Single-Core Performance Optimization: Improved Memory Efficiency across Compilers

codee

`This article addresses single-core performance optimizations that enable an efficient usage of the memory available in the hardware. We use Codee to pinpoint opportunities in the code to benefit from loop interchange, which enables sequential memory accesses and favors vectorization.`

Why is Loop Interchange important?

Loop interchange is a performance optimization technique that is used to improve the loop's memory access pattern and potentially enable vectorization. Loop interchange if applied correctly can yield a huge performance improvement.

Loop interchange is applicable to loop nests, which consist of two or more nested loops. After loop interchange, two nested loops are swapped so that the inner loop becomes the outer and the outer becomes the inner.

Automation of Loop Interchange

Loop interchange is a sophisticated programming technique that is not fully automated in current tools. Codee brings novel innovations in this space through its built-in support to detect Loop Interchange opportunities in perfectly-nested and non-perfectly nested loops (including PWR039, PWR042, PWR043 from the open catalog of performance optimization best practices). Consider the matrix-matrix multiplication code (MATMUL) shown below:

```
void matmul(int n, double *A, double *B, double *C) {
  for (int i = 0; i < n; ++i)
  {
    for (int j = 0; j < n; ++j)
    {
      double c = 0.0;
      for (int k = 0; k < n; ++k)
      {
        c += A[i * n + k] * B[k * n + j];
      }
      C[i * n + j] += c;
    }
  }
}
```

Running Codee for this code, it reports PWR043 "consider loop interchange by replacing the scalar reduction value". After rewriting the source code according to best practices, the following loop is created:

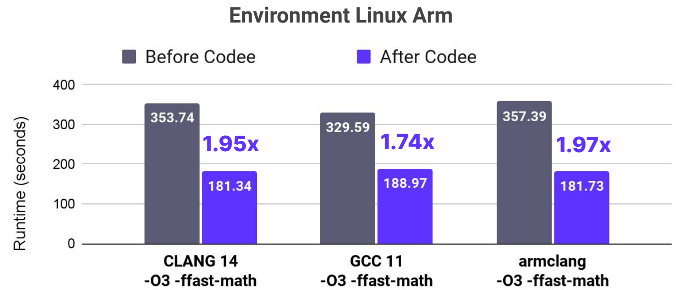
```
void matmul(int n, double *A, double *B, double *C) {
  for (int i = 0; i < n; ++i)
  {
    for (int k = 0; k < n; ++k)
    {
      for (int j = 0; j < n; ++j)
      {
        C[i * n +j] += A[i * n + k] * B[k * n + j];
      }
    }
  }
}
```

Note that after loop interchange, the ordering of the nested loops changes from IJK to IKJ. This favours sequential memory accesses in the innermost loop, which also enables its vectorization in this source code snippet.

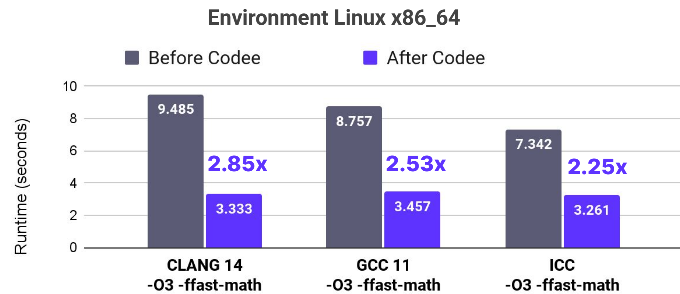
Check out performance-demos GitHub repository and follow the instructions to reproduce the speed improvements obtainable by using Codee.

Performance Evaluation

The performance gain achieved through loop interchange in the MATMUL code is **2x-3x** on **Arm and x86 processors**.



Codee brings 2x faster code on Arm environments through loop interchange and vectorization



Codee brings 3x faster code on x86 environments through loop interchange and vectorization

From the technology perspective, Codee automates loop interchange and enables the efficient vectorization of the innermost loop. Note that GNU, LLVM and Intel compilers do not apply loop interchange in the maximum performance optimization level setup.

Innermost loop matmul_naive:35	GCC 11 -O3 -ffast-math	CLANG 14 -O3 -ffast-math	armclang -O3 -ffast-math	ICC -O3 -ffast-math	Codee
Loop interchange					YES
Loop vectorization	YES	NO (cost model)	NO (cost model)	YES	YES
Loop peeling				YES	
Loop interleaving		NO (cost model)	NO (cost model)		
Loop turned into non-loop	YES				

What to expect from Codee in the future?

We are continuously working on improving the capabilities of Codee in all aspects of software performance. We are working on the automation of more single-core performance optimizations, including loop interchange and similar techniques. Stay tuned by subscribing to Codee newsletter!

Open catalog of performance optimization rules (https://www.codee.com/knowledge/).
Codee performance demos (https://github.com/codee-com/performance-demos).