

Declarative Infrastructure for Automated Scientific Computing

Matthew Rocklin

March 5th, 2013

Domain knowledge is important for efficient computation

Expertise

Most scientific programmers aren't experts in all requisite domains

Expertise

// A: Naive

```
int fact(int n){  
    if (n == 0)  
        return 1;  
    return n*fact(n-1);  
}
```

// B: Programmer

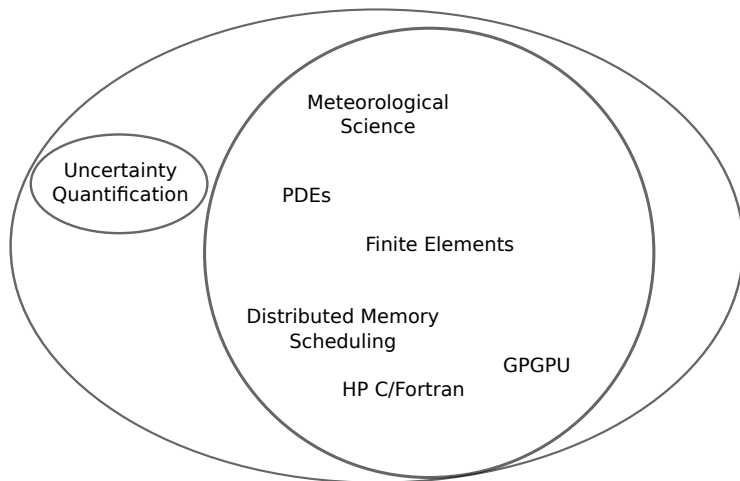
```
int fact(int n){  
    int prod = n;  
    while(n--)  
        prod *= n;  }  
    return prod;  
}
```

// C: Mathematician

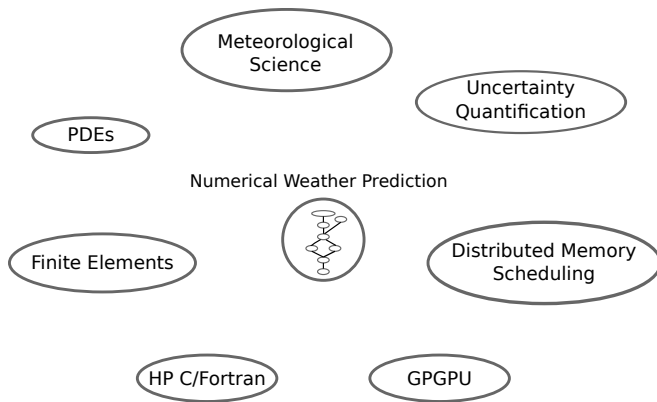
```
int fact(int n){  
    //  $n! = \text{Gamma}(n+1)$   
    return lround(exp(lgamma(n+1)))
```

- ▶ Modern compilers automate B
- ▶ Humans do C by hand
- ▶ The people who know C are rarely trained to build compilers
- ▶ Both B and C are commonly necessary within one project

Numerical Weather Prediction



Composable Software - Composable



Outline

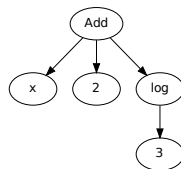
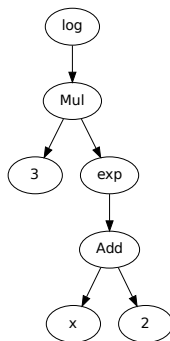
- ▶ Probability Modeling
 - ▶ Bayesian inference simulations (MCMC)
- ▶ Matrix algebra
 - ▶ BLAS/LAPACK computations
- ▶ Static Scheduling
 - ▶ Blocked Matrix Algorithms

Section 1

Probability Modeling

Computer Algebra - SymPy

```
>>> expr = log(3*exp(x + 2))  
>>> print simplify(expr)  
x + 2 + log(3)
```



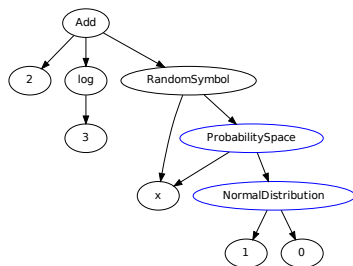
Random Variables

```
>>> x = Normal('x', 0, 1)
```

```
>>> expr = log(3*exp(x + 2))
```

```
>>> print simplify(expr)
```

```
x + log(3) + 2
```



