## Automatic Library Tuning:

# An overview of ATLAS

## (*Automatic Tuned Linear Algebra Software*)

**Tony Drummond**

**Lawrence Berkeley National Laboratory**

**acts-support@nersc.gov**

**LADrummond@lbl.gov**

## Why is it so hard to obtain performance?

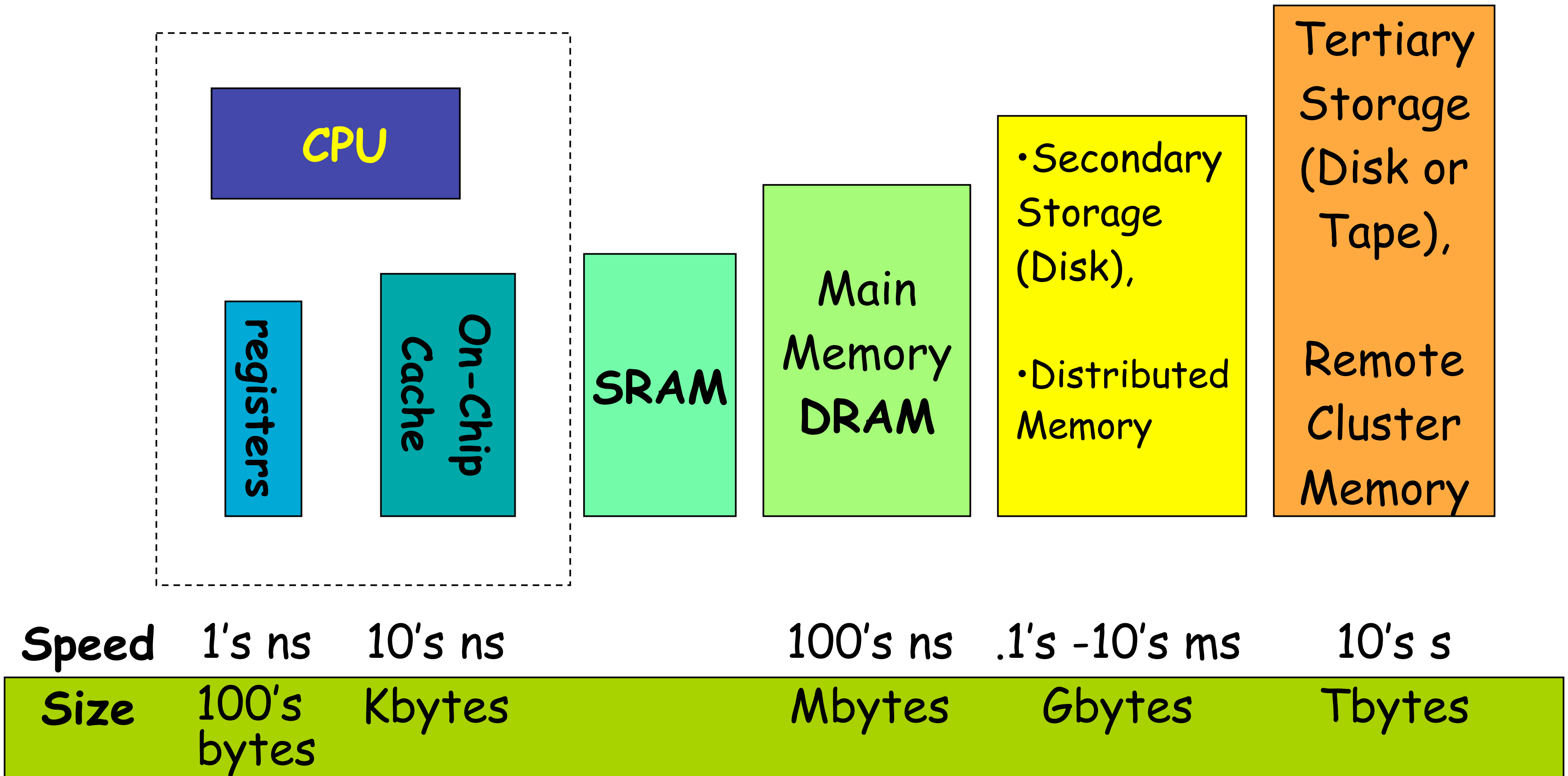
Some common answers:

- Algorithms
- Programming Practices
- Computer and Software Technology
- Memory latency

