Imperial College
London

More Definite Results From the Pluto Scheduling Algorithm

By

Athanasios Konstantinidis

Supervisor

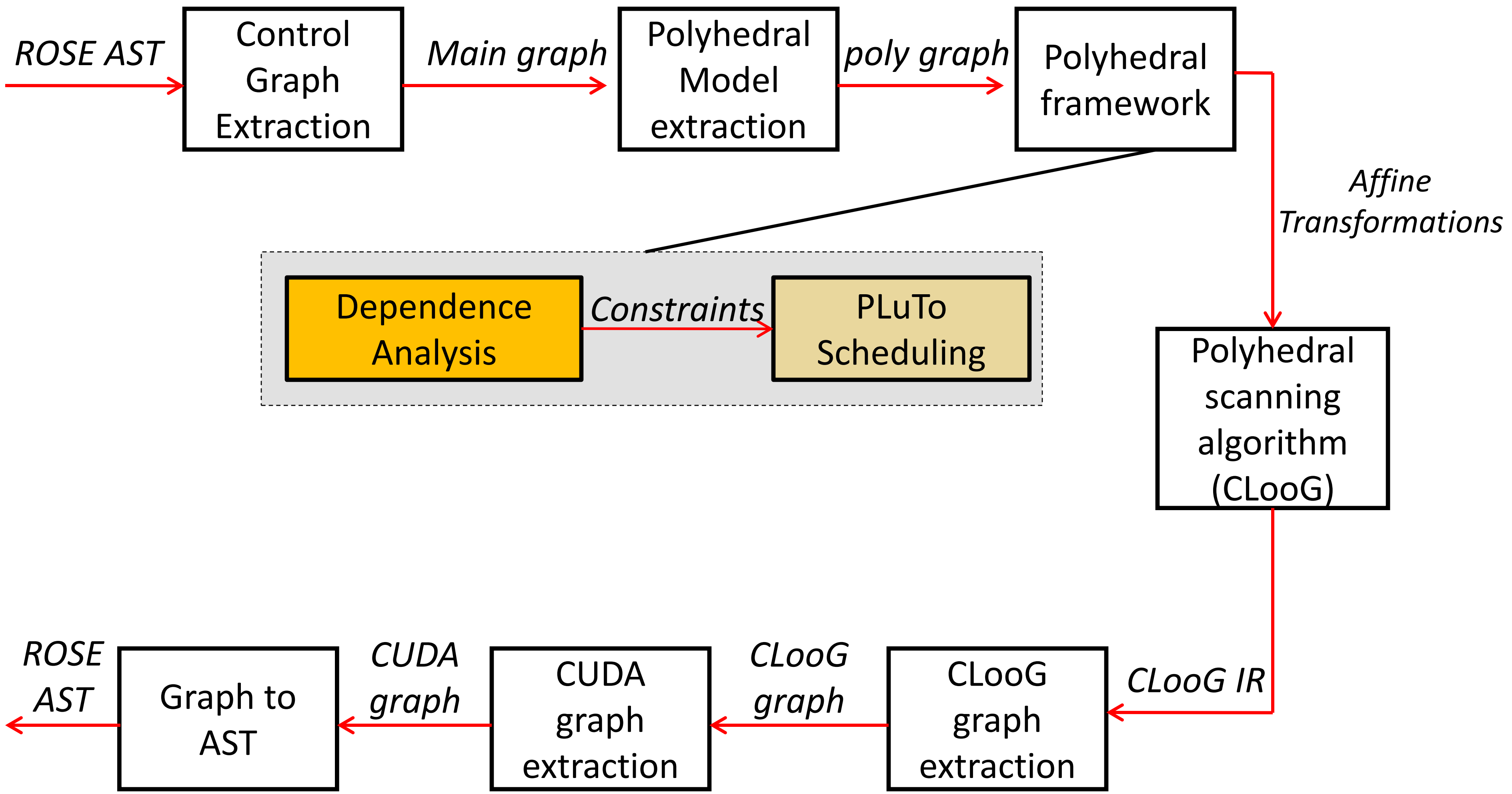
Paul H. J. Kelly

Imperial College London

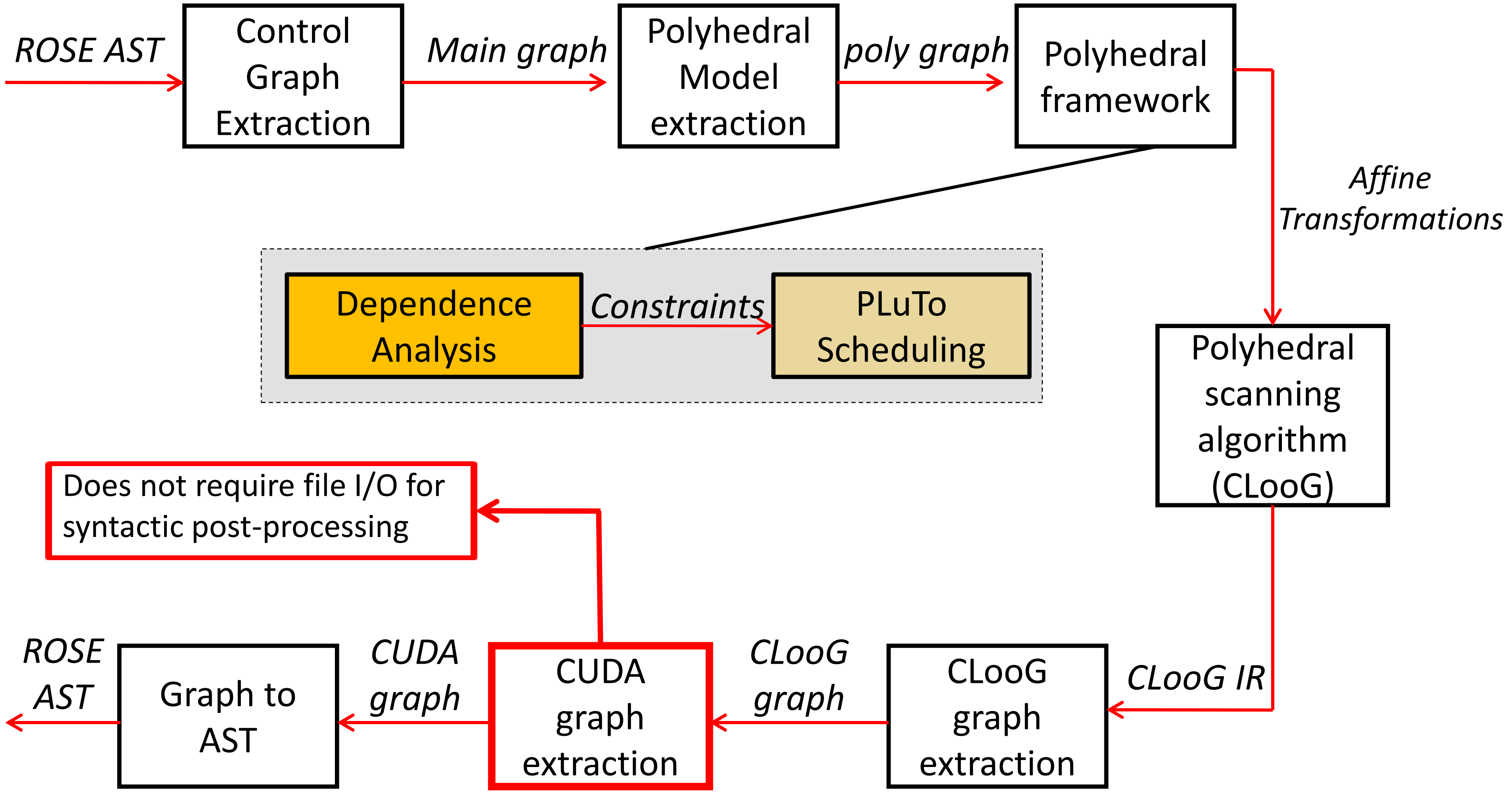
About Me

- PhD student at Imperial College London supervised by Paul H. J. Kelly.
- Compiler and Language support for Heterogeneous parallel architectures (e.g. GPGPUs, Cell BE, Multicore etc.).
- Developing our own source-to-source polyhedral compiler (CUDA back-end).
- Sponsored by EPSRC and Codeplay Software Ltd.

