Energy Minimization in DAG Scheduling on MPSoCs at Run-Time: Theory and Practice

Bertrand Simon$^1$  Joachim Falk$^2$  Nicole Megow$^1$  Jürgen Teich$^1$

1: University of Bremen, Germany.
2: FAU Erlangen-Nürnberg, Germany.

Bologna – January 2020
**Introduction**

**DVFS (Dynamic Voltage and Frequency Scaling)**

- **Objective:** decrease the energy consumption of processors
- **Constraint:** respect a strong deadline

**Motivation example**

- **Application** run multiple times, exact characteristics depend on the workload
- **Some settings** [Choudhury et al. 2007, Singh et al. 2013]: compute a (pessimistic) pseudo-schedule offline, adapt it online
- **Ideal:** compute online (i.e., fast) a solution with low energy consumption

**Problematic**

Are there theoretically-guaranteed algorithms that are fast enough to be executed at run-time?
Preliminaries

General statement of the problem

- DAG $G = (V, E)$ of $n$ tasks of known lengths $w_j$
- $m$ cores, whose speeds can be modified between two tasks
- Strong deadline $D$

$\Rightarrow$ Minimize the energy

\[ \text{Energy model: DVFS} \]

\[ \text{Power} = \text{speed}^\alpha, \text{for some } \alpha > 1 \]

Energy consumed for a task of length $w_j$, run at speed $s_j$

- Execution time: $x_j = w_j/s_j$
- Energy $E_j = x_j \cdot s_j^\alpha = w_j \cdot s_j^{\alpha-1} = w_j^\alpha/x_j^{\alpha-1}$

$\Rightarrow$ Objective: minimize $\sum E_j$
Four variants of the problem

Two scenarios

▶ **Speed&Scheduling** – the problem is to:
  - decide at which speed each task is run;
  - schedule each task to a core, respecting precedences.

▶ **SpeedScaling** – the task-to-core mapping and each core’s execution order is fixed
  The problem is to:
  - decide the speed of each task

Two speed models

▶ Continuous speeds: all speeds in \([s_{\text{min}}, s_{\text{max}}]\)
▶ Discrete speeds: choose speeds among \(v_1, \ldots, v_k\)

Theoretical guarantees targeted: e.g., 2-approximation

▶ Deadline is always met (if feasible)
▶ Energy consumed is at most 2 times the best possible
Outline

1. Introduction

2. Continuous speeds
   - The **SPEEDSCALING** setting
   - The **SPEED&SCHEDULING** setting

3. Discrete speeds
   - The **SPEEDSCALING** setting
   - The **SPEED&SCHEDULING** setting

4. Experimental results
Optimal polynomial solution: convex program
[Aupy et al. 2013]

Note: include each core’s order into the precedence constraints

→ execution time = critical path

Convex program computing the optimal speeds

\( x_j \): execution duration of task \( j \) \( (x_j = w_j/s_j) \)

\( d_j \): completion time of task \( j \)

\[
\begin{align*}
\min \quad & \sum_{j \in V} \frac{w_j^\alpha}{x_j^{\alpha-1}} \\
\text{s.t.} \quad & d_j \leq D, \quad \forall j \in V \\
\quad & x_j \leq d_j, \quad \forall j \in V \\
\quad & d_j + x_k \leq d_k, \quad \forall (j, k) \in E \\
\quad & \frac{w_j}{s_{\text{max}}} \leq x_j \leq \frac{w_j}{s_{\text{min}}}, \quad \forall j \in V.
\end{align*}
\]
Optimal polynomial solution: convex program
[Aupy et al. 2013]

**Note:** include each core’s order into the precedence constraints

→ execution time = critical path

Convex program computing the optimal speeds

$x_j$: execution duration of task $j$ ($x_j = w_j/s_j$)
$d_j$: completion time of task $j$

\[
\text{min} \quad \sum_{j \in V} \frac{w_j^{\alpha}}{x_j^{\alpha-1}}
\]

s.t.

\[
d_j \leq D, \quad \forall j \in V
\]

\[
x_j \leq d_j, \quad \forall j \in V
\]

\[
d_j + x_k \leq d_k, \quad \forall (j, k) \in E
\]

\[
w_j/s_{\text{max}} \leq x_j \leq w_j/s_{\text{min}}, \quad \forall j \in V.
\]
Optimal linear-time solution for Series-Parallel graphs

[Prasanna Musicus 1996] in an other context

Series-Parallel graph

Main algorithm idea: each subgraph is equivalent to a single task

$\begin{align*}
\text{1.} & \quad w(T) = w_T \\
\text{2.} & \quad w(G_1; G_2) = w(G_1) + w(G_2) \\
\text{3.} & \quad w(G_1 \parallel G_2)^\alpha = w(G_1)^\alpha + w(G_2)^\alpha
\end{align*}$

Algorithm sketch (drawback: ignores $s_{\text{min}}$ and $s_{\text{max}}$)

1. Compute the equivalent task of $G$: assign it the speed $w(G)/D$
2. Propagate the speed assignment through the graph structure (series: conserve speed, parallel: conserve execution time)
Optimal linear-time solution for Series-Parallel graphs

[Prasanna Musicus 1996] in an other context

Series-Parallel graph

\[ G_1 \to G_2 \]

\( G_1; G_2 \)

Main algorithm idea: each subgraph is equivalent to a single task

\[ w(T) = w_T \]

\[ w(G_1 ; G_2) = w(G_1) + w(G_2) : \]

\[ w(G_1 \parallel G_2)^\alpha = w(G_1)^\alpha + w(G_2)^\alpha : \]

Algorithm sketch (drawback: ignores \( s_{min} \) and \( s_{max} \))

1. Compute the equivalent task of \( G \): assign it the speed \( w(G)/D \)
2. Propagate the speed assignment through the graph structure
   (series: conserve speed, parallel: conserve execution time)
Optimal linear-time solution for Series-Parallel graphs
[Prasanna Musicus 1996] in an other context

Series-Parallel graph

$G_1 \parallel G_2$

Main algorithm idea: each subgraph is equivalent to a single task

- $w(T) = w_T$
- $w(G_1; G_2) = w(G_1) + w(G_2)$:
- $w(G_1 \parallel G_2)^\alpha = w(G_1)^\alpha + w(G_2)^\alpha$:

Algorithm sketch (drawback: ignores $s_{min}$ and $s_{max}$)

1. Compute the equivalent task of $G$: assign it the speed $w(G)/D$
2. Propagate the speed assignment through the graph structure
   (series: conserve speed, parallel: conserve execution time)
Optimal linear-time solution for Series-Parallel graphs

[Prasanna Musicus 1996] in an other context

Series-Parallel graph

Main algorithm idea: each subgraph is equivalent to a single task

1. \( w(T) = w_T \)
2. \( w(G_1 ; G_2) = w(G_1) + w(G_2) : \)
3. \( w(G_1 \parallel G_2)^\alpha = w(G_1)^\alpha + w(G_2)^\alpha : \)

Algorithm sketch (drawback: ignores \( s_{\text{min}} \) and \( s_{\text{max}} \))

1. Compute the equivalent task of \( G \): assign it the speed \( w(G)/D \)
2. Propagate the speed assignment through the graph structure
   (series: conserve speed, parallel: conserve execution time)
Optimal linear-time solution for Series-Parallel graphs

[Prasanna Musicus 1996] in an other context

Series-Parallel graph

Main algorithm idea: each subgraph is equivalent to a single task

- \( w(T) = w_T \)
- \( w(G_1 ; G_2) = w(G_1) + w(G_2) : \)
- \( w(G_1 \parallel G_2)^\alpha = w(G_1)^\alpha + w(G_2)^\alpha : \)

Algorithm sketch (drawback: ignores \( s_{\text{min}} \) and \( s_{\text{max}} \))

1. Compute the equivalent task of \( G \): assign it the speed \( w(G)/D \)
2. Propagate the speed assignment through the graph structure
   (series: conserve speed, parallel: conserve execution time)
Optimal linear-time solution for Series-Parallel graphs
[Prasanna Musicus 1996] in an another context

Series-Parallel graph

\[ G_1 \rightarrow G_2 \]

\( G_1; G_2 \)

Main algorithm idea: each subgraph is equivalent to a single task

- \( w(T) = w_T \)
- \( w(G_1; G_2) = w(G_1) + w(G_2) \):
- \( w(G_1 \parallel G_2)^\alpha = w(G_1)^\alpha + w(G_2)^\alpha \):

Algorithm sketch (drawback: ignores \( s_{\text{min}} \) and \( s_{\text{max}} \))

1. Compute the equivalent task of \( G \): assign it the speed \( w(G)/D \)
2. Propagate the speed assignment through the graph structure
   (series: conserve speed, parallel: conserve execution time)
Optimal linear-time solution for Series-Parallel graphs

[Prasanna Musicus 1996] in an other context

Series-Parallel graph

\[ G_1 \parallel G_2 \]

Main algorithm idea: each subgraph is equivalent to a single task

- \( w(T) = w_T \)
- \( w(G_1; G_2) = w(G_1) + w(G_2) \):
  \[ 1 \rightarrow 1 = 2 \]
- \( w(G_1 \parallel G_2)^\alpha = w(G_1)^\alpha + w(G_2)^\alpha \):
  \[ 1 \rightarrow 1^\alpha = 2^{1/\alpha} \]

Algorithm sketch (drawback: ignores \( s_{min} \) and \( s_{max} \))

1. Compute the equivalent task of \( G \): assign it the speed \( w(G)/D \)
2. Propagate the speed assignment through the graph structure
   (series: conserve speed, parallel: conserve execution time)
Optimal linear-time solution for Series-Parallel graphs

[Prasanna Musicus 1996] in another context

Series-Parallel graph

Main algorithm idea: each subgraph is equivalent to a single task

- \( w(T) = w_T \)
- \( w(G_1; G_2) = w(G_1) + w(G_2) \):
- \( w(G_1 \parallel G_2)^{\alpha} = w(G_1)^{\alpha} + w(G_2)^{\alpha} \):

Algorithm sketch (drawback: ignores \( s_{\text{min}} \) and \( s_{\text{max}} \))

1. Compute the equivalent task of \( G \): assign it the speed \( w(G)/D \)
2. Propagate the speed assignment through the graph structure (series: conserve speed, parallel: conserve execution time)
Outline

1. Introduction

2. Continuous speeds
   - The SPEEDSCALING setting
   - The SPEED&SCHEDULING setting

3. Discrete speeds
   - The SPEEDSCALING setting
   - The SPEED&SCHEDULING setting

4. Experimental results
Approximate solution  
similar to [Bampis, Letsios, Lucarelli 2014]

**Issue to obtain an approximation-algorithm**

- with fixed speeds, scheduling is NP-hard
- need to assume that the deadline is loose to be able to meet it

**Theorem**

*If OPT uses speeds at most $s_{\text{max}}/2$, there is a $2^{\alpha-1}$-approximation.*

**Algorithm sketch**

- Solve the previous CP adding the constraints ($m = \#\text{cores}$):

  \[
  \sum_{j \in V} x_j / m \leq D/2 \quad ; \quad d_j \leq D/2 \quad \forall j \in V
  \]

- Use Graham’s list-scheduling or any such simple algorithm
- If there is some slack towards the deadline, scale down the speeds
Outline

1. Introduction

2. Continuous speeds
   - The SPEEDSCALING setting
   - The SPEED&SCHEDULING setting

3. Discrete speeds
   - The SPEEDSCALING setting
   - The SPEED&SCHEDULING setting

4. Experimental results
### Results for continuous speeds

<table>
<thead>
<tr>
<th>Speed Scaling</th>
<th>Exact Solution</th>
<th>Approximate Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed&amp;Scaling</td>
<td>Convex Program</td>
<td>SP-graphs (restricted instances)</td>
</tr>
</tbody>
</table>

- **Speed&Scaling**
  - Convex Program
  - + Rounding
  - + List Scheduling
  - $(2^{\alpha-1}\text{-approx})$

### This section: discrete speeds

- $v_1 \leq v_2 \leq \cdots \leq v_k$
- Define $r := \max_i \frac{v_{i+1}}{v_i}$
# Outline

1. **Introduction**

2. **Continuous speeds**
   - The \textsc{SpeedScaling} setting
   - The \textsc{Speed&Scheduling} setting

