

Determining the idle time of a *K*-dimensional tiling

Karin Högstedt
Larry Carter, Jeanne Ferrante

Department of Computer Science and Engineering
University of California, San Diego

Goal

Optimize tiling choices for minimal execution time

Execution time of a tiling

$$ExecTime_k = Wasted_k + Ideal_k$$

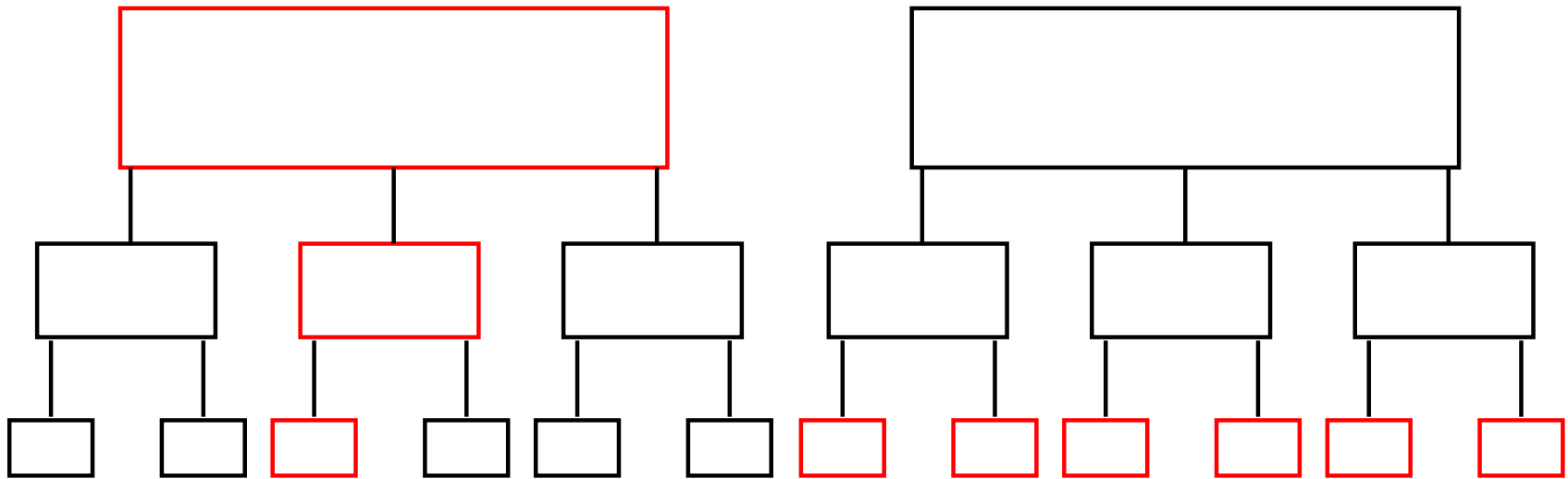
$$Ideal_k = ExecTime_{k-1} \frac{NumTiles_k}{NumProcs_k}$$

