Extending OpenMP to Facilitate Loop Optimization

Ian Bertolacci	-	University of Arizona
Michelle Mills Strout	-	University of Arizona
Bronis R. de Supinski	-	Lawrence Livermore National Laboratory
Tom R. W. Scogland	-	Lawrence Livermore National Laboratory
Eddie C. Davis	-	Boise State University
Catherine Olschanowsky	-	Boise State University

Outline

- Limitations of OpenMP.
- Motivation for sequential transformations in OpenMP.
- Proposed Optimizations.
  - Unroll
  - Fuse
  - Tile
- Composition of loop transformations.
- Interaction with other OpenMP constructs.
- Discussion.

Limitations

OpenMP Does:

- Loop Parallelism
- Task Parallelism
- Accelerator Offloading

OpenMP Doesn't:

- Basic loop transformations
  - Unrolling
  - Fusion
  - Tiling

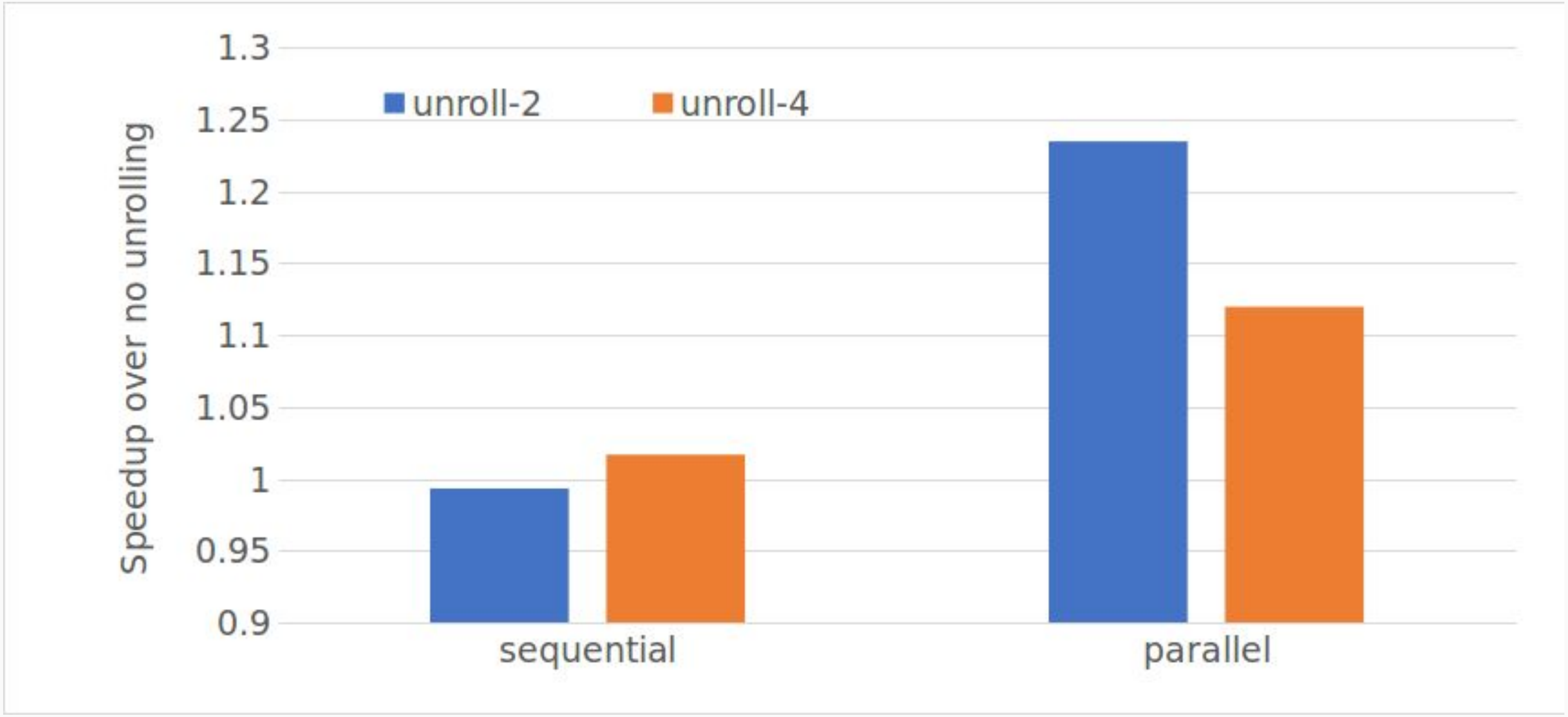
Why?

Why should OpenMP provide sequential loop optimizations?

1. Provide standard for portable sequential loop optimization directives and semantics.
2. Sequential optimizations improve parallel code performance.
3. OpenMP semantics obscure/prevent them from being automatically performed by the compiler.

Performance Improvements with Loop Unroll

CLOMPK (CORAL2 benchmark)



Possible Unroll Syntax and Semantics Example

```
#pragma omp for **unroll( 2 )** schedule( static, 1 ) nowait
for( int i = 0; i < n; i += 1 )
  A[i] += B[i] * c;
```

```
#pragma omp for schedule( static, 1 ) nowait
for( int i = 0; i < n; i += 2 ){
  A[i] += B[i] * c;
  A[i+1] += B[i+1] * c;
}
```

Loop Chain Optimizations

Context:

- Data sharing between loop nests are an opportunity to exploit caching and reduce temporary storage.

Problem:

- Manual modifications difficult.
- Automatic transformation requires complex analysis.

Solution:

- Provide user high-level loop transformation directives for fusion and tiling.
- Developer describes loop bounds and data accesses.
- Compiler uses information to perform transformations legally.

Loop Chain Annotations

```
for( int i = lb; i <= ub; i += 1 )
  A[i] = B[i-1] + B[i] + B[i+1];
```

```
for( int i = lb; i <= ub; i += 1 )
  A[i] = A[i] / 3.0;
```

Loop Chain Annotations

```
#pragma omplc loopchain schedule( /* ... */ )
{
  #pragma omplc for domain(lb:ub) \
    with (i) \
      read  B {(i-1), (i), (i+1)}, \
      write A {(i)}
  for( int i = lb; i <= ub; i += 1 )
    A[i] = B[i-1] + B[i] + B[i+1];
  #pragma omplc for domain(lb:ub) \
    with (i) \
      write A {(i)}, \
      read  A {(i)}
  for( int i = lb; i <= ub; i += 1 )
    A[i] = A[i] / 3.0;
}
```

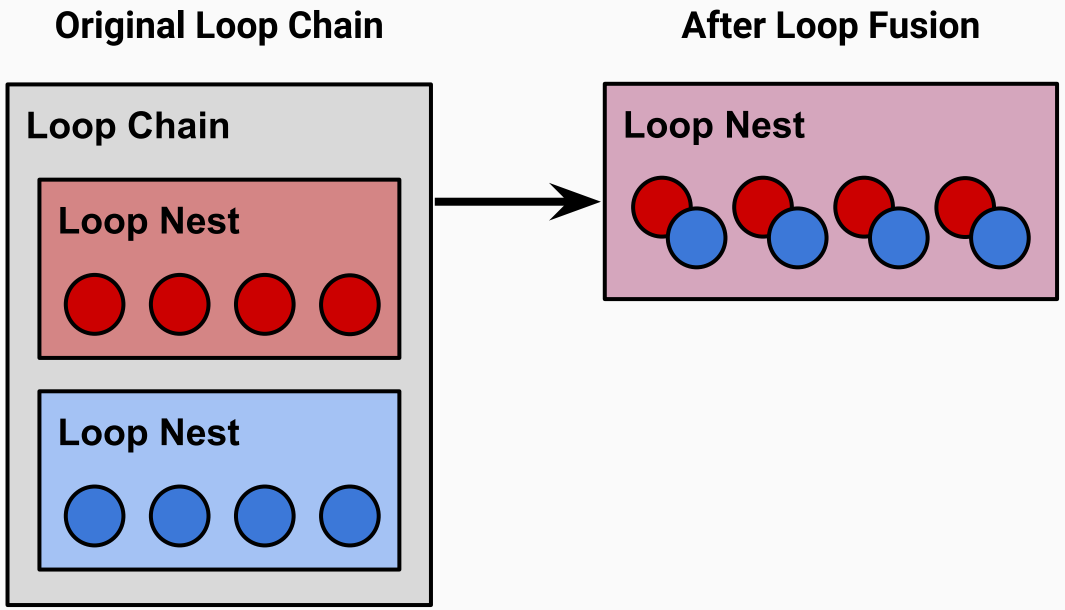
Denotes that this scope contains loop-chain, and list scheduling operations.

Denotes loop nests. Defines iteration spaces.

Define read/writes set for dataspaces at iteration **i**.

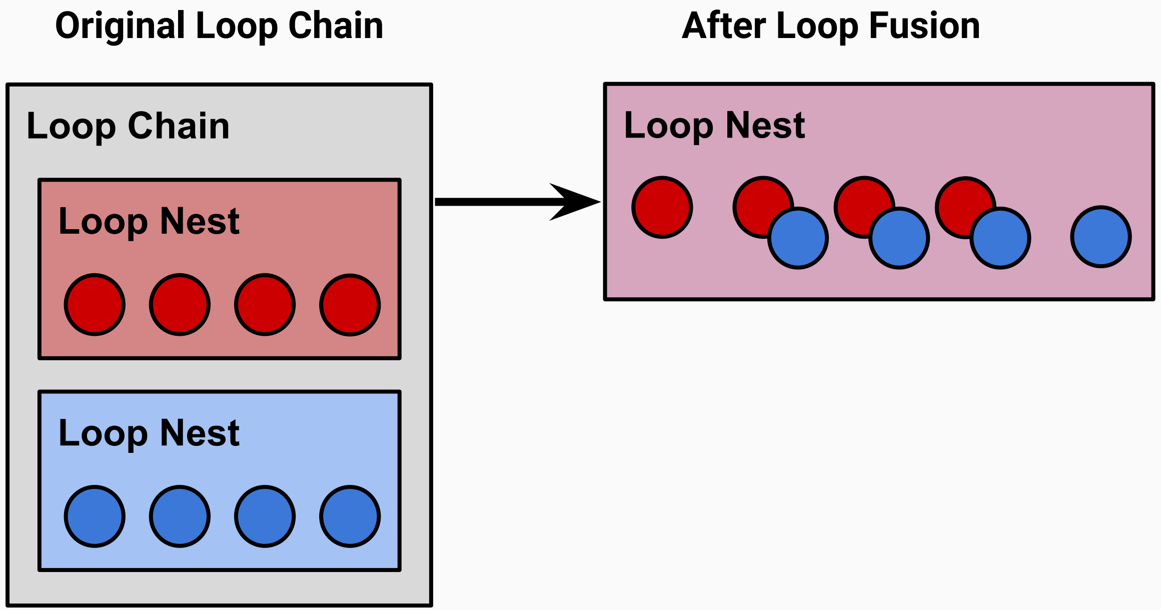
Loop Chain Fusion

```
schedule( fuse() )
```



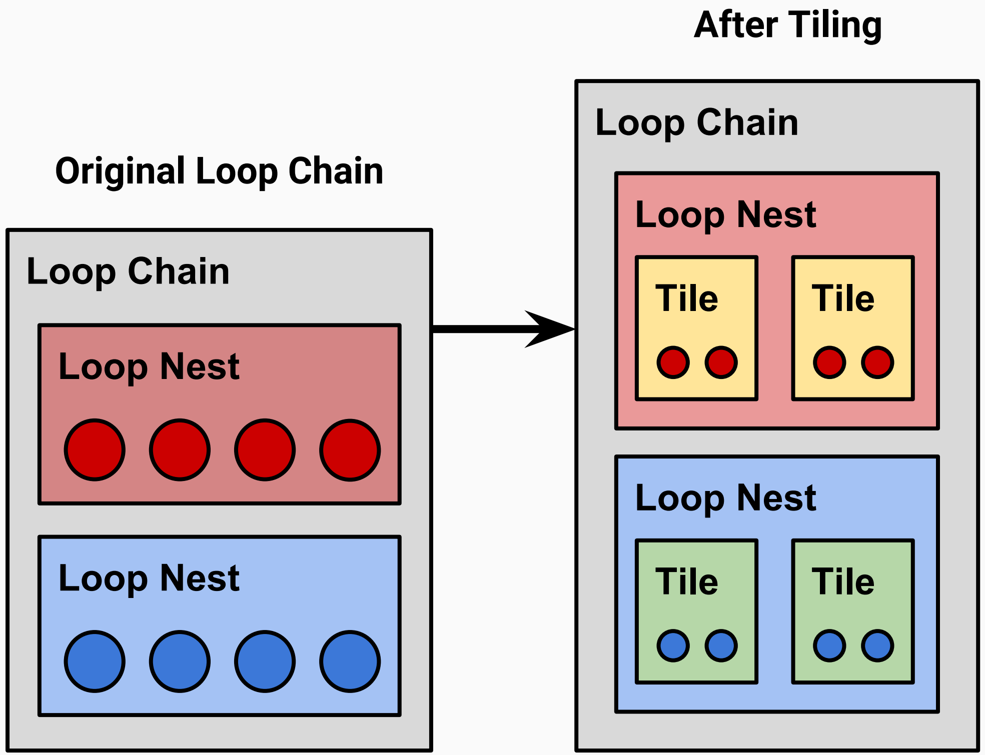
Loop Chain Fusion with Explicit Shift

```
schedule( fuse( (0), (1) ) )
```



Loop Chain Tiling

```
schedule( tile( (2), parallel, serial ) )
```

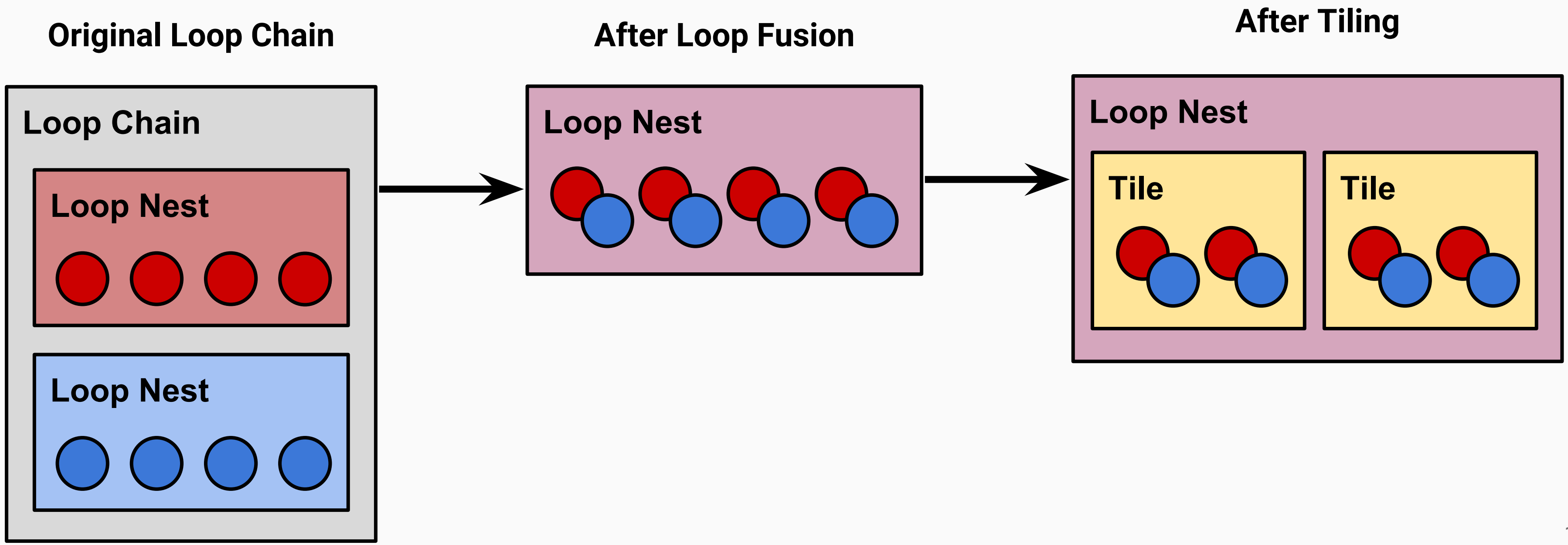


Composition of Multiple Sequential Loop Optimizations

- How should sequential loop optimizations compose?
  - Previous loop chain tool easily composes optimizations.
  - How should we do this in OpenMP?
    - Unify into one framework?
    - Piecemeal?
    - Something in between?

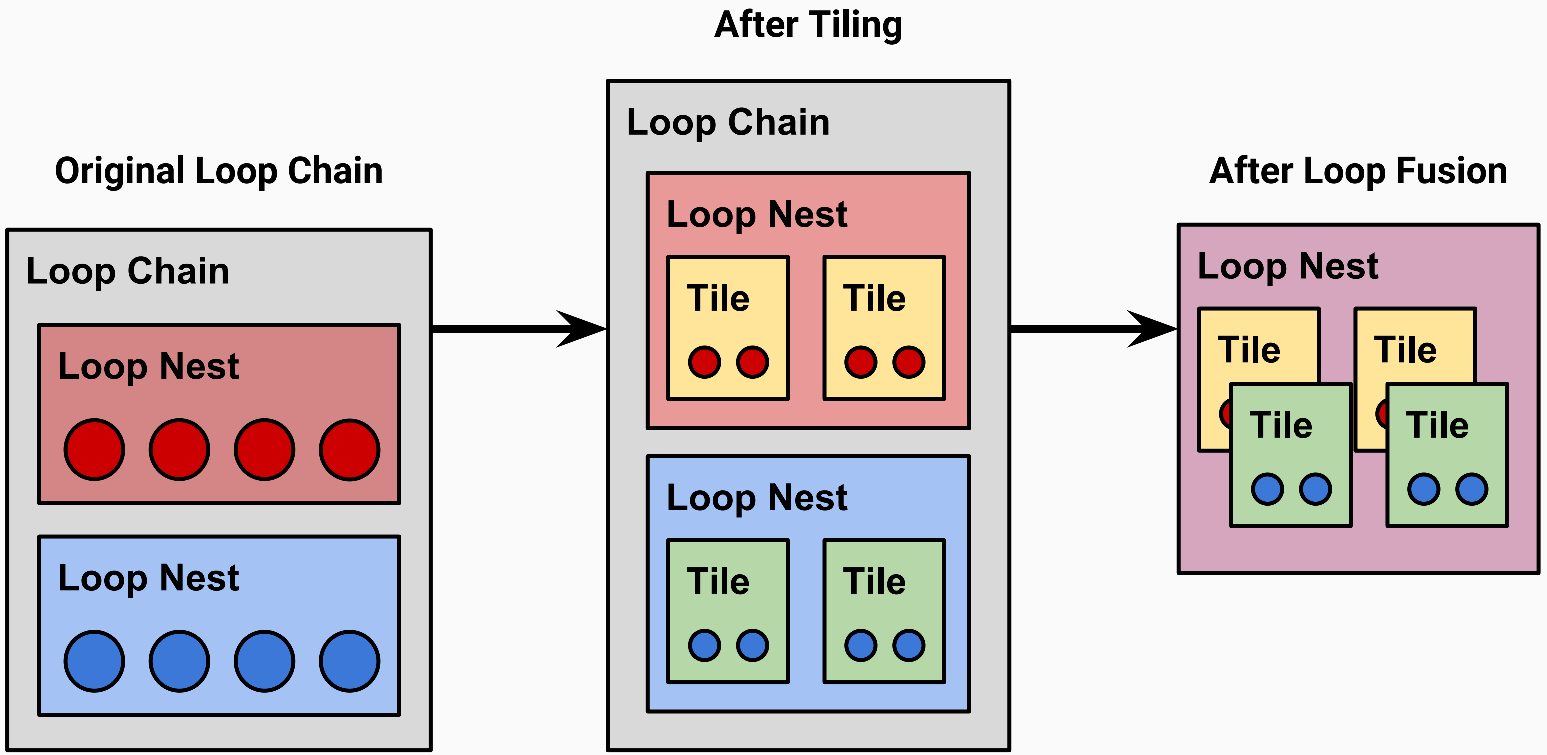
Loop Chain Fuse + Tile

```
schedule( fuse(), tile( (2), parallel, serial ) )
```



Loop Chain Tile + Fuse

```
schedule( tile( (2), parallel, serial ), fuse() )
```



Interaction With Other OpenMP Constructs

How do sequential loop optimizations interact with different OpenMP constructs?

- For example: what would loop fusion on two OpenMP parallel loops do?
    - Union clauses?
    - Invalid?
- Cross product of constructs is large.

Summary

- OpenMP API can provide methods of prescribing sequential loop optimizations.
- Sequential loop optimizations have positive performance impacts for parallel code.
- Developer can augment portions of analysis.
- API can provide multiple pathways for performing same transformations.

Discussion

- How could we compose different sequential loop optimizations?
    - Unified framework (à la loop chain)?
    - Piecemeal?
    - Something in between?
- How do we deal with interactions between different sequential loop and other OpenMP constructs?
    - Unified model?
    - Explicitly connect different components?
- What is the set of important sequential loop optimizations?