3. **Discrete speeds**
   - The \textsc{SpeedScaling} setting
   - The \textsc{Speed&Scheduling} setting

4. **Experimental results**
Optimal exponential-time solution: ILP

$y_{i,\ell}$: boolean variable deciding whether task $i$ is run at speed $v_\ell$

$d_i$: completion time of task $i$

\[
\begin{align*}
\text{minimize} & \quad \sum_{i \in V} w_i \left( \sum_{\ell \leq k} v_\ell^{\alpha-1} y_{i,\ell} \right) \\
\text{subject to} & \quad d_i \leq D \quad \forall i \in V \\
& \quad \left( \sum_{\ell \leq k} \frac{w_i}{v_\ell} y_{i,\ell} \right) \leq d_i \quad \forall i \in V \\
& \quad d_i + \left( \sum_{\ell \leq k} \frac{w_j}{v_\ell} y_{j,\ell} \right) \leq d_j \quad \forall (i,j) \in E \\
& \quad \sum_{\ell \leq k} y_{i,\ell} = 1 \quad \forall i \in V \\
& \quad y_{i,\ell} \in \{0, 1\} \quad \forall i \in V, \forall \ell \leq k. 
\end{align*}
\]
Approximate solution

Simple algorithm

1. Compute the optimal continuous-speed solution
   (with $s_{\text{min}} = v_1$, $s_{\text{max}} = v_k$)
2. Round up each speed

Note: we can use the fast SP-graph algorithm or the convex program

Theorem

This algorithm is a $r^{\alpha-1}$-approximation.

Recall: $r = \max_i \frac{v_{i+1}}{v_i}$
Outline

1 Introduction

2 Continuous speeds
   - The SPEEDSCALING setting
   - The SPEED&SCHEDULING setting

3 Discrete speeds
   - The SPEEDSCALING setting
   - The SPEED&SCHEDULING setting

4 Experimental results
The Speed&Scheduling setting

Optimal exponential-time solution via an ILP

- Needs $n(n + m)$ boolean variables: really prohibitive complexity

Approximate solution

- Combine both previous approximation schemes (assuming OPT uses speeds at most $v_k/2$)
- Compute the approximate continuous speed solution then round up the speeds
- Guarantee: $(2r)^{\alpha-1}$-approximation
Outline

1. Introduction

2. Continuous speeds
   - The SPEEDSCALING setting
   - The SPEED&SCHEDULING setting

3. Discrete speeds
   - The SPEEDSCALING setting
   - The SPEED&SCHEDULING setting

4. Experimental results
The **SpeedScaling** setting

<table>
<thead>
<tr>
<th>Approximate Solution</th>
<th>Exact Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuous speeds</td>
<td>SP-graph (exact solution)</td>
</tr>
<tr>
<td>Discrete speeds</td>
<td>SP-G + rounding ($r^{\alpha-1}$-approx)</td>
</tr>
</tbody>
</table>

### Datasets (all SP-graphs)
- **E3S** ($\approx 10$ tasks)
- **GENOME** (Pegasus, 50 to 1000 tasks)
- Discrete speeds: 20 equally distributed

### Results (bottom left is better)
- **SP-G**: 1ms for 1000 tasks
- **CVX**: 15ms for 100 tasks
- Discrete speeds: almost optimal
The **SPEED & SCHEDULING** setting

<table>
<thead>
<tr>
<th></th>
<th>Approximate Solution</th>
<th>Exact Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuous speeds</td>
<td>Convex Program + List Scheduling $(2^{\alpha-1}\text{-approx})$</td>
<td></td>
</tr>
<tr>
<td>Discrete speeds</td>
<td>Convex Program + Rounding + List Scheduling $(2r)^{\alpha-1}\text{-approx})$</td>
<td>Prohibitive ILP</td>
</tr>
</tbody>
</table>

**Convex Program running time:**

- 100 tasks in 25ms
- 500 tasks in 75ms
- 1000 tasks in 140ms
Conclusion

Results

- SpeedScaling, SP-graphs: almost-optimal solution can be computed very fast
- Other settings: guaranteed algorithms exist but are slower benefits depend on the application

Future work

- Integration of such algorithms in a run-time resource management framework