Energy-efficient scheduling

Guillaume Aupy\textsuperscript{1}, Anne Benoit\textsuperscript{1,2},
Paul Renaud-Goud\textsuperscript{1} and Yves Robert\textsuperscript{1,2,3}

1. Ecole Normale Supérieure de Lyon, France
2. Institut Universitaire de France
3. University of Tennessee Knoxville, USA

Anne.Benoit@ens-lyon.fr
http://graal.ens-lyon.fr/~abenoit/

Dagstuhl Seminar 13381, September 2013
Algorithms and Scheduling Techniques for Exascale Systems
Energy: a crucial issue

- Data centers
  - 330,000,000,000 Watts hour in 2007: more than France
  - 533,000,000 tons of CO₂: in the top ten countries

- Exascale computers (10^{18} floating operations per second)
  - Need effort for feasibility
  - 1% of power saved \( \sim \) 1 million dollar per year

- Lambda user
  - 1 billion personal computers
  - 500,000,000,000,000 Watts hour per year

\( \sim \) crucial for both environmental and economical reasons
Energy: a crucial issue

- Data centers
  - 330,000,000,000 Watts hour in 2007: more than France
  - 533,000,000,000 tons of CO₂: in the top ten countries

- Exascale computers (10¹⁸ floating operations per second)
  - Need effort for feasibility
  - 1% of power saved; 1 million dollar per year

- Lambda user
  - 1 billion personal computers
  - 500,000,000,000,000 Watts hour per year

- Crucial for both environmental and economical reasons
Power dissipation of a processor

\[ P = P_{\text{leak}} + P_{\text{dyn}} \]

- \( P_{\text{leak}} \): constant
- \( P_{\text{dyn}} = B \times V^2 \times f \)

Standard approximation: \( P = P_{\text{leak}} + f^\alpha \) \quad (2 \leq \alpha \leq 3)

Energy \( E = P \times \text{time} \)

**Dynamic Voltage and Frequency Scaling**
- Real life: discrete speeds
- Continuous speeds can be emulated
Outline

1. Revisiting the greedy algorithm for independent jobs
2. Reclaiming the slack of a schedule
3. Tri-criteria problem: execution time, reliability, energy
4. Checkpointing and energy consumption
5. Conclusion
Framework

- Scheduling independent jobs

- **Greedy algorithm**: assign next job to least-loaded processor

- Two variants:
  - **Online-Greedy**: assign jobs on the fly
  - **Offline-Greedy**: sort jobs before execution
Classical problem

- $n$ independent jobs $\{J_i\}_{1 \leq i \leq n}$, $a_i =$ size of $J_i$
- $p$ processors $\{P_q\}_{1 \leq q \leq p}$
- allocation function $\text{alloc}: \{J_i\} \rightarrow \{P_q\}$
- load of $P_q = \text{load}(q) = \sum\{i \mid \text{alloc}(J_i) = P_q\} \ a_i$

Execution time:
$\max_{1 \leq q \leq p} \text{load}(q)$
**Theorem**

**ONLine-Greedy** *is a* $2 - \frac{1}{p}$ *approximation (tight bound)*

<table>
<thead>
<tr>
<th>$\mathcal{P}_1$</th>
<th>$\mathcal{P}_2$</th>
<th>$\mathcal{P}_3$</th>
<th>$\mathcal{P}_4$</th>
<th>$\mathcal{P}_5$</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**ONLine-Greedy**

**Optimal solution**
**OffLine-Greedy**

---

**Theorem**

**OffLine-Greedy** is a $\frac{4}{3} - \frac{1}{3p}$ approximation (tight bound)

---

<table>
<thead>
<tr>
<th>$p_1$</th>
<th>9</th>
<th>5</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_2$</td>
<td>9</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>$p_3$</td>
<td>8</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>$p_4$</td>
<td>8</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>$p_5$</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$p_1$</th>
<th>5</th>
<th>5</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_2$</td>
<td>9</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>$p_3$</td>
<td>9</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>$p_4$</td>
<td>8</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>$p_5$</td>
<td>8</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

OffLine-Greedy

Optimal solution
Bi-criteria problem

- Minimizing (dynamic) power consumption:
  \[ P_{dyn} = f^3 \]
  \[ \Rightarrow \] use slowest possible speed

- Bi-criteria problem:
  Given bound \( M = 1 \) on execution time,
  minimize power consumption while meeting the bound
Bi-criteria problem statement

