WIKIPEDIA

Automatically Tuned Linear Algebra Software

Automatically Tuned Linear Algebra Software (**ATLAS**) is a software library for linear algebra. It provides a mature open source implementation of BLAS APIs for C and Fortran77.

ATLAS

Repository	github.com/math-atlas/math-atlas (https://github.com/math-atlas/math-atlas)
Type	Software library
License	BSD License
Website	math-atlas.sourceforge.net (http://math-atlas.sourceforge.net)

ATLAS is often recommended as a way to automatically generate an optimized BLAS library. While its performance often trails that of specialized libraries written for one specific hardware platform, it is often the first or even only optimized BLAS implementation available on new systems and is a large improvement over the generic BLAS available at Netlib. For this reason, ATLAS is sometimes used as a performance baseline for comparison with other products.

ATLAS runs on most Unix-like operating systems and on Microsoft Windows (using Cygwin). It is released under a BSD-style license without advertising clause, and many well-known mathematics applications including MATLAB, Mathematica, Scilab, SageMath, and some builds of GNU Octave may use it.

Contents

Functionality

ATLAS provides a full implementation of the BLAS APIs as well as some additional functions from LAPACK, a higher-level library built on top of BLAS. In BLAS, functionality is divided into three groups called levels 1, 2 and 3.

- Level 1 contains *vector operations* of the form

$$\mathbf{y} \leftarrow \alpha\mathbf{x} + \mathbf{y}$$

as well as scalar dot products and vector norms, among other things.

- Level 2 contains *matrix-vector operations* of the form

  $$\mathbf{y} \leftarrow \alpha A\mathbf{x} + \beta\mathbf{y}$$

  as well as solving $T\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$ with T being triangular, among other things.

- Level 3 contains *matrix-matrix operations* such as the widely used General Matrix Multiply (GEMM) operation

  $$C \leftarrow \alpha AB + \beta C$$

  as well as solving $B \leftarrow \alpha T^{-1}B$ for triangular matrices T, among other things.

Optimization approach

The optimization approach is called Automated Empirical Optimization of Software (AEOS), which identifies four fundamental approaches to computer assisted optimization of which ATLAS employs three:[1]

1. Parameterization—searching over the parameter space of a function, used for blocking factor, cache edge, etc.
2. Multiple implementation—searching through various approaches to implementing the same function, e.g., for SSE support before intrinsics made them available in C code
3. Code generation—programs that write programs incorporating what knowledge they can about what will produce the best performance for the system

- Optimization of the level 1 BLAS uses parameterization and multiple implementation

  Every ATLAS level 1 BLAS function has its own kernel. Since it would be difficult to maintain thousands of cases in ATLAS there is little architecture specific optimization for Level 1 BLAS. Instead multiple implementation is relied upon to allow for compiler optimization to produce high performance implementation for the system.

- Optimization of the level 2 BLAS uses parameterization and multiple implementation

  With N^2 data and N^2 operations to perform the function is usually limited by bandwidth to memory, and thus there is not much opportunity for optimization
  All routines in the ATLAS level 2 BLAS are built from two Level 2 BLAS kernels:

- - GEMV—matrix by vector multiply update:

    $$\mathbf{y} \leftarrow \alpha A\mathbf{x} + \beta\mathbf{y}$$

- - GER—general rank 1 update from an outer product:

    $$A \leftarrow \alpha\mathbf{x}\mathbf{y}^T + A$$

- Optimization of the level 3 BLAS uses code generation and the other two techniques

  Since we have N^3 ops with only N^2 data, there are many opportunities for optimization

Level 3 BLAS

Most of the Level 3 BLAS is derived from GEMM, so that is the primary focus of the optimization.

$O(n^3)$ operations vs. $O(n^2)$ data

The intuition that the n^3 operations will dominate over the n^2 data accesses only works for roughly square matrices. The real measure should be some kind of surface area to volume. The difference becomes important for very non-square matrices.

Can it afford to copy?

Copying the inputs allows the data to be arranged in a way that provides optimal access for the kernel functions, but this comes at the cost of allocating temporary space, and an extra read and write of the inputs.

So the first question GEMM faces is, can it afford to copy the inputs?

If so,

- Put into block major format with good alignment
- Take advantage of user contributed kernels and cleanup
- Handle the transpose cases with the copy: make everything into TN (transpose - no-transpose)
- Deal with α in the copy

If not,

- Use the nocopy version
- Make no assumptions on the stride of matrix *A* and *B* in memory
- Handle all transpose cases explicitly
- No guarantee about alignment of data
- Support α specific code
- Run the risk of TLB issues, bad strides, etc.

The actual decision is made through a simple heuristic which checks for "skinny cases".

Cache edge

For 2nd Level Cache blocking a single cache edge parameter is used. The high level choose an order to traverse the blocks: *ijk, jik, ikj, jki, kij, kji*. These need not be the same order as the product is done within a block.

Typically chosen orders are *ijk* or *jik*. For *jik* the ideal situation would be to copy *A* and the *NB* wide panel of *B*. For *ijk* swap the role of *A* and *B*.

Choosing the bigger of *M* or *N* for the outer loop reduces the footprint of the copy. But for large *K* ATLAS does not even allocate such a large amount of memory. Instead it defines a parameter, *Kp*, to give best use of the L2 cache. Panels are limited to *Kp* in length. It first tries to allocate (in the *jik* case) $M \cdot p + NB \cdot Kp + NB \cdot NB$. If that fails it tries $2 \cdot Kp \cdot NB + NB \cdot NB$. (If that fails it uses the no-copy version of GEMM, but this case is unlikely for reasonable choices of cache edge.) *Kp* is a function of cache edge and *NB*.

LAPACK

When integrating the ATLAS BLAS with LAPACK an important consideration is the choice of blocking factor for LAPACK. If the ATLAS blocking factor is small enough the blocking factor of LAPACK could be set to match that of ATLAS.

To take advantage of recursive factorization, ATLAS provides replacement routines for some LAPACK routines. These simply overwrite the corresponding LAPACK routines from Netlib.

Need for installation

Installing ATLAS on a particular platform is a challenging process which is typically done by a system vendor or a local expert and made available to a wider audience.

For many systems, architectural default parameters are available; these are essentially saved searches plus the results of hand tuning. If the arch defaults work they will likely get 10-15% better performance than the install search. On such systems the installation process is greatly simplified.

References

1. R. Clint Whaley; Antoine Petitet & Jack J. Dongarra (2001). "Automated Empirical Optimization of Software and the ATLAS Project" (http://www.netlib.org/lapack/lawnspdf/lawn147.pdf) (PDF). *Parallel Computing*. **27** (1–2): 3–35. CiteSeerX 10.1.1.35.2297 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.2297). doi:10.1016/S0167-8191(00)00087-9 (https://doi.org/10.1016%2FS0167-8191%2800%2900087-9). Retrieved 2006-10-06.

External links

- Automatically Tuned Linear Algebra Software (https://sourceforge.net/projects/math-atlas/) on SourceForge.net
- User contribution to ATLAS (http://math-atlas.sourceforge.net/devel/atlas_contrib/)
- A Collaborative guide to ATLAS Development (http://math-atlas.sourceforge.net/devel/atlas_devel/)
- The FAQ (http://math-atlas.sourceforge.net/faq.html#doc) has links to the Quick reference guide to BLAS and Quick reference to ATLAS LAPACK API reference
- Microsoft Visual C++ Howto (http://www.terborg.net/research/kml/installation.html) for ATLAS