Loop interchange

Troels Henriksen
Based on material by Cosmin Oancea

Determining parallel loops from direction vectors

Parallel loop theorem

A loop in a loop nest is parallel if all its directions are either = or there exists an outer loop whose direction is <.

Determining parallel loops from direction vectors

Parallel loop theorem

A loop in a loop nest is parallel if all its directions are either = or there exists an outer loop whose direction is <.

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    S1: A[j][i] = A[j][i]...
```

[=,=]

Determining parallel loops from direction vectors

Parallel loop theorem

A loop in a loop nest is parallel if all its directions are either = or there exists an outer loop whose direction is <.

```
#pragma omp parallel for
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    S1 : A[j][i] = A[j][i]...
```

[=,=]

Determining parallel loops from direction vectors

Parallel loop theorem

A loop in a loop nest is parallel if all its directions are either = or there exists an outer loop whose direction is <.

```
#pragma omp parallel for
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    S1: A[j][i] = A[j][i]...
```

[=,=]

```
for (int i = 1; i < N; i++)
  for (int j = 0; j < N; j++)
    S1: A[i][j] = A[i-1][j+1]...
```

[<,>]

Determining parallel loops from direction vectors

Parallel loop theorem

A loop in a loop nest is parallel if all its directions are either = or there exists an outer loop whose direction is <.

```
#pragma omp parallel for
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    S1 : A[j][i] = A[j][i]...
```

[=,=]

```
for (int i = 1; i < N; i++)
  #pragma omp parallel for
  for (int j = 0; j < N; j++)
    S1 : A[i][j] = A[i-1][j+1]...
```

[<,>]

Next

- Direction vectors are not just useful for figuring out where we can add OpenMP pragmas to get free performance.
- They can also be used to tell us which *loop transformations* are valid.

Interchange

```
for (int j = 0; i < N; i++)
  for (int i = 0; i < N; i++)
    A[i][j] = i*j;
```

$\Rightarrow$

```
for (int i = 0; i < N; i++)
  for (int j = 0; i < N; i++)
    A[i][j] = i*j;
```

- Improves spatial locality in this case.
- Can we argue with direction vectors that this is a valid transformation?

Interchange

```
for (int j = 0; i < N; i++)              for (int i = 0; i < N; i++)
  for (int i = 0; i < N; i++)    ⇒        for (int j = 0; i < N; i++)
    A[i][j] = i*j;                           A[i][j] = i*j;
```

- Improves spatial locality in this case.
- Can we argue with direction vectors that this is a valid transformation?

Validity of loop interchange

A permutation of the loops in a loop nest is legal if permuting the direction matrix in the same way does not result in a > direction as the leftmost non-= direction in a row.

The loop above has direction vectors `[=,=]` for access to `A`, so interchange is allowed.

A useful corollary

Interchanging a parallel loop inwards

It is always safe to interchange a *parallel loop* inwards one step at a time (i.e., if the parallel loop is the k^{th} loop in the nest then one can always interchange it with loop $k+1$, then with loop $k+2$, etc.).

- This means that if somehow know (by eyeballing, or an existing OpenMP pragma we trust) that a loop is parallel, we can interchange it inwards.
- We don't need to calculate the direction vectors.
- **Even if we don't actually want to execute a loop in parallel, the independence of its iterations can still be exploited for locality optimisations via interchange.**

What about loop B?

```
for (int i = 1; i < N; i++)
  for (int j = 1; j < N; j++) {
    S1: A[j][i] = A[j-1][i-1]...
    S2: B[j][i] = B[j-1][i]...
  }
```

$$M = \begin{cases} [<,<] \\ [=,<] \end{cases}$$

What about loop B?

```
for (int i = 1; i < N; i++)
  for (int j = 1; j < N; j++) {
    S1 : A[j][i] = A[j-1][i-1]...
    S2 : B[j][i] = B[j-1][i]...
  }
```

$$M = \begin{cases} [<,<] \\ [=,<] \end{cases}$$

```
for (int j = 1; j < N; j++)
  for (int i = 1; i < N; i++) {
    S1 : A[j][i] = A[j-1][i-1]...
    S2 : B[j][i] = B[j-1][i]...
  }
```

$$M = \begin{cases} [<,<] \\ [<,=] \end{cases}$$

After interchange the (now) innermost loop is parallel!

What about loop B?

