Hydra: Generation and Tuning of parallel solutions for linear algebra equations

Alexandre X. Duchâteau
University of Illinois at Urbana Champaign

I2PC

ILLINOIS
illinois.edu

UNIVERSITÉ DE BORDEAUX

Inria
INVENTEURS DU MONDE NUMÉRIQUE

Collaborators

- Thesis Advisors
  - Denis Barthou (Labri/INRIA Bordeaux)
  - David Padua (University of Illinois at UC)

- Future collaboration
  - starPU team, INRIA Bordeaux

illinois.edu

I2PC

INVENTEURS DU MONDE NUMÉRIQUE

UNIVERSITÉ DE BORDEAUX

Objectives and Contributions

INTRODUCTION

illinois.edu

Objectives

- Compile linear algebra equations
  - Compute X for $L * X - X * U = C$ [CTSY]
  - 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

Description Language - Operands

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

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

%% Parameters
@name ctsy
```

- All operands
- (Type inference)

illinois.edu

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

Description language - Equation

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

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

%% Parameters
@name ctsy
```

- Base for decomposition

- Simple equations
  - Assignments
    - X = A*B
- Solvers
  - LU
  - Triangular Sylvester
  - L*X=B
  - Cholesky

illinois.edu

Description language - Parameters

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

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

%% Parameters
@name ctsy
```

- Drive code generation

- Set of parameters
  - Name
  - Codelet
  - Order of operands
  - Data type

- For customization
  - Default values used

illinois.edu

Kernel Declaration

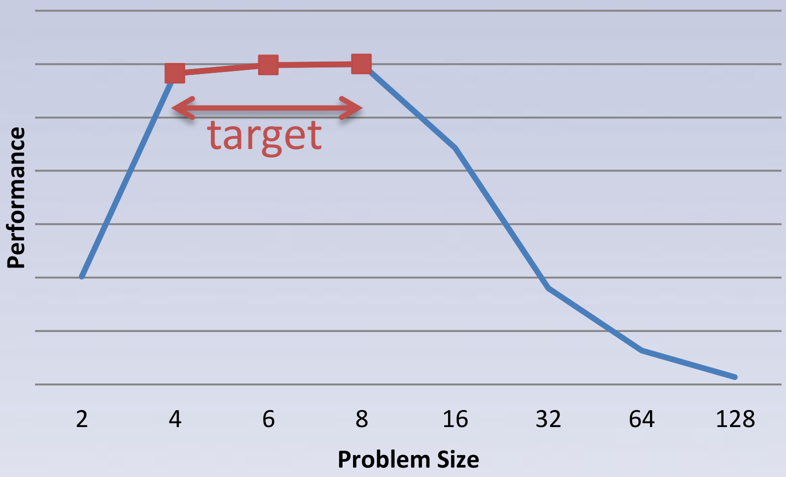
- Generate a full solution
  - Cost of full recursion
  - Usually not a good idea

- Use existing kernels and libraries
  - Already optimized

illinois.edu

Kernel Performance

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



Starpu Codelet

Codelet declaration

