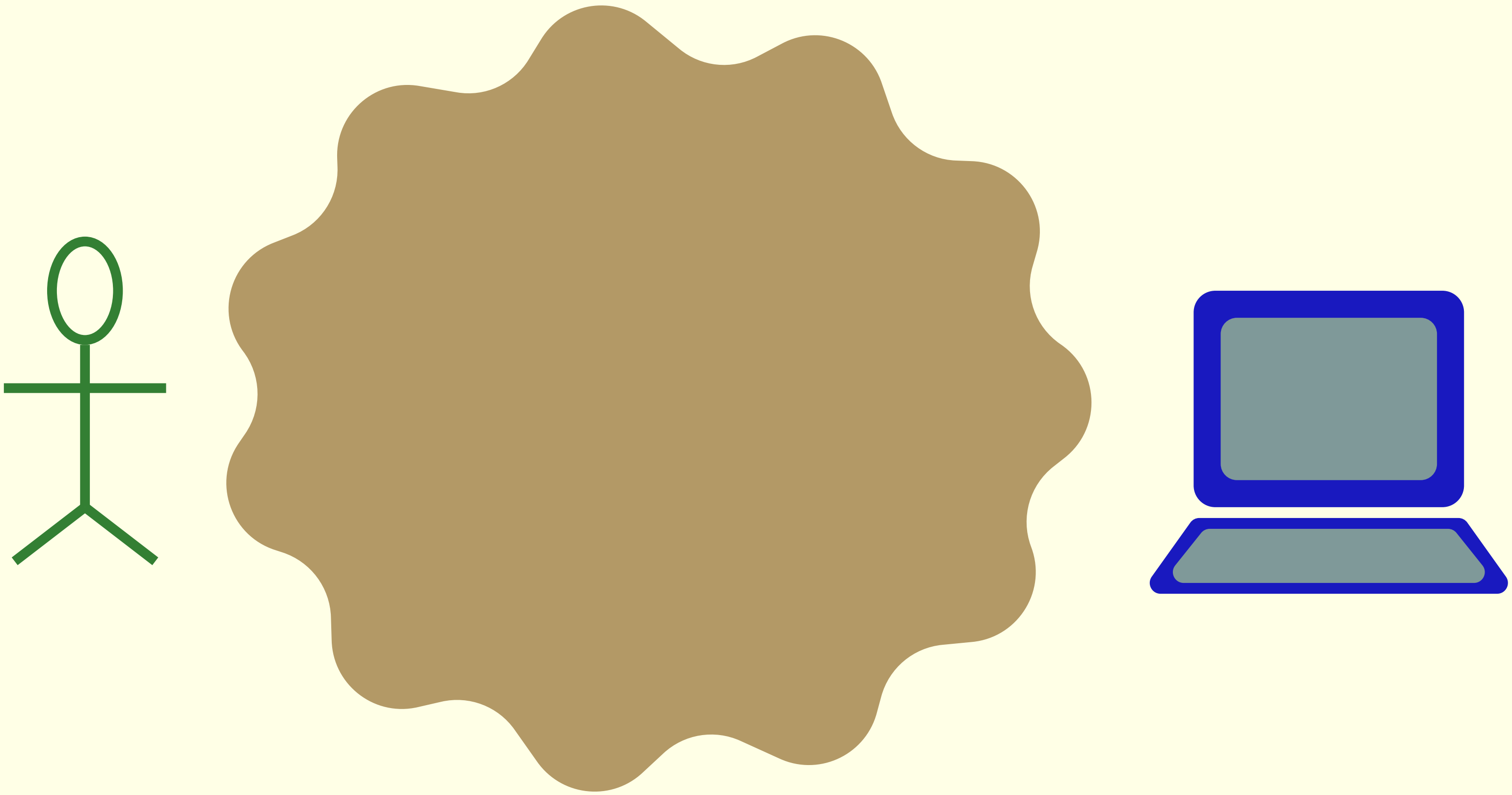
Pondering the Problem of Programmers' Productivity

Are we there yet?

