HPC - Skopje

date: Nov. 7, 2011

Some Benchmarks

First, we have to provide a few basic benchmarks examples:

(A) The inner product of two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^N$:

$$\langle \mathbf{x}, \mathbf{y} \rangle := \sum_{i=0}^{N-1} x_i \cdot y_i$$

Use $x_i := (i \mod 219) + 1$ and $y_i := 1.0/x_i$ as test data. (Why !?)

(B) Matrix-Vector product using an $M \times N$ matrix A and vectors $\mathbf{x} \in \mathbb{R}^N$, $\mathbf{b} \in \mathbb{R}^M$:

$$\begin{aligned} \mathbf{b} &:= A \cdot \mathbf{x} \\ b_i &:= \sum_{j=0}^{N-1} A_{i,j} x_j \qquad \forall i = 0, \ldots, M-1 \end{aligned}$$

Use $A_{i,j} := ((i + j) \mod 219) + 1$ and $x_j := 1.0/A_{17,j}$ as test data. (Why !?)

(C) Matrix-Matrix product: $C_{M \times N} := A_{M \times L} \cdot B_{L \times N}$

$$C_{i,j} := \sum_{k=0}^{L-1} A_{i,k} \cdot B_{k,j} \qquad \forall i = 0, \ldots, M-1, \quad \forall j = 0, \ldots, N-1$$

(D) Evaluation of a polynomial function of degree p with given coefficients $a_k, k = 0, \ldots, p$ for a vector $\mathbf{x} \in \mathbb{R}^N$, i.e., $\mathbf{y} := \mathtt{poly}_p(\mathbf{x}) \in \mathbb{R}^N$ with

$$y_i := \mathtt{poly}_p(x_i) = \sum_{k=0}^{p} a_k \cdot x^k \qquad \forall i = 0, \ldots, N-1$$

5. Measure the achieved floating point performance and and memory bandwidth of your test system by compiling and using flops.c and stream.c . Have a look at the first lines in the source code for compiling options (Benchmarking is science by itself - we need the above codes to have some numbers in our hands).

6. Some counting and calculating for benchmarks (A)–(D):

   - Calculate the amount of memory needed for your data as function of the data dimensions M, N, L, p depending on the chosen data type.
   - Calculate the number of floating point operations ($+, -, *, /$ count as one operation) as function of the data dimensions M, N, L, p.
   - Express the number of Read/Write operations from/to memory as function of the data dimensions M, N, L, p and the chosen data type.

7. Implement benchmarks (A)–(D) in C/C++ as **separate functions** where all data will be transferred via the parameter list, i.e., no global data are allowed. The data types for **double precision** accuracy have to be used.
   In order to simplify code development and benchmarking you are encouraged to use the provided template. After extracting the files you have to type

   ```
   make
   ./main
   ```

   into your (LINUX-) Terminal to compile, link and run the code.

8. Measure the runtime of your implementations. Calculate the achieved double precision performance in GFLOPS and the achieved memory bandwidth in GiB/sec for your implementations.
   Take care that you use compiler options wrt. optimization, i.e., "-O0" versus "-O3" and more advanced options.
   Chose the data dimensions for each benchmark such that the code runs at least 10 sec. (Why?!), i.e.. If the runtime for (A) is too low although you use already 50% of your memory than you have to increase `NLOOPS` in the template accordingly. The memory required to store your data should be approximately 10–100 times larger than your largest cache.

9. Some special tasks:

   (a) Measure the performance in (A) by replacing the call to $< x, y >$ by a call to $\| x \| := \sqrt{\sum_{i=0}^{N-1} x_i^2}$.
   What do you observe? Why?

   (b) Assume row-wise access to matrix A in (C). Compare performance issues for row-wise access of B versus column-wise access of B.

10. Write additional functions for (A)-(C) that use the BLAS[1] library, especially the calls DOT, GEMV, GEMM, see reference card in pdf or html.

[1] `/usr/share/doc/libblas-doc`

Measure run time and calculate GFLOPS and GiB/sec.
(*) What is the best BLAS available? Atlas[2] with Fortran-Blas and cblas[3], uBLAS (C++ boost) or the Intel-library?
Check your computed GFLOPS with the output for the appropriate ATLAS test routines ($< N >$, $< M >$, $< L >$ have to be replaced by *your* parameter choice):

(A) `/usr/lib/libatlas-test/xdl1blastst -R dot -n <N>  -F 1000`

(B) `/usr/lib/libatlas-test/xdl2blastst -R gemv -n <N> -m <M> -F 1000`

(C) `/usr/lib/libatlas-test/xdl3blastst -R gemm -n <N> -m <M> -k <L>  -F 1000`

11. Improve your implementations, especially in example (D).
Take into account what limits your application and try to improve it by mainly reducing memory transfers and better use of temporal and spatial locality of data.
Some keywords:

    - data layout,
    - loop interchange,
    - automatic loop unrolling,
    - manual loop unrolling,
    - vectorization

[2] `/usr/share/doc/libatlas-doc`
[3] `/usr/share/doc/libblas-doc`