On the scheduling of graphs of tasks: 
A scalable clustering-based approach using DAG partitioning

Anne Benoit

LIP, ENS Lyon, France

PPAM 2019
September 8-11, 2019 – Bialystok, Poland
Solve the linear system $Ax = b$, where $A$ is $n \times n$ nonsingular lower triangular matrix, and $b$ a vector with $n$ components:

$$\text{for } i = 1 \text{ to } n \text{ do }$$

$$\text{Task } T_{i,i}: x_i \leftarrow b_i/a_{i,i}$$

$$\text{for } j = i + 1 \text{ to } N \text{ do }$$

$$\text{Task } T_{i,j}: b_j \leftarrow b_j - a_{j,i} \times x_i$$

Tasks are nodes, with different completion times.

Data dependencies among tasks are represented as edges.
Scientific workflows

Pegasus, pegasus.isi.edu

Scheduling graphs of tasks
How can we efficiently execute a task graph on a parallel platform? How to schedule it?
Applications modeled as a directed acyclic graph (DAG) $G = (V, E)$

- **Nodes**: tasks with different completion times
- **Edges**: data dependencies among tasks

Need of efficient scheduling techniques

Objective function

- Minimize the total execution time, i.e., the makespan of the DAG
- Scheduling literature: $P|\text{prec, } c_{i,j}|C_{\text{max}}$ problem

History

- List-based scheduling
- Clustering-based scheduling
List-based scheduling example

- Tasks ordered based on some predetermined priority
- Greedily assign a ready task to an available processor as early as possible (don’t leave a processor idle unnecessarily)

3 processors

Makespan = 16; Critical path length = 15; Idle time = 1+5+5+8 = 19

2-approximation algorithm

TDA lab

September 10, 2019,
Anne.Benoit@ens-lyon.fr

Scheduling graphs of tasks
List-based scheduling example

- Tasks ordered based on some predetermined priority
- Greedily assign a ready task to an available processor as early as possible (don't leave a processor idle unnecessarily)

3 processors

Makespan = 16; Critical path length = 15; Idle time = 1+5+5+8 = 19

2-approximation algorithm
List-based scheduling example

- Tasks ordered based on some predetermined priority
- Greedily assign a ready task to an available processor as early as possible (don't leave a processor idle unnecessarily)
List-based scheduling example

- Tasks ordered based on some predetermined priority
- Greedily assign a ready task to an available processor as early as possible (don't leave a processor idle unnecessarily)

3 processors

Makespan = 16; Critical path length = 15; Idle time = 1+5+5+8 = 19

2-approximation algorithm

TDA lab

September 10, 2019,
Anne.Benoit@ens-lyon.fr

Scheduling graphs of tasks

Introduction 5 / 33
List-based scheduling example

- Tasks ordered based on some predetermined priority
- Greedily assign a ready task to an available processor *as early as possible* (don’t leave a processor idle unnecessarily)
List-based scheduling example

- Tasks ordered based on some predetermined priority
- Greedily assign a ready task to an available processor as early as possible (do not leave a processor idle unnecessarily)

3 processors

Makespan = 16; Critical path length = 15; Idle time = 1+5+5+8 = 19
List-based scheduling example

- Tasks ordered based on some predetermined priority
- Greedily assign a ready task to an available processor *as early as possible* (don't leave a processor idle unnecessarily)

3 processors

Makespan = 16; Critical path length = 15; Idle time = 1+5+5+8 = 19

**2-approximation algorithm**
Cluster-based scheduling

Account for **communications**: execute on same processor two tasks with large communications
Cluster-based scheduling

Account for **communications**: execute on same processor two tasks with large communications

*Kim and Browne’s linear clustering*
Cluster-based scheduling

Account for **communications**: execute on same processor two tasks with large communications

---

*Kim and Browne’s linear clustering*

*Sarkar’s greedy clustering*

---

September 10, 2019,
Anne.Benoit@ens-lyon.fr

---

TDA lab
A novel approach

Further motivation

- Consider the **realistic duplex single-port communication model**
  - only one send and one receive at a time
- Find a way to take **global clustering** decisions

