Generation and Tuning of parallel solutions for linear algebra equations

Alexandre X. Duchâteau
University of Illinois

ILLINOIS

illinois.edu

Collaborators

- Thesis Advisors
  - Denis Barthou (Labri/INRIA)
  - David Padua (UIUC)

- Future collaboration
  - starPU team, INRIA Bordeaux

illinois.edu

Objectives and Contributions

INTRODUCTION

illinois.edu

Objectives

- Compile linear algebra equations
  - Compute X for $L * X * U - X = C$ [DTSY]
  - Compute L and U for $L * U = A$ [LU]

- Generate efficient task parallel code
  - Identify tasks
  - Generate task dependence graph

illinois.edu

Motivation

- Focus is on linear algebra
  - Start from high level description
  - No code or algorithm
- Derivation through blocking of operands
  - Data centric approach
- Derivation for parallelism
  - Output is a parallel task graph

illinois.edu

Contributions

- A specification language
  - Express computation
  - Characterize operands (shapes)
  - Identify wanted result
- Derivation rules
  - Validity/applicability patterns
  - Operators symbolic execution rules
  - Dependence build engine

illinois.edu

A detailed view of the generator

SYSTEM DESCRIPTION

illinois.edu

Description Language - Operands

```
%% Operands
X: Unknown Matrix
L: Lower Triangular Square Matrix
U: Upper Triangular Square Matrix
C: Square Matrix

%% Equation
L*X*U-X=C
```

- All operands

- (Type inference)

- Status
  - Known, Unknown
- Shape
  - Triangular, diagonal
- Type
  - Matrix, (vector, scalar)
- (Sizes)
- (Density)

illinois.edu

Description language - Equation

```
%% Operands
X: Unknown Square Matrix
L: Lower Triangular Square Matrix
U: Upper Triangular Square Matrix
C: Square Matrix

%% Equation
L*X*U-X=C
```

- Simple equations
  - Assignments
    - X = A*B
- Solvers
  - LU, Sylvester
  - L*X=B
- Base for decomposition

Kernel Declaration

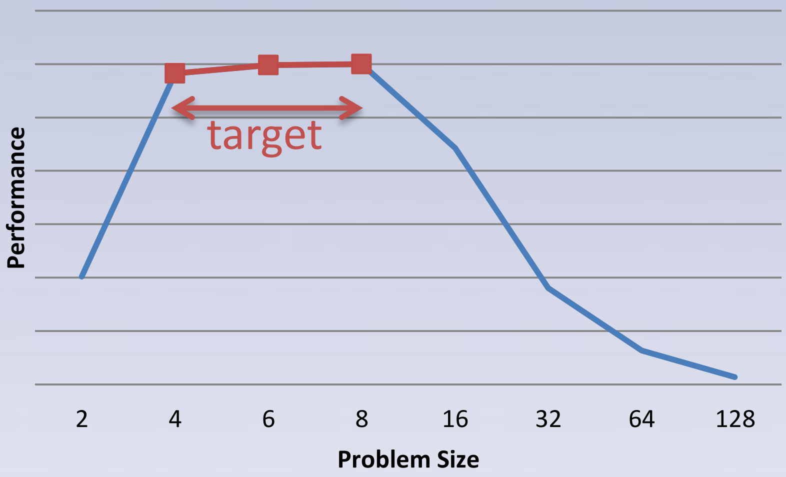
- Generate a full solution
  - Cost of full recursion
  - Usually not a good idea

- Use existing kernels and libraries
  - Already optimized

illinois.edu

Kernel Performance

- Measure performance
  - Depend on size
- Guide exploration
  - Optimal nodes
  - Similar to ATLAS



illinois.edu

Equation Derivation

- Start from input formula
  - divide and conquer approach finds algorithms
- Operands are blocked
  - Explore many possible blockings
  - Now matrices of blocks and not elements
- Symbolically execute equation
  - Expose problem subdivision

