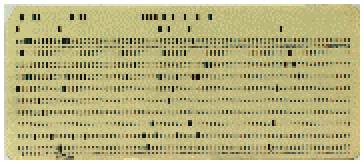
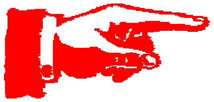
# Java Access to Numerical Libraries: Compiling Fortran to Java





Jack Dongarra

Christian Deane

Keith Seymour

Clint Whaley

**University of Tennessee**

**Oak Ridge National Laboratory**

# Motivation

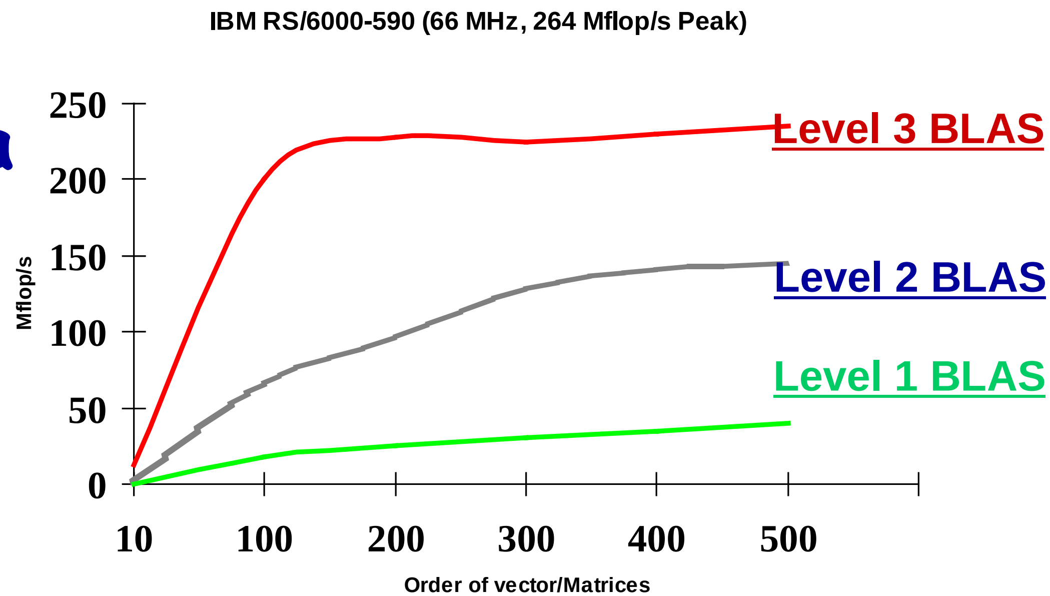
- Provide well-known and reliable libraries
- Avoid re-writing numerical code
- Quick and reliable translation
- Performance

# LAPACK

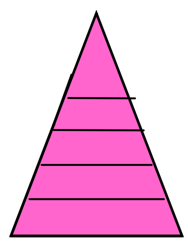
- Linear Algebra library in Fortran 77 (binding to c)
  - State of the art numerical routines
  - Extensive coverage
  - Solution of systems of equations
  - Solution of eigenvalue problems
- Block algorithms
  - Parameterized for memory hierarchies
    - Built on the Level 1, 2, and 3 BLAS
  - Efficient on a wide range of computers
    - RISC, Vector, SMPs
- User interface provides similar calls in:
  - Single, Double, Complex, Double Complex
- Used by vendors: HP-48G to Teraflop/s Machines

# Why Higher Level BLAS?

- Can only do arithmetic on data at the top of the hierarchy
- Higher level BLAS lets us do this




| BLAS | Memory Refs | Flops | Flops/ Memory Refs |
|---|---|---|---|
| Level 1 $y = y + \alpha x$ | 3 n | 2 n | 2 / 3 |
| Level 2 $y = y + A x$ | $n^2$ | $2 n^2$ | 2 |
| Level 3 $C = C + A B$ | $4 n^2$ | $2 n^3$ | n/ 2 |



- Development of blocked algorithms important for performance

# LAPACK

- 600K lines of code
- extensive test package
- developed over 10 years with input from the linear algebra community
- state of the art methods
- Automatically translate to Java

# Outline of Project

- **Phase 1: Write Fortran front end to lex and parse subset Fortran 77.**
- **Phase 2: Generate Java source and Jasmin assembly code for use with JVM.**
- **Phase 3: Test, document and distribute BLAS and LAPACK class files.**

# Array Access/Argument Passing

- All arrays are declared as 1D and accessed with index arithmetic.
- Array indices must be passed separately as arguments, which changes the user interface.
- Primitives are passed in object wrappers to emulate pass-by-reference (only when needed, though).

