How to Write Fast Code

18-645, spring 2008
24th Lecture, Apr. 14th

Guest Lecturer: *Daniel McFarlin*

Instructor: Markus Püschel

TAs: Srinivas Chellappa (Vas) and Frédéric de Mesmay (Fred)

How to Write Fast “High-Level” Code

Productivity VS. Performance

- Tradeoffs
- Agility vs. Robustness
- Continuum of PLs
- New PL "Sweetspot"
- OpenMP C/Fortran
  - Chapel, Fortress, X10
- *Hybrid Systems*

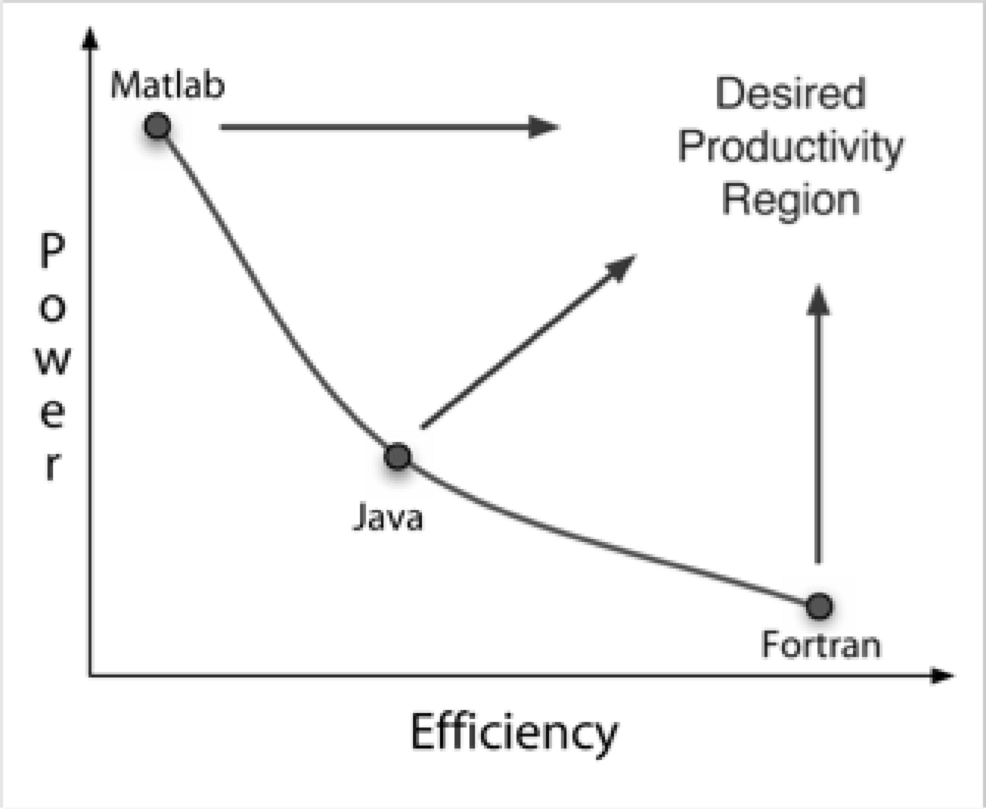


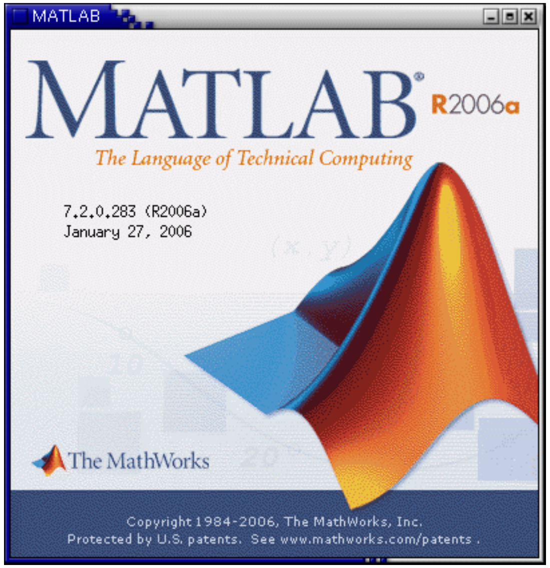
Figure 1. Power-efficiency graph.