```
void __gemm(void *buffers[], void *cl_arg);
struct starpu_codelet gemm_cl = {
    .where = STARPU_CPU,
    .cpu_funcs = {__gemm, NULL},
    .nbuffers = 3,
    .modes = {STARPU_R, STARPU_R,
STARPU_RW}
};
```

Kernel code

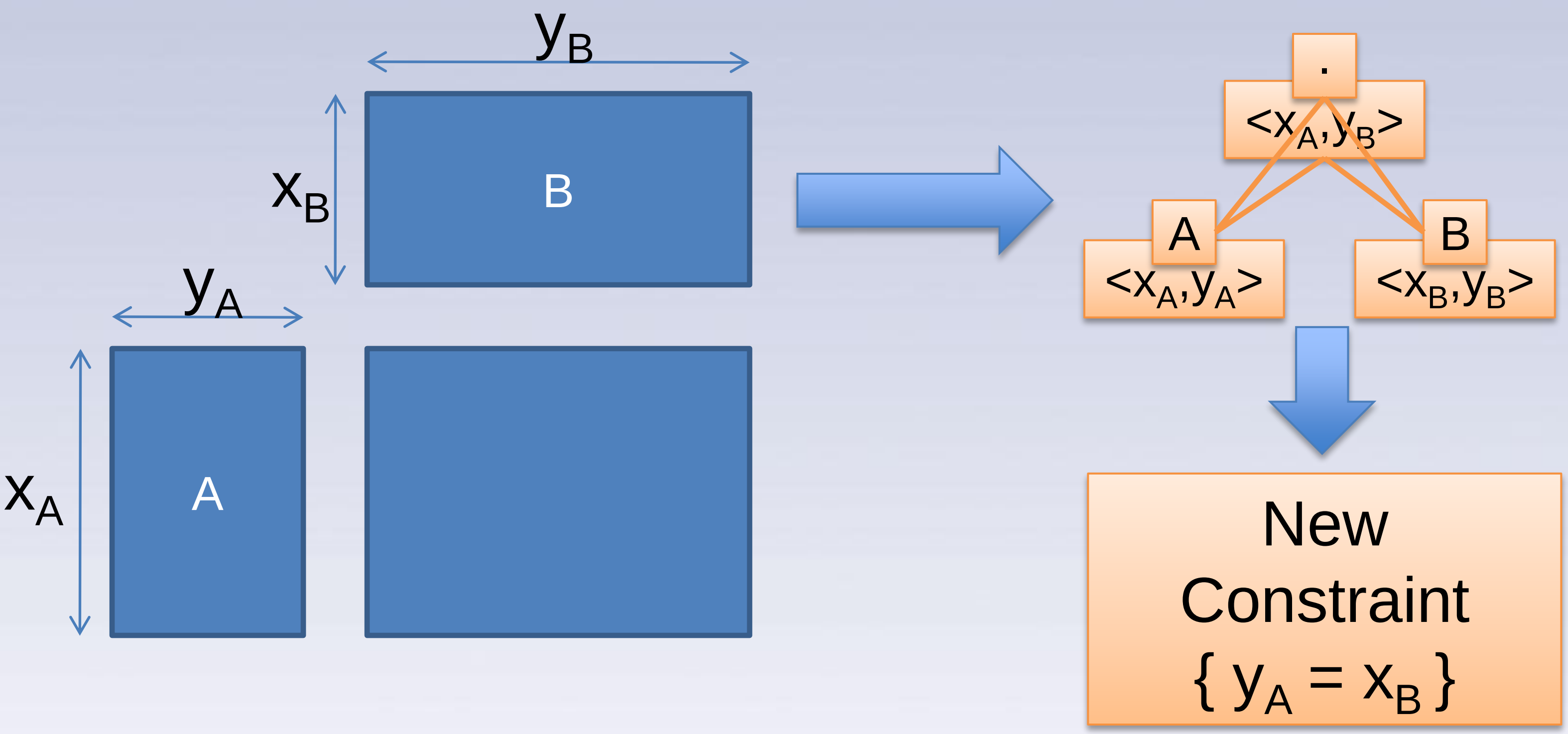
```
void __gemm(void *buffers[], void *cl_arg) {
    struct params *params = cl_arg;
    int n = params->n;
    double *a = (double *)
