OpenMP

SC'20 Booth Talk Series

OMP5.1: Loop Transformations

Michael Kruse, Argonne National Laboratory

Outline

Vectorization Compiler Hints

- Cray
  `#pragma ivdep`

Vectorization Compiler Hints

- Cray
  `#pragma ivdep`
- SGI/Open64
  `#pragma ivdep`
- HP
  `#pragma IVDEP`
- Intel
  `#pragma ivdep`
- PGI
  `#pragma ivdep`

Vectorization Compiler Hints

- Cray
  `#pragma [_CRI] ivdep`
- SGI/Open64
  `#pragma ivdep`
- HP
  `#pragma IVDEP`
- Intel
  `#pragma ivdep`
- PGI
  `#pragma ivdep`
- gcc
  `#pragma GCC ivdep`

Vectorization Compiler Hints

- Cray
  `#pragma [_CRI] ivdep`
- SGI/Open64
  `#pragma ivdep`
- HP
  `#pragma IVDEP`

- Intel
  `#pragma ivdep`

- PGI
  `#pragma ivdep`

- gcc
  `#pragma GCC ivdep`
- msvc
  `#pragma loop(ivdep)`
- clang
  `#pragma clang loop vectorize(enable)`

Vectorization Compiler Hints

- Cray
  `#pragma [_CRI] ivdep`
- SGI/Open64
  `#pragma ivdep`
- HP
  `#pragma IVDEP`

- Intel
  `#pragma ivdep`
  `#pragma vector`
  `#pragma simd`
- PGI
  `#pragma ivdep`
  `#pragma vector`

- gcc
  `#pragma GCC ivdep`
- msvc
  `#pragma loop(ivdep)`
- clang
  `#pragma clang loop vectorize(enable)`

Vectorization Compiler Hints

- Cray
  `#pragma [_CRI] ivdep`
- SGI/Open64
  `#pragma ivdep`
- HP
  `#pragma IVDEP`
  `#pragma NODEPCHK`
- Intel
  `#pragma ivdep`
  `#pragma vector`
  `#pragma simd`
- PGI
  `#pragma ivdep`
  `#pragma vector`
  `#pragma nodepchk`
- gcc
  `#pragma GCC ivdep`
- msvc
  `#pragma loop(ivdep)`
- clang
  `#pragma clang loop vectorize(enable)`
- Oracle Developer Studio
  `#pragma nomemorydepend`

WARNING: Different Semantics

- Cray's `ivdep` semantics is defined by its implementation:
  Assume only dependencies exist that the compiler detects
- Others are implementation-independent:
  Assume no loop-carried dependencies exist
  (implies no reductions)
- Some implement both:
  `ivdep`/`nodepchk`

SIMD Directive

OpenMP 4.0

```
#pragma omp simd
for (int i = 0; i < n; ++i)
  body(i);
```


SIMD Directive

OpenMP 4.0

```
#pragma omp simd
for (int i = 0; i < n; ++i)
  body(i);
```

- Clang supports a SIMD-only mode
  `clang -fopenmp-simd`

Outline

Unrolling Compiler Hints

- Cray
  ```
  #pragma [_CRI] unroll 4
  ```
- icc
  ```
  #pragma unroll 4
  ```
- xlc
  ```
  #pragma unroll(4)
  ```
- HP
  ```
  #pragma UNROLL_FACTOR 4
  ```
- gcc
  ```
  #pragma GCC unroll 4
  ```
- clang
  ```
  #pragma unroll 4
  #pragma clang loop unroll_count(4)
  ```
- msvc

  ???