- $n$ independent jobs $\{J_i\}_{1 \leq i \leq n}$, $a_i =$ size of $J_i$
- $p$ processors $\{\mathcal{P}_q\}_{1 \leq q \leq p}$
- allocation function $\text{alloc} : \{J_i\} \rightarrow \{\mathcal{P}_q\}$
- load of $\mathcal{P}_q = \text{load}(q) = \sum\{i \mid \text{alloc}(J_i) = \mathcal{P}_q\} a_i$

$(\text{load}(q))^3$ power dissipated by $\mathcal{P}_q$

$$\sum_{q=1}^{p} (\text{load}(q))^3 \quad \text{Power}$$

$$\max_{1 \leq q \leq p} \text{load}(q) \quad \text{Execution time}$$
Same **Greedy** algorithm . . .

- **Strategy:** assign next job to least-loaded processor

- **Natural for execution-time**
  - smallest increment of maximum load
  - minimize objective value for currently processed jobs

- **Natural for power too**
  - smallest increment of total power (convexity)
  - minimize objective value for currently processed jobs
... but different optimal solution!

<table>
<thead>
<tr>
<th>( p_1 )</th>
<th>( p_2 )</th>
<th>( p_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

- Makespan 10, power 2531.441
- Makespan 10.1, power 2488.301
Greedy and $L_r$ norms

$$N_r = \left( \sum_{q=1}^{p} (\text{load}(q))^r \right)^{\frac{1}{r}}$$

- Execution time $N_\infty = \lim_{r \to \infty} N_r = \max_{1 \leq q \leq p} \text{load}(q)$
- Power $(N_3)^3$
Known results

$N_2$, \texttt{OffLine-Greedy}
- Chandra and Wong 1975: upper and lower bounds
- Leung and Wei 1995: tight approximation factor

$N_3$, \texttt{OffLine-Greedy}
- Chandra and Wong 1975: upper and lower bounds

$N_r$
- Alon et al. 1997: PTAS for offline problem
- Avidor et al. 1998: upper bound $2 - \Theta(\ln \frac{r}{r})$ for \texttt{OnLine-Greedy}
Contribution

\( N_3 \)

- Tight approximation factor for **ONLINE-GREEDY**
- Tight approximation factor for **OFFLINE-GREEDY**

- Greedy for power fully solved!
Approximation for **ONLINE-GREEDY**

\[ \frac{P_{\text{online}}}{P_{\text{opt}}} \leq \frac{\frac{1}{p^3} \left( (1 + (p - 1)\beta)^3 + (p - 1)(1 - \beta)^3 \right)}{\beta^3 + \frac{(1-\beta)^3}{(p-1)^2}} \]

**Theorem**

- \( f_p^{(\text{on})} \) has a single maximum in \( \beta_p^{(\text{on})} \in \left[ \frac{1}{p}, 1 \right] \)
- **ONLINE-GREEDY** is a \( f_p^{(\text{on})}(\beta_p^{(\text{on})}) \) approximation
- This approximation factor is tight
Approximation for **OffLine-Greedy**

\[
\frac{P_{\text{offline}}}{P_{\text{opt}}} \leq \frac{1}{p^3} \left( \left( 1 + \frac{(p-1)\beta}{3} \right)^3 + (p - 1) \left( 1 - \frac{\beta}{3} \right)^3 \right) \beta^3 + \frac{(1-\beta)^3}{(p-1)^2} \]

**Theorem**

- \( f_p^{(\text{off})} \) has a single maximum in \( \beta_p^{(\text{off})} \in \left[ \frac{1}{p}, 1 \right] \)
- **OffLine-Greedy** is a \( f_p^{(\text{off})}(\beta_p^{(\text{off})}) \) approximation
- This approximation factor is tight
### Numerical values of approximation ratios

<table>
<thead>
<tr>
<th>$p$</th>
<th><strong>ONLINE-GREEDY</strong></th>
<th><strong>OFFLINE-GREEDY</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.866</td>
<td>1.086</td>
</tr>
<tr>
<td>3</td>
<td>2.008</td>
<td>1.081</td>
</tr>
<tr>
<td>4</td>
<td>2.021</td>
<td>1.070</td>
</tr>
<tr>
<td>5</td>
<td>2.001</td>
<td>1.061</td>
</tr>
<tr>
<td>6</td>
<td>1.973</td>
<td>1.054</td>
</tr>
<tr>
<td>7</td>
<td>1.943</td>
<td>1.048</td>
</tr>
<tr>
<td>8</td>
<td>1.915</td>
<td>1.043</td>
</tr>
<tr>
<td>64</td>
<td>1.461</td>
<td>1.006</td>
</tr>
<tr>
<td>512</td>
<td>1.217</td>
<td>1.00083</td>
</tr>
<tr>
<td>2048</td>
<td>1.104</td>
<td>1.00010</td>
</tr>
<tr>
<td>$2^{24}$</td>
<td>1.006</td>
<td>1.000000025</td>
</tr>
</tbody>
</table>
Large values of $p$

Asymptotic approximation factor

\begin{align*}
\text{ONLINE-GREEDY} & : \frac{4}{3} \quad 1 \\
\text{OFFLINE-GREEDY} & : 2 \quad 1 \quad \uparrow \\
& \quad \text{optimal}
\end{align*}
Outline

1. Revisiting the greedy algorithm for independent jobs

2. Reclaiming the slack of a schedule

3. Tri-criteria problem: execution time, reliability, energy

4. Checkpointing and energy consumption

5. Conclusion
Motivation

- Mapping of tasks is given (ordered list for each processor and dependencies between tasks)
- If deadline not tight, why not take our time?
- Slack: unused time slots

Goal: efficiently use speed scaling (DVFS)
## Speed models

<table>
<thead>
<tr>
<th>Type of speeds</th>
<th>Anytime</th>
<th>Change speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>([s_{\min}, s_{\max}]) ({s_1, \ldots, s_m})</td>
<td>CONTINUOUS</td>
<td>Beginning of tasks</td>
</tr>
<tr>
<td></td>
<td>VDD-HOPPING</td>
<td>DISCRETE, INCREMENTAL</td>
</tr>
</tbody>
</table>

- **CONTINUOUS**: great for theory
- Other ”discrete” models more realistic
- **VDD-HOPPING** simulates CONTINUOUS
- **INCREMENTAL** is a special case of DISCRETE with equally-spaced speeds: for all \(1 \leq q < m\), \(s_{q+1} - s_q = \delta\)
Tasks

- DAG: $\mathcal{G} = (V, E)$
- $n = |V|$ tasks $T_i$ of weight $w_i = \int_{t_i-d_i}^{t_i} s_i(t)dt$
- $d_i$: task duration; $t_i$: time of end of execution of $T_i$

Parameters for $T_i$ scheduled on processor $p_j$
Assume $T_i$ is executed at constant speed $s_i$

\[ d_i = E\text{xe}(w_i, s_i) = \frac{w_i}{s_i} \]

$t_j + d_i \leq t_i$ for each $(T_j, T_i) \in E$

Constraint on makespan:

$t_i \leq D$ for each $T_i \in V$
Energy to execute task $T_i$ once at speed $s_i$:

$$E_i(s_i) = d_i s_i^3 = w_i s_i^2$$

→ Dynamic part of classical energy models

**Bi-criteria problem**

- Constraint on deadline: $t_i \leq D$ for each $T_i \in V$
- Minimize energy consumption: $\sum_{i=1}^{n} w_i \times s_i^2$
Minimizing energy with fixed mapping on $p$ processors:

- **Continuous**: Polynomial for some special graphs, geometric optimization in the general case
- **Discrete**: NP-complete (reduction from 2-partition); approximation algorithm
- **Incremental**: NP-complete (reduction from 2-partition); approximation algorithm
- **VDD-Hopping**: Polynomial (linear programming)
Summary

- Results for **Continuous**, but not very practical

- In real life, **Discrete** model (DVFS)

- **Vdd-Hopping**: good alternative, mixing two consecutive modes, smoothes out the discrete nature of modes

- **Incremental**: alternate (and simpler in practice) solution, with one unique speed during task execution; can be made arbitrarily efficient
Outline

1. Revisiting the greedy algorithm for independent jobs
2. Reclaiming the slack of a schedule
3. Tri-criteria problem: execution time, reliability, energy
4. Checkpointing and energy consumption
5. Conclusion
Framework

- DAG: $\mathcal{G} = (V, E)$
- $n = |V|$ tasks $T_i$ of weight $w_i$
- $p$ identical processors fully connected
- DVFS: interval of available continuous speeds $[s_{\text{min}}, s_{\text{max}}]$ 
- One speed per task