Arun Chauhan

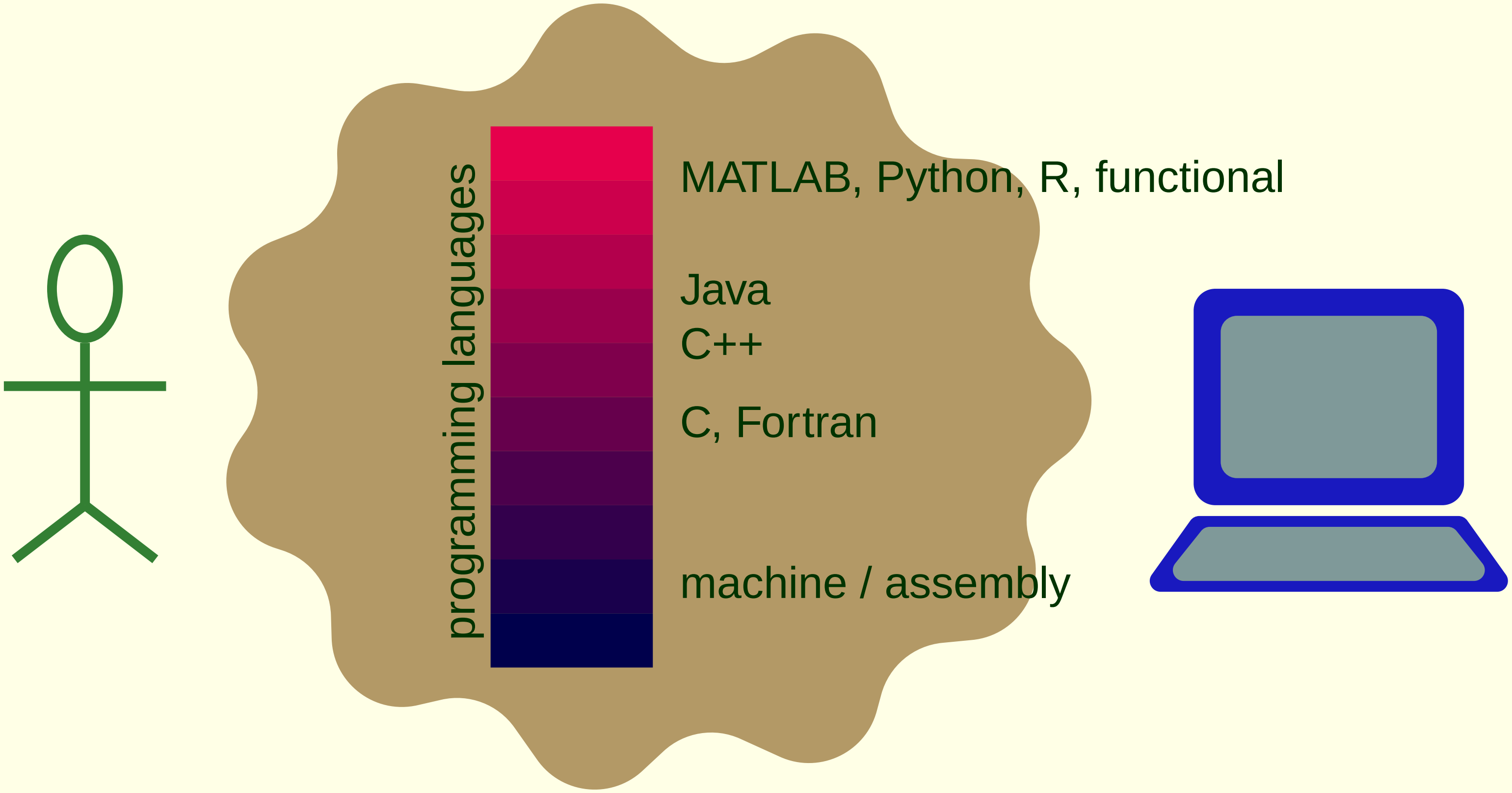
Indiana University

The Big Picture



Human-Computer Interface

The Big Picture



Human-Computer Interface

Widely Recognized Problem

High-end computing is experiencing a full-blown productivity crisis, whose dimensions are only now beginning to be understood. We have entered a period of decreasing productivity as measured by the programming skill required . . . to make a lasting impact on an applications area by repeatedly developing new applications for the new machine.

–Anonymous contributor to HPCwire

DoD has Noticed!

High Productivity Computing Systems (HPCS)

Provide economically viable high productivity computing systems for the national security and industrial user communities with the following design attributes in the latter part of this decade:

- Performance
- Programmability
- Portability
- Robustness

`http://www.darpa.mil/ipto/programs/hpcs/`

Evolution of Languages

- Machine and assembly languages
- Procedural languages (Fortran, Pascal, C)
- Object-oriented languages (Smalltalk, Eiffel, C++, Java, C#)
- Functional / declarative languages (Scheme, Lisp, ML, Prolog)

Evolution of Languages

- Machine and assembly languages
- Procedural languages (Fortran, Pascal, C)
- Object-oriented languages (Smalltalk, Eiffel, C++, Java, C#)
- Functional / declarative languages (Scheme, Lisp, ML, Prolog)
- Domain-specific languages
  - MATLAB, Perl, Python, S+, Mathematica, Maple
  - MATLAB has over 500,000 licenses worldwide

Evolution of Languages

- Machine and assembly languages
- Procedural languages (Fortran, Pascal, C)
- Object-oriented languages (Smalltalk, Eiffel, C++, Java, C#)
- Functional / declarative languages (Scheme, Lisp, ML, Prolog)
- Domain-specific languages
  - MATLAB, Perl, Python, S+, Mathematica, Maple
  - MATLAB has over 500,000 licenses worldwide
  - also called “little languages”
  - APL, SQL

Evolution of Languages

- Machine and assembly languages
- Procedural languages (Fortran, Pascal, C)
- Object-oriented languages (Smalltalk, Eiffel, C++, Java, C#)
- Functional / declarative languages (Scheme, Lisp, ML, Prolog)
- Domain-specific languages
  - MATLAB, Perl, Python, S+, Mathematica, Maple
  - MATLAB has over 500,000 licenses worldwide
  - also called “little languages”
  - APL, SQL

Scripting Languages

A 25-year Old Problem

[T]he next revolution in programming will take place only when *both* of the following requirements have been met: (a) a new kind of programming language, far more powerful than those of today, has been developed and (b) a technique has been found for executing its programs at not much greater cost than that of today's programs.

–John Backus, the "Father of Fortran"