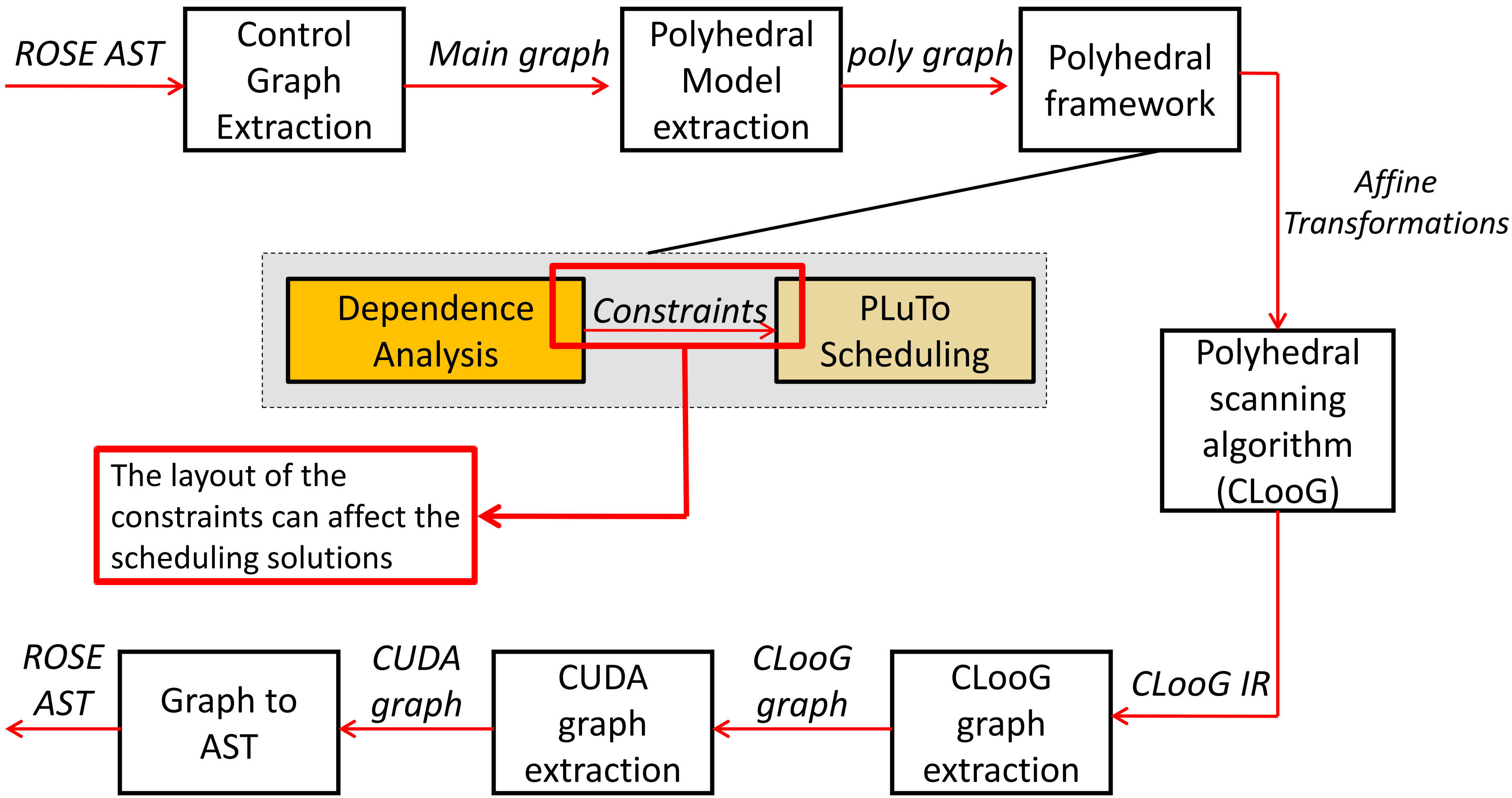
Our Polyhedral Framework



Our Polyhedral Framework



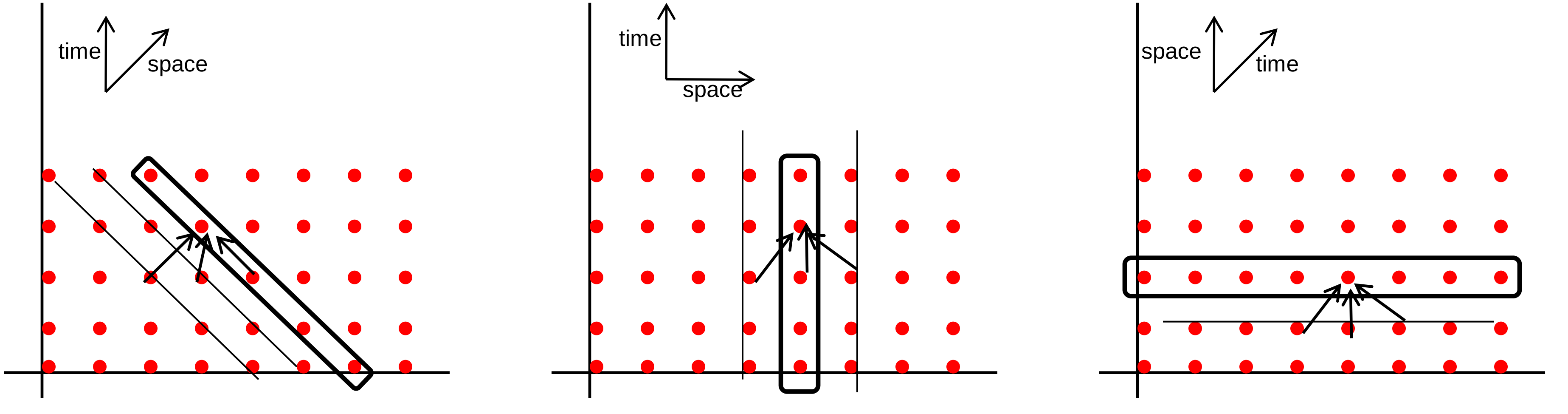
Our Polyhedral Framework



Imperial College London

PLuTo Scheduling Algorithm 1

- Iteratively looks for a maximal set of linearly independent affine transforms of the original iteration space.
- An affine transform is a hyperplane representing a loop in the transformed iteration space.
- Each hyperplane needs to respect a set of constraints that guarantee legality and minimum communication between hyperplane instances (i.e. between different loop iterations).



