

Lecture 9

Performance programming, cont'd

Coding: style and idioms

Experimental design

Announcements

- Midterm next Thursday
- No Quiz
- No office hours next Thursday

Blocked Matrix Multiplication

for i = 0 to N-1

for j = 0 to N-1

// load each block C[i,j] into cache, once : n^2

// B = n/N = cache line size

for k = 0 to N-1

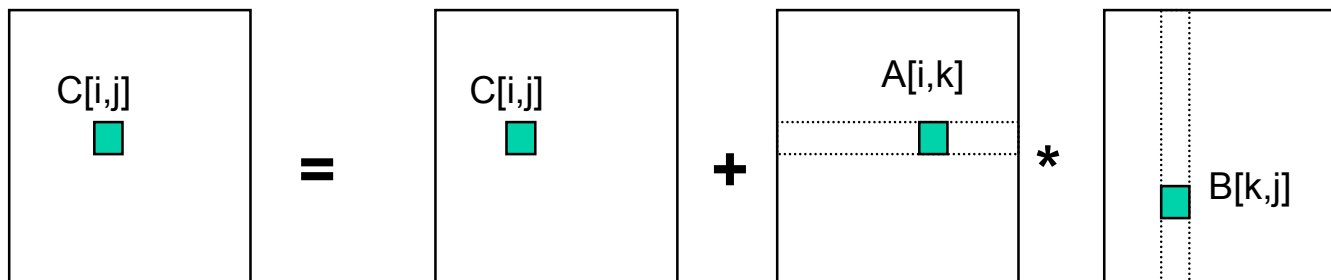
// load each block A[i,k] and B[k,j] N^3 times