$$Wasted_k = UselessWork_k + IdleTime_k$$

$$IdleTime_k = IdleTimePara_k + IdleTimeLoc_k$$

$$UselessWork_k = LoopOverhead_k + DataMovementOverhead_k$$

Idle time due to locality vs parallelism



Execution time of a tiling

$$ExecTime_k = Wasted_k + Ideal_k$$

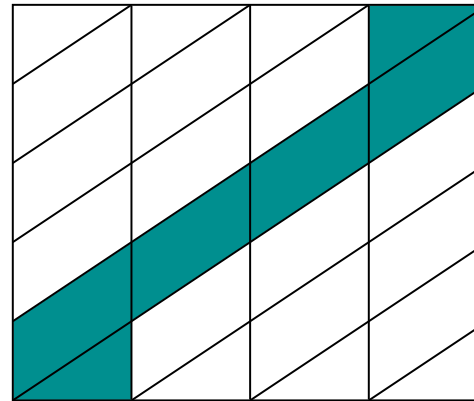
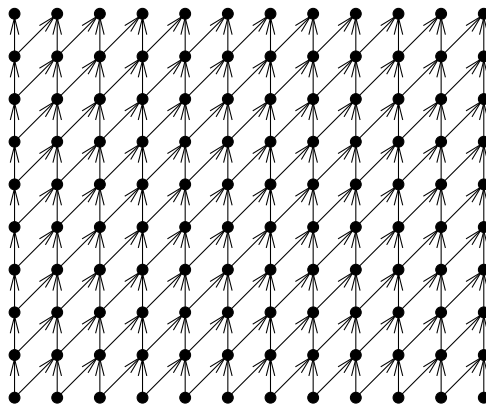
$$Ideal_k = ExecTime_{k-1} \frac{NumTiles_k}{NumProcs_k}$$

$$Wasted_k = UselessWork_k + IdleTime_k$$

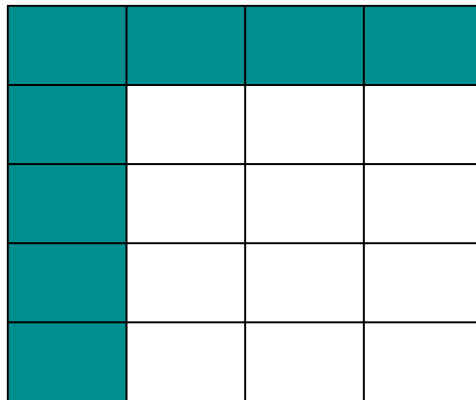
$$IdleTime_k = IdleTimePara_k + IdleTimeLoc_k$$