Unroll Directive

OpenMP 5.1

```
#pragma omp unroll
for (int i = 0; i < n; ++i)
  body(i);
```


Unroll Directive Effect

Full Unrolling

```
#pragma omp unroll full
for (int i = 0; i < 4; i += 1)
  Stmt(i);
```

```
Stmt(0);
Stmt(1);
Stmt(2);
Stmt(3);
```


Unroll Directive Effect

Partial Unrolling

```
#pragma omp unroll partial(4)
for (int i = 0; i < n; i += 1)
  body(i);
```

```
for (int i = 0; i < n; i += 4) {
  body(i);
  if (i + 1 < n) body(i + 1);
  if (i + 2 < n) body(i + 2);
  if (i + 3 < n) body(i + 3);
}
```


Unroll Directive Effect

Partial Unrolling

```
#pragma omp unroll partial(4)
for (int i = 0; i < n; i += 1)
  body(i);
```

```
int i = 0;
for (; i+3 < n; i += 4) {
  body(i);
  body(i + 1);
  body(i + 2);
  body(i + 3);
}
for (; i < n; i += 1)
  body(i);
```


Unroll Directive Effect

Compiler-determined unroll factor

```
#pragma omp unroll partial
for (int i = 0; i < n; i += 1)
  body(i);
```

```
int i = 0;
for (; i+? < n; i += ?) {
  body(i);
  ...
}
for (; i < n; i += 1)
  body(i);
```


Unroll Directive Effect

Compiler-determined unrolling mode

```
#pragma omp unroll
for (int i = 0; i < n; i += 1)
  body(i);
```

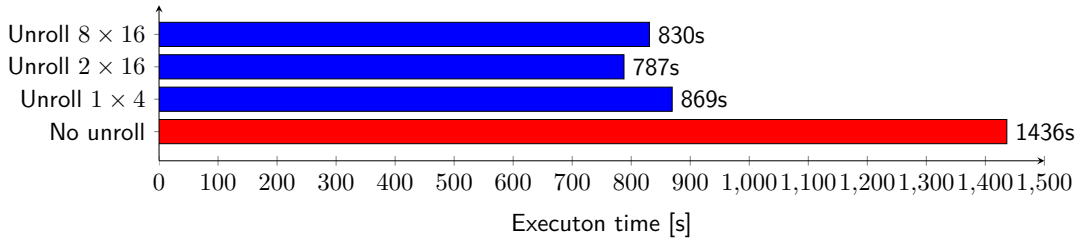

When To Use

- Full Unrolling
  - Small constant trip-count in hot code
    (Unless L1i is a bottleneck)
  - Compiler will probably unroll them automatically
- Partial Unrolling
  - X86 (Intel/AMD): Probably unnecessary
  - Accelerators (Nvida/AMD/Intel): Innermost loops with small bodies

GEMM Performance

NVidia 8800 GTX

```
for (int i = 0; i < NI; i++)
  #pragma omp unroll partial(x)
  for (int j = 0; j < NJ; j++)
    #pragma omp unroll partial(y)
    for (int k = 0; k < NK; k++)
      C[i][j] += alpha * A[i][k] * B[k][j];
```



G. S. Murthy et. al. "Optimal Loop Unrolling for GPGPU Programs" (IPDPS'10)

Outline

OpenACC Tile Clause

- OpenACC
  `#pragma acc loop tile(...)`

Tile Directive

OpenMP 5.1

```
#pragma omp tile sizes(8,8)
for (int i = 0; i < 128; ++i)
  for (int j = 0; j < 128; ++j)
    body(i,j);
```

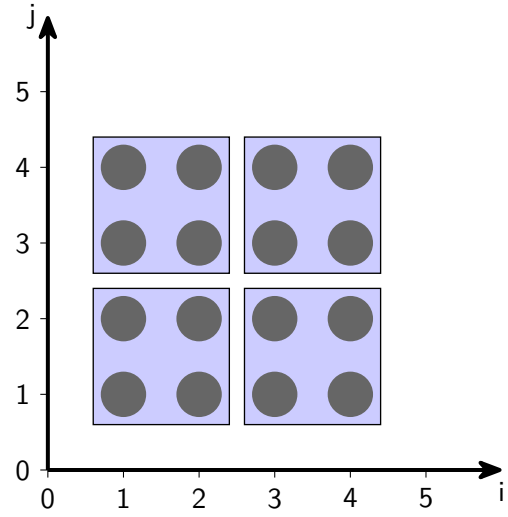

Tile Directive

OpenMP 5.1

```
#pragma omp tile sizes(8,8)
for (int i = 0; i < 128; ++i)
  for (int j = 0; j < 128; ++j)
    body(i,j);
```

```
for (int i1 = 0; i1 < 128; i1 += 8)
  for (int j1 = 0; j1 < 128; j1 += 8)
    for (int i2 = i1; i2 < i1 + 8; i2 += 1)
      for (int j2 = j1; j2 < j2 + 8; j2 += 1)
        body(i2,j2);
```