# GOTO Translation

- **First Step**

**Try to identify Fortran constructs containing GOTO statements that can be translated to equivalent Java constructs (which cannot contain a goto statement).**

```
10    CONTINUE
      IF(C .EQ. ONE) THEN
         A = 2 * A
         GO TO 10
      END IF
```

→

```
while(c == one)
{
   a = 2 * a;
}
```

# GOTO Translation

- **Second Step**

  **Remaining GOTO statements must be transformed at the bytecode level.**

- **Bytecode Transformer**

  - **Use bytecode parsing code from javab (Indiana University) .**
  - **Provides efficient translation of GOTO statements.**

# Input/Output

- Small subset of WRITE/FORMAT has been implemented -- just enough to allow translation of BLAS/LAPACK test routines.
- Some unformatted READ statements supported.
- File I/O not supported yet, but may be necessary for future testing.

# Current Status of Project

- f2j: formal compiler of Fortran 77 subset sufficient for BLAS, LAPACK and other numerical libraries.
- Most I/O statements are not fully supported.
- Double precision BLAS levels 1, 2, and 3 successfully tested.
- Double precision LAPACK routines successfully tested.
- released: May 22, 1998

# Future of Fortran-to-Java Project

- **Extend to wider subsets of Fortran and translate more numerical libraries.**
- **Support for Complex data type.**
- **Provide a large reliable Java numerical software repository.**
- **Focus on optimization**

# How To Get Performance From Commodity Processors?

- Today's processors can achieve high-performance, but this requires extensive machine-specific hand tuning.
- Routines have a large design space w/many parameters
  - blocking sizes, loop nesting permutations, loop unrolling depths, software pipelining strategies, register allocations, and instruction schedules.
  - Complicated interactions with the increasingly sophisticated microarchitectures of new microprocessors.
- A few months ago no tuned BLAS for Pentium for Linux.
- Need for quick deployment of optimized routines.
- ATLAS - Automatic Tuned Linear Algebra Software

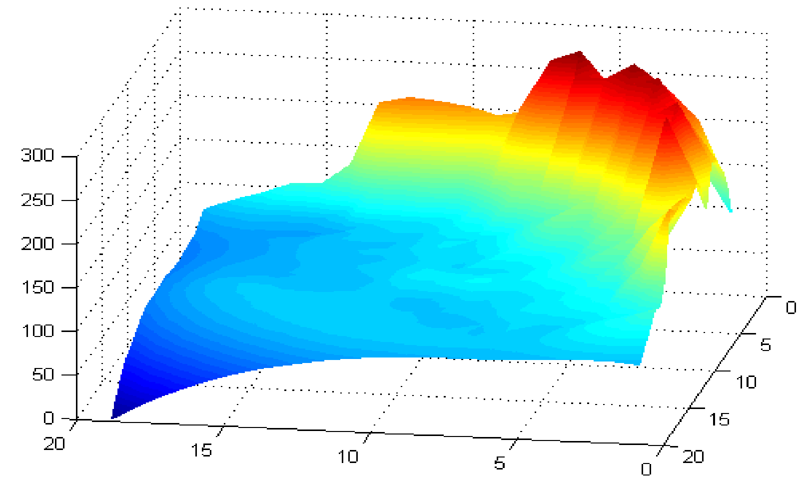
# What is ATLAS

- A package that adapts itself to differing architectures via code generation coupled with timing
  - Initially, supply BLAS
- Package contains:
  - Code generators
  - Sophisticated timers
  - Robust search routines
- Currently provided:
  - Real matrix matrix multiply
    - 1-2 hours install time

