Multi-Resource Scheduling of Parallel Jobs

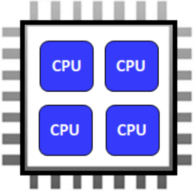
Hongyang Sun

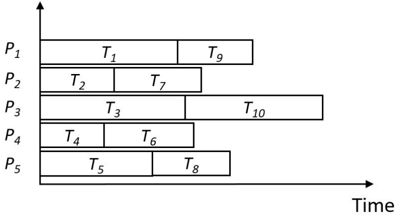
Vanderbilt University

The 13th Scheduling for Large Scale Systems Workshop
June 18-20 2018, Berkeley, CA, USA

Single-resource scheduling

- Most traditional scheduling problems target a single type of resource (e.g., CPUs)



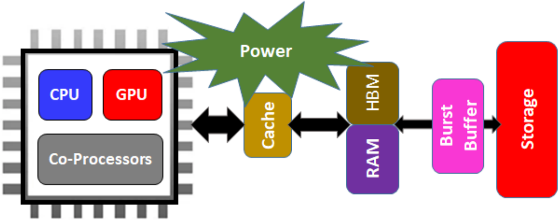


- For example: classic NP-complete problem of makespan minimization on identical machines ($P||C_{\max}$)
  - List scheduling is $(2 - \frac{1}{P})$-approx. [Graham 1969]
  - Many other heuristics

Introduction

The case for multi-resource scheduling

- HPC systems embrace more heterogeneous components (e.g., *CPU, GPU, FPGA, MIC, APU*)
- Data-intensive applications drive architectural enhancement to support better data-transfer efficiency (e.g., *High-Bandwidth Memory, Partitionable Cache, Burst Buffers*)
- Power has become a first-class resource (e.g., *due to thermal/cooling/energy constraints*)



Optimal system/application performance may be achieved by scheduling two or more types of resources simultaneously

Focus of This Work

Simple algorithms (e.g., list) with **approximation guarantee**:

$$\rho\text{-approx.} \iff M_{\text{alg}} \leq \rho \cdot M_{\text{opt}} \text{ for all instances}$$

Few **prior works** on multi-resource scheduling:

- Rigid Job Scheduling [Garey & Graham 1975]
  - Jobs have fixed resource requirements and execution times
  - $(d+1)$-approximation with d resource types
- Job/DAG-Shop Scheduling [Shmoy, Stein & Wein 1994]
  - Jobs have chains/DAGs of heterogeneous tasks
  - Each task requires a specific machine type to process
  - Tasks of each job must be processed sequentially
  - Polylog approximation in number of machines and job length

Outline

Multi-Resource Scheduling of Moldable Jobs[1]

Scheduling Parallel Tasks under Multiple Resources: List Scheduling vs. Pack Scheduling. H. Sun, R. Elghazi, A. Gainaru, G. Aupy and P. Raghavan. *In Proceedings of The 32nd International Parallel and Distributed Processing Symposium (IPDPS)*, 2018

[1] Jobs can be executed with *different* amounts of resources, but resource allocations *cannot* be changed during runtime

Model and Objective

Model:

- System with d resource types; i-th type has $P^{(i)}$ identical resources
- Set $\{1, 2, \cdots, n\}$ of independent jobs all released at time 0
- Each job j's execution time $t_j(\vec{p}_j)$ depends on its resource allocation vector $\vec{p}_j = (p_j^{(1)}, p_j^{(2)}, \cdots, p_j^{(d)})$
- Assumption: *non-increasing execution time*

$$\vec{p}_j \preceq \vec{q}_j \ \left(\text{or } p_j^{(i)} \leq q_j^{(i)}, \forall i\right) \implies t_j(\vec{p}_j) \geq t_j(\vec{q}_j)$$

Objective:

- Find a moldable schedule, i.e., resource allocation vector $\vec{p}_j$ and starting time s_j for each job j
  - minimize makespan: $T = \max_j(s_j + t_j(\vec{p}_j))$
  - subject to resource constraint: $\sum_{j \text{ active at time } t} p_j^{(i)} \leq P^{(i)}, \forall i, t$

Preliminaries

Definitions: for a given resource allocation $\mathbf{p} = (\vec{p}_1, \vec{p}_2, \cdots, \vec{p}_n)^T$

