Energy-efficient scheduling

Guillaume Aupy[1], Anne Benoit[1,2],
Paul Renaud-Goud[1] and Yves Robert[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 CO_2: in the top ten countries
- Exascale computers (10^{18} floating operations per second)
  - Need effort for feasibility
  - 1% of power saved $\rightsquigarrow$ 1 million dollar per year
- Lambda user
  - 1 billion personal computers
  - 500, 000, 000, 000, 000 Watts hour per year
- $\rightsquigarrow$ crucial for both environmental and economical reasons

Energy: a crucial issue

- Data centers
  - 330, 000, 00 … ore than France
  - 533, 000, 00 … countries
- Exascale compu … ons per second)
  - Need effort
  - 1% of powe … r year
- Lambda user
  - 1 billion per
  - 500, 000, 00 … ear

- $\rightsquigarrow$ crucial for both environmental and economical reasons

Power dissipation of a processor

- $P = P_{\text{leak}} + P_{\text{dyn}}$
  - P_{leak}: constant
  - $P_{\text{dyn}} = B \times V^2 \times f$

    constant   supply voltage   frequency

- Standard approximation: $P = P_{\text{leak}} + f^{\alpha}$ $(2 \leq \alpha \leq 3)$
- Energy $E = P \times \mathit{time}$
- **D**ynamic **V**oltage and **F**requency **S**caling
  - Real life: discrete speeds
  - Continuous speeds can be emulated

Outline

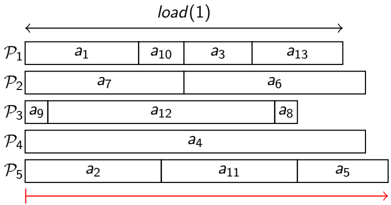
1. Revisiting the greedy algorithm for independent jobs
2. Reclaiming the slack of a schedule
3. Tri-criteria problem: execution time, reliability, energy
4. Checkpointing and energy consumption
5. Conclusion

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

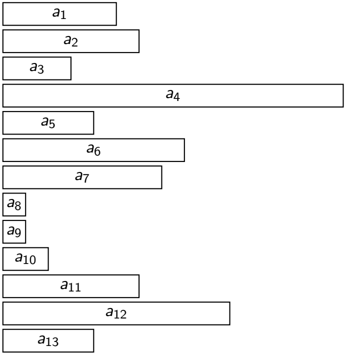
Classical problem

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



Execution time:

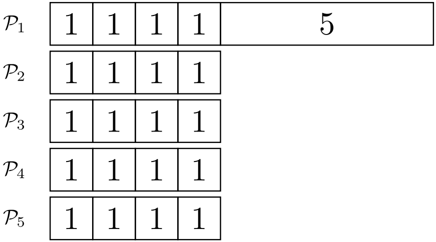
$\max_{1 \le q \le p} \quad load(q)$



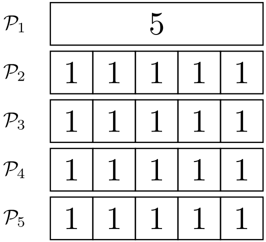
OnLine-Greedy

Theorem

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



OnLine-Greedy



Optimal solution

OffLine-Greedy

Theorem

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

$\mathcal{P}_1$	9	5	5
$\mathcal{P}_2$	9	5	
$\mathcal{P}_3$	8	6	
$\mathcal{P}_4$	8	6	
$\mathcal{P}_5$	7	7	

OffLine-Greedy

$\mathcal{P}_1$	5	5	5
$\mathcal{P}_2$	9	6	
$\mathcal{P}_3$	9	6	
$\mathcal{P}_4$	8	7	
$\mathcal{P}_5$	8	7	

Optimal solution

Bi-criteria problem

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

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

Bi-criteria problem statement

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

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

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

$\max_{1 \le q \le p} load(q)$ **Execution time**

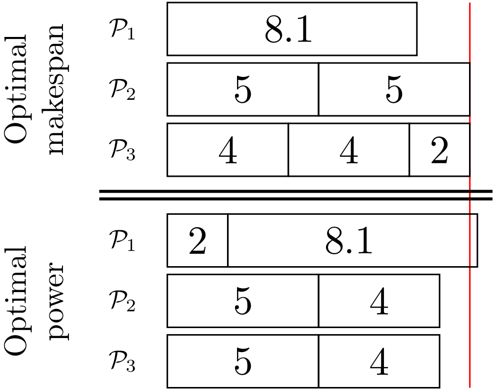
Same GREEDY algorithm . . .