Imperial College London

PLuTo Scheduling Algorithm 2

- For each Statement $\boldsymbol{S_i}$ find a schedule $\boldsymbol{\Phi_i}(\vec{\boldsymbol{x}}_i) = \underbrace{\left[\boldsymbol{\Phi_{i0}}\ \boldsymbol{\Phi_{i1}}\ \cdots\ \boldsymbol{\Phi_{iN}}\right]}_{\textit{MAX + scalar dimensions}}^{\mathbf{T}} \cdot \vec{\boldsymbol{x}}_i$

- Each hyperplane $\boldsymbol{\Phi_{ij}}$ of statement $\boldsymbol{S_i}$ is a solution to a global constraint matrix consisted of the following constraints for each dependence edge :

Global Constraint Matrix M → Empty
M ← Legality Constraints $\left[\boldsymbol{\Phi_{dj}} \cdot \vec{x}_d - \boldsymbol{\Phi_{sj}} \cdot \vec{x}_s \geq 0\right]$
M ← Communication Bounding $[\mathbf{cost} \geq \boldsymbol{\Phi_{dj}} \cdot \vec{x}_d - \boldsymbol{\Phi_{sj}} \cdot \vec{x}_s]$
M ← Non-Trivial Solution Constraints $[\boldsymbol{\Phi_{ij}} \cdot \vec{x}_i \geq 1]$
Solution $\boldsymbol{\Phi_{ij}}$ ← **Solve(**M**)**

- **Solve(**M**)** → Uses a Parametric Integer Programming Library (PIP) to find the lexicographic minimum solution.

Imperial College London

PLuTo Scheduling Algorithm 3

- Iteratively find as many linearly independent solution as possible

Global Constraint Matrix M → *Empty*
M ← Legality
M ← Communication Bounding
M ← Non-Trivial solution
While **(** Solve(*M*) **) {**
    M ← Linear Independence
}

PLuTo Scheduling Algorithm 4

- If NO MORE solutions can be found → remove any killed dependences
- If NO solution was found → cut the dependence graph into Strongly Connected Components (SCC) – *loop distribution* – and remove the killed dependences

Global Constraint Matrix M → *Empty*
M ← Legality
M ← Communication Bounding
M ← Non-Trivial solution
While (Solve(*M*) **) {**
M ← Linear Independence
}

Cut in SCC If **NO** solution is found
Remove Killed dependences

Imperial College London

PLuTo Scheduling Algorithm 5

- Iteratively find bands of fully permutable loop nests

do {

Global Constraint Matrix M → *Empty*

M ← Legality

M ← Communication Bounding

M ← Non-Trivial solution

While **(** Solve(*M*) **)** **{**

M ← Linear Independence

}

Cut in SCC If **NO** solution is found

Remove Killed dependences

} **While** **(** (*total_sols < MAX)* **OR** *(deps ≠ 0)* **)**

Imperial College London

Communication Bounding Constraints

- For every dependence edge *e* :

$$\boxed{\mathbf{cost}} \geq \boxed{\Phi_{dj} \cdot \vec{x}_d - \Phi_{sj} \cdot \boxed{h_e(\vec{x}_d)}}$$

h-transformation

Affine Form
on
Structure Parameters

$$\boxed{u(p)} - \boxed{\delta_e(\vec{x}_d)} \geq 0$$

Farkas Lemma

$$u(p) - \delta_e(\vec{x}_d) \equiv \lambda_{e0} + \sum_{k=1}^{m_e} \lambda_{ek} \cdot P_e\,, \qquad \lambda_{ek} \geq 0$$

Constant

Identification

$$min_{\prec}\{\boxed{p_1, p_2, \dots, p_n}, \boxed{w,} \boxed{\dots, a_{ij}, \dots}\}$$

Parameters

Unknown schedule
coefficients

Ordering Sensitivity 1

$$min_{\prec}\{\underbrace{p_1, p_2, \ldots, p_n, w}_{\textbf{Cost}}, \underbrace{\ldots, a_{ij}, \ldots}_{\textbf{Ordering of Transformation Coefficients}}\}$$

- For the same **Cost** the solution we will get from the PIP solver will eventually depend on the **ordering of the transformation coefficients**.

