Introduction to Parallel Computing — Matthias K. Gobbert
Wintersemester 2011/2012 — Universität Kassel
Homework 6 — due on January 24, 2012

This homework uses the example of matrix-matrix multiplication to demonstrate the use of the celebrated Basic Linear Algebra Subprograms (BLAS) (Problem 1.) as well as their integration in utility functions (Problem 2.). Let $A \in \mathbb{R}^{m \times k}$ and $B \in \mathbb{R}^{k \times n}$ be given matrices. We wish to compute the matrix-matrix product

$$C := AB \in \mathbb{R}^{m \times n}. \tag{1}$$

We will denote the components in C-style counting as $A = (A_{iq})$, $B = (B_{qj})$, $C = (C_{ij})$ with indices $0 \leq i < m$, $0 \leq q < k$, and $0 \leq j < n$. By definition of the matrix product, the components of C are given by $C_{ij} = \sum_{q=0}^{k-1} A_{iq} B_{qj}$. The idea is to interpret this summation in different ways below to motivate different algorithms.
This homework builds on the previous homework, because you may be able able to re-use some of the utility functions and Problem 2. specifically aims at modifying the existing utility functions.

1. [10 points.] The purpose of this problem is to give you some basic experience with the BLAS. I am posting three files for this homework that are modified from the homework for the power method: (i) a new Makefile; the real point is only to show the changes in the LDFLAGS and DEFS definitions that are necessary to use the BLAS; (ii) utilities.c and utilities.h that demonstrate the use of the define statement `-DPARALLEL` in the Makefile on the example of a dot product using the BLAS `ddot` to allow for use of BLAS or not, based on the DEFS in the Makefile.

   This Problem 1. requires actually only to run the code in serial; you may still use MPI functions (like `MPI_Wtime`) by leaving the the pre-processor definition `-DPARALLEL` in the `Makefile` in place.

   Information on the BLAS including documentation on their use can be found at the Netlib Repository webpage `www.netlib.org`, then “Browse” under the entry `blas`.

   (a) Use command-line arguments to your program to read in the integers m, k, and n that determine all matrix dimensions. Program a function that sets up the matrices A and B. You can choose how to handle the setup and allocation of matrices; I recommend the one-dimensional data structures in `memory.c` to re-use the utility functions from before. You can choose random matrices in principle, but I recommend to use an example, for which the result C can be checked for correctness analytically.

   (b) Program all methods described below and ensure that the different methods give the same correct result.

   (c) Provide timing results that compare the different methods used in this problem. Report your results in table form with 7 columns that list m, k, n, and timings (in units of seconds) for naive, BLAS1, BLAS2, and BLAS3. To allow a comparison of results from all of us, submit your timings (in units of seconds) for the case $m = k = n = 8192$ by e-mail to me; this should be one row of your table. Analyze how the choices for the dimensions m, k, and n influence your results. [Hint:

Design one study for each of the three integers m, k, and n by fixing two of them at a suitable value (e.g., 8192) and varying only one of them (e.g., to smaller values like 4096, 2048, and 1024). Use powers of 2 for these integers such that the problem size doubles from one row of the table to the next.]

Naive C-code: Since we want to ensure that your code also works in an environment, where BLAS may not be available, provide code that computes the conventional component-wise definition

$$C_{ij} = \sum_{q=0}^{k-1} A_{iq} B_{qj} \tag{2}$$

directly without the use of BLAS. Notice that there may be more than one way to implement the details of this formula. If you have a pre-processor definition `-DBLAS` in the `Makefile`, this code should be in the `#else` case of the `#ifdef BLAS`.

BLAS 1: Write a function to compute C, but this time, using the BLAS1 routine `ddot` to compute the inner products of rows of A with columns of B to get C_{ij}.

BLAS 2: Write a function that uses the BLAS2 routine `dger` to implement the following alternative formula

$$C = AB = \sum_{q=0}^{k-1} \mathbf{a}_q \mathbf{b}_q^T, \tag{3}$$

where $\mathbf{a}_q$ and $\mathbf{b}_q^T$ are the qth column of A and the qth row of B, respectively.

BLAS 3: Write a routine that computes C using a call to the BLAS3 routine `dgemm`.

What to submit: Discuss what you did to ensure that the results from all methods used give the correct result. Submit the tables of observed wall clock times (in plain-text in body of e-mail). How does your naive code compare to the BLAS codes? What is your final recommendation for which BLAS to use?

2. [10 points.]

   (a) Use the strategy involving `#ifdef BLAS` ... `#else` ... to implement an alternative version using BLAS into your existing utility functions, such as the parallel dot product and the matrix-vector product function for dense matrices. Go through all your utility functions to find places where you can use this strategy.

   What to turn in: E-mail me a short report which utility functions you modified here and which BLAS function(s) you used.

   (b) Re-run some performance studies. For instance, the power method should now profit from the use of some BLAS.

   What to turn in: Submit (in plain-text in body of e-mail) tables of observed wall clock times both without and with BLAS used. Are you seeing improvements?