Compiler Optimizations for Improving Data Locality

Cache Stuff, ya hear? (also maybe some scheduling things)

2.28.2025

Arjun Bhamra

Contents

Contents (ii)

What to get from this talk

Goals for this talk:

- Get a decent understanding of caches/caching, and why it's needed
- Going over the titular paper
- An intro to polyhedral optimization

1 What’s a cache? Why care?

Problem: Memory is bad

Specifically, at getting things to us fast

Registers are on the order of 10s of cycles, at most

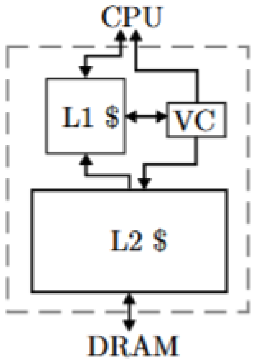
DRAM is on the order of 100s of cycles (ouch)

How do we fix this?

Solution: Make small fast memory

Call it a cache because it costs so much damn money (this is a lie) (also they were originally called buffers because IBM)

Between CPU and Memory, create intermediary layers that we can access which use faster (but costlier) technology to retrieve information



Cache details

- Caches can be fully associative, N-way set associative, or direct mapped (1-way) (Arjun will draw diagrams)
- A cache can be specified with C, B, and S values (maybe get into this)
- Caches typically store data in “blocks”, which are **rows of bytes** (this will be important for later)
- We read data in at the block level (there are hardware optimizations to access specific words first)

What does a compiler consider when optimizing for caches?

Data Locality

The concept of having related data be either physically close by (spatial locality) or be repeatedly accessed (temporal locality)

Examples:

- Loops
- Array Accesses
- The ability to vectorize instructions (SIMD, etc.)

Can you think of how these exploit locality?

2 So there’s this cool paper

Compiler Optimizations for Improving Data Locality

Steve Carr, Kathryn S. McKinley, and Chau-Wen Tseng. 1994. https://doi.org/10.1145/195470.195557

Discusses a variety of potential optimization angles related to memory accesses, including:

- Loop Permutation/Interchange
- Loop Fusion and Fission
- Tiling

Their paper goes over a cost model for determining what potential optimizations may be ideal, using Profile Guided Optimization (PGO)

Fun Note: For some of the above optimizations, knowing cache specifics are required, and PGO can allow us to “reconstruct” an image of a memory heirarchy given some background information!

Loop Permutation/Interchange

Consider the two loop programs below, which one do you think will have better data locality and why?

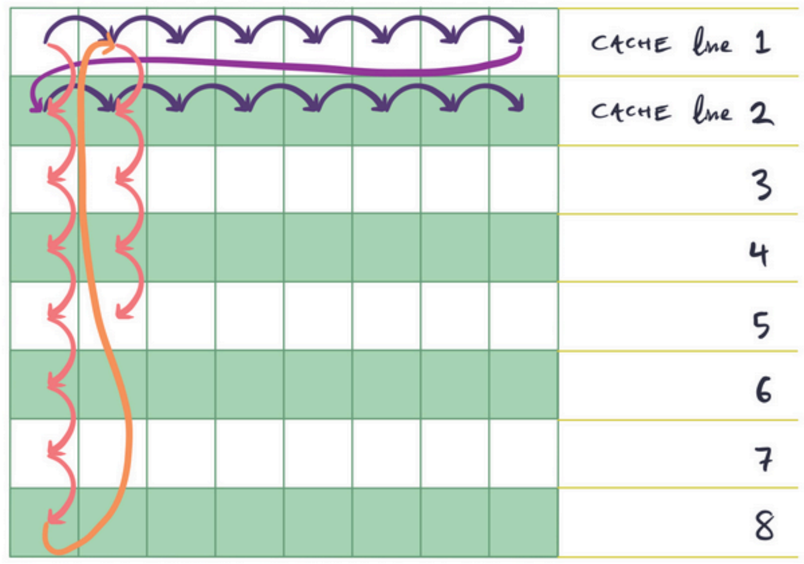
Column Major: (for each column, go through the row)

```
int A[DIM1][DIM2];
for (int iter = 0; iter < iters; iter++)
    for (int j = 0; j < DIM2; j ++)
        for (int i = 0; i < DIM1; i++)
            A[i][j]++;
```

Row Major: (vice versa)

```
int A[DIM1][DIM2];
for (int iter = 0; iter < iters; iter++)
    for (int i = 0; i < DIM1; i++)
        for (int j = 0; j < DIM2; j ++)
            A[i][j]++;
```

Loop Permutation/Interchange (ii)



Loop Fusion

Reduce duplicate work by combining loops! This improves locality because you are reading in blocks of data into the cache less times

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

becomes

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

There is an easy optimization to halve loads here, can you find it?