$$UselessWork_k = LoopOverhead_k + DataMovementOverhead_k$$

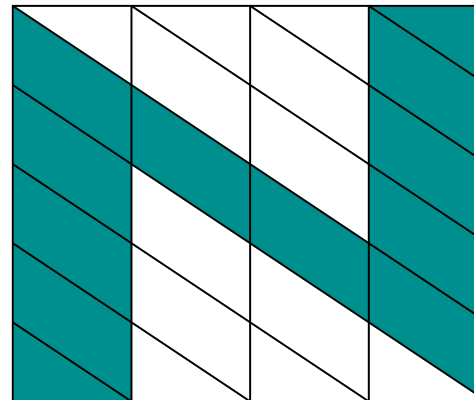
Idle time is a function of the *rise*



$$E_k = 5E_{k-1}$$
$$r = -1$$



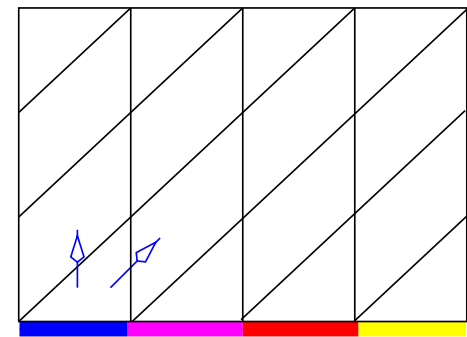
$$E_k = 8E_{k-1}$$
$$r = 0$$



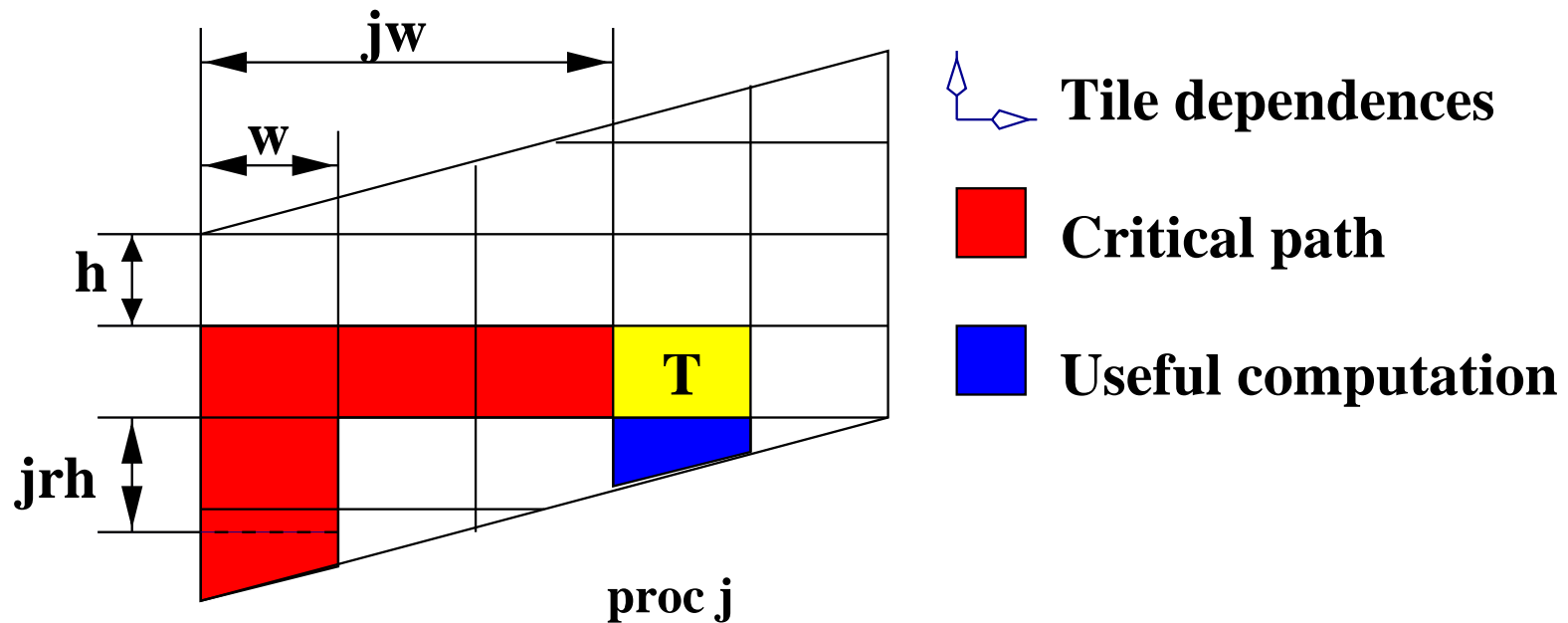
$$E_k = 11E_{k-1}$$
$$r = 1$$

Assumptions [Högstedt et al 97]

- Source code
 - Perfectly nested loops
 - Dependence distances known and uniform
