Performance Engineering of Software Systems — October 27, 2021
Massachusetts Institute of Technology — 6.172
Saman Amarasinghe, Charles E. Leiserson, and Jonathan Ragan-Kelley — Handout 13

Homework 8: Cache-Oblivious Algorithms

Due: 11:59 P.M. (ET) on Tuesday, Nov 2, 2021

Last Updated: October 27, 2021

Contents

1 Getting started

Please answer the recitation Checkoff Item and ask your TA for a checkoff. Then, answer the writeup questions in this handout and submit an *individual* writeup on Gradescope.

For more information on cache-oblivious algorithms, see the following paper: `https://doi.org/10.1145/2071379.2071383`.

For this homework, assume that all matrices are stored in row-major layout.

2 Cache complexity of matrix multiplication

During Lecture 15 we discussed the cache complexity of $n \times n$ matrix multiplication, under the tall cache assumption. Let $\mathcal{M}$ be the cache size and $\mathcal{B}$ be the cache line size. For the naive approach, there were two cases: (i) if $n > \mathcal{M}/\mathcal{B}$, then $\Theta(n^3)$ cache misses occur; and (ii) if $\mathcal{M}^{1/2} < n \leq \mathcal{M}/\mathcal{B}$, then $\Theta(n^3/\mathcal{B})$ cache misses occur. For the blocking approach, with block size $s < \mathcal{M}^{1/2}$, the number of cache misses that occur is $\Theta(n^3/\mathcal{B}\mathcal{M}^{1/2})$. The cache-oblivious approach achieves the same complexity as the blocking approach without the need of the voodoo parameter s.

Checkoff Item 1: Assume we want to multiply two rectangular matrices with sizes $m \times n$ and $n \times r$. Given the same tall cache assumption, analyze the complexity for one of the following three options: the two cases for the naive approach, $n > \mathcal{M}/\mathcal{B}$ and $\mathcal{M}/r < n < \mathcal{M}/\mathcal{B}$; the blocking approach; and the cache-oblivious approach. You may pick whichever approach you want to analyze.

3 Tableau construction

Consider the tableau-construction problem from the Lecture 9 Addendum. The problem involves filling an $N \times N$ tableau, where each entry of the tableau is calculated as a function of some of its neighbors. Specifically, consider that the (i, j)-th element of the tableau is filled using an equation of the form

$$A[i][j] = f\big(A[i-1][j-1],\, A[i][j-1],\, A[i-1][j]\big),$$

where f is an arbitrary function.

3.1 Iterative formulation

Consider the simple iterative loop in the following code snippet for filling a tableau:

```
#define A(i, j) A[N + (i) - (j) - 1]

void tableau(double *A, size_t N) {
  for (size_t i = 1; i < N; i++) {
    for (size_t j = 1; j < N; j++) {
      A(i, j) = f(A(i-1, j-1), A(i, j-1), A(i-1, j));
    }
  }
}
```

In this problem, we are only interested in computing the final value of the tableau, stored in `A(N-1,N-1)`, hence we really only need to store $2N - 1$ elements during computation. The algorithm declares `A` as an array of size $2N - 1$.

The algorithm initializes the first row and column of the tableau and then invokes the `tableau()` function as shown in the code snippet below:

```
for (size_t i = 0; i < N; i++) {
  A(i, 0) = INIT_VAL;
}
for (size_t j = 0; j < N; j++) {
  A(0, j) = INIT_VAL;
}
tableau(A, N);
res = A(N - 1, N - 1);
```

Write-up 1: Explain why $2N - 1$ space is sufficient and how the `tableau()` function utilizes the $2N - 1$ space.

Recall the tall cache assumption, which states that $\mathcal{B}^2 < c\mathcal{M}$, where $\mathcal{B}$ is the size of the cache line, $\mathcal{M}$ is the size of the cache, and $c \leq 1$ is a constant.

Write-up 2: Assuming that the cache is tall and uses an optimal replacement strategy, give a tight upper bound on the cache complexity $Q(n)$ for each of the following cases using O-notation:

1. $n \geq \alpha\mathcal{M}$,
2. $n < \alpha\mathcal{M}$,

where $\alpha \leq 1$ is a sufficiently small constant.

3.2 Recursive formulation

Now consider the recursive tableau implementation shown in the following code snippet:

```
#define A(i, j) A[N + (i) - (j) - 1]

void recursive_tableau(double *A, size_t rbegin, size_t rend, size_t cbegin,
                       size_t cend) {
  if (rend-rbegin == 1 && cend-cbegin == 1) {
    size_t i = rbegin, j = cbegin;
    A(i, j) = f(A(i-1, j-1), A(i, j-1), A(i-1, j));
  } else {
    size_t rmid = rend-rbegin > 1 ? (rbegin + (rend-rbegin) / 2) : rend;
    size_t cmid = cend-cbegin > 1 ? (cbegin + (cend-cbegin) / 2) : cend;
    recursive_tableau(A, rbegin, rmid, cbegin, cmid);
    if (cend > cmid)
      recursive_tableau(A, rbegin, rmid, cmid, cend);
    if (rend > rmid)
      recursive_tableau(A, rmid, rend, cbegin, cmid);
    if (rend > rmid && cend > cmid)
      recursive_tableau(A, rmid, rend, cmid, cend);
  }
}
```

This algorithm also stores only $2N-1$ elements during the computation. The algorithm initializes `A` and invokes the `recursive_tableau()` function similarly to the iterative algorithm, as shown below:

```
for (size_t i = 0; i < N; i++) {
  A(i, 0) = INIT_VAL;
}
for (size_t j = 0; j < N; j++) {
  A(0, j) = INIT_VAL;
}
if (N > 1) {
  recursive_tableau(A, 1, N, 1, N);
}
res = A(N-1, N-1);
```

This recursive algorithm divides the tableau into four quadrants to compute. As shown in the Tableau Construction addendum, slides 3–5, after the first quadrant is computed, we can then compute the second and third quadrants in parallel. Parallelizing this way results in $\Theta(n^2)$ work, $\Theta(n^{\lg 3})$ span, and $\Theta(n^{2-\lg 3})$ parallelism. We also show in slides 7–9 a more parallel construction that divides the tableau 9 ways.

Write-up 3: Derive the general formula for work and span, assuming a k^2-way tableau construction (i.e., the tableau is divided up into k^2 pieces of size $n/k \times n/k$).

Write-up 4: Answer the following, assuming that the cache is tall and uses an optimal replacement strategy.

1. Show the recurrence relation for the cache complexity $Q(n)$ using the 4-way construction of the `recursive_tableau()` function.
2. Draw the recursion tree and label the internal nodes and leaves with their cache complexity $Q(n)$. What's the height of the recursion tree?
3. How many leaves are in the recursion tree?
4. Using the recursion tree and the recurrence relation, derive a simplified expression for $Q(n)$.

Write-up 5: Assume, as usual, that the cache is tall and uses an optimal replacement strategy. Assuming a k^2-way tableau construction, show that if we are "unlucky," meaning that the size of a subpiece is just slightly above the cache size, then we have $Q(n) = \Theta(n^2 k/\mathcal{M}\mathcal{B})$. Also show that if we are lucky and this situation does not arise, then we have $Q(n) = \Theta(n^2/\mathcal{M}\mathcal{B})$.