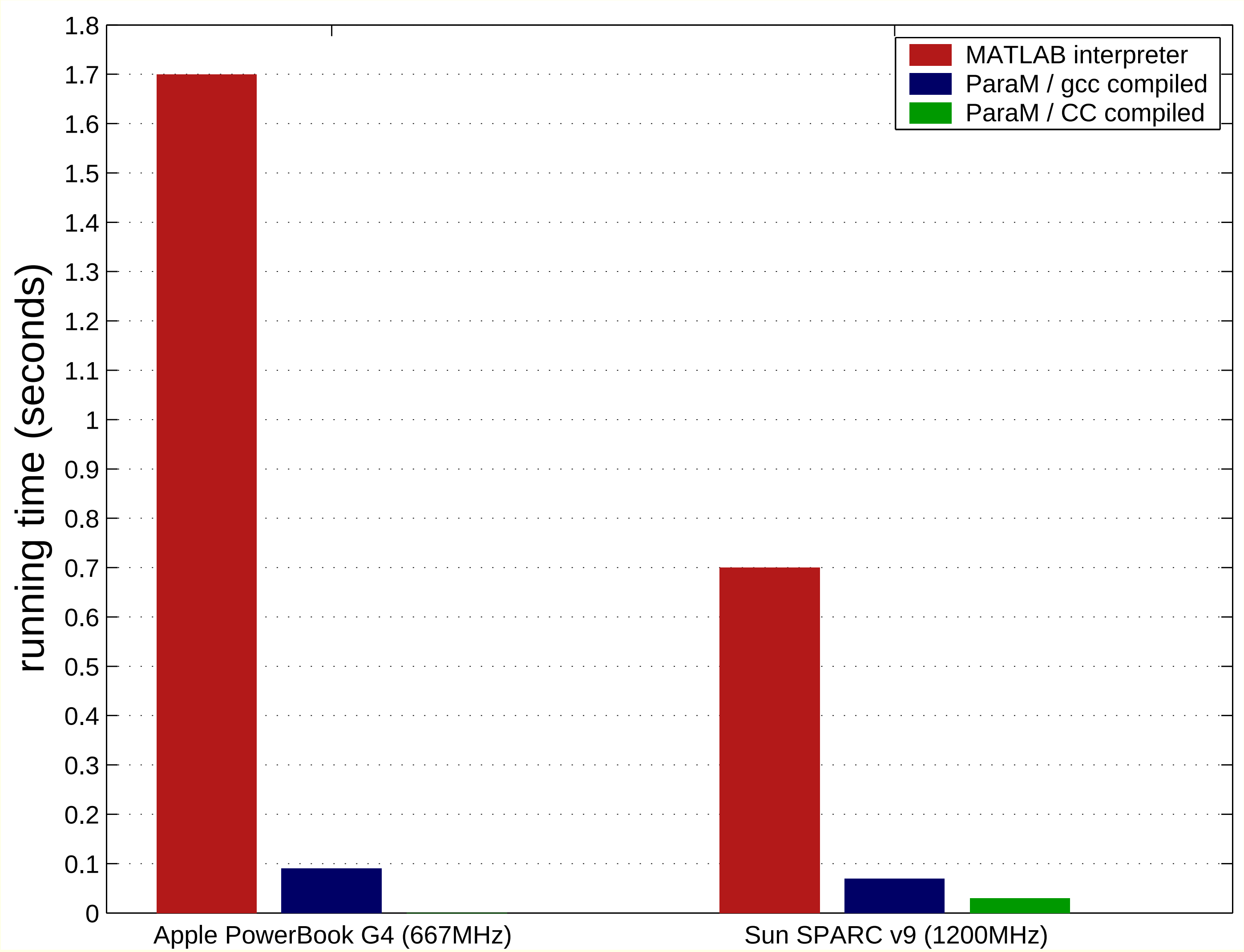
A 25-year Old Problem

[T]he next revolution in programming will take place only when *both* of the following requirements have been met: (a) a new kind of programming language, far more powerful than those of today, has been developed and (b) a technique has been found for executing its programs at not much greater cost than that of today's programs.

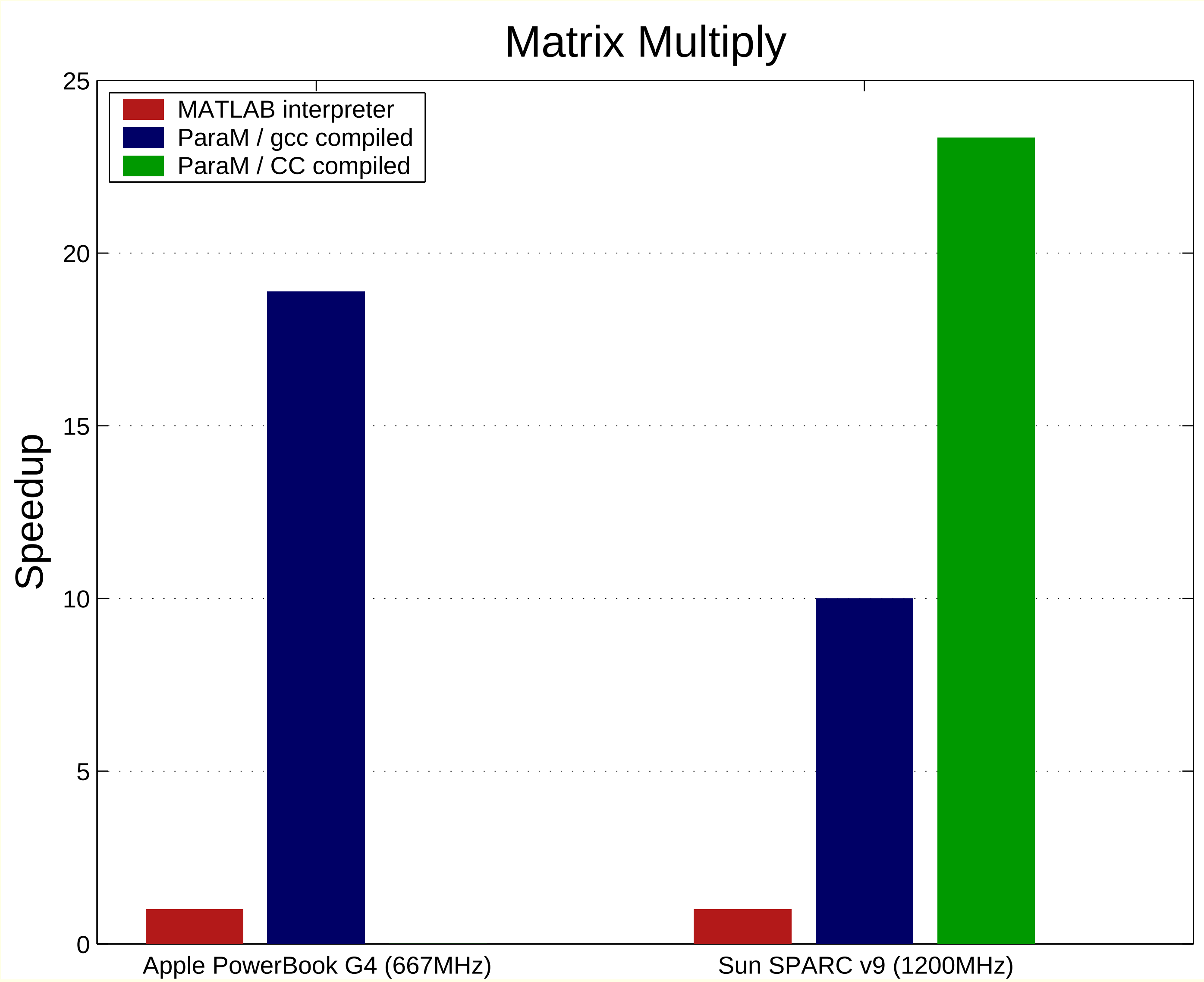
–John Backus, the "Father of Fortran"