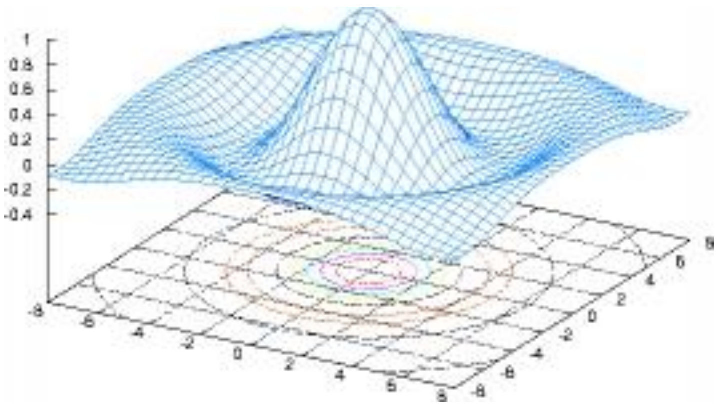
Hybrid Systems

- Interfacing High-Level Languages with High-Performance libraries
- A Brief HLL History
- HLL Implementations
- Programmatic Interfaces
- Best Practices
- Case Study

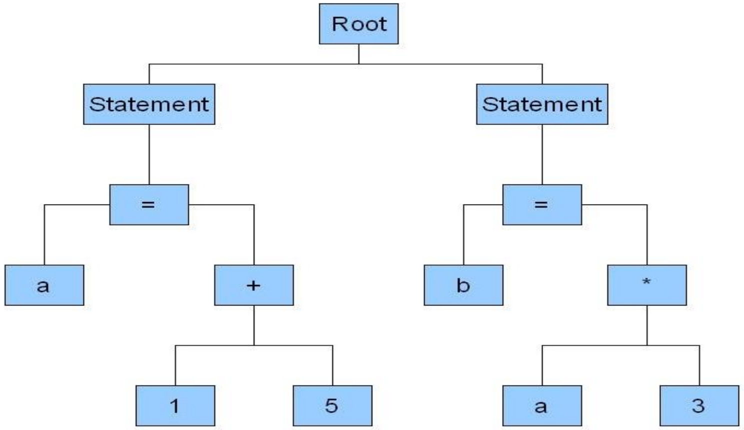
A Brief History of High Level Languages

- Both Octave and MATLAB were designed in the 80's as high level interfaces to LINPACK
- Fundamental datatype is the matrix
- Syntactically similar
- Near mutual compatibility (syntactic sugar/toolkits)





HLL Implementations



- Interpreters that traverse the AST representation of the input
- May have to "pointer-chase" through AST data structure
- MATLAB operates on a linearizedopcode representation which is JIT compiled
- Operations on C/C++ fundamental type: mxArray/octave_value
- Use hash tables to maintain identifiers

Implementations continued…

- **Most overhead associated with:**
  - run-time type identification
  - boxing/unboxing
  - operator overloading
  - identifier resolution
  - garbage collection

- **A * B → mult(A,B) → mm_mult(unbox(A), unbox(B)) → gemm(A,B)**

- **Most functionality embedded in library calls (MATLAB/Octave or BLAS/LAPACK etc)**

