A scalable clustering-based task scheduler for homogeneous processors using DAG partitioning

M. Yusuf Özkaya[1], Julien Herrmann[1], **Anne Benoit**[1,2], Bora Uçar[1,2], Ümit V. Çatalyürek[1]

[1]School of Computational Science and Engineering, Georgia Institute of Technology, GA, USA

[2]CNRS and LIP, ENS Lyon, France

IPDPS 2019
May 20-24, 2019 – Rio de Janeiro, Brazil

Motivation

Context

- Applications modeled as a graph $G = (V, E)$:
  $\hookrightarrow$ Nodes: tasks with different completion times
  $\hookrightarrow$ Edges: data dependencies among tasks
- Need of efficient scheduling techniques

History

- List-based scheduling
- Clustering-based scheduling

Idea

- Build upon DAG partitioner to design scheduling heuristics accounting for data locality

Outline

Problem

Model

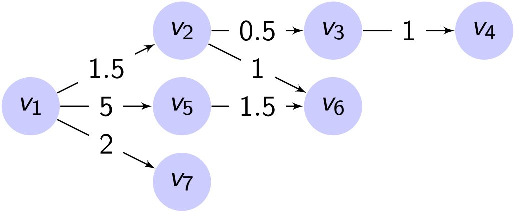
- Directed acyclic task graph: $G = (V, E)$
  w_i: task weight – $c_{i,