⋮

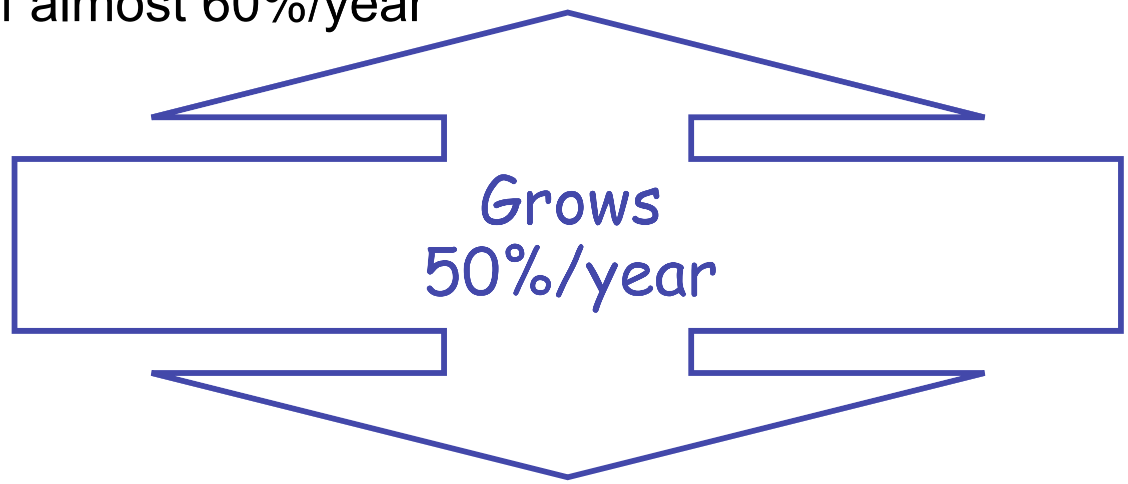
# Memory Hierarchy

- *Where is the data? Why is data locality important?*



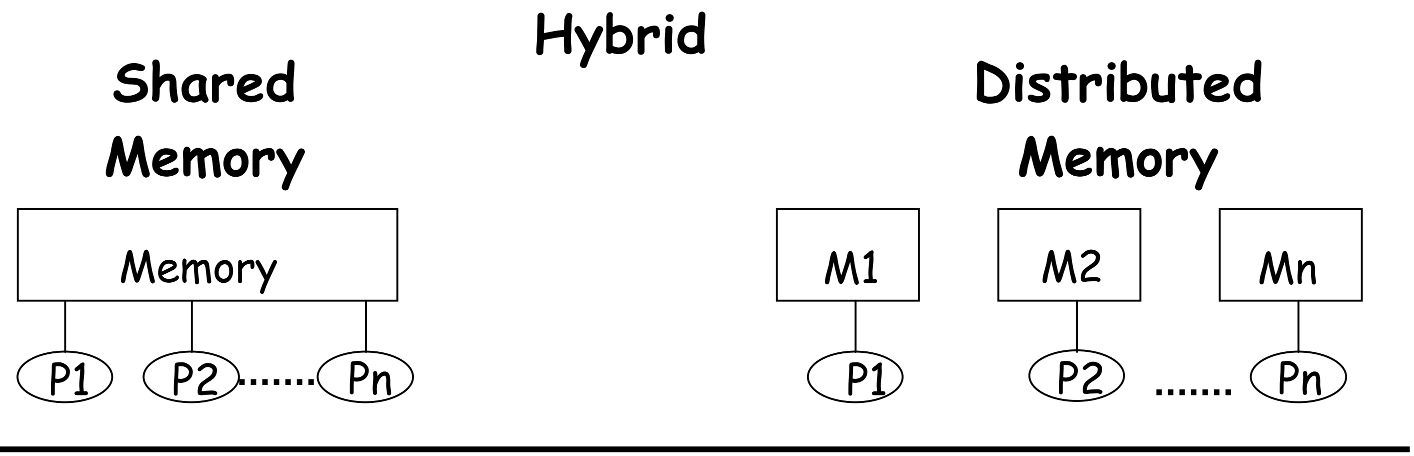
# CPU vs. DRAM Performance

• Since 1980's, mProcs performance has increased at a rate of almost 60%/year




• Since 1980's, DRAM (latency) has improved at a rate of almost 9%/year

# Some Parallel Programming Tools



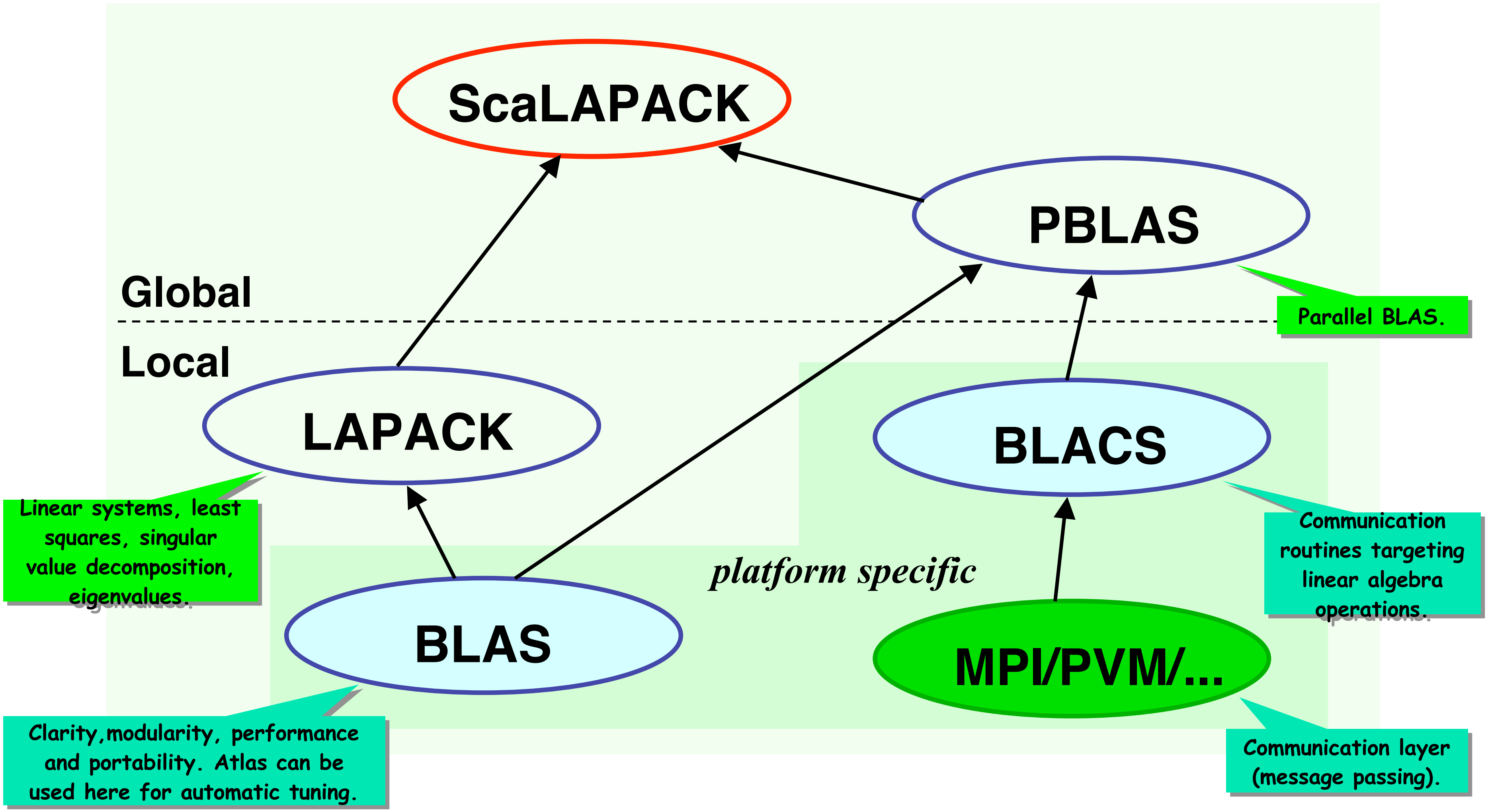