(I will not discuss results for the \textsc{Vdd-Hopping} model)
Execution time of $T_i$ at speed $s_i$:

$$d_i = \frac{w_i}{s_i}$$

If $T_i$ is executed twice on the same processor at speeds $s_i$ and $s'_i$:

$$d_i = \frac{w_i}{s_i} + \frac{w_i}{s'_i}$$

Constraint on makespan: end of execution before deadline $D$
Reliability

- **Transient fault**: local, no impact on the rest of the system
- Reliability $R_i$ of task $T_i$ as a function of speed $s$
- Threshold reliability (and hence speed $s_{rel}$)
Re-execution: a task is re-executed \textit{on the same processor, just after its first execution}

With two executions, reliability $R_i$ of task $T_i$ is:

$$R_i = 1 - (1 - R_i(s_i))(1 - R_i(s_i'))$$

Constraint on reliability:

\textbf{Reliability:} $R_i \geq R_i(s_{rel})$, and at most one re-execution
Energy

- Energy to execute task $T_i$ once at speed $s_i$:

$$E_i(s_i) = w_i s_i^2$$

→ Dynamic part of classical energy models

- With re-executions, it is natural to take the worst-case scenario:

$$\text{ENERGY} : E_i = w_i \left( s_i^2 + s_i'^2 \right)$$
Given $G = (V, E)$

Find

- A schedule of the tasks
- A set of tasks $I = \{ i \mid T_i \text{ is executed twice} \}$
- Execution speed $s_i$ for each task $T_i$
- Re-execution speed $s'_i$ for each task in $I$

such that

$$\sum_{i \in I} w_i (s_i^2 + s'_i^2) + \sum_{i \notin I} w_i s_i^2$$

is minimized, while meeting reliability and deadline constraints
Complexity results

- **One speed** per task
- **Re-execution at same speed** as first execution, i.e., \( s_i = s'_i \)

- **Tri-Crit-Cont** is NP-hard even for a linear chain, but not known to be in NP (because of Continuous model)
- Polynomial-time solution for a fork
Energy-reducing heuristics

Two steps:
- Mapping (NP-hard) $\rightarrow$ List scheduling
- Speed scaling + re-execution (NP-hard) $\rightarrow$ Energy reducing

- The list-scheduling heuristic maps tasks onto processors at speed $s_{\text{max}}$, and we keep this mapping in step two
- Step two = slack reclamation! Use of deceleration and re-execution
Deceleration and re-execution

- **Deceleration**: select a set of tasks that we execute at speed
  \[
  \max\left(s_{rel}, s_{\max} \frac{\max_{i=1}^{n} \frac{t_i}{D}}{n}\right): \text{ slowest possible speed meeting both reliability and deadline constraints}
  \]

- **Re-execution**: greedily select tasks for re-execution
Super-weight (SW) of a task

- SW: sum of the weights of the tasks (including $T_i$) whose execution interval is included into $T_i$’s execution interval
- SW of task slowed down = estimation of the total amount of work that can be slowed down together with that task
**Selected heuristics**

- **A. SUS-Crit**: efficient on DAGs with low degree of parallelism
  - Set the speed of every task to $\max(s_{rel}, s_{max} \frac{\max_i t_i}{D})$
  - Sort the tasks of every critical path according to their SW and try to re-execute them
  - Sort all the tasks according to their **weight** and try to re-execute them

- **B. SUS-Crit-Slow**: good for highly parallel tasks: re-execute, then decelerate
  - Sort the tasks of every critical path according to their SW and try to re-execute them. If not possible, then try to slow them down
  - Sort all tasks according to their **weight** and try to re-execute them. If not possible, then try to slow them down
We compare the impact of:

- the number of processors $p$
- the ratio $D$ of the deadline over the minimum deadline $D_{\text{min}}$ (given by the list-scheduling heuristic at speed $s_{\text{max}}$)

on the output of each heuristic

Results normalized by heuristic running each task at speed $s_{\text{max}}$; the lower the better
**Results**

With increasing $p$, $D = 1.2$ (left), $D = 2.4$ (right)

- A better when number of processors is small
- B better when number of processors is large
- Superiority of B for tight deadlines: decelerates critical tasks that cannot be re-executed
Summary

- Tri-criteria energy/makespan/reliability optimization problem

- Various theoretical results

- Two-step approach for polynomial-time heuristics:
  - List-scheduling heuristic
  - Energy-reducing heuristics

- Two complementary energy-reducing heuristics for **Tri-Crit-Cont**
Outline

1. Revisiting the greedy algorithm for independent jobs
2. Reclaiming the slack of a schedule
3. Tri-criteria problem: execution time, reliability, energy
4. Checkpointing and energy consumption
5. Conclusion
Framework

- Execution of a divisible task ($W$ operations)
- Failures may occur
  - Transient faults
  - Resilience through checkpointing
- Objective: minimize expected energy given a deadline bound
- Decisions before execution:
  - Chunks: how many ($n$)? which sizes ($W_i$ for chunk $i$)?
  - Speeds of each chunk: first run ($s_i$)? re-execution ($\sigma_i$)?