illinois.edu

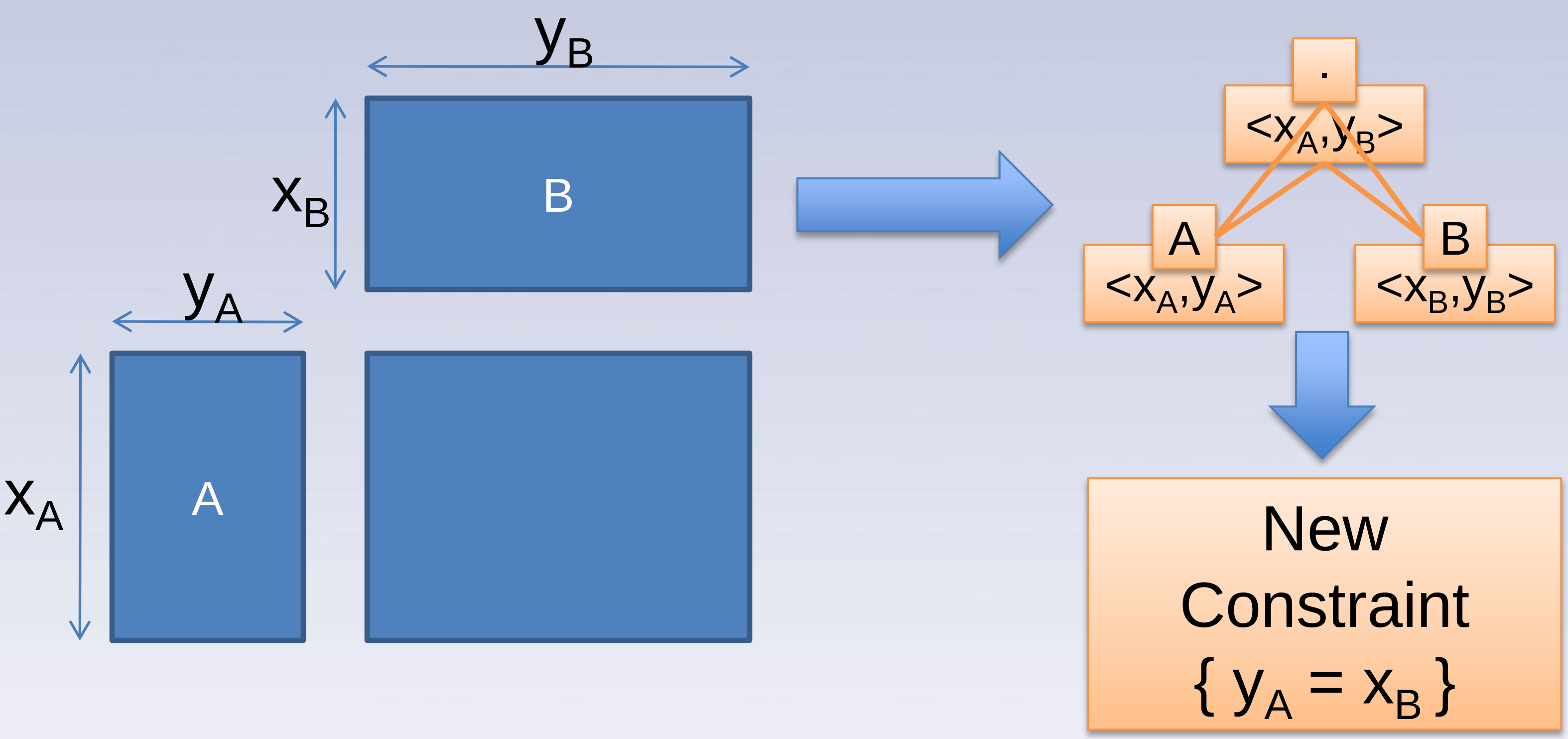
Defining blocking space

- Blocking defines the space of solutions
  – Must only generate valid solutions

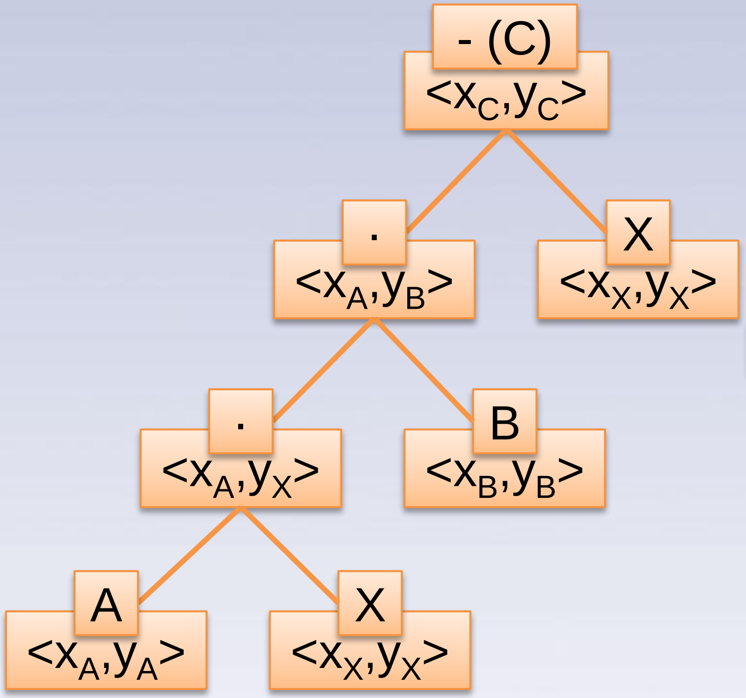
- Look at the equation's operation tree
  – Each node gets a set of dimensions
  – Generate constraints depending on operation