- Iteration space
 - Two-dimensional
 - Two parallel sides
- Tiling
 - Full tiles are parallelogram-shaped
 - One side parallel to iteration space
- Block and block cyclic distribution
- Execution time of tiles proportional to volume

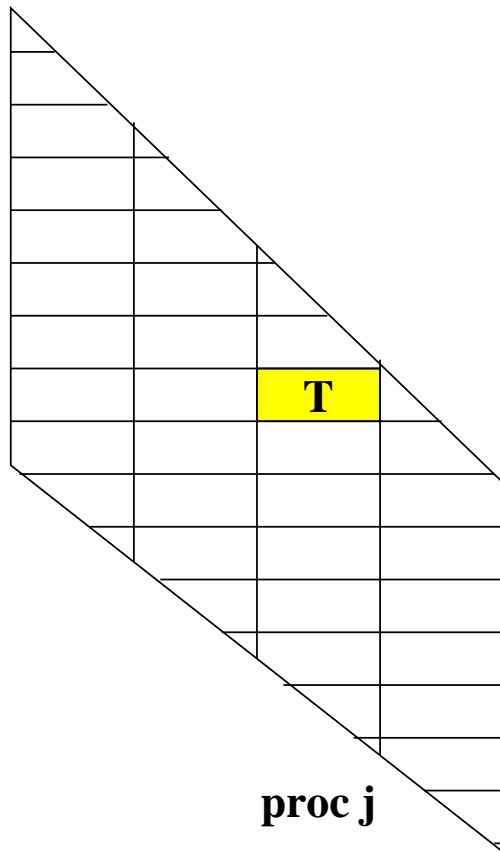


Idle time when $rise \geq -1$



$$\begin{aligned}
 \text{Idle time} &= \text{Red L-shaped area} - \text{Blue triangle} \\
 &= jhw + jrhw = j(1+r)hw
 \end{aligned}$$

