=1,h7d
      do h2h3=1,h3d*h2d
       triplesx(h2h3,h1,p6,p5,p4)=triplesx(h2h3,h1,p6,p5,p4)
     1    - t2sub(h7,p4,p5,h1)*v2sub(h2h3,p6,h7)
      end do
      end do
      end do
      end do
      end do
      end do
      end do
!$omp end teams distribute parallel do
!$omp end target
      end subroutine
```

- All kernels have the same structure
- 7 perfectly nested loops
- Some kernels contain inner product loop (then, 6 perfectly nested loops)
- Trip count per loop is equal to “tile size” (20-30 in production)

# Example Kernel (1 of 27 in total)

```
      subroutine sd_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,
     1                 h7d,triplesx,t2sub,v2sub)
c     Declarations omitted.
      double precision triplesx(h3d*h2d,h1d,p6d,p5d,p4d)
      double precision t2sub(h7d,p4d,p5d,h1d)
      double precision v2sub(h3d*h2d,p6d,h7d)
!$omp target „presence?(triplesx,t2sub,v2sub)"
!$omp teams distribute parallel do private(p4,p5,p6,h2,h3,h1,h7)
      do p4=1,p4d
      do p5=1,p5d
      do p6=1,p6d
      do h1=1,h1d
      do h7=1,h7d
      do h2h3=1,h3d*h2d
       triplesx(h2h3,h1,p6,p5,p4)=triplesx(h2h3,h1,p6,p5,p4)
     1    - t2sub(h7,p4,p5,h1)*v2sub(h2h3,p6,h7)
      end do
      end do
      end do
      end do
      end do
      end do
      end do
!$omp end teams distribute parallel do
!$omp end target
      end subroutine
```

- All kernels have the same structure
- 7 perfectly nested loops
- Some kernels contain inner product loop (then, 6 perfectly nested loops)
- Trip count per loop is equal to “tile size” (20-30 in production)
- Naïve data allocation (tile size 24)
  - Per-array transfer for each target construct
  - triplesx: 1458 MB
  - t2sub, v2sub: 2.5 MB each

# Example Kernel (1 of 27 in total)

```
      subroutine sd_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,
     1                     h7d,triplesx,t2sub,v2sub)
c     Declarations omitted.
      double precision triplesx(h3d*h2d,h1d,p6d,p5d,p4d)
      double precision t2sub(h7d,p4d,p5d,h1d)
      double precision v2sub(h3d*h2d,p6d,h7d)
!$omp target „presence?(triplesx,t2sub,v2sub)"
!$omp teams distribute parallel do private(p4,p5,p6,h2,h3,h1,h7)
      do p4=1,p4d
      do p5=1,p5d
      do p6=1,p6d
      do h1=1,h1d
      do h7=1,h7d
      do h2h3=1,h3d*h2d
       triplesx(h2h3,h1,p6,p5,p4)=triplesx(h2h3,h1,p6,p5,p4)
     1   - t2sub(h7,p4,p5,h1)*v2sub(h2h3,p6,h7)
      end do
      end do
      end do
      end do
      end do
      end do
!$omp end teams distribute parallel do
!$omp end target
      end subroutine
```

1.5GB data transferred (host to device)

1.5GB data transferred (device to host)

- All kernels have the same structure
- 7 perfectly nested loops
- Some kernels contain inner product loop (then, 6 perfectly nested loops)
- Trip count per loop is equal to "tile size" (20-30 in production)
- Naïve data allocation (tile size 24)
  - Per-array transfer for each target construct
  - triplesx: 1458 MB
  - t2sub, v2sub: 2.5 MB each

# Invoking the Kernels / Data Management

- Simplified pseudo-code

```
!$omp target enter data map(alloc:triplesx(1:tr_size))
c     for all tiles
      do ...
        call zero_triplesx(triplesx)
        do ...
          call comm_and_sort(t2sub, v2sub)
!$omp target data map(to:t2sub(t2_size)) map(to:v2sub(v2_size))
          if (...)
            call sd_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,h7,triplesx,t2sub,v2sub)
          end if
c         same for sd_t_d1_2 until sd_t_d1_9
!$omp target end data
        end do
        do ...
