Loop Transformations

Sebastian Hack
hack@cs.uni-saarland.de

January 19, 2018

Saarland University, Computer Science

Loop Transformations: Example

matmul.c

Optimization Goals

- Increase locality (caches)
- Facilitate Prefetching (contiguous access patterns)
- Vectorization (SIMD instructions, contiguity, avoid divergence)
- Parallelization (shared and non-shared memory systems)

Loop Unswitching

```
for i = 1 to N
  for j = 1 to M
    if X[i] > 0
      S
    else
      T
```

```
for i = 1 to N
  if X[i] > 0
    for j = 1 to M
      S
  else
    for j = 1 to M
      T
```

- Hoist conditional as far outside as possible
- Overhead of branch "multiplied" by the loop
- Enables other transformations (that require branch-free loop bodies)

Loop Peeling

```
for i = 1 to N
  S
```

```
if N ≥ 1
  S
  for i = 2 to N
    S
```

- Align trip count to a certain number (multiple of N)
- Peeling the exit condition yields a place where loop invariant code can be executed non-redundantly

Index Set Splitting

```
for i = 1 to N
  S
```

```
assert 1 ≤ M < N
for i = 1 to M
  S
for i = M + 1 to N
  S
```

- Create specialized variants for different cases
  e.g. vectorization (aligned and contiguous accesses)
- Can be used to remove conditionals from loops

Loop Unrolling

```
for i = 1 to N
  S
```

```
for (i = 0; i < n; i += U)
  S(i+0)
  S(i+1)
  ...
  S(i+U-1)
for (; i < N; i++)
  S(i)
```

- Create more instruction-level parallelism inside the loop
- Less specualtion on OOO processors, less branching
- Increases pressure on instruction / trace cache (code bloat)

Loop Fusion

```
for i = 1 to N
  S
for i = 1 to N
  T
```

```
for i = 1 to N
  S
  T
```

- Save loop control overhead
- Increase locality if both loops access same data
- Increase instruction-level parallelism
- Not always legal: Dependences must be preserved

Loop Interchange

```
for i = 1 to N
  for j = 1 to M
    S
```

```
for j = 1 to M
  for i = 1 to N
    S
```

- Expose more locality
- Expose parallelism
- Legality: Preserve data dependences

Parallelization / Vectorization

```
for i = 1 to N          parallel for i = 1 to N
    S                       S
```

- Loop must not carry dependence
- Vectorization nowadays uses SIMD code $\rightarrow$ strip mining

Strip Mining

```
for i = 1 to N
  S
```

```
for (i = 0; i < N; i += U)
  for (j = 0; j < U; j++)
    S(i + j)
```

- Simple vectorization can be facilitated by strip mining
- For SIMD instruction sets, set `U` to the vector width
- Vectorize `S` and drop inner loop
- Add Epilogue for $N \% U \neq 0$

Tiling

Original

```
for i = 1 to N
  for j = 1 to M
    S(i, j)
```

Strip-mined

```
for i = 1 to N step S
  for ii = 1 to S
    for j = 1 to M step T
      for jj = 1 to T
        S(i + ii, j + jj)
```

Tiled (stepping loops interchanged to the outside)

```
for i = 1 to N step S
  for j = 1 to M step T
    for ii = 1 to S
      for jj = 1 to T
        S(i + ii, j + jj)
```

- Tiling = strip-mine + interchange
- Increases locality
- Enables distribution to multiple cores