Symmetry Breaking for Multi-Criteria Mapping and Scheduling on Multicores

Pranav Tendulkar   Peter Poplavko   Oded Maler

Verimag, FRANCE

August 2013

Context

- Typical in parallel programming: spawn multiple identical tasks
  - data parallelism
  - obtain hyperperiod of a multi-periodic system
  - duplicate tasks for fault-tolerance

Context

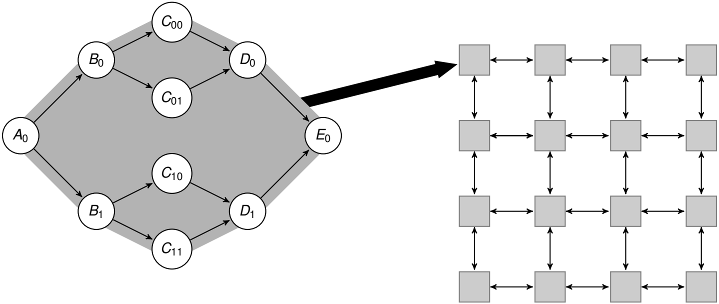
- Typical in parallel programming: spawn multiple identical tasks
  - data parallelism
  - obtain hyperperiod of a multi-periodic system
  - duplicate tasks for fault-tolerance
- Often the platform have multiple identical processors.

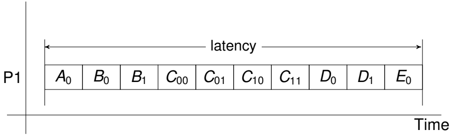
Context

- Typical in parallel programming: spawn multiple identical tasks
  - data parallelism
  - obtain hyperperiod of a multi-periodic system
  - duplicate tasks for fault-tolerance
- Often the platform have multiple identical processors.
- Hence, symmetry in the solution space.

Multi-criteria Optimization

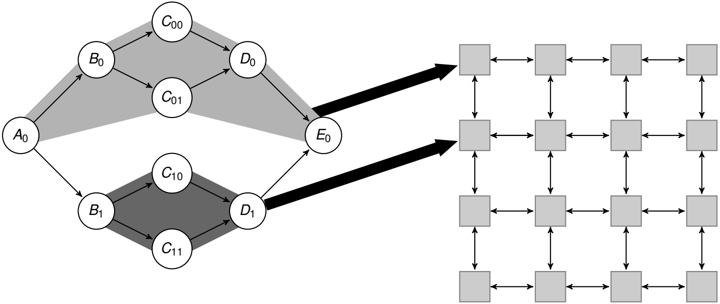
minimize latency using minimal number of processors

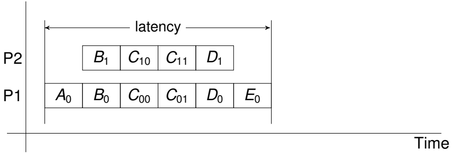




Multi-criteria Optimization

minimize latency using minimal number of processors





Contribution

context:

static mapping and scheduling for programs with data-parallelism

multi-criteria optimization using SMT solvers

Contribution

context:

static mapping and scheduling for programs with data-parallelism

multi-criteria optimization using SMT solvers

symmetry breaking in solution space for identical tasks and processors

Contribution

context:

static mapping and scheduling for programs with data-parallelism

multi-criteria optimization using SMT solvers

symmetry breaking in solution space for identical tasks and processors

goal: increase the tractable problem size of SMT solvers

experiments : problem size increase from 20 to 50 tasks

Outline

1. Motivation
2. Application Model
3. Problem Formulation - SMT
4. Symmetry Breaking
5. Cost Space Exploration
6. Experiments and Results
7. Conclusions

Outline

Model of Computation

synchronous dataflow graphs (SDF)

- by E. Lee and D. Messerschmitt in 1987
- task graph + symbolic representation of data parallelism
- signal-processing, video-coding applications

a ‘standard’ in academic multicore compilers:

- StreamIt compiler of MIT

Model of Computation

synchronous dataflow graphs (SDF)

- by E. Lee and D. Messerschmitt in 1987
- task graph + symbolic representation of data parallelism
- signal-processing, video-coding applications

a ‘standard’ in academic multicore compilers:

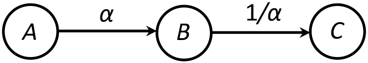
- StreamIt compiler of MIT

we introduce split-join graphs : restriction of SDF

- still covering perhaps 90% of use cases

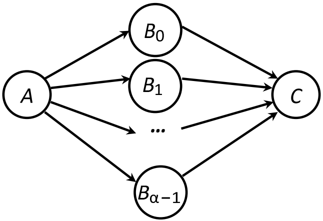
Split-Join Graphs

a simple split-join graph example:



α : spawn and split

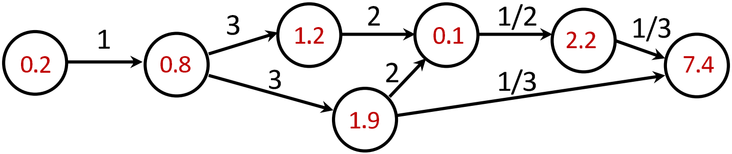
$1/\alpha$: wait and join



Split-Join Graphs

Definition (Split-Join Graph)

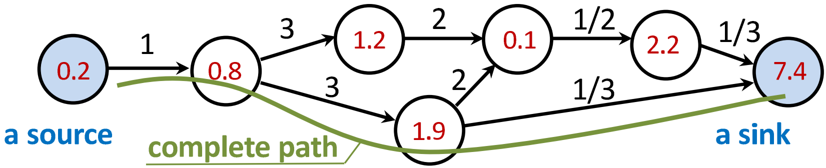
$S = (V, E, d, \alpha)$, (V, E) : DAG, V:actors, E:channels
$d : V \to \mathbb{R}_+$: actor execution time,
$\alpha : E \to \mathbb{Q}$: channel counter: split (> 1), join (< 1) or neutral ($= 1$)



Split-Join Graphs

Definition (Split-Join Graph)