illinois.edu

Validity of Blocking



Valid Blockings - DTSY



Constraints

$y_A = x_X$

$y_X = x_B$

$x_A = x_X$

$y_B = y_X$

$x_C = x_X$

$y_C = y_X$

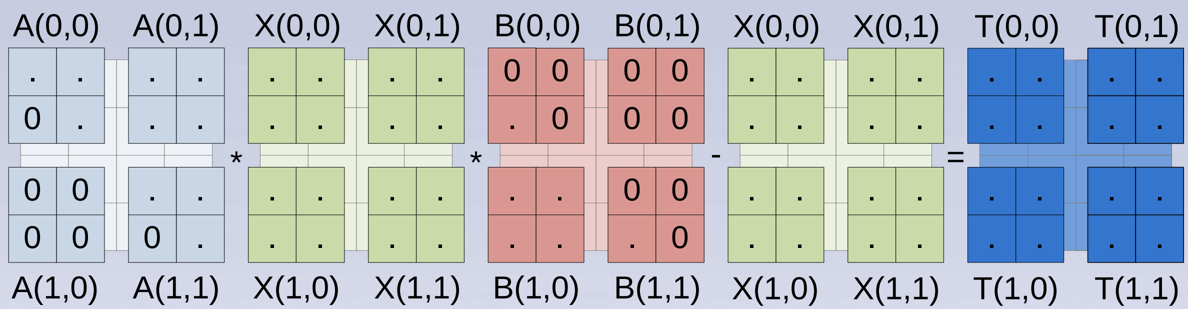
$\{ x_A, y_A, x_X, x_C \}$

$\{ x_B, y_B, y_X, y_C \}$

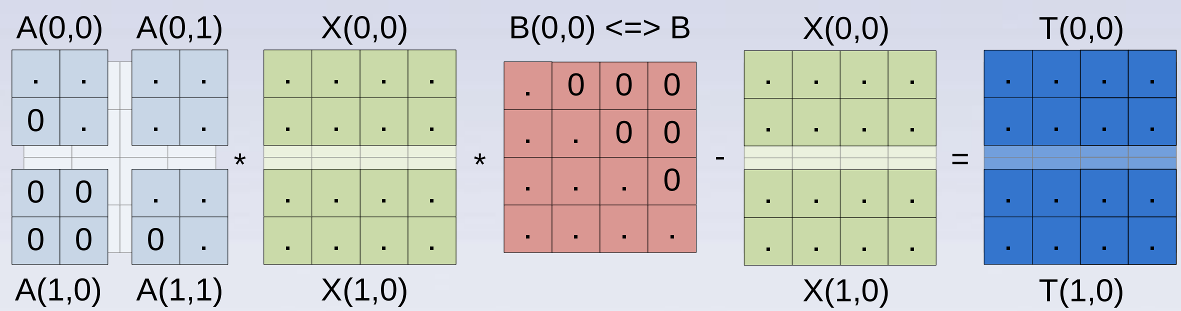
Valid Blockings – DTSY (2)

`A*X*B – X = C | A lower triangular, B upper triangular`

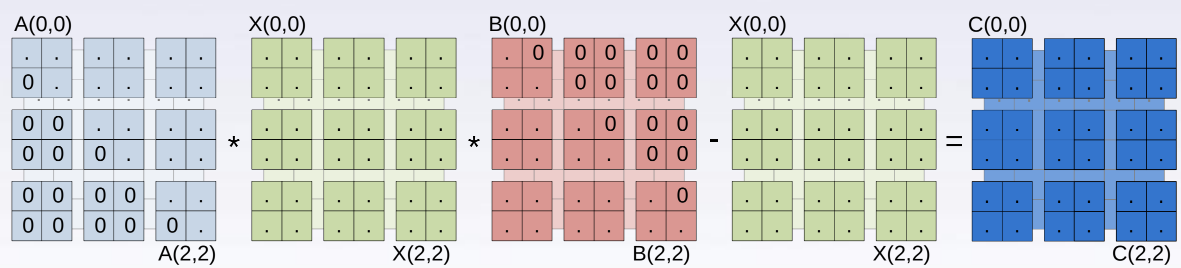
xA = yA = xX = 2
xB = yB = yX = 2



xA = yA = xX = 2
xB = yB = yX = 1

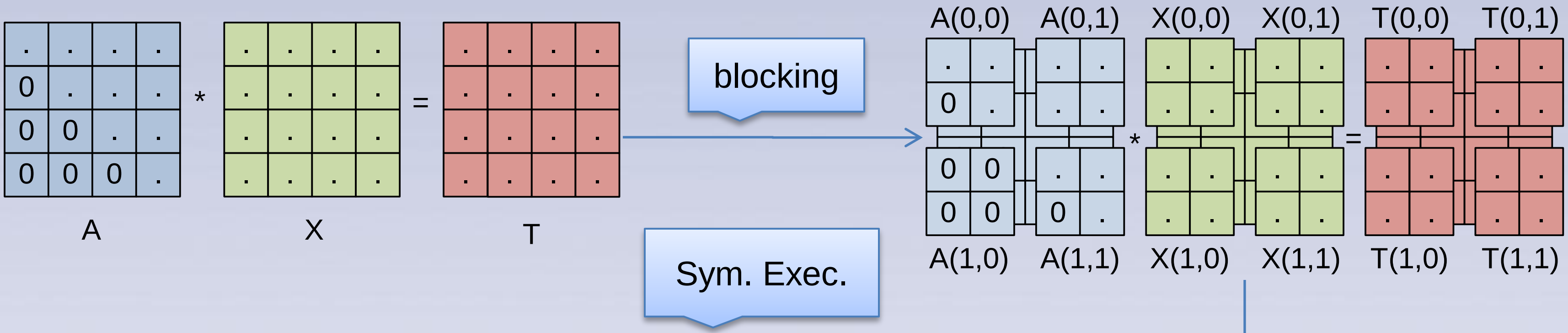


xA = yA = xX = 3
xB = yB = yX = 3



illinois.edu

Derivation example




```
T(0,0) = A(0,0)*X(0,0) + A(0,1)*X(1,0)
T(0,1) = A(0,0)*X(0,1) + A(0,1)*X(1,1)
T(1,0) = A(1,0)*X(0,0) + A(1,1)*X(1,0)
T(1,1) = A(1,0)*X(0,1) + A(1,1)*X(1,1)
```