<table>
<thead>
<tr>
<th>$W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 4$</td>
</tr>
<tr>
<td>$W_1$</td>
</tr>
</tbody>
</table>
Framework

- Execution of a divisible task ($W$ operations)
- Failures may occur
  - Transient faults
  - Resilience through checkpointing
- Objective: minimize expected energy given a deadline bound
- Decisions before execution:
  - Chunks: how many ($n$)? which sizes ($W_i$ for chunk $i$)?
  - Speeds of each chunk: first run ($s_i$)? re-execution ($\sigma_i$)?
Framework

- Execution of a divisible task ($W$ operations)
- Failures may occur
  - Transient faults
  - Resilience through checkpointing
- Objective: minimize expected energy given a deadline bound
- Decisions before execution:
  - Chunks: how many ($n$)? which sizes ($W_i$ for chunk $i$)?
  - Speeds of each chunk: first run ($s_i$)? re-execution ($\sigma_i$)?
Models

- Chunks

  Single chunk VS Multiple chunks

- Speed per chunk

  Single speed VS Multiple speeds

- Deadline bound

  Hard (∼ Worst-case) VS Soft (Expected)
Summary of results: single chunk

- **Single speed**
  - $s \mapsto \mathbb{E}(E)$ convex (expected energy consumption)
  - $s \mapsto \mathbb{E}(T)$ (expected execution time) and $s \mapsto T_{wc}$ (worst-case execution time) decreasing
  - $\rightarrow$ Expression of $s$ and $\mathbb{E}(E)$ (function of $\lambda, W, s, E_c, T_c$)

- **Multiple speeds**
  - Energy minimized when deadline tight
  - $\sim \sigma$ expressed as a function of $s$
  - $\rightarrow$ Minimization of single-variable function
Summary of results: multiple chunks

- Single speed
  - Equal-sized chunks, executed at same speed
  - Bound on \( s \) given \( n \)
  \[ \rightarrow \text{Minimization of double-variable function} \]

- Multiple speeds
  - Conjecture: equal-sized chunks, same first-execution / re-execution speeds
  - \( \sigma \) as a function of \( s \), bound on \( s \) given \( n \)
  \[ \rightarrow \text{Minimization of double-variable function} \]
Simulation settings

- Large set of simulations: illustrate differences between models
- **Maple** software to solve problems
- We plot relative energy consumption as a function of $\lambda$
  - The lower the better
  - Given a deadline constraint (hard or expected), normalize with the result of single-chunk single-speed
  - Impact of the constraint: normalize expected deadline with hard deadline
- Parameters varying within large ranges
Comparison with single-chunk single-speed

- Results identical for any value of $W/D$

- For expected deadline, with small $\lambda (< 10^{-2})$, using multiple chunks or multiple speeds do not improve energy ratio: re-execution term negligible; increasing $\lambda$: improvement with multiple chunks

- For hard deadline, better to run at high speed during second execution: use multiple speeds; use multiple chunks if frequent failures
Expected vs hard deadline constraint

- **Important differences for single speed models**, confirming previous conclusions: with hard deadline, use multiple speeds.

- **Multiple speeds**: no difference for small $\lambda$: re-execution at maximum speed has little impact on expected energy consumption; increasing $\lambda$: more impact of re-execution, and expected deadline may use slower re-execution speed, hence reducing energy consumption.
Outline

1. Revisiting the greedy algorithm for independent jobs
2. Reclaiming the slack of a schedule
3. Tri-criteria problem: execution time, reliability, energy
4. Checkpointing and energy consumption
5. Conclusion
Conclusion

- **ONLINE-GREEDY** and **OFFLINE-GREEDY** for power: tight approximation factor for any $p$, extends long series of papers and completely solves $N_3$ minimization problem 😊

- Different energy models, from continuous to discrete (through VDD-hopping and incremental)

- Tri-criteria heuristics with re-execution to deal with reliability

- Checkpointing techniques for reliability while minimizing energy consumption
What we had:

Energy-efficient scheduling
+ frequency scaling

What we aim at:

Energy-efficient scheduling
+ frequency scaling