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 :
    // $B = n/N = \text{cache line size}$
    for $k = 0$ to $N-1$
      // load each block $A[i,k]$ and $B[k,j]$ $N^3$ times
      // $= 2N^3 \times (n/N)^2 = 2Nn^2$
      $C[i,j] += A[i,k] \times B[k,j]$ // do the matrix multiply
      // write each block $C[i,j]$ once :
      $n^2$

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

<table>
<thead>
<tr>
<th>N,B</th>
<th>Unblocked Time (sec) (grind time in microsec)</th>
<th>Blocked Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>256, 64</td>
<td>0.55 (0.033)</td>
<td>0.048 (.0029)</td>
</tr>
<tr>
<td>512, 128</td>
<td>15 (0.11)</td>
<td>0.37 (.0028)</td>
</tr>
</tbody>
</table>
Optimizing the block size

N=512

![Bar chart showing running time for different block sizes. The x-axis represents block size (816, 32, 64, 128, 256) and the y-axis represents running time (0 to 3.5). The chart indicates that 256 has the highest running time.](chart.png)
Comparison

![Comparison Diagram]

The diagram illustrates the comparison between Unblocked and Blocked conditions. The x-axis represents the value of $N$, and the y-axis represents $T_{\text{grind}}$, scaled by $10^{-5}$. The blue line represents Unblocked conditions, while the green line represents Blocked conditions.
Log Y scale improves readability
Crossover

![Graph showing the comparison between Unblocked and Blocked T_{\text{grind}} values over N, with notable divergence at higher N values.](image-url)
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] = Σ(k) { 1.0/((i+k+1)*(k+j+1)) } */
Timing the computation

```c
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?

<table>
<thead>
<tr>
<th>B</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0.0543</td>
</tr>
<tr>
<td>32</td>
<td>0.0465</td>
</tr>
<tr>
<td>64</td>
<td>0.0437</td>
</tr>
<tr>
<td>128</td>
<td>0.0488</td>
</tr>
</tbody>
</table>
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


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

![Miss rate vs. Blocking factor graph]