Random Variables

```
>>> x = Normal('x', 0, 1)

>>> expr = log(3*exp(x + 2))
>>> print simplify(expr)
x + log(3) + 2

>>> P(expr > 4)
```

$$\int_0^{\infty} \frac{\sqrt{2}e^{-\frac{1}{2}(z-\log(3)+2)^2}}{2\sqrt{\pi}} dz$$

1. Uncertainty doesn't interfere
2. Probability query \rightarrow integral expression is a simple transformation
3. Integral problems have mature solutions

Random Variables

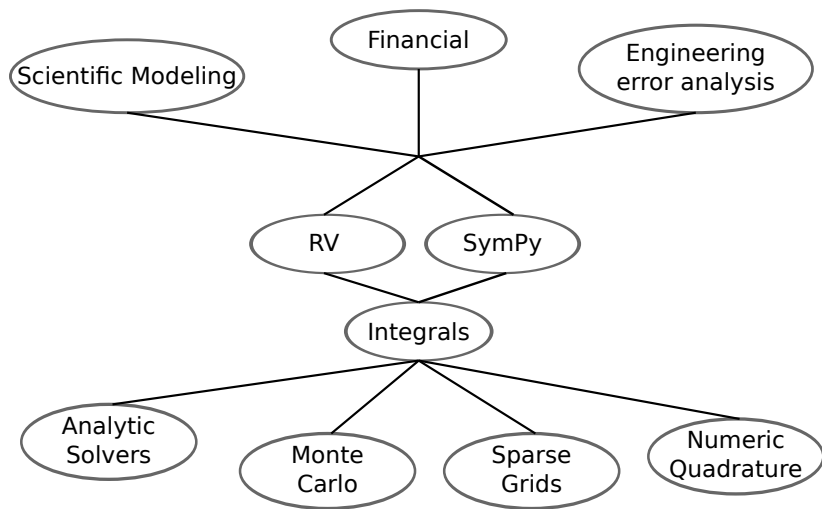


Figure:

Bayesian Inference

```
>>> rate = Beta('lambda', a, b)
>>> count = Poisson('count', rate)
```

$$p(x|\lambda) = \frac{\lambda^x}{e^\lambda x!} \quad p(\lambda) = \frac{\lambda^{a-1} (-\lambda + 1)^{b-1} \Gamma(a+b)}{\Gamma(a) \Gamma(b)}$$

Infer rate given many observations for count

$$\text{Maximize } p(\lambda|x_i) \propto \prod_i p(x_i|\lambda) \cdot p(\lambda)$$

$$0 = \frac{d}{d\lambda} \log \left(\prod_i p(x_i|\lambda) \cdot p(\lambda) \right) = \frac{d}{d\lambda} \sum_i \log(p(x_i|\lambda) \cdot p(\lambda))$$

Need to find the roots of

$$\sum_{i=1}^n \frac{a(\lambda - 1) + b\lambda - \lambda(\lambda - 1) - 2\lambda + (\lambda - 1) \text{data}[i] + 1}{\lambda(\lambda - 1)} = 0$$

Bayesian Inference

PyMC:

SymPy + RV \rightarrow Theano (array computations) \rightarrow C + CUDA

How do we solve math problems?

```
>>> A = Normal('a', 0, 1)
>>> density(A)
```

$$\frac{\sqrt{2}e^{-\frac{1}{2}z^2}}{2\sqrt{\pi}}$$

```
>>> density(A**2)
```

Use generic transformations taught in Statistics 101, e.g.

$$f_Y(y) = f_X(g^{-1}(y)) \left| \frac{dg^{-1}(y)}{dy} \right|$$

$$\frac{\sqrt{2}e^{-\frac{1}{2}z} \left| \frac{1}{\sqrt{z}} \right|}{2\sqrt{\pi}}$$

How do we solve math problems?

```
>>> A = Normal('a', 0, 1)
>>> B = Normal('b', 0, 1)
>>> density(A**2 + B**2)
```

$$\int_{-\infty}^{\infty} \frac{e^{-\frac{1}{2}(b-a)^2} e^{-\frac{1}{2}z + \frac{1}{2}b^2}}{2\pi|\sqrt{z-b^2}|} db$$