Idle time when $rise < -1$



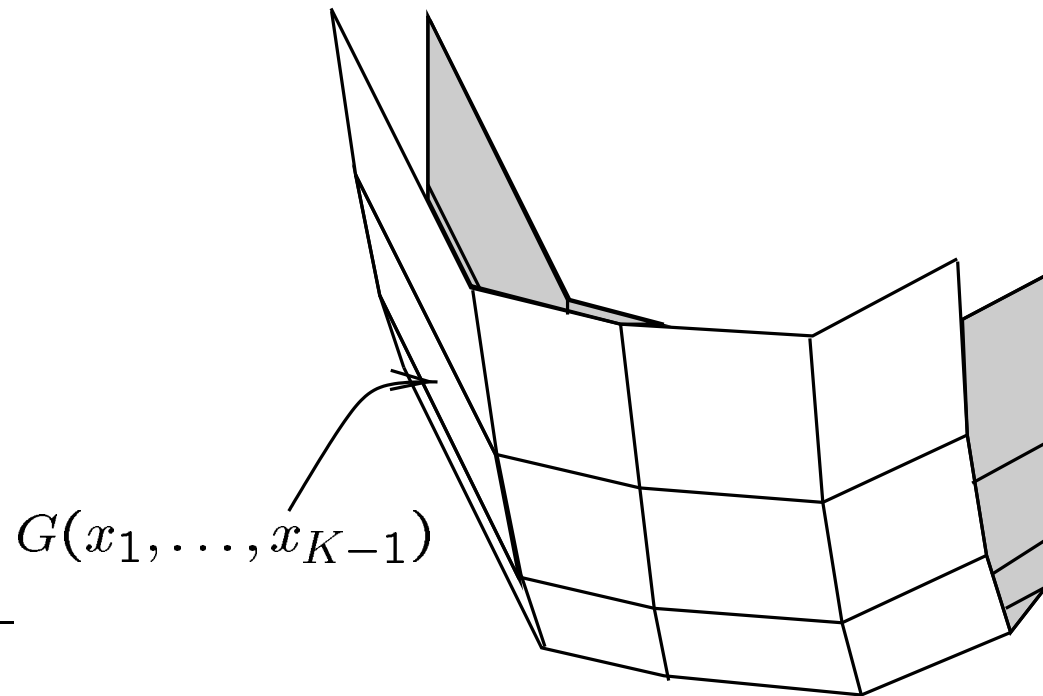
Idle Time=0



Assumptions

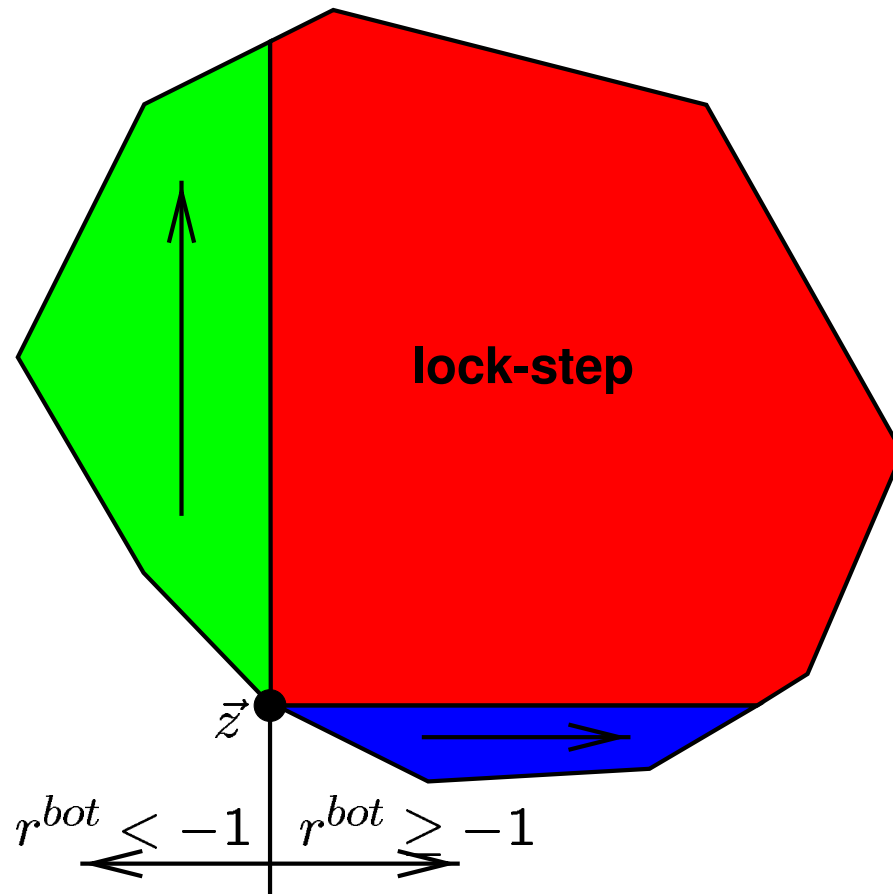
- Source code
 - Perfectly nested loops
 - Dependence distances known and uniform
- Architecture
 - (non-)overlappable communication
- Tiling
 - Parallelogram-shaped tiles
 - Vertical stacks
- Block and block cyclic distribution
- Execution time of partial tiles proportional to volume