// = $2N^3 \times (n/N)^2 = 2Nn^2$

C[i,j] += A[i,k] * B[k,j] // do the matrix multiply

// write each block C[i,j] once : n^2

Total: $(2*N+2)*n^2$

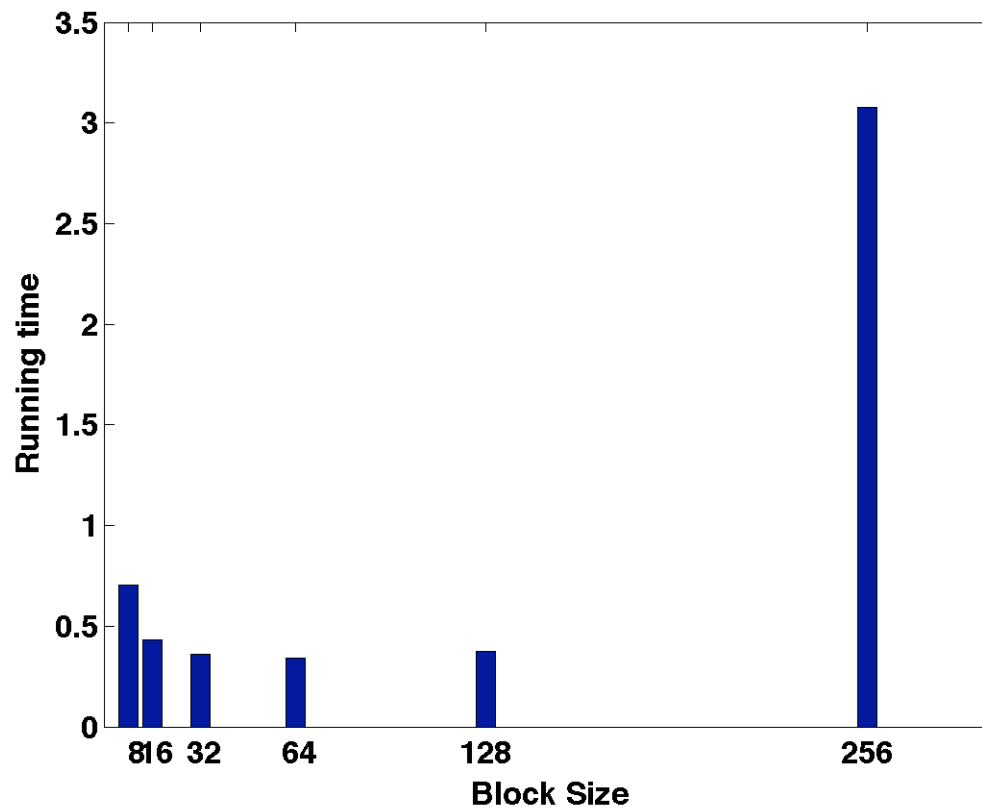


The results

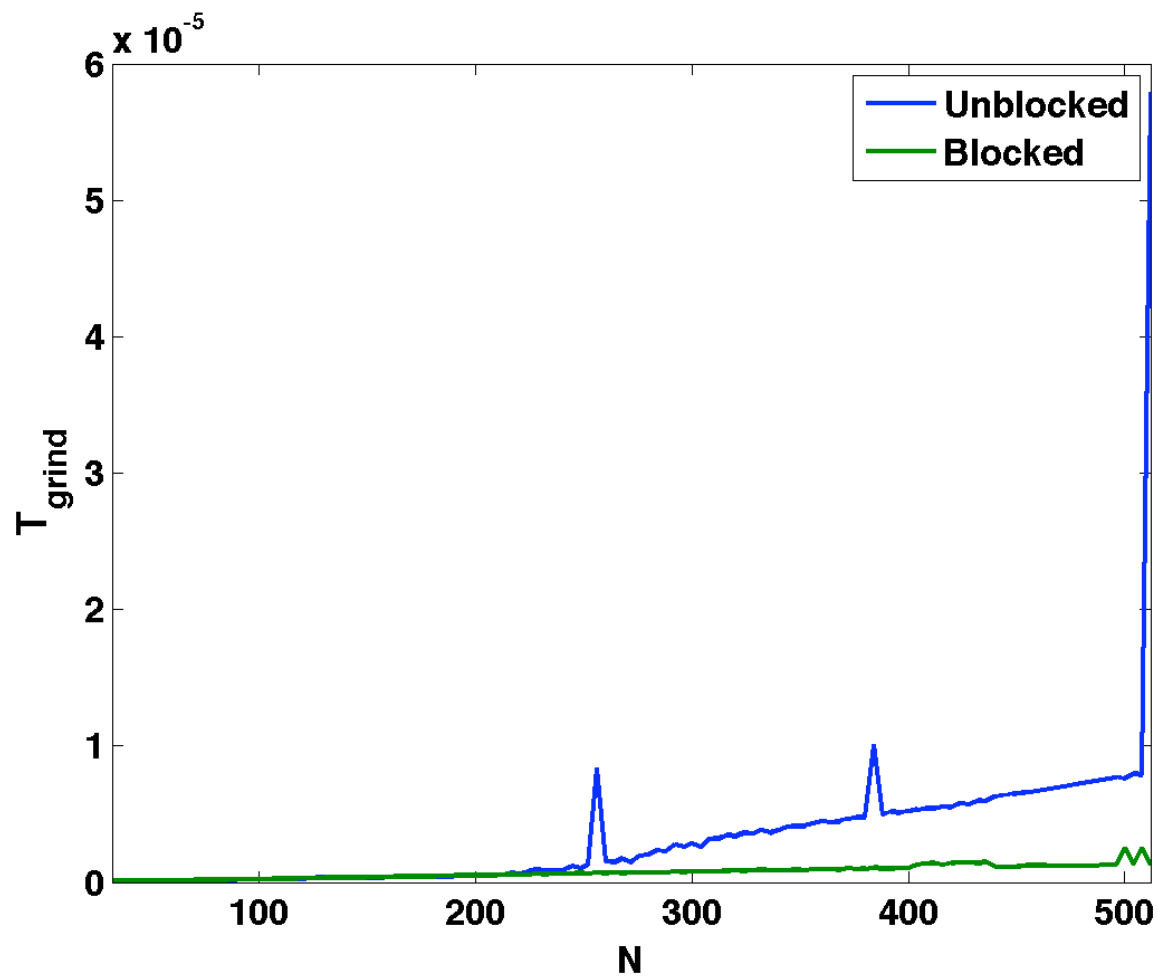
N,B	Unblocked Time (sec) (grind time in microsec)	Blocked Time
256, 64	0.55 (0.033)	0.048 (.0029)
512,128	15 (0.11)	0.37 (.0028)