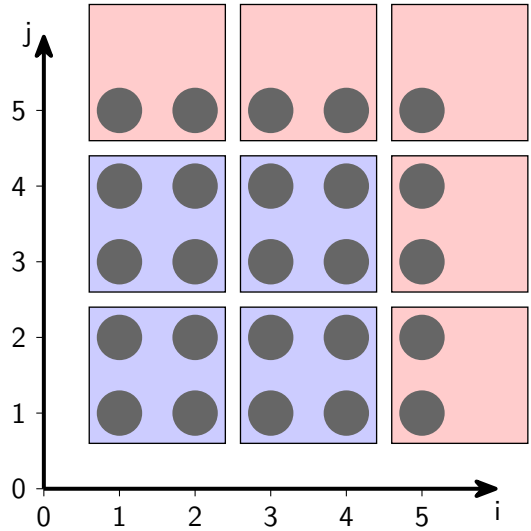

Illustration




```
#pragma omp tile sizes(2,2)
for (int i = 1; i <= 4; ++i)
  for (int j = 1; j <= 4; ++j)
    body(i,j);
```

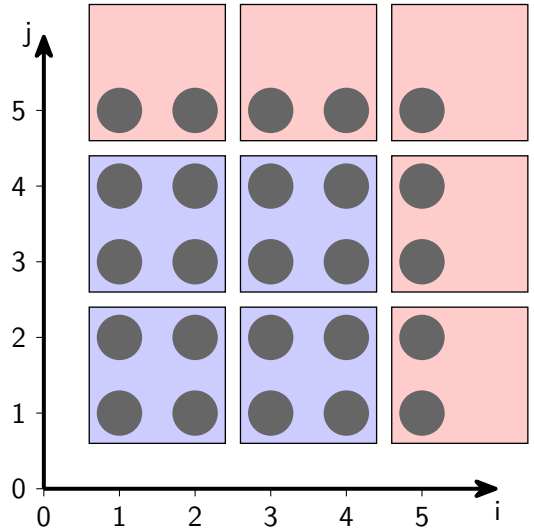
```
/* floor loops iterating over tiles */
for (int i1 = 0; i1 < 4; i1 += 2)
  for (int j1 = 0; j1 < 4; j1 += 2)
    /* tile loops over iterations */
    for (int i2 = i1; i2 < i1 + 2; i2 += 1)
      for (int j2 = j1; j2 < j2 + 2; j2 += 1)
        /* an iteration */
        body(1+i2,1+j2);
```


Partial/Complete Tiles




```
#pragma omp tile sizes(2,2)
for (int i = 1; i <= 5; ++i)
  for (int j = 1; j <= 5; ++j)
    body(i,j);
```


Partial/Complete Tiles



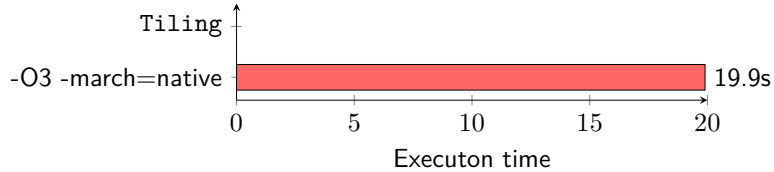

```
#pragma omp tile sizes(2,2)
for (int i = 1; i <= 5; ++i)
  for (int j = 1; j <= 5; ++j)
    body(i,j);
```

```
/* hot, streamlined code */
for (int i1 = 0; i1 < 4; i1 += 2)
  for (int j1 = 0; j1 < 4; j1 += 2)
    for (int i2 = i1; i2 < i1 + 2; i2 += 1)
      for (int j2 = j1; j2 < j2 + 2; j2 += 1)
        body(i2+1,j2+1);

/* special case code */
for (int i = 1; i < 5; ++i)
  for (int j = 1; j < 5; ++j)
    if (i >= 5 || j >= 5)
      body(i,j);
```