- Total area (normalized): $A(\mathbf{p}) = \sum_{j=1}^{n} \sum_{i=1}^{d} \frac{p_j^{(i)}}{P^{(i)}} \cdot t_j(\vec{p}_j)$
- Maximum execution time: $t_{\max}(\mathbf{p}) = \max_{j=1\ldots n} t_j(\vec{p}_j)$

Preliminaries

Definitions: for a given resource allocation $\mathbf{p} = (\vec{p}_1, \vec{p}_2, \cdots, \vec{p}_n)^T$

- Total area (normalized): $A(\mathbf{p}) = \sum_{j=1}^{n} \sum_{i=1}^{d} \frac{p_j^{(i)}}{P^{(i)}} \cdot t_j(\vec{p}_j)$
- Maximum execution time: $t_{\max}(\mathbf{p}) = \max_{j=1\ldots n} t_j(\vec{p}_j)$

Lower bound (on makespan): $L(\mathbf{p}, d) = \max\left(\frac{A(\mathbf{p})}{d}, t_{\max}(\mathbf{p})\right)$

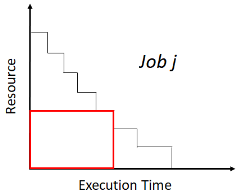
Proposition

*The **optimal makespan** satisfies*

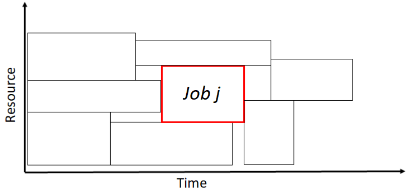
$$T_{\text{OPT}} \geq L_{\min}(d) = \min_{\mathbf{p}} L(\mathbf{p}, d)$$

Two-Phase Approach [Turek et al. 1992]

- *Phase 1*: Determines a resource allocation for each moldable job



- *Phase 2*: Constructs a rigid schedule based on the fixed resource allocations of all jobs

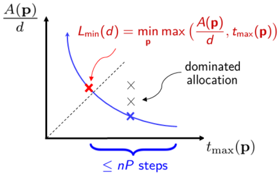


Phase 1: Resource Allocation

Goal: find allocation $\mathbf{p}^d_{\min}$ matching lower bound $L_{\min}(d) = \min_{\mathbf{p}} L(\mathbf{p}, d)$

Resource Allocation (RA_d)

- Step (1). For each job j:
  - Linearize all $P = \prod_{i=1}^{d}(P^{(i)} + 1)$ allocations
  - Remove ones with both higher execution time and larger area
  - Sort in order of increasing execution time and decreasing area
- Step (2). For all n jobs:
  - Traverse the n lists in $\leq nP$ steps by tracing $t_{\max}(\mathbf{p})$ at each step until dominated by $\frac{A(\mathbf{p})}{d}$ (v.s. exhaustive search in P^n time)



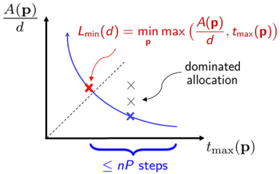
Complexity: $O(nP(\log P + \log n + d))$

Phase 1: Resource Allocation

Goal: find allocation $\mathbf{p}^d_{\min}$ matching lower bound $L_{\min}(d) = \min_{\mathbf{p}} L(\mathbf{p}, d)$

Resource Allocation (RA_d)

- Step (1). For each job j:
  - Linearize all $P = \prod_{i=1}^{d}(P^{(i)} + 1)$ allocations
  - Remove ones with both higher execution time and larger area
  - Sort in order of increasing execution time and decreasing area
- Step (2). For all n jobs:
  - Traverse the n lists in $\leq nP$ steps by tracing $t_{\max}(\mathbf{p})$ at each step until dominated by $\frac{A(\mathbf{p})}{d}$ (v.s. exhaustive search in P^n time)



Complexity: $O(nP(\log P + \log n + d))$

Proposition

If a ***rigid scheduling algorithm*** R_d *that uses* $\mathbf{p}^d_{\min}$ *produces a makespan*

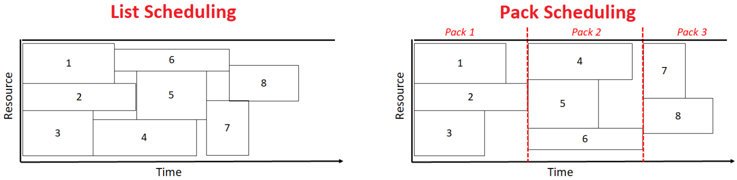
$$T_{\mathrm{R}_d}(\mathbf{p}^d_{\min}) \leq c \cdot L_{\min}(d)$$

then the ***two-phase algorithm*** $\mathrm{RA}_d + \mathrm{R}_d$ *is c-approximation*