Rectilinear iteration spaces

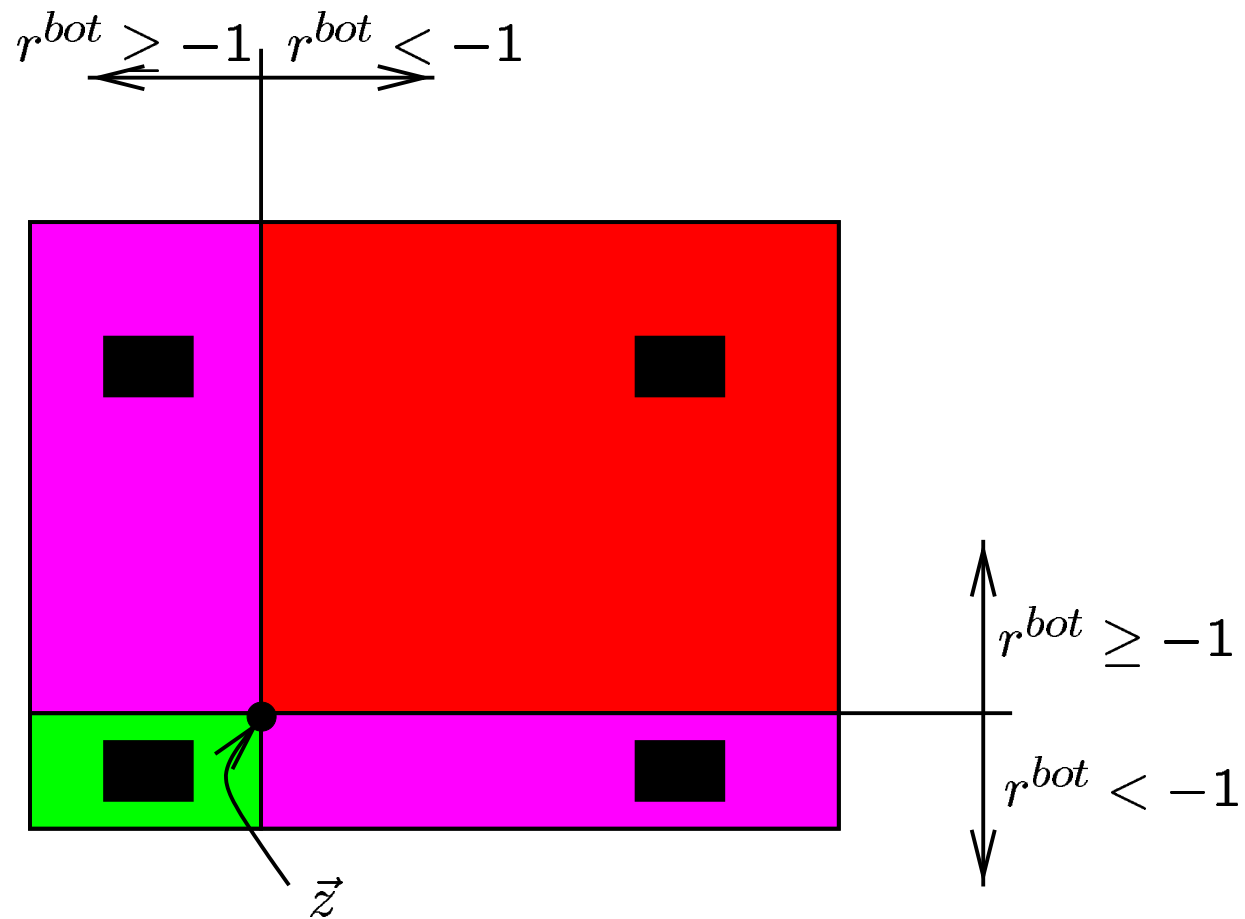


A *rectilinear* iteration space is any convex iteration space such that $\forall d : 1 \leq d \leq K - 1 \left(\frac{\delta G(x_1, \dots, x_{K-1})}{\delta x_d} = g(x_d) \right)$