Still not satisfactorily solved!

The Performance Gap



The Performance Gap



MATLAB Example

```
function mcc_demo
    x = 1;
    y = x / 10;
    z = x * 20;
    r = y + z;
```

MATLAB Example

```
static void Mmcc_demo (void) {
    ...
    mxArray * r = NULL;
    mxArray * z = NULL;
    mxArray * y = NULL;
    mxArray * x = NULL;
    mlfAssign(&x, _mxarray0_); /* x = 1; */
    mlfAssign(&y, mclMrdivide(mclVv(x, "x"), _mxarray1_)); /* y = x / 10; */
    mlfAssign(&z, mclMtimes(mclVv(x, "x"), _mxarray2_)); /* z = x * 20; */
    mlfAssign(&r, mclPlus(mclVv(y, "y"), mclVv(z, "z"))); /* r = y + z; */
    mxDestroyArray(x);
    mxDestroyArray(y);
    mxDestroyArray(z);
    mxDestroyArray(r);
    ...
}
```

```
function mcc_demo
    x = 1;
    y = x / 10;
    z = x * 20;
    r = y + z;
```

MATLAB Example

```
static void Mmcc_demo (void) {
    . . .
    double r;
    double z;
    double y;
    double z;
    mlfAssign(&x, _mxarray0_); /* x = 1; */
    mlfAssign(&y, mclMrdivide(mclVv(x, ”x”), _mxarray1_)); /* y = x / 10; */
    mlfAssign(&z, mclMtimes(mclVv(x, ”x”), _mxarray2_)); /* z = x * 20; */
    mlfAssign(&r, mclPlus(mclVv(y, ”y”), mclVv(z, ”z”))); /* r = y + z; */
    mxDestroyArray(x);
    mxDestroyArray(y);
    mxDestroyArray(z);
    mxDestroyArray(r);
    . . .
}
```

```
function mcc_demo
    x = 1;
    y = x / 10;
    z = x * 20;
    r = y + z;
```

MATLAB Example

```
static void Mmcc_demo (void) {
    ...
    double r;
    double z;
    double y;
    double z;
    scalarAssign(&x, 1); /* x = 1; */
    scalarAssign(&y, scalarDivide(x, 10)); /* y = x / 10; */
    scalarAssign(&z, scalarTimes(x, 20)); /* z = x * 20; */
    scalarAssign(&r, scalarPlus(y, z)); /* r = y + z; */
    mxDestroyArray(x);
    mxDestroyArray(y);
    mxDestroyArray(z);
    mxDestroyArray(r);
    ...
}
```

```
function mcc_demo
    x = 1;
    y = x / 10;
    z = x * 20;
    r = y + z;
```

MATLAB Example

```
static void Mmcc_demo (void) {
    ...
    double r;
    double z;
    double y;
    double z;
    x = 1; /* x = 1; */
    y = x / 10; /* y = x / 10; */
    z = x * 20; /* z = x * 20; */
    r = y + z; /* r = y + z; */
    /* mxDestroyArray(x); */
    /* mxDestroyArray(y); */
    /* mxDestroyArray(z); */
    /* mxDestroyArray(r); */
    ...
}
```

```
function mcc_demo
    x = 1;
    y = x / 10;
    z = x * 20;
    r = y + z;
```

