Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Systems
H. Topcuoglu, S. Hariri, M. Wu

Jait Dixit

21.05.2014
Outline

1 Task Scheduling
   - Classic Model
   - Theoretical Background
   - Heterogeneity
   - Algorithms

2 HEFT & CPOP
   - Heterogeneous Earliest Finish Time (HEFT)
   - Critical-Path-on-Processor (CPOP)
   - Experiments

3 Conclusion
Basics

- Static task scheduling.
Basics

- **Static task scheduling.**
- Everything is known *a priori.*
Basics

- **Static task scheduling**.
- Everything is known *a priori*.
- Problem:
Basics

- **Static task scheduling.**
- Everything is known *a priori.*
- Problem:
  - **Input:** number of tasks and a set of processors
Basics

- **Static task scheduling.**
- Everything is known *a priori.*
- Problem:
  - **Input:** number of tasks and a set of processors
  - **Output:** schedule with minimal overall completion time
Tasks

- DAG
Tasks

- DAG
- $G = (V, E)$
Tasks

- DAG
- \( G = (V, E, w) \)
Tasks

- DAG
- $G = (V, E, w, c)$
Tasks

- DAG
  
  \[ G = (V, E, w, c) \]
  
- Edges show precedence relation
Tasks

- DAG
- $G = (V, E, w, c)$
- Edges show precedence relation
- Entry and exit task
Processors

- Set of processors
Processors

- Set of processors
- Homogeneous
Processors

- Set of processors
- Homogeneous
- Non-preemptive
Processors

- Set of processors
- Homogeneous
- Non-preemptive
- Cost-free local communication
Processors

- Set of processors
- Homogeneous
- Non-preemptive
- Cost-free local communication
- Communication subsystem
Processors

- Set of processors
- Homogeneous
- Non-preemptive
- Cost-free local communication
- Communication subsystem
- Concurrent communication
Processors

- Set of processors
- Homogeneous
- Non-preemptive
- Cost-free local communication
- Communication subsystem
- Concurrent communication
- Fully connected
Processors

- Set of processors
- Homogeneous
- Non-preemptive
- Cost-free local communication
- Communication subsystem
- Concurrent communication
- Fully connected
- Parallel system, $P$
A schedule $S$ for task graph $G = (V, E, w, c)$ on a finite set $P$ of processors:

- Allocation of tasks in $G$ to a processor in $P$.
- Defining a start time for the node on the respective processor.

A schedule is feasible only if:

- Precedence constraints in $G$ are satisfied.
- Non-preemption is enforced.

Feasibility of schedule can be verified in polynomial time.
Schedule

- A schedule $S$ for task graph $G = (V, E, w, c)$ on a finite set $P$ of processors:
  - allocation of tasks in $G$ to a processor in $P$

Feasibility of schedule can be verified in polynomial time:

- Last finishing time of the given jobs $J$
Schedule

- A schedule $\mathcal{S}$ for task graph $G = (V, E, w, c)$ on a finite set $P$ of processors:
  - allocation of tasks in $G$ to a processor in $P$
  - defining a start time for the node on the respective processor

Schedule is feasible only if:
- precedence constraints in $G$ are satisfied
- non-preemption is enforced

Feasibility of schedule can be verified in polynomial time

$$\text{makespan} = \text{sl}(\mathcal{S})$$

- Last finishing time of the given jobs

Jait Dixit
Better Task Scheduling
21.05.2014 6 / 38
Schedule

- A schedule $S$ for task graph $G = (V, E, w, c)$ on a finite set $P$ of processors:
  - allocation of tasks in $G$ to a processor in $P$
  - defining a start time for the node on the respective processor

- Schedule is feasible only if:
  - precedence constraints in $G$ are satisfied
  - non-preemption is enforced
  - Feasibility of schedule can be verified in polynomial time
A schedule $S$ for task graph $G = (V, E, w, c)$ on a finite set $P$ of processors:

- allocation of tasks in $G$ to a processor in $P$
- defining a start time for the node on the respective processor

Schedule is feasible only if:

- precedence constraints in $G$ are satisfied

Feasibility of schedule can be verified in polynomial time.
Schedule

- A schedule $S$ for task graph $G = (V, E, w, c)$ on a finite set $P$ of processors:
  - allocation of tasks in $G$ to a processor in $P$
  - defining a start time for the node on the respective processor