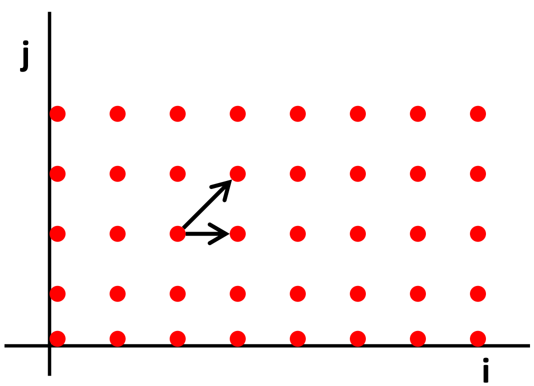
Imperial College London

Ordering Sensitivity (example)

$min_{\prec}\{\boxed{N}, \boxed{w}, \ldots\}$

$\boxed{0 \quad 1}$ → **Cost = 1**

- Minimum **Cost** is **1**.
- No outer parallel loop.




```
for i = 0,N
   for j = 0, N
      A[i][j] = A[i-1][j]*A[i-1][j-1]
```

Imperial College London

Ordering Sensitivity (example)

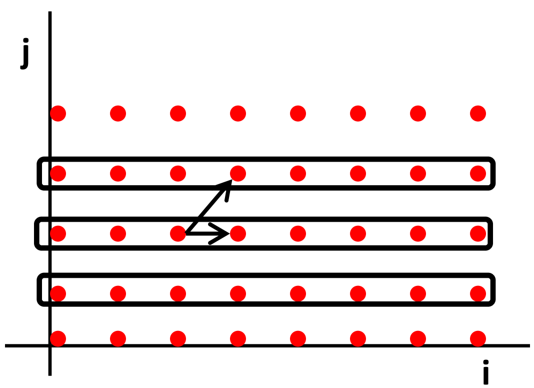
- By changing the order of the transformation coefficients we get **two different solutions** both having **Cost = 1**.

Cost = 1

Order 1 : $min_{\prec}\{\boxed{N,w},\boxed{a_i},\boxed{a_j},\boxed{c}\}$

$\boxed{0 \quad 1 \quad 0}$

$\Phi_1\begin{pmatrix} i \\ j \\ c \end{pmatrix} = j$

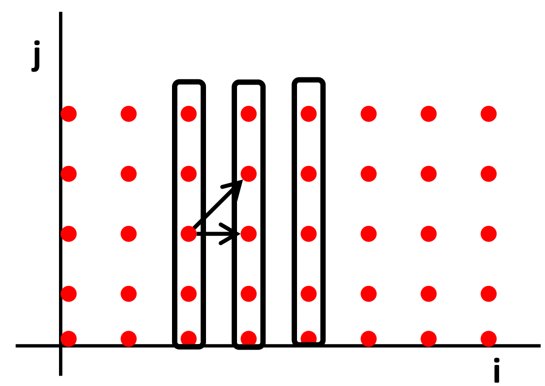


Cost = 1

Order 2 : $min_{\prec}\{\boxed{N,w},\boxed{a_j},\boxed{a_i},\boxed{c}\}$

$\boxed{0 \quad 1 \quad 0}$

$\Phi_1\begin{pmatrix} i \\ j \\ c \end{pmatrix} = i$



Imperial College London

Ordering Sensitivity (example)

- By adding the **linear independence constraints** we get a **second solution**.
- Order 2 yields an **inner loop** that is **fully parallel**.
- Which solution/order is better ?

Cost = 1

Order 1 : $min_{\prec}\{\boxed{N,w},\boxed{a_i},\boxed{a_j},\boxed{c}\}$

1 0 0

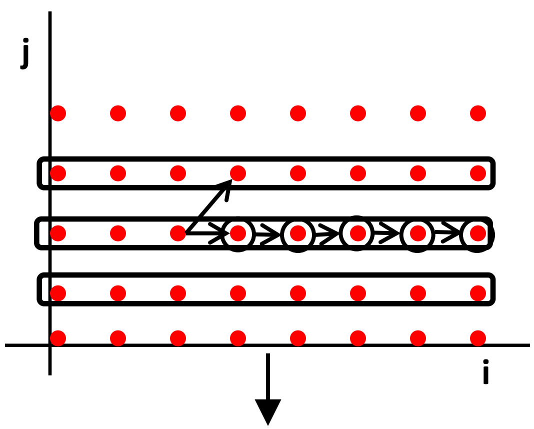
$$\Phi_2\begin{pmatrix} i \\ j \\ c \end{pmatrix} = i$$