Idea

- Build upon **DAG partitioner** to design scheduling heuristics accounting for data locality
- Recent paper in IPDPS’19: A **scalable clustering-based task scheduler for homogeneous processors using DAG partitioning**, M. Yusuf Özkaya, Julien Herrmann, Anne Benoit, Bora Uçar, Umit V. Çatalyürek, from CSE, Georgia Institute of Technology, GA, USA, and CNRS and LIP, ENS Lyon, France
Outline

1 Model

2 Algorithms

3 Experiments

4 Conclusion
Problem

Model

- Directed acyclic task graph: $G = (V, E)$
  - $w_i$: task weight
  - $c_{i,j}$: communication cost
- Homogeneous platform:
  - $p$ identical processors
  - fully connected homogeneous network
- **Duplex single-port model**: Each processor can, in parallel, without contention:
  - execute a task
  - send one data to one processor
  - receive one data from one processor

**MinMakespan**

Find the task mapping onto processors, the task starting times and communication starting times, so that the makespan is minimized
An example

For each task $v_i \in V$, $w_i = 1$

Scheduling graphs of tasks

Model 10 / 33
Outline

1 Model

2 Algorithms

3 Experiments

4 Conclusion
Algorithms: the competitors

Winners of the recent comparison done by Wang and Sinnen
[List-scheduling vs. cluster-scheduling, IEEE TPDS, 2018]

List schedulers

- **BL-EST**: chooses task with largest bottom-level first (BL), and assigns task on processor with earliest start time (EST)
- **ETF**: tries all ready tasks on all processors and picks the combination with the earliest EST first

Cluster-based scheduler

- **DSC-GLB-ETF**: uses dominant sequence clustering (DSC), then merges clusters with guided load balancing (GLB), and finally orders tasks using earliest EST first (ETF).

... And realistic *duplex single-port* communication model!
Prioritizing phase

Prioritizing tasks according to their **bottom level**:

\[ b1(i) = w_i + \begin{cases} 
0 & \text{if } \text{Succ}[v_i] = \emptyset; \\
\max_{v_j \in \text{Succ}[v_i]} \{ c_{i,j} + b1(j) \} & \text{otherwise.} 
\end{cases} \quad (1) \]

Assigning tasks to processors

Until the list of ready tasks is not empty:

- Select a ready task with the highest priority
- Compute start time of the task on each processor (with ASAP strategy for communications)
- Map the task on the processor with **earliest start time**
BL-EST example

- Vertices are numbered according to their priority
- BL-EST has a local view of the graph
- BL-EST can be arbitrarily worse than the best schedule
ETF: earliest EST first

Dynamic priority list scheduler

- Compute EST of each ready task
- Schedule task with earliest EST
- Similar lack of general view of the graph than BL-EST
- Higher complexity than BL-EST
Partition-based scheduling

**Principle**
- Partition the DAG into $K > p$ parts to enhance data locality
- Weights of parts are balanced with a 10% ratio (other values give similar results)
- The edge cut is reduced
- The partition is acyclic (dependence graph for parts is acyclic)
- Use the global view of the partition in the list-based scheduling

**Partition-based scheduler**
- Once a task of a part has been mapped, enforce that other tasks of the same part share the same processors
- Three variants, used on top of classical list-based scheduler
Assigning tasks to processors

Follow list-scheduler, with additional constraint:

- If a task from the same part has already been assigned to a processor, map the task onto the same processor
- Else, behave similarly to list scheduler
**-Busy**

**Drawback of **-**Part**
- May overload a processor with several on-going parts
- When starting a new part, ignores previous decisions

**How to deal with this problem?**
- Maintain list of busy processors (i.e., processors that have been assigned a task from a part but not all of them yet assigned)

**Assigning tasks to processors**
Select ready task with highest priority:
- If a task from the same part has already been assigned to a proc., map it onto the same proc.
- Else, if all processors are busy, behave like list-scheduler
- Else, behave like list-scheduler on non-busy processors only
$p = 2$ and $K = 3$
Concept
- Map a whole part before moving to the next one
- Priority of a part is the maximum bottom level of its tasks
- Maintain list of ready parts

Assigning tasks to processors
- Two priority algorithms: one for parts and one for tasks
- Select ready part with highest priority
- Tentatively schedules the whole part on each processor
  - Select ready task with highest priority
  - Incoming communications are scheduled ASAP, ensuring one-port model
- Map part on processor with earliest finish time for the last task
\( p = 2 \) and \( K = 3 \)
Outline