        STARPU_MATRIX_GET_PTR(buffers[0]);
    // …
    cblas_dgemm(CblasRowMajor,
        CBlasNoTrans, CBlasNoTrans,
        n, n,  n,
        1.0,
        a, n,  b, n,
        1.0,
        c, n);
}
```

illinois.edu

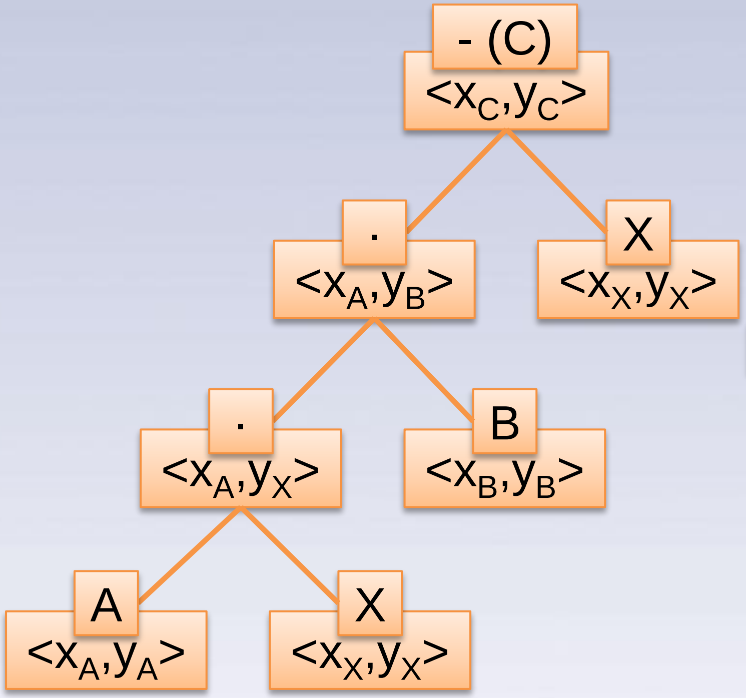
Defining blocking space

- 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

Validity of Blocking



Valid Blockings - DTSY



Constraints

$y_A = x_X$

