Outline

Introduction
Introduction

Co-Scheduling

Outline
Outline

Introduction

Co-Scheduling

2-Cores
Outline

Introduction

Co-Scheduling

2-Cores

3+ Cores

Reduction from Sat
Outline

Introduction

Co-Scheduling

2-Cores

3+ Cores

Reduction from Sat

What next?
The paper

The Complexity of Optimal Job Co-Scheduling on Chip Multiprocessors and Heuristics-Based Solutions
by Yunlian Jiang, Kai Tian, Xipeng Shen, Jinghe Zhang, Jie Chen, Rahul Tripathi
The Problem

1. Estimation
2. Optimize
The Problem

1. Estimation
2. Optimize

We only care about the second
Simplification

- $n$ jobs. All same priority etc
Introduction
Co-Scheduling
2-Cores
3+ Cores
What next?

Simplification

- $n$ jobs. All same priority etc
- $u$ identical cores with shared resources
Simplification

- \( n \) jobs. All same priority etc
- \( u \) identical cores with shared resources
- leaves \( n/u \) co-slots
Simplification

- $n$ jobs. All same priority etc
Simplification

- $n$ jobs. All same priority etc
- $u$ identical cores with shared resources
Simplification

- $n$ jobs. All same priority etc
- $u$ identical cores with shared resources
- Leaves $n/u$ co-slots
Definitions and Goals

- CPI—Cycles per instruction
Definitions and Goals

- CPI—Cycles per instruction
- Cu run Degradation: slow down of one job $i$ when co run with set $S$

$$d_{i,S} = \frac{(CPI_{i,S} - CPI_{i,\emptyset})}{CPI_{i,\emptyset}}$$
Definitions and Goals

- **CPI**—Cycles per instruction
- Cu run Degradation: slow down of one job $i$ when co run with set $S$
  \[
  d_{i,S} = (CPI_{i,S} - CPI_{i,\emptyset}) / CPI_{i,\emptyset}
  \]
- We want to minimize total degradation
  \[
  \sum_{i=1}^{n} d_{i,S}
  \]
Perfect Matching

A perfect matching on a graph \((V, E)\) is a subset of \(M\) of \(E\) such that for each \(v \in V\) there exists a unique \(e \in M\) such that \(m = (x, v)\) or \(m = (v, x)\) for some \(x\).
A perfect matching on a graph \((V, E)\) is a subset of \(M\) of \(E\) such that for each \(v \in V\) there exists a unique \(e \in M\) such that \(m = (x, v)\) or \(m = (v, x)\) for some \(x\).

Visualize this by coloring edges
A perfect matching on a graph \((V, E)\) is a subset of \(M\) of \(E\) such that for each \(v \in V\) there exists a unique \(e \in M\) such that \(m = (x, v)\) or \(m = (v, x)\) for some \(x\).

Visualize this by coloring edges

A maximal weight matching is a perfect matching on a graph with maximal weight.
2-Cores as Maximal Matching

*Theorem* when $u = 2$, the degradation information can be regarded as a completely connected graph. A co-schedule in this framework is a perfect matching.
2-Cores as Maximal Matching

*Theorem* when $u = 2$, the degradation information can be regarded as a completely connected graph. A co-schedule in this framework is a perfect matching further, an optimal co-schedule is a maximal weight matching.
Fast Solution

Edmonds blossom algorithm is polynomial time
Fast Solution

Edmonds blossom algorithm is polynomial time
I’m not going to talk about it. Take the “Advanced Data Structures” course or talk to me later
One complaint I have about the paper is that it confuses optimization and decision problems.
One complaint I have about the paper is that it confuses optimization and decision problems.

Co-scheduling is an optimization problem
One complaint I have about the paper is that it confuses optimization and decision problems.

Co-scheduling is an optimization problem

Decision problems are easier to think about and analyze (NP completeness?)
One complaint I have about the paper is that it confuses optimization and decision problems.

- Co-scheduling is an optimization problem
- Decision problems are easier to think about and analyze (NP completeness?)
- We consider a problem that is at least as hard as three way scheduling
3 Way Scheduling

- 3 dimensional maximal weight matching is just maximal matching on graphs where the edges each connect 3 vertices.
3 Way Scheduling

- 3 dimensional maximal weight matching is just maximal matching on graphs where the edges each connect 3 vertices
- by same argument as above, 3 way scheduling is 3 dimensional maximal weight matching
3 Way Scheduling

- 3 dimensional maximal weight matching is just maximal matching on graphs where the edges each connect 3 vertices
- by same argument as above, 3 way scheduling is 3 dimensional maximal weight matching
- at least as hard as “does some 3-graph have a matching?” (why?)
3 Way Scheduling

- 3 dimensional maximal weight matching is just maximal matching on graphs where the edges each connect 3 vertices
- by same argument as above, 3 way scheduling is 3 dimensional maximal weight matching
- at least as hard as “does some 3-graph have a matching?” (why?)
Three sets $W, X, Y$ all the same size
3DM

- Three sets $W$, $X$, $Y$ all the same size
- and $E$ is a subset of $W \times X \times Y$
Introduction
Co-Scheduling
2-Cores
3+ Cores
What next?

Reduction from Sat

3DM

- Three sets $W, X, Y$ all the same size
- and $E$ is a subset of $W \times X \times Y$
- Question: does there exist a subset of $E$ such that each element appears exactly once
SAT

- Variables: $U = \{u_1, u_2 \ldots\}$
- Clauses: $C = \{c_1, c_2 \ldots\}$
Truth Setting

- We define $2 \times |U| \times |C|$ new triples for the variables (one copy of each variable and its conjugate for each clause)
Introduction
Co-Scheduling
2-Cores
3+ Cores
What next?

Reduction from Sat

Truth Setting

- We define $2 \times |U| \times |C|$ new triples for the variables (one copy of each variable and its conjugate for each clause)
- We use two hidden variables $a, b$ and one public one $u$

$$T_i = \bigcup_{1 \leq j \leq m} \{(\bar{u}_i[j], a_i[j], b_i[j]) \cup \{u_i[j], a_i[j + 1], b_i[j]\}\} \cup \{(u_i[m], a_i[1], b_i[m])\}$$
Truth Setting

- We define $2 \times |U| \times |C|$ new triples for the variables (one copy of each variable and its conjugate for each clause)
- We use two hidden variables $a, b$ and one public one $u$

$$T_i = \bigcup_{1 \leq j \leq m} \{ (\bar{u}_i[j], a_i[j], b_i[j]) \} \cup \{ (u_i[j], a_i[j + 1], b_i[j]) \} \cup \{ (u_i[m], a_i[1], b_i[m]) \}$$

- Either all the $\bar{u}_i$ or all the $u_i$ are covered
Satisfaction Testing

▶ Each clause requires two hidden variables $s_1[j], s_2[j]$
Satisfaction Testing

- Each clause requires two hidden variables $s_1[j], s_2[j]$
- These will be matched with exactly one of the variables

$$C_j = \{(u_i[j], s_1[j], s_2[j]) : u_i \in c_j\} \cup \{(ar{u}_i[j], s_1[j], s_2[j]) : \bar{u}_i \in c_j\}$$
Garbage collection

Add more elements with complete connections to all the \( u \) and \( \bar{u} \) until the three sets have the same number of elements.
Formulation of the problem as Integer Programming
Formulation of the problem as Integer Programming
Of the shell IP solvers
pause This also gives an efficient lower bound technique
Formulation of the problem as Integer Programming
Of the shell IP solvers
pause This also gives an efficient lower bound technique
...forget the integer constraint and do LP
Heuristics

IMO, much better option.
Much of the presentation came from “Computers and Intractability” by Garey and Johnson
I also have read Tarjan’s book on graph algorithms (for Andrzej’s class) which I recommend if you want to understand matching