1 Model

2 Algorithms

3 Experiments

4 Conclusion
Graph instances

Instances from the **SuiteSparse** Matrix Collection (denoted UFL):

<table>
<thead>
<tr>
<th>Graph</th>
<th></th>
<th>( V )</th>
<th></th>
<th>( E )</th>
<th>max.</th>
<th>avg.</th>
<th>#source</th>
<th>#target</th>
</tr>
</thead>
<tbody>
<tr>
<td>598a</td>
<td>110,971</td>
<td>741,934</td>
<td>26</td>
<td>13.38</td>
<td>6,485</td>
<td>8,344</td>
<td></td>
<td></td>
</tr>
<tr>
<td>caidaRouterLev.</td>
<td>192,244</td>
<td>609,066</td>
<td>1,071</td>
<td>6.34</td>
<td>7,791</td>
<td>87,577</td>
<td></td>
<td></td>
</tr>
<tr>
<td>delaunay-n17</td>
<td>131,072</td>
<td>393,176</td>
<td>17</td>
<td>6.00</td>
<td>17,111</td>
<td>10,082</td>
<td></td>
<td></td>
</tr>
<tr>
<td>email-EuAll</td>
<td>265,214</td>
<td>305,539</td>
<td>7,630</td>
<td>2.30</td>
<td>260,513</td>
<td>56,419</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fe-ocean</td>
<td>143,437</td>
<td>409,593</td>
<td>6</td>
<td>5.78</td>
<td>40</td>
<td>861</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ford2</td>
<td>100,196</td>
<td>222,246</td>
<td>29</td>
<td>4.44</td>
<td>6,276</td>
<td>7,822</td>
<td></td>
<td></td>
</tr>
<tr>
<td>luxembourg-osm</td>
<td>114,599</td>
<td>119,666</td>
<td>6</td>
<td>4.16</td>
<td>3,721</td>
<td>9,171</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rgg-n-2-17-s0</td>
<td>131,072</td>
<td>728,753</td>
<td>28</td>
<td>5.56</td>
<td>598</td>
<td>615</td>
<td></td>
<td></td>
</tr>
<tr>
<td>usroads</td>
<td>129,164</td>
<td>165,435</td>
<td>7</td>
<td>2.56</td>
<td>6,173</td>
<td>6,040</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vsp-mod2-pgp2.</td>
<td>101,364</td>
<td>389,368</td>
<td>1,901</td>
<td>7.68</td>
<td>21,748</td>
<td>44,896</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Instances from the **Open Community Runtime** collection (denoted OCR):

<table>
<thead>
<tr>
<th>Graph</th>
<th></th>
<th>( V )</th>
<th></th>
<th>( E )</th>
<th>max.</th>
<th>avg.</th>
<th>#source</th>
<th>#target</th>
</tr>
</thead>
<tbody>
<tr>
<td>cholesky</td>
<td>1,030,204</td>
<td>1,206,952</td>
<td>5,051</td>
<td>2.34</td>
<td>333,302</td>
<td>505,003</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fibonacci</td>
<td>1,258,198</td>
<td>1,865,158</td>
<td>206</td>
<td>3.96</td>
<td>2</td>
<td>296,742</td>
<td></td>
<td></td>
</tr>
<tr>
<td>quicksort</td>
<td>1,970,281</td>
<td>2,758,390</td>
<td>5</td>
<td>2.80</td>
<td>197,030</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSBench</td>
<td>766,520</td>
<td>1,502,976</td>
<td>3,074</td>
<td>3.96</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smith-water.</td>
<td>58,406</td>
<td>83,842</td>
<td>7</td>
<td>2.88</td>
<td>164</td>
<td>6,885</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UTS</td>
<td>781,831</td>
<td>2,061,099</td>
<td>9,727</td>
<td>5.28</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XSBench</td>
<td>898,843</td>
<td>1,760,829</td>
<td>6,801</td>
<td>3.92</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Datasets and CCR

### Three datasets
- **Small** dataset: 1600 graph instances with 50 to 1151 nodes, from [Wang and Sinnen]
- **Medium** dataset: subset of UFL/OCR graphs, with 10k to 150k nodes
- **Big** dataset: all UFL and OCR graphs

### Communication-to-computation ratio (CCR) definition
For a graph $G = (V, E)$, the CCR is formally defined as

$$\text{CCR} = \frac{\sum_{(v_i,v_j) \in E} c_{i,j}}{\sum_{v_i \in V} w_i}$$

Create instances with a target CCR for UFL and OCR graphs:

1. randomly assign chosen costs and weights between 1 and 10 to each edge and vertex
2. scale edge costs appropriately to yield the desired CCR
Communication-delay model vs. realistic model

Comm-delay: [Wang&Sinnen] vs our implementation, small data set, CCR=0.1, 1, 10
Performance profiles (the higher the better)

Similar results to [W&S] for cluster-based scheduling vs list scheduling (static and dynamic), and our ETF is better

Duplex single-port: baselines on small data set, CCR=0.1, 1, 10

DSC-GLB-ETF not well suited to realistic communication model
Impact of number of parts, CCR, edge cut (big dataset)

- Relative performance of proposed heuristics compared to baseline BL-EST
- Left: $\text{CCR}=10$, $p = \{2, 4, 8, 16, 32\}$, number of parts $K = \alpha \times p$, where $\alpha = \{1, 2, 3, 4, 6, 8, 10, 12, 14, 16\}$ → New algorithms better than baseline - Pick $\alpha \leq 4$
- Right: Best $\alpha$ value in $\{1, 2, 3, 4\}$, $p = \{2, 4, 8, 16, 32\}$, $\text{CCR}=\{1, 5, 10, 20\}$ → significantly better results than BL-EST; BL-MACRO less stable, but outperforms all heuristics for large values of CCR

Smaller edge cut in DAG partitioning → better makespan 82% of the time (CCR=10)
Comparing all algorithms: small and medium datasets

**Small dataset, CCR={0.1, 1, 10}**

→ ETF remains the best with CCR=0.1, ETF-PART becomes better as soon as CCR=1, striking performance of *-MACRO for CCR=10

**Medium dataset, CCR=10, performance profiles of makespan and runtime**

→ ETF and ETF-based algorithms perform better but at the cost of much higher time complexity; overhead of partitioner negligible for BL-EST variants; XSBench graph: 9.5 seconds to partition, plus 0.5 second for BL-EST variants, while ETF takes 4759 seconds on two processors
Comparing algorithms: big dataset

CCR=\{1, 5, 10, 20\}, BL-EST variants only

- CCR=1, BL-EST performs best, BL-EST-BUSY is very close
- Increasing CCR: need to handle communications correctly
- CCR=5: 90% of all cases, BL-EST-BUSY’s makespan within 1.5× of best result; only 40% of cases for BL-EST
- BL-EST-MACRO works only for high values of CCR
Comparing algorithms: big dataset with many source nodes

CCR=$\{1, 5, 10, 20\}$, BL-EST variants only, with **many source nodes**

- More than 10% of the nodes are sources
- BL-EST performs badly
- BL-Macro even better: can start efficiently using more processors right from the start
Take-aways from experiments

- Proposed meta-heuristics significantly *improve baseline makespan*
- Benefit of *good partitioning* with minimum edge cut objective shows itself clearly, especially when CCR is high
- *-PART* and *-BUSY* behave consistently, scale well
- *-MACRO* has a higher variance, due to *global* view during scheduling: does not scale with number of processors, but outperforms all heuristics with large CCR
- *-MACRO* performs even better with large number of source nodes
Conclusion

Contributions

- Usage of **partitioning** to enhance data locality in list-based scheduling heuristics
- Acyclic partitions allow us to design specific list-based sched. techniques (identify **data locality**)
- Three proposed **generic meta-heuristics**, can be combined with any classical list-scheduling heuristic and acyclic partitioner
- Comparison with baseline heuristics: **striking results in terms of makespan improvement**
  
  *-Part (resp. *-Busy, *-Macro, best of three) algorithms achieve a makespan 2.6 (resp. 3.1, 3.3, 4) times smaller than bl-est (**big** dataset, **CCR = 20**, average over all processor numbers)

Future work

- Use **convex partitioning** instead of acyclic part.: less restrictive, hence exposes more parallelism
- Adaptation to **heterogeneous** processing systems
- Further use of the partitioner
Thanks...

... to the PPAM organizers (Roman and Ewa) for their kind invitation

... to my co-authors (Yusuf, Julien, Bora, Ümit)

For more information:

- Email: Anne.Benoit@ens-lyon.fr
- Visit: tda.gatech.edu