```
for (int i = 1; i < N; i++)
  for (int j = 1; j < N; j++) {
    S1: A[j][i] = A[j-1][i-1]...
    S2: B[j][i] = B[j-1][i]...
  }
```

$$M = \begin{cases} [<,<] \\ [=,<] \end{cases}$$

```
for (int j = 1; j < N; j++)
  #pragma omp parallel for
  for (int i = 1; i < N; i++) {
    S1: A[j][i] = A[j-1][i-1]...
    S2: B[j][i] = B[j-1][i]...
  }
```

$$M = \begin{cases} [<,<] \\ [<,=] \end{cases}$$

After interchange the (now) innermost loop is parallel!

What about loop C?

```
for (int i = 1; i < N; i++)
  for (int j = 0; j < N; j++)
    S1: A[i][j] = A[i-1][j+1]...
```

$M = [<, >]$

What about loop C?

```
for (int i = 1; i < N; i++)
  for (int j = 0; j < N; j++)
    S1: A[i][j] = A[i-1][j+1]...
```

$M = [<, >]$

```
for (int j = 0; j < N; j++)
  for (int i = 1; i < N; i++)
    S1: A[i][j] = A[i-1][j+1]...
```

$M = [>, <]$

Interchange is illegal, because direction vectors are not allowed to start with >.

Interchange to increase parallelism

```
#pragma omp parallel for
for (int i = 0; i < n; i++)
  A[i] = f(A[i]);
```

- A common pattern in data analysis is that we have a parallel loop.

Interchange to increase parallelism

```
for (int j = 0; j < m; j++)
  #pragma omp parallel for
  for (int i = 0; i < n; i++)
    A[i] = f(A[i]);
```

- A common pattern in data analysis is that we have a parallel loop.
- This loop is then contained in a sequential *time series loop*.

Interchange to increase parallelism

```
#pragma omp parallel for
for (int l = 0; l < k; l++)
  for (int j = 0; j < m; j++)
    #pragma omp parallel for
    for (int i = 0; i < n; i++)
      A[j][i] = f(A[j][i]);
```

- A common pattern in data analysis is that we have a parallel loop.
- This loop is then contained in a sequential *time series loop*.
- And this whole thing is then applied to multiple sets simultaneously.

Interchange to increase parallelism

```
#pragma omp parallel for
for (int l = 0; l < k; l++)
  for (int j = 0; j < m; j++)
    #pragma omp parallel for
    for (int i = 0; i < n; i++)
      A[j][i] = f(A[j][i]);
```

- A common pattern in data analysis is that we have a parallel loop.
- This loop is then contained in a sequential *time series loop*.
- And this whole thing is then applied to multiple sets simultaneously.
- **But wait**—usually OpenMP will only parallelise the outer `k`-iteration loop.

Interchange will fix this

```
#pragma omp parallel for
for (int l = 0; l < k; l++)
  for (int j = 0; j < m; j++)
    #pragma omp parallel for
    for (int i = 0; i < n; i++)
      A[j][i] = f(A[j][i]);
```

- Since the outermost loop is parallel, we can interchange it *inwards*.

Interchange will fix this

```
for (int j = 0; j < m; j++)
  #pragma omp parallel for
  for (int l = 0; l < k; l++)
    #pragma omp parallel for
    for (int i = 0; i < n; i++)
      A[j][i] = f(A[j][i]);
```

- Since the outermost loop is parallel, we can interchange it *inwards*.
- And then we can collapse the two OpenMP directives.

Interchange will fix this

```
for (int j = 0; j < m; j++)
  #pragma omp parallel for collapse(2)
   for (int l = 0; l < k; l++)
     for (int i = 0; i < n; i++)
       A[j][i] = f(A[j][i]);
```

- Since the outermost loop is parallel, we can interchange it *inwards*.
- And then we can collapse the two OpenMP directives.
- And now we have a parallel loop with `k*n` iterations, which is run `m` times.

Is that better? Depends.

Summary

- Direction vectors...
  - tell us whether a loop is parallel.
  - tell us whether loop interchange is valid.
- Loop interchange...
  - can change the locality of the loop nest (for better or worse).
  - can reveal more possibilities for parallelisation.