$y_X = x_B$

$x_A = x_X$

$y_B = y_X$

$x_C = x_X$

$y_C = y_X$

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

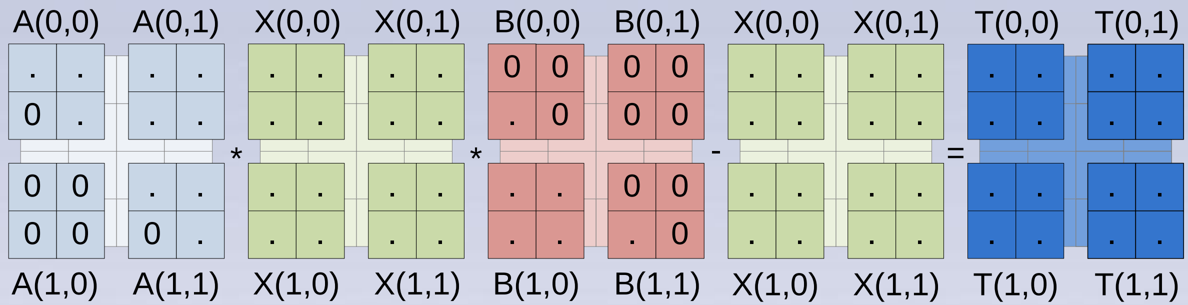
$\{ x_B, y_B, y_X, y_C \}$

illinois.edu

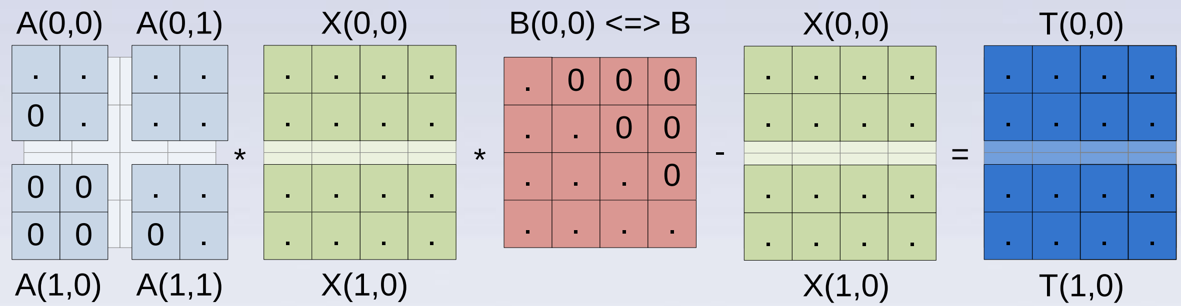
Valid Blockings – DTSY (2)

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

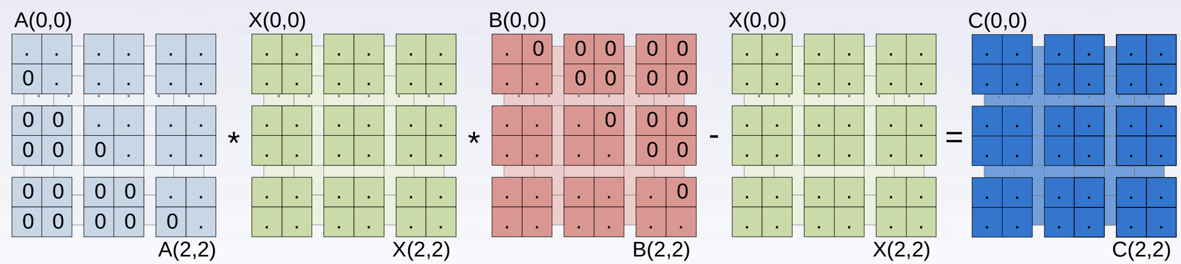
xA = yA = xX = 2
xB = yB = yX = 2