How do we solve math problems?

```
>>> A = Normal('a', 0, 1)
>>> B = Normal('b', 0, 1)
>>> density(A**2 + B**2)
```

$$\frac{1}{2}e^{-\frac{1}{2}z}$$

This is a Chi squared distribution with two degrees of freedom

Phenomenological relations:

$$N(0,1)^2 \sim \chi^2(1)$$
$$\chi^2(n) + \chi^2(m) \sim \chi^2(n+m)$$

Term Rewrite System

Rewrite rule:

Source pattern	$\tan(x)$
Target pattern	$\sin(x)/\cos(x)$
Variables	$x,$

Example:

From: $3 + \tan(a + b)**2$

To: $3 + (\sin(a + b) / \cos(a + b))**2$

Term Rewrite System

Rules:

$$\tan(a) \rightarrow \sin(a)/\cos(a)$$

$$\sin^2(a) \rightarrow \frac{1 - \cos(2a)}{2}$$

$$\cos^2(a) \rightarrow \frac{1 + \cos(2a)}{2}$$

$$\sin(a) + \sin(b) \rightarrow 2\sin\left(\frac{a+b}{2}\right)\cos\left(\frac{a-b}{2}\right)$$

$$\sin^2(a) + \cos^2(a) \rightarrow 1$$

$$\sin(a)/\cos(a) \rightarrow \tan(a)$$

...

Simplify:

$$\sin^2(y) + \frac{\sin(z)}{\cos(z)} + \cos^2(y)$$

Encode Statistical Rewrite Rules

Express patterns:

```
patterns = [  
    (Normal(0, 1), StandardNormal(), []),  
    (StandardNormal()2, ChiSquared(1), []),  
    (ChiSquared(m) + ChiSquared(n), ChiSquared(n + m), [n, m]),  
    ...  
]
```

Define control flow:

```
canonicalize = chain(repeat, bottom_up, choose)
```

Combine:

```
stat_simplify = canonicalize(patterns)
```

Status and Evaluation

- ▶ Software:
 - ▶ Fully functional
 - ▶ Lacks efficient matching for many patterns
 - ▶ Maybe integrate into PyMC
- ▶ Evaluation:
 - ▶ Compare numeric runtimes
 - ▶ Compare complexity of problem description

- ▶ Symbolic Statistics
 - ▶ L. Leemis, GD Evans, *APPL: A Probability Programming Language*
 - ▶ M. Erwig and S. Kollmansberger *Probabilistic Functional Programming in Haskell*, 2006
- ▶ Markov chain Monte Carlo
 - ▶ WinBUGS
 - ▶ JAGS
 - ▶ PyMC
- ▶ Term Rewrite Systems - Elan, Maude, Mathematica, Stratego/XT, Coq
- ▶ Term Rewrite Systems (CAS)
 - ▶ RUBI - AD Rich, DJ Jeffrey *A Knowledge Repository for Indefinite Integration Based on Transformation Rules*
 - ▶ H. Fu, X. Zhong, Z. Zeng, *Automated and Readable Simplification of Trigonometric Expressions*

Section 2

Numerical Linear Algebra

The need for a high level array compiler

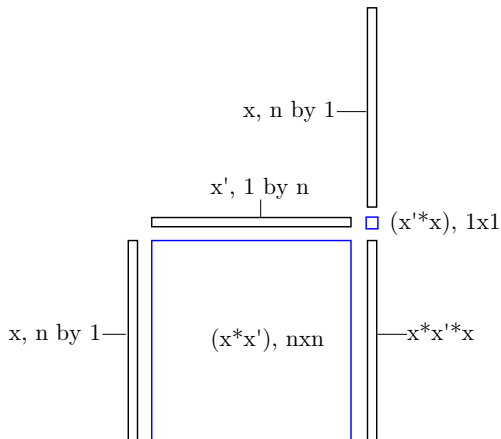
```
x = ones(10000, 1)
```

```
(x*x')*x
```

Elapsed time is 0.337711 seconds.

```
x*(x'*x)
```

Elapsed time is 0.000956 seconds.