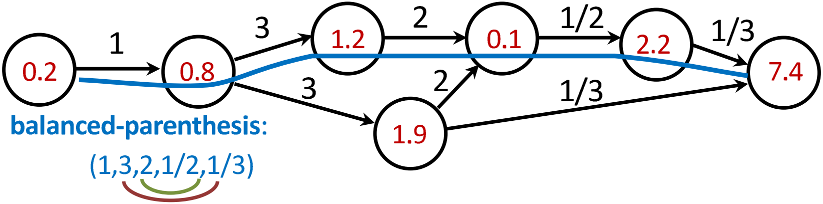
$S = (V, E, d, \alpha)$, (V, E) : DAG, V:actors, E:channels
$d : V \to \mathbb{R}_+$: actor execution time,
$\alpha : E \to \mathbb{Q}$: channel counter: split (> 1), join (< 1) or neutral ($= 1$)



Split-Join Graphs

Definition (Split-Join Graph)

$S = (V, E, d, \alpha)$, (V, E) : DAG, V:actors, E:channels
$d : V \to \mathbb{R}_+$: actor execution time,
$\alpha : E \to \mathbb{Q}$: channel counter: split (> 1), join (< 1) or neutral ($= 1$)



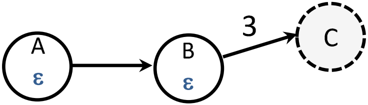
Well-behaved Graphs

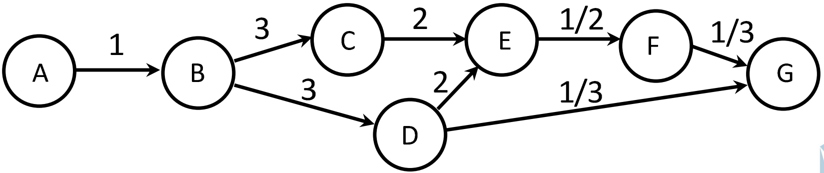
Definition (Well-behaved)

$S = (V, E, d, \alpha)$ is well-behaved if any complete path has balanced-parenthesis signature

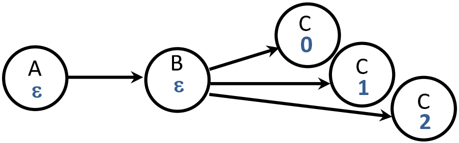
Such a graph can be unfolded to a task graph.