HPC software Toolkits like tools under **ACTS**

Basic support libraries: shmem, PVM 3, MPI

Compiler directives, optimization options, , OpenMP, multi-thread

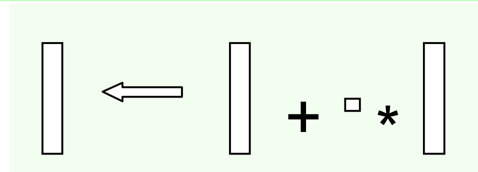
Fortran 77, F90, C, C++, Java HPF

# ScaLAPACK: *software structure*

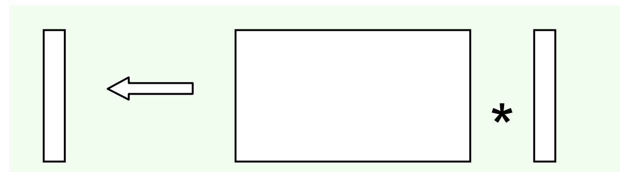


# BLAS: *3 levels*

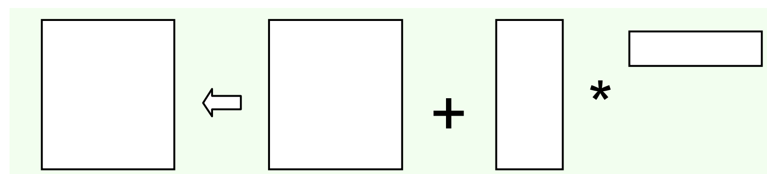
- Level 1 BLAS:
  vector-vector operations.