The need for a high level array compiler

$$\beta = (X^T X)^{-1} X^T y$$

```
beta = (X.T*X).I*X.T*y
```

The need for a high level array compiler

$$\beta = (X^T X)^{-1} X^T y$$

```
beta = solve(X.T*X, X.T*y)
```

The need for a high level array compiler

$$\beta = (X^T X)^{-1} X^T y$$

```
beta = spd_solve(X.T*X, X.T*y)
```

Numeric libraries for dense linear algebra

- ▶ DGEMM - **D**ouble precision **GE**neral **M**atrix **M**ultiply – $\alpha AB + \beta C$

Numeric libraries for dense linear algebra

- ▶ **DGEMM** - **D**ouble precision **GE**neral **M**atrix **M**ultiply – $\alpha AB + \beta C$
 - ▶ SUBROUTINE DGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)

Numeric libraries for dense linear algebra

- ▶ **DGEMM** - **D**ouble precision **GE**neral **M**atrix **M**ultiply – $\alpha AB + \beta C$
 - ▶ SUBROUTINE DGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
- ▶ **DSYMM** - **D**ouble precision **SY**mmetric **M**atrix **M**ultiply – $\alpha AB + \beta C$

Numeric libraries for dense linear algebra

- ▶ **DGEMM** - **D**ouble precision **GE**neral **M**atrix **M**ultiply – $\alpha AB + \beta C$
 - ▶ SUBROUTINE DGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
- ▶ **DSYMM** - **D**ouble precision **SY**mmetric **M**atrix **M**ultiply – $\alpha AB + \beta C$
 - ▶ SUBROUTINE DSYMM(SIDE,UPLO,M,N,ALPHA,A,LDA,B,LDB,BETA,C,LDC)

Numeric libraries for dense linear algebra

- ▶ **DGEMM** - **D**ouble precision **GE**neral **M**atrix **M**ultiply – $\alpha AB + \beta C$
 - ▶ SUBROUTINE DGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
- ▶ **DSYMM** - **D**ouble precision **SY**mmetric **M**atrix **M**ultiply – $\alpha AB + \beta C$
 - ▶ SUBROUTINE DSYMM(SIDE,UPLO,M,N,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
- ▶ ...

Numeric libraries for dense linear algebra

- ▶ **DGEMM** - **D**ouble precision **GE**neral **M**atrix **M**ultiply – $\alpha AB + \beta C$
 - ▶ SUBROUTINE DGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
- ▶ **DSYMM** - **D**ouble precision **SY**mmetric **M**atrix **M**ultiply – $\alpha AB + \beta C$
 - ▶ SUBROUTINE DSYMM(SIDE,UPLO,M,N,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
- ▶ ...
- ▶ **DPOSV** - **D**ouble symmetric **PO**sitive definite matrix **SolVe** – $A^{-1}y$

Numeric libraries for dense linear algebra

- ▶ **DGEMM** - **D**ouble precision **GE**neral **M**atrix **M**ultiply – $\alpha AB + \beta C$
 - ▶ SUBROUTINE DGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
- ▶ **DSYMM** - **D**ouble precision **SY**mmetric **M**atrix **M**ultiply – $\alpha AB + \beta C$
 - ▶ SUBROUTINE DSYMM(SIDE,UPLO,M,N,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
- ▶ ...
- ▶ **DPOSV** - **D**ouble symmetric **PO**sitive definite matrix **SolVe** – $A^{-1}y$
 - ▶ SUBROUTINE DPOSV(UPLO, N, NRHS, A, LDA, B, LDB, INFO)

Necessary Definitions

Language: Multiply, addition, inverse, transpose, trace, determinant, blocks, etc...

$\text{beta} = (\mathbf{X}^T \mathbf{X}) \cdot \mathbf{I} \cdot \mathbf{X}^T \mathbf{y}$ $\mathbf{X} \cdot \mathbf{I} \cdot \mathbf{X} \rightarrow \text{Identity}$

Predicates: symmetric, positive_definite, full_rank, orthogonal, lower_triangular, etc...

$\text{fullrank}(\mathbf{X})$ $\text{fullrank}(\mathbf{X}) \rightarrow \text{positive_definite}(\mathbf{X}^T \mathbf{X})$

Computations:

```
class SYMM(BLAS):
    inputs      = [alpha, A, B, beta, C]
    outputs     = [alpha*A*B + beta*C]
    condition   = symmetric(A) or symmetric(B)
    inplace     = {0: 4} # 0th output stored in 4th input
    fortran     = ....
```

