Parallelizing MATLAB

Arun Chauhan

Indiana University

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

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

```
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);
    . . .
}
```

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

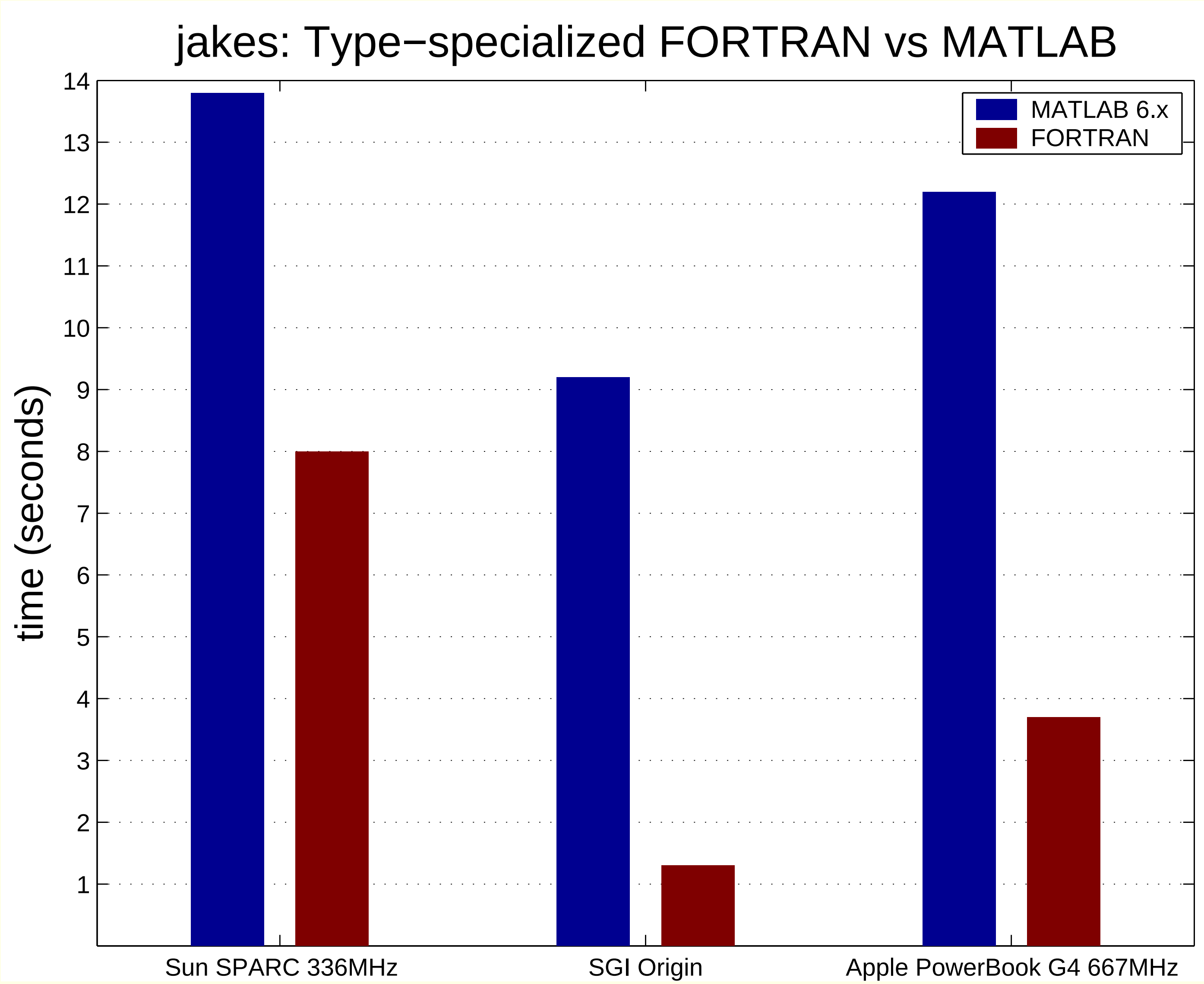
- type $\equiv <\tau, \delta, \sigma, \psi>$
  - $\tau =$ intrinsic type, e.g., int, real, complex, etc.
  - $\delta =$ array dimensionality (or rank), 0 for scalars
  - $\sigma =$ size (or shape), δ-tuple of positive integers
  - $\psi =$ "structure" (or pattern) of an array