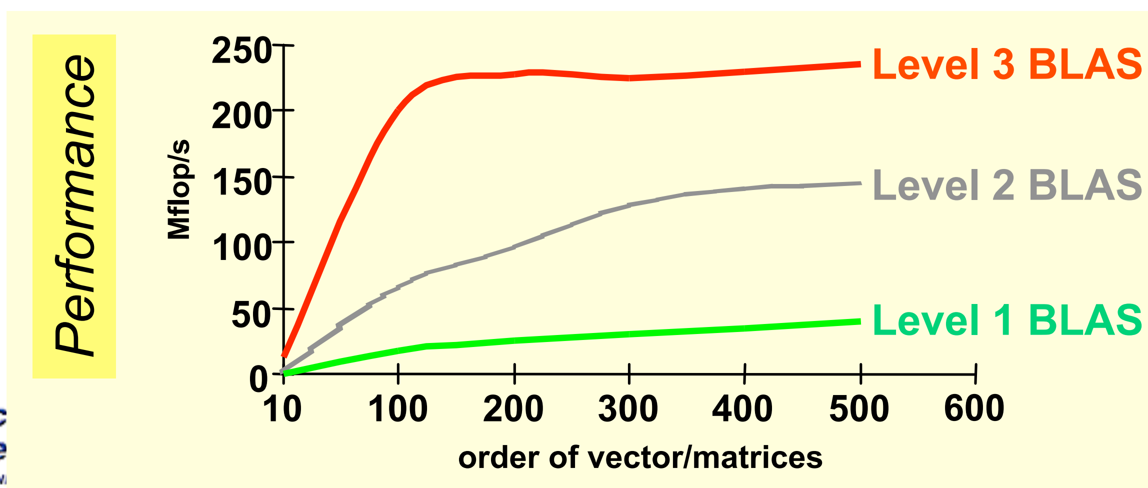



- Level 2 BLAS:
  matrix-vector operations.




- Level 3 BLAS:
  matrix-matrix operations.




**Development of blocked algorithms is important for performance!**

# The ATLAS Project

Automatic Tuned Linear Algebra Software
J. Dongarra, A. Petite and R. Whaley
http://math-atlas.sourceforge.net/

- Automatic Optimization of Basic Linear Algebra and some LAPACK routines
- Design for RISC architectures
- Parameter Optimization (automatic - several tests)
  - TBL access
  - L1 cache reuse
  - FP use
  - memory fetch
  - register use
  - Loop overhead reuse

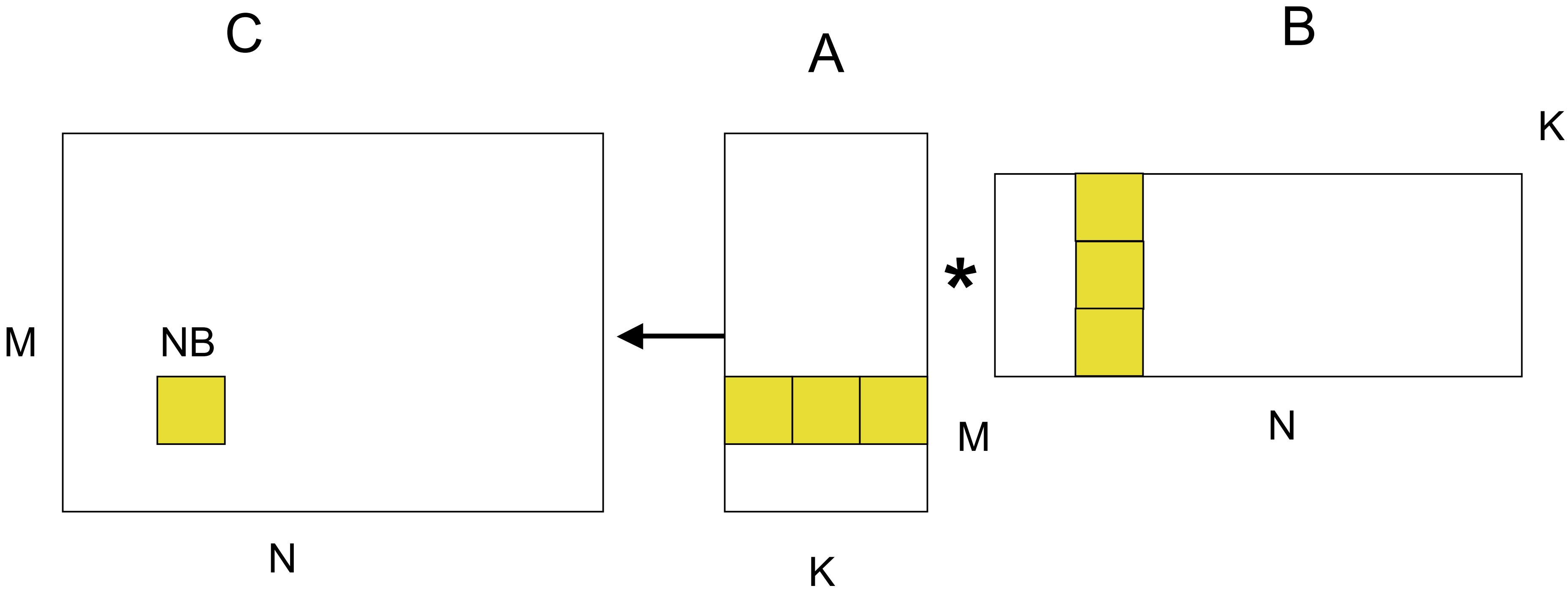
# The ATLAS Project

**Portability Across Platforms**

- Contains:
  - Code Generators (automatically generate BLAS routines)
  - Sophisticated Timers
  - Robust search routines
- Code is iteratively generated and timed until optimal case is found by varying sequentially the *performance-critical parameters* (number of Blocks, Breaking false dependencies, loop unrolling)