Mathematical code

Original language definition in Maude

```
...  
eq A (B + C) = (A B) + (A C) .  
eq (alpha A)' = alpha A' .  
eq A'' = A .  
eq inverse(A) A = Identity .  
...  
eq X X' is symmetric = True .  
ceq X Y is positive-definite = True if X is positive-definite  
                                     and Y is positive-definite .  
ceq X X' is positive-definite = True if X is full-rank .
```

Eventually moved to SymPy for distribution reasons

Computation

Given:

$(X.T * X) . I * X.T * y$
`full_rank(X)`

Produce:

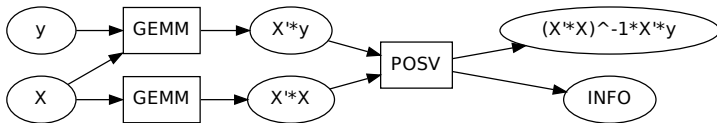
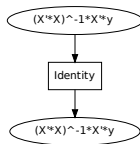
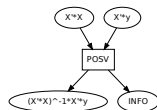
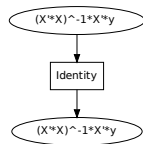


Figure:

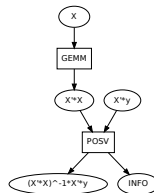
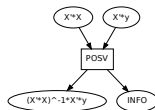
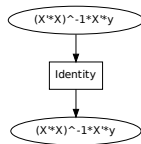
Computation



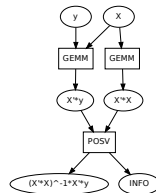
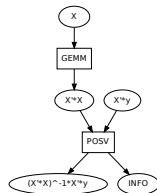
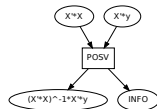
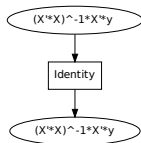
Computation



Computation



Computation



User Experience

```
X = MatrixSymbol('X', n, m)
y = MatrixSymbol('y', n, 1)
```

```
inputs  = [X, y]
outputs = [(X.T*X).I*X.T*y]
facts   = Q.fullrank(X)
```

```
f = f2py(next(compile(inputs, outputs, facts)))
```

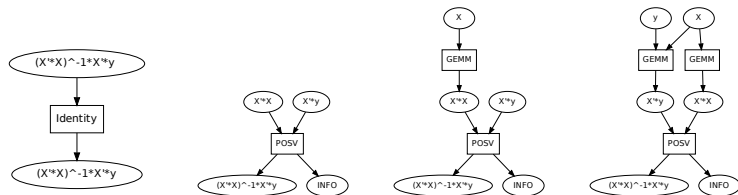
```
subroutine f(X, y, var_7, m, n)
implicit none
```

```
integer, intent(in) :: m
integer, intent(in) :: n
real*8, intent(in) :: y(n)           ! y
real*8, intent(in) :: X(n, m)        ! X
real*8, intent(out) :: var_7(m)       ! 0, X'*y, (X'*X)^-1*X'*y
real*8 :: var_8(m, m)                ! 0, X'*X
integer :: INFO                      ! INFO
```

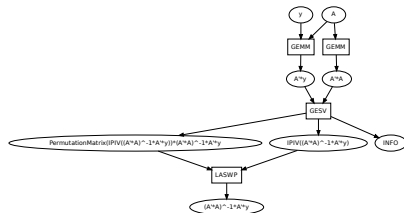
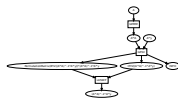
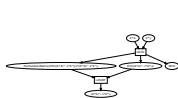
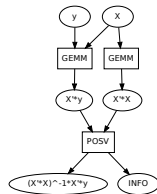
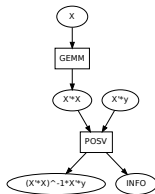
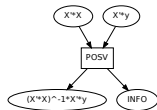
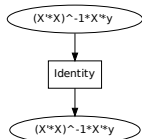
```
call dgemm('N', 'N', m, 1, n, 1, X, n, y, n, 0, var_7, m)
call dgemm('N', 'N', m, m, n, 1, X, n, X, n, 0, var_8, m)
call dposv('U', m, 1, var_8, m, var_7, m, INFO)
```

```
RETURN
END
```