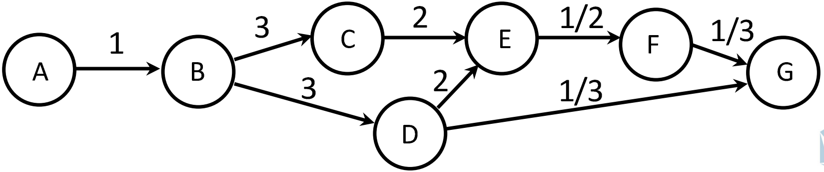
Unfolding to Task Graph



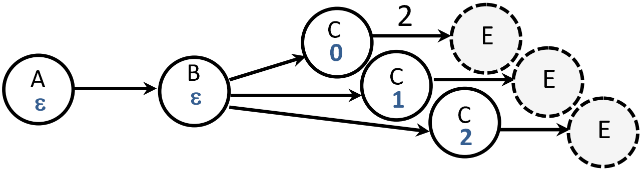


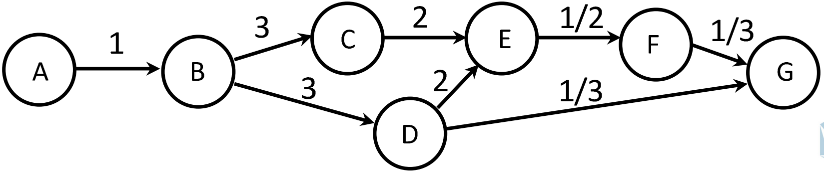
Unfolding to Task Graph



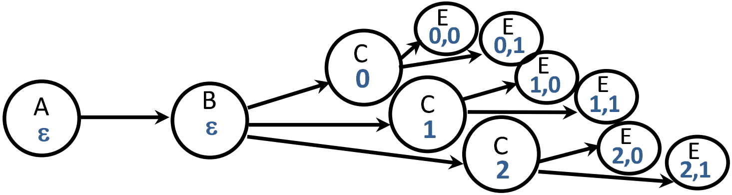


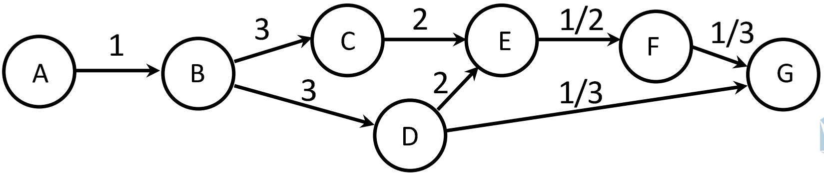
Unfolding to Task Graph



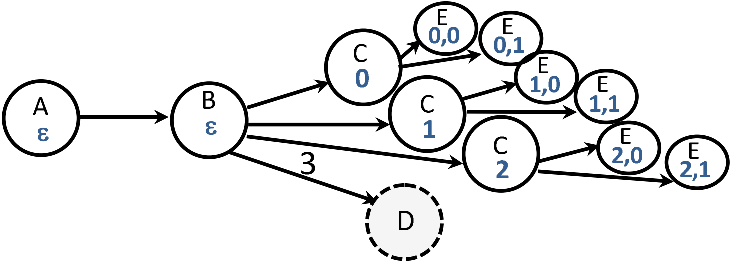


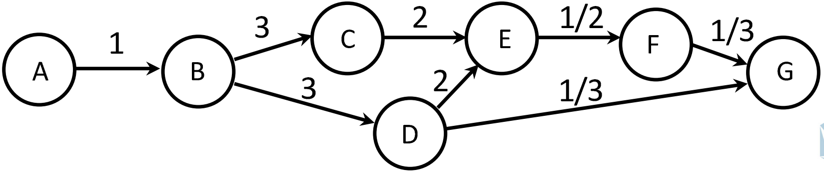
Unfolding to Task Graph



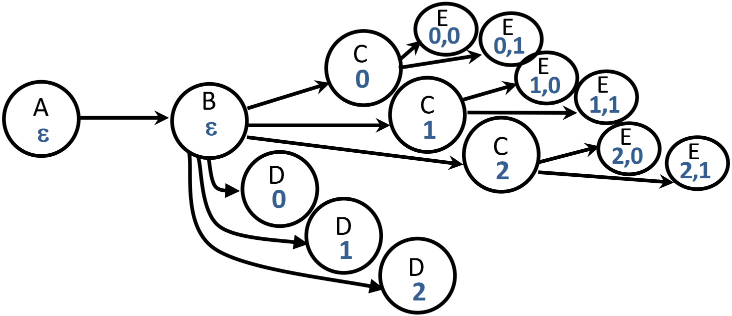


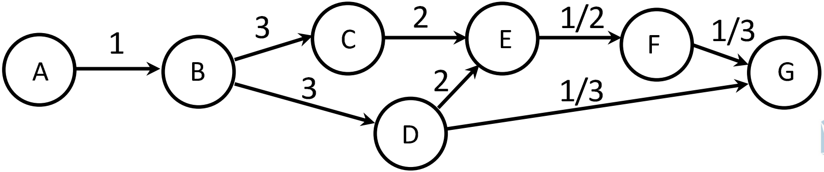
Unfolding to Task Graph



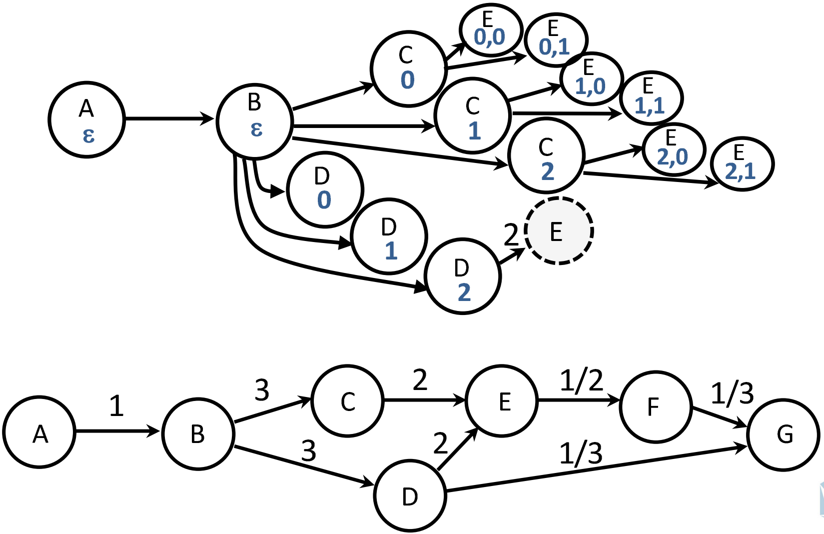


Unfolding to Task Graph

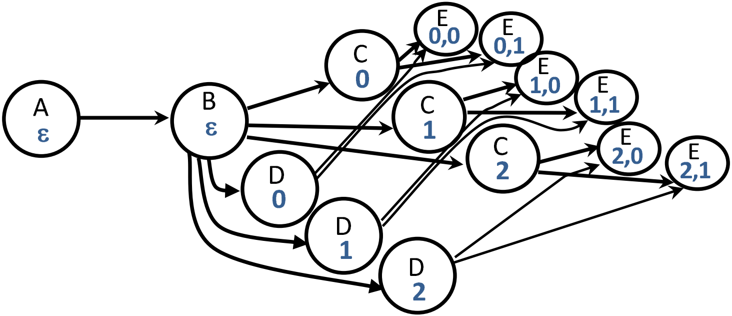


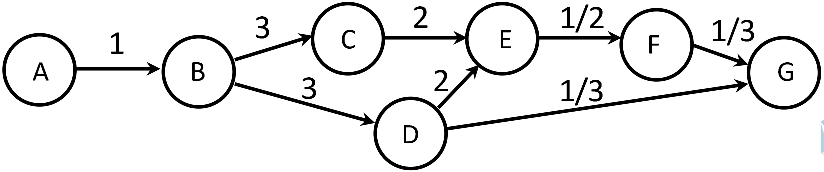


Unfolding to Task Graph

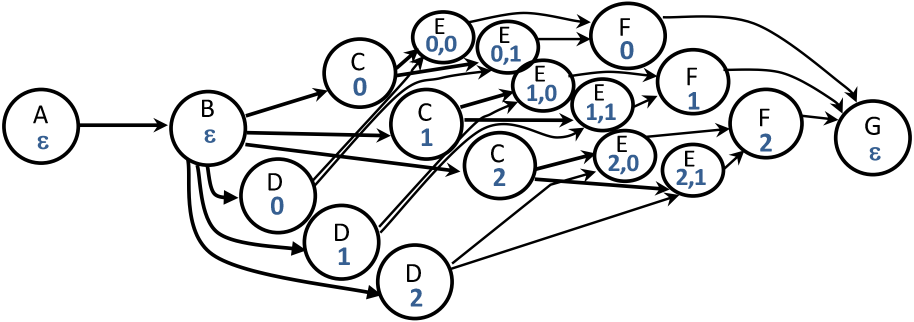


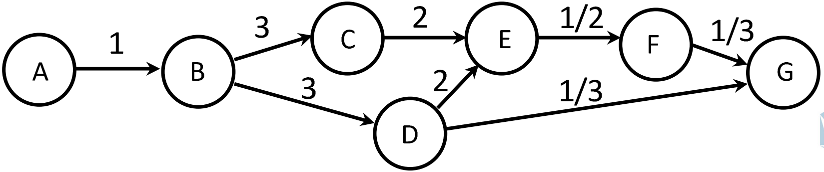
Unfolding to Task Graph



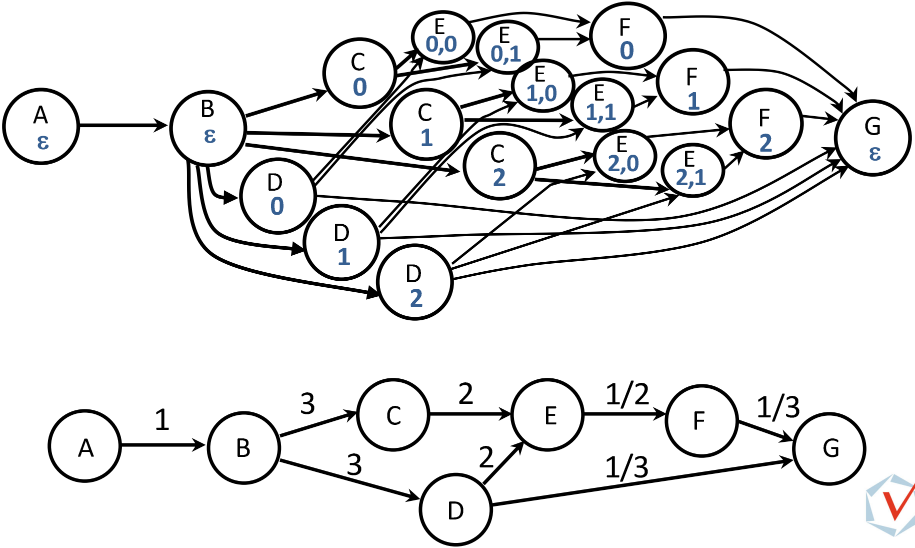


Unfolding to Task Graph



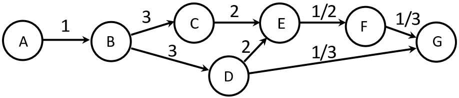


Unfolding to Task Graph



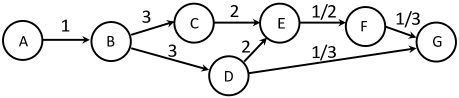
Actors, Tasks, Lexicographic Order

split-join graph: actors *e.g., A, B, C*



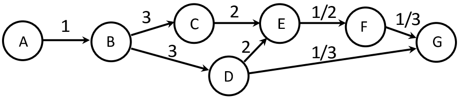
Actors, Tasks, Lexicographic Order

notation for actors: v, $v \in V$

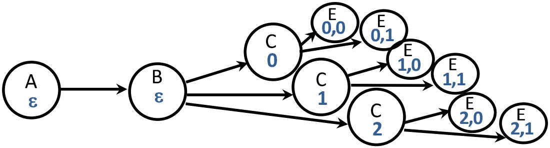


Actors, Tasks, Lexicographic Order

notation for actors: v, $v \in V$

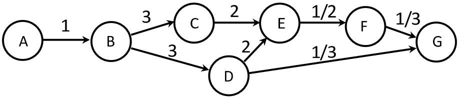


unfolded task graph: tasks *e.g.*, $E_{0,1}$, B, C_2

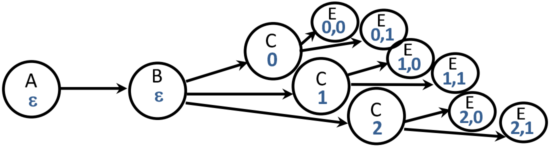


Actors, Tasks, Lexicographic Order

notation for actors: v, $v \in V$

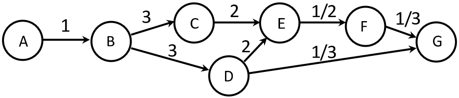


notation for tasks: $u \in U$



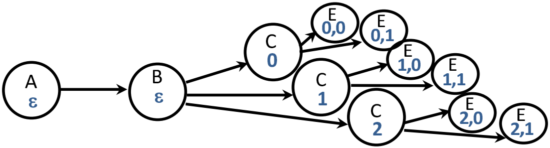
Actors, Tasks, Lexicographic Order

notation for actors: v, $v \in V$



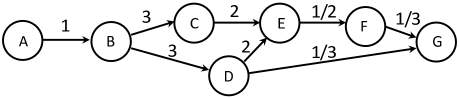