- Examples
  - x is scalar, integer

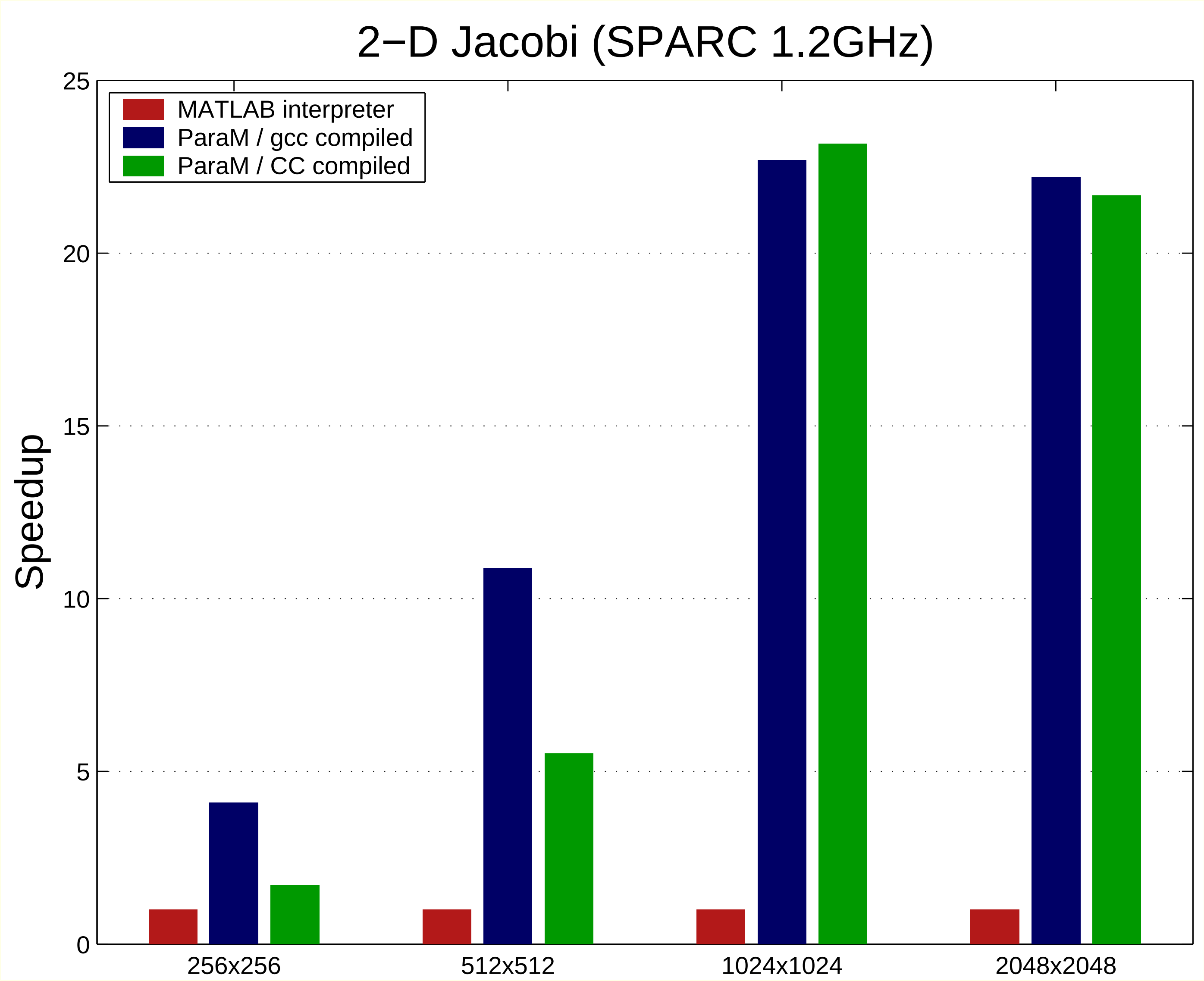
    $\Rightarrow$ type of x $= <$int, 0, $\perp$, $\perp>$
  - y is 3-D $10 \times 5 \times 20$ dense array of reals