Multiple Results



Multiple Results

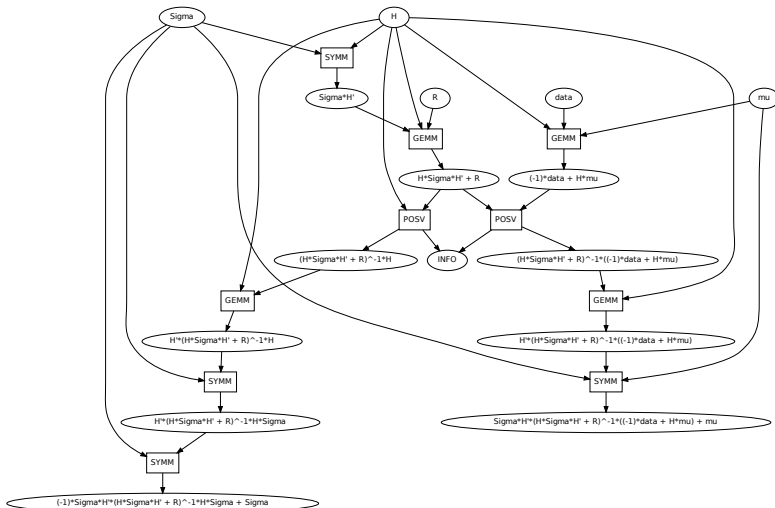


Section 3

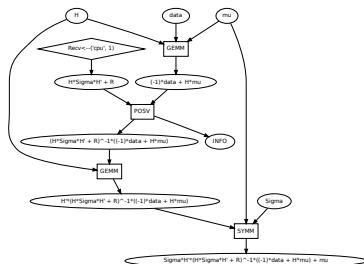
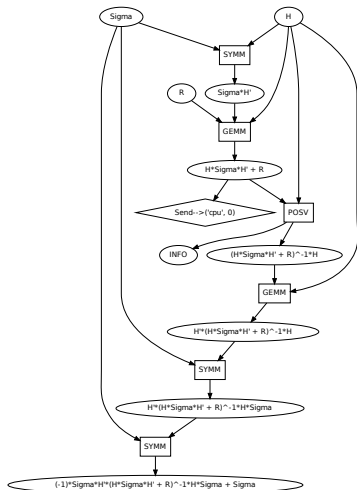
Static Scheduling

Static Scheduling

$\text{newmu} = \mu + \text{Sigma} \cdot \text{H.T} * (\text{R} + \text{H} \cdot \text{Sigma} \cdot \text{H.T}).\text{I} * (\text{H} \cdot \mu - \text{data})$
 $\text{newSigma} = \text{Sigma} - \text{Sigma} \cdot \text{H.T} * (\text{R} + \text{H} \cdot \text{Sigma} \cdot \text{H.T}).\text{I} * \text{H} * \text{Sigma}$



Static Scheduling



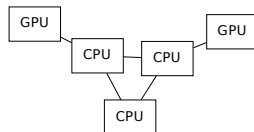
Static Scheduling

Given:

Computation Graph



Worker network



Computation times
Communication times

task, worker \rightarrow time
variable, source, target \rightarrow time

Produce:

Set of computation subgraphs to minimize total runtime

Related work

► Performance Modeling

- E Peise and P Bientinesi. *Performance Modeling for Dense Linear Algebra*. 2012
- Roman Iakymchuk. *Performance Modeling and Prediction for Linear Algebra Algorithms* 2012
- JJ Dongarra, RA Vandegeijn, and DW Walker. *Scalability issues affecting the design of a dense linear algebra library* 1994

► Heterogeneous Static Scheduling

- M. Tompkins. *Optimization Techniques for Task Allocation and Scheduling in Distributed Multi-Agent Operations*. 2003
- H. Topcuoglu, S. Hariri, M. Wu. *Performance-effective and low-complexity task scheduling for heterogeneous computing*. 2002
- Suggestions?

► Automated Dense Linear Algebra

- ScaLAPACK, PlaLAPACK, BLACS
- FLAME - Language for blocked matrix algorithms
 - SuperMatrix - Dynamic shared memory variant
 - Elemental - Distributed memory variant
- Magma - Hybrid LAPACK
- Spiral - Hardware specific numeric code generation with internal computation language