# On-Chip Multiply

BLAS OPERATIONS WRITEN IN TERMS OF ON-CHIP MULTIPLY



# On-Chip Multiply

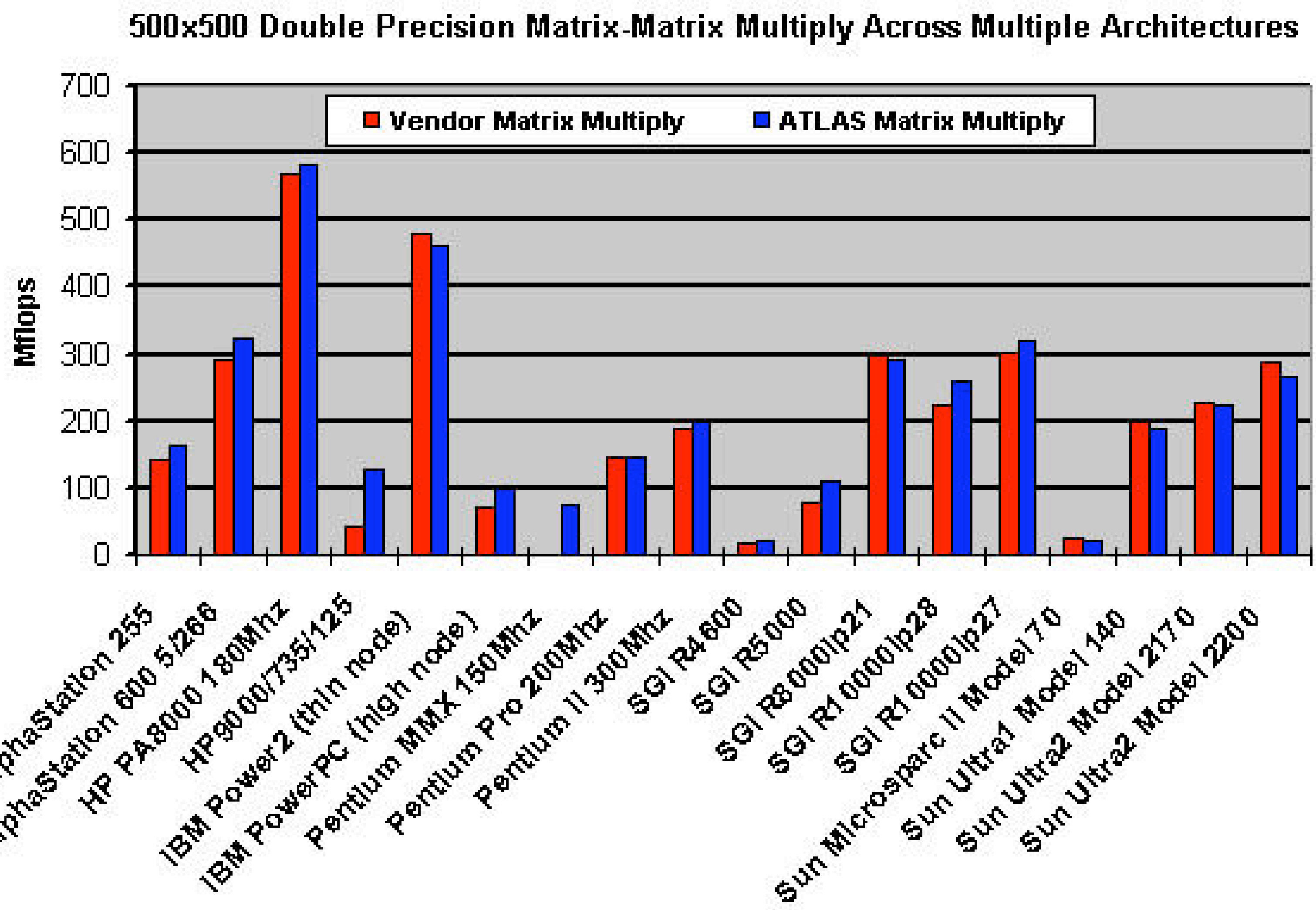
Optimization for

- TLB Access
- L1 cache reuse
- FP unit usage
- Memory Fetch
- Register reuse
- Loop overhead minimization

# BLAS: *3 levels*

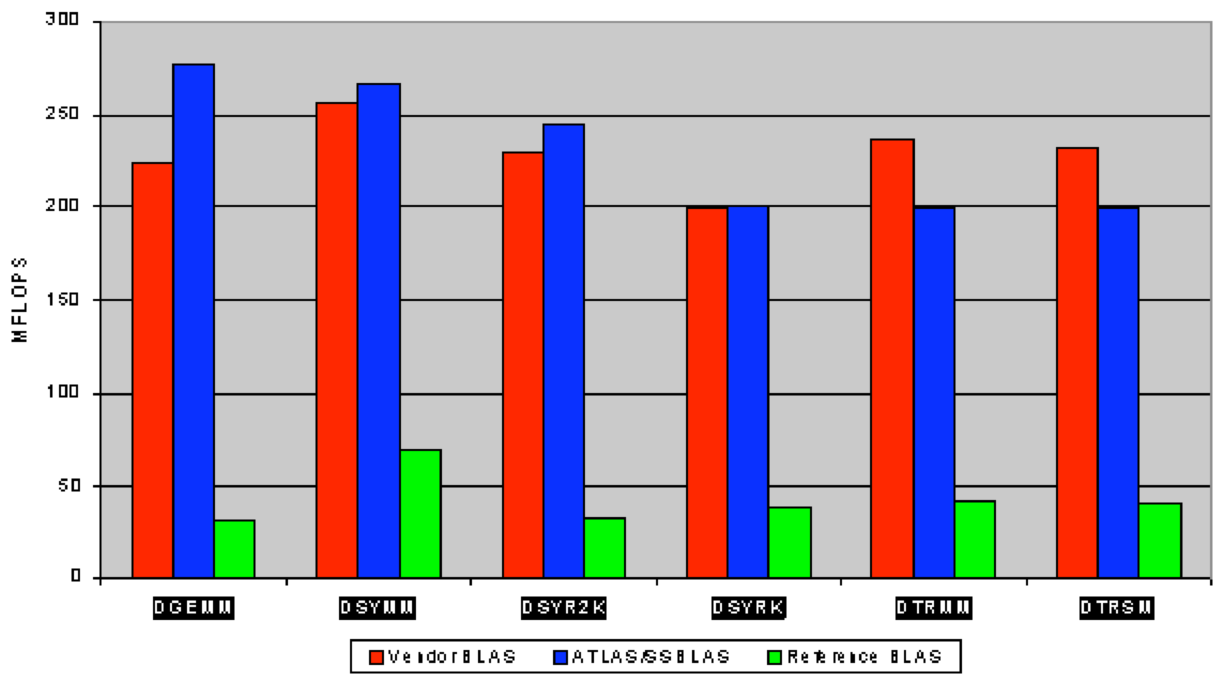
- Level 1 BLAS:
  - vector-vector operations.
  - Compilers mostly do a good job
  - 10-15% improvement
- Level 2 BLAS:
  - matrix-vector operations.
  - Some vector blocking is possible
  - Some Loop unrolling
  - 15-30% improvement
- Level 3 BLAS:
  - matrix-matrix operations.
  - Matrix blocking is possible
  - Fetches $O(n^3)$ to $O(n^2)$
  - Higher improvement percentages