Removal of 0-computation

```
T(0,0) = A(0,0)*X(0,0) + A(0,1)*X(1,0)
T(0,1) = A(0,0)*X(0,1) + A(0,1)*X(1,1)
T(1,0) =                 A(1,1)*X(1,0)
T(1,1) =                 A(1,1)*X(1,1)
```

illinois.edu

Operand characterization

- Blocks of X are outputs (unknown)
- A(0,0) and A(1,1) are lower triangular

Equation	Input	Output
$T(1,1) = A(1,1) * X(1,1)$	$T(1,1)\ A(1,1)$	$X(1,1)$
$T(1,0) = A(1,1) * X(1,0)$	$T(1,0)\ A(1,1)$	$X(1,0)$
$T(0,1) = A(0,0) * X(0,1) + A(0,1) * X(1,1)$	$T(0,1)\ A(0,0)\ A(0,1)$	$X(0,1)\ X(1,1)$
$T(0,0) = A(0,0) * X(0,0) + A(0,1) * X(1,0)$	$T(0,0)\ A(0,0)\ A(0,1)$	$X(0,0)\ X(1,0)$

illinois.edu

Equation Signature

- Need to identify equations
- Identification through types and operators
  - $A * X = T : LT * UNK = MT$
- Set of simplification rules
  - $UNK + MT * MT \Rightarrow UNK + MT$

illinois.edu

Identify task

- $T(1,1) \; = \; A(1,1) * X(1,1)$
  - Signature : $LT * UNK = MT$

- Instance of original problem
  - Solvable

- $X(1,1)$ can now be considered known

illinois.edu

Building dependence

- Find instances of $X(1,1)$
  - Shift in input set
  - Add dependence edge

Equation	Input	Output
$T(0,1) = A(0,0) * X(0,1) + A(0,1) * X(1,1)$	$T(0,1)\ A(0,0)$ $A(0,1)\ \boldsymbol{X(1,1)}$	$X(0,1)$

New Dependence

$$\{T(1,1) = A(1,1) * X(1,1) \rightarrow T(0,1) = A(0,0) * X(0,1) + A(0,1) * X(1,1)\ \}$$

illinois.edu

Signature Simplification

- $T(0,1) = A(0,0) * X(0,1) + A(0,1) * X(1,1)$
  1. $MT = LT * UNK + MT * MT$
  2. $MT = LT * UNK + MT$
  3. $MT - MT = LT * UNK$
  4. $MT = LT * UNK$

- Match to the original problem !

illinois.edu

Post identification expansion

Simplification step	Corresponding equation
$MT = LT * UNK + MT * MT$	$T(0,1) = A(0,0) * X(0,1) + A(0,1) * X(1,1)$
$MT = LT * UNK + MT$	$T1 = A(01) * X(1,1)$ $T(0,1) = A(0,0) * X(0,1) + T1$
$MT - MT = LT * UNK$	
$MT = LT * UNK$	$\mathbf{T1 = A(01) * X(1,1)}$ $\mathbf{T2 = T(0,1) - T1}$ $\mathbf{T2 = A(0,0) * X(0,1)}$

illinois.edu

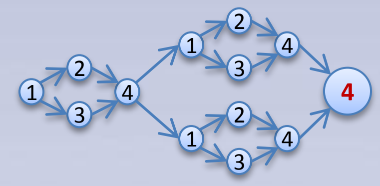
Generator

Equation Dependence Graph

$$L * X * U - X = C$$

$$\emptyset$$

$$L11 * X11 * U11 - X11 = T$$



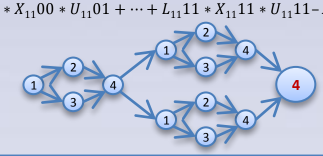
Derivation

Set Equations Dependence Graph

1: $L00 * X00 * U00 - X00 = C00$
2: $L10 * X00 * U00 + L11 * X10 * U00 - X10 = C10$
3: $L00 * X00 * U01 + L00 * X01 * U11 - X01 = C01$
4: $L10 * X00 * U01 + \ldots + L11 * X11 * U11 - X11 = C11$

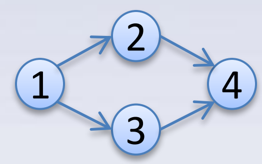
$$\emptyset$$

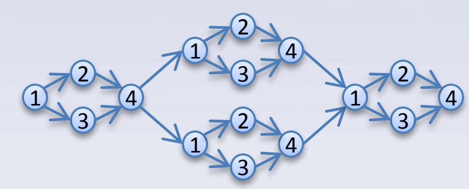
1: $L_{11}00 * X_{11}00 * U_{11}00 - X_{11}00 = T00$
2: $L_{11}10 * X_{11}00 * U_{11}00 + L_{11}11 * X_{11}10 * U_{11}00 - X_{11}10 = T10$
3: $L_{11}00 * X_{11}00 * U_{11}01 + L_{11}00 * X_{11}01 * U_{11}11 - X_{11}01 = T01$
4: $L_{11}10 * X_{11}00 * U_{11}01 + \cdots + L_{11}11 * X_{11}11 * U_{11}11 - X_{11}11 = T11$