Phase 2: Rigid Scheduling

Two scheduling paradigms:

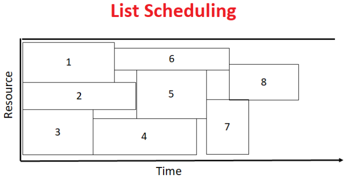
- List Scheduling (LS_d): 2-approx. for $d = 1$
  - Greedily schedules jobs in a list with sufficient resources
- Pack Scheduling (PS_d): 3-approx. for $d = 1$
  - Partitions jobs in packs to be scheduled one after another

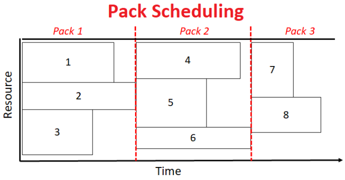


Phase 2: Rigid Scheduling

Two scheduling paradigms:

- List Scheduling (LS_d): 2-approx. for $d = 1$
  - Greedily schedules jobs in a list with sufficient resources
- Pack Scheduling (PS_d): 3-approx. for $d = 1$
  - Partitions jobs in packs to be scheduled one after another





Proposition

For a set of rigid tasks with any ***fixed resource allocation*** $\mathbf{p}$, *we have*

$$\textbf{\textit{List Scheduling}}: \quad T_{\mathrm{LS}_d}(\mathbf{p}) \leq 2d \cdot L(\mathbf{p}, d)$$

$$\textbf{\textit{Pack Scheduling}}: \quad T_{\mathrm{PS}_d}(\mathbf{p}) \leq (2d+1) \cdot L(\mathbf{p}, d)$$

Put Them Together

Proposition

*The **two-phase algorithms** have the following approximation ratios:*

$$\mathbf{RA}_d + \mathbf{LS}_d \textbf{\textit{ (List)}} : 2d\text{-approx.}$$
$$\mathbf{RA}_d + \mathbf{PS}_d \textbf{\textit{ (Pack)}} : (2d+1)\text{-approx.}$$

*Moreover, the **bounds are tight** for both algorithms*

Put Them Together

Proposition

*The **two-phase algorithms** have the following approximation ratios:*

$$\mathbf{RA}_d + \mathbf{LS}_d \ \textit{(List)} \ : \ 2d\text{-approx.}$$
$$\mathbf{RA}_d + \mathbf{PS}_d \ \textit{(Pack)} : (2d+1)\text{-approx.}$$

*Moreover, the **bounds are tight** for both algorithms*

Tightness instance (for list):

- $n = 2d$ jobs, and $P^{(i)} = 2P$ for each resource type i
- All jobs have the following profiles:

  (1) $t_j(0, \cdots, 0, P, 0, \cdots, 0) = 1$, where P appears in position $\lceil \frac{j}{2} \rceil$
  (2) $t_j(P+1, 0, \cdots, 0) = \frac{P-1}{P+1}$
- $\mathrm{RA}_d + \mathrm{LS}_d$ chooses allocation (2), since allocation (1) is dominated in both execution time and area, thus all jobs are executed sequentially
- OPT chooses allocation (1), thus is able to run all jobs in parallel
- $\frac{T_{\mathrm{RA}_d+\mathrm{LS}_d}}{T_{\mathrm{OPT}}} = 2d\frac{P-1}{P+1} \to 2d$ as $P \to \infty$

Transformation

Transformation (TF):

- Step (1). d-**resource instance** $I \Longrightarrow$ 1-**resource instance** I'
  - I' has same number n of jobs and total resource $Q = \text{lcm}_{i=1\cdots d} P^{(i)}$
  - For any job j' in I': execution time $t_{j'}(q) = t_j((\lfloor \frac{q \cdot P^{(i)}}{Q} \rfloor)_{i=1\cdots d}) \ \forall q$
- Step (2). **Solve the 1-resource instance** I'
- Step (3). 1-**resource solution** $S' \Longrightarrow d$-**resource solution** S
  - For any job j in I: starting time is same $s_j = s_{j'}$

    resource allocation is $\vec{p}_j = (\lfloor \frac{q_{j'} \cdot P^{(i)}}{Q} \rfloor)_{i=1\cdots d}$

