A Divide and Conquer Algorithm for DAG Scheduling under Power Constraints

Gökalp Demirci    Ivana Marincic    Henry Hoffmann

THE UNIVERSITY OF
CHICAGO

Department of
Computer Science
Scheduling **DAGs** under **Power** Constraints
Scheduling **DAGs** under **Power** Constraints

Consider n tasks

Both constraints are easy to handle separately (to get theoretical bounds):

**DAG** scheduling:
- Precedence constraints
- Any greedy algorithm: 2-approx ['66]
- Given >50 years ago

**Power** constrained scheduling
- Independent tasks
- Any greedy algorithm: 3-approx ['75]
- Similar to packing problems

Scheduling **DAGs** under **Power** Constraints:
- No formal bounds until very recently
- Greedy algorithms could be $O(n)$-approx
- $O(\log n)$-approx using a divide-and-conquer technique [Demirci, Hoffmann, Kim’18]

This paper:
Scheduling **DAGs** under **Power** Constraints on **Configurable** Hardware (e.g., DVFS)
Scheduling **DAGs** under **Power** Constraints

Schedule \( n \) tasks on \( m \) parallel machines to minimize total run-time

**DAG constraints**
- If \( j \prec j' \): finish \( j \) before starting \( j' \)
- Task based programming, workflow management, precedence constraints...

**Power constraints**
- Each task \( j \) needs \( p_j \) amount power
- Total available power at any point: \( P \)
- Tasks running in parallel \( \sum_j p_j \leq P \)

Start with the simpler problem:
No configurations – e.g. fixed \( (t_j, p_j) \)
Scheduling **DAGs** under **Power** Constraints

**D&C Algorithm**: No configurations, e.g. fixed \((t_j, p_j)\)

- Get a schedule satisfying the **DAG** w/ \(\infty\) parallelism ignoring the power constraints
- Consider the subset of tasks crossing the mid point: no **DAG dependencies** among them
- Schedule them in a separate fragment in the middle (greedily) now considering **power**
- Recurse on both sides
Scheduling DAGs under Power Constraints

D&C Algorithm: No configurations, e.g. fixed \((t_j, p_j)\)

Analysis idea:

- O(1) loss in every level of the recursion
- Depth of the recursion: O(log n)
- O(log n) - approximation

Length of the fragment \(\approx\) max-time + total area / \(P\)

\(\approx O(1)\times\) optimum schedule length
Scheduling **DAGs** under **Power** Constraints

**Scheduling with *Configurations***:

- Each task can be one of many rectangles (determined by configurations)
  - Length is time
  - Height is power

- **Intuition**:
  - Tallest rectangle completes work fastest (*race* configuration)
  - Smallest area rectangle is most energy efficient (*pace* configuration)
  - Greedy schedulers have to favor one or the other

✓ We can choose *configurations* such that it schedules within $O(\log n)$ of optimum!
Scheduling **DAGs** under **Power** Constraints

Scheduling with *Configurations*:

- Motivated by HPC systems: hardware configs with different power/run-time tradeoffs
  - (2 active cores, high DVFS, 2 mem. contr.) vs (4 active cores, low DVFS, 1 mem. contr.)

> We can choose *configurations* such that it schedules within $O(\log n)$ of optimum!
Scheduling **DAGs** under **Power** Constraints

Scheduling with *configurations*: (≈ no configurations)

- Not hard to argue there exists an iteration where
  1) \( \text{max-time} \leq \text{max-time}_{\text{OPT}} \) (Tall rectangles)
  2) \( \text{area-of-each} \leq \text{area-of-each}_{\text{OPT}} \) (Minimal area rectangles)

- Our configs need to be comparable to opt. schedule`s configs in terms of 1) max-time and 2) total area
- Start with pace configurations (min area configurations)
- Enumerate set of configs by replacing only the max-time task in every iteration
- Replace it w/ the configuration that has the smallest area among configs “shorter” than the current

\[ \approx \max\{t_\text{j}\} + \text{total area}/P \]
Scheduling **DAGs** under **Power** Constraints

Greedy Algorithms:

- Only major design choice is the order in which available tasks are considered

- Three popular choices in state-of-the-art **DAG** schedulers or **Power** schedulers:
  - G1: first in (the queue of available), first out
  - G2: best-fit to power budget, i.e. pace power closest to available power
  - G3: non-increasing run-time (in pace configuration) order

- We implement and compare our D&C algorithm to all three
### Scheduling DAGs under Power Constraints

Percent improvement of D&C over best of G1, G2, G3 in each setting:

<table>
<thead>
<tr>
<th>Different DAGs</th>
<th>Backprop</th>
<th>Kmeans</th>
<th>Npb-Dc</th>
<th>Npb-Ic</th>
<th>Swift1</th>
<th>Swift2</th>
<th>Synth-Ig-long</th>
</tr>
</thead>
<tbody>
<tr>
<td>#machines=10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P=100W</td>
<td>60.3%</td>
<td>60.0</td>
<td>67.4</td>
<td>64.6</td>
<td>63.9</td>
<td>72.0</td>
<td>32.7</td>
</tr>
<tr>
<td>P=200</td>
<td>35.4</td>
<td>36.9</td>
<td>36.2</td>
<td>43.8</td>
<td>40.4</td>
<td>46.2</td>
<td>13.4</td>
</tr>
<tr>
<td>P=300</td>
<td>5.8</td>
<td>18.5</td>
<td>9.9</td>
<td>15.5</td>
<td>4.5</td>
<td>3.6</td>
<td>5.0</td>
</tr>
<tr>
<td>P=400</td>
<td>3.8</td>
<td>3.1</td>
<td>-7.2</td>
<td>11.8</td>
<td>-12.2</td>
<td>0.0</td>
<td>2.3</td>
</tr>
<tr>
<td>P=500</td>
<td>-8.2</td>
<td>5.3</td>
<td>-15.7</td>
<td>-3.0</td>
<td>-14.3</td>
<td>2.4</td>
<td>-18.8</td>
</tr>
</tbody>
</table>

| #machines=20   |          |        |        |        |        |        |              |
| P=200W         | 62.3%    | 61.5   | 59.8   | 72.8   | 74.6   | 64.3   |              |
| P=400          | 39.4     | 37.8   | 49.0   | 39.2   | 27.7   | 30.0   |              |
| P=600          | 2.3      | 12.9   | -8.7   | 8.2    | 10.0   | -13.2  |              |
| P=800          | 1.9      | 3.4    | -27.2  | 5.6    | 6.8    | -20.4  |              |
| P=1000         | -25.5    | -10    | -15.5  | -9.3   | -2.3   | -24.7  |              |

Mean, median pace power $\approx 20W$

We have more results in the paper!

As we relax the power-cap,
- Problem turns into no power DAG scheduling
- D&C algorithm (designed for power-cap) loses its edge
Scheduling **DAGs** under **Power** Constraints

Edge of D&C explained:

To achieve min total run-time: (for any algorithm)

1) High power budget utilization (close to race **configs**), and
2) Low energy consumption of individual configurations (close to pace **configs**)

D&C aligns tasks with “similar run-times” together which, in turn, allows it to achieve high power utilization (1) without having to deviate from low energy configs too much (2) 

---------- (1) and (2) **simultaneously**!
Takeaways:

• “Scheduling **DAGs** under **power** constraints” has a significantly different structure than just “scheduling **DAGs**”

• Greedy algorithms have no formal bounds

• D&C algorithm:
  ✓ has a formal bound
  ✓ performs better than greedy, because
    ✓ high power utilization without having to deviate from pace configs too much
    ✓ aligns similar run-time tasks in the same batch and runs batches w/o interleaving
  ✓ promising for other future applications to **DAG** scheduling possibly w/o resources