Identification

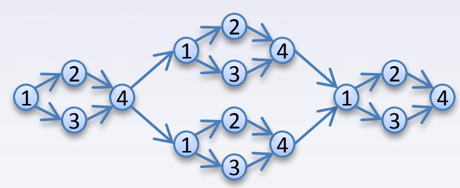
Dependence Graph





Termination

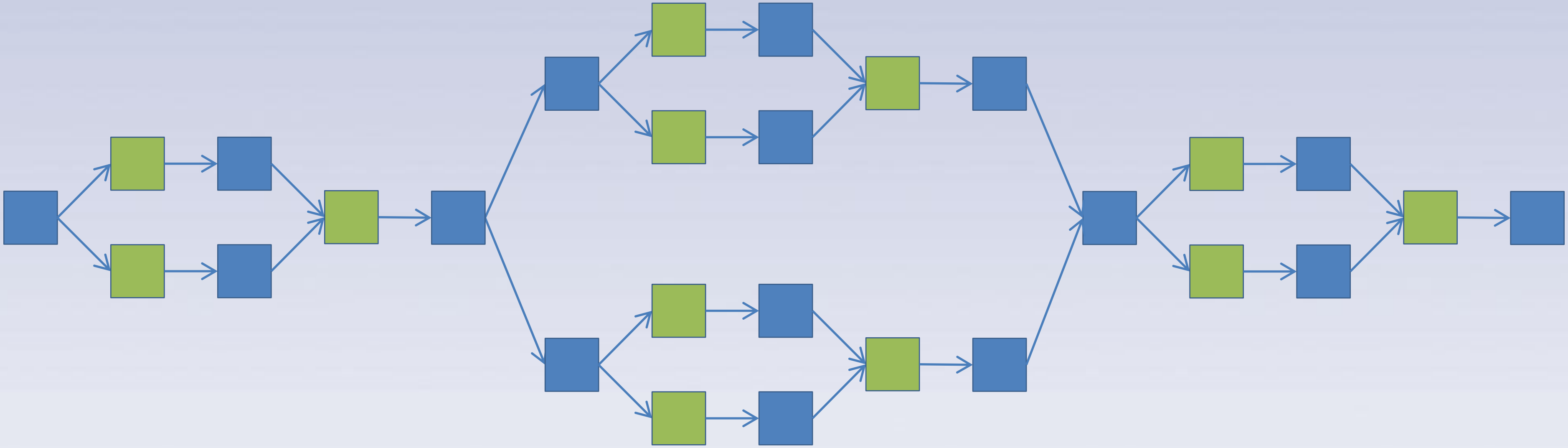
Final Graph



Selection

illinois.edu

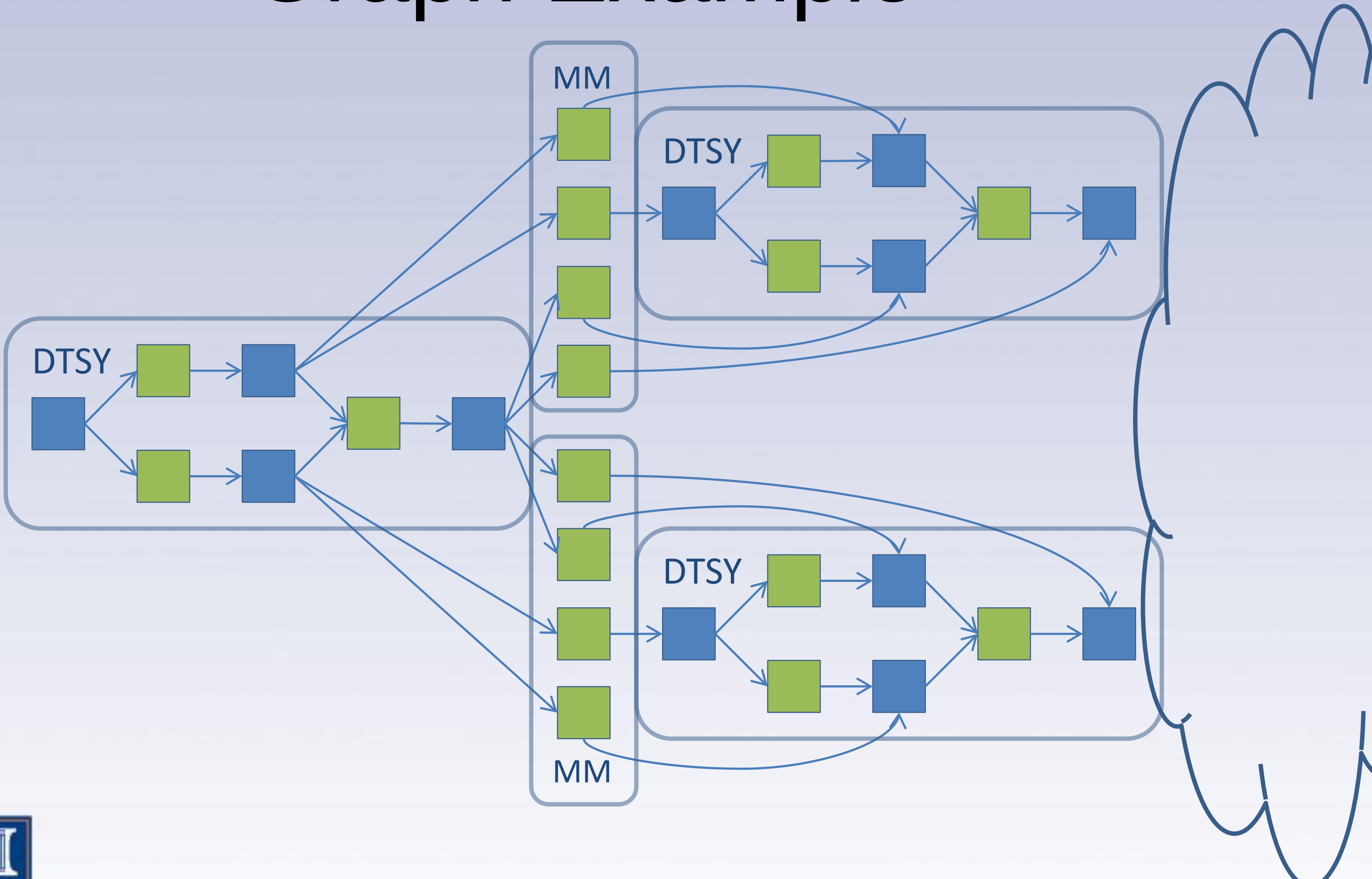
Simple graph example



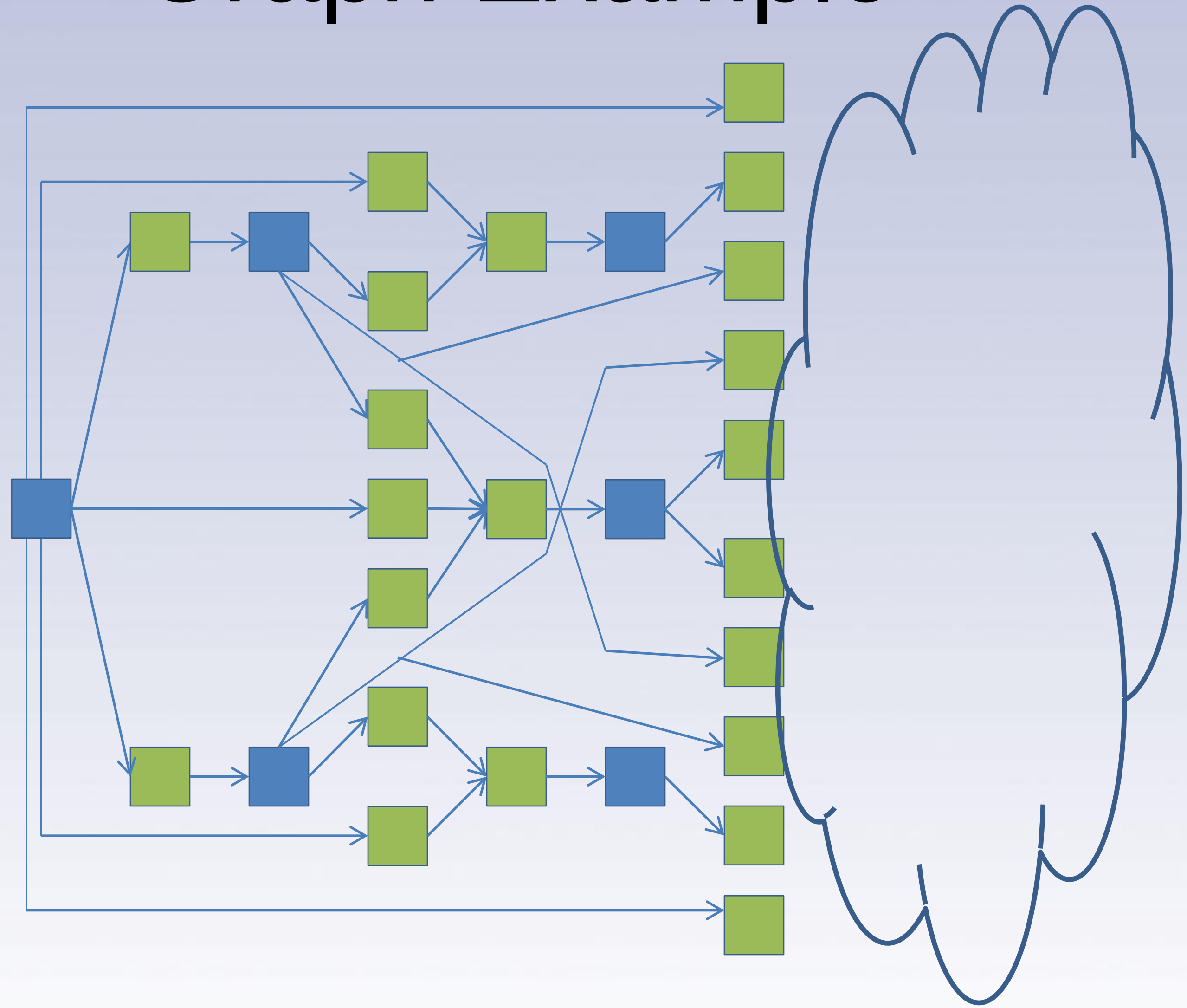
illinois.edu

Graph Example

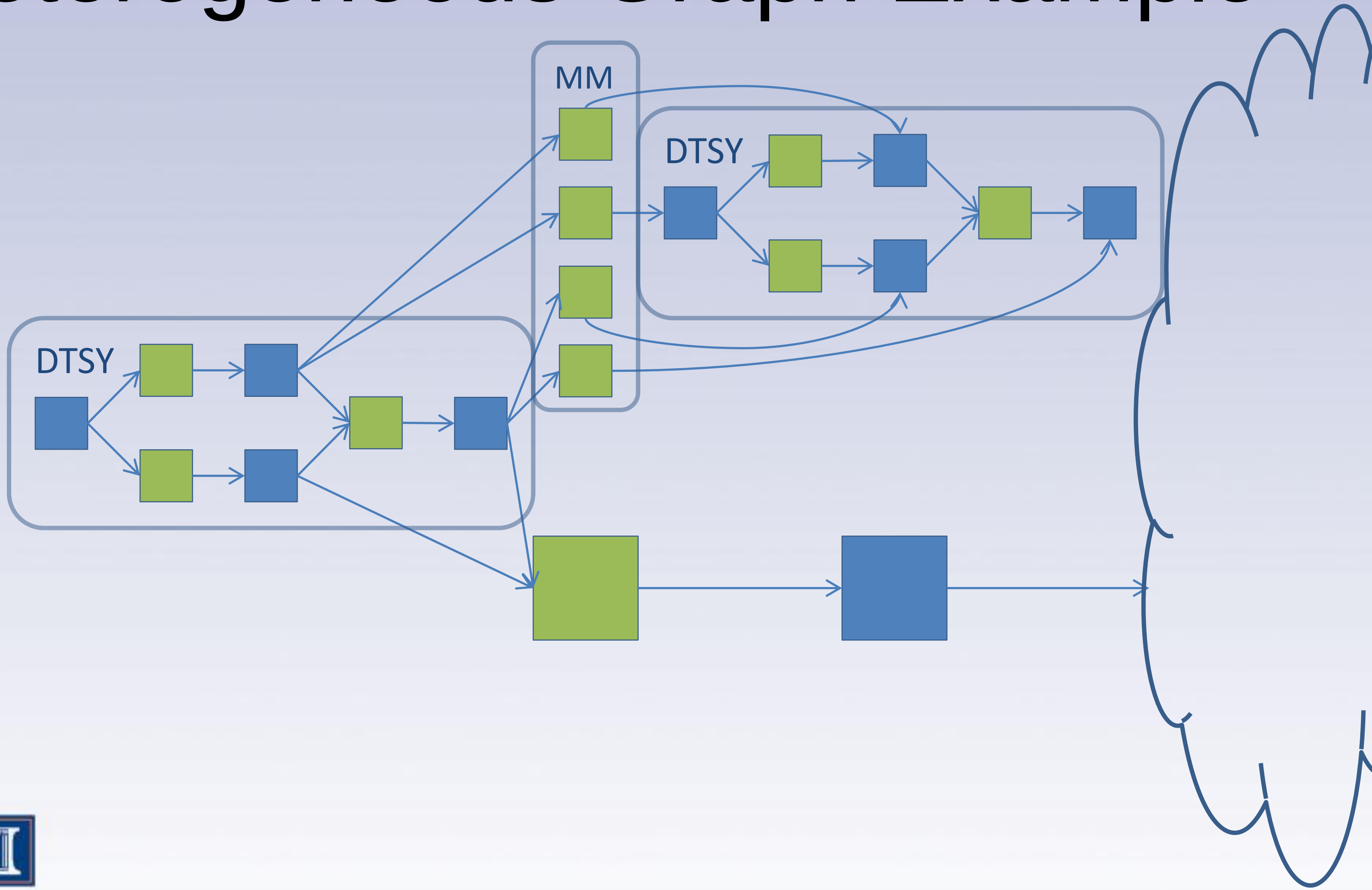
2 by 2



Graph Example



Heterogeneous Graph Example



Translating task graphs into code