# Automatic Tuning Pays Off!

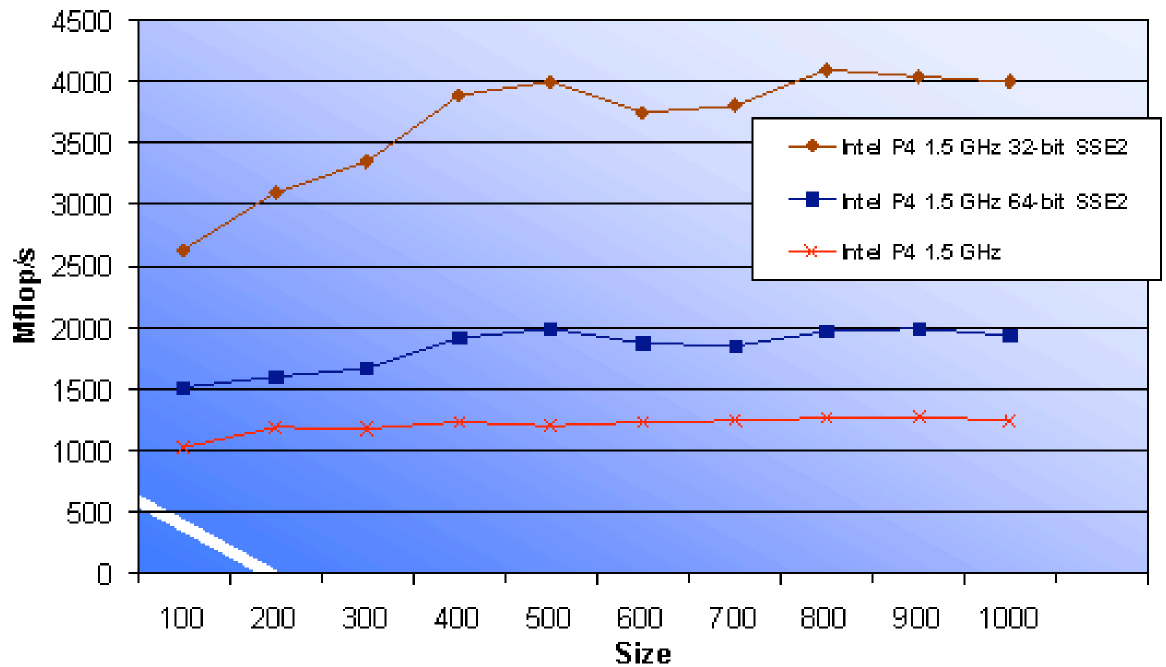


# Automatic Tuning Pays Off!

## 500x500 gemm-based BLAS on SGI R10000ip28

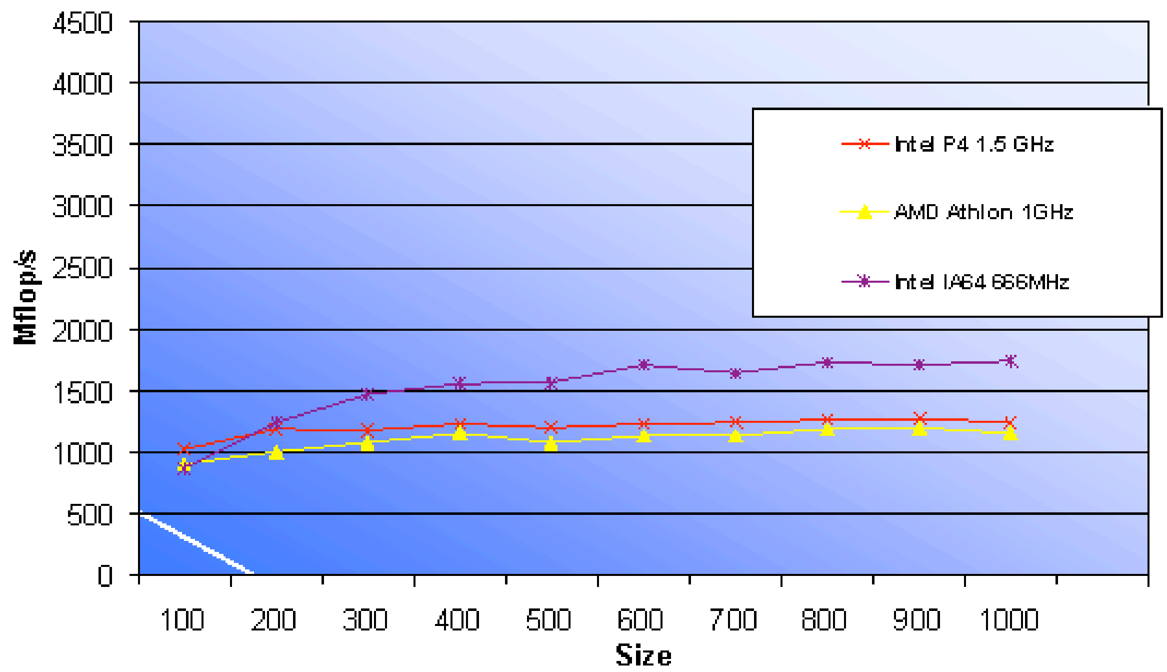


# Automatic Tuning Pays Off!



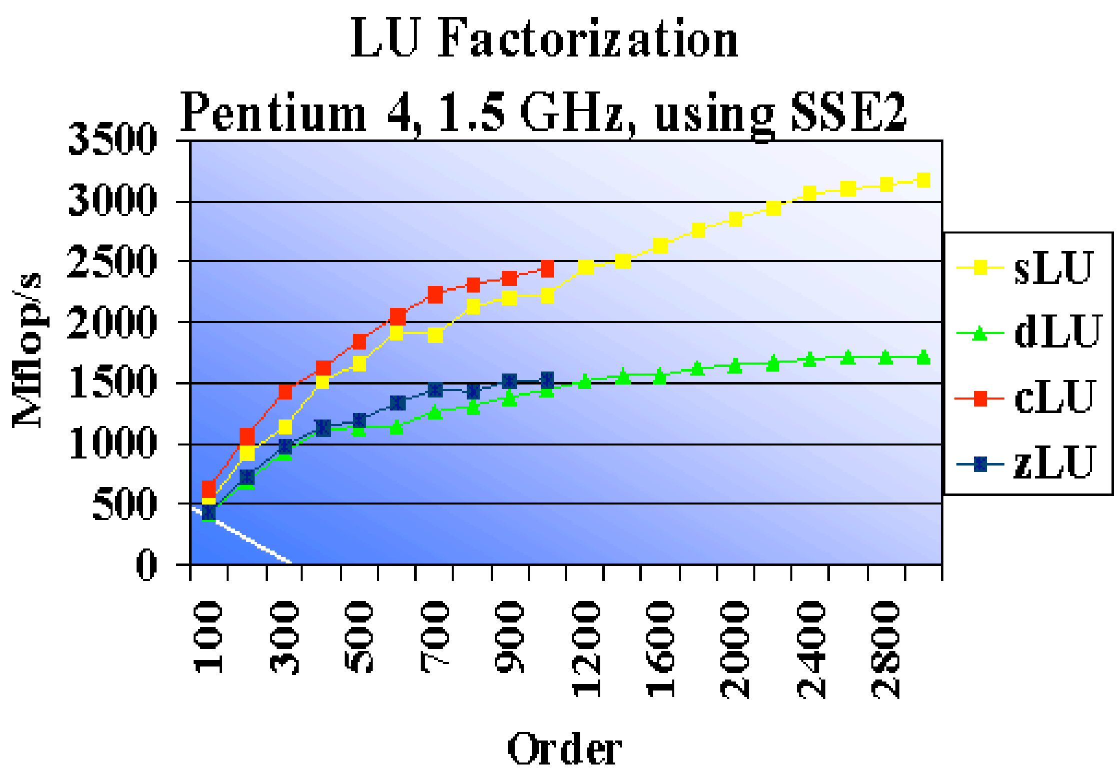

Performance of a matrix multiply code generated by ATLAS for a 1.5 GHz Pentium 4 - SSE2.