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

Type-based Specialization



Type-based Specialization



Fundamental Observation

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

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

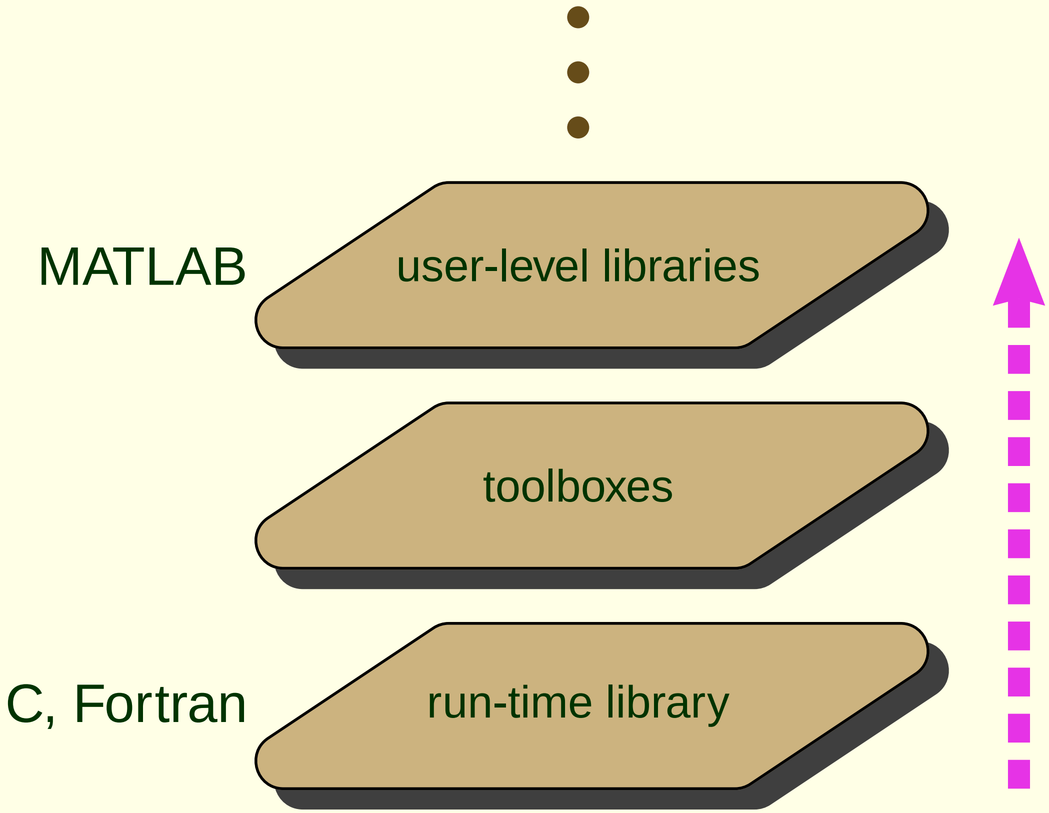
Fundamental Observation

- 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

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

Domain Algebra

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

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

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

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

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

$$\dots$$

. . . and beyond

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

Library Identities

Procedure Strength Reduction

for i = 1:N

  . . .

  f (c_1, c_2, i, c_3);

  . . .