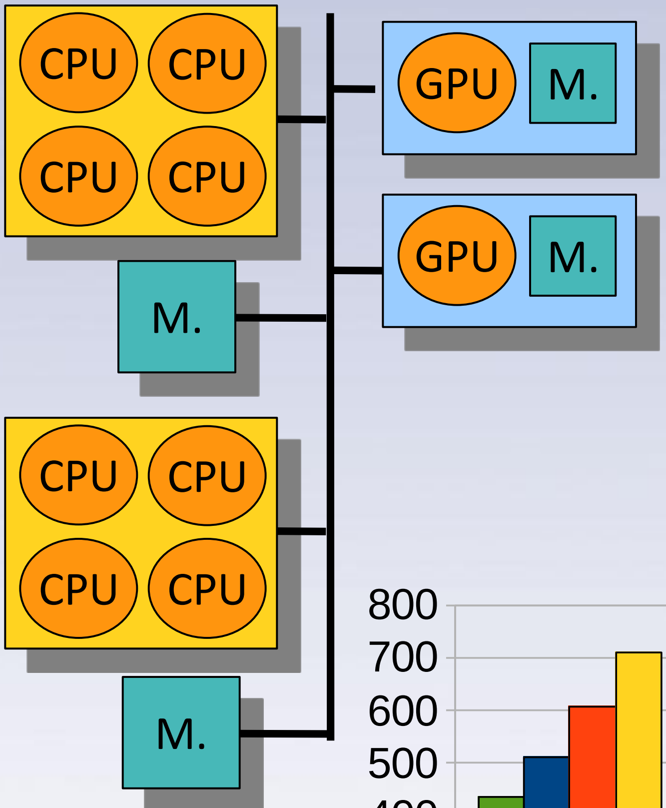
FUTURE WORK

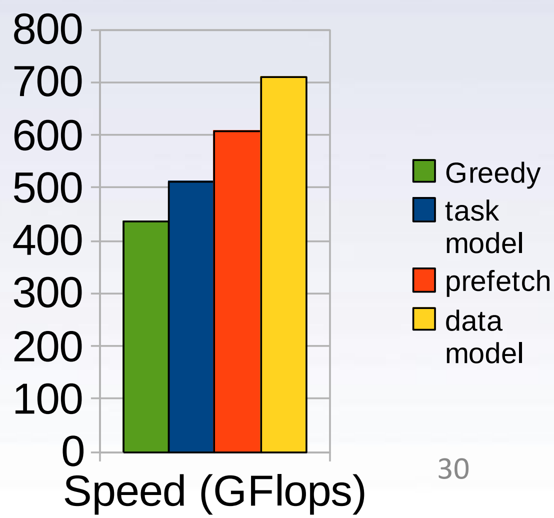
Exploiting heterogeneous machines : starPU Runtime

- Goal: Scheduling (≠ offloading) tasks over heterogeneous machines
  - CPU + GPU + SPU = *PU
  - Auto-tuning of performance models
  - Optimization of memory transfers

- Target for
  - Compilers
    - StarSs [UPC], HMPP [CAPS]
  - Libraries
    - PLASMA/MAGMA [UTK]
  - Applications
    - Fast multipole methods [CESTA]

