Wed. 04/25/12:

HW: Report is half of the work and the score.

HW1 often too short.

HW2 too short

HW3 nice attempt at reports, with some significant problems: If you change the algorithm itself, you need to discuss in detail!

→ Report text must explain what algorithm you implemented. Code listing is just backup.

HW3 #1 (b) Printing from copyright page

It is first edition.

05 04 03 14 13 12

might mean 11th printing from 02 = 2002.

(a) → Appendix A!

#3 (a) Ideas for improvement:

- improve output (output more variables, output format)
- implement command-line inputs

  ./trap a b n
- some error checking (n<0, b≥a, argc < 4, n must be int, n%np != 0)
- implement MPI_Wtime and outputs
- go to double-precision
- implement load-balancing if n%.np!=0

(e) Run for several n! Ex. 10^7, 10^8, 10^9

Comment on how close to optimal speedup you are.

Describe the table!

Ch. 5 MPI_Gather, etc.

Last time: Traditional C for matrices = multi-dimensional arrays does not work with MPI commands, since we cannot be sure that the data is consecutive in memory.
Therfore, let's program in 1-D arrays only, and use column-oriented storage to also be compatible with Fortran and Matlab for instance

Fortran: BLAS = basic linear algebra subroutines
LAPACK and other packages, that are actually Fortran code.

Ex.: $C = AB$ = matrix-matrix product

C_{ij} = dot product of row of A with col of B

= BLAS 1 = level-1 BLAS

= dot product fct. has 1 for loop

$C = \Sigma$ of outer products of col. of A with row of B

= BLAS 2 has 2 for-loops

$C = AB$ = BLAS 3 = has 3 for loops

$n = 8192$ for dim. of all matrices

BLAS 1	BLAS 2	BLAS 3
11,000 sec.	1,100 sec.	100 sec.

Could be faster depending on loop ordering!

Key reason: BLAS actually uses block algorithms! with blocks that are optimal for the cache!

This should motivate us to use a data structure that is compatible.

```
double *l_A   /* l_A ∈ R^{n×l_n} */
l_A = (double *) malloc( (n*l_n) * sizeof(double) )
```

Ex.: $A_{ij} = \frac{1}{i+j+1}$ $\quad 0 \le i, j < n$

Hilbert matrix

```
for (j = 0; j<n; j++)
    for (i = 0; i<n; i++)
        A[i+n*j] = 1.0 / ((double)(i+j+1))
```

The i index is in the inner loop because in that case the memory of A is accessed consecutively. since the code will load several elements of A into cache at the same time.

Ex. MPI_Scatter(A, n*l_n, MPI_DOUBLE, l_A, n*l_n, MPI_DOUBLE, 0, MPI_COMM_WORLD)

$l\_n = n/np$

$A = \left[l\_A \mid l\_A \mid \cdots \mid l\_A \right]$

HW4: Some ideas:

#1 (a) MPI_Gather from l_A to A → above

(b) dot product → last time

(c) double Enorm (double *l_x, ...)

return sqrt(parallel_dot(l_x, l_x, ...))

→ include math.h and link -lm

(d) $y = Ax$ for $x, y \in \mathbb{R}^n$, $A \in \mathbb{R}^{n \times n}$

and requirement that A is given as l_A, x as l_x, and output must be y as l_y

$$y = \begin{bmatrix} \underline{l\_y} \\ \underline{l\_y} \\ l\_y \end{bmatrix} = Ax = \left[\begin{array}{c|c|c} l\_A & l\_A & l\_A \end{array}\right] \begin{bmatrix} \underline{l\_x} \\ \underline{l\_x} \\ l\_x \end{bmatrix}$$