Cost = 1

Order 2 : $min_{\prec}\{\boxed{N,w},\boxed{a_j},\boxed{a_i},\boxed{c}\}$

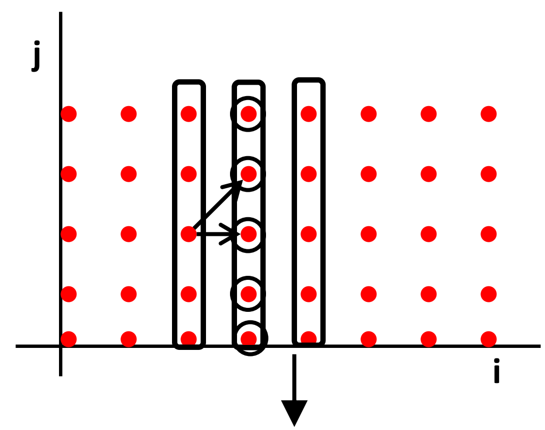
1 0 0

$$\Phi_2\begin{pmatrix} i \\ j \\ c \end{pmatrix} = j$$



Pipeline/Wavefront

$$\Phi\begin{pmatrix} i \\ j \\ c \end{pmatrix} = \{\Phi_1, \Phi_2\}$$



Fully Parallel Inner Loop

Imperial College
London

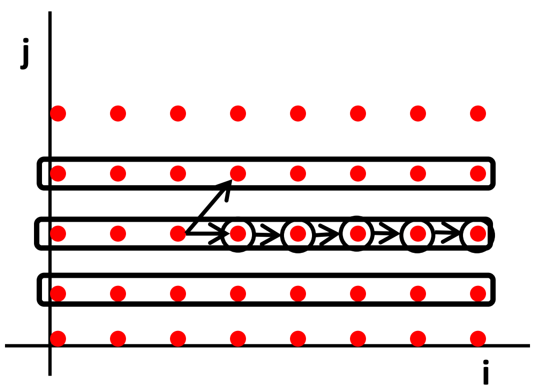
Pipeline Degrees of Parallelism

- ***N*** Non-parallel loops can be **transformed into a *wavefront/pipeline*** consisted of one sequential and ***N-1*** parallel loops i.e. degrees of parallelism.

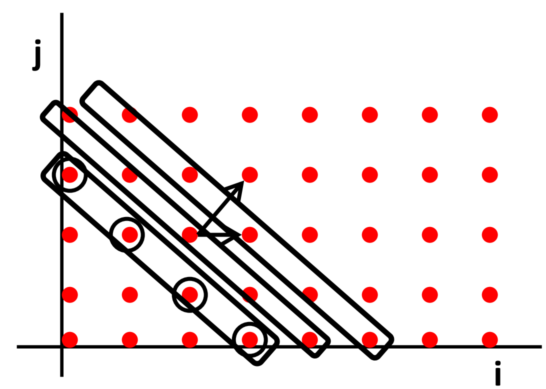
Wavefront/pipeline

$$\boxed{\begin{pmatrix} \Phi'_1 \\ \Phi'_2 \\ \vdots \\ \Phi'_n \end{pmatrix}} = \begin{pmatrix} 1 & 1 & \dots & 1 & 1 \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix} \cdot \boxed{\begin{pmatrix} \Phi_1 \\ \Phi_2 \\ \vdots \\ \Phi_n \end{pmatrix}}$$

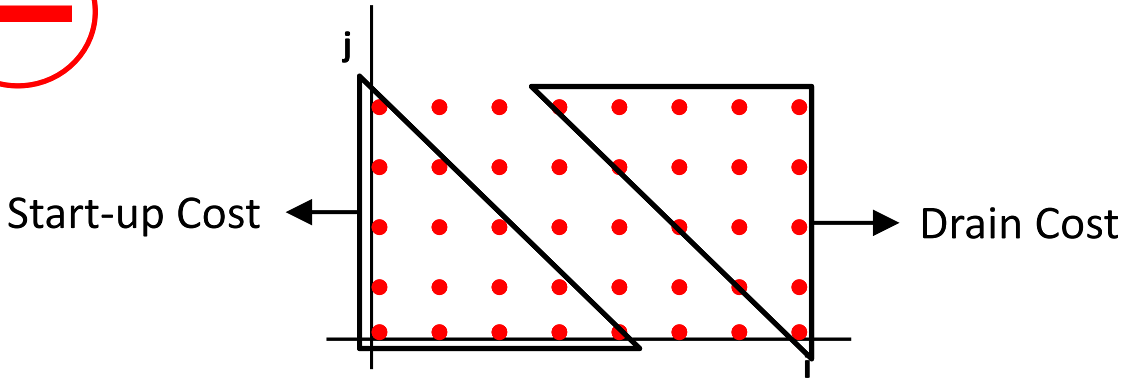
Non-parallel loops

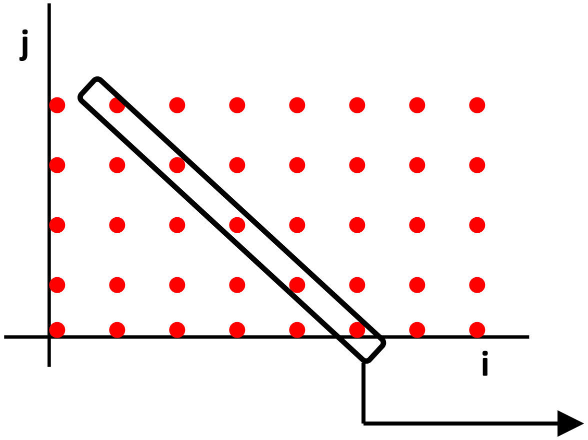


$$\begin{pmatrix} j \\ i \end{pmatrix} \longrightarrow \begin{pmatrix} i+j \\ j \end{pmatrix}$$



