OpenMP and Cache Optimization with Tiling

M1 CSMI

January 26, 2026

What is OpenMP?

- API for shared memory multiprocessing
- Compiler directives + runtime routines
- Key features:
    - Parallel regions
    - Work sharing constructs
    - Synchronization
    - Memory model
- Supported by most modern C/C++/Fortran compilers

Basic OpenMP Syntax

```c
#include <omp.h>

int main() {
    #pragma omp parallel
    {
        int tid = omp_get_thread_num();
        printf("Hello␣from␣thread␣%d\n", tid);
    }
    return 0;
}
```

Compiling and Running

- Compile with OpenMP support:

```
gcc -fopenmp program.c -o program
```

- Control thread count:

```
export OMP_NUM_THREADS=4
./program
```

- Common environment variables:
  - OMP_NUM_THREADS
  - OMP_SCHEDULE
  - OMP_DYNAMIC

Loop Parallelism with OpenMP

```
#pragma omp parallel for reduction(+:sum)
for (int i = 0; i < N; i++) {
    sum += compute(i);
}

// Matrix multiplication example
#pragma omp parallel for collapse(2)
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        // ... computation ...
    }
}
```

Cache Awareness and Tiling

- Cache hierarchy (L1, L2, L3) has limited size
- Temporal vs spatial locality
- Tiling benefits:
  - Improve data reuse
  - Reduce cache misses
  - Better memory access patterns
- Block size should match cache size

Tiling Example (Matrix Multiplication)

```
// Original nested loops
for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        for (int k = 0; k < N; k++)
            C[i][j] += A[i][k] * B[k][j];

// Tiled version
const int block_size = 32;
for (int ii = 0; ii < N; ii += block_size)
    for (int jj = 0; jj < N; jj += block_size)
        for (int kk = 0; kk < N; kk += block_size)
            for (int i = ii; i < ii+block_size; i++)
                for (int j = jj; j < jj+block_size; j++)
                    for (int k = kk; k < kk+block_size;
                        C[i][j] += A[i][k] * B[k][j];
```

Tiling with OpenMP

```
#pragma omp parallel for collapse(2)
for (int ii = 0; ii < N; ii += block_size)
    for (int jj = 0; jj < N; jj += block_size)
        for (int kk = 0; kk < N; kk += block_size)
            // Tile computation (sequential within tile)
            for (int i = ii; i < ii+block_size; i++)
                for (int j = jj; j < jj+block_size; j++)
                    for (int k = kk; k < kk+block_size;
                        C[i][j] += A[i][k] * B[k][j];
```

- Parallelize outer loops with OpenMP
- Maintain cache-friendly inner loops
- Choose block size carefully (experiment!)

Other Optimization Tips

- Minimize false sharing
  - Pad critical data structures
  - Use thread-local storage
- Balance workload
  - Use schedule(dynamic) for uneven workloads
- Avoid over-parallelization
- Use vectorization with SIMD
- Profile with tools:
  - perf, Intel VTune, gprof

Case Study and Results

- Matrix multiplication (4096x4096)
- Hardware: 8-core CPU, 32KB L1, 256KB L2, 8MB L3

Version	Time (s)	Speedup
Sequential	152.3	1.0x
OpenMP (8 threads)	24.7	6.2x
Tiled (block=32)	89.1	1.7x
OpenMP + Tiling	14.2	10.7x

- Best results combine parallelism and cache optimization

Conclusion

- OpenMP provides simple pragma-based parallelism
- Tiling improves cache utilization
- Combined approach gives best results
- Remember:
  - Profile before optimizing
  - Test different block sizes
  - Consider memory hierarchy in design
- Further reading:
  - OpenMP specification (4.5+)
  - Computer Architecture: A Quantitative Approach