- Schedule is feasible only if:
  - precedence constraints in $G$ are satisfied
  - non-preemption is enforced

Feasibility of schedule can be verified in polynomial time

makespan $= \text{sl}(S)$

- Last finishing time of the given jobs
Schedule

- A schedule $S$ for task graph $G = (V, E, w, c)$ on a finite set $P$ of processors:
  - allocation of tasks in $G$ to a processor in $P$
  - defining a start time for the node on the respective processor

- Schedule is feasible only if:
  - precedence constraints in $G$ are satisfied
  - non-preemption is enforced

- Feasibility of schedule can be verified in polynomial time
Schedule

- A schedule $S$ for task graph $G = (V, E, w, c)$ on a finite set $P$ of processors:
  - allocation of tasks in $G$ to a processor in $P$
  - defining a start time for the node on the respective processor

- Schedule is feasible only if:
  - precedence constraints in $G$ are satisfied
  - non-preemption is enforced

- Feasibility of schedule can be verified in polynomial time

- $\text{makespan} = sl(S)$
Schedule

A schedule $S$ for task graph $G = (V, E, w, c)$ on a finite set $P$ of processors:

- allocation of tasks in $G$ to a processor in $P$
- defining a start time for the node on the respective processor

Schedule is feasible only if:

- precedence constraints in $G$ are satisfied
- non-preemption is enforced

Feasibility of schedule can be verified in polynomial time

**makespan** $\equiv sl(S)$

- Last finishing time of the given jobs
NP-completeness

- $G = (V, E, \omega, c)$
NP-completeness

- $G = (V, E, w, c)$
- $P$, a parallel system
NP-completeness

- $G = (V, E, w, c)$
- $P$, a parallel system
- $SCHED(G, P)$ is the associated decision problem
NP-completeness

- $G = (V, E, w, c)$
- $P$, a parallel system
- $\text{SCHED}(G, P)$ is the associated decision problem
  ▶ Is there a schedule $S$ for $G$ on $P$ with length $sl(S) \leq T$?
NP-completeness

- \( G = (V, E, w, c) \)
- \( P \), a parallel system
- \( \text{SCHED}(G, P) \) is the associated decision problem
  - Is there a schedule \( S \) for \( G \) on \( P \) with length \( sl(S) \leq T \)?
- \( \text{SCHED}(G, P) \) is **strongly NP-hard**
Proof

1. It is argued that SCHED belongs to NP.
Proof

1. It is argued that SCHED belongs to NP

2. 3-PARTITION is NP-complete in the strong sense
Proof

1. It is argued that SCHED belongs to NP
2. 3-PARTITION is NP-complete in the strong sense
3. By reducing 3-PARTITION in polynomial time to SCHED, it’s shown that SCHED is strongly NP-hard
SCHED \in NP

- For any $S$ from $\text{SCHED}(G, P)$
SCHED $\in$ NP

- For any $S$ from $\text{SCHED}(G, P)$
- It can be verified in polynomial time whether $S$ is feasible
SCHED $\in$ NP

- For any $S$ from $\text{SCHED}(G, P)$
- It can be verified in polynomial time whether $S$ is feasible
- and $sl(S) \leq T$
**SCHED \in NP**

- For any \( S \) from \( \text{SCHED}(G, P) \)
- It can be verified in polynomial time whether \( S \) is feasible
- and \( sl(S) \leq T \)
- Hence, \( \text{SCHED}(G, P) \in NP \)
3-PARTITION

3-PARTITION:

- A set $A$ of $3m$ positive integers $a_i$
- A positive integer bound $B$ such that $\sum_{i=1}^{3m} a_i = mB$
- $4 < a_i < 2B$

Can $A$ be partitioned into $m$ disjoint sets $A_1, \ldots, A_m$ such that each $A_i$ is a triplet whose sum is $B$?

Strongly NP-complete

Proved by Garey & Johnson, 1975
3-PARTITION

3-PARTITION:
- a set $A$ of $3m$ positive integers $a_i$
3-PARTITION

3-PARTITION:

- a set $A$ of $3m$ positive integers $a_i$
- a positive integer bound $B$ s.t. $\sum_{i=1}^{3m} a_i = mB$

Strongly NP-complete

Proved by Garey & Johnson, 1975
3-PARTITION

3-PARTITION:

- a set $A$ of $3m$ positive integers $a_i$
- a positive integer bound $B$ s.t. $\sum_{i=1}^{3m} a_i = mB$
- with $\frac{B}{4} < a_i < \frac{B}{2}$

Strongly NP-complete
Proved by Garey & Johnson, 1975
3-PARTITION

3-PARTITION:

- A set $A$ of $3m$ positive integers $a_i$
- A positive integer bound $B$ s.t. $\sum_{i=1}^{3m} a_i = mB$
- With $\frac{B}{4} < a_i < \frac{B}{2}$
- Can $A$ be partitioned into $m$ disjoint sets $A_1, \ldots, A_m$
3-PARTITION

3-PARTITION:

- a set $A$ of $3m$ positive integers $a_i$
- a positive integer bound $B$ s.t. $\sum_{i=1}^{3m} a_i = mB$
- with $\frac{B}{2} < a_i < \frac{B}{4}$
- Can $A$ be partitioned into $m$ disjoint sets $A_1, \ldots, A_m$
- s.t. each $A_i$ is a triplet whose sum is $B$?
3-PARTITION

3-PARTITION:

- a set $A$ of $3m$ positive integers $a_i$
- a positive integer bound $B$ s.t. $\sum_{i=1}^{3m} a_i = mB$
- with $\frac{B}{4} < a_i < \frac{B}{2}$
- Can $A$ be partitioned into $m$ disjoint sets $A_1, \ldots, A_m$
- s.t. each $A_i$ is a triplet whose sum is $B$?

Strongly NP-complete
3-PARTITION

3-PARTITION:

- a set $A$ of $3m$ positive integers $a_i$
- a positive integer bound $B$ s.t. $\sum_{i=1}^{3m} a_i = mB$
- with $\frac{B}{4} < a_i < \frac{B}{2}$
- Can $A$ be partitioned into $m$ disjoint sets $A_1, \ldots, A_m$
- s.t. each $A_i$ is a triplet whose sum is $B$?

- Strongly NP-complete
- Proved by Garey & Johnson, 1975
Construction

Constructing SCHED from arbitrary instance of 3-PARTITION

$|V| = 3m + 1$ nodes, $|P| = m$ and $T = B + 1$.5
Construction

Constructing SCHED from arbitrary instance of 3-PARTITION
Constructing SCHED from arbitrary instance of 3-PARTITION

- $|V| = 3m + 1$ nodes, $|P| = m$ and $T = B + 1.5$
Proving Reduction

Input ∈ 3-PARTITION → Input ∈ Construction
Proving Reduction

- Input $\in \text{3-PARTITION} \rightarrow$ Input $\in \text{Construction}$
  - $A$, an arbitrary instance of 3-PARTITION which admits a solution
Proving Reduction

- Input $\in$ 3-PARTITION $\rightarrow$ Input $\in$ Construction
  - A, an arbitrary instance of 3-PARTITION which admits a solution
  - $n_0$ is allocated to $P_1$
Proving Reduction

- Input $\in$ 3-PARTITION $\rightarrow$ Input $\in$ Construction
  - $A$, an arbitrary instance of 3-PARTITION which admits a solution
  - $n_0$ is allocated to $P_1$
  - Remaining triplets are allocated to $P_1, \ldots, P_m$
Proving Reduction

- Input $\in$ 3-PARTITION $\rightarrow$ Input $\in$ Construction
  - $A$, an arbitrary instance of 3-PARTITION which admits a solution
  - $n_0$ is allocated to $P_1$
  - Remaining triplets are allocated to $P_1, \ldots, P_m$
  - $sl(S)$?
Proving Reduction

- **Input ∈ 3-PARTITION → Input ∈ Construction**
  - A, an arbitrary instance of 3-PARTITION which admits a solution
  - $n_0$ is allocated to $P_1$
  - Remaining triplets are allocated to $P_1, \ldots, P_m$
  - $sl(S) = B + 1.5 \leq T$. 
Proving Reduction

- Input ∈ 3-PARTITION → Input ∈ Construction
  - A, an arbitrary instance of 3-PARTITION which admits a solution
  - \( n_0 \) is allocated to \( P_1 \)
  - Remaining triplets are allocated to \( P_1, \ldots, P_m \)
  - \( sl(S) = B + 1.5 \leq T \).

- Input ∈ Construction → Input ∈ 3-PARTITION
Proving Reduction

- **Input ∈ 3-PARTITION → Input ∈ Construction**
  - A, an arbitrary instance of 3-PARTITION which admits a solution
  - $n_0$ is allocated to $P_1$
  - Remaining triplets are allocated to $P_1, \ldots, P_m$
  - $sl(S) = B + 1.5 \leq T$.

- **Input ∈ Construction → Input ∈ 3-PARTITION**
  - An instance of SCHED which admits a solution
Proving Reduction

- Input $\in$ 3-PARTITION $\rightarrow$ Input $\in$ Construction
  - $A$, an arbitrary instance of 3-PARTITION which admits a solution
  - $n_0$ is allocated to $P_1$
  - Remaining triplets are allocated to $P_1, \ldots, P_m$
  - $sl(S) = B + 1.5 \leq T$.

- Input $\in$ Construction $\rightarrow$ Input $\in$ 3-PARTITION
  - An instance of SCHED which admits a solution
  - Each processor can spend at most $B$ time units
Proving Reduction

- **Input \( \in \text{3-PARTITION} \) → Input \( \in \text{Construction} \)
  - A, an arbitrary instance of 3-PARTITION which admits a solution
  - \( n_0 \) is allocated to \( P_1 \)
  - Remaining triplets are allocated to \( P_1, \ldots, P_m \)
  - \( sl(S) = B + 1.5 \leq T \).

- **Input \( \in \text{Construction} \) → Input \( \in \text{3-PARTITION} \)
  - An instance of SCHED which admits a solution
  - Each processor can spend at most B time units
  - \( \sum_{i=1}^{3m} w(n_i) = mB \) and \( |P| = m \)

Due to \( w(n_i) = a_i, B < a_i < B \), only 3 nodes can have the exact execution time of B

3-PARTITION reduces to SCHED \( \Rightarrow \) SCHED is strongly NP-hard
Proving Reduction

- **Input $\in$ 3-PARTITION $\rightarrow$ Input $\in$ Construction**
  - A, an arbitrary instance of 3-PARTITION which admits a solution
  - $n_0$ is allocated to $P_1$
  - Remaining triplets are allocated to $P_1, \ldots, P_m$
  - $sl(S) = B + 1.5 \leq T$.

- **Input $\in$ Construction $\rightarrow$ Input $\in$ 3-PARTITION**
  - An instance of SCHED which admits a solution
  - Each processor can spend at most $B$ time units
  - $\sum_{i=1}^{3m} w(n_i) = mB$ and $|P| = m$
  - Due to $w(n_i) = a_i$, $\frac{B}{4} < a_i < \frac{B}{2}$
Proving Reduction

- **Input ∈ 3-PARTITION → Input ∈ Construction**
  - **A**, an arbitrary instance of 3-PARTITION which admits a solution
  - $n_0$ is allocated to $P_1$
  - Remaining triplets are allocated to $P_1, \ldots, P_m$
  - $sl(S) = B + 1.5 \leq T$.

- **Input ∈ Construction → Input ∈ 3-PARTITION**
  - An instance of SCHED which admits a solution
  - Each processor can spend at most $B$ time units
  - $\sum_{i=1}^{3m} w(n_i) = mB$ and $|P| = m$
  - Due to $w(n_i) = a_i$, $\frac{B}{4} < a_i < \frac{B}{2}$
  - only 3 nodes can have the exact execution time of $B$
Proving Reduction

- Input ∈ 3-PARTITION → Input ∈ Construction
  - A, an arbitrary instance of 3-PARTITION which admits a solution
  - $n_0$ is allocated to $P_1$
  - Remaining triplets are allocated to $P_1, \ldots, P_m$
  - $sl(S) = B + 1.5 \leq T$.

- Input ∈ Construction → Input ∈ 3-PARTITION
  - An instance of SCHED which admits a solution
  - Each processor can spend at most $B$ time units
  - $\sum_{i=1}^{3m} w(n_i) = mB$ and $|P| = m$
  - Due to $w(n_i) = a_i$, $\frac{B}{4} < a_i < \frac{B}{2}$
  - only 3 nodes can have the exact execution time of B

- 3-PARTITION reduces to SCHED ⇒ SCHED is strongly NP-hard
Pop Quiz #1

- *Unlimited processors*
Pop Quiz #1

- Unlimited processors

Complexity

$\text{SCHED}(G, P_\infty)$ is NP-complete
Pop Quiz #2

- No communication costs
Pop Quiz #2

- No communication costs

**Complexity**

$\text{SCHED-C0}(G, P_{c0})$ is NP-complete
Pop Quiz #3

- No communication costs
- Unlimited processors
Pop Quiz #3

- No communication costs
- Unlimited processors

Complexity

$\text{SCHED-C0}(G, P_{c0})$ is solvable in polynomial time
Heterogeneous Systems

- Diverse set of processors
Heterogeneous Systems

- Diverse set of processors
- Interconnected with a high-speed network
Heterogeneous Systems

- Diverse set of processors
- Interconnected with a high-speed network
- Can mean:

\[ V \times P \rightarrow Q + \text{NP-hard} \]
Heterogeneous Systems

- Diverse set of processors
- Interconnected with a high-speed network
- Can mean:
  1. Same functionality, different speeds
Heterogeneous Systems

- Diverse set of processors
- Interconnected with a high-speed network
- Can mean:
  1. Same functionality, different speeds
  2. Different functional capabilities
Heterogeneous Systems

- Diverse set of processors
- Interconnected with a high-speed network
- Can mean:
  1. Same functionality, different speeds
  2. Different functional capabilities
Heterogeneous Systems

- Diverse set of processors
- Interconnected with a high-speed network
- Can mean:
  1. Same functionality, different speeds
  2. Different functional capabilities
- $w$ replaced by $\omega : V \times P \rightarrow \mathbb{Q}^+$
Heterogeneous Systems

- Diverse set of processors
- Interconnected with a high-speed network
- Can mean:
  1. Same functionality, different speeds
  2. Different functional capabilities

- \( w \) replaced by \( \omega : V \times P \rightarrow \mathbb{Q}^+ \)

- NP-hard
Algorithms - Motivation

- TS is NP-complete in most cases
Algorithms - Motivation

- TS is NP-complete in most cases
- Intractable even for moderate-sized input
Algorithms - Motivation

- TS is NP-complete in most cases
- Intractable even for moderate-sized input
- What can we do?
Algorithms - Motivation

- TS is NP-complete in most cases
- Intractable even for moderate-sized input
- What can we do?

![Cartoon Image](image.png)
Algorithms - Motivation

- TS is NP-complete in most cases
- Intractable even for moderate-sized input
- What can we do?
  - Heuristics!
Algorithms - Motivation

- TS is NP-complete in most cases
- Intractable even for moderate-sized input
- What can we do?
  - Heuristics!
  - and/or other optimization techniques
Taxonomy

Static Task-Scheduling Algorithms

Heuristic Based

Guided Random Search Based

List Scheduling Heuristics

Task Duplication Heuristics

Clustering Heuristics
Taxonomy

Static Task-Scheduling Algorithms

Heuristic Based

Guided Random Search Based

List Scheduling Heuristics

Task Duplication Heuristics

Clustering Heuristics
Taxonomy

Static Task-Scheduling Algorithms

- Heuristic Based
- Guided Random Search Based

Heuristic Based

- List Scheduling Heuristics

Guided Random Search Based

- Task Duplication Heuristics

Clustering Heuristics
Taxonomy

Static Task-Scheduling Algorithms

- Heuristic Based
- Guided Random Search Based

- List Scheduling Heuristics
- Task Duplication Heuristics

- Clustering Heuristics
Taxonomy

Static Task-Scheduling Algorithms

- Heuristic Based
  - List Scheduling Heuristics
  - Clustering Heuristics
- Guided Random Search Based
  - Task Duplication Heuristics
List Scheduling - Motivation

- No FPTAS for TS
List Scheduling - Motivation

- No FPTAS for TS
- PTAS in restricted cases
List Scheduling - Motivation

- No FPTAS for TS
- PTAS in restricted cases
  - $2\sqrt{m}$-approximation for restricted heterogeneous systems
List Scheduling - Motivation

- No FPTAS for TS
- PTAS in restricted cases
  - $2\sqrt{m}$-approximation for restricted heterogeneous systems
  - 2-approximation with greedy approach
List Scheduling - Motivation