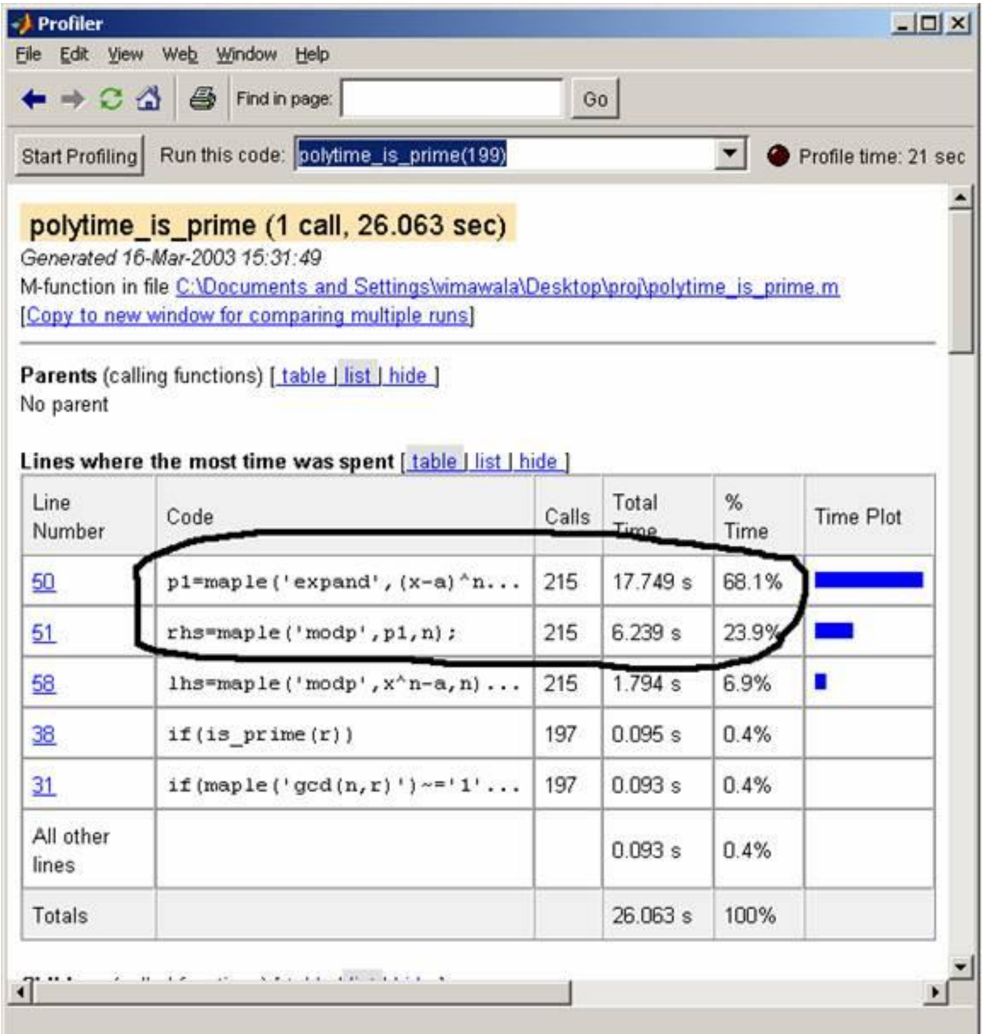
MATLAB Best Practices

- Pre-allocate (zeros, cell, matrix)
- Select appropriate intrinsic type
- Prefer vector constructs over looping
- Avoid global
- Avoid dangling-reference induced memory leaks
- Avoid excessive branching and input argument modification
- Use in-place functions

Profiling MATLAB

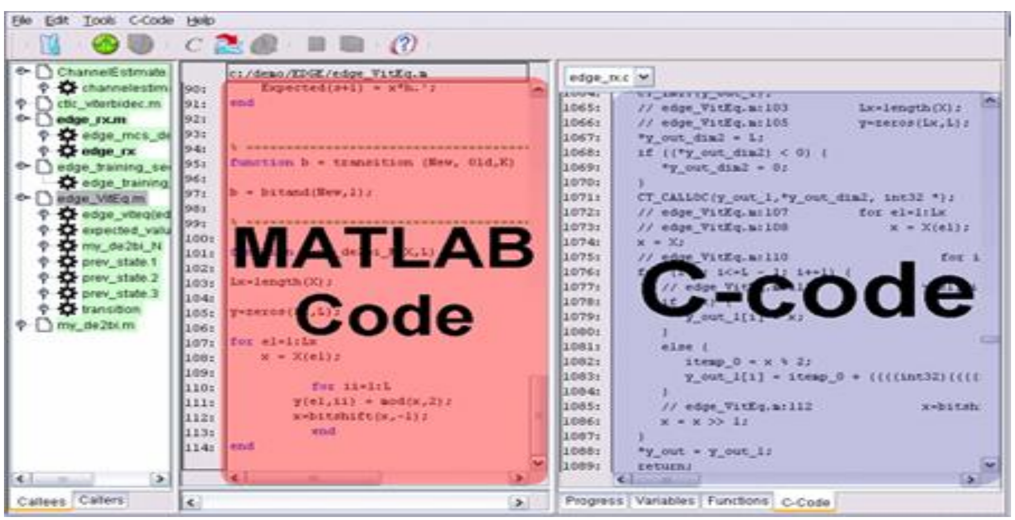
- Initial MATLAB implementation
- Use tic and toc for coarse grain wallclock timing
- Use cputime for finer grain timing measurements
- Use MATLAB profiler for gprof-like profiling information