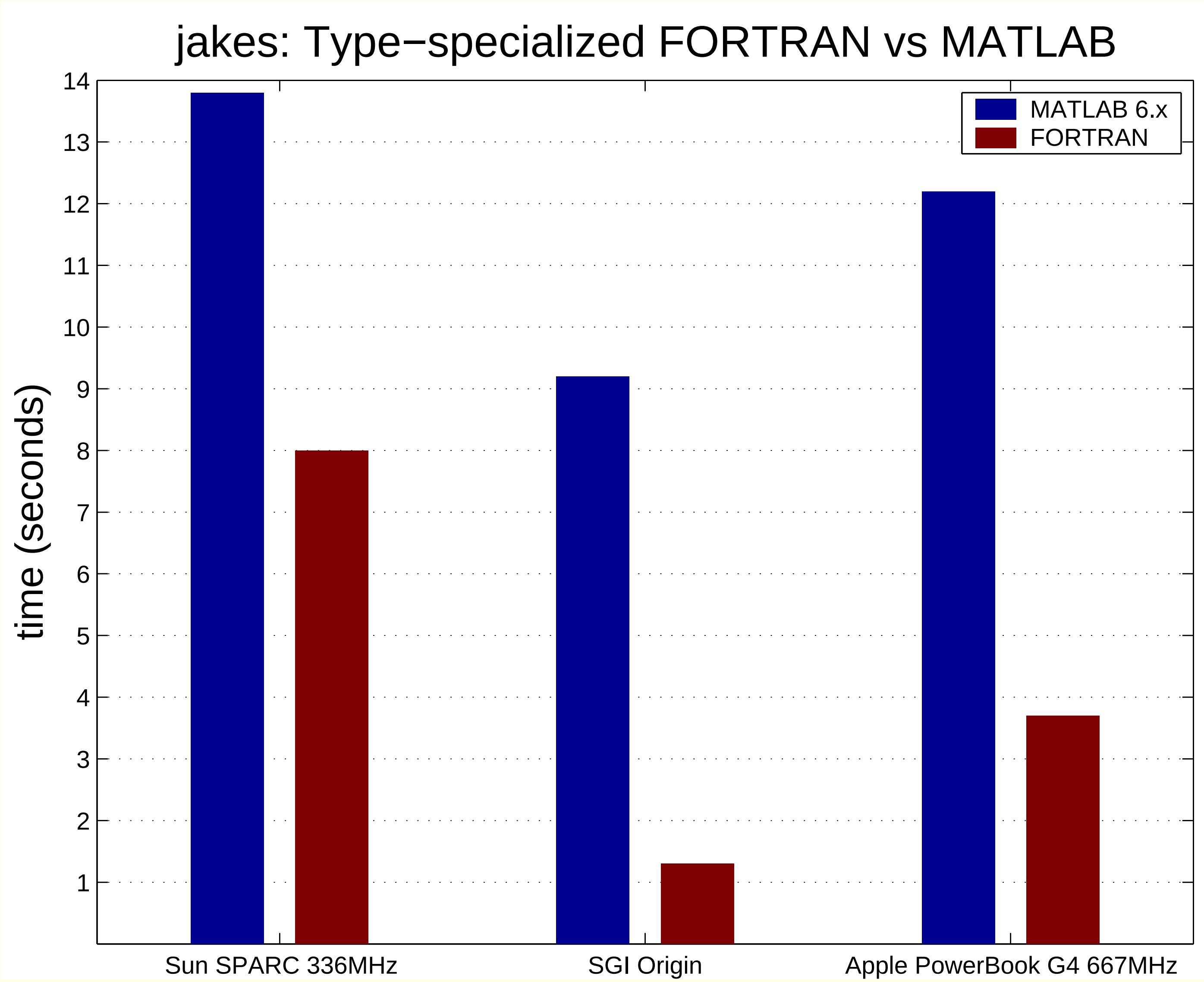
Inferring Types

- type $\equiv <\tau$, δ, σ, $\psi>$
  - $\tau =$ intrinsic type, e.g., int, real, complex, etc.
  - $\delta =$ array dimensionality (or rank), 0 for scalars
  - $\sigma =$ size (or shape), δ-tuple of positive integers
  - $\psi =$ “structure” (or pattern) of an array
- Examples
  - x is scalar, integer

    $\Rightarrow$ type of x $= <$int, 0, $\perp$, $\perp>$
  - y is 3-D $10 \times 5 \times 20$ dense array of reals