- No FPTAS for TS
- PTAS in restricted cases
  - $2\sqrt{m}$-approximation for restricted heterogeneous systems
  - 2-approximation with greedy approach
- HEFT & CPOP
List Scheduling Heuristics

- Class/category of algorithms
List Scheduling Heuristics

- Class/category of algorithms
- Two phase heuristic:
List Scheduling Heuristics

- Class/category of algorithms
- Two phase heuristic:
  - task prioritization
List Scheduling Heuristics

• Class/category of algorithms

• Two phase heuristic:
  ▶ task prioritization
  ▶ processor selection/allocation
List Scheduling Heuristics

- Class/category of algorithms
- Two phase heuristic:
  - task prioritization
  - processor selection/allocation
- Heuristic skeleton
List Scheduling Heuristics

- Class/category of algorithms
- Two phase heuristic:
  - task prioritization
  - processor selection/allocation
- Heuristic skeleton
- Different method in each phase
List Scheduling Heuristics

- Class/category of algorithms
- Two phase heuristic:
  - task prioritization
  - processor selection/allocation
- Heuristic skeleton
- Different method in each phase
- Practical, better results + better scheduling time
List Scheduling Heuristics

- Class/category of algorithms
- Two phase heuristic:
  - task prioritization
  - processor selection/allocation
- Heuristic skeleton
- Different method in each phase
- Practical, better results + better scheduling time
- Complexity dependent on scheme in phases
Additional Definitions

- \( \text{rank}_u \)
- Cost after and including task
- Defined recursively
Additional Definitions

- $\text{rank}_u$
  - Cost after and including task
  - Defined recursively

- $\text{rank}_d$
  - Cost up to task
  - Defined recursively
HEFT & CPOP

- Implement list-scheduling heuristics
- HEFT
  - Heterogeneous Earliest Finish Time
  - Implements an insertion-based policy
- CPOP
  - Critical-Path-on-Processor
  - Tries to speed up the execution of tasks on the critical path
HEFT

- 2 phases

Task prioritization:
- Priority of task: $u$
- Sorting tasks by decreasing order of rank $u$
- Tie-breaking is done randomly:
  - Topological order

Processor selection:
- Insertion-based policy
  - Assign task to processor which minimizes EFT
HEFT

- 2 phases
  - task prioritization

Task prioritization:
- Priority of task = rank
- Sorting tasks by decreasing order of rank
- Tie-breaking is done randomly

Processor selection:
- Insertion-based policy
- Assign task to processor which minimizes EFT
HEFT

- 2 phases
  - task prioritization
  - processor selection/allocation
HEFT

- 2 phases
  - task prioritization
  - processor selection/allocation
- Task prioritization:

  - Priority of task = rank
  - Sorting tasks by decreasing order of rank
  - Tie-breaking is done randomly
  - Topological order

- Processor selection:
  - Insertion-based policy
  - Assign task to processor which minimizes EFT
HEFT

- 2 phases
  - task prioritization
  - processor selection/allocation

- Task prioritization:
  - Priority of task = \( rank_u \)
HEFT

- 2 phases
  - task prioritization
  - processor selection/allocation

Task prioritization:
  - Priority of task = $rank_u$
  - Sorting tasks by decreasing order of $rank_u$
HEFT

- 2 phases
  - task prioritization
  - processor selection/allocation

- Task prioritization:
  - Priority of task = $rank_u$
  - Sorting tasks by decreasing order of $rank_u$
  - Tie-breaking is done randomly
HEFT

- 2 phases
  - task prioritization
  - processor selection/allocation

Task prioritization:
- Priority of task = $rank_u$
- Sorting tasks by decreasing order of $rank_u$
- Tie-breaking is done randomly
- Topological order
HEFT

- 2 phases
  - task prioritization
  - processor selection/allocation

**Task prioritization:**
- Priority of task = $rank_u$
- Sorting tasks by decreasing order of $rank_u$
- Tie-breaking is done randomly
- Topological order

**Processor selection:**
HEFT

- 2 phases
  - task prioritization
  - processor selection/allocation

Task prioritization:
  - Priority of task = $rank_u$
  - Sorting tasks by decreasing order of $rank_u$
  - Tie-breaking is done randomly
  - Topological order