- 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!



- Makespan 10, power 2531.441
- Makespan 10.1, power 2488.301

GREEDY and L_r norms

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

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

Known results

N_2, OffLine-Greedy

- Chandra and Wong 1975: upper and lower bounds
- Leung and Wei 1995: tight approximation factor

N_3, 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 $2 - \Theta(\frac{\ln r}{r})$ for 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

$$\frac{P_{\text{online}}}{P_{\text{opt}}} \leq \underbrace{\frac{\frac{1}{p^3}\left(\left(1+(p-1)\beta\right)^3+(p-1)\left(1-\beta\right)^3\right)}{\beta^3+\frac{(1-\beta)^3}{(p-1)^2}}}_{f_p^{(\text{on})}(\beta)}$$

Theorem

- *$f_p^{(\text{on})}$ has a single maximum in $\beta_p^{(\text{on})} \in [\frac{1}{p}, 1]$*
- OnLine-Greedy *is a $f_p^{(\text{on})}(\beta_p^{(\text{on})})$ approximation*
- *This approximation factor is tight*

Approximation for OffLine-Greedy

$$\frac{P_{\text{offline}}}{P_{\text{opt}}} \leq \underbrace{\frac{\frac{1}{p^3}\left(\left(1+\frac{(p-1)\beta}{3}\right)^3 + (p-1)\left(1-\frac{\beta}{3}\right)^3\right)}{\beta^3 + \frac{(1-\beta)^3}{(p-1)^2}}}_{f_p^{(\text{off})}(\beta)}$$

Theorem

- $f_p^{(\text{off})}$ *has a single maximum in* $\beta_p^{(\text{off})} \in [\frac{1}{p}, 1]$
- OffLine-Greedy *is a* $f_p^{(\text{off})}(\beta_p^{(\text{off})})$ *approximation*
- *This approximation factor is tight*

Numerical values of approximation ratios

p	ONLINE-GREEDY	OFFLINE-GREEDY
2	1.866	1.086
3	2.008	1.081
4	2.021	1.070
5	2.001	1.061
6	1.973	1.054
7	1.943	1.048
8	1.915	1.043
64	1.461	1.006
512	1.217	1.00083
2048	1.104	1.00010
2^{24}	1.006	1.000000025

Large values of p

Asymptotic approximation factor

OnLine-Greedy	$\frac{4}{3}$	1
OffLine-Greedy	2	1
		↑
		optimal

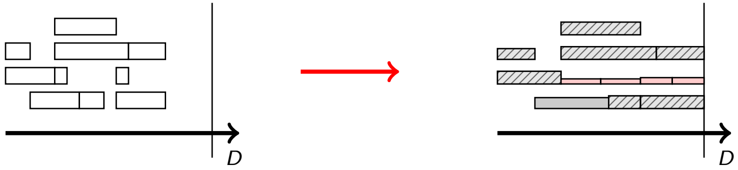
Outline

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)



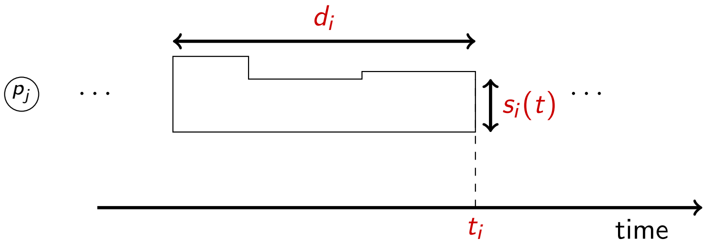
Speed models

		Change speed: Anytime	Change speed: Beginning of tasks
Type of speeds	$[s_{\min}, s_{\max}]$	CONTINUOUS	-
	$\{s_1, ..., s_m\}$	VDD-HOPPING	DISCRETE, INCREMENTAL