- StarPU provides an Open Scheduling platform
  - Scheduling algorithm = plugins

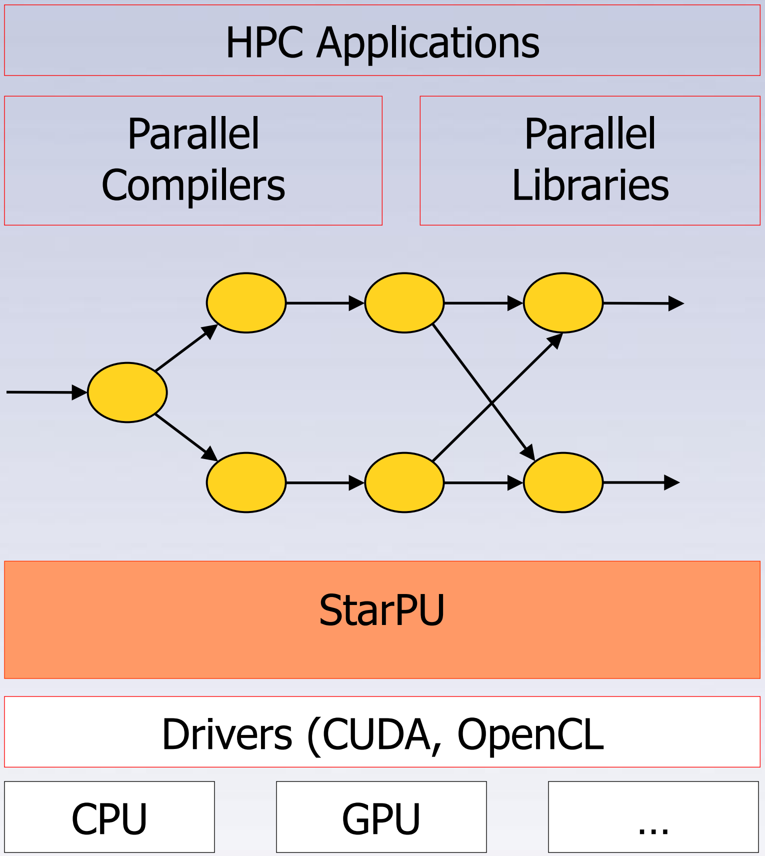




Overview of StarPU

Maximizing PU occupancy, minimizing data transfers

- Principle
  - Accept tasks that may have multiple implementations
    - Together with potential interdependencies
      - Leads to a dynamic acyclic graph of tasks
      - Data-flow approach
  - Provide a high-level data management layer
    - Application should only describe
      - Which data may be accessed by tasks
      - How data may be divided



illinois.edu

CONCLUSION

illinois.edu

Conclusion

- Faster development cycle for architectures
  - No time to hand-tune everything anymore
  - Can’t hand tune for every HW iteration
- Increased complexity
  - Heterogeneous systems
- Description of an automatic methodology
  - Leverage existing “small scale” libraries

illinois.edu