Processor selection:
  - Insertion-based policy
HEFT

- 2 phases
  - task prioritization
  - processor selection/allocation

Task prioritization:
  - Priority of task = $rank_u$
  - Sorting tasks by decreasing order of $rank_u$
  - Tie-breaking is done randomly
  - Topological order

Processor selection:
  - Insertion-based policy
  - Assign task to processor which minimizes EFT
CPOP

- 2 phases
CPOP

- 2 phases
  - task prioritization
CPOP

- 2 phases
  - task prioritization
  - processor selection/allocation
CPOP

- 2 phases
  - task prioritization
  - processor selection/allocation

- Uses a different metric for priorities
CPOP

- 2 phases
  - task prioritization
  - processor selection/allocation

- Uses a different metric for priorities

- Different strategy when assigning tasks to processors
CPOP - Task Prioritization

- Priority of task $= rank_u + rank_d$

Algorithm for finding $CP$:

1. $n_0$ is selected and marked as critical path task
2. Next critical path task, immediate successor with highest priority
3. Until exit node is reached

Implemented using a priority queue
CPOP - Task Prioritization

- Priority of task = $rank_u + rank_d$
- Uses critical path of the given task graph
CPOP - Task Prioritization

- Priority of task = \( rank_u + rank_d \)
- Uses critical path of the given task graph
- \( \text{priority}(n_0) = |CP| \)
CPOP - Task Prioritization

- Priority of task = \( rank_u + rank_d \)
- Uses critical path of the given task graph
- \( \text{priority}(n_0) = |CP| \)
- Algorithm for finding \( CP \):
  1. \( n_0 \) is selected and marked as critical path task
  2. Next critical path task, immediate successor with highest priority
  3. Until exit node is reached

Implemented using a priority queue
CPOP - Task Prioritization

- Priority of task = $rank_u + rank_d$
- Uses critical path of the given task graph
- $\text{priority}(n_0) = |CP|$
- Algorithm for finding $CP$:
  1. $n_0$ is selected and marked as critical path task
CPOP - Task Prioritization

- Priority of task $= \text{rank}_u + \text{rank}_d$
- Uses critical path of the given task graph
- \( \text{priority}(n_0) = |CP| \)
- Algorithm for finding \( CP \):
  1. \( n_0 \) is selected and marked as critical path task
  2. Next critical path task, immediate successor with highest priority
CPOP - Task Prioritization

- Priority of task = \( rank_u + rank_d \)
- Uses critical path of the given task graph
- \( \text{priority}(n_0) = |CP| \)
- Algorithm for finding \( CP \):
  1. \( n_0 \) is selected and marked as critical path task
  2. Next critical path task, immediate successor with highest priority
  3. Until exit node is reached
CPOP - Task Prioritization

- Priority of task = $rank_u + rank_d$
- Uses critical path of the given task graph
- $priority(n_0) = |CP|$
- Algorithm for finding $CP$:
  1. $n_0$ is selected and marked as critical path task
  2. Next critical path task, immediate successor with highest priority
  3. Until exit node is reached
- Implemented using a priority queue
CPOP - Processor Allocation

- Select a $p_{CP}$ which minimizes the cumulative computation cost on the critical path.
CPOP - Processor Allocation

- Select a $p_{CP}$ which minimizes the cumulative computation cost on the critical path
- If a selected task is on the critical path, schedule on $p_{CP}$
CPOP - Processor Allocation

- Select a $p_{CP}$ which minimizes the cumulative computation cost on the critical path
- If a selected task is on the critical path, schedule on $p_{CP}$
- Else assign it to a processor which minimizes its EFT
CPOP - Processor Allocation

- Select a $p_{CP}$ which minimizes the cumulative computation cost on the critical path
- If a selected task is on the critical path, schedule on $p_{CP}$
- Else assign it to a processor which minimizes its EFT
- Both cases consider an insertion-based scheduling policy
Experiments

- Algorithms tested on two sets of graphs:
  - Randomly generated application graphs
  - Graphs representing real world problems
    - Parametrized random graph generator
    - About 56K DAGs.
  - Task graphs of real world applications
    - Gaussian Elimination
    - FFT
    - Molecular Dynamics Code
Experiments

- Algorithms tested on two sets of graphs:
  - Randomly generated application graphs
  - Graphs representing real world problems
    - Gaussian Elimination
    - FFT
    - Molecular Dynamics Code
Experiments