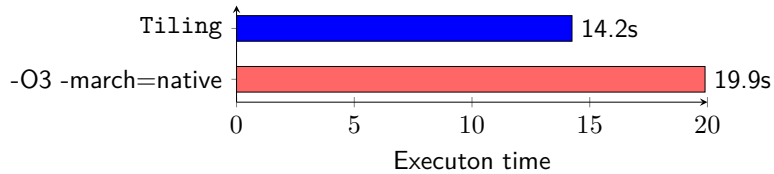

heat-3d Stencil

```
for (int i = 1; i < N-1; i++)
    for (int j = 1; j < N-1; j++)
        for (int k = 1; k < N-1; k++)
            B[i][j][k] =  0.125*(A[i+1][j][k] - 2.0*A[i][j][k] + A[i-1][j][k])
                        + 0.125*(A[i][j+1][k] - 2.0*A[i][j][k] + A[i][j-1][k])
                        + 0.125*(A[i][j][k+1] - 2.0*A[i][j][k] + A[i][j][k-1])
                        + A[i][j][k];
```



heat-3d Stencil

```
#pragma omp tile sizes(16,1,1024)
for (int i = 1; i < N-1; i++)
    for (int j = 1; j < N-1; j++)
        for (int k = 1; k < N-1; k++)
            B[i][j][k] =  0.125*(A[i+1][j][k] - 2.0*A[i][j][k] + A[i-1][j][k])
                        + 0.125*(A[i][j+1][k] - 2.0*A[i][j][k] + A[i][j-1][k])
                        + 0.125*(A[i][j][k+1] - 2.0*A[i][j][k] + A[i][j][k-1])
                        + A[i][j][k];
```



When To Use

- Localize workings set
  - Stencils
  - BLAS (e.g. gemm)
  - Any computation reusing relative indices multiple times
- Split workload (chunking)
  - Coarser-grain parallelism
- Preparation for other transformations
  - Partial unrolling is defined as 1-dimensional tiling followed by a full unroll

Outline

Composition of Tiling

Tiling

- Rule: Directives apply to the next line
  1. A base language canonical loop

```
#pragma omp taskloop
for (int i = 0; i < 128; ++i)
    body(i);
```


Composition of Tiling

Tiling

- Rule: Directives apply to the next line
  1. A base language canonical loop

```
#pragma omp taskloop
for (int i = 0; i < 128; ++i)
    body(i);
```

  2. The output of another loop transformation

```
#pragma omp taskloop
#pragma omp tile sizes(8)
for (int i = 0; i < 128; ++i)
    body(i);
```

```
#pragma omp taskloop
for (int i1 = 0; i1 < 128; i1 += 8)
    for (int i2 = i1; i2 < i1 + 8; i2 += 1)
        body(i2);
```


Multi-Level Tiling

```
#pragma omp tile sizes(4, 4)
#pragma omp tile sizes(5,16)
for (int i = 0; i < 100; ++i)
  for (int j = 0; j < 128; ++j)
    A[i][j] = i*1000 + j;
```

```
#pragma omp tile sizes(4,4)
for (int i1 = 0; i1 < 100; i1+=5)
  for (int j1 = 0; j1 < 128; j1+=16)
    for (int i2 = i1; i2 < i1+5; ++i2)
      for (int j2 = j1; j2 < j1+16; ++j2)
        A[i2][j2] = i2*1000 + j2;
```

```
for (int i11 = 0; i11 < 100; i11+= 5*4)
for (int j11 = 0; j11 < 128; j11+=16*4)
    for (int i12 = i11; i12 < i11+( 5*4); i12+= 5)
    for (int j12 = j11; j12 < j11+(16*4); j12+=16)
        for (int i2 = i12; i2 < i12+ 5; ++i2)
        for (int j2 = j12; j2 < j12+16; ++j2)
            A[i2][j2] = i2*1000 + j2;
```


Composition of Unrolling

```
#pragma omp parallel for
#pragma omp unroll partial(4)
for (int i = 0; i < 128; ++i)
    body(i);
```

```
#pragma omp parallel for
for (int i = 0; i < 128; i += 4) {
    body(i);
    body(i + 1);
    body(i + 2);
    body(i + 3);
}
```


Generated Loops by Unrolling

Unrolling

Unroll directive generated a loop with partial unrolling only:

- `#pragma omp unroll partial(n)`
- `#pragma omp unroll partial`

The replacement of these are not transformable:

- `#pragma omp unroll full`
- `#pragma omp unroll`

Split Optimization from Semantics

- Semantics: Single Fortran/C/C++ source
- Optimization: Pragmas based on target hardware

```
for (int i = 0; i < m; ++i)
  for (int j = 0; j < n; ++j)
    body(i,j);
```


Split Optimization from Semantics

- Semantics: Single Fortran/C/C++ source
- Optimization: Pragmas based on target hardware
  - Using Preprocessor

```
#ifdef ENABLE_OFFLOADING
  #pragma omp distribute parallel for collapse(2)
#else
  #pragma omp for
  #pragma omp tile sizes(16,32)
#endif
for (int i = 0; i < m; ++i)
  #ifdef ENABLE_OFFLOADING
    #pragma omp unroll partial(8)
  #endif
  for (int j = 0; j < n; ++j)
    body(i,j);
```


Split Optimization from Semantics

- Semantics: Single Fortran/C/C++ source
- Optimization: Pragmas based on target hardware
  - Using Metadirective (OpenMP 5.0)

```
#pragma omp metadirective                                          \
    when(device={kind(gpu)}: distribute parallel for collapse(2)) \
    when(device={kind(cpu)}: for)
#pragma omp metadirective                                          \
    when(device={kind(cpu)}: tile sizes(16,32))
for (int i = 0; i < m; ++i)
  #pragma omp metadirective                                        \
      when(device={kind(gpu)}: unroll partial(8))
  for (int j = 0; j < n; ++j)
    body(i,j);
```


Outline

Use Cases

- Target audience: performance specialists
  - *Know your hardware!*
- Workflow
  1. Write application to be correct first
  2. Do some profiling
  3. Apply low-hanging fruits
     - Use vendor-optimized libraries
     - Optimize (e.g. `schedule(static,`*chunk-size*`)`)
     - Add optimization directives: `simd`, `unroll`, `tile`
  4. Extreme optimization
     - Rewrite kernel using target hardware assembly

Compiler Support

- Clang: In progress
  - Tile directive: `https://reviews.llvm.org/D76342`

Possible Extensions for OpenMP 6.0

Additional Transformations

- Loop interchange
- Loop fission/fusion
- Peeling
- Loop unswitching
- Space-filling curves

Auxiliary Transformations

- Nestify
- Rectangify
- Collapse

More Clauses

- Control over remainder loops
- Enable safety checks

Compose Non-Outermost Generated Loops

- Loop identifiers
- `apply` clause

```
#pragma omp tile apply(floor: parallel for) \
                 apply(tile : simd)
for (int i = 0; i < n; ++i)
  body(i);
```

Semantics/Optimization Hints

- Assumptions:
  "ivdep", parallelizable, interchangeable, ...
- Expectations:
  loop trip count, hot/cold/dead, ...

openmp.org OpenMP API specs, forum, reference guides, and more

link.openmp.org/sc20 Videos and PDFs of OpenMP SC'20 presentations

Acknowledgments

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration, in particular its subproject SOLLVE.

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

This material was based in part upon funding from the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357.