notation for tasks: $u \in U$

$u = v_h$, $v \in V$ and h - hier. index, *e.g.*, $v_h = E_{0,1}$



Actors, Tasks, Lexicographic Order

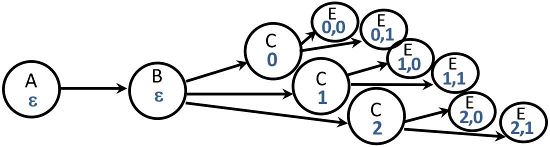
notation for actors: v, $v \in V$



notation for tasks: $u \in U$

$U_v = \{v_h\}$: lexicographically ordered ($\ll$) set of instances of v

$U_E : E_{0,0} \ll E_{0,1} \ll E_{1,0} \ll E_{1,1} \ll E_{2,0} \ll E_{2,1}$



Outline

Multi-criteria Optimization Strategy

Given a split-join graph S, we perform the following steps:

Multi-criteria Optimization Strategy

Given a split-join graph S, we perform the following steps:

- 1. Check whether S is well-behaved
- 2. Unfold S into task graph $T = (U, \mathcal{E}, \delta)$

Multi-criteria Optimization Strategy

Given a split-join graph S, we perform the following steps:

- 1. Check whether S is well-behaved
- 2. Unfold S into task graph $T = (U, \mathcal{E}, \delta)$
- 3. Generate the mapping and scheduling constraints:
  - Precedence
  - Mutual Exclusion
  - Buffer Capacity

Multi-criteria Optimization Strategy

Given a split-join graph S, we perform the following steps:

- 1. Check whether S is well-behaved
- 2. Unfold S into task graph $T = (U, \mathcal{E}, \delta)$
- 3. Generate the mapping and scheduling constraints:
  - Precedence
  - Mutual Exclusion
  - Buffer Capacity (Extended Problem - see the paper)

Multi-criteria Optimization Strategy

Given a split-join graph S, we perform the following steps:

- 1. Check whether S is well-behaved
- 2. Unfold S into task graph $T = (U, \mathcal{E}, \delta)$
- 3. Generate the mapping and scheduling constraints:
  - Precedence
  - Mutual Exclusion
  - Buffer Capacity (Extended Problem - see the paper)
- 4. Cost-space exploration using SMT solver.