c         Similar structure for sd_t_d2_1 until sd_t_d2_9, incl. target data
        end do
        call sum_energy(energy, triplesx)
      end do
!$omp target exit data map(release:triplesx(1:size))
```

- Reduced data transfers:

# Invoking the Kernels / Data Management

- Simplified pseudo-code

```
!$omp target enter data map(alloc:triplesx(1:tr_size))
c     for all tiles
      do …
       call zero_triplesx(triplesx)
       do …
        call comm_and_sort(t2sub, v2sub)
!$omp target data map(to:t2sub(t2_size)) map(to:v2sub(v2_size))
         if (…)
          call sd_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,h7,triplesx,t2sub,v2sub)
         end if
c         same for sd_t_d1_2 until sd_t_d1_9
!$omp target end data
       end do
       do …
c        Similar structure for sd_t_d2_1 until sd_t_d2_9, incl. target data
       end do
       call sum_energy(energy, triplesx)
      end do
!$omp target exit data map(release:triplesx(1:size))
```

Allocate 1.5GB data once, stays on device.

- Reduced data transfers:
  - triplesx:
    - allocated once
    - always kept on the target

# Invoking the Kernels / Data Management

- Simplified pseudo-code

```
!$omp target enter data map(alloc:triplesx(1:tr_size))
c     for all tiles
      do ...
       call zero_triplesx(triplesx)
       do ...
        call comm_and_sort(t2sub, v2sub)
!$omp target data map(to:t2sub(t2_size)) map(to:v2sub(v2_size))
         if (...)
          call sd_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,h7,triplesx,t2su
         end if
c         same for sd_t_d1_2 until sd_t_d1_9
!$omp target end data
       end do
       do ...
c        Similar structure for sd_t_d2_1 until sd_t_d2_9, incl. target data
       end do
       call sum_energy(energy, triplesx)
      end do
!$omp target exit data map(release:triplesx(1:size))
```

Allocate 1.5GB data once, stays on device.

Update 2x2.5MB of data for (potentially) multiple kernels.

- Reduced data transfers:
  - triplesx:
    - allocated once
    - always kept on the target
  - t2sub, v2sub:
    - allocated after comm.
    - kept for (multiple) kernel invocations

# Invoking the Kernels / Data Management

- Simplified pseudo-code

```
!$omp target enter data map(alloc:triplesx(1:tr_size))
c     for all tiles
      do …
        call zero_triplesx(triplesx)
        do …
          call comm_and_sort(t2sub, v2sub)
!$omp target data map(to:t2sub(t2_size)) map(to:v2sub(v2_size))
          if (…)
            call sd_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,h7,triplesx,t2sub,v2sub)
          end if
c         same for sd_t_d1_2 until sd_t_d1_9
!$omp target end data
        end do
        do …
c         Similar structure for sd_t_d2_1 until sd_t_d2_9, incl. target data
        end do
        call sum_energy(energy, triplesx)
      end do
!$omp target exit data map(release:triplesx(1:size))
```

Allocate 1.5G
stays on

Update 2x2.5
(potentially) r

```
      subroutine sd_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,
     1                h7d,triplesx,t2sub,v2sub)
c     Declarations omitted.
      double precision triplesx(h3d*h2d,h1d,p6d,p5d,p4d)
      double precision t2sub(h7d,p4d,p5d,h1d)
      double precision v2sub(h3d*h2d,p6d,h7d)
!$omp target „presence?(triplesx,t2sub,v2sub)"
!$omp teams distribute parallel do private(p4,p5,p6,h2,h3,h1,h7)
      do p4=1,p4d
      do p5=1,p5d
      do p6=1,p6d
      do h1=1,h1d
      do h7=1,h7d
      do h2h3=1,h3d*h2d
       triplesx(h2h3,h1,p6,p5,p4)=triplesx(h2h3,h1,p6,p5,p4)
     1   - t2sub(h7,p4,p5,h1)*v2sub(h2h3,p6,h7)
      end do
      end do
      end do
      end do
      end do
      end do
      end do
!$omp end teams distribute parallel do
!$omp end target
      end subroutine
```

# Invoking the Kernels / Data Management

- Simplified pseudo-code

```
!$omp target enter data map(alloc:triplesx(1:tr_size))
c     for all tiles
      do ...
        call zero_triplesx(triplesx)
        do ...
          call comm_and_sort(t2sub, v2sub)
!$omp target data map(to:t2sub(t2_size)) map(to:v2sub(v2_size))
          if (...)
            call sd_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,h7,triplesx,t2sub,v2sub)
          end if
c         same for sd_t_d1_2 until sd_t_d1_9
!$omp target end data
        end do
        do ...