end

→

f_init (c_1, c_2, c_3);

for i = 1:N

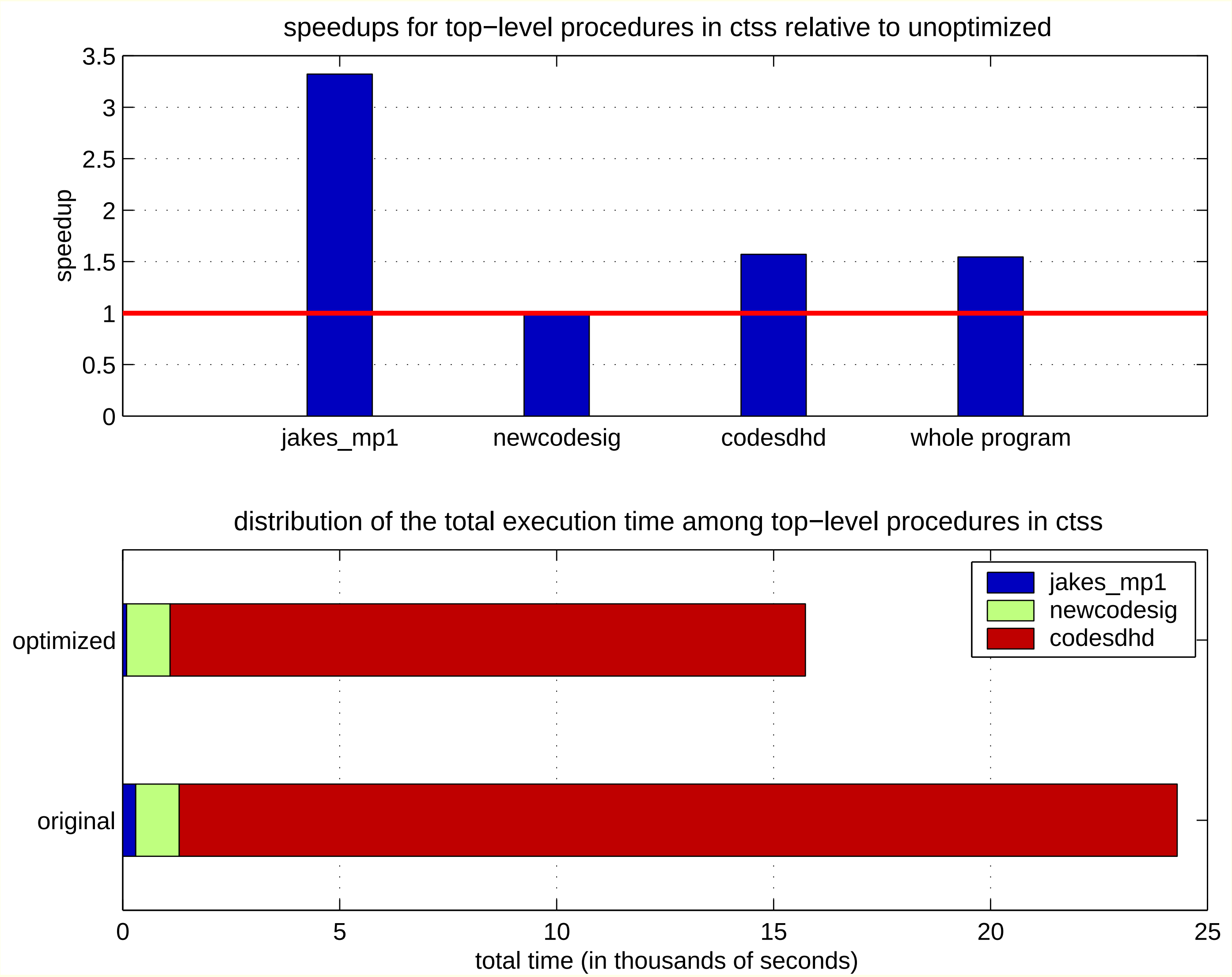
  . . .

  f_iter (i);

  . . .