    $\Rightarrow$ type of y $= <$real, 3, $<$10,5,20$>$, dense$>$

Type-based Specialization



Fundamental Observation

- Libraries are the key in optimizing high-level scripting languages

`a = x * y` $\Rightarrow$ `a = mclMtimes(x, y)`

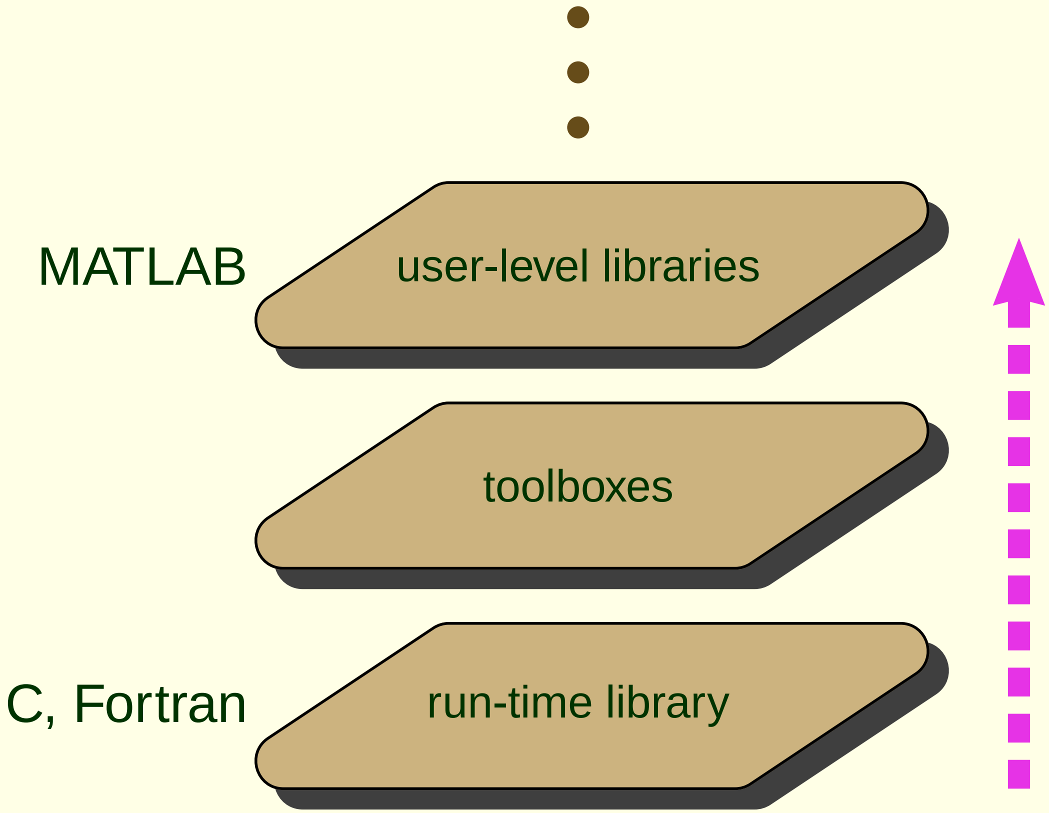
Fundamental Observation

- Libraries are the key in optimizing high-level scripting languages

`a = x * y` $\Rightarrow$ `a = mclMtimes(x, y)`

- Libraries practically **define** high-level scripting languages
  - high-level operations are often "syntactic sugar"
    - runtime libraries implement operations
  - a large effort in HPC is toward writing libraries
  - domain-specific libraries make scripting languages useful and popular

Hierarchy of Libraries



Domain Algebra

$$\sin^2(\theta) + \cos^2(\theta) \equiv 1$$

$$\tan^2(\theta) + 1 \equiv \sec^2(\theta)$$

$$\tan(\theta) \equiv \frac{\sin(\theta)}{\cos(\theta)}$$

$$\sin(2\theta) \equiv 2\sin(\theta)\cos(\theta)$$

$$\cos(2\theta) \equiv \cos^2(\theta) - \sin^2(\theta)$$

$$\ldots$$

Domain Algebra

$$\sin^2(\theta) + \cos^2(\theta) \equiv 1$$

$$\tan^2(\theta) + 1 \equiv \sec^2(\theta)$$

$$\tan(\theta) \equiv \frac{\sin(\theta)}{\cos(\theta)}$$

$$\sin(2\theta) \equiv 2\sin(\theta)\cos(\theta)$$

$$\cos(2\theta) \equiv \cos^2(\theta) - \sin^2(\theta)$$

$$\ldots$$

. . . and beyond

$$\begin{array}{l} x = \sin(\theta) \\ y = \cos(\theta) \end{array} \equiv \quad [x, y] = \text{sincos}(\theta)$$

Domain Algebra

$$\sin^2(\theta) + \cos^2(\theta) \equiv 1$$

$$\tan^2(\theta) + 1 \equiv \sec^2(\theta)$$

$$\tan(\theta) \equiv \frac{\sin(\theta)}{\cos(\theta)}$$

$$\sin(2\theta) \equiv 2\sin(\theta)\cos(\theta)$$

$$\cos(2\theta) \equiv \cos^2(\theta) - \sin^2(\theta)$$

$$\ldots$$

. . . and beyond

$$\begin{array}{l} x = \sin(\theta) \\ y = \cos(\theta) \end{array} \equiv [x, y] = \mathrm{sincos}(\theta)$$

Library Identities

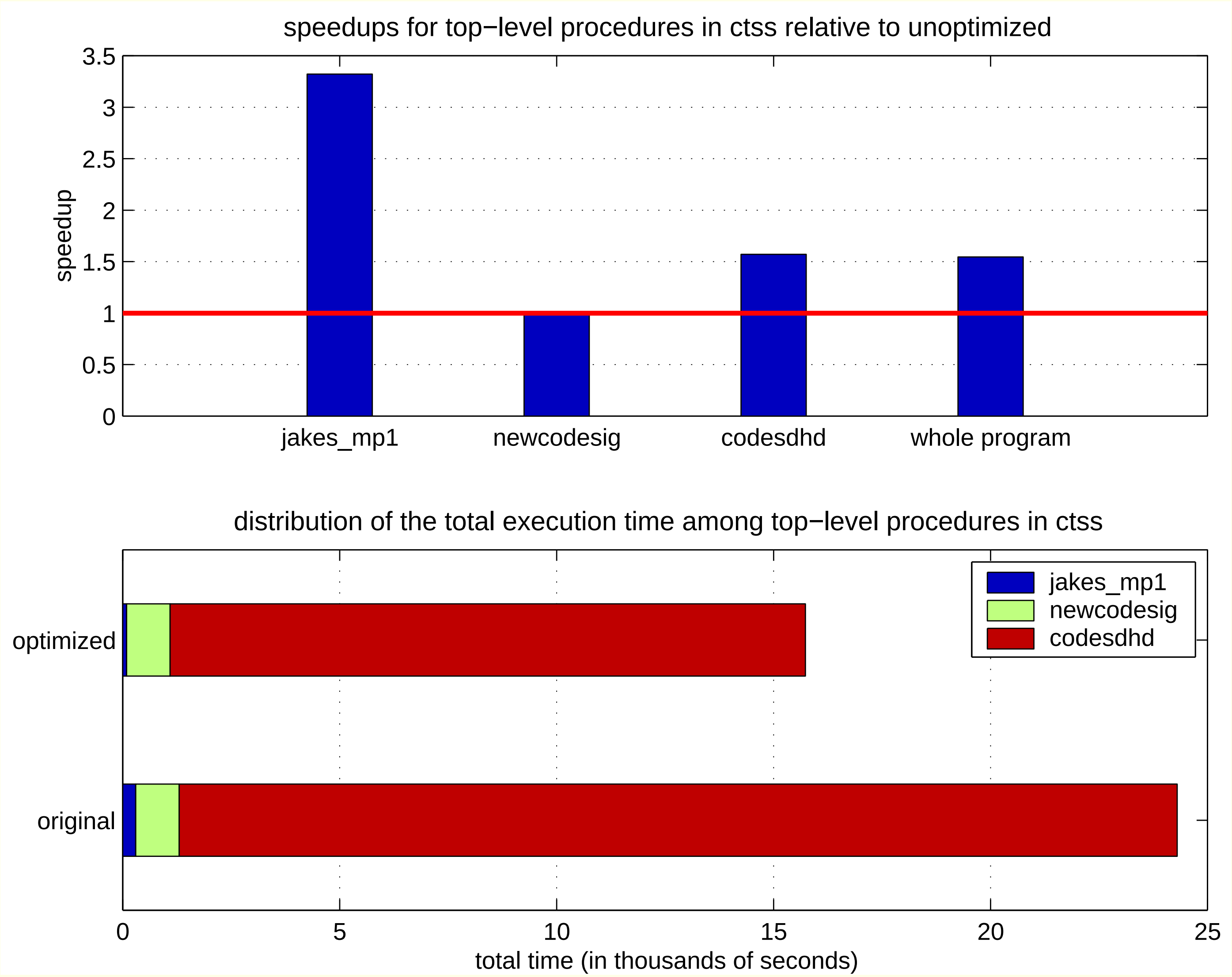
Procedure Strength Reduction

```
for i = 1:N
    ...
    f (c1, c2, i, c3);
    ...
end
```