# Why ATLAS is needed

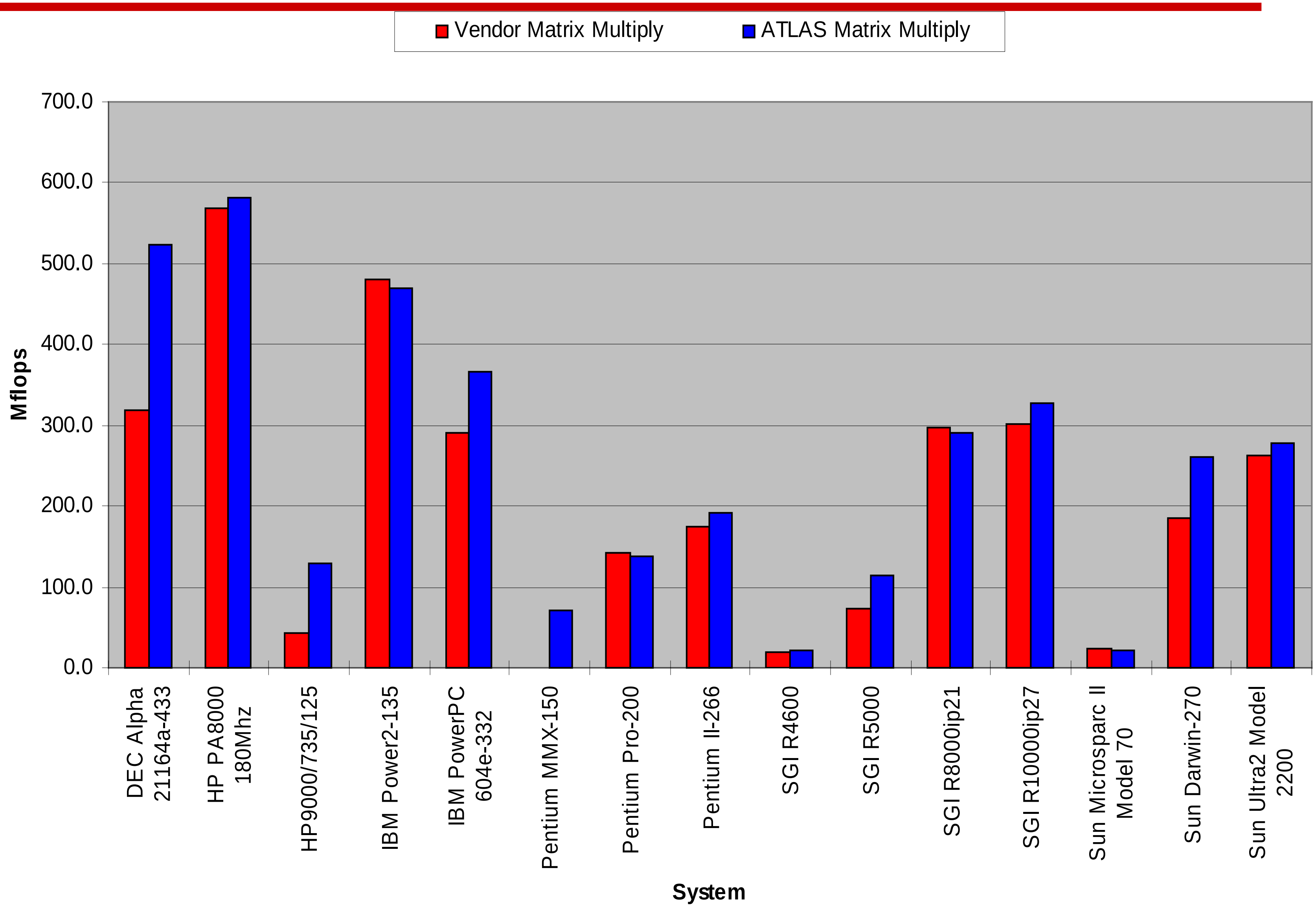
- BLAS require many man-hours / platform
  - Only done if financial incentive is there
    - Many platforms will never have an optimal version
  - Lags behind hardware
  - May not be affordable by everyone
- Operations may be important, but not general enough for standard
- Allows for portably optimal codes
- Pentium's running Linux