Decision variables:

Multi-criteria Optimization Strategy

Given a split-join graph S, we perform the following steps:

- 1. Check whether S is well-behaved
- 2. Unfold S into task graph $T = (U, \mathcal{E}, \delta)$
- 3. Generate the mapping and scheduling constraints:
  - Precedence
  - Mutual Exclusion
  - Buffer Capacity (Extended Problem - see the paper)
- 4. Cost-space exploration using SMT solver.

Decision variables:

- $\mu(u), u \in U$ - the mapping: processor (1,2,. . . ,M) for u

Multi-criteria Optimization Strategy

Given a split-join graph S, we perform the following steps:

- 1. Check whether S is well-behaved
- 2. Unfold S into task graph $T = (U, \mathcal{E}, \delta)$
- 3. Generate the mapping and scheduling constraints:
  - Precedence
  - Mutual Exclusion
  - Buffer Capacity (Extended Problem - see the paper)
- 4. Cost-space exploration using SMT solver.

Decision variables:

- $\mu(u), u \in U$ - the mapping: processor (1,2,...,M) for u
- $s(u)$ - the schedule: start time of u

Constraints

Predicate $\varphi(u, u')$:
task u' starts after the completion of task u

$$\varphi(u, u') : s(u') \geq s(u) + \delta(u)$$

Constraints

Predicate $\varphi(u, u')$:
task u' starts after the completion of task u

$$\varphi(u, u') : s(u') \geq s(u) + \delta(u)$$

Precedence:

$$\bigwedge_{(u,u') \in \mathcal{E}} \varphi(u, u')$$

Constraints

Predicate $\varphi(u, u')$:
task u' starts after the completion of task u

$$\varphi(u, u') : s(u') \geq s(u) + \delta(u)$$

Precedence:

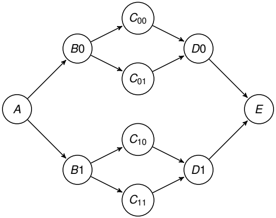
$$\bigwedge_{(u,u') \in \mathcal{E}} \varphi(u, u')$$

Mutual exclusion:

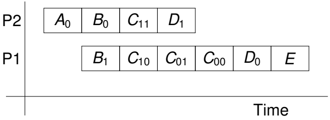
$$\bigwedge_{u \neq u' \in U} (\mu(u) = \mu(u')) \Rightarrow \varphi(u, u') \vee \varphi(u', u)$$

Outline

Task Symmetry



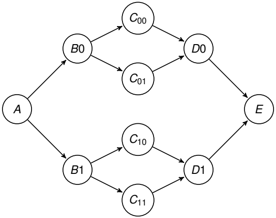
task graph



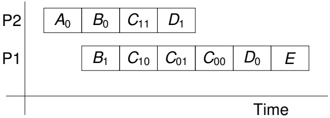
a schedule

- all instances of given actor v are similar (symmetric)

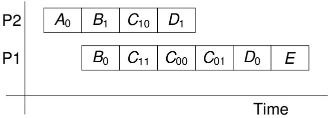
Task Symmetry



task graph



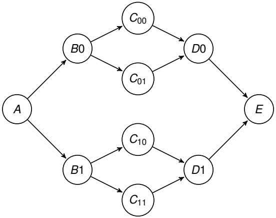
a schedule



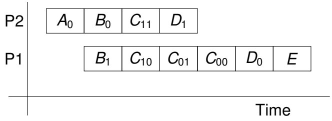
a permuted schedule

- all instances of given actor v are similar (symmetric)

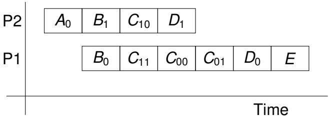
Task Symmetry



task graph



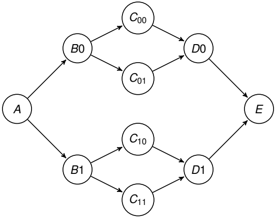
a schedule



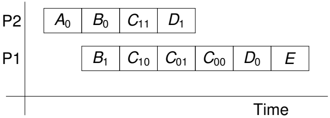
a permuted schedule

- all instances of given actor v are similar (symmetric)
- permutation of symmetric tasks does not change the latency,
- . . . but extends the solution space exponentially

Task Symmetry



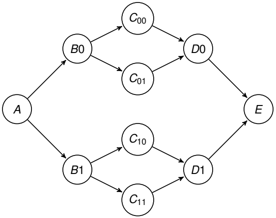
task graph



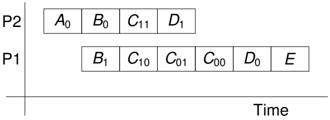
schedule

- enforce the schedule to be compatible with lexicographic order: $s(C_{00}) \leq s(C_{01}) \leq s(C_{10}) \leq s(C_{11})$

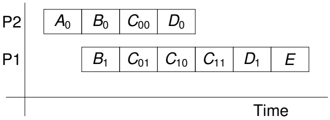
Task Symmetry



task graph



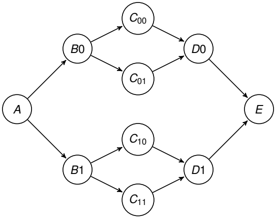
schedule



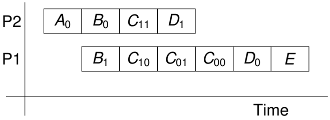
compatible schedule

- enforce the schedule to be compatible with lexicographic order: $s(C_{00}) \leq s(C_{01}) \leq s(C_{10}) \leq s(C_{11})$