MATLAB Profiler DEMO



Interfaces

- Identified hotspot(s)
- Optimized MATLAB implementation
- Want to incorporate optimized "low-level" code
  - Compiler (Catalytic, Polaris, ParaM, Star-P)



What about the MATLAB compiler, mcc?

- **Once upon a time....**
  - Could actually see mxArray manipulations
  - Or ... at least library calls with mxArray inputs
- **Now used for portable deployment**
  - Embed M-code in exe
  - Embed JIT-accelerator, interpreter and support libraries into exe
  - Result: no speedup
- **Can still auto-generate header file for external functions**
  - Use %#external
  - Static linkage of external functions

Interfaces…

- **Incorporate optimized C/C++ code directly into the interpreter (Octave only)**
  - Source code is fairly readable

- **MEX/Octfile**
  - Octave now supports the MEX interface
  - DLLs loaded at call time
  - Explicitly box/unbox input/output arguments
  - All of your C/C++ optimization knowledge is useful but…
    - Must be aware of DLL interface pitfalls

Interface pitfalls

- **Underlying DLL overhead**
  - Mostly unavoidable but there is extensive documentation on how to extract some performance improvement (Drepper 2006)
- **MATLAB/Octave DLL function calls are about two orders of magnitude slower than C function calls**
  - Argument resolution/unboxing
  - Determining which function to call (.m or DLL)
  - Possibly reloading or unloading the DLL
- **Bottom Line: push all functionality into a single DLL**
  - Ideally into a single function
    - MATLAB Limitation: only one function per DLL
    - Octave: any number of functions but have to use symlinks to because DLLs are opened based on name
  - Avoid calling DLL functions in loops

MEX File Optimizations

- **Slab Allocations**
  - Requires logic and state in the library
- **Input argument mangling**
  - Semi-endorsed by Mathworks

Case Study: Synthetic Aperture Radar

- **Interpolate.m vs. interpolaton55.c**
  - Partial loop unrolling
  - Computer generated vectorization
  - Loop merging
  - Iteration space transposition

References

- http://www.youtube.com/watch?v=lDPLy7MyDMY
- http://people.redhat.com/drepper/dsohowto.pdf
- http://blogs.mathworks.com/loren/

Allocation Example

```
function w = test()
x = [1:16];
%% allocation
y = zeros(16,1);
z = zeros(16,1);
w = zeros(16,1);
n =  4;
m = 4;
for i=1:4,
    for j=1:4,
       y(i + 4*(j-1)) = x(4*(i-1) + j);

   end
end
y
z(1:1:4) = y(1:1:4);
for i=2:4,
   z((i-1)*n+1:1:(i-1)*n+n) = y((i-1)*n+1:1:(i-1)*n+n);
end
z
w(1:n:n*(m-1)) = z(1:n:n*(m-1));
w
for i=2:4,
   w((i):n:(i)+n*(m-1)) = z((i):n:(i)+n*(m-1));
End
```

Internal vs. External Looping

```
function [L,ierr] = Chol(A);

      [n,n] = size(A);
      ierr = 0;
   %
      for k = 1:n,
   %
   %    exit if A is not positive definite
   %
        if (A(k,k) <= 0), ierr = k; return; end
   %
   %    Compute main diagonal elt. and then scale the k-th column
   %

         A(k,k) = sqrt(A(k,k));
         A(k+1:n,k) = A(k+1:n,k)/A(k,k);
   %
   %     Update lower triangle of the trailing (n-k) by (n-k) block
   %

         for j = k+1:n,
            A(j:n,j) = A(j:n,j) - A(j:n,k)*A(j,k);
         end
      end
      L = tril(A);
```