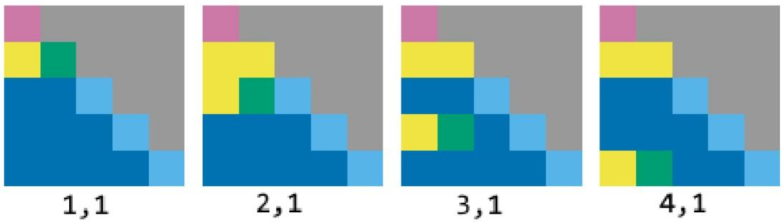
<Cholesky_kernel>

- Andrea Tirelli
- Amanjot Singh
- Danilo Bellocchio
- Domenico Rossini

CHOLESKY DECOMPOSITION (or Cholesky Factorization)

```
for (i = 0; i < _PB_N; ++i)
{
  x = A[i][i];
  for (j = 0; j <= i - 1; ++j)
    x = x - A[i][j] * A[i][j];
  p[i] = 1.0 / sqrt(x);
  for (j = i + 1; j < _PB_N; ++j)
  {
    x = A[i][j];
    for (k = 0; k <= i - 1; ++k)
      x = x - A[j][k] * A[i][k];
    A[j][i] = x * p[i];
  }
}
```

es.



Example of data dependencies of a single column on a 5x5 matrix. Green elements are the active elements for each iteration, which direclty depend on the yellow elements, indirectly on the pink ones, and not dependent of the blue ones.

Sequential Benchmarking

```
Performance counter stats for './cholesky_acc':

     361.898.961      cache-references
      24.269.269      cache-misses              #    6,706 % of all cache refs
     922.929.941      cycles
   1.087.772.189      instructions              #    1,18  insn per cycle
 <not supported>      branches
             581      faults
               0      migrations

     0,629194695 seconds time elapsed
```

Possible solution:

- **Vectorize** the problem
- Manipulate the memory **init** and **access**

PARALLELIZED VERSION 1

```c
static void kernel_cholesky(int n,
                            DATA_TYPE POLYBENCH_1D(p, N, n),
                            DATA_TYPE POLYBENCH_2D(A, N, N, n, n))
{
    int i, j, k;

    DATA_TYPE x;
    for (i = 0; i < _PB_N; ++i)
    {
        x = A[i][i];

        #pragma omp parallel for reduction(-:x)
        for (j = 0; j <= i - 1; ++j)
            x = x - A[i][j] * A[i][j];

        p[i] = 1.0 / sqrt(x);

        #pragma omp parallel for private(x, k)
        for (j = i + 1; j < _PB_N; ++j)
        {
            x = A[i][j];

            #pragma omp simd reduction(-:x)
            for (k = 0; k <= i - 1; ++k)
                x = x - A[j][k] * A[i][k];

            A[j][i] = x * p[i];
        }
    }
}
```

Exec time: ~14s

PARALLELIZED VERSION 2

```c
static void opt_kernel_cholesky(int n,
                                DATA_TYPE POLYBENCH_1D(p, N, n),
                                DATA_TYPE POLYBENCH_2D(A, N, N, n, n))
{
  int i, j, k;

  DATA_TYPE x;
    for (i = 0; i < _PB_N; ++i) {
        p[i] = 1 / sqrt(A[i][i] - p[i]);

        #pragma omp parallel for private(j, k, x)
        for (j = i + 1; j < _PB_N; j++) {
            x = A[i][j];

            #pragma omp simd reduction(-:x)
            for (k = 0; k <= i - 1; ++k)
                x = x - A[j][k] * A[i][k];

            A[j][i] = x * p[i];
            p[j] += A[j][i] * A[j][i];
        }
    }
}
```

Exec time: ~13s

AMDAHL'S LAW COMPARISON

Benchmark :

Dataset = EXTRALARGE

Sequential execution time = 32 sec

Amdahl's Law :

$$speedup = \frac{1}{(1-p) + \frac{p}{n}}$$

Amdahl's Law -> n = 4 p = 90%

Expected SpeedUp = 3x

Parallelized Version :

Parallel Execution time = 13 sec

Achieved SpeedUp = 2,46x

CONCLUSIONS

Let's analyze the results obtained in detail :