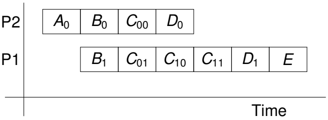
Task Symmetry



task graph



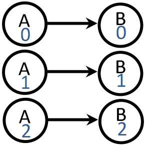
schedule



compatible schedule

- enforce the schedule to be compatible with lexicographic order: $s(C_{00}) \leq s(C_{01}) \leq s(C_{10}) \leq s(C_{11})$
- **Theorem:** adding constraints $s(u) \leq s(u')$ for $u \ll u'$ does not eliminate optimality

Proof Sketch

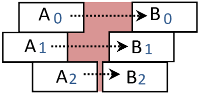


modify a feasible schedule such that:

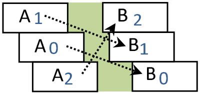
$s(v_0) \leq s(v_1) \leq s(v_2) \leq \ldots$

prove that precedence constraints are satisfied

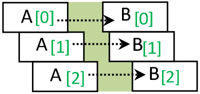
here: for neutral channels ($\alpha = 1$), unfolded to (v_h, v'_h)



lexicographic order

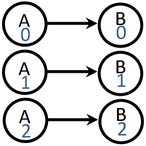


start-time compatible



new hier. index;
new precedence relation

Proof Sketch

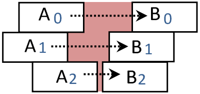


modify a feasible schedule such that:

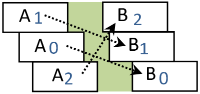
$s(v_0) \leq s(v_1) \leq s(v_2) \leq \ldots$

prove that precedence constraints are satisfied

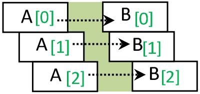
here: for neutral channels ($\alpha = 1$), unfolded to (v_h, v'_h)



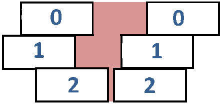
lexicographic order

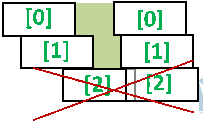


start-time compatible

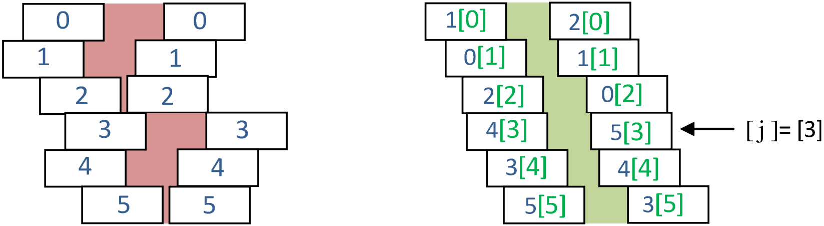


new hier. index; new precedence relation



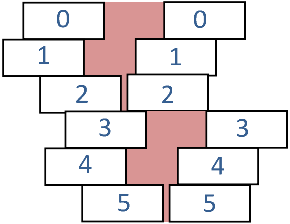


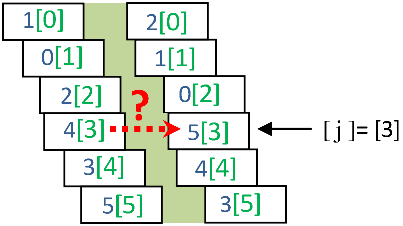
Proof Sketch



take successor [*j*]

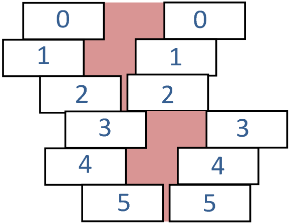
Proof Sketch

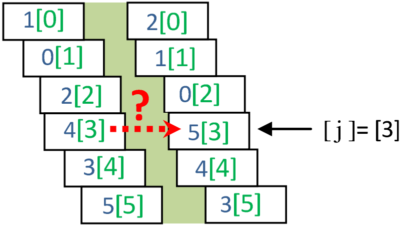




take successor [*j*]

Proof Sketch

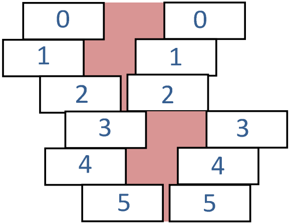


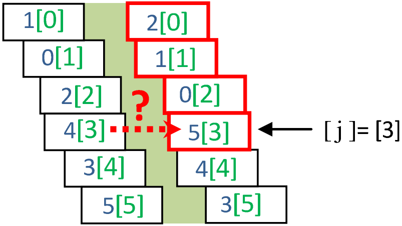


take successor [j]

by definition there exist $j + 1$ same or earlier successors

Proof Sketch

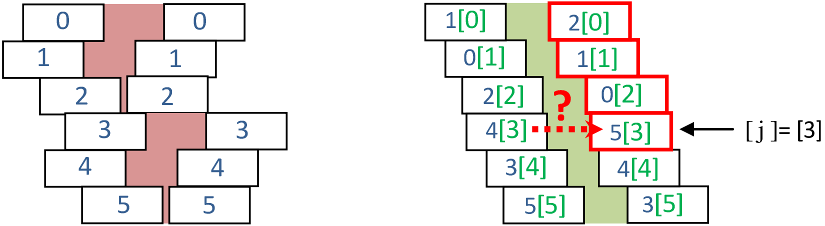




take successor [j]

by definition there exist $j + 1$ same or earlier successors

Proof Sketch

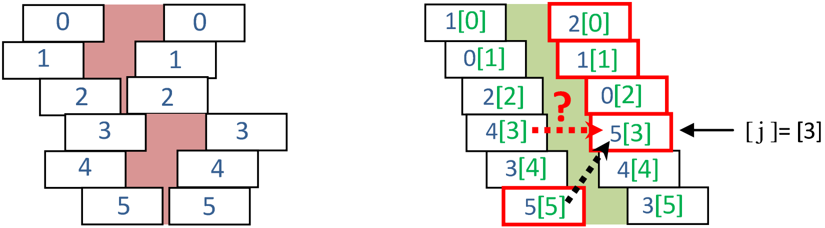


take successor [j]

by definition there exist $j + 1$ same or earlier successors

their original predecessors finish before successor [j]:

Proof Sketch

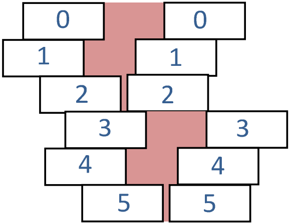


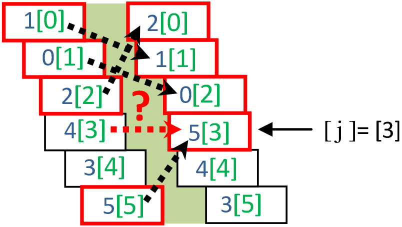
take successor [j]

by definition there exist $j+1$ same or earlier successors

their original predecessors finish before successor [j]:

Proof Sketch



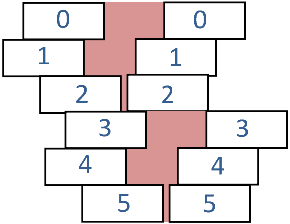


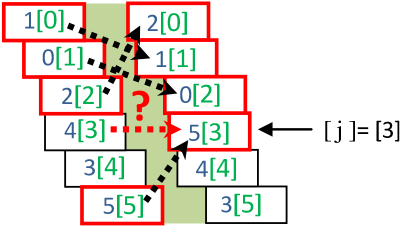
take successor [*j*]

by definition there exist $j + 1$ same or earlier successors

their original predecessors finish before successor [*j*]:

Proof Sketch





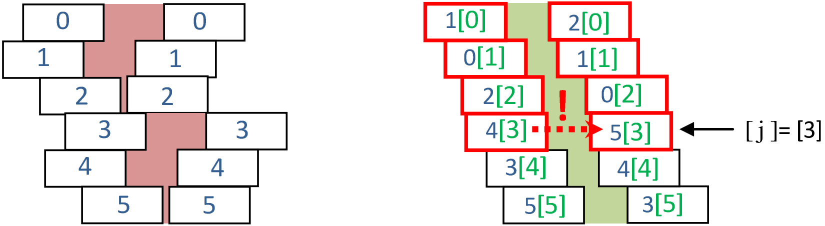
take successor [j]

by definition there exist $j+1$ same or earlier successors

their original predecessors finish before successor [j]:

$j+1$ predecessors finish before, hence the earliest $j+1$ ones as well

Proof Sketch



take successor [j]

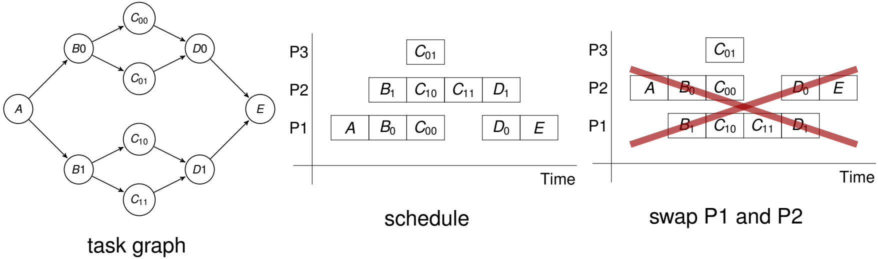
by definition there exist $j + 1$ same or earlier successors

their original predecessors finish before successor [j]:

$j + 1$ predecessors finish before, hence the earliest $j + 1$ ones as well

predecessor [j] finishes before successor [j]

Processor Symmetry

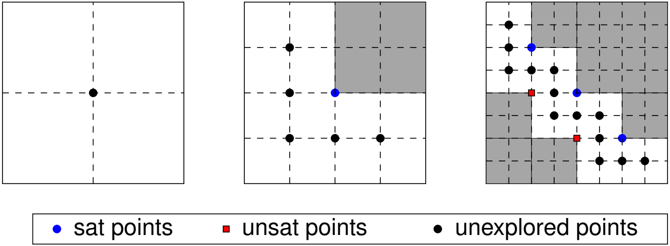


Outline

Exploring the Design Space

One SMT query for a given point (C_L, C_M) in the cost space:

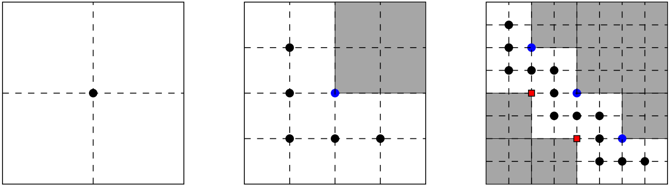
- C_L - latency
- C_M - processor count



Exploring the Design Space

One SMT query for a given point (C_L, C_M) in the cost space:

- C_L - latency
- C_M - processor count



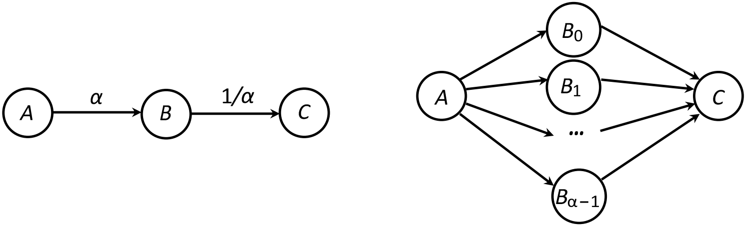
• sat points ▪ unsat points • unexplored points

- Precedence and Mutual Exclusion Constraints
- Cost Constraints

$$\bigwedge_{u \in U} s(u) + \delta(u) \leq C_L \ \wedge \ \bigwedge_{u \in U} \mu(u) \leq C_M$$