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

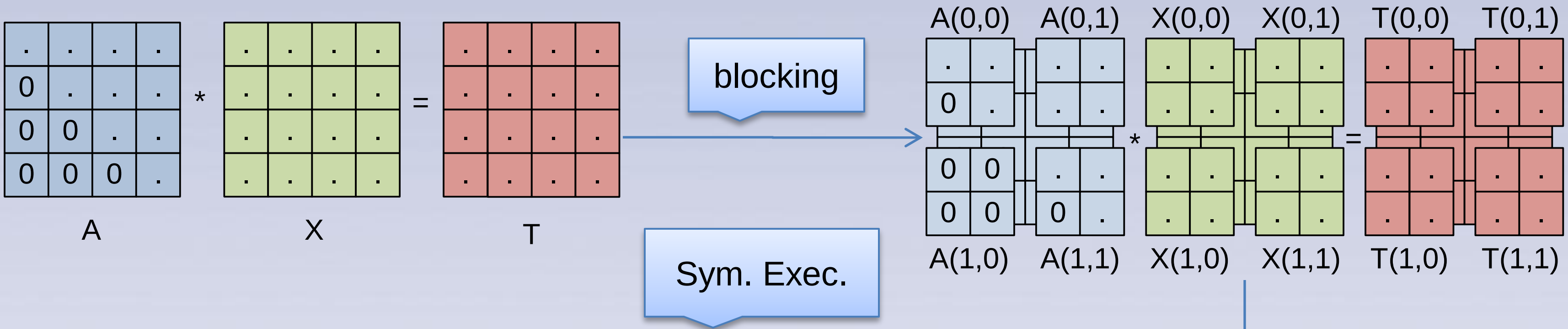


xA = yA = xX = 3
xB = yB = yX = 3



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 !

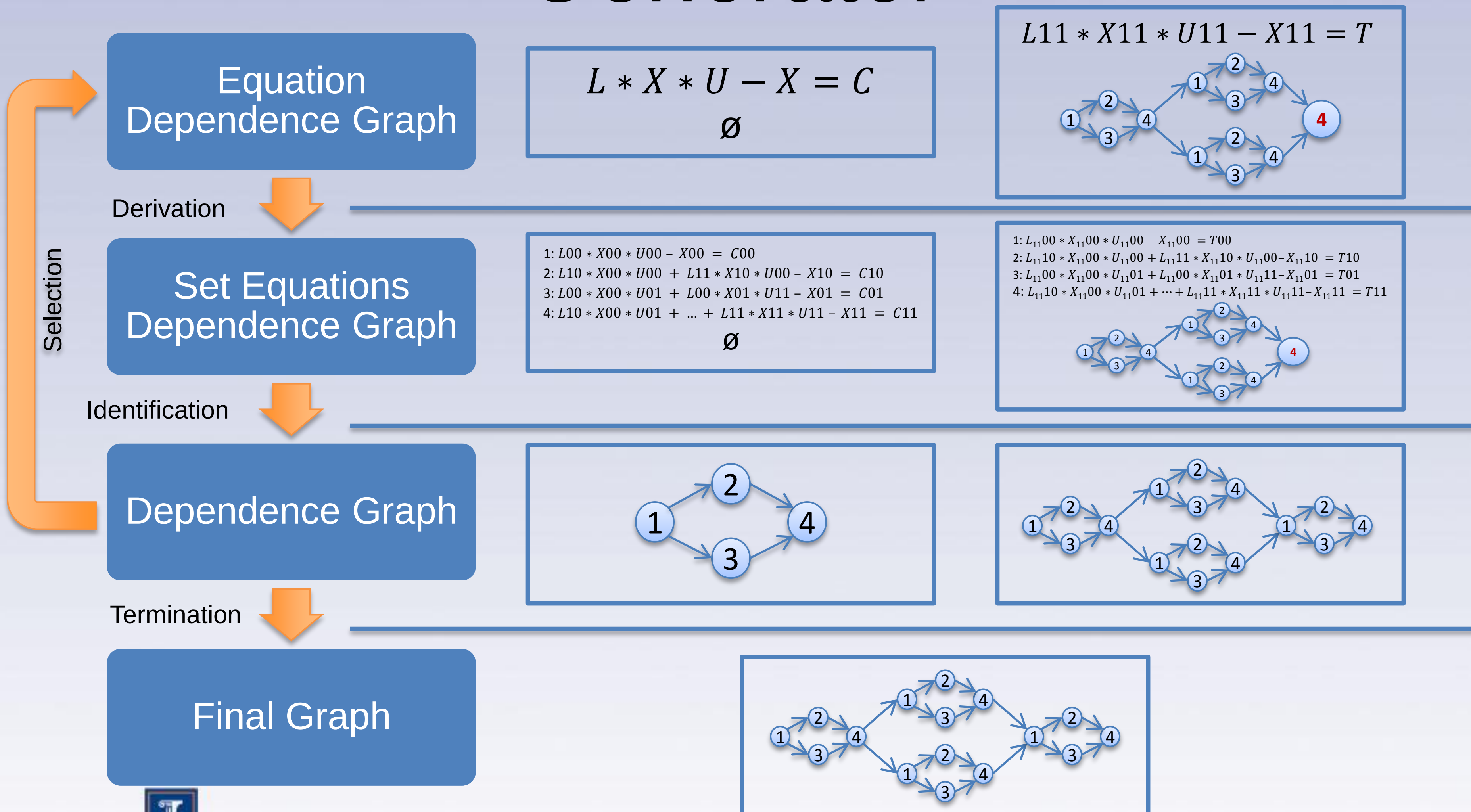
illinois.edu

Post identification expansion

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

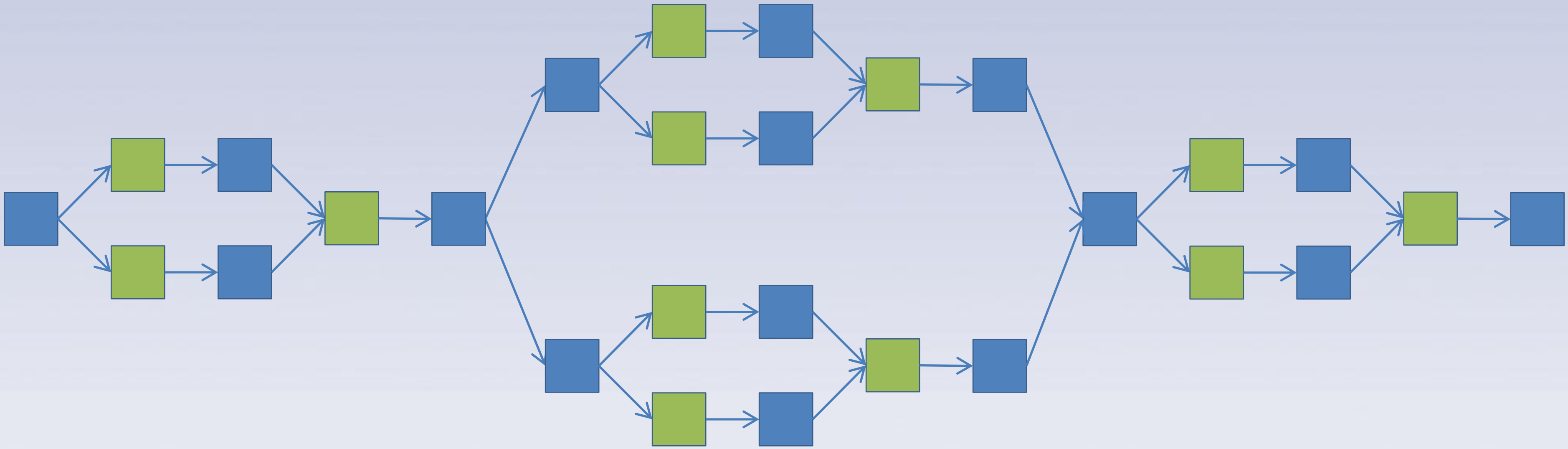
illinois.edu

Generator



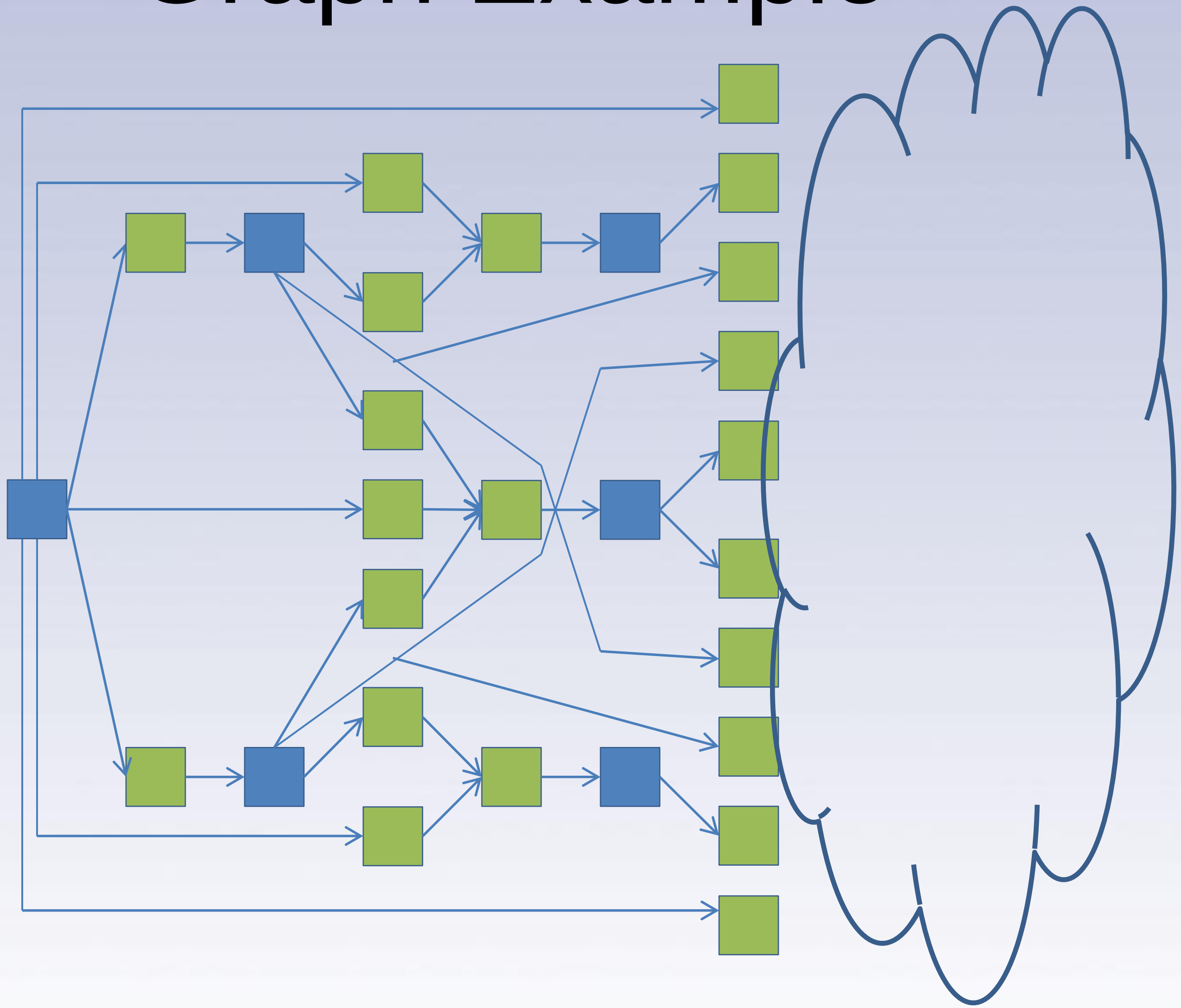