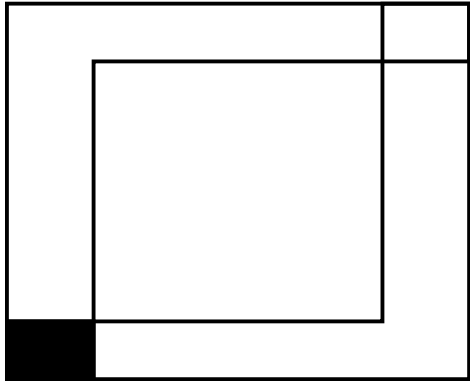
Critical path, $K = 2$ dimensions



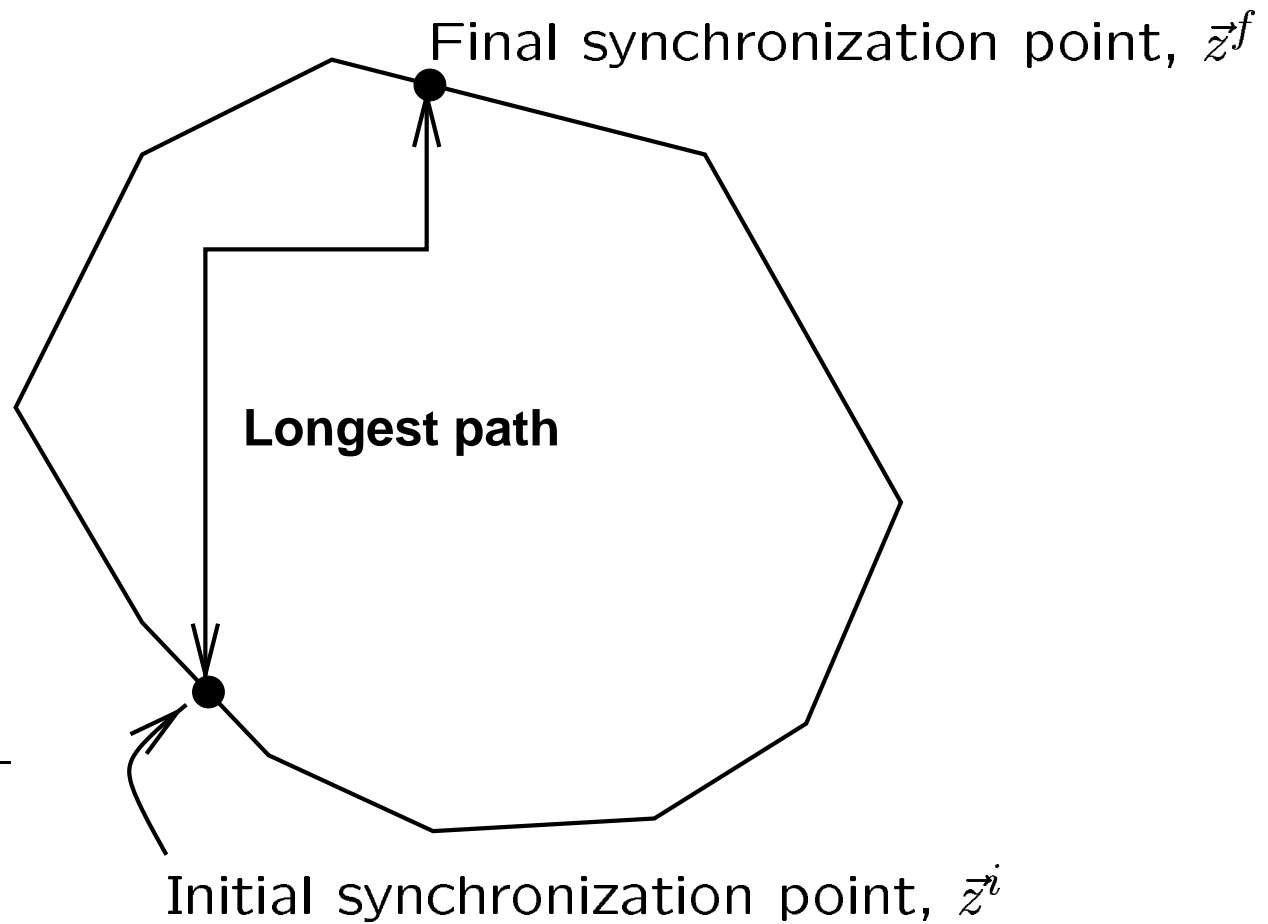
Critical path, $K = 3$ dimensions



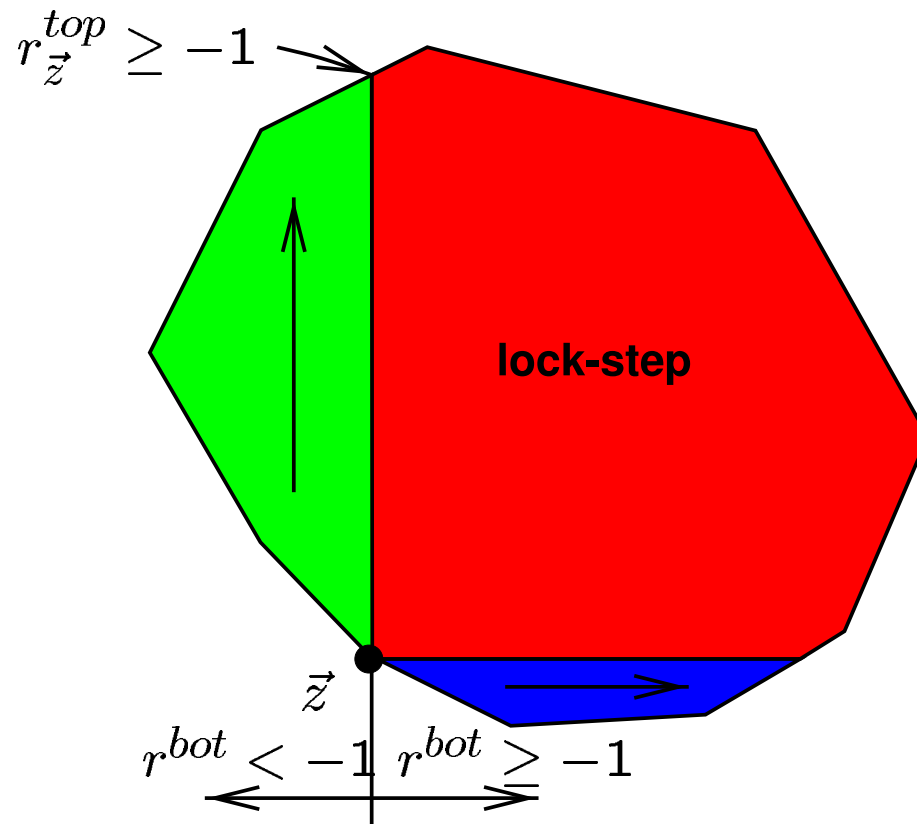




Longest path in iteration space



Initial&final synchronization point, $r_{\vec{z}}^{top} \geq -1$



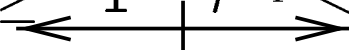


$\vec{z}^i =$

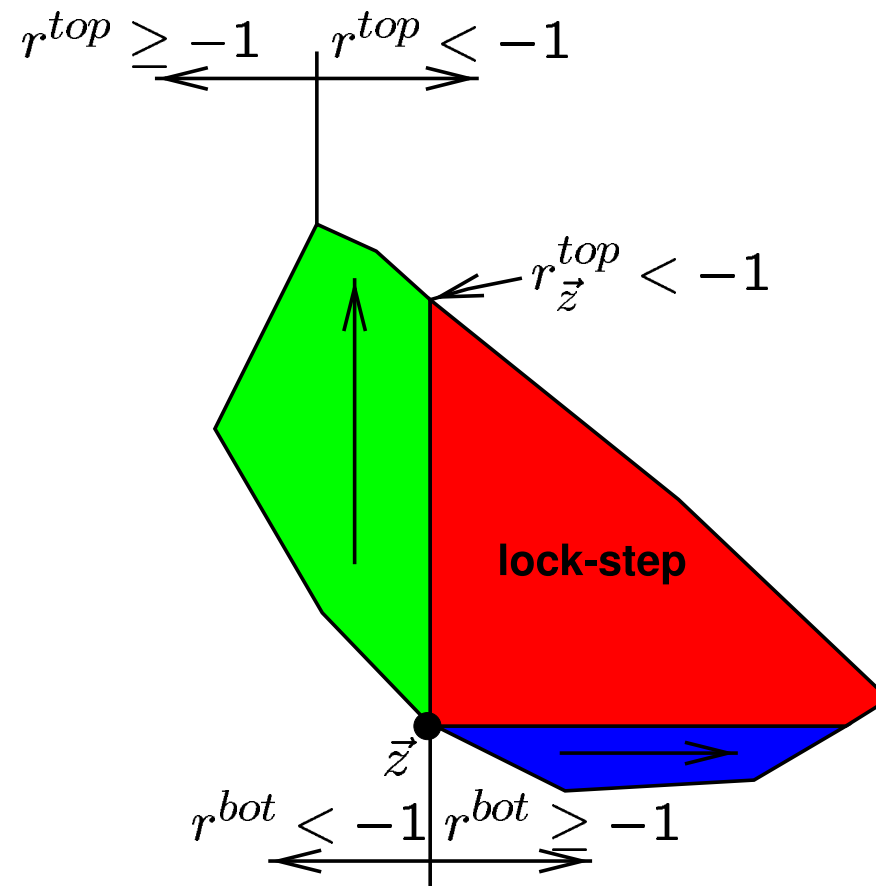


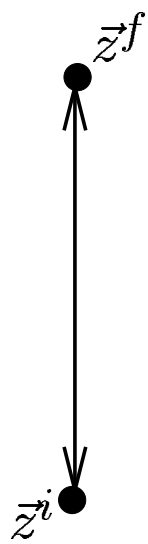
\vec{z}^f

$r^{top} \geq -1$ $r^{top} < -1$



Initial&final synchronization point, $r_{\vec{z}}^{top} < -1$





Solved and open problems

- Facts:
 - Finding the critical path for specific tile in any convex iteration space is *as hard as* general linear programming
 - We have derived simple, solvable formulas for finding the critical path of a rectilinear iteration space
 - One can divide any rectilinear iteration space into regions, each with a different execution pattern
- Questions:
 - Is finding the critical path for any convex iteration space as hard as general linear programming?
 - Can one divide any convex iteration space into regions, each with a different execution pattern?