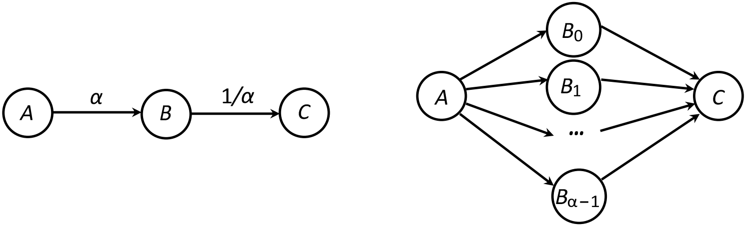
Outline

Synthetic-Graph Experiments



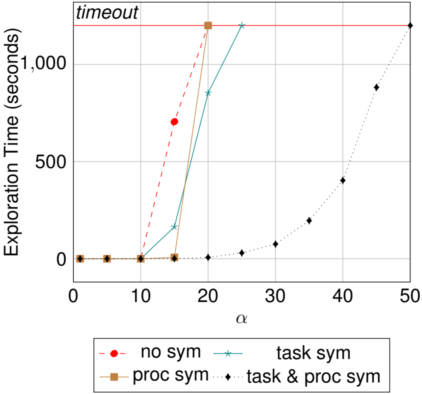
- Fix processor cost C_M and perform binary search for optimal C_L
- Increase α and measure increase in computation time
- With(out) breaking of task symmetry and processor symmetry

Synthetic-Graph Experiments



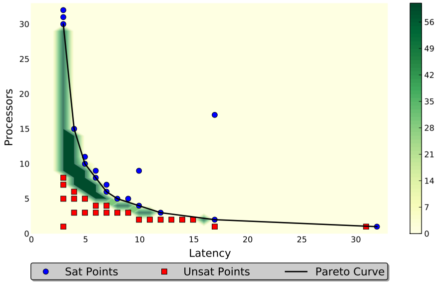
- Fix processor cost C_M and perform binary search for optimal C_L
- Increase α and measure increase in computation time
- With(out) breaking of task symmetry and processor symmetry
- Z3 solver v4.1 on i7 core at 1.73GHz

Synthetic-Graph Experiments



5-processor deployments

Pareto Exploration

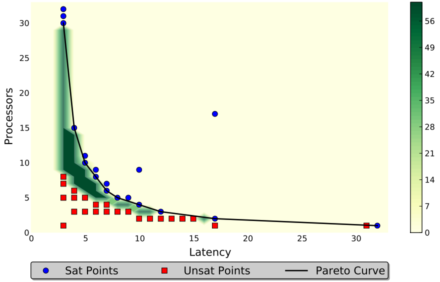


without symmetry breaking

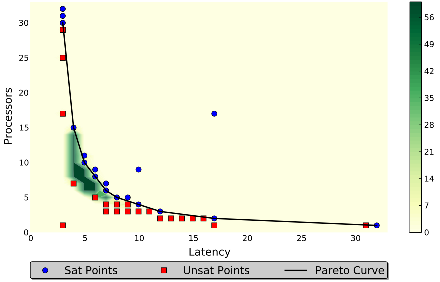
cost space (C_L, C_M) exploration for $\alpha = 30$

evaluate task and processor symmetry breaking

Pareto Exploration



without symmetry breaking



with symmetry breaking

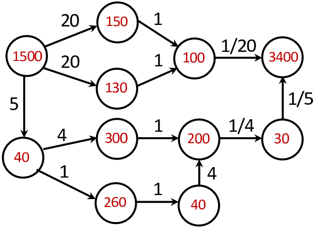
cost space (C_L, C_M) exploration for $\alpha = 30$

evaluate task and processor symmetry breaking

Video Decoder

3D cost space (C_L, C_M, C_B) exploration, C_B - total buffer size

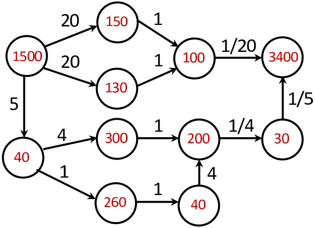
MPEG video decoder:

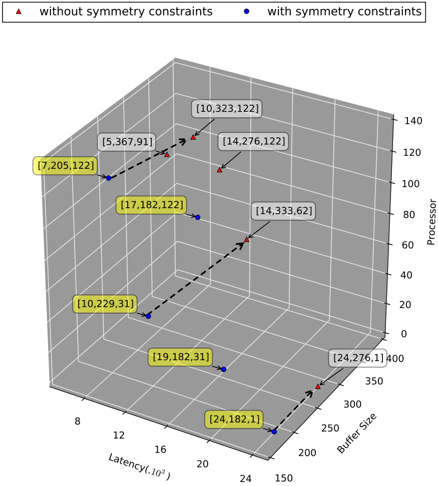


Video Decoder

3D cost space (C_L, C_M, C_B) exploration, C_B - total buffer size

MPEG video decoder:





Conclusions

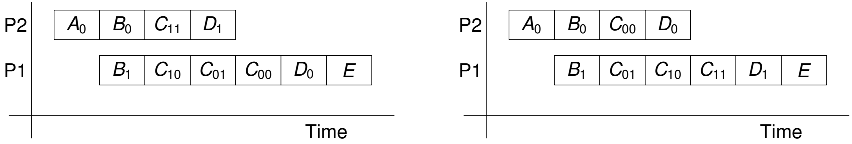
- Symbolic representation of data-parallel programs
  - a useful subclass of SDF model
- Framework for multi-criteria optimal deployment
- Symmetry breaking: prove task symmetry and use processor symmetry

Conclusions

- Symbolic representation of data-parallel programs
  - a useful subclass of SDF model
- Framework for multi-criteria optimal deployment
- Symmetry breaking: prove task symmetry and use processor symmetry
- Future work:

Conclusions

- Symbolic representation of data-parallel programs
  - a useful subclass of SDF model
- Framework for multi-criteria optimal deployment
- Symmetry breaking: prove task symmetry and use processor symmetry
- Future work:
- More symmetry breaking, also approximation and heuristics
- More refined data communication: data transfer delays
- Pipelined scheduling
- Scheduling under uncertainty
- Multistage design flow


P2
A_0 B_0 C_{11} D_1
P1
B_1 C_{10} C_{01} C_{00} D_0 E
Time
P2
A_0 B_0 C_{00} D_0
P1
B_1 C_{01} C_{10} C_{11} D_1 E
Time

QUESTIONS?