Optimizing the block size

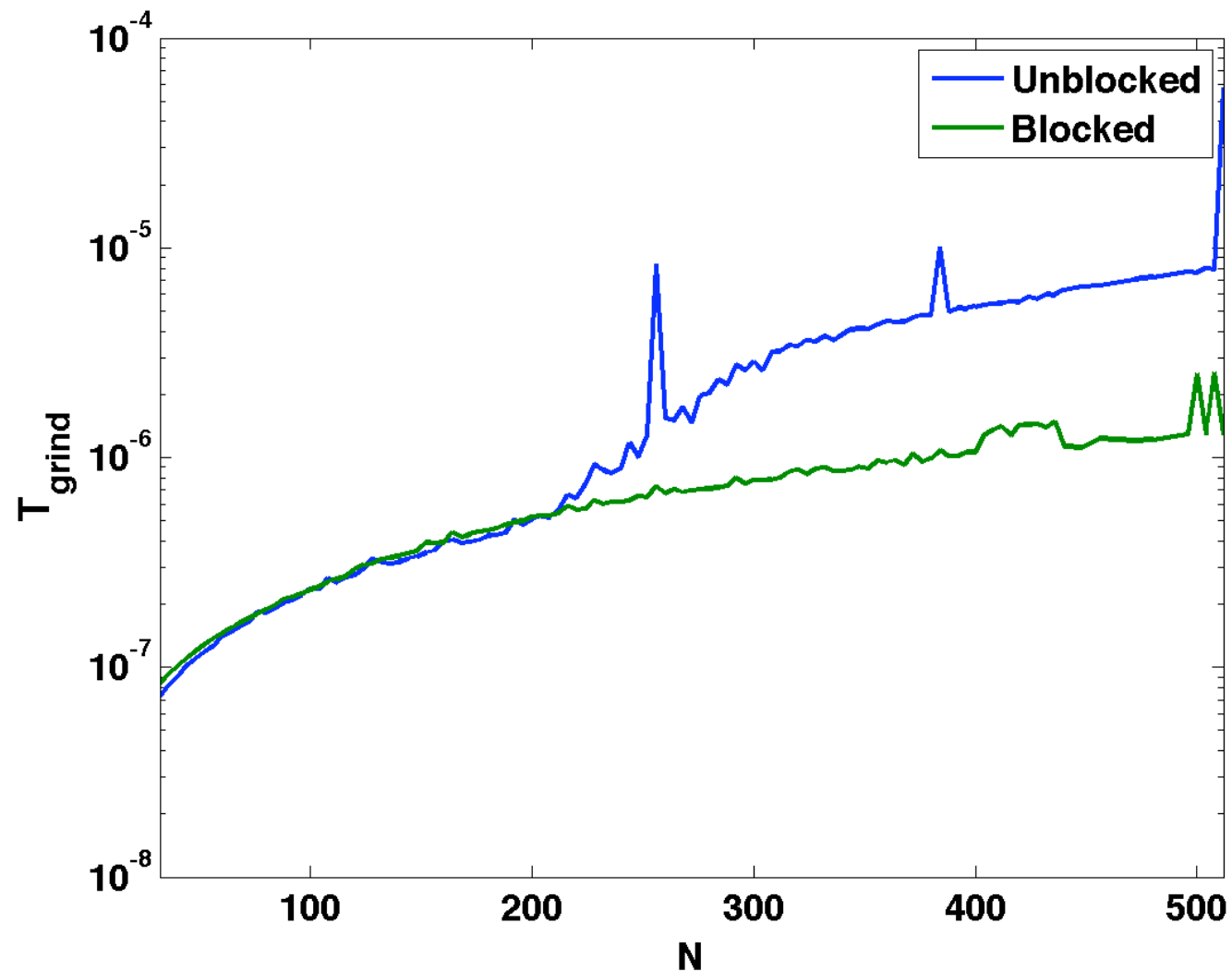
N=512



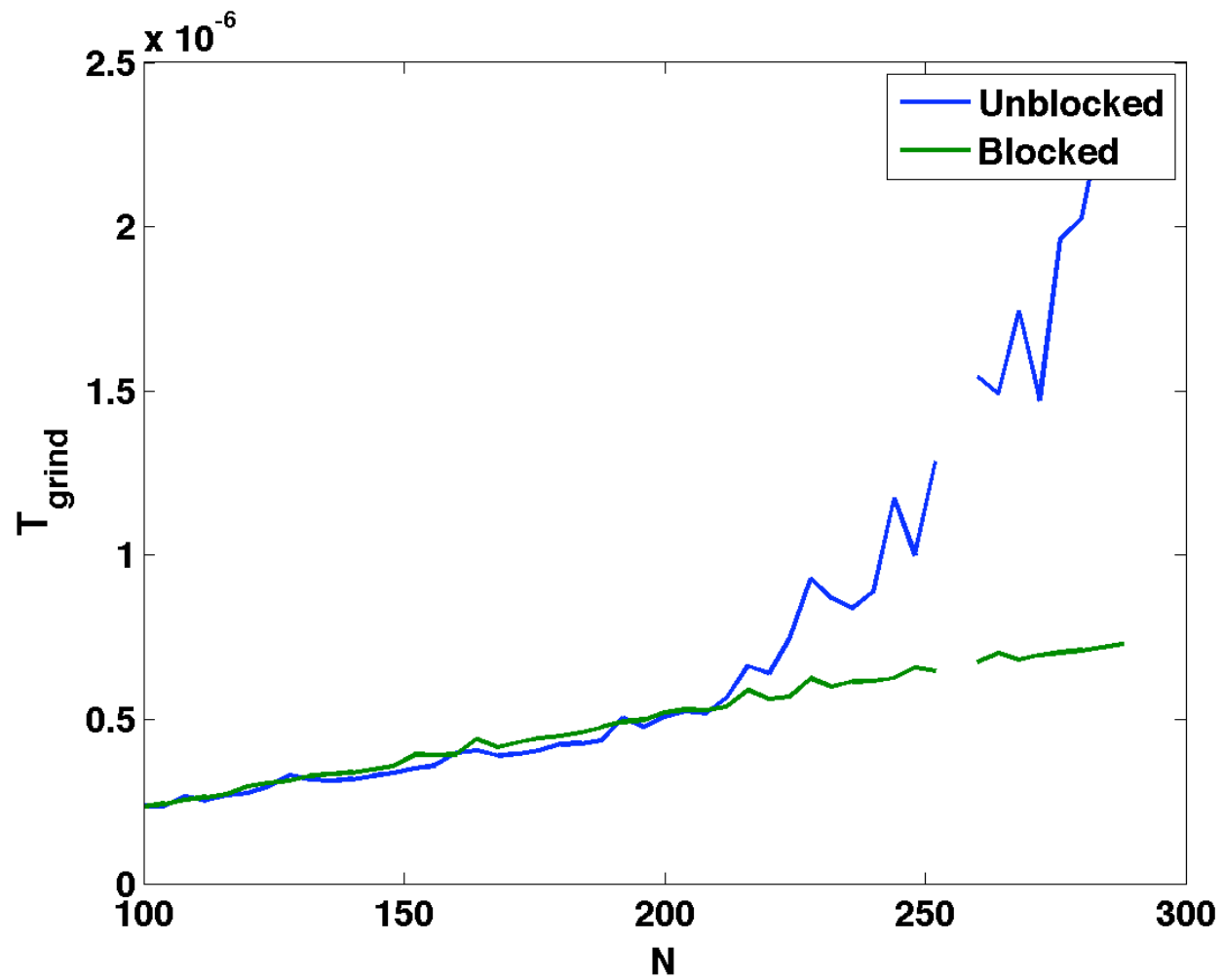
Comparison



Log Y scale improves readability



Crossover



Coding

```
double **A=NULL,**B=NULL, **C=NULL;

assert(A = (double**)
        malloc(sizeof(double*)*n + sizeof(double)*n*n));
for(i=0;i<n;i++)
    A[i] = (double*)(A+n) + i*n;

/* Generates a Hilbert Matrix  $H(i,j) = 1/(i+j+1)$ 
   It's easy to check if the multiplication is correct;
    $(H * H) [i,j] = \sum (k) \{ 1.0/((i+k+1)*(k+j+1)) \}$ 
   */
```