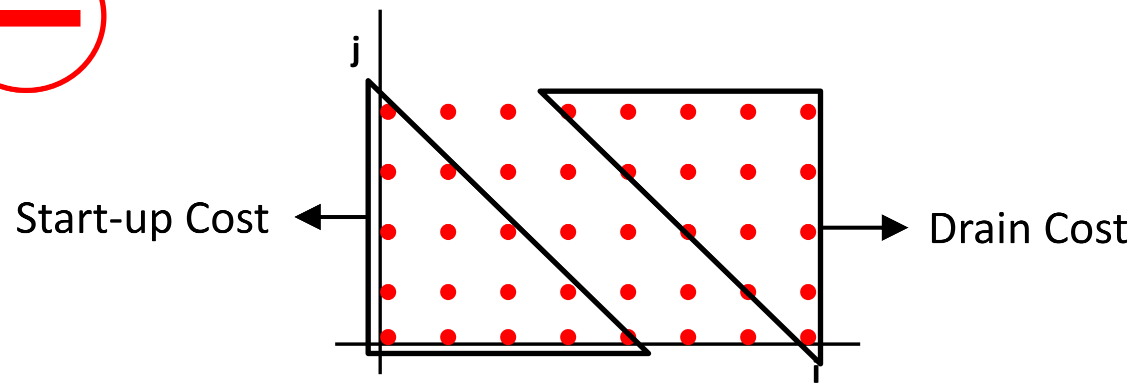
Pipeline Degrees of Parallelism



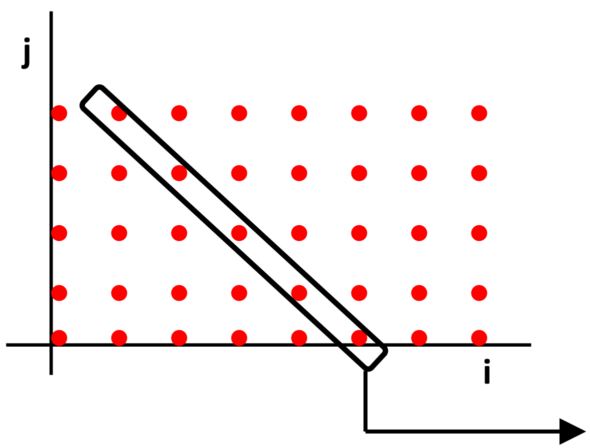


Better *spatial/temporal* Locality
along a wavefront

Pipeline Degrees of Parallelism

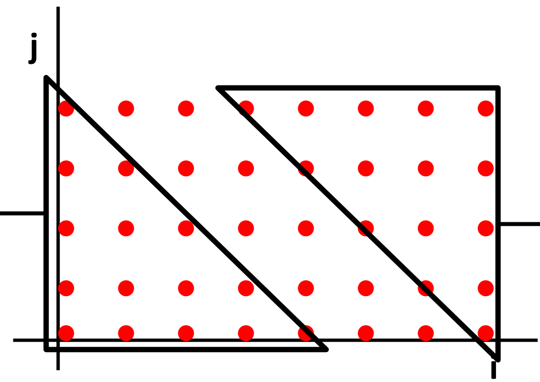


Depend on structure parameters



Better *spatial/temporal* Locality along a wavefront

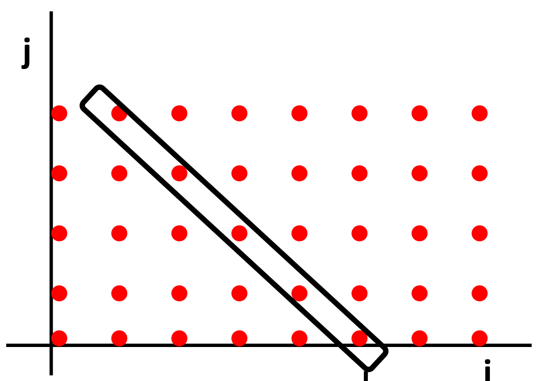
Pipeline Degrees of Parallelism



Start-up Cost

Drain Cost

Depend on structure parameters



Number of *Read-after-Read* dependences that lie within the wavefront

Better *spatial/temporal* Locality along a wavefront

Imperial College London

Fully Parallel vs Pipeline Degrees of Parallelism 1

- We propose a way of distinguishing between **fully parallel** and **pipeline** degrees of parallelism.
- We use **dependence direction vectors** in order to expose **inner fully parallel** degrees of parallelism.