Loop Fission

If you have unrelated operations in the same loop (such as distinct array accesses), you can separate these into separate loops to reduce memory pressure

```
int A[MAXN][MAXN], B[MAXN][MAXN];
for (int i = 0; i < MAXN; i++) {
    for (int j=0; j<MAXN; j+=2) {
        A[i][j] ++;
        B[j][i] ++;
    }
}
```

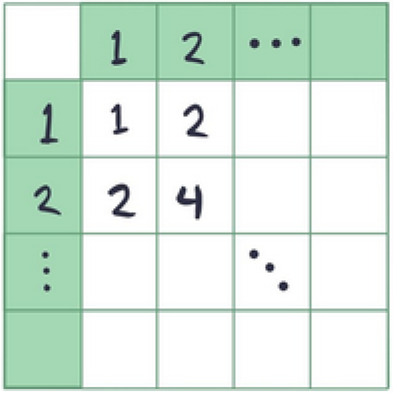
^ This is bad.

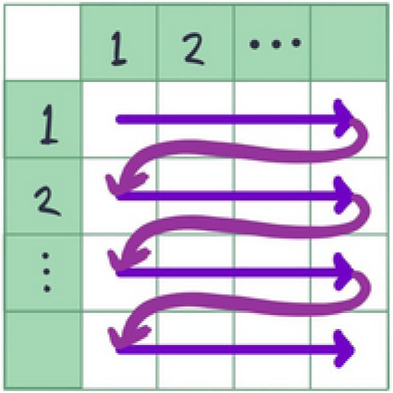
Vectorization

- Flynn's Taxonomy (Brief overview)
- Vectorization is using SIMD instructions that have hardware support, instructions that can work across multiple inputs in parallel.

Tiling

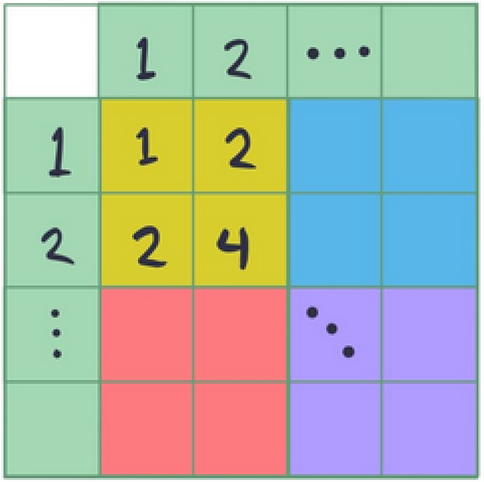
When we have specific access patterns that result in more misses than we want, we can sometimes subdivide our accesses into groupings that match these patterns

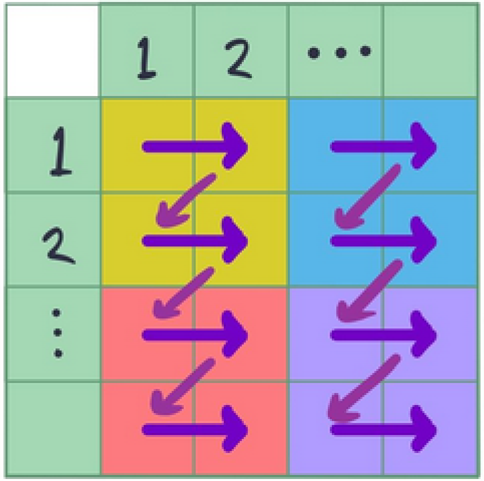




^ This is bad.

Tiling (ii)





^ This is good.

Important note

- A lot of these optimizations have examples that can easily be displayed in 2D, but just remember that they work in higher dimensions as well.
- In fact, for multi-level caches, tiling can be particularly poweful.

