Optimizing loops for the memory hierarchy

A typical loop walks over lots of data (long arrays)

Enclosing loops lead to repeated traversals of same data

Want to restructure loops to revisit data while it's in cheap memory

Example: matrix multiply

```
for i := 1 to N
  for j := 1 to N
    for k := 1 to N
      c[i,j] += a[i,k] * b[k,j]
    end
  end
end
```

`a[i,*]` visited repeatedly, but usually flushed from cache

`b[*,j]` similar, though less extreme

Blocking

A.k.a. **tiling**

Restructure loop so that it revisits a chunk that fits in registers/cache/page before moving on

```
for k' := 1 to N by ChunkSize
  for i := 1 to N
    for j := 1 to N
      for k := k' to k'+ChunkSize
        c[i,j] += a[i,k] * b[j,k]
      end
    end
  end
end
```

Implement transformation with combination of **strip mining** and **loop interchange**

Strip mining

Strip mining: turn single loop into 2 loops

- inner loop up to some chunk size
- outer loop repeats chunks till done

Before:

```
for k := 1 to N
  c[i,j] += a[i,k] * b[k,j]
end
```

After:

```
for k' := 1 to N by ChunkSize
  for k := k' to k'+ChunkSize-1
    c[i,j] += a[i,k] * b[k,j]
  end
end
```


Loop interchange

Apply loop interchange to move outer strip-mined outside of enclosing loop(s)

Before:

```
for i := 1 to N
  for j := 1 to N
    for k' := 1 to N by ChunkSize
      for k := k' to k'+ChunkSize-1
        c[i,j] += a[i,k] * b[j,k]
      end
    end
  end
end
```

After:

```
for k' := 1 to N by ChunkSize
  for i := 1 to N
    for j := 1 to N
      for k := k' to k'+ChunkSize-1
        c[i,j] += a[i,k] * b[j,k]
      end
    end
  end
end
```

Do again

Block the `j` loop, too

```
for k' := 1 to N by ChunkSize
  for j' := 1 to N by ChunkSize
    for i := 1 to N
      for j := j' to j'+ChunkSize-1
        for k := k' to k'+ChunkSize-1
          c[i,j] += a[i,k] * b[j,k]
        end
      end
    end
  end
end
```