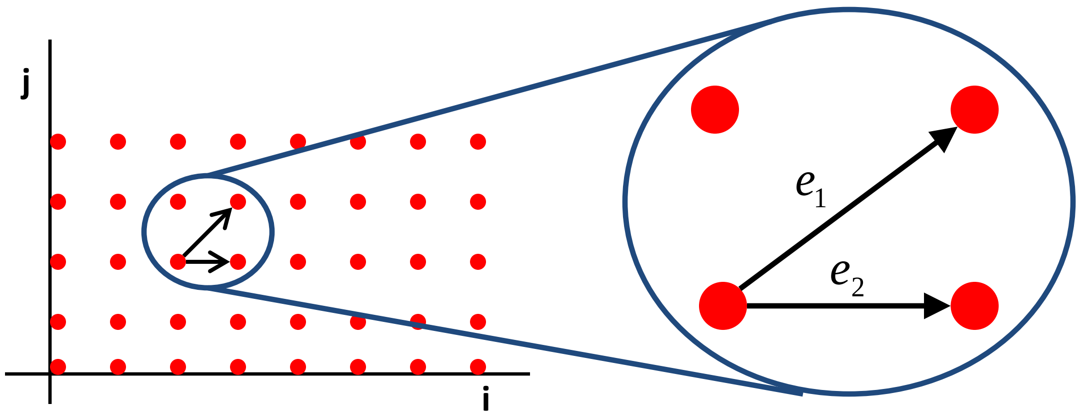
Direction Information :

$$\forall e \in E \xrightarrow[0 \leq i \leq min(dim_{dest}, dim_{src})]{\text{bit vector}} \vec{V_e}[i] = \begin{cases} 1, & \text{If } e \text{ extends along } i \\ 0, & \text{If } e \text{ does not extend along } i \end{cases}$$

$$\forall e \in E \xrightarrow{\text{Boolean}} H_e = \begin{cases} true, & \text{If } e \text{ extends in } \textbf{only 1} \text{ dimension} \\ false, & \text{If } e \text{ extends in } \textbf{more than 1} \text{ dimensions} \end{cases}$$

Imperial College London

Fully Parallel vs Pipeline Degrees of Parallelism 2



$$\vec{V}_{e1} = [1 \quad 1], \quad H_{e1} = false$$

$$\vec{V}_{e2} = [1 \quad 0], \quad H_{e1} = true$$

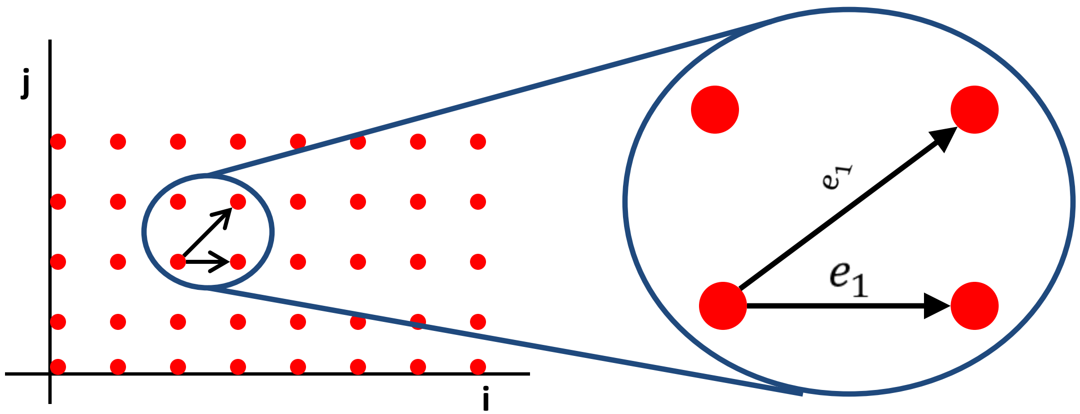
- The **disjunction** of all bit vectors that extend in **only 1** dimension ($H_e = true$) will **expose inner fully parallel loops**.

$$\boxed{\vec{V}_{tot} = \bigvee_{\forall e \in E, H_e = true} \vec{V}_e}$$

- If $\vec{\mathbf{V}}_{\mathbf{tot}}[\mathbf{i}] = \mathbf{0}$ **then put $\boldsymbol{a_i}$ in leading minimization position**.

Imperial College London

Fully Parallel vs Pipeline Degrees of Parallelism 3



$$\vec{V}_{tot} = \begin{array}{cc} i & j \\ \left[1\right. & \left.\boxed{0}\right] \end{array}$$

Fully parallel dimension

- By placing the coefficients of **fully parallel dimensions** in **leading minimization positions** we are effectively **pushing** them towards inner nest levels.
- As a result **fully parallel degrees of parallelism** can be **recovered**.

Imperial College London

Conclusions

- The PLuTo scheduling algorithm iteratively finds **affine transformations** that **minimize communication**.
- For the **same minimum communication** the solution might be **sensitive to the ordering of the affine transformation coefficients** in the global constraint matrix.
- We might have to **choose** between **fully parallel** and **pipeline** degrees of parallelism.
- We **propose** a method for **distinguishing** between **fully parallel** and **pipeline** degrees of parallelism.
- We use **dependence direction information** in order to **expose inner fully parallel loops**.

Thank You !

Any Questions ?