3 Polyhedral Loop Optimization

What on god's green earth is going on

- I underestimated how hard this would be to explain, and so I will be doing the best I can given two (2) days of research into this
- Polyhedral optimization (aka, using the polytope model) derives its name from convex optimization and intuition about convex sets, as well as, by natural extension, (integer) linear programming.

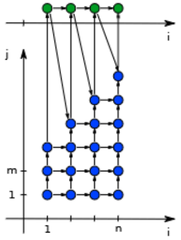
The Process:

1. We consider our iteration space as a series of points on an N-dimensional grid (draw this, vaguely)
2. We can then draw a dependency graph based on recurrence relations in our access patterns (what is dependency? Also, draw this)
3. We apply an affine (linear + const) transformation to our loop structure and schedule

Note: Not all loops are affine, but around 99% of practical looping is

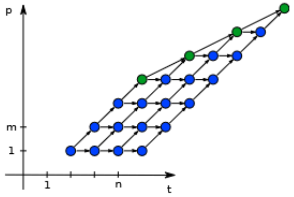
4. We do code generation on our transformed structure, noting its (hopefully new) parallelization (crucially, this can be done via dependency reduction)

Examples from the Polly Slides



Original Code

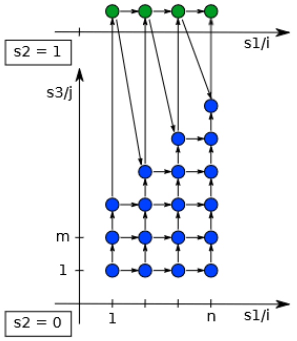
```
for (i = 1; i <= n; i++) {

  for (j = 1; j <= i + m; j++)
    A[i][j] = A[i-1][j] + A[i][j-1]

  A[i][i+m+1] = A[i-1][i+m] + A[i][i+m]
}
```

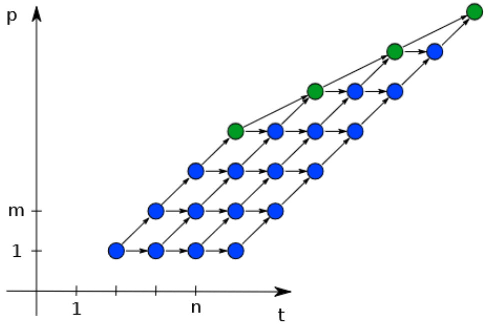


Transformed Code

```
parfor (p = 1; p <= m+n+1; p++) {
  if (p >= m+2)
    A[p-m-1][p] = A[p-m-2][p-1]
  for (t = max(p+1, 2*p-m); t <= p+n; t++)
    A[-p+t][p] = A[-p+t-1][p] + A[-p+t][p-1]
}
```

Examples from the Polly Slides (ii)





Original Schedules

$s_1 = i$	$s_1 = i$
$s_2 = 0$	$s_2 = 1$
$s_3 = j$	

Transformed Schedules

$p = j$	$p = i + m + 1$
$t = i + j$	$t = 2i + m + 1$

Autoscheduling

Some very cool industry standard tools to check out that do “autoscheduling” of loop structures for a variety of applications

- Tiramisu: modern polyhedral compiler)
- Halide: image processing parallelization framework
- Apache TVM: DL framework for lower level optimization and scheduling (closest to today’s talk, in some sense)

4 The End

References

Major credit due to the below Cornell blogpost, which provided a great foundation (and some graphics) for the presentation, as well as the Polly talk (and slides) for their help with explaining polyhedral/polytope optimization.

1. https://www.cs.cornell.edu/courses/cs6120/2019fa/blog/loops/
2. https://dl.acm.org/doi/pdf/10.1145/195470.195557
3. https://www.youtube.com/watch?v=WwfZkQEuwEE&ab_channel=LLVM (polyhedral talk, has slides in desc)
4. http://polyhedral.info/
5. https://www.infosun.fim.uni-passau.de/cl/loopo/doc/loopo_doc/node3.html
6. https://www.infosun.fim.uni-passau.de/cl/papers/concur93c.pdf (for the mathematical formalism, which I do not yet understand)
7. https://tvm.apache.org/docs//v0.8.0/how_to/optimize_operators/opt_gemm.html
8. https://halide-lang.org/papers/halide_autoscheduler_2019.pdf