c         Similar structure for sd_t_d2_1 until sd_t_d2_9, incl. target data
        end do
        call sum_energy(energy, triplesx)
      end do
!$omp target exit data map(release:triplesx(1:size))
```

Allocate 1.5G
stays on

Update 2x2.5
(potentially) r

```
      subroutine sd_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,
     1                     h7d,triplesx,t2sub,v2sub)
c     Declarations omitted.
      double precision triplesx(h3d*h2d,h1d,p6d,p5d,p4d)
      double precision t2sub(h7d,p4d,p5d,h1d)
      double precision v2sub(h3d*h2d,p6d,h7d)
!$omp target „presence?(triplesx,t2sub,v2sub)"
!$omp teams distribute parallel do private(p4,p5,p6,h2,h3,h1,h7)
      do p4=1,p4d
      do p5=1,p5d
      do p6=1,p6d
      do h1=1,h1d
      do h7=1,h7d
      do h2h3=1,h3d*h2d
       triplesx(h2h3,h1,p6,p
     1   - t2sub(h7,p4,p5,h1)*v2sub(h2h3,p6,h7)
      end do
      end do
      end do
      end do
      end do
      end do
!$omp end teams distribute parallel do
!$omp end target
      end subroutine
```

Presence check determines that arrays have been allocated in the device data environment already.

# Programming OpenMP

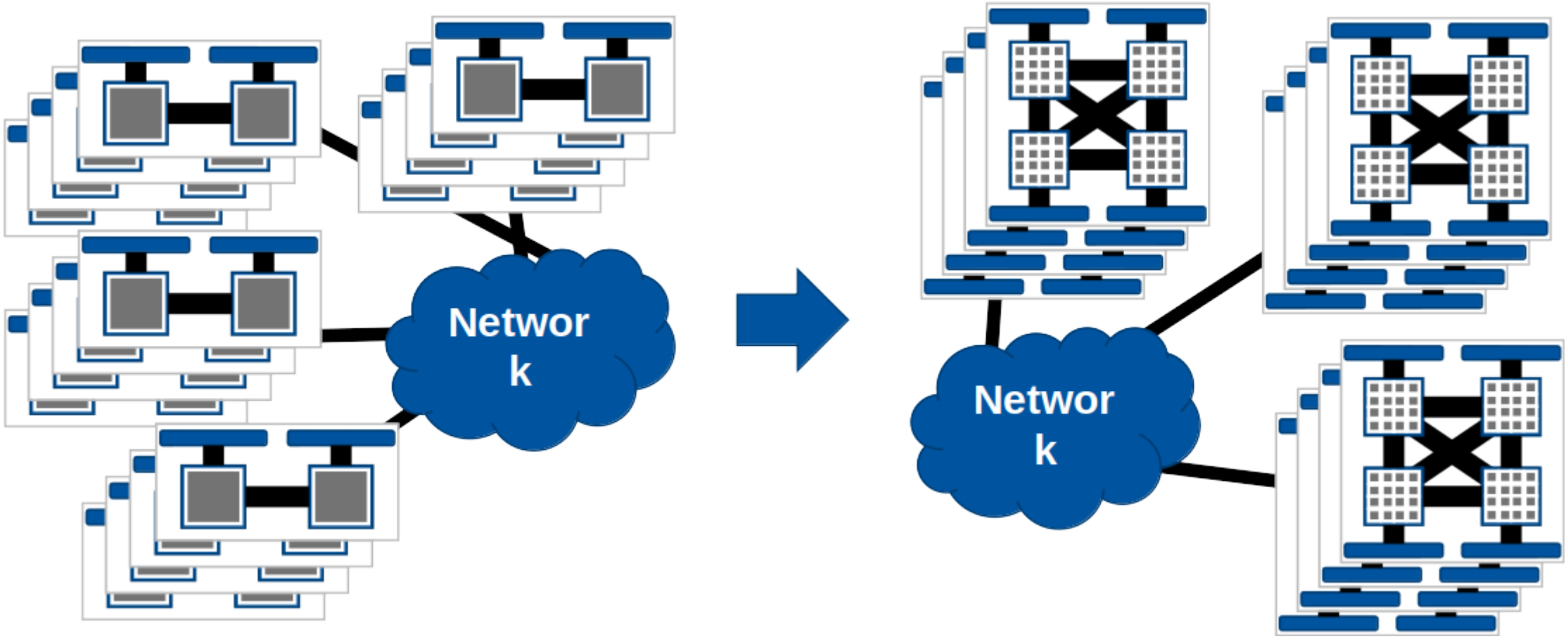
## *OpenMP and MPI*

**Christian Terboven**
Michael Klemm

# Motivation

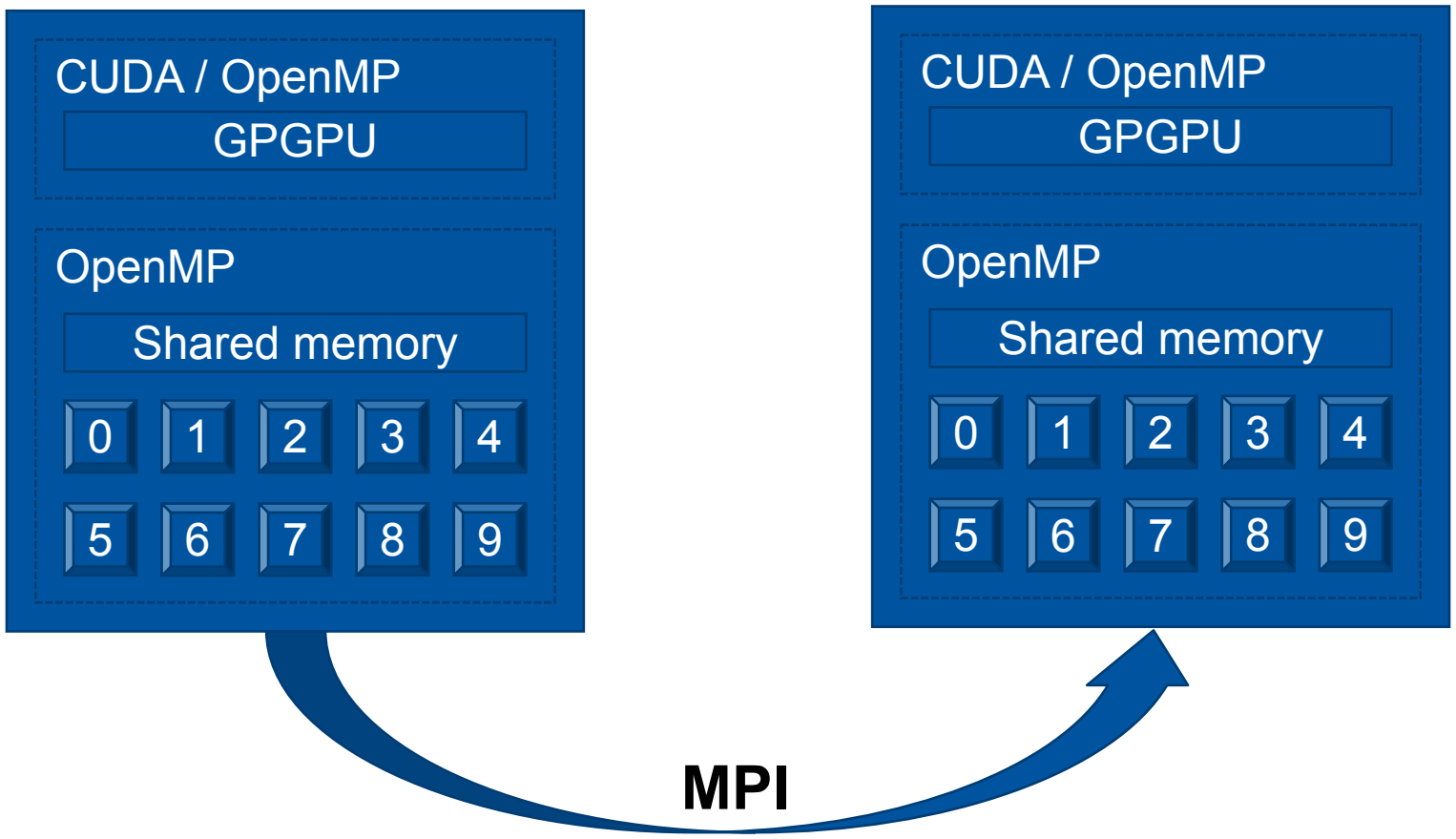