# Code Generation Strategy

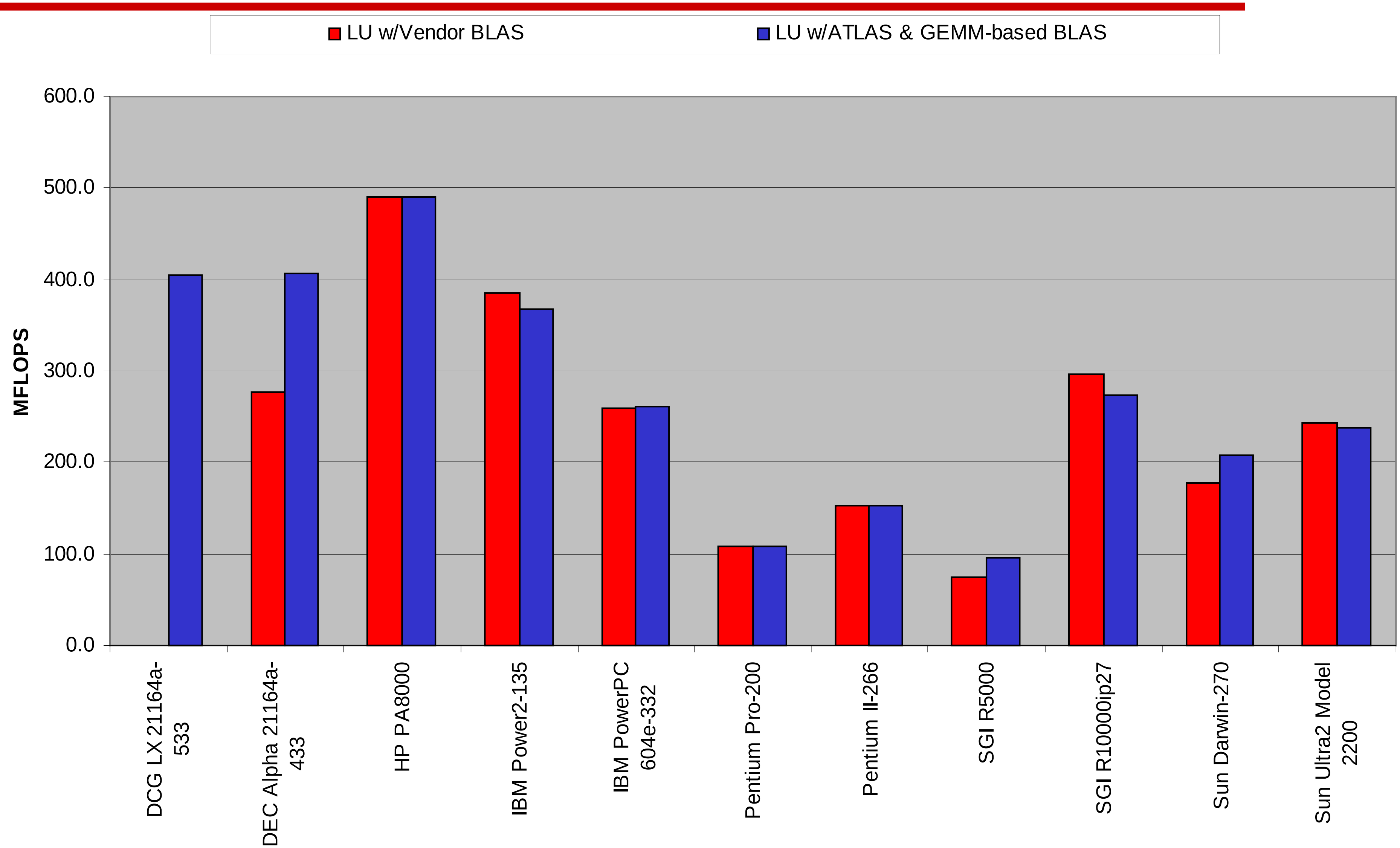



- Cache based multiply optimizes for:
  - TLB access
  - L1 cache reuse
  - FP unit usage
  - Memory fetch
  - Register reuse
  - Loop overhead minimization
- Takes a couple of hours to run.
- Code is iteratively generated & timed until optimal case is found. We try:
  - Differing NBs
  - Breaking false dependencies
  - M, N and K loop unrolling

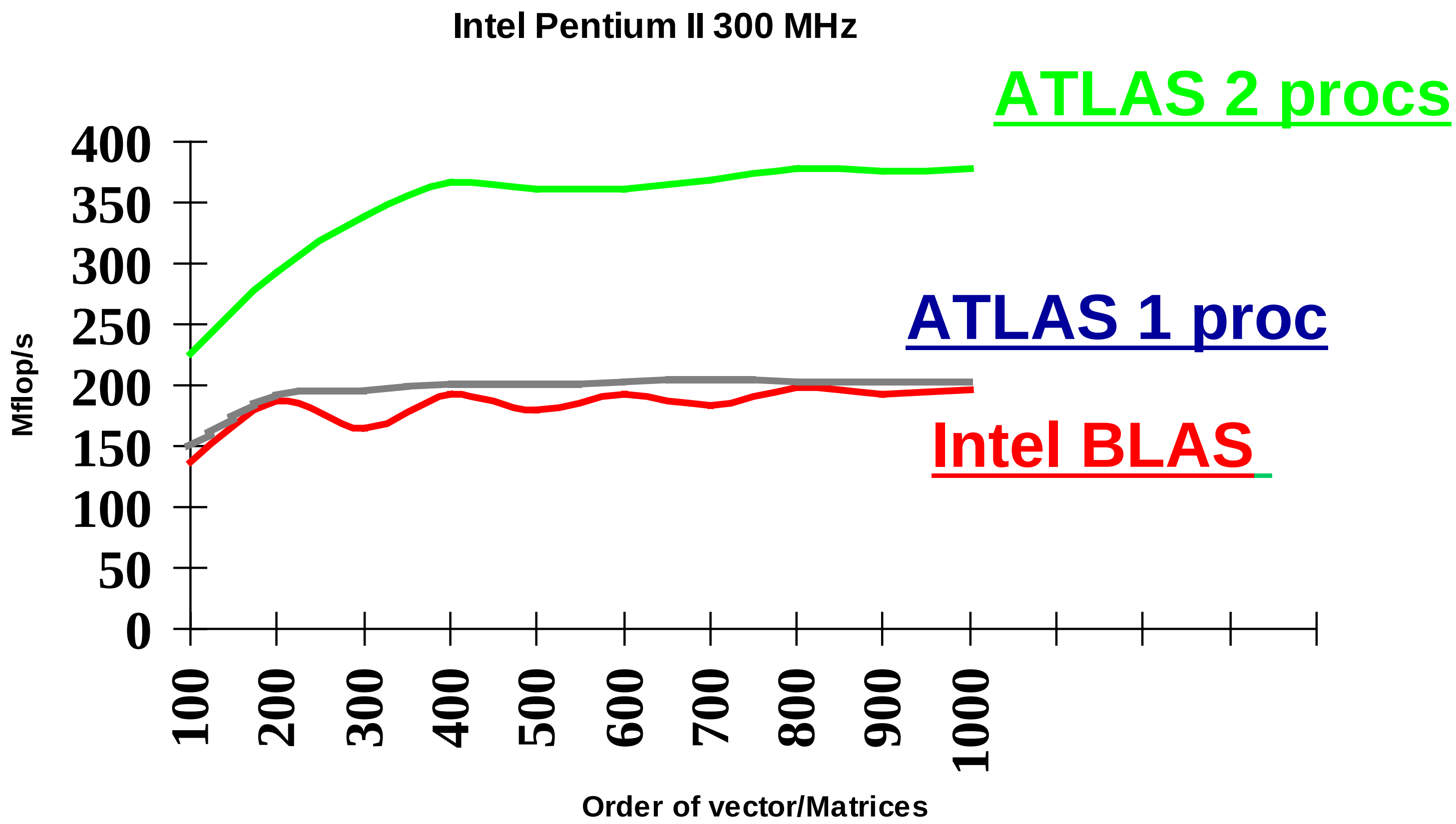
# 500x500 Double Precision Matrix-Matrix Multiply Across Multiple Architectures



## 500 x 500 Double Precision LU Factorization Performance Across Multiple Architectures



# Multithreaded BLAS for Performance



# ATLAS

- Keep a repository of kernels for specific machines.
- Develop a means of dynamically downloading code
- Extend work to allow sparse matrix operations
- Extend work to include arbitrary code segments
- See: http://www.netlib.org/atlas/

# What is needed to use ATLAS with Java:

1. The Java calling program (jdmmtst.java)
2. The Java loadLibrary class (CWRAP.java)
3. The java loadLibrary header file (CWRAP.h)
4. The C functions (c wrappers to call ATLAS and ATLAS c software library).
5. The shared object (libCWRAP.so) which contains executable binaries of the C wrapper functions and all the ATLAS functions.

# Test Results for Matrix Multiply UltraSparc 2200 (200 MHz, peak of 400 Mflop/s)

| Array Size M=N=K | Pure Java MM Mflops | From SUN Perflib Mflops | C call to ATLAS DGEMM Mflops | Java call to ATLAS DGEMM Mflops |
|---|---|---|---|---|
| 100 | 1.03 | 335.3 | 271.6 | 246.4 |
| 200 | 1.00 | 310.5 | 288.5 | 249.9 |
| 300 | 0.99 | 295.4 | 290.7 | 286.4 |
| 400 | 0.97 | 273.4 | 290.8 | 276.0 |
| 500 | 0.98 | 277.5 | 290.9 | 282.5 |

# Futures

- Extend these ideas to Java directly

- References:

http://www.netlib.org/

http://www.netlib.org/atlas/

http://www.cs.utk.edu/f2j/

http://www.netlib.org/utk/people/JackDongarra/