Tools and Resources for Task Scheduling Research

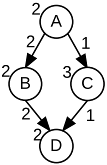
Oliver Sinnen

PARALLEL AND RECONFIGURABLE COMPUTING GROUP

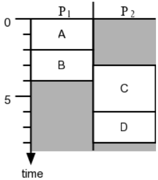
Department of Electrical and Computer Engineering
University of Auckland, New Zealand

Parallel computing scheduling

Scheduling task graphs with communication delays on homogeneous processors

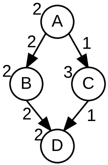


$\Rightarrow$

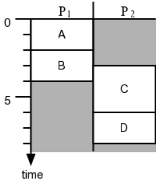


Parallel computing scheduling

Scheduling task graphs with communication delays on homogeneous processors



$\Longrightarrow$



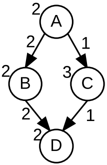
$P|prec, c_{ij}|C_{max}$

- Strong NP-hard

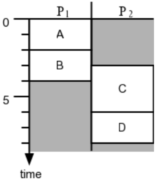
$\Rightarrow$ Heuristics, most popular is list scheduling

Parallel computing scheduling

Scheduling task graphs with communication delays on homogeneous processors



$\Longrightarrow$



$P|prec, c_{ij}|C_{max}$

- Strong NP-hard

$\Rightarrow$ Heuristics, most popular is list scheduling

Here: Finding optimal solutions nevertheless ☺

- For small to mid sized instances
- Important for time critical systems
- Evaluation of heuristics
- Increasing our knowledge about optimal scheduling solvers

Content

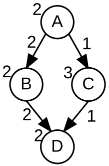
1. Scheduling problem
2. Visual scheduling tool
3. Optimal solvers
   - ILP solver
   - A* solver
4. Solution database

Content

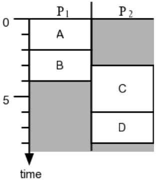
1. Scheduling problem
2. Visual scheduling tool
3. Optimal solvers
   - ILP solver
   - A* solver
4. Solution database

Scheduling problem

Finding start time and processor allocation for every task



$\Rightarrow$

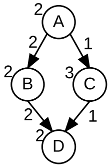


- t_i : start time of task i
- p_i : processor of task i

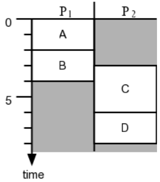
Given by task graph $G = (V, E)$

- L_i : execution time of task i
  - weight of node
- γ_{ij} : remote communication cost between tasks i and j
  - weight of edge

Constraints



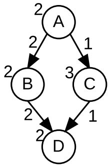
$\Rightarrow$



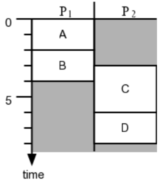
Processor constraint

$$p_i = p_j \Rightarrow \begin{cases} & t_i + L_i \leq t_j \\ \text{or} & t_j + L_j \leq t_i \end{cases}$$

Constraints



$$\Rightarrow$$



Processor constraint

$$p_i = p_j \Rightarrow \begin{cases} & t_i + L_i \leq t_j \\ \text{or} & t_j + L_j \leq t_i \end{cases}$$

Precedence constraint

For each edge e_{ij} of E

$$t_j \geq t_i + L_i + \begin{cases} 0 & \text{if } p_i = p_j \\ \gamma_{ij} & \text{otherwise} \end{cases}$$

Content

1 Scheduling problem

2 Visual scheduling tool

3 Optimal solvers
- ILP solver
- A* solver

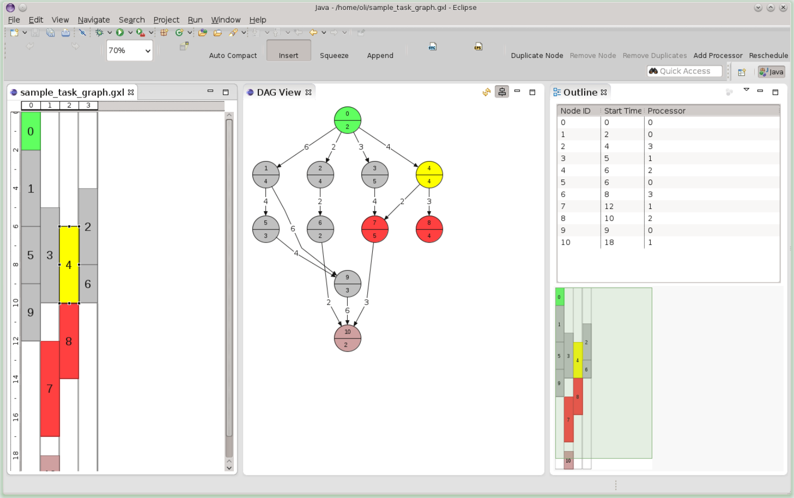
4 Solution database

Motivation

Why a visual scheduling tool?

- Understanding more about nature of schedules
- Finding patterns
- Manually experiment
- Behaviour for certain graph types

Task Scheduling Eclipse Plugin



http://www.ece.auckland.ac.nz/~parallel/plugins/TaskScheduleEclipsePlugin/

Plugin features

- Load/store task schedules (gxl format)

Plugin features

- Load/store task schedules (gxl format)
- Visualise schedules

Plugin features

- Load/store task schedules (gxl format)
- Visualise schedules
- Visualise task graphs/DAGs

Plugin features

- Load/store task schedules (gxl format)
- Visualise schedules
- Visualise task graphs/DAGs
- Visual relation between schedule and task graph, dependences

Plugin features

- Load/store task schedules (gxl format)
- Visualise schedules
- Visualise task graphs/DAGs
- Visual relation between schedule and task graph, dependences
- Built-in list scheduler, interface for external schedulers

Plugin features

- Load/store task schedules (gxl format)
- Visualise schedules
- Visualise task graphs/DAGs
- Visual relation between schedule and task graph, dependences
- Built-in list scheduler, interface for external schedulers
- Node duplication

Plugin features

- Load/store task schedules (gxl format)
- Visualise schedules
- Visualise task graphs/DAGs
- Visual relation between schedule and task graph, dependences
- Built-in list scheduler, interface for external schedulers
- Node duplication
- One-port model (currently no manipulation)

Plugin features

- Load/store task schedules (gxl format)
- Visualise schedules
- Visualise task graphs/DAGs
- Visual relation between schedule and task graph, dependences
- Built-in list scheduler, interface for external schedulers
- Node duplication
- One-port model (currently no manipulation)
- Manual schedule support, insertion, append, squeeze

Plugin features

- Load/store task schedules (gxl format)
- Visualise schedules
- Visualise task graphs/DAGs
- Visual relation between schedule and task graph, dependences
- Built-in list scheduler, interface for external schedulers
- Node duplication
- One-port model (currently no manipulation)
- Manual schedule support, insertion, append, squeeze
- Export of schedules to svg and eps

Content

1 Scheduling problem

2 Visual scheduling tool

3 Optimal solvers
- ILP solver
- A* solver

4 Solution database

Content

1 Scheduling problem

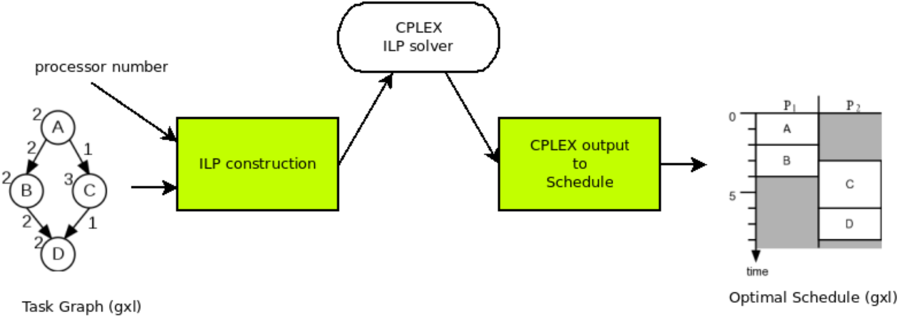
2 Visual scheduling tool

3 Optimal solvers
- ILP solver
- A* solver

4 Solution database

Green Banana

- MILP solver for $P|prec, c_{ij}|C_{max}$



- At http://www.ece.auckland.ac.nz/~parallel/OptimalTaskScheduling/

Principle ILP variant

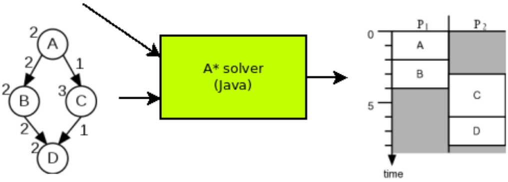
min	W	MinMax
$\forall i \in V$	$t_i + L_i \leq W$	
$\forall i \neq j \in V$	$\sigma_{ij} + \sigma_{ji} + \epsilon_{ij} + \epsilon_{ji} \geq 1$	Overlap
$\forall i \neq j \in V$	$\sigma_{ij} + \sigma_{ji} \leq 1$	
$\forall i \neq j \in V$	$\epsilon_{ij} + \epsilon_{ji} \leq 1$	
$\forall j \in V \ : \ i \in \delta^{-}(j)$	$\sigma_{ij} = 1$	Edge
$\forall j \in V \ : \ i \in \delta^{-}(j)$	$p_j - p_i \leq \epsilon_{ij}\|P\|$	Processor
$\forall j \in V \ : \ i \in \delta^{-}(j)$	$p_i - p_j \leq \epsilon_{ji}\|P\|$	
$\forall i \neq j \in V$	$p_j - p_i - 1 - (\epsilon_{ij} - 1)\|P\| \geq 0$	
$\forall i \neq j \in V$	$t_j - t_i - L_i - (\sigma_{ij} - 1)W_{max} \geq 0$	Precedence
$\forall j \in V \ : \ i \in \delta^{-}(j)$	$t_i + L_i + \gamma_{ij}(\epsilon_{ij} + \epsilon_{ji}) \leq t_j$	

Content

1. Scheduling problem
2. Visual scheduling tool
3. Optimal solvers
   - ILP solver
   - A* solver
4. Solution database

A* solver

- A* solver for $P|prec, c_{ij}|C_{max}$



Task Graph (gxl)

Optimal Schedule (gxl)

- Soon at
  http://www.ece.auckland.ac.nz/~parallel/OptimalTaskScheduling/

A*

- Exhaustive search through all possible solutions

A*

- Exhaustive search through all possible solutions
- State space
  - Every state (node) s represents partial solution
  - Combinatorial problems $\Rightarrow$ search tree
  - Deeper nodes are more complete solutions

A*

- Exhaustive search through all possible solutions
- State space
  - Every state (node) s represents partial solution
  - Combinatorial problems $\Rightarrow$ search tree
  - Deeper nodes are more complete solutions
- Best first search
  - Next node to consider has best cost function $f(s)$
  - Cost $f(s)$ must be underestimate to find optimal solution

A*

- Exhaustive search through all possible solutions
- State space
  - Every state (node) s represents partial solution
  - Combinatorial problems ⇒ search tree
  - Deeper nodes are more complete solutions
- Best first search
  - Next node to consider has best cost function $f(s)$
  - Cost $f(s)$ must be underestimate to find optimal solution
- Property: with same given cost estimate function, A* explores least number of states

Task scheduling with A*

Essentially: list scheduling, trying out all task orders and all processor allocations

- State: partial schedule
- Initial state: empty schedule
- Cost function $f(s)$: underestimate of makespan for complete schedule based on s

Task scheduling with A*

Essentially: list scheduling, trying out all task orders and all processor allocations

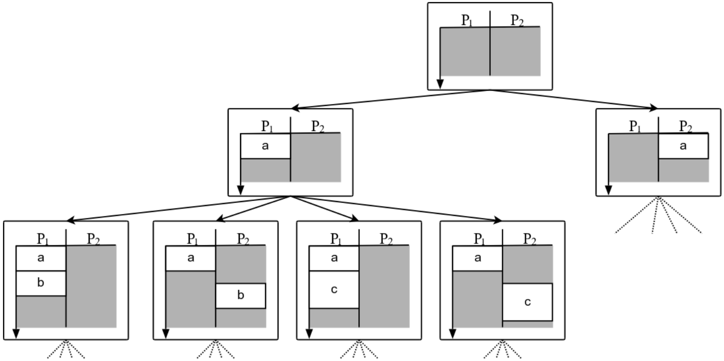
- State: partial schedule
- Initial state: empty schedule
- Cost function $f(s)$: underestimate of makespan for complete schedule based on s

Expansion

- Given state s, let **free**(s) be free tasks

for all $n \in$ **free**(s) **do**
  for all $P \in \mathbf{P}$ **do**
    Create new state: n scheduled on P as early as possible

State tree example



Cost function $f(s)$

Several components, two examples

- Perfect load balance plus current idle time

$$f_{idle-time}(s) = \frac{\sum_{i \in \mathbf{V}} L_i + idle(s)}{|\mathbf{P}|}$$

Cost function $f(s)$

Several components, two examples

- Perfect load balance plus current idle time

$$f_{idle-time}(s) = \frac{\sum_{i \in \mathbf{V}} L_i + idle(s)}{|\mathbf{P}|}$$

- Max (start time of scheduled tasks plus their bottom level)

$$f_{bl}(s) = \max_{i \in s}\{t_i + bl_w(i)\}$$

Cost function $f(s)$

Several components, two examples

- Perfect load balance plus current idle time

$$f_{idle-time}(s) = \frac{\sum_{i \in \mathbf{V}} L_i + idle(s)}{|\mathbf{P}|}$$