→

```
f_init (c1, c2, c3);
for i = 1:N
    ...
    f_iter (i);
    ...
end
```

Speedup by PSR



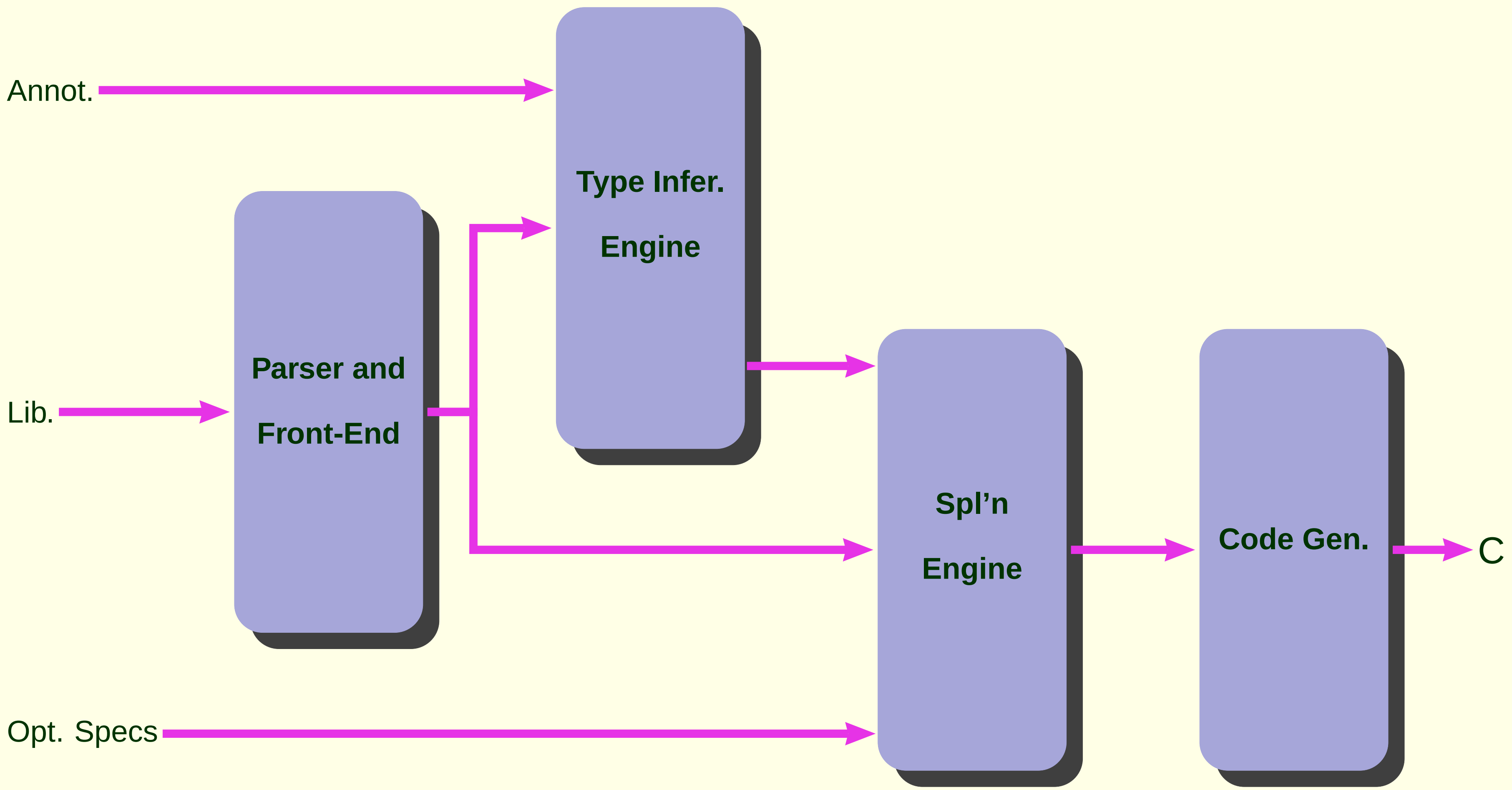
Open Issues

- Language to express identities
- Developing a cost metric
- Techniques to exploit identities
- Automatic discovery of identities
- Effect on compilation time

System Building

- Research through building systems
- Systems expose several unforeseen problems
- Real-life applications stress-test research ideas
- Give a real product to users—fruits of research

MATLAB Compilation System



Why there isn't a Parallel MATLAB

It doesn't make good business sense for us to undertake fundamental changes in MATLAB's architecture. There are not enough potential customers with parallel machines.