illinois.edu

Simple graph example

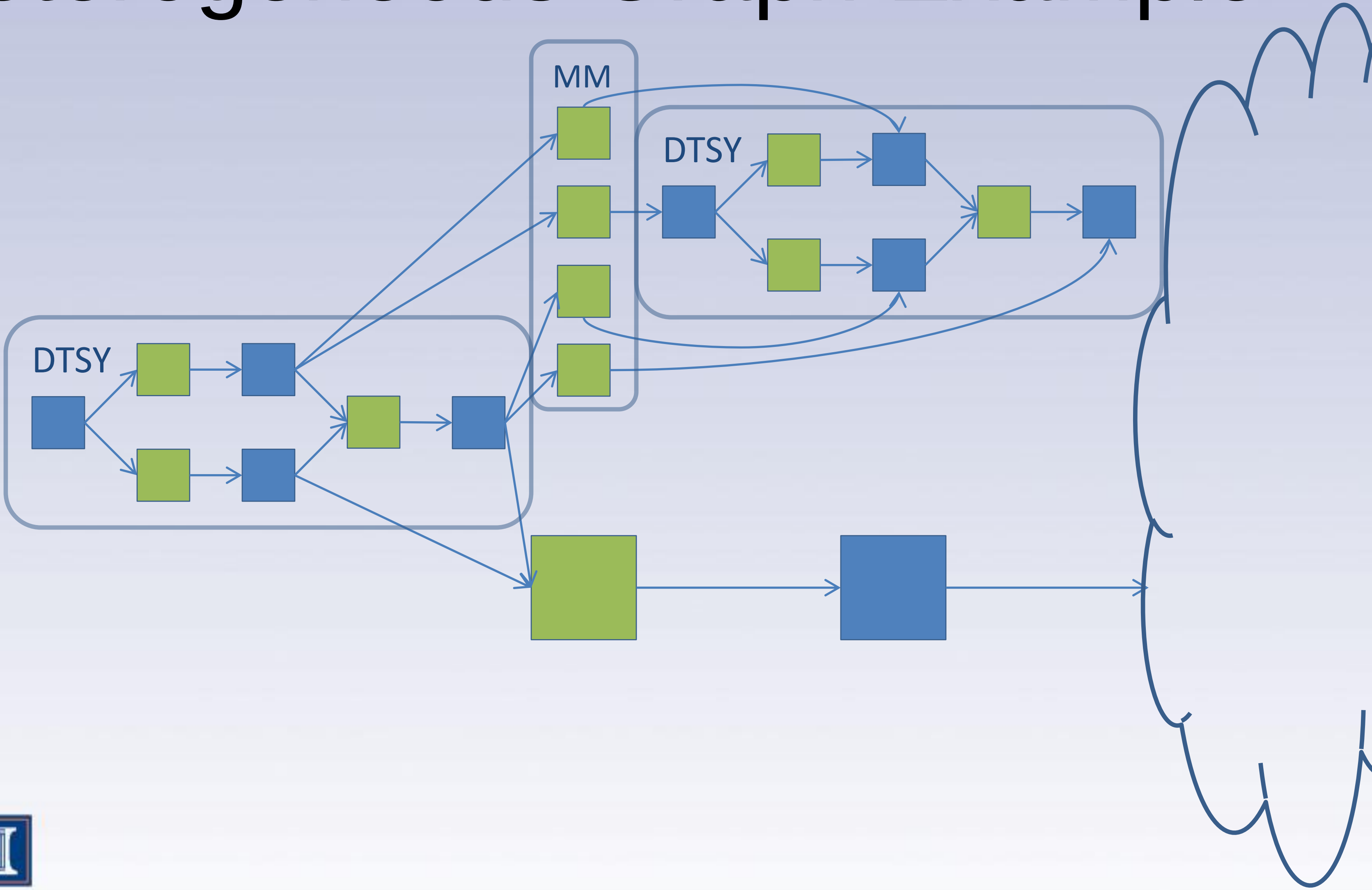


illinois.edu

Graph Example



Heterogeneous Graph Example



INITIAL RESULTS AND CHALLENGES

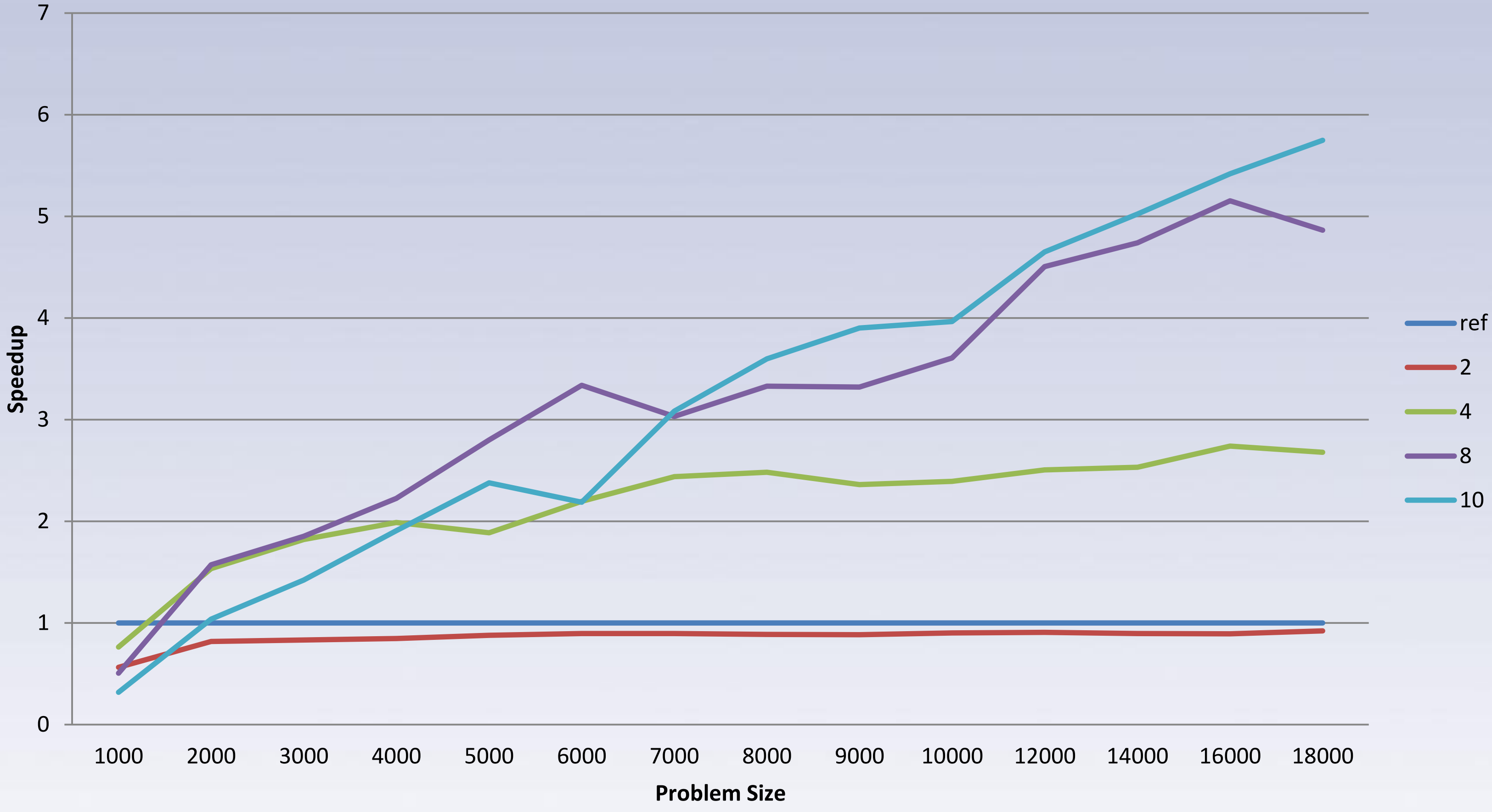
illinois.edu

Challenges

- Version generation time
  - 5 minutes in generator
  - 20 minutes compiling
- Generated code size
  - 500k lines of code in single function
  - (icc segfaults)