- Max (start time of scheduled tasks plus their bottom level)

$$f_{bl}(s) = \max_{i \in s}\{t_i + bl_w(i)\}$$

Complete $f(s)$ function:

$$f(s) = \max\{f_{idle-time}(s), f_{bl}(s), \dots\}$$

Pruning

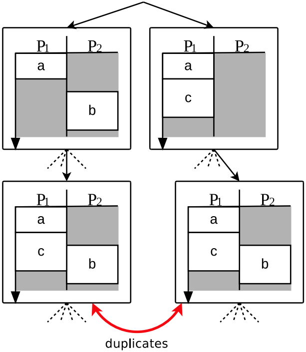
- Pruning is crucial
  - Even with good cost functions $f(s)$

Basic techniques

- Processor normalisation
- Node equivalence
- Duplicate elimination

- Pruning is crucial
  - Even with good cost functions $f(s)$

Duplication elimination

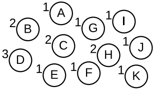


Basic techniques

- Processor normalisation
- Node equivalence
- Duplicate elimination

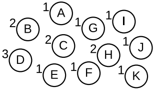
Task order observations

- For certain graph structures, task order does not matter
  - Independent tasks



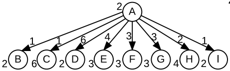
Task order observations

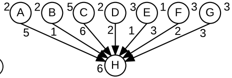
- For certain graph structures, task order does not matter
  - Independent tasks

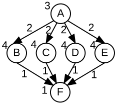


- For other structures, optimal order can be computed
  - Fork: order tasks by non-decreasing incoming communication
  - Join: order tasks by non-increasing outgoing communication

  $\Rightarrow$ leads to optimal orders
  - Fork-join: there is no easy optimal order
    - But, in some cases tasks can be fork-order and join-order at once $\Rightarrow$ optimal







Fixed task order

Generalisation for pruning

- Fix order of free tasks
  - Use fork-order, join-order
- For certain sub-structures
  - independent, fork, join, fork-join

$\Rightarrow$ Only consider one task for expansion

- Reduces branching factor to $|P|$ instead $\mathit{free}(s) \cdot |P|$

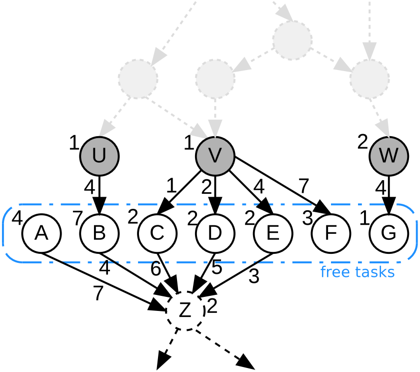
Fixed task order

Generalisation for pruning

- Fix order of free tasks
  - Use fork-order, join-order
- For certain sub-structures
  - independent, fork, join, fork-join

$\Rightarrow$ Only consider one task for expansion

  - Reduces branching factor to $|P|$ instead $\mathit{free}(s) \cdot |P|$



Content

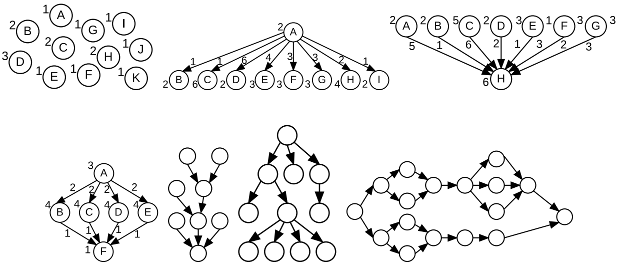
1. Scheduling problem
2. Visual scheduling tool
3. Optimal solvers
   - ILP solver
   - A* solver
4. Solution database

Optimal scheduling solutions database

- Large set of task graphs
- Optimal schedules of these graphs on different number of processors

Task Graphs

- Graph structures



- Also: **pipeline, stencil and random**
- Sizes: up to 30 tasks (currently)
- Communication-to-computation ratio (CCR): 0.1, 1.0, 2.0, 10.0.

Optimal Schedules

Graphs scheduled on different number of processors 2-16 processors

Information stored:

- Number of processors
- Detailed schedule (in gxl format)
- Schedule length
- How obtained

Optimal Schedules

Graphs scheduled on different number of processors 2-16 processors
Information stored:

- Number of processors
- Detailed schedule (in gxl format)
- Schedule length
- How obtained

Currently some hundred schedules at
http://www.ece.auckland.ac.nz/~parallel/OptimalTaskScheduling/

Summary

Tools and resources for $P|prec, c_{ij}|C_{max}$

- Visual scheduling tool
- Optimal solvers
  - Green banana (ILP solver)
  - A* scheduling
- Solution database of optimal schedules

http://www.ece.auckland.ac.nz/~parallel/OptimalTaskScheduling/