- 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

Makespan

Assume T_i is executed at constant speed s_i

$$d_i = \mathcal{E}xe(w_i, s_i) = \frac{w_i}{s_i}$$

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

Constraint on makespan:
$t_i \leq D$ for each $T_i \in V$

Energy

Energy to execute task T_i once at speed s_i:

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

$\rightarrow$ 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: $\sum_{i=1}^{n} w_i \times s_i^2$

Complexity results

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 VDD-HOPPING model)

Makespan

Execution time of T_i at speed s_i:

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

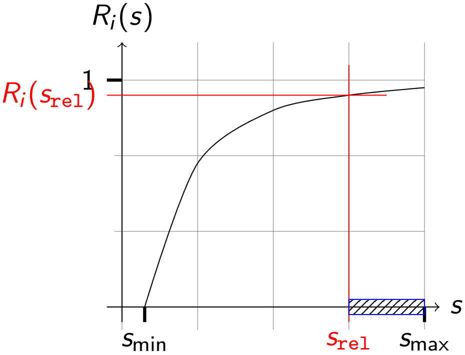
If T_i is executed twice on the same processor at speeds s_i and s_i':

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

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 $s_{\mathtt{rel}}$)



Re-execution: a task is re-executed *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:
RELIABILITY: $R_i \geq R_i(s_{\texttt{rel}})$, and at most one re-execution

Energy

- Energy to execute task T_i once at speed s_i:

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

$\rightarrow$ Dynamic part of classical energy models

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

$$\textsc{Energy} : E_i = w_i \left(s_i^2 + s_i'^2 \right)$$

Tri-Crit-Cont

Given $\mathcal{G} = (V, E)$
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) → List scheduling
- Speed scaling + re-execution (NP-hard) → Energy reducing

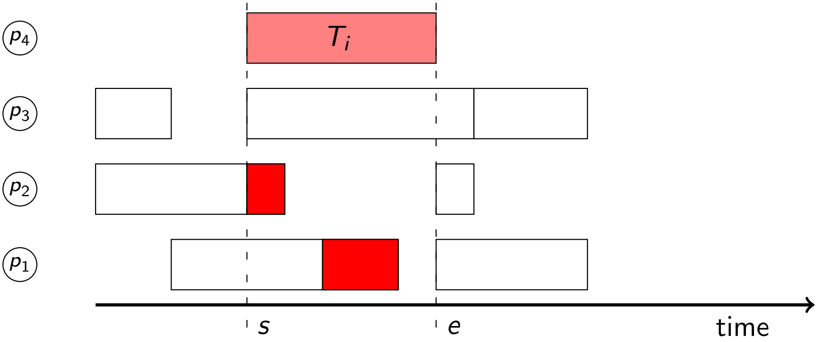
- The list-scheduling heuristic maps tasks onto processors at speed $s_{\max}$, and we keep this mapping in step two
- Step two = slack reclamation! Use of deceleration and re-execution

Deceleration and re-execution

- Deceleration: select a set of tasks that we execute at speed $\max(s_{\mathtt{rel}}, s_{\max}\frac{\max_{i=1..n} t_i}{D})$: 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 $\max(s_{\mathtt{rel}}, s_{\max}\frac{\max_{i=1..n} t_i}{D})$
  - 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

Results

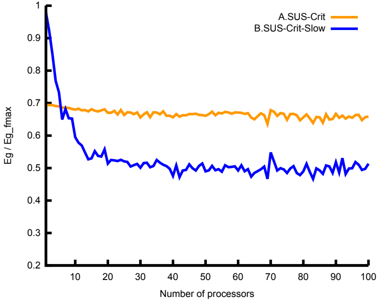
We compare the impact of:

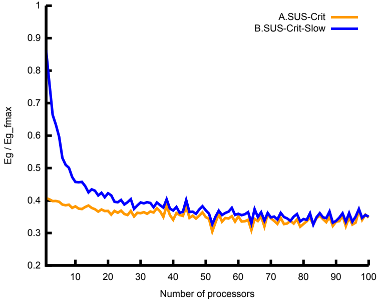
- the number of processors p
- the ratio D of the deadline over the minimum deadline D_{min} (given by the list-scheduling heuristic at speed s_{max})

on the output of each heuristic

Results normalized by heuristic running each task at speed s_{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

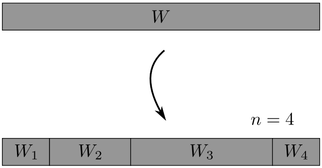
- Tri-criteria energy/makespan/reliability optimization problem
- Various theoretical results
- Two-step approach for polynomial-time heuristics:
  - List-scheduling heuristic
  - Energy-reducing heuristics
- Two complementary energy-reducing heuristics for TRI-CRIT-CONT

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

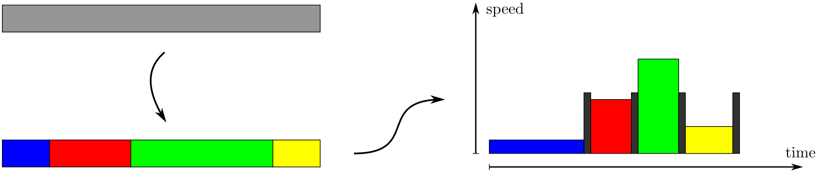
Framework

- 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 (σ_i)?



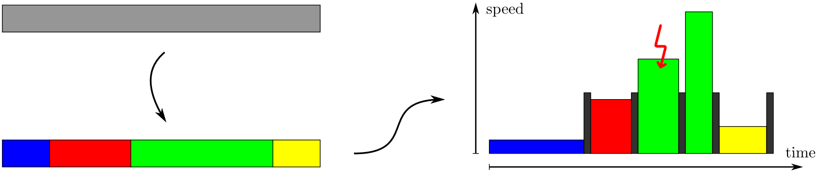
Framework

- 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 (σ_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 (σ_i)?



Models

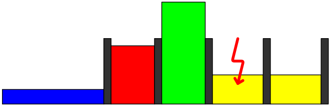
- Chunks

Single chunk

VS

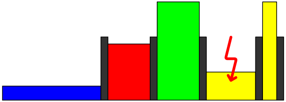
Multiple chunks

- Speed per chunk



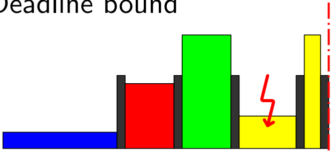
Single speed

VS



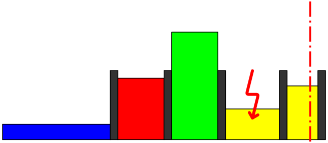
Multiple speeds

- Deadline bound



Hard ($\sim$ 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 λ, W, s, E_c, T_c)

- Multiple speeds
  - Energy minimized when deadline tight
  - $\rightsquigarrow \sigma$ 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

$\rightarrow$ Minimization of double-variable function

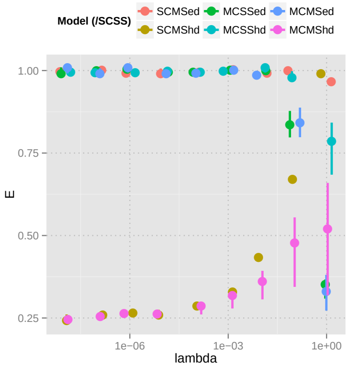
- Multiple speeds
  - Conjecture: equal-sized chunks, same first-execution / re-execution speeds
  - σ as a function of s, bound on s given n

$\rightarrow$ Minimization of double-variable function

Simulation settings

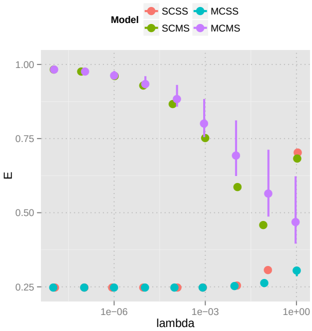
- Large set of simulations: illustrate differences between models
- Maple software to solve problems
- We plot relative energy consumption as a function of λ
  - The lower the better
  - Given a deadline constraint (hard or expected), normalize with the result of single-chunk single-speed
  - Impact 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 λ ($< 10^{-2}$), using multiple chunks or multiple speeds do not improve energy ratio: re-execution term negligible;
  increasing λ: 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 λ: re-execution at maximum speed has little impact on expected energy consumption;
  increasing λ: 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: 

What we aim at: 

Energy-efficient scheduling
+
frequency scaling