# Motivation for hybrid programming

- Increasing number of cores per node



# Hybrid programming

- (Hierarchical) mixing of different programming paradigms



# MPI and OpenMP

# MPI – threads interaction

- MPI needs special initialization in a threaded environment
  - Use MPI_Init_thread to communicate thread support level

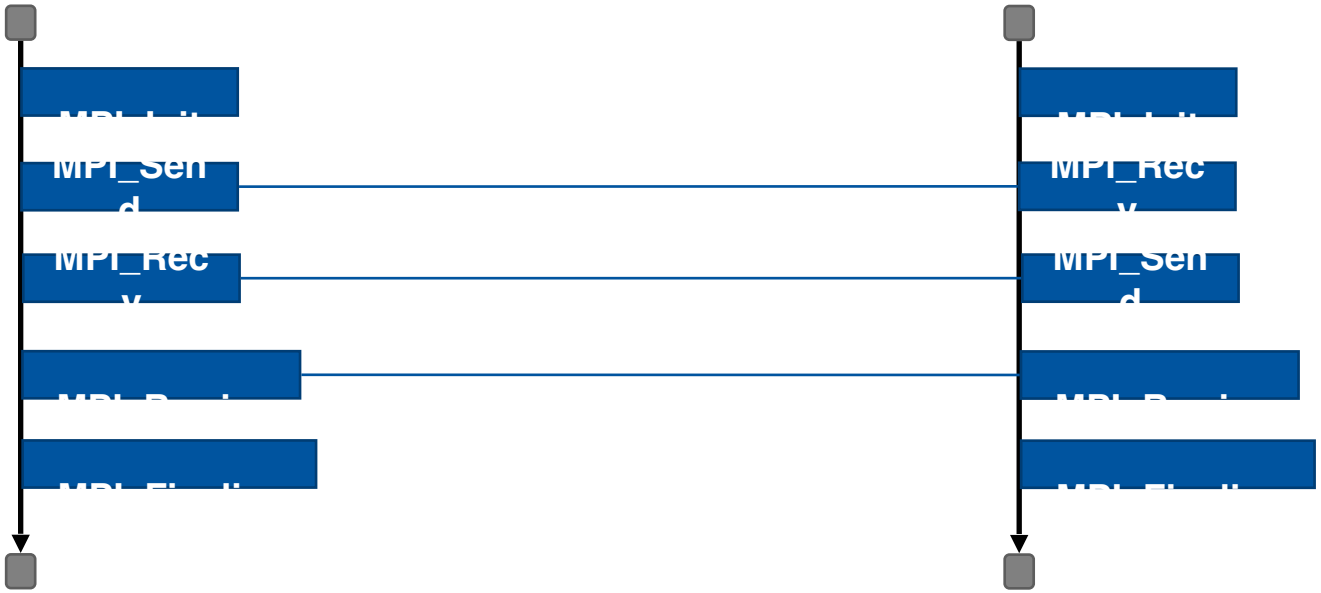
- Four levels of threading support


| Level identifier | Description |
|---|---|
| **MPI_THREAD_SINGLE** | Only one thread may execute |
| **MPI_THREAD_FUNNELED** | Only the main thread may make MPI calls |
| **MPI_THREAD_SERIALIZED** | Any one thread may make MPI calls at a time |