# Automatic Tuning Pays Off!




**Performance of a matrix multiply code generated by ATLAS for different processors (64-bit floating point).**

# Automatic Tuning Pays Off!




Performance of a LU factorization code generated by ATLAS for a 1.5 GHz Pentium 4 - SSE2 (sLU = 32-bit real, dLU = 64-bit real, cLU = 32-bit complex, zLU = 64-bit complex).

# Using ATLAS

1. Download ATLAS
   **https://sourceforge.net/project/showfiles.php?group_id=23725**
2. Uncompress and untar the file
3. From the ATLAS directory
4. Type
   - make config CC=<*your ANSI C compiler*> (if you don’t know how to answer to something during the config answer n and let the system figure it out)
   - make install arch=<*arch_name*>

# While Building ATLAS

```
Calculating L1 cache size:
      L1CS=2, time=1.010000
      L1CS=4, time=0.980000
      L1CS=8, time=0.990000
      L1CS=16, time=0.960000
      L1CS=32, time=0.980000
      L1CS=64, time=3.100000

      Confirming result of 32kb:
      L1CS=2, time=1.040000
      L1CS=4, time=0.970000
      L1CS=8, time=0.970000
      L1CS=16, time=0.960000
      L1CS=32, time=0.9
      L1CS=64, time=3.0
```

```
      L1CS=2, time=1.020000
      L1CS=4, time=0.970000
      L1CS=8, time=0.990000
      L1CS=16, time=0.990000
      L1CS=32, time=0.990000
      L1CS=64, time=3.040000

Calculated L1 cache size = 32kb; Correct=1
```

# While Building ATLAS

```
DD=1, lat=1, mf=391.86
      MULADD=1, lat=2, mf=769.95
      MULADD=1, lat=3, mf=1174.04
      MULADD=1, lat=4, mf=1569.35
      MULADD=1, lat=5, mf=1567.40
      MULADD=1, lat=6, mf=1562.18
      MULADD=0, lat=1, mf=331.05
      MULADD=0, lat=2, mf=662.09
      MULADD=0, lat=3, mf=781.09
      MULADD=0, lat=4, mf=781.71
      MULADD=0, lat=5, mf=790.18
      MULADD=0, lat=6, mf=797.25
      nreg=8, mflop = 1625.47 (peak 1569.35)
      nreg=16, m
      nreg=32, m

  nreg < 32 (dro
```

```
          nreg=24, mflop = 1172.83 (peak 1569.35)
          nreg=20, mflop = 1396.51 (peak 1569.35)
          nreg=18, mflop = 1569.14 (peak 1569.35)
          nreg=19, mflop = 1501.23 (peak 1569.35)

Lower bound on number of registers = 18
```

# Checking ATLAS results

- `Make sanity_check arch=<arch_name>`
- You should then read ATLAS/doc/TestTime.txt for instructions on testing and timing your installation.

# What you get from ATLAS

**SUMMARY.LOG** : The SUMMARY.LOG created by atlas_install.
**cblas.h** : The C header file for the C interface to the BLAS.
**clapack.h** : The C header file for the C interface to LAPACK.
**liblapack.a** : The LAPACK routines provided by ATLAS.
**libcblas.a** : The ANSI C interface to the BLAS.
**libf77blas.a** : The Fortran77 interface to the BLAS.
**libatlas.a** : The main ATLAS library, providing low-level routines for all interface libs.

Additional libraries, if it has posix thread support:

**libptcblas.a** : The ANSI C interface to the threaded (SMP) BLAS.
**libptf77blas.a** : The Fortran77 interface to the threaded (SMP) BLAS.

# Where is ATLAS used today

**Problem Solving Environments**

- MAPLE, MATLAB, Mathematica, MAPLE and Octave

**Compilers**

- Absoft Pro Fortran

**Libraries (optionally)**

- GSL, HPL, LAPACK, MPB, The R Project, Scientific Python, Scilab, PWSCF

**Operating Systems (included in some way)**

- Debian Linux, FreeBSD, Mac OS 10, Scyld Beowulf, SuSE Linux

# Some Related Projects

- Automatic performance tuning systems
  - Berkeley Benchmarking and Optimization Project (BeBOP) **http://bebop.cs.berkeley.edu**
  - Portable High Performance ANSI C (PHiPAC) **http://www.icsi.berkeley.edu/%7Ebilmes/phipac**
  - FFTW **http://www.fftw.org**
  - UHFFT **http://www2.cs.uh.edu/~mirkovic/fft/parfft.htm**
  - SANS (Self Adapting Numerical Software) **http://icl.cs.utk.edu/sans/index.html**