Timing the computation

```
hrtime_t time_un = -gethrtime(); // Start the timer
for (r=0; r<nreps; r++){
    for (i=0; i<n; i++)
        for (j=0; j<n; j++){
            sum = 0;
            for (k=0; k<n; k++)
                sum+= A[i][k] * B[k][j];
            C[i][j] = sum;
        }
}
time_un += gethrtime(); // Stop the timer
```

Timer

```
const double micro = 1.0e-6;
double gethrtime()
{
    struct timeval TV;    struct timezone TZ;

    int RC = gettimeofday(&TV,&TZ);

    if (RC == -1){
        printf("Error in call to gettimeofday\n");
        return(-1);
    }
    return ( ((double)TV.tv_sec ) +
             micro * ((double) TV.tv_usec));
}
```

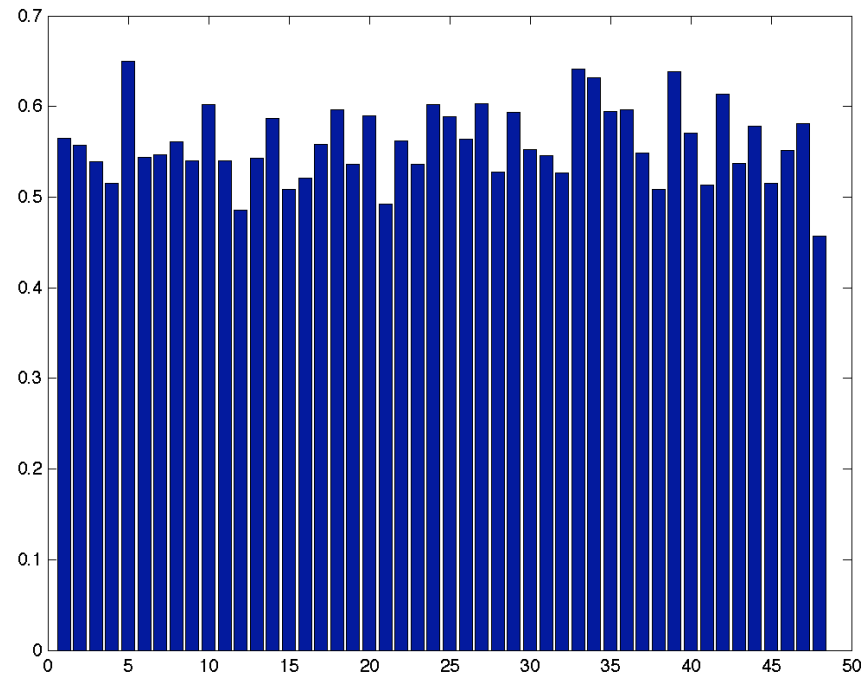
Verification

```
for ( i=0; i<n; i++ ) {  
    double A_exact = 0.0;  
    // Exact value is Sum(k) { 1.0/((i+k+1)*(k+j+1)) }  
    for (k=0;k<n; k++)  
        A_exact += 1.0/((double) (i+k+1)*(k+j+1));  
  
    double diff = fabs( A[i][j] - A_exact);  
    // Avoid roundoff problems, epsilon = 1.0e-8;  
    if ( diff > epsilon ) {  
        nerror++;  
        error = Max(error,delta);  
    }  
}
```

Experimentation

- Make several runs, take the minimum time
- Unblocked time ranges from 0.457 to 0.650
- Why doesn't varying the block size make much of a difference?

B	Time (sec)
16	0.0543
32	0.0465
64	0.0437
128	0.0488



More on blocked algorithms

- Data in the sub-blocks are contiguous within rows only
- We may incur conflict cache misses
- Since re-use is so high... we copy the subblocks into contiguous memory before passing to our matrix multiply routine

“The Cache Performance and Optimizations of Blocked Algorithms,”
ASPLOS IV, 1991

<http://www-suif.stanford.edu/papers/lam91.ps>