| **MPI_THREAD_MULTIPLE** | Multiple threads may call MPI concurrently with no restrictions |

- MPI_THREAD_MULTIPLE may incur significant overhead inside an MPI implementation

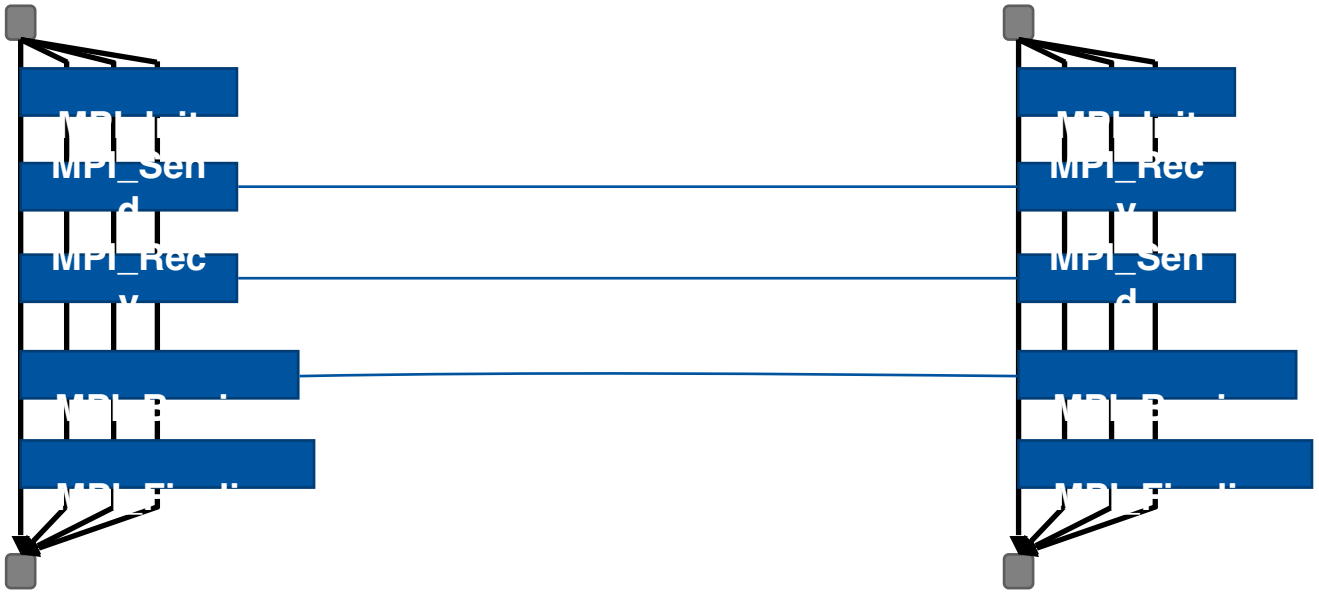
# MPI – Threading support levels

- MPI_THREAD_SINGLE
  - Only one thread per MPI rank

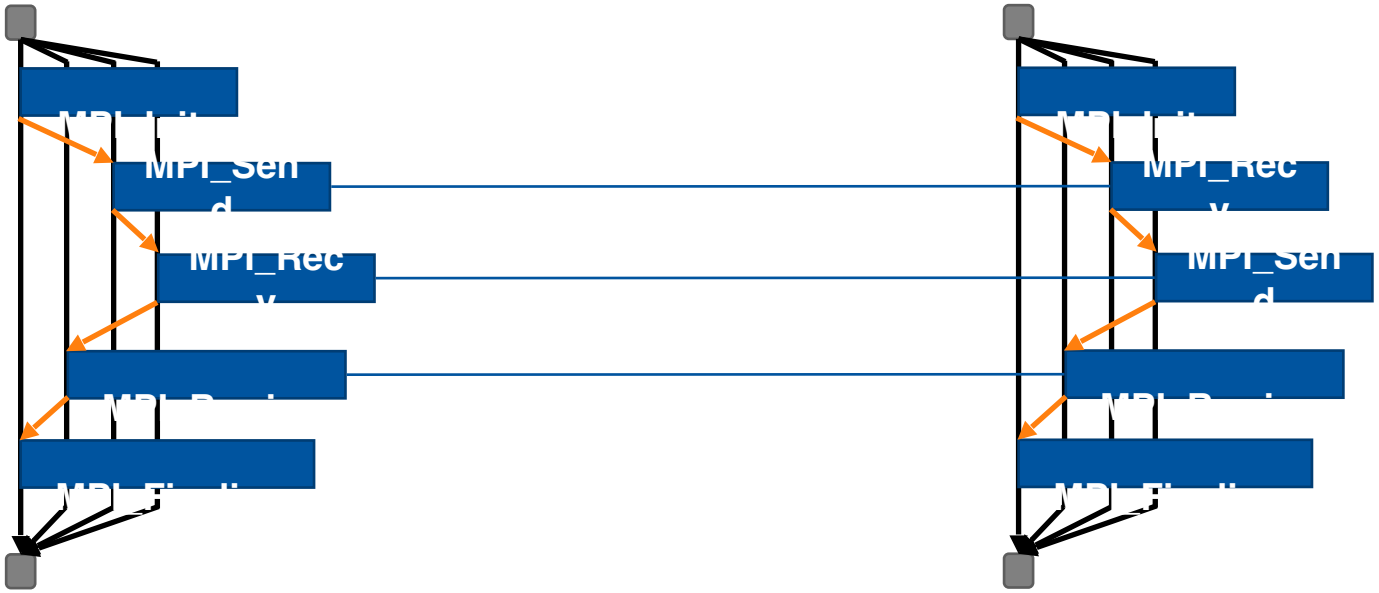
# MPI – Threading support levels

- MPI_THREAD_FUNNELED
  - Only one thread communicates

# MPI – Threading support levels

- MPI_THREAD_SERIALIZED
  - Only one thread communicates at a time

# MPI – Threading support levels

- MPI_THREAD_MULTIPLE
  - All threads communicate concurrently without synchronizatio

MPI Communication
Thread Synchronization