illinois.edu

Trisolve (L*X=B)



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

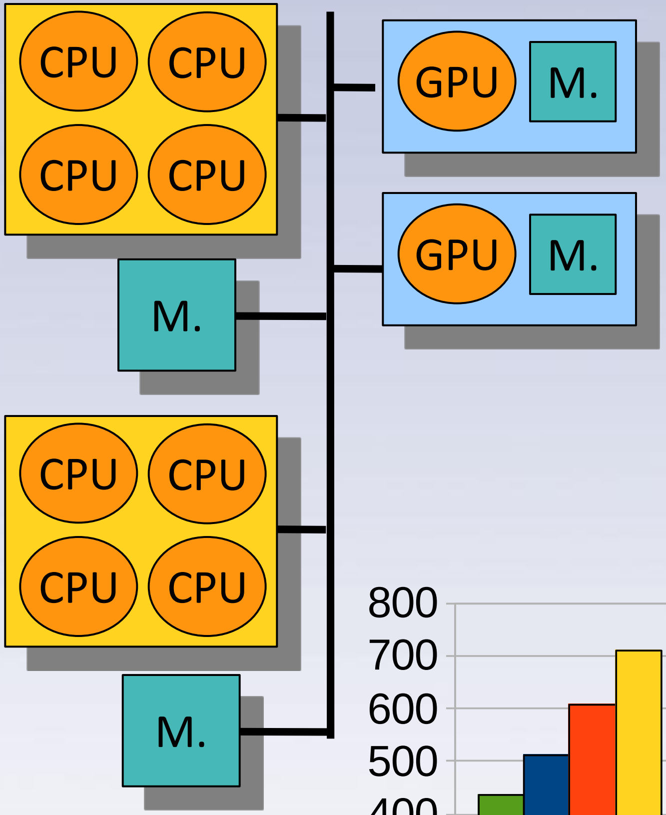
illinois.edu

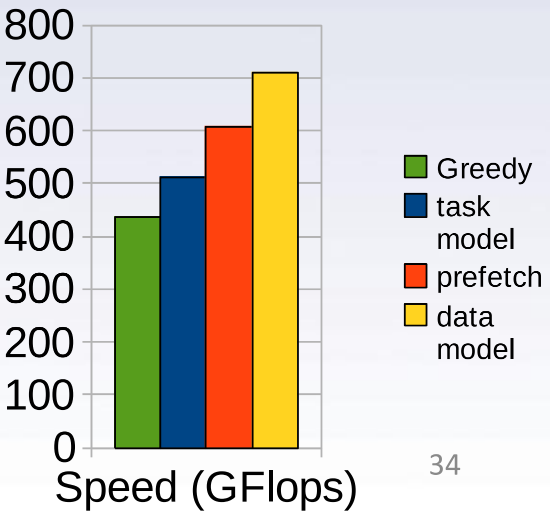
AU CAS OU

illinois.edu

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



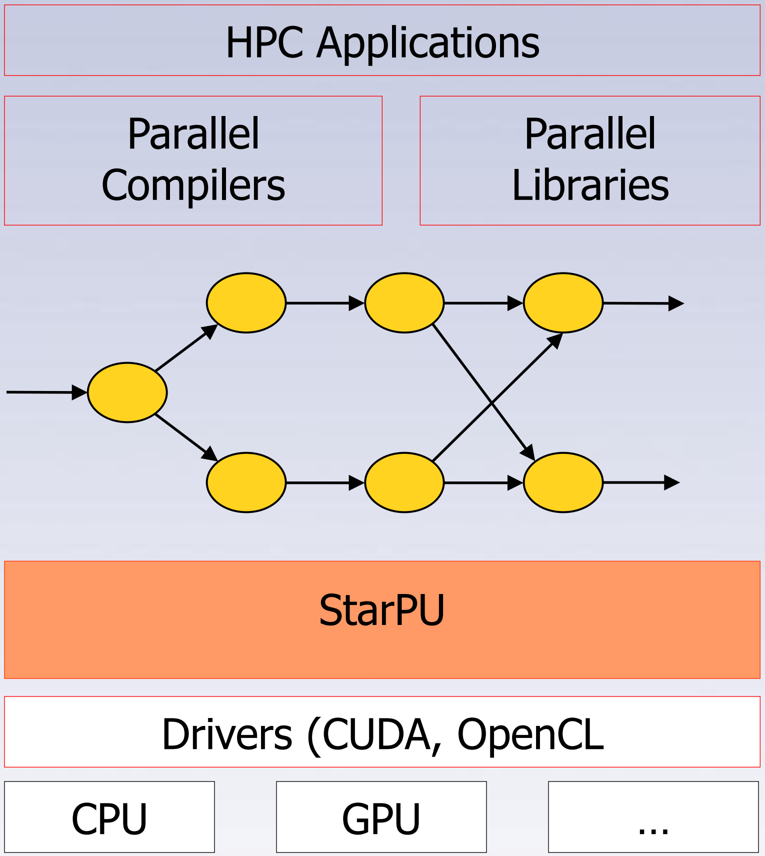


illinois.edu

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

Code styles and optimization

```
for(int i = 0 ; i < N ; i++)
 for(int j = 0 ; j < N ; j++)
  for(int k = 0 ; k < N ; k++)
   c[i][j] += a[i][k] * b[k][j];
```

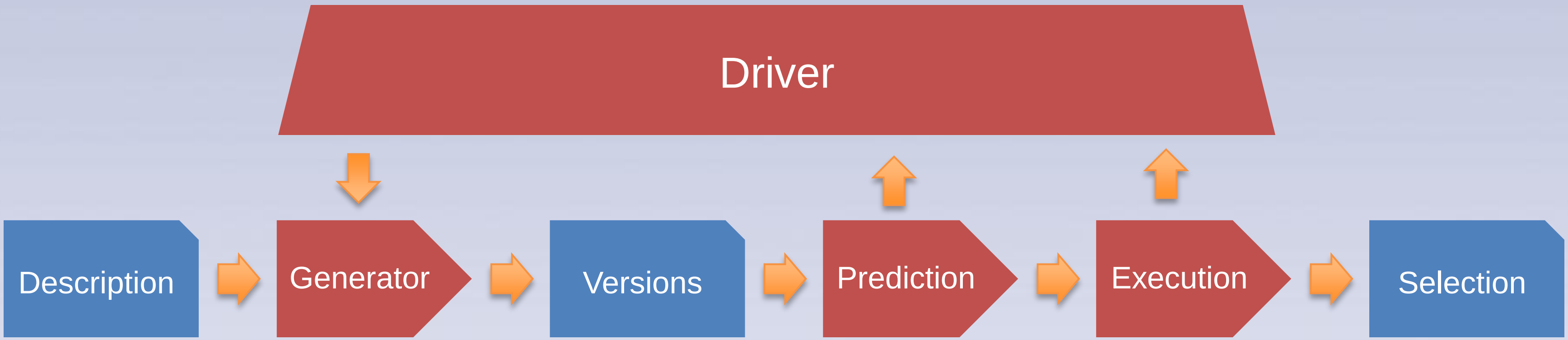
```
for(int kk = 0 ; kk < N ; kk+=B)
 for(int ii = 0 ; ii < N ; ii+=B)
  for(int jj = 0 ; jj < N ; jj+=B)
   for(int i = ii ; i < ii+B ; i++)
    for(int k = kk ; k < kk+B ; k++) {
     c_i = c[i];
     a_ik = a[i][k];
     b_k = b[k];
     for(int j = jj ; i < jj+B ; i+=2) {
      c_i[j] += a_ik * b_k[j];
      c_u[j+1] += a_ik * b_k[j+1];
     }
    }
```

illinois.edu

Empirical search

- Generation of multiple solutions
  - Algorithmic exploration
- Theoretical evaluation
  - Usually unreliable
  - Filtering heuristic
- Empirical evaluation (execution)
  - slow (<u>very slow</u> when looking at bad versions)

Generalized system overview



- Predictor
  - Filters what versions are actually executed
- Driver
  - Guide the search

illinois.edu