Designing and building applications for extreme scale systems
CS598 SP2016
HW4: Matrix-matrix multiply revisited

Objectives

1. See the impact of spatial and temporal locality on performance
2. See the value of blocking
3. Use simple performance estimates to guide decisions

Tasks

Part 1

Below is a simple dense matrix-matrix multiply code, written in Fortran:

```
do i=1,matSize
  do j=1,matSize
    sum = 0.0
    do k=1,matSize
      !(Blocking for Part 2)
      sum = sum + matA(i,k)*matB(k,j)
    enddo
    matC(i,j) = sum
  enddo
enddo
```

Make two performance estimates for this code based on the two assumptions below. For these estimates, assume that the cost of a floating-point operation is c, a read from memory is r, and a write to memory is w. Assume reading and writing data from the cache is free (cost is zero); in other words, this is a simple model and should not include the effect of cachelines and cache policies.

Assumptions:

1. Assume that data is read once (it remains in cache after that and costs no more to access it). Create a formula for the time T to perform the matrix-matrix multiplication as a function of (c,r,w,n), where n is the size of the matrix (matSize in the code above).
2. Assume that MatB does not fit in cache and each data item in MatB must be read from memory each time it is used. All other data fits in cache. Create a formula for the time T(c,r,w,n) to perform the matrix-matrix multiplication in this case.

To get a better feel for the time models, compute the elapsed time for each estimate for these values of the parameters:

$C = .2 \times 10^{-9}$ sec/floating point operation
$R=W=10^{-9}$ sec/double precision word
$N=10^3$

Processor performance = 5 GFlops
Memory bandwidth = 8 GB/sec

Part 2
Modify this code to use blocking: In the above code, where "Blocking for **Part 2**" appears, add blocking in i and j of size bsize. That is, add at "! (Blocking for **Part 2**)"

```
do ii=i,min(i+bsize-1,matSize)
    do jj=j,min(j+bsize-1,matSize)
```

Then adjust the other loop limits accordingly.

Make sure that your code is correct. Compile your code with different degrees of optimization (e.g., for many compilers, -O0, -O1, -O2, and –O3) and report the times and computation rates for the original (unblocked) code and the blocked code, under different degrees of compiling optimization.

Note
Run your code with several matrix sizes: Use at least 20, 100, and 1000. For the largest size, pick a matrix size (matSize) that is large enough that both matrices does not fit into cache. For example, for a 16MB cache, the maximum matrix size that fits in cache is

$$\text{matSize}^2 = (16 \text{ MB}/8\text{B per double precision number}) = 2 \times 10^6$$
$$\text{Or matSize} = \text{sqrt}(2 \times 10^6) \approx 1413$$

So a matrix size of 1500 is large enough to ensure that the entire matrix won't fit in cache. In this case, you would provide results for matSize = 20, 100, 1000, 1500.

Be aware that your compiler may recognize the matrix-matrix multiply and replace it entirely with an optimized version (the Cray Fortran compiler on Blue Waters does just that). You can avoid that by inserting a call to a dummy routine (as you've done in the earlier homeworks) where the dummy routine is defined in a different file. This can keep the compiler from recognizing the computational pattern and replacing it with special code.

Submit
A PDF file containing the estimation formula for each assumption in **Part 1**, and computed elapsed time as well. For **Part 2**, the file should include the blocked code, a table and a plot on measurements (at least 3 sets of timings and computation rates)

and a brief description on determination of largest matSize so that matB does not fit in the cache.

Last but not least, a short but clear discussion of the results (are they what you expected? Is so, why? If not, why not?)

Thinking (but not turn in)

1. Did the results at any level of optimization surprise you? If so, look at the assembly output or optimization listing to see if you can find an explanation. For example, some compilers may detect the matrix-matrix multiply pattern and replace your code with optimized code.
2. What other arrangements of the loops should be considered? E.g., would you change the order of i, j, and/or k? Why? Does your compiler make any of those interchanges of order? Why?
3. Try modifying the operation in a small way so that it is no longer matrix-matrix multiplication (thus keeping the compiler from replacing your code with its own library implementation). For example, consider adding i to the value of matC in row i. Does that change the results? If you look at the assembly output or optimization report, do you see different code?