end

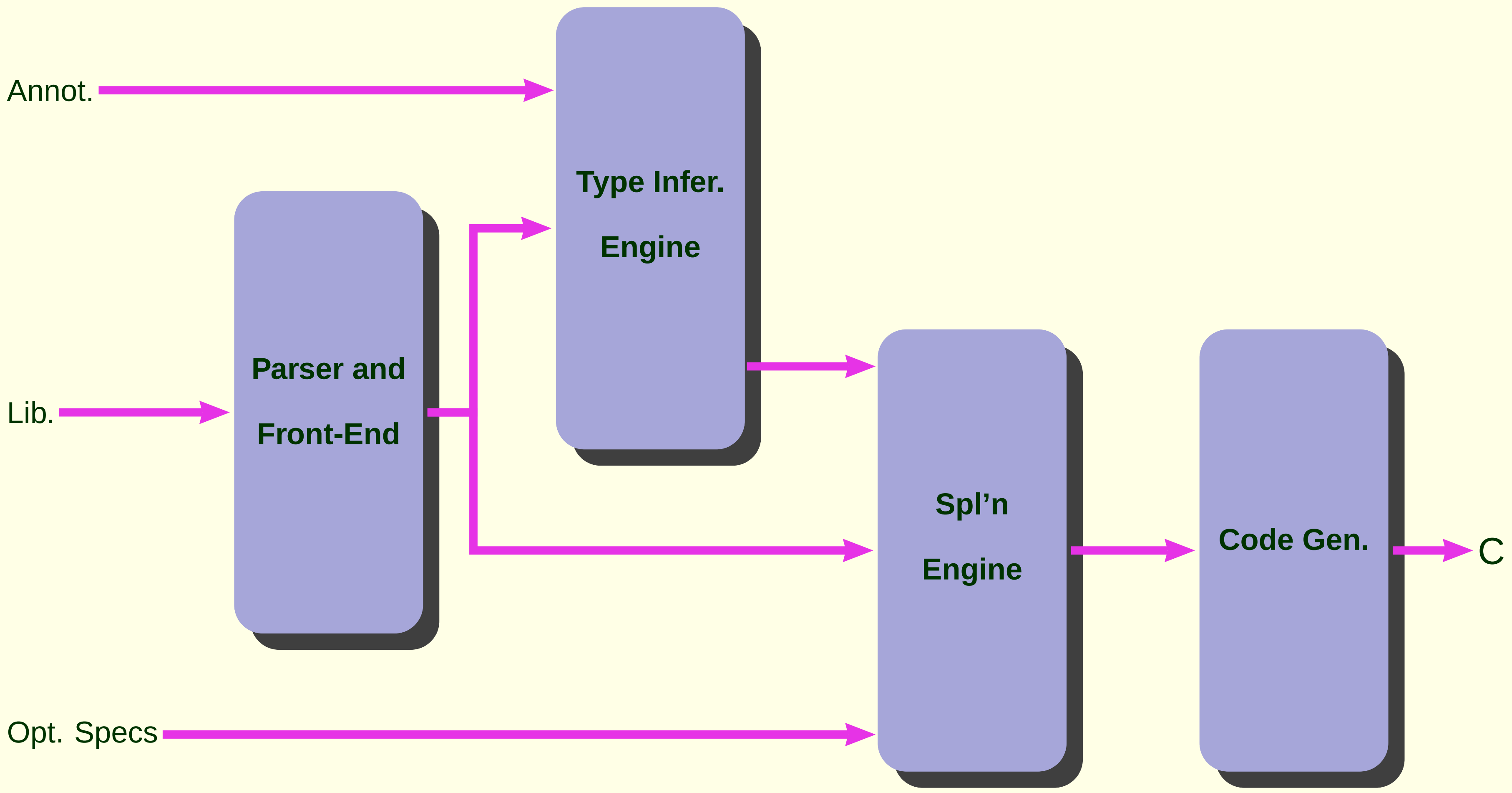
Speedup by PSR



Open Issues

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

MATLAB Compilation System



Improving the Performance of MATLAB

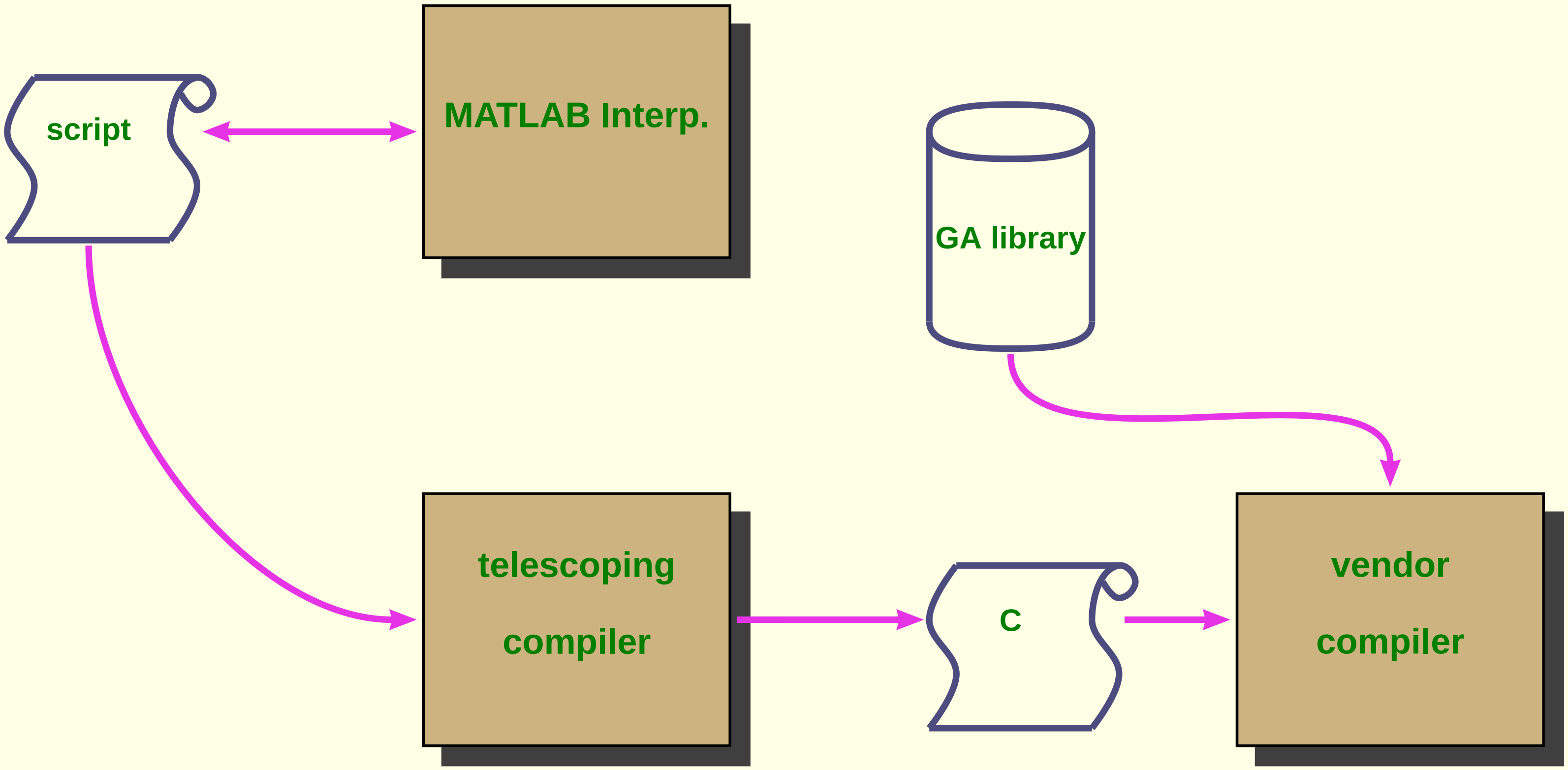
		compilation	
		no	*yes*
parallelization	*no*	MATLAB	FALCON, MaJIC, MATCH, Telescoping Languages, CONLAB, Otter, MENHIR
	yes	MATLAB*p, multiMATLAB, pMATLAB	**proposed**

ParaM

Sponsored 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



Issues

- Performance evaluation of Global Array abstraction
- Automatic analysis to extract parallelism at suitable granularity
- Data distribution analysis
- Working with parallel libraries

http://www.cs.indiana.edu/~achauhan/ParaM/