Application - Blocked Cholesky Decomposition

Math Problem:

$$Ax = y \rightarrow LL^T x = y \rightarrow x = L^{-T} L^{-1} y$$

A symmetric positive definite, L lower triangular

$$\begin{bmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}^T$$

$$L_{11} := \text{cholesky}(A_{11})$$

$$L_{21} := A_{21} L_{11}^{-T}$$

$$L_{22} := \text{cholesky}(A_{22} - L_{21} L_{21}^T)$$

Status and Evaluation

- ▶ Have done - Software
 - ▶ Implemented Schedulers
 - ▶ Implemented rudimentary cost model
 - ▶ Everything hooks up well
- ▶ Will do - Compare existing schedulers with
 - ▶ Schedulers: IP, Heuristic, Naive dynamic scheduling
 - ▶ Variety of block / problem sizes
 - ▶ Variety of desired algorithms
 - ▶ Variety of timing models
 - ▶ Under uncertain conditions
 - ▶ Compare scheduling times vs execution times
- ▶ Could do -
 - ▶ Implement direct cost model (run and profile each task)
 - ▶ Comparison against FLAME/Magma

DAG Ordering

Recv and Send return control immediately.

RecvWait and SendWait block until communication terminates.

A, B, C, D are computation tasks

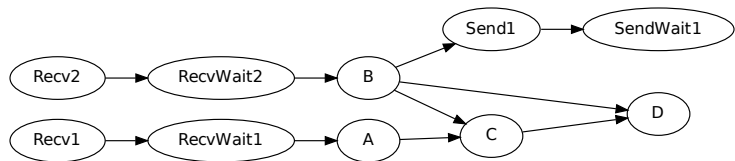


Figure:

Order 1: R1 RW1 R2 RW2 A B C D S1 SW1

Order 2: R2 R1 RW2 B S1 RW1 A C D SW1

DAG Ordering

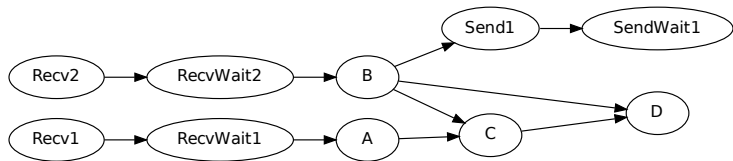
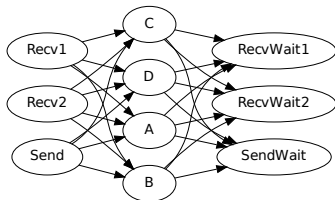


Figure:



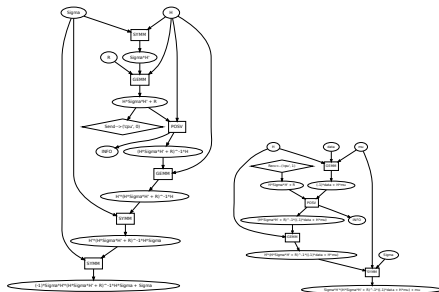
DAG Ordering

```
def dependence(a, b):
    if depends((a, b)): return 1
    if depends((b, a)): return -1
    return 0

def mpi_send_wait_key(a):
    """ Wait as long as possible on Waits, Start Send/Recvs early """
    if isinstance(a.op, (MPIRecvWait, MPISendWait)): return 1
    if isinstance(a.op, (MPIRecv, MPISend)): return -1
    return 0

def mpi_tag_key(a):
    """ Break MPI ties by using the variable tag - prefer lower tags first """
    if isinstance(a.op, (MPISend, MPIRecv, MPIRecvWait, MPISendWait)):
        return a.op.tag
    return 0
```

Thank You



Slides: <http://github.com/mrocklin/candidacy-exam/>

Section 4

Extras

Inplace



Figure:

Inplace

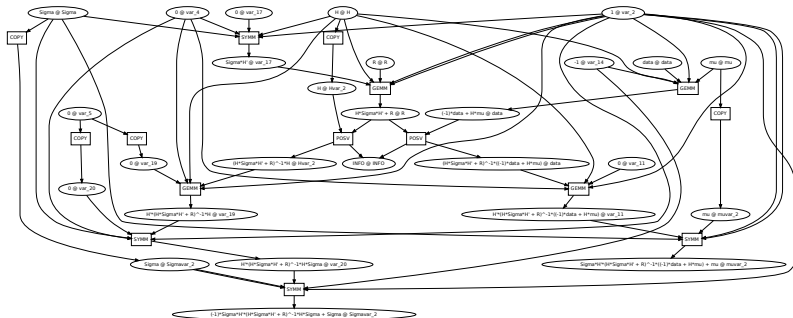


Figure: