Scheduling on heterogeneous machines

Lionel Eyraud-Dubois

New Trends in Computing Summer School
August 2024

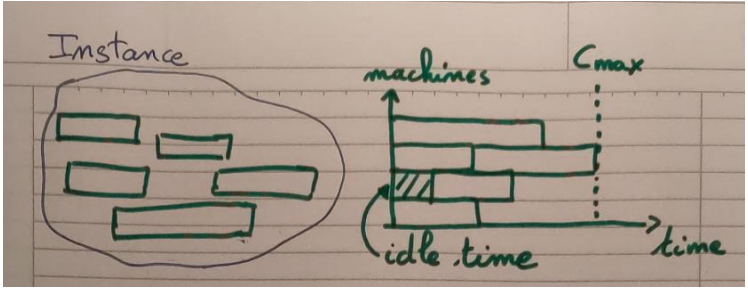
Outline

Outline

Scheduling independent tasks

Multiprocessor scheduling problem $P||C_{\max}$

- Input instance: n tasks with duration p_i, m machines
- Solution: a schedule where each task has a starting time s_i, so that each machine runs only one job at a time
- Objective: minimize the *makespan* $C_{\max} = \max_i C_i = \max_i s_i + p_i$



Scheduling independent tasks

Multiprocessor scheduling problem $P||C_{\max}$

- Input instance: n tasks with duration p_i, m machines
- Solution: a schedule where each task has a starting time s_i, so that each machine runs only one job at a time
- Objective: minimize the *makespan* $C_{\max} = \max_i C_i = \max_i s_i + p_i$

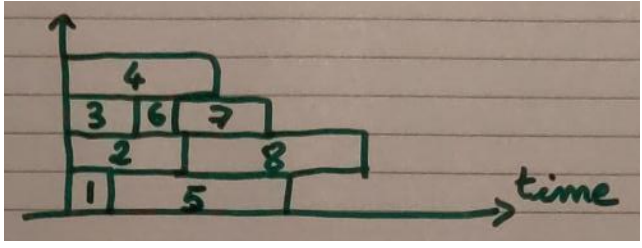
Worst-case Approximation Algorithm

- Algorithm which never gives a very bad solution
- Algorithm $\mathcal{A}$ is a ρ-approximation if:
  1. It solves the problem: $\forall$ instance $I, \mathcal{A}(I)$ is a valid solution
  2. With an approximation guarantee: $\forall$ instance $I, \forall$ solution $s, \text{cost}(\mathcal{A}(I)) \leq \rho \cdot \text{cost}(s)$

List Scheduling

List Scheduling Algorithm (Graham, 1956)

Organize tasks into a list $\mathcal{L}$, in an arbitrary order;
while $\mathcal{L}$ *is non empty* **do**
    When *a machine* k *is available*
        start the first task in $\mathcal{L}$ on machine k;
end



List is an approximation

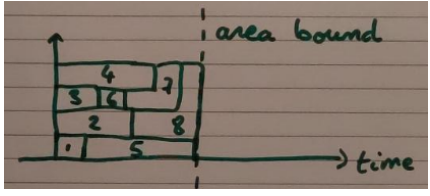
Theorem

List is a 2-approximation algorithm

Lower bounds

For any instance I with m machines and task durations p_i,

- $C^*_{\max}(I) \geq \max_i p_i$ – "unbounded resources" case
- $C^*_{\max}(I) \geq (1/m) \sum_i p_i$ – "area bound"



Approximation proof

Theorem

List is a 2-approximation algorithm

Proof:

- Let I be an instance, and $\mathcal{O}$ the schedule computed by **List**.
- Let j be the task that finishes last in $\mathcal{O}$, s_j its starting time, and $C_j = C_{\max}(\mathcal{O})$ its ending time.
- From time 0 to time s_j, *all machines are busy*
- This means $m \cdot s_j \leq \sum_i p_i$
- $C_{\max}(\mathcal{O}) = s_j + p_j \leq (1/m) \sum_i p_i + \max_i p_i$
- Hence, $C_{\max}(\mathcal{O}) \leq 2C^*_{\max}(I)$

Non-approximation proof

Theorem

For any $\rho < 2$, **List** is *not* a ρ-approximation algorithm

Non-approximation proof

Theorem

For any $\rho < 2$, **List** is *not* a ρ-approximation algorithm

For any m, we can build an instance I with m machines so that $C_{\max}^{\textbf{List}}(I) = (2 - \frac{1}{m}) C_{\max}^{*}(I)$

Non-approximation proof

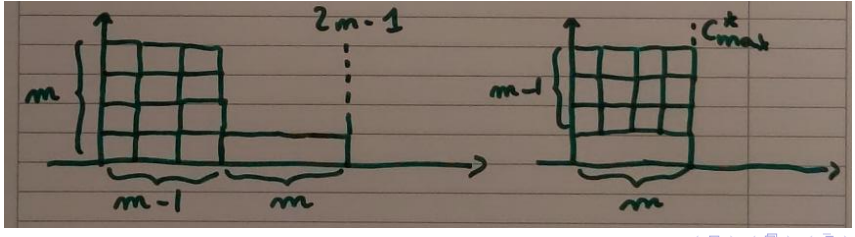
Theorem

For any $\rho < 2$, **List** is *not* a ρ-approximation algorithm

For any m, we can build an instance I with m machines so that $C_{\max}^{\mathbf{List}}(I) = (2 - \frac{1}{m}) C_{\max}^{*}(I)$

Proof:

- Let I be an instance with $m(m-1)$ tasks of duration 1, et 1 task of duration m.
- $C_{\max}^{*}(I) = m$
- Si la tâche de durée m est à la fin de $\mathcal{L}$, $C_{\max}^{\mathbf{List}} = (m-1) + m$



More precise approximation proof

Theorem

List is a $2 - \frac{1}{m}$-approximation algorithm on any instance with m machines

Proof:

- Let I be an instance, and $\mathcal{O}$ the schedule computed by **List**.
- Let j be the task that finishes last in $\mathcal{O}$, s_j its starting time, and $C_j = C_{\max}(\mathcal{O})$ its ending time.
- From time 0 to time s_j, *all machines are busy* with tasks other than j
- This means $m \cdot s_j \leq \sum_i p_i$
- $C_{\max}(\mathcal{O}) = s_j + p_j \leq (1/m) \sum_i p_i \qquad + p_j$
- Hence, $C_{\max}(\mathcal{O}) \leq 2 \qquad C^*_{\max}(I)$

More precise approximation proof

Theorem

List is a $2 - \frac{1}{m}$-approximation algorithm on any instance with m machines

Proof:

- Let I be an instance, and $\mathcal{O}$ the schedule computed by **List**.
- Let j be the task that finishes last in $\mathcal{O}$, s_j its starting time, and $C_j = C_{\max}(\mathcal{O})$ its ending time.
- From time 0 to time s_j, *all machines are busy* with tasks other than j
- This means $m \cdot s_j \leq \sum_i p_i - p_j$
- $C_{\max}(\mathcal{O}) = s_j + p_j \leq (1/m) \sum_i p_i \qquad + p_j$
- Hence, $C_{\max}(\mathcal{O}) \leq 2 \qquad C^*_{\max}(I)$

More precise approximation proof

Theorem

List is a $2 - \frac{1}{m}$-approximation algorithm on any instance with m machines

Proof:

- Let I be an instance, and $\mathcal{O}$ the schedule computed by **List**.
- Let j be the task that finishes last in $\mathcal{O}$, s_j its starting time, and $C_j = C_{\max}(\mathcal{O})$ its ending time.
- From time 0 to time s_j, *all machines are busy* with tasks other than j
- This means $m \cdot s_j \leq \sum_i p_i - p_j$
- $C_{\max}(\mathcal{O}) = s_j + p_j \leq (1/m) \sum_i p_i - p_j/m + p_j$

- Hence, $C_{\max}(\mathcal{O}) \leq 2 \qquad C^*_{\max}(I)$

More precise approximation proof

Theorem

List is a $2 - \frac{1}{m}$-approximation algorithm on any instance with m machines

Proof:

- Let I be an instance, and $\mathcal{O}$ the schedule computed by **List**.
- Let j be the task that finishes last in $\mathcal{O}$, s_j its starting time, and $C_j = C_{\max}(\mathcal{O})$ its ending time.
- From time 0 to time s_j, *all machines are busy* with tasks other than j
- This means $m \cdot s_j \leq \sum_i p_i - p_j$
- $C_{\max}(\mathcal{O}) = s_j + p_j \leq (1/m) \sum_i p_i - p_j/m + p_j$
- $C_{\max}(\mathcal{O}) \leq (1/m) \sum_i p_i + (\max_i p_i)(1 - \frac{1}{m})$
- Hence, $C_{\max}(\mathcal{O}) \leq 2 \quad C^*_{\max}(I)$

More precise approximation proof

Theorem

List is a $2 - \frac{1}{m}$-approximation algorithm on any instance with m machines

Proof:

- Let I be an instance, and $\mathcal{O}$ the schedule computed by **List**.
- Let j be the task that finishes last in $\mathcal{O}$, s_j its starting time, and $C_j = C_{\max}(\mathcal{O})$ its ending time.
- From time 0 to time s_j, *all machines are busy* with tasks other than j
- This means $m \cdot s_j \leq \sum_i p_i - p_j$
- $C_{\max}(\mathcal{O}) = s_j + p_j \leq (1/m) \sum_i p_i - p_j/m + p_j$
- $C_{\max}(\mathcal{O}) \leq (1/m) \sum_i p_i + (\max_i p_i)(1 - \frac{1}{m})$
- Hence, $C_{\max}(\mathcal{O}) \leq (2 - \frac{1}{m}) C^*_{\max}(I)$

Outline

Interlude: Graham three field notation for scheduling problems

Three field notation: $\alpha \,|\, \beta \,|\, \gamma$

- α: specifies the architecture
- β: describes the tasks
- γ: describes the objective function to be optimized

Interlude: Graham three field notation for scheduling problems

Three field notation: $\alpha \,|\, \beta \,|\, \gamma$

- α: specifies the architecture
  - 1 for single machine
  - P for parallel (identical) machines, $P2$ for exactly 2 processors
  - Q for parallel machines with different speeds (*related*)
  - R when task i on machine j has arbitrary duration p_{ij} (*unrelated*)
  - and other more specific cases (flow-shop, open-shop, ...) related to production environments
- β: describes the tasks
- γ: describes the objective function to be optimized

Interlude: Graham three field notation for scheduling problems

Three field notation: $\alpha \,|\, \beta \,|\, \gamma$

- α: specifies the architecture
- β: describes the tasks
  - for "default" behavior (non-preemptive, fixed durations, ...)
  - *prec* for precedence constraints
  - *pmtn* for tasks that can be paused and resumed
  - $p_j = 1$ for tasks with Unitary Execution Time
  - d_j for tasks with due dates, r_j for release times
  - $size_j$ for tasks which require several machines
- γ: describes the objective function to be optimized

Interlude: Graham three field notation for scheduling problems

Three field notation: $\alpha \,|\, \beta \,|\, \gamma$

- α: specifies the architecture
- β: describes the tasks
- γ: describes the objective function to be optimized
  - Makespan $C_{\max}$
  - Sum of completion times $\sum C_i$, weighted version $\sum w_i C_i$
  - Lateness $L_i = C_i - d_i$, tardiness $T_i = \max(C_i - d_i, 0)$, earliness $E_i = \max(0, d_j - C_j)$
  - Throughput $U_j = 1$ if $C_j \leq d_j$, 0 otherwise

Interlude: Graham three field notation for scheduling problems

Three field notation: $\alpha \,|\, \beta \,|\, \gamma$

- α: specifies the architecture
- β: describes the tasks
- γ: describes the objective function to be optimized

Examples

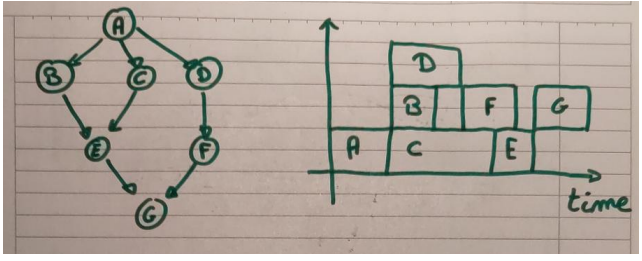
- $1||\sum w_j C_j$: minimize sum of weighted completion times on one machine
- $1|prec|L_{\max}$: minimize maximum lateness with precedence constraints
- $P||C_{\max}$: multiprocessor scheduling
- $Q2|r_j|\sum C_i$: 2 related processors, minimize average completion time with release dates

Outline

Scheduling with precedence constraints

Multiprocessor scheduling problem with precedence $P|\text{prec}|C_{\max}$

- Input instance: n tasks with duration p_i, with a precedence relationship $i \rightsquigarrow j$
  m identical machines
- Solution: a schedule where each task has a starting time s_i, so that
  - each machine runs only one job at a time
  - $C_i \leq s_j$ for any precedence constraint $i \rightsquigarrow j$
- Objective: minimize the *makespan* $C_{\max} = \max_i C_i = \max_i s_i + p_i$

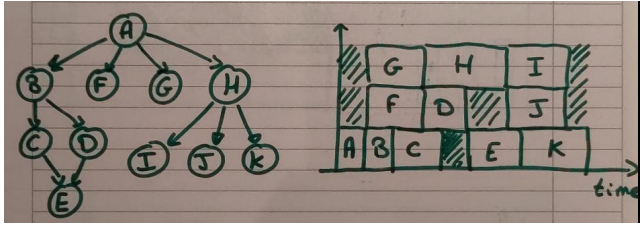


List Scheduling with precedence constraints

List Scheduling

Organize tasks into a list $\mathcal{L}$, in an arbitrary order;
while $\mathcal{L}$ *is non empty* **do**
  When *a machine* k *is available*
    start the first ready task in $\mathcal{L}$ on machine k;
end

A task j is ready if all its predecessors are completed



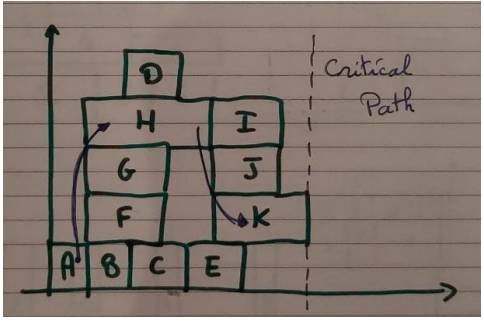
Approximation proof with precedence constraints

Lower bounds

For any instance I with m machines and task durations p_i,

- $C^*_{\max}(I) \geq (1/m) \sum_i p_i$ – same "area bound" as before
- $C^*_{\max}(I) \geq \max_{\mathcal{P} \text{ path in } \rightsquigarrow} \sum_{i \in \mathcal{P}} p_i$ – "unbounded resources" case

The second bound is called the critical path CP.



Approximation proof with precedence constraints

Theorem

List is a 2-approximation algorithm for $P|\text{prec}|C_{\max}$

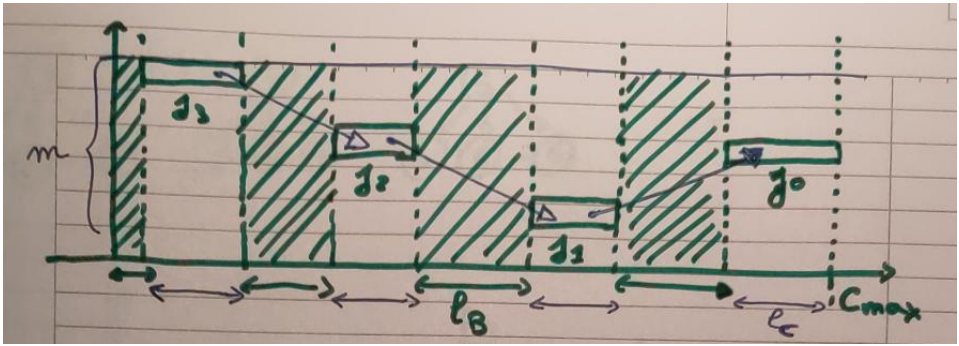
Proof:

- Let I be an instance, and $\mathcal{O}$ the schedule computed by **List**.
- Let j_0 be the task that finishes last in $\mathcal{O}$. Consider j_1, the predecessor of j_0 that finishes last in $\mathcal{O}$. Continue: j_{k+1} is the predecessor of j_k that finishes last
- From time C_{j_1} to time s_{j_0}, *all machines are busy*
- Partition the time into *busy* intervals (from $C_{j_{k+1}}$ to s_{j_k}) and *critical* intervals (from s_{j_k} to C_{j_k})
- The total length ℓ_B of busy intervals satisfy $m \cdot \ell_B \leq \sum_i p_i$
- $j_K \rightsquigarrow j_{K-1} \rightsquigarrow \cdots \rightsquigarrow j_1 \rightsquigarrow j_0$ is a path in the precedence graph
- The total length ℓ_C of critical intervals satisfy $\ell_C \leq CP$
- $C_{\max}(\mathcal{O}) = \ell_B + \ell_C \leq (1/m)\sum_i p_i + CP$
- Hence, $C_{\max}(\mathcal{O}) \leq 2C^*_{\max}(I)$

Approximation proof

Theorem

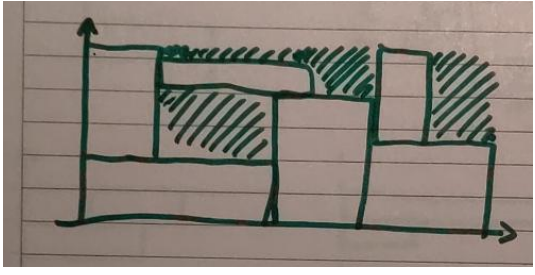
List is a 2-approximation algorithm for $P|\text{prec}|C_{\max}$



List Scheduling with parallel tasks

Parallel tasks scheduling problem $P|\text{size}_j|C_{\max}$

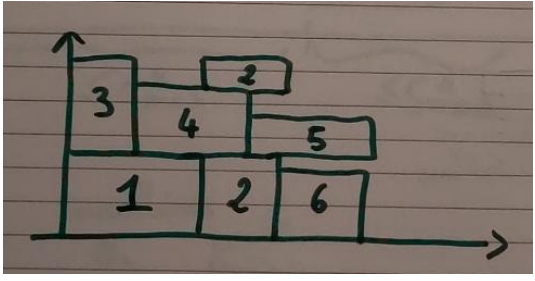
- Input instance: n tasks with duration p_i requiring q_i machines
  m identical machines
- Solution: a schedule where each task has a starting time s_i, so that each machine runs only one job at a time
- Objective: minimize the *makespan* $C_{\max} = \max_i C_i = \max_i s_i + p_i$



List Scheduling with parallel tasks

List Scheduling

Organize tasks into a list $\mathcal{L}$, in an arbitrary order;
while $\mathcal{L}$ *is non empty* **do**
  When $k \geq 1$ *machines are available*
    start the first task in $\mathcal{L}$ with $q_i \leq k$;
end



Approximation proof with parallel tasks

Theorem

List is a 2-approximation algorithm for $P|\text{size}_j|C_{\max}$. More precisely:

$$C_{\max}^{\textbf{List}}(I) \leq 2\max(p_{\max}, \tfrac{1}{m}\sum_i p_i)$$

Proof:

- Let I be an instance, and $\mathcal{O}$ the schedule computed by **List**. Set $C = C_{\max}(\mathcal{O})$.
- Denote with $r(t)$ the number of machines busy at time t in $\mathcal{O}$
- $\forall t, t' < C$ with $t' \geq t + p_{\max}, r(t) + r(t') > m$

Approximation proof with parallel tasks

Theorem

List is a 2-approximation algorithm for $P|\text{size}_j|C_{\max}$. More precisely:

$$C_{\max}^{\textbf{List}}(I) \leq 2\max(p_{\max}, \tfrac{1}{m}\textstyle\sum_i p_i)$$

Proof:

- Let I be an instance, and $\mathcal{O}$ the schedule computed by **List**. Set $C = C_{\max}(\mathcal{O})$.
- Denote with $r(t)$ the number of machines busy at time t in $\mathcal{O}$
- $\forall t, t' < C$ with $t' \geq t + p_{\max}, r(t) + r(t') > m$
- Assume $C > 2p_{\max}$. Then $p_{\max} < \frac{C}{2}$, so that $\forall t < C/2, r(t) + r(t + C/2) > m$

Approximation proof with parallel tasks

Theorem

List is a 2-approximation algorithm for $P|\text{size}_j|C_{\max}$. More precisely:

$$C_{\max}^{\textbf{List}}(I) \leq 2\max(p_{\max}, \tfrac{1}{m}\textstyle\sum_i p_i)$$

Proof:

- Let I be an instance, and $\mathcal{O}$ the schedule computed by **List**. Set $C = C_{\max}(\mathcal{O})$.
- Denote with $r(t)$ the number of machines busy at time t in $\mathcal{O}$
- $\forall t, t' < C$ with $t' \geq t + p_{\max}, r(t) + r(t') > m$
- Assume $C > 2p_{\max}$. Then $p_{\max} < \frac{C}{2}$, so that $\forall t < C/2, r(t) + r(t + C/2) > m$
- Integrate: $\displaystyle\int_0^{\frac{C}{2}} r(t)dt + \int_{\frac{C}{2}}^{C} r(t)dt > m \cdot \frac{C}{2}$

Approximation proof with parallel tasks

Theorem

List is a 2-approximation algorithm for $P|\text{size}_j|C_{\max}$. More precisely:

$$C_{\max}^{\textbf{List}}(I) \leq 2\max(p_{\max}, \tfrac{1}{m}\textstyle\sum_i p_i)$$

Proof:

- Let I be an instance, and $\mathcal{O}$ the schedule computed by **List**. Set $C = C_{\max}(\mathcal{O})$.
- Denote with $r(t)$ the number of machines busy at time t in $\mathcal{O}$
- $\forall t, t' < C$ with $t' \geq t + p_{\max}, r(t) + r(t') > m$
- Assume $C > 2p_{\max}$. Then $p_{\max} < \frac{C}{2}$, so that $\forall t < C/2, r(t) + r(t + C/2) > m$
- Integrate: $\underbrace{\int_0^{\frac{C}{2}} r(t)dt + \int_{\frac{C}{2}}^{C} r(t)dt}_{\sum_i p_i} > m \cdot \frac{C}{2}$

Approximation proof with parallel tasks

Theorem

List is a 2-approximation algorithm for $P|\text{size}_j|C_{\max}$. More precisely:

$$C_{\max}^{\textbf{List}}(I) \leq 2\max(p_{\max}, \tfrac{1}{m}\textstyle\sum_i p_i)$$

Proof:

- Let I be an instance, and $\mathcal{O}$ the schedule computed by **List**. Set $C = C_{\max}(\mathcal{O})$.
- Denote with $r(t)$ the number of machines busy at time t in $\mathcal{O}$
- $\forall t, t' < C$ with $t' \geq t + p_{\max}, r(t) + r(t') > m$
- Assume $C > 2p_{\max}$. Then $p_{\max} < \frac{C}{2}$, so that $\forall t < C/2, r(t) + r(t + C/2) > m$
- Integrate: $\underbrace{\int_0^{\frac{C}{2}} r(t)dt + \int_{\frac{C}{2}}^{C} r(t)dt}_{\sum_i p_i} > m \cdot \frac{C}{2}$
- Conclusion: $\sum_i p_i > \frac{mC}{2}$, so $C < 2 \cdot \frac{\sum_i p_i}{m}$

Comments on the previous results

- More general result in [Garey,Graham 1975]: **List** is an $s+1$-approximation algorithm with s different resources.
- Can also be refined to $2-\frac{1}{m}$ with more careful proof
- Previously $C \leq \text{area} + \text{CP}$, this one is weaker: $C \leq 2 \cdot \max(\text{area}, \text{CP})$
- This is related to the Work Stealing results

Work Stealing

- Work Stealing was made popular with the `Cilk` runtime
- Each thread can *spawn* new threads, and *wait* for completion of spawned threads
- Each worker has a *queue* of threads, process them in order
- Idle workers can *steal* available threads from the queue of other workers

Performance guarantee

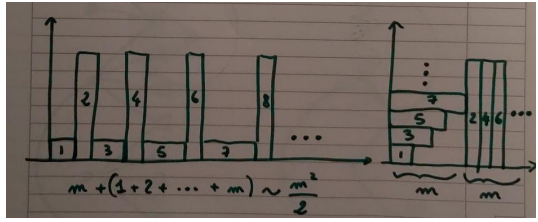
With the right implementation, Work Stealing on a program with area T_1 and critical path T_∞ can achieve a running time of

$$\frac{T_1}{P} + \mathcal{O}(T_\infty)$$

Warning: Wrong List Scheduling implementation

Task-centric List Scheduling

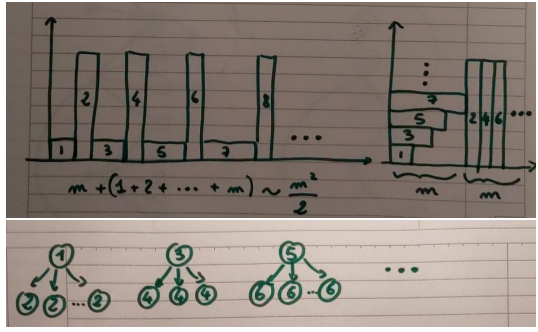
Organize tasks into a list $\mathcal{L}$, in an arbitrary order;
while $\mathcal{L}$ *is non empty* **do**
    Pick the first task j in $\mathcal{L}$;
    Schedule j at the earliest possible time s_j;
end



Warning: Wrong List Scheduling implementation

Task-centric List Scheduling

Organize tasks into a list $\mathcal{L}$, in an arbitrary order;
while $\mathcal{L}$ *is non empty* **do**
    Pick the first task j in $\mathcal{L}$;
    Schedule j at the earliest possible time s_j;
end



Outline

Outline

Heterogeneous Earliest Finish Time – $R|\text{prec}, \text{comm}|C_{\max}$

HEFT is a very well known heuristic for this problem. It has two phases:

- Compute a ranking of the tasks (priorities):

$$r_i = \overline{p_i} + \max_{k \mid i \rightsquigarrow k} \left(\overline{c_{i,k}} + r_k \right)$$

- Consider tasks by highest priority first
- Assign each task to the machine where it finishes earliest

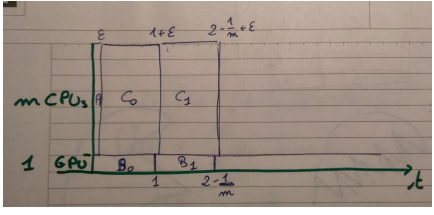
Heterogeneous Earliest Finish Time – $R|\text{prec}, \text{comm}|C_{\max}$

HEFT is not an approximation algorithm, even for $R||C_{\max}$ [Bleuse, Monna 2015]

Instance with m CPUs and 1 GPU:

Type	Number	p_{CPU}	p_{GPU}	rank
A	m	ϵ	$m+1$	$(\epsilon m + m + 1)/(m+1)$
B_i, $i = 0 \dots m-1$	1	$1 - i/m$	$1 - i/m$	$1 - i/m$
C_i, $i = 0 \dots m-1$	m	$1 - i/m$	$1/m^2$	$(m - i + 1/m^2)/(m+1)$

- $r_A > r_{B_0} > r_{C_0} > r_{B_1} > r_{C_1} > \cdots$
- $C_{\max}^{\text{HEFT}} = \frac{m}{2} + \frac{3}{2} - \frac{1}{m}$, whereas $C_{\max}^{*} = 1$



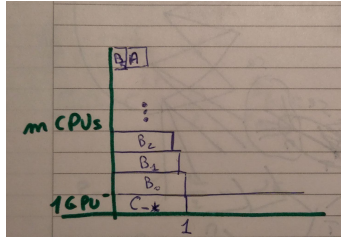
Heterogeneous Earliest Finish Time – $R|\text{prec}, \text{comm}|C_{\max}$

HEFT is not an approximation algorithm, even for $R||C_{\max}$ [Bleuse, Monna 2015]

Instance with m CPUs and 1 GPU:

Type	Number	p_{CPU}	p_{GPU}	rank
A	m	ϵ	$m+1$	$(\epsilon m + m + 1)/(m+1)$
B_i, $i = 0 \dots m-1$	1	$1 - i/m$	$1 - i/m$	$1 - i/m$
C_i, $i = 0 \dots m-1$	m	$1 - i/m$	$1/m^2$	$(m - i + 1/m^2)/(m+1)$

- $r_A > r_{B_0} > r_{C_0} > r_{B_1} > r_{C_1} > \cdots$
- $C_{\max}^{\text{HEFT}} = \frac{m}{2} + \frac{3}{2} - \frac{1}{m}$, whereas $C_{\max}^{*} = 1$



Outline

Scheduling on heterogeneous machines

Scheduling on unrelated machines $R||C_{\max}$

- Input: n tasks, m machines, processing times $p_{i,j}$
- Solution: a schedule where each task has a starting time s_i on a machine σ_i, so that each machine runs only one job at a time
- Objective: minimize makespan $C_{\max}$

Scheduling on heterogeneous machines

Scheduling on unrelated machines $R||C_{\max}$

- Input: n tasks, m machines, processing times $p_{i,j}$
- Solution: a schedule where each task has a starting time s_i on a machine σ_i, so that each machine runs only one job at a time
- Objective: minimize makespan $C_{\max}$

Remarks

- How to generalize the notion of "area bound"?
- More relevant to view it as an assignment problem

Scheduling on heterogeneous machines

Scheduling on unrelated machines – assignment version $R||C_{\max}$

- Input: n tasks, m machines, processing times $p_{i,j}$
- Solution: for each machine j, a set S_j of tasks assigned to machine j
- Objective: minimize the highest load, $C_{\max} = \max_{j=1}^{m} \left(\sum_{i \in S_j} p_{i,j} \right)$

Scheduling on heterogeneous machines

Scheduling on unrelated machines – assignment version $R||C_{\max}$

- Input: n tasks, m machines, processing times $p_{i,j}$
- Solution: for each machine j, a set S_j of tasks assigned to machine j
- Objective: minimize the highest load, $C_{\max} = \max_{j=1}^{m} \left(\sum_{i \in S_j} p_{i,j} \right)$

Linear Programming Formulation

$x_{i,j} = 1$ if task i assigned to machine j, 0 otherwise

$$\begin{array}{rrl} & \text{minimize} & C \quad \text{s.t.} \\ \forall & 1 \leq i \leq n, & \sum_{j=1}^{m} x_{i,j} = 1 \\ \forall & 1 \leq j \leq m, & \sum_{i=1}^{n} x_{i,j} p_{i,j} \leq C \\ & \forall \; i, \; j, & x_{i,j} \in \{0,1\} \end{array}$$

Scheduling on heterogeneous machines

Scheduling on unrelated machines – assignment version $R||C_{\max}$

- Input: n tasks, m machines, processing times $p_{i,j}$
- Solution: for each machine j, a set S_j of tasks assigned to machine j
- Objective: minimize the highest load, $C_{\max} = \max_{j=1}^{m} \left(\sum_{i \in S_j} p_{i,j} \right)$

Linear Programming Formulation (relaxed)

$x_{i,j}$ = fraction of task i assigned to machine j

$$\begin{array}{rrl} & \text{minimize} & C \quad \text{s.t.} \\ \forall & 1 \leq i \leq n, & \sum_{j=1}^{m} x_{i,j} = 1 \\ \forall & 1 \leq j \leq m, & \sum_{i=1}^{n} x_{i,j} p_{i,j} \leq C \\ & \forall \; i, \; j, & x_{i,j} \in [0,1] \end{array}$$

Scheduling on heterogeneous machines

Scheduling on unrelated machines – assignment version $R||C_{\max}$

- Input: n tasks, m machines, processing times $p_{i,j}$
- Solution: for each machine j, a set S_j of tasks assigned to machine j
- Objective: minimize the highest load, $C_{\max} = \max_{j=1}^{m} \left(\sum_{i \in S_j} p_{i,j} \right)$

Linear Programming Formulation (relaxed)

$x_{i,j}$ = fraction of task i assigned to machine j

$$\begin{array}{rrl} & \text{minimize} & C \quad \text{s.t.} \\ \forall & 1 \leq i \leq n, & \sum_{j=1}^{m} x_{i,j} = 1 \\ \forall & 1 \leq j \leq m, & \sum_{i=1}^{n} x_{i,j} p_{i,j} \leq C \\ & \forall \; i, \; j, & x_{i,j} \in [0,1] \end{array}$$

The optimal solution of the relaxed problem is a lower bound

$\Rightarrow$ generalization of "area bound"

Scheduling on heterogeneous machines

Scheduling on unrelated machines – assignment version $R||C_{\max}$

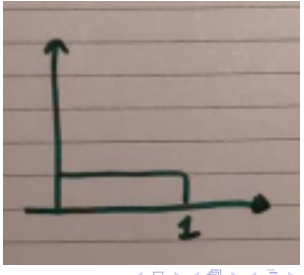
- Input: n tasks, m machines, processing times $p_{i,j}$
- Solution: for each machine j, a set S_j of tasks assigned to machine j
- Objective: minimize the highest load, $C_{\max} = \max_{j=1}^{m} \left(\sum_{i \in S_j} p_{i,j} \right)$

Linear Programming Formulation (relaxed)

$x_{i,j}$ = fraction of task i assigned to machine j

$$\begin{array}{rrl} & \text{minimize} & C \quad \text{s.t.} \\ \forall & 1 \leq i \leq n, & \sum_{j=1}^{m} x_{i,j} = 1 \\ \forall & 1 \leq j \leq m, & \sum_{i=1}^{n} x_{i,j} p_{i,j} \leq C \\ & \forall \ i, \ j, & x_{i,j} \in [0,1] \end{array}$$

But this lower bound may not be very good



Scheduling on heterogeneous machines

Scheduling on unrelated machines – assignment version $R||C_{\max}$

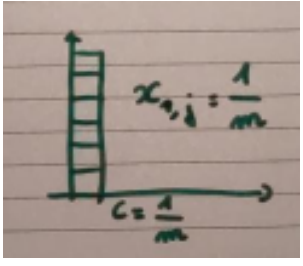
- Input: n tasks, m machines, processing times $p_{i,j}$
- Solution: for each machine j, a set S_j of tasks assigned to machine j
- Objective: minimize the highest load, $C_{\max} = \max_{j=1}^{m} \left(\sum_{i \in S_j} p_{i,j} \right)$

Linear Programming Formulation (relaxed)

$x_{i,j}$ = fraction of task i assigned to machine j

$$\begin{array}{rrl} & \text{minimize} & C \quad \text{s.t.} \\ \forall & 1 \leq i \leq n, & \sum_{j=1}^{m} x_{i,j} = 1 \\ \forall & 1 \leq j \leq m, & \sum_{i=1}^{n} x_{i,j} p_{i,j} \leq C \\ & \forall \; i, \; j, & x_{i,j} \in [0,1] \end{array}$$

But this lower bound may not be very good



Scheduling on heterogeneous machines

Linear Programming Formulation for fixed T (relaxed)

Denote $I_T = \{(i,j) \mid p_{i,j} \leq T\}$. Then (LP_T) is:

$$\begin{aligned} \forall \quad 1 \leq i \leq n, \quad & \textstyle\sum_{j \mid (i,j) \in I_T} x_{i,j} \geq 1 \\ \forall \quad 1 \leq j \leq m, \quad & \textstyle\sum_{i \mid (i,j) \in I_T} x_{i,j} p_{i,j} \leq T \\ \forall (i,j) \in I_T, \quad & x_{i,j} \in \{0,1\} \end{aligned}$$

Scheduling on heterogeneous machines

Linear Programming Formulation for fixed T

Denote $I_T = \{(i,j) \mid p_{i,j} \leq T\}$. Then (LP_T) is:

$$\begin{aligned} \forall \quad 1 \leq i \leq n, \quad & \textstyle\sum_{j \mid (i,j) \in I_T} x_{i,j} \geq 1 \\ \forall \quad 1 \leq j \leq m, \quad & \textstyle\sum_{i \mid (i,j) \in I_T} x_{i,j} p_{i,j} \leq T \\ \forall (i,j) \in I_T, \quad & x_{i,j} \geq 0 \end{aligned}$$

Clearly : $\quad T \geq C^*_{\max} \Rightarrow (LP_T)$ is feasible

We will design an algorithm S so that:

$$(x) \text{ solution of } (LP_T) \Rightarrow C_{\max}(S(x)) \leq 2T$$

Scheduling on heterogeneous machines: approximation

$T \geq C^*_{\max} \Rightarrow (LP_T)$ is feasible $\qquad$ (x) solution of $(LP_T) \Rightarrow C_{\max}(S(x)) \leq 2T$

Find $C^*_{\max}$ with dichotomic search

$L \leftarrow 1, U \leftarrow \min_j \sum_i p_{i,j}$;
while $U - L > 1$ **do**
  $C \leftarrow (L + U)/2$;
  if (LP_C) *feasible* **then** $U \leftarrow C$;
  else $L \leftarrow C$;
end
(x^*) solution of (LP_U);
return $S(x^*)$;

This is a polynomial-time algorithm:

- $\log(\min_j \sum_i p_{i,j})$ iterations
- One iteration is polynomial
- $S(x^*)$ is computed in polynomial time

Scheduling on heterogeneous machines: approximation

$T \geq C^*_{\max} \Rightarrow (LP_T)$ is feasible $\qquad$ (x) solution of $(LP_T) \Rightarrow C_{\max}(S(x)) \leq 2T$

Find $C^*_{\max}$ with dichotomic search

$L \leftarrow 1, U \leftarrow \min_j \sum_i p_{i,j}$;
while $U - L > 1$ **do**
  $C \leftarrow (L + U)/2$;
  if (LP_C) *feasible* **then** $U \leftarrow C$;
  else $L \leftarrow C$;
end
(x^*) solution of (LP_U);
return $S(x^*)$;

It is a 2-approximation algorithm:

- U is the smallest value so that (LP_U) is feasible: (LP_{U-1}) is not feasible
- So $U - 1 < C^*_{\max}$: $U \leq C^*_{\max}$
- Hence $C_{\max}(S(x^*)) \leq 2C^*_{\max}$

Scheduling on heterogeneous machines: how to build $S(x)$?

Linear Formulation (LP_T)

$$\forall \quad 1 \leq i \leq n, \quad \sum_{j \,|\, (i,j) \in I_T} x_{i,j} \geq 1$$
$$\forall \quad 1 \leq j \leq m, \quad \sum_{i \,|\, (i,j) \in I_T} x_{i,j} p_{i,j} \leq T$$
$$\forall (i,j) \in I_T, \quad x_{i,j} \geq 0$$

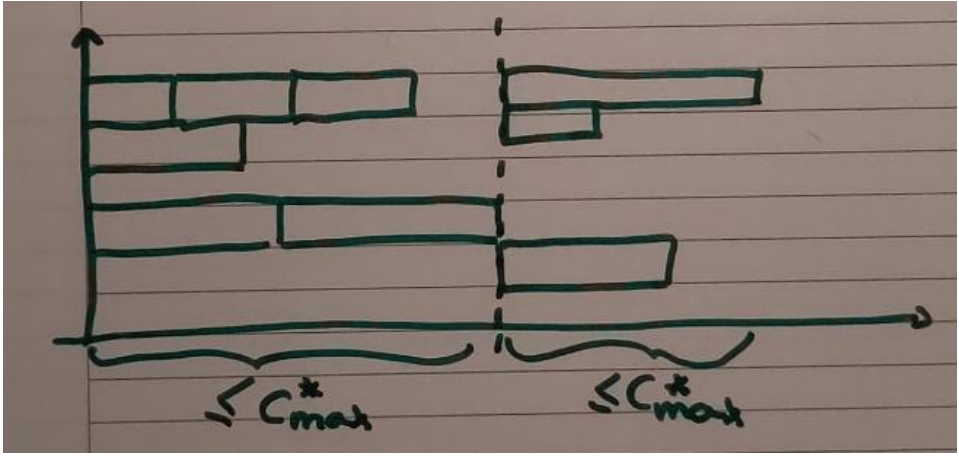
- (LP_T) has $v = \text{card}(I_T)$ variables and $n + m + v$ constraints
- There exists an optimal solution x^* with v saturated constraints
- Which means in x^*, at most $n + m$ variables $x_{i,j}$ are non-zero

- Consider graph $G = (V, E)$ with $V = \{\text{machines}\} \cup \{\text{tâches}\}$, and $E = \{(i,j) | x_{i,j} > 0\}$
- This graph has $n + m$ vertices, $\leq n + m$ edges: at most one cycle

Scheduling on heterogeneous machines: how to build $S(x)$?

Graph G with $n + m$ vertices, $\leq n + m$ edges: at most one cycle

Objective: each machine gets its "fully assigned" tasks, plus at most one extra task



Scheduling on heterogeneous machines: how to build $S(x)$?

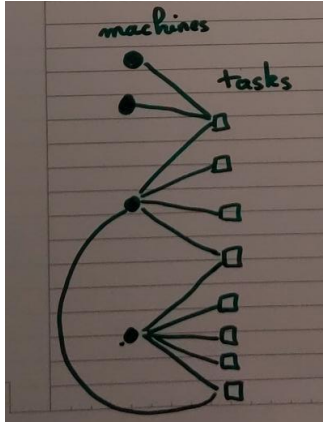
Graph G with $n + m$ vertices, $\leq n + m$ edges: at most one cycle

1. Fix the *leaf* tasks
2. Find a matching in the remaining graph

Scheduling on heterogeneous machines: how to build $S(x)$?

Graph G with $n + m$ vertices, $\leq n + m$ edges: at most one cycle

1. Fix the *leaf* tasks



Scheduling on heterogeneous machines: how to build $S(x)$?

Graph G with $n+m$ vertices, $\leq n+m$ edges: at most one cycle

1. Fix the *leaf* tasks

Let i be a task of degree 1 in G. There is only one machine j with $x_{i,j} > 0$, so $x_{i,j} = 1$. We build $S_j^{(1)} = \{i \,|\, x_{i,j} = 1\}$. Then

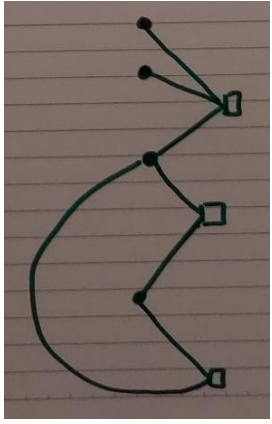
$$\forall j, \sum_{i \in S_j^{(1)}} p_{i,j} \leq T$$

Remove these tasks from G, and go to step 2

Scheduling on heterogeneous machines: how to build $S(x)$?

Graph G with $n + m$ vertices, $\leq n + m$ edges: at most one cycle

1. Fix the *leaf* tasks $\forall j, \sum_{i \in S_j^{(1)}} p_{i,j} \leq T$
2. Find a matching in the remaining graph



Scheduling on heterogeneous machines: how to build $S(x)$?

Graph G with $n+m$ vertices, $\leq n+m$ edges: at most one cycle

1. Fix the *leaf* tasks
2. Find a matching in the remaining graph

$\forall j, \sum_{i \in S_j^{(1)}} p_{i,j} \leq T$

while *there is a machine* j *with degree* 1 *in* G **do**
  $S_j^{(2)} \leftarrow S_j^{(1)} \cup \{\text{task } i \text{ connected to } j\}$;
  Remove i and j from G;
end
Let M be a matching of G;
$\forall (i,j) \in M, S_j^{(2)} \leftarrow S_j^{(1)} \cup \{i\}$;

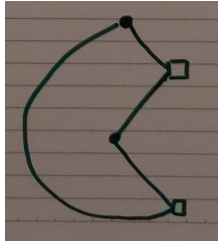
Scheduling on heterogeneous machines: how to build $S(x)$?

Graph G with $n+m$ vertices, $\leq n+m$ edges: at most one cycle

1. Fix the *leaf* tasks
2. Find a matching in the remaining graph

$\forall j, \sum_{i \in S_j^{(1)}} p_{i,j} \leq T$

while *there is a machine* j *with degree* 1 *in* G **do**
  $S_j^{(2)} \leftarrow S_j^{(1)} \cup \{\text{task } i \text{ connected to } j\}$;
  Remove i and j from G;
end
Let M be a matching of G;
$\forall (i,j) \in M, S_j^{(2)} \leftarrow S_j^{(1)} \cup \{i\}$;



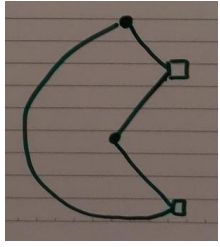
Scheduling on heterogeneous machines: how to build $S(x)$?

Graph G with $n+m$ vertices, $\leq n+m$ edges: at most one cycle

1. Fix the *leaf* tasks
2. Find a matching in the remaining graph

$\forall j, \sum_{i \in S_j^{(1)}} p_{i,j} \leq T$

while *there is a machine* j *with degree* 1 *in* G **do**
  $S_j^{(2)} \leftarrow S_j^{(1)} \cup \{\text{task } i \text{ connected to } j\}$;
  Remove i and j from G;
end
Let M be a matching of G;
$\forall (i,j) \in M, S_j^{(2)} \leftarrow S_j^{(1)} \cup \{i\}$;



- At the end, no machine remains with degree 1: G is empty or a cycle
- At most one task i added to each machine j, with $(i,j) \in I_T$
- So $\forall j, \sum_{i \in S_j^{(2)}} p_{i,j} \leq 2T$

Outline

Scheduling with two types of resources – $Pm, Pk||C_{\max}$

- Special case of $R||C_{\max}$
- Meaningful in practice for typical hardware: m CPUs, k GPUs
- Design specific efficient approximation algorithms

Notation

For task i, p_i^+ is the CPU time, p_i^- is the GPU time.

Acceleration ratio $\rho_i = \frac{p_i^+}{p_i^-}$

Two types of resources: area bound

Special case of area bound

$x_i^+ = 1$ if task i is assigned to CPU

$$\begin{aligned} \text{minimize} \quad & C \quad \text{s.t.} \\ & \textstyle\sum_i x_i^+ p_i^+ \leq m \cdot C \\ & \textstyle\sum_i (1 - x_i^+) p_i^- \leq k \cdot C \\ \forall \ i, \quad & x_i^+ \in [0, 1] \end{aligned}$$

- Optimal solution has a special structure: all tasks with $x_i^+ = 1$ have lower acceleration ratio than all tasks with $x_i^+ = 0$
- Can be obtained greedily, by sorting tasks in increasing values of ρ_i

Outline

HeteroPrio algorithm

HeteroPrio algorithm

- Sort tasks by increasing ρ_i in a double-ended queue
- When a CPU is idle, it picks from the start of the queue
- When a GPU is idle, it picks from the end of the queue

This is not an approximation algorithm, because it is **List**-like: never leave a resource idle if there is an available task

In an instance with $m = 1, k = 1$, and two tasks with $p_i^+ = M$ and $p_i^- = 1$, HP schedules one of them on CPU, even if M is very large.

HeteroPrio algorithm

HeteroPrio algorithm

- Sort tasks by increasing ρ_i in a double-ended queue
- When a CPU is idle, it picks from the start of the queue
- When a GPU is idle, it picks from the end of the queue

HP is a good approximation of the area bound idea, as long as all processors are busy:

Lemma

For any instance I, if all processors are busy up to time t, then

$$AreaBound(I) \geq t + AreaBound(I'(t))$$

where $I'(t)$ is the sub-instance made of (partial) tasks not completed at time t.

HeteroPrio algorithm

Theorem: HP is a 2-approximation in the "high load" case

If $\forall i, \max(p_i^+, p_i^-) \leq C^*_{\max}$, then HP is a 2-approximation.

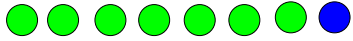
Proof:

- Denote by t_0 the first time a processor is idle in the HP schedule
- By previous Lemma, $t_0 \leq AreaBound(I)$
- After time t_0, each processor processes at most one task
- $C^{HP}_{\max} \leq t_0 + \max_i \left(\max(p_i^+, p_i^-)\right) \leq 2 \cdot C^*_{\max}$

HeteroPrio algorithm – the spoliation mechanism

HeteroPrio algorithm with spoliation

- Sort tasks by increasing ρ_i in a double-ended queue
- When a CPU is idle, it picks from the start of the queue
- When a GPU is idle, it picks from the end of the queue
- If no available task, pick a running task if it can make it finish earlier
  - Choose the one with highest current finish time



HeteroPrio algorithm – the spoliation mechanism

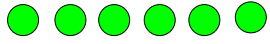
HeteroPrio algorithm with spoliation

- Sort tasks by increasing ρ_i in a double-ended queue
- When a CPU is idle, it picks from the start of the queue
- When a GPU is idle, it picks from the end of the queue
- If no available task, pick a running task if it can make it finish earlier
  - Choose the one with highest current finish time

HeteroPrio algorithm – the spoliation mechanism

HeteroPrio algorithm with spoliation

- Sort tasks by increasing ρ_i in a double-ended queue
- When a CPU is idle, it picks from the start of the queue
- When a GPU is idle, it picks from the end of the queue
- If no available task, pick a running task if it can make it finish earlier
  - Choose the one with highest current finish time



HeteroPrio algorithm – the spoliation mechanism

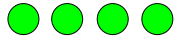
HeteroPrio algorithm with spoliation

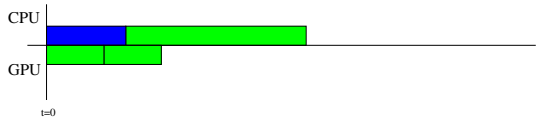
- Sort tasks by increasing ρ_i in a double-ended queue
- When a CPU is idle, it picks from the start of the queue
- When a GPU is idle, it picks from the end of the queue
- If no available task, pick a running task if it can make it finish earlier
  - Choose the one with highest current finish time

HeteroPrio algorithm – the spoliation mechanism

HeteroPrio algorithm with spoliation

- Sort tasks by increasing ρ_i in a double-ended queue
- When a CPU is idle, it picks from the start of the queue
- When a GPU is idle, it picks from the end of the queue
- If no available task, pick a running task if it can make it finish earlier
  - Choose the one with highest current finish time

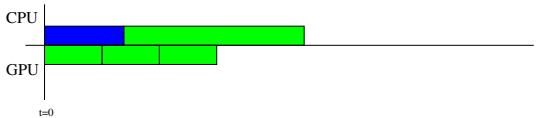




HeteroPrio algorithm – the spoliation mechanism

HeteroPrio algorithm with spoliation

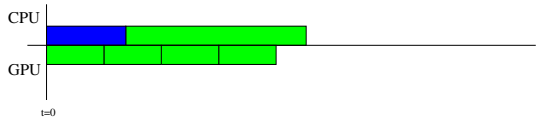
- Sort tasks by increasing ρ_i in a double-ended queue
- When a CPU is idle, it picks from the start of the queue
- When a GPU is idle, it picks from the end of the queue
- If no available task, pick a running task if it can make it finish earlier
  - Choose the one with highest current finish time



HeteroPrio algorithm – the spoliation mechanism

HeteroPrio algorithm with spoliation

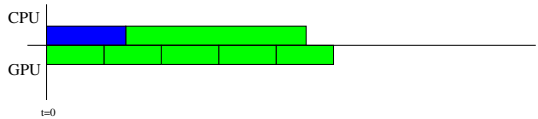
- Sort tasks by increasing ρ_i in a double-ended queue
- When a CPU is idle, it picks from the start of the queue
- When a GPU is idle, it picks from the end of the queue
- If no available task, pick a running task if it can make it finish earlier
  - Choose the one with highest current finish time



HeteroPrio algorithm – the spoliation mechanism

HeteroPrio algorithm with spoliation

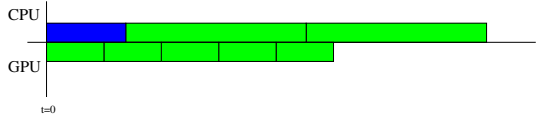
- Sort tasks by increasing ρ_i in a double-ended queue
- When a CPU is idle, it picks from the start of the queue
- When a GPU is idle, it picks from the end of the queue
- If no available task, pick a running task if it can make it finish earlier
  - Choose the one with highest current finish time



HeteroPrio algorithm – the spoliation mechanism

HeteroPrio algorithm with spoliation

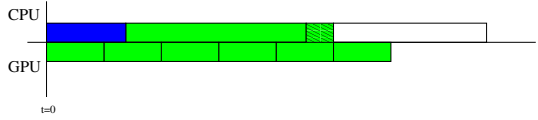
- Sort tasks by increasing ρ_i in a double-ended queue
- When a CPU is idle, it picks from the start of the queue
- When a GPU is idle, it picks from the end of the queue
- If no available task, pick a running task if it can make it finish earlier
  - Choose the one with highest current finish time



HeteroPrio algorithm – the spoliation mechanism

HeteroPrio algorithm with spoliation

- Sort tasks by increasing ρ_i in a double-ended queue
- When a CPU is idle, it picks from the start of the queue
- When a GPU is idle, it picks from the end of the queue
- If no available task, pick a running task if it can make it finish earlier
  - Choose the one with highest current finish time



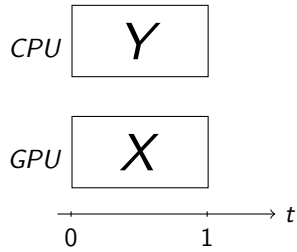
HeteroPrio approximation results

Approximation results

(#CPUs, #GPUs)	Approximation ratio	Worst case ex.
(1,1)	$\frac{1+\sqrt{5}}{2} \approx 1.62$	$\frac{1+\sqrt{5}}{2}$
(m,1)	$\frac{3+\sqrt{5}}{2} \approx 2.62$	$\frac{3+\sqrt{5}}{2}$
(m,n)	$2+\sqrt{2} \approx 3.41$	$2+\frac{2}{\sqrt{3}} \approx 3.15$

Task	CPU Time	GPU Time	accel ratio
X	ϕ	1	ϕ
Y	1	$\frac{1}{\phi}$	ϕ

Where $\phi = \frac{1+\sqrt{5}}{2}$



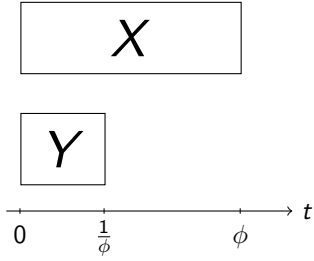
HeteroPrio approximation results

Approximation results

(#CPUs, #GPUs)	Approximation ratio	Worst case ex.
(1,1)	$\frac{1+\sqrt{5}}{2} \approx 1.62$	$\frac{1+\sqrt{5}}{2}$
(m,1)	$\frac{3+\sqrt{5}}{2} \approx 2.62$	$\frac{3+\sqrt{5}}{2}$
(m,n)	$2+\sqrt{2} \approx 3.41$	$2+\frac{2}{\sqrt{3}} \approx 3.15$

Task	CPU Time	GPU Time	accel ratio
X	ϕ	1	ϕ
Y	1	$\frac{1}{\phi}$	ϕ

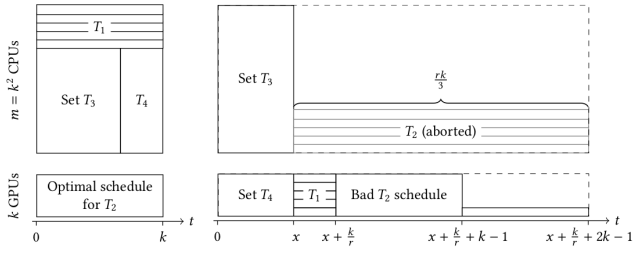
Where $\phi = \frac{1+\sqrt{5}}{2}$



HeteroPrio approximation results

Approximation results

(#CPUs, #GPUs)	Approximation ratio	Worst case ex.
(1,1)	$\frac{1+\sqrt{5}}{2} \approx 1.62$	$\frac{1+\sqrt{5}}{2}$
(m,1)	$\frac{3+\sqrt{5}}{2} \approx 2.62$	$\frac{3+\sqrt{5}}{2}$
(m,n)	$2+\sqrt{2} \approx 3.41$	$2+\frac{2}{\sqrt{3}} \approx 3.15$



(a) Optimal schedule

(b) HeteroPrio when the last task can be spoliated

Outline

DualHP algorithm: 3/2-approximation

Main idea

Given a guess λ on the optimal makespan:

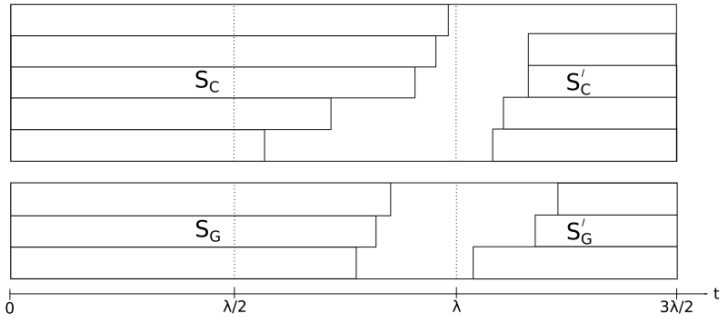
- Build a schedule with makespan $\frac{3\lambda}{2}$
- Or prove that λ is not feasible

DualHP algorithm: 3/2-approximation

Main idea

Given a guess λ on the optimal makespan:

- Build a schedule with makespan $\frac{3\lambda}{2}$
- Or prove that λ is not feasible



DualHP algorithm: 3/2-approximation

Main idea

Given a guess λ on the optimal makespan:

- Build a schedule with makespan $\frac{3\lambda}{2}$
- Or prove that λ is not feasible

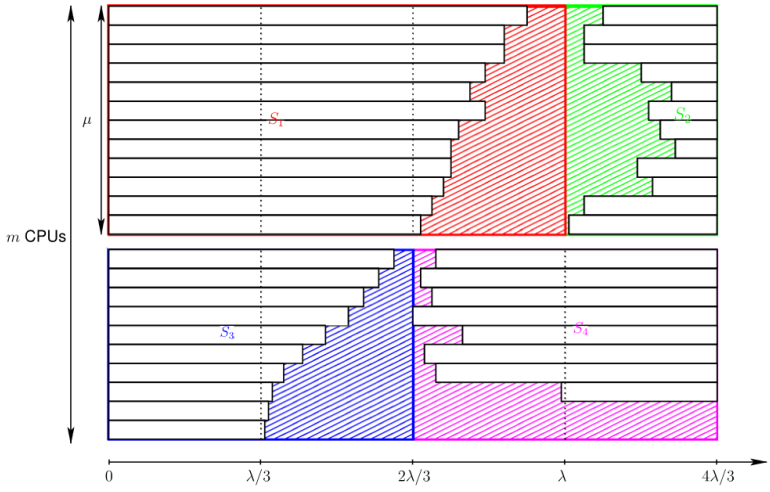
Solve a multi-dimensional knapsack (for example with Dynamic Programming):

$$
\begin{aligned}
\text{Find} \quad & x_i \in \{0,1\} \quad \text{s.t.} \\
& \textstyle\sum_{i,p_i^+ > \lambda/2} x_i \leq m \\
& \textstyle\sum_i x_i p_i^+ \leq m\lambda \\
& \textstyle\sum_{i,p_i^- > \lambda/2} (1 - x_i) \leq k \\
& \textstyle\sum_i (1 - x_i) p_i^- \leq k\lambda
\end{aligned}
$$

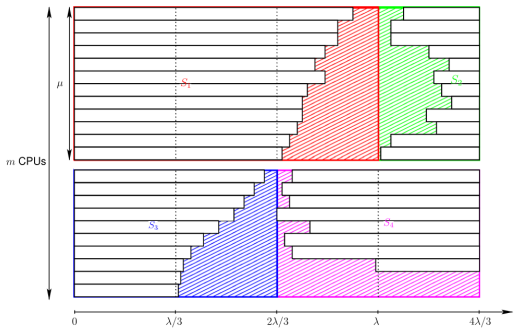
Theorem

Any solution of this knapsack problem can be arranged in the shelves as shown in the previous image.

DualHP algorithm: extend to get lower approximation ratio



DualHP algorithm: extend to get lower approximation ratio



Find $x_i \in \{0,1\}$ s.t.

$$\sum_{i,p_i^+>2\lambda/3} x_i + \frac{1}{2} \sum_{i,2\lambda/3\geq p_i^+>\lambda/3} x_i \leq m$$

$$\sum_i x_i p_i^+ \leq m\lambda$$

$$\sum_{i,p_i^->2\lambda/3} (1-x_i) + \frac{1}{2} \sum_{i,2\lambda/2\geq p_i^->\lambda/3} (1-x_i) \leq k$$

$$\sum_i (1-x_i) p_i^- \leq k\lambda$$

This yields a $\frac{4}{3}$-approximation algorithm. With more shelves, one can obtain stronger approximation ratios, but the complexity of the knapsack problem increases very quickly

Outline

HLP – Approximation for $Pm, Pk|\text{prec}|C_{\max}$

Linear Programming Lower Bound

$$
\begin{aligned}
\text{Minimize} \quad & C \\
\text{subject to:} \quad & \sum_i p_i^+ x_i \leq m \cdot C \\
& \sum_i p_i^- (1 - x_i) \leq k \cdot C \\
\forall i \rightsquigarrow j, \; & C_i + p_i^+ x_i + p_i^- (1 - x_i) \leq C_j \\
\forall i, \; & 0 \leq C_i \leq C \\
\forall i, \; & x_i \in [0, 1]
\end{aligned}
$$

HLP algorithm

HLP algorithm

- Solve Linear Programming lower bound, obtain solution (x)
- Round x to the nearest integer: $\tilde{x}_i = 0$ iff $x_i < 0.5$
- Schedule with List Scheduling, according to $\tilde{x}$

Lemma: cost of rounding

The rounded solution $\tilde{x}$ has cost at most twice the cost of x

$$p_i^+ \tilde{x}_i \leq 2 p_i^+ x_i \qquad\qquad p_i^- (1 - \tilde{x}_i) \leq 2 p_i^- (1 - x_i)$$

HLP algorithm

Theorem

HLP is a 6-approximation algorithm

Sketch of proof:

- Divide the schedule in 3 types of intervals: I^+ all CPUs busy, I^- all GPUs busy, I_{CP} at least one idle of each type
- I_{CP} is bounded by the critical path according to $\tilde{x}$
- I^+ and I^- are bounded by the average workloads
- $C_{max}^{HLP} \leq I_{CP} + I^+ + I^- \leq 3 \cdot cost(\tilde{x}) \leq 6 \cdot cost(x) \leq 6 \cdot C_{max}^*$

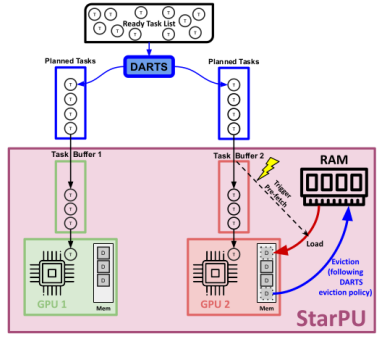
Outline

Locality-focused: DARTS scheduler

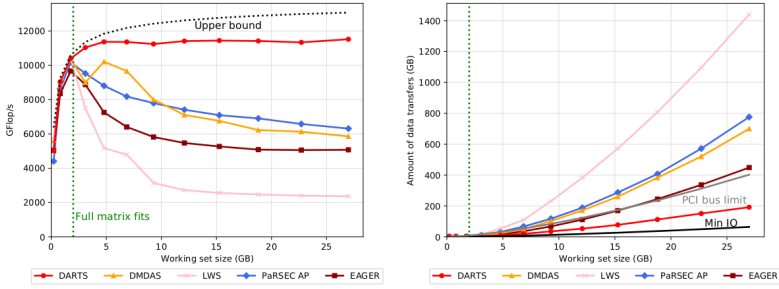
Data-Aware Reactive Task Scheduling

Main idea: focus on placing data rather than scheduling tasks

- Plan tasks in advance, with flexibility: *plannedTasks* and *taskBuffer*
- When a resource is idle, select the data that will unlock the largest number of tasks
- Add all unlocked tasks to *plannedTasks*
- When memory is full, evict data Least Used in the Future



Locality-focused: DARTS scheduler



(a) Performance.

(b) Amount of data transfers.

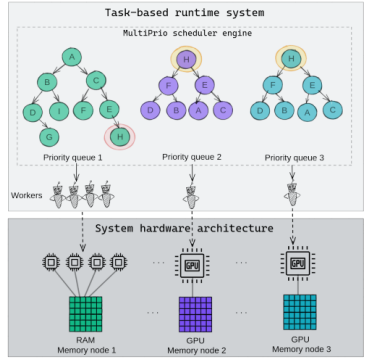
Figure 5.2: Results on the Cholesky factorization with 1 Tesla V100 GPU. Memory limited to 2000 MB.

Beyond HeteroPrio: MultiPrio

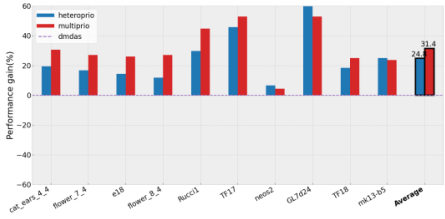
Refined version of HeteroPrio

Additional features:

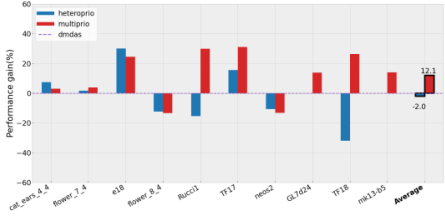
- *Affinity* score to evaluate how tasks are adapted to resources
- *Criticality* score to evaluate how much workload is released when the task completes
- *Locality* score to evaluate how much tasks can reuse already present data



Beyond HeteroPrio: MultiPrio



(a) Performance on Intel-V100 platform.



(b) Performance on AMD-A100 platform.

Fig. 8: Performance of QR factorization (ordering METIS) on Intel-V100 and AMD-A100 both with 2 GPUs, relative to the Dmdas scheduler. Matrices sorted by Gflops count.

More research going on

Ongoing scheduling research around StarPU

- Dynamic task and data partitioning thanks to hierarchical tasks
- Dynamic scheduler selection:
  - HeteroPrio-like when parallelism is high,
  - HEFT-like scheduling when critical path matters more,
  - DARTS-like when memory is constrained
  - ...

In conclusion

Scheduling is difficult, but fun! If you want to join, you're welcome!