–Cleve Moler, Chairman and Chief Scientist
The MathWorks

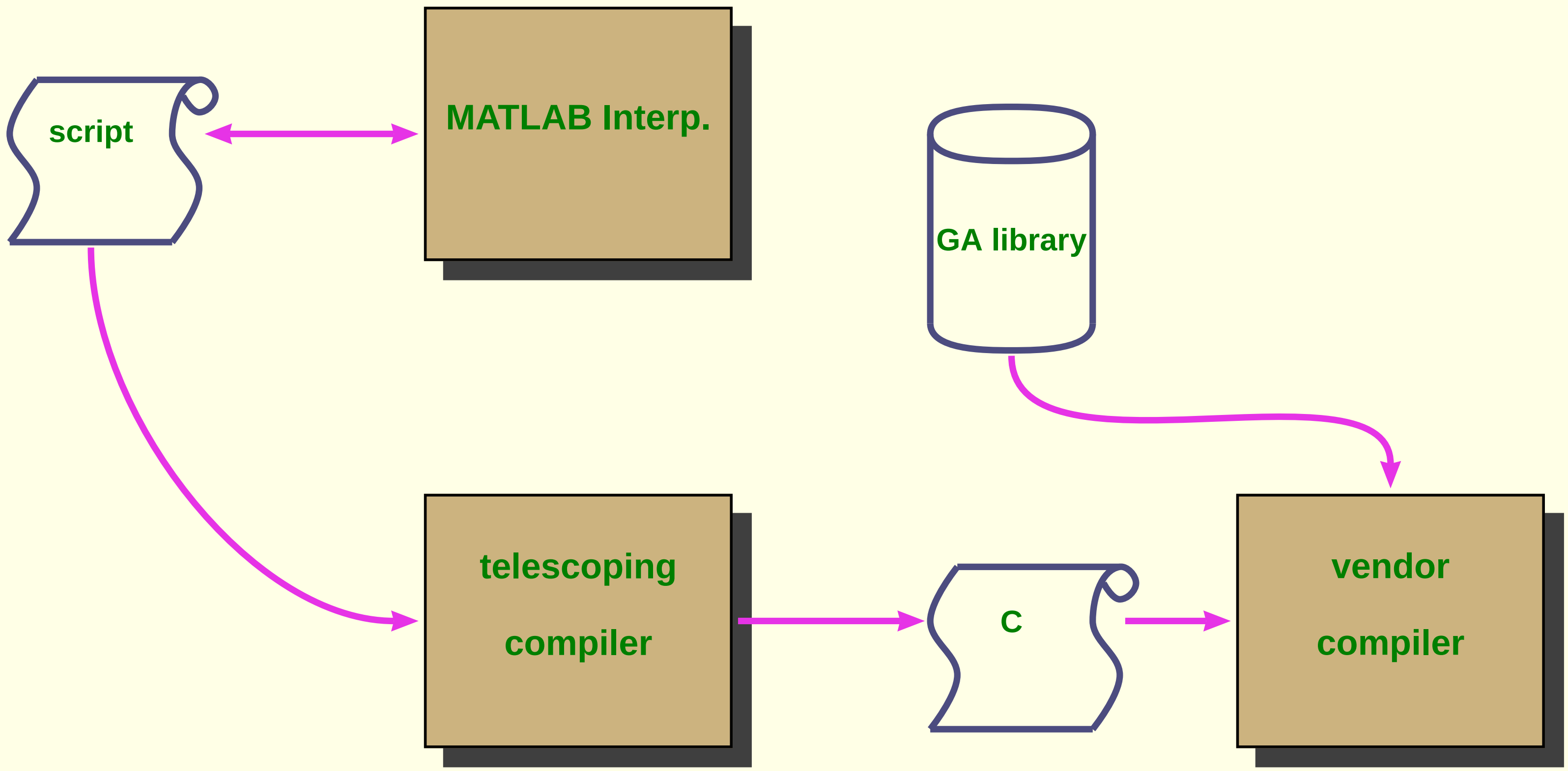
Improving the Performance of MATLAB

		compilation	
		no	*yes*
parallelization	*no*	MATLAB	FALCON, MaJIC, MATCH, Telescoping Languages, CONLAB, Otter, MENHIR
	yes	MATLAB*p, multiMATLAB, pMATLAB	**proposed**

ParaM
Funded by Ohio Supercomputing Center

- Collaborators (P.I.s)
  - Ashok Krishnamurthy (Ohio Supercomputing Center)
  - P. Sadayappan (Ohio-State University)
- Technical Collaborators
  - Ken Kennedy (Rice University)
  - Jarek Nieplocha (Pacific-Northwest National Lab)

ParaM: Architecture



Challenges

- Performance bottlenecks in using the Global Array abstraction
- Automatic analysis to extract parallelism at suitable granularity
- Data distribution analysis
- Working with parallel libraries
- Extending to grid-computing environment

Research Problems

- Library identities
  - language, rewriting systems, theorem proving, databases
- Evolving library compilation system
  - databases, machine learning, AI
- Data-distribution based library-aware automatic parallelization
  - parallel and dist'd computing, runtime systems, algorithms
- Automatic adaptation to dynamic environments
  - AI, runtime systems, grid computing, algorithms

Conclusions

- Scripting or Domain-Specific Languages have gained enormous popularity
  - enable users to program easily in their domains of interest
- Key compilation technologies needed to take these languages beyond protoyping tools
- Library-centric approach shows a lot of promise

Conclusions

- Scripting or Domain-Specific Languages have gained enormous popularity
  - enable users to program easily in their domains of interest
- Key compilation technologies needed to take these languages beyond protoyping tools
- Library-centric approach shows a lot of promise

Not yet, but we are getting close!