Example

Given $P^{(1)} = 4, P^{(2)} = 8, P^{(3)} = 16 \ \Rightarrow \ Q = \text{lcm}(4, 8, 16) = 16$
Step (1): $t_{j'}(8) = t_j(2, 4, 8)$
Step (3): $q_{j'} = 4 \ \Rightarrow \ \vec{p}_j = (1, 2, 4)$

Transformation

Proposition

*The **transformation process** preserves the approximation ratios:*

$$\textbf{TF} + \textbf{RA}_1 + \textbf{LS}_1 \ \textbf{\textit{(List)}} \quad : \ 2d\text{-}\textit{approx.}$$

$$\textbf{TF} + \textbf{RA}_1 + \textbf{PS}_1 \ \textbf{\textit{(Pack)}} : (2d+1)\text{-}\textit{approx.}$$

Transformation

Proposition

*The **transformation process** preserves the approximation ratios:*

$$\mathbf{TF} + \mathbf{RA}_1 + \mathbf{LS}_1 \ \textit{\textbf{(List)}} \ : 2d\text{-approx.}$$

$$\mathbf{TF} + \mathbf{RA}_1 + \mathbf{PS}_1 \ \textit{\textbf{(Pack)}} : (2d+1)\text{-approx.}$$

Complexity: If $P^{(i)} = p \ \forall i = 1 \ldots d$

- Transformation $\propto Q = \operatorname{lcm}_i P^{(i)} = p$
- Direct Solution $\propto P = \prod_i (P^{(i)} + 1) = p^d$

Significantly faster for large d

Multi-Resource Scheduling of Malleable Jobs[2]

Scheduling Functional Heterogeneous Systems with Utilization Balancing. Y. He, J. Liu and H. Sun. *In Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2011

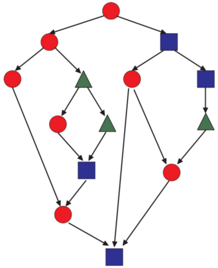
Adaptive Scheduling of Parallel Jobs on Functionally Heterogeneous Resources. Y. He, H. Sun and W-J. Hsu. *In Proceedings of the International Conference on Parallel Processing (ICPP)*, 2007

[2] Jobs can be executed with varying amount of resources during runtime

Model and Objective

Model:

- System with d resource types; i-th type has $P^{(i)}$ identical resources
- Set $\{1, 2, \cdots, n\}$ of independent jobs with arbitrary release time
- Each job j is represented as a DAG of heterogeneous tasks, each of unit size
- Tasks can be executed in parallel, but each task can only be executed by a resource of corresponding type



Objective:

- Find a malleable schedule, i.e., resource allocation vector $\vec{p}_j(t) = \left(p_j^{(1)}(t), p_j^{(2)}(t), \cdots, p_j^{(d)}(t)\right)$ and set of tasks $V_j(t)$ to execute for each job j at any time t
  - minimize makespan: $T = \max_j c_j$ (c_j is completion time of j)
  - subject to resource and precedence constraints

Preliminaries

Definitions for any job j:

- Work of resource type i: $T_{1,j}^{(i)}$
- Critical-path length: $T_{\infty,j}$
- Release time: r_j

Definitions for job set:

- Total work of resource type i: $T_1^{(i)} = \sum_j T_{1,j}^{(i)}$
- Maximum critical-path length: $T_\infty = \max(r_j + T_{\infty,j})$

Analogous to total area and maximum execution time in moldable model

Preliminaries

Definitions for any job j:

- Work of resource type i: $T_{1,j}^{(i)}$
- Critical-path length: $T_{\infty,j}$
- Release time: r_j

Definitions for job set:

- Total work of resource type i: $T_1^{(i)} = \sum_j T_{1,j}^{(i)}$
- Maximum critical-path length: $T_\infty = \max(r_j + T_{\infty,j})$

Analogous to total area and maximum execution time in moldable model

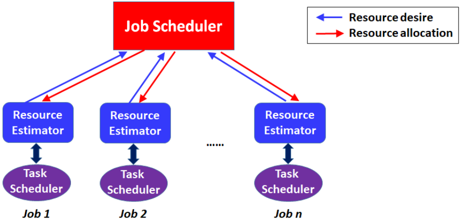
Lower bound (on makespan):

Proposition

The ***optimal makespan*** *satisfies*

$$T_{\text{OPT}} \geq \max\left(T_\infty, \max_i \frac{T_1^{(i)}}{P^{(i)}}\right)$$

Two-Level Approach



At each step t:

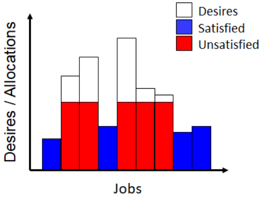
- *Phase 1*: Resource Estimator computes for each job j
  a resource desire vector $\vec{d}_j(t) = (d_j^{(1)}(t), d_j^{(2)}(t), \cdots, d_j^{(d)}(t))$
- *Phase 2*: Job Scheduler based on desires of all jobs and system policy determines for each job j
  a resource allocation vector $\vec{p}_j(t) = (p_j^{(1)}(t), p_j^{(2)}(t), \cdots, p_j^{(d)}(t))$
- *Phase 3*: Task Scheduler schedules ready tasks of each job using allocated resources

This approach can also be applied to non-clairvoyant, adaptive scheduling

Algorithm

Adaptive Greedy (AG_d): 2-approx. for $d = 1$

- *Phase 1*: Resource Estimator
  - Use instantaneous parallelism as resource desire
  - $d_j^{(i)}(t)$ = number of ready tasks of type i for job j at time t
- *Phase 2*: Job Scheduler
  - Use dynamic equi-partitioning [McCann et al. 1993]
  - Satisfy jobs with low desires
  - Equally partition remaining resources on high-desire jobs



- *Phase 3*: Task Scheduler
  - Schedule ready tasks of each type greedily, i.e.
    if $p_j^{(i)}(t) = d_j^{(i)}(t)$, schedule all ready tasks
    if $p_j^{(i)}(t) < d_j^{(i)}(t)$, schedule any $p_j^{(i)}(t)$ ready tasks

Desire, allocation and scheduling are handled independently for different resource types

Performance

Proposition

*The **Adaptive Greedy** algorithm achieves*

$$T_{\mathrm{AG}_d} \leq \sum_{i=1}^{d} \frac{T_1^{(i)}}{P^{(i)}} + \left(1 - \frac{1}{P_{\max}}\right) T_\infty$$

and is therefore $\left(d + 1 - \frac{1}{P_{\max}}\right)$*-approximation, where* $P_{\max} = \max_i P^{(i)}$
*Moreover, the **bound is tight** for the algorithm*

Performance

Proposition

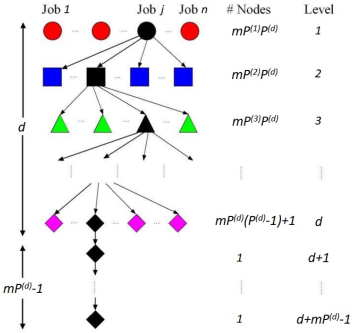
*The **Adaptive Greedy** algorithm achieves*

$$T_{\mathrm{AG}_d} \leq \sum_{i=1}^{d} \frac{T_1^{(i)}}{P^{(i)}} + \left(1 - \frac{1}{P_{\max}}\right) T_\infty$$

and is therefore $\left(d + 1 - \frac{1}{P_{\max}}\right)$*-approximation, where* $P_{\max} = \max_i P^{(i)}$
*Moreover, the **bound is tight** for the algorithm*

Tightness instance (as $m \to \infty$):

- AG_d chooses "wrong" tasks and uses different resources sequentially
- OPT picks "right" tasks and uses different resources in parallel
- Same bound even applies to randomized algorithms
- Lookahead may help ☺



Outline

Conclusion

Now is a good time to revisit multi-resource scheduling problems

Conclusion

Now is a good time to revisit multi-resource scheduling problems

Open Question 1: List/greedy-scheduling for moldable jobs

- Rigid jobs: $(d+1)$-approx. [Garey and Graham, 1975]
- Moldable jobs: $2d$-approx. [Sun et al. 2018]
- Malleable jobs: $(d+1-1/P_{\max})$-approx. [He et al. 2007]
  (Represented as DAGs containing unit-size tasks of different types)

- *Can we achieve* $(d+1)$*-approx. for moldable jobs (possibly with an alternative resource allocation strategy or a more coupled design/analysis of resource allocation and rigid scheduling), or is it inherently harder?*

Open Question 2: Moldable job scheduling under general models

- 2-Pack Sol.: $(1.5+\epsilon)$-approx. [Mounié et al. 2004, Jansen 2012]
- Precedence constraints: e.g., $(3+\sqrt{5})$-approx. [Lepère et al. 2001]

- *Could these single-resource results be extended to multi-resource?*