- Algorithms tested on two sets of graphs:
  - Randomly generated application graphs
  - Graphs representing real world problems
Experiments

- Algorithms tested on two sets of graphs:
  - Randomly generated application graphs
  - Graphs representing real world problems

- Randomly generated application graphs
Experiments

- Algorithms tested on two sets of graphs:
  - Randomly generated application graphs
  - Graphs representing real world problems

- Randomly generated application graphs
  - Parametrized random graph generator
Experiments

- Algorithms tested on two sets of graphs:
  - Randomly generated application graphs
  - Graphs representing real world problems

- Randomly generated application graphs
  - Parametrized random graph generator
  - About 56K DAGs.
Experiments

- Algorithms tested on two sets of graphs:
  - Randomly generated application graphs
  - Graphs representing real world problems

- Randomly generated application graphs
  - Parametrized random graph generator
  - About 56K DAGs.

- Task graphs of real world applications
Experiments

- Algorithms tested on two sets of graphs:
  - Randomly generated application graphs
  - Graphs representing real world problems

- Randomly generated application graphs
  - Parametrized random graph generator
    - About 56K DAGs.

- Task graphs of real world applications
  - Gaussian Elimination
Experiments

- Algorithms tested on two sets of graphs:
  - Randomly generated application graphs
  - Graphs representing real world problems

- Randomly generated application graphs
  - Parametrized random graph generator
  - About 56K DAGs.

- Task graphs of real world applications
  - Gaussian Elimination
  - FFT
Experiments

- Algorithms tested on two sets of graphs:
  - Randomly generated application graphs
  - Graphs representing real world problems

- Randomly generated application graphs
  - Parametrized random graph generator
  - About 56K DAGs.

- Task graphs of real world applications
  - Gaussian Elimination
  - FFT
  - Molecular Dynamics Code
Competing Algorithms

- Dynamic-Level Scheduling (DLS)
Competing Algorithms

- Dynamic-Level Scheduling (DLS)
- Mapping Heuristic (MH)
Competing Algorithms

- Dynamic-Level Scheduling (DLS)
- Mapping Heuristic (MH)
- Levelized-Min Time (LMT)
Comparison Metrics

- **Schedule Length Ratio (SLR)**
  - SLR is a normalized schedule length for an algorithm
  - The SLR value for an algorithm is given by:
    \[
    SLR = \frac{\text{makespan}}{\sum_{n_i \in CP_{\text{min}}} \min_{p_j \in Q} w_{ij}}
    \]

- **Run time**
Avg. SLR

![Graph showing the average SLR across different numbers of nodes for various scheduling algorithms: HEFT, CPOP, MH, DLS, and LMT.](image)
Avg. Runtime

![Graph showing average runtime vs number of nodes for different task scheduling algorithms: HEFT, CPOP, MH, DLS, LMT. Each algorithm is represented by a unique symbol and line style.]
Comparison Metrics (contd.)

- **Speedup**
  - The speedup value for a given graph is computed by dividing the sequential execution time by the parallel execution time.
  - Its value is given by:
    \[
    Speedup = \sum_{n_i \in CP_{\text{min}}} \min_{p_j \in Q} w_{ij} \over make\text{span}
    \]

- **Efficiency**
  - Efficiency is calculated by dividing the speedup by the number of processors.
Avg. Speedup

![Graph showing the average speedup for different task scheduling algorithms. The x-axis represents the number of nodes, and the y-axis represents the average speedup. The graph compares HEFT, CPOP, MH, DLS, and LMT algorithms.]
Efficiency - Gaussian Elimination
Result Summary

- HEFT pwns everyone
- CPOP isn’t far behind
- Alternative task prioritizing
- and processor selection policies for HEFT
Conclusion

- Static TS is NP-complete in a strong sense
- Heterogeneous systems are important, TS on them more so
- Two list heuristic based algorithms: CPOP and HEFT
- Significantly outperform their competitors
Questions?
Bibliography

- Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Systems; H. Topcuoglu, S. Hariri and M. Wu
- Task Scheduling for Parallel Systems; O. Sinnen
- Approximation Algorithms for Scheduling Unrelated Parallel Machines; J. Lenstra, D. Shmoys and E. Tardos
- Algorithms for Scheduling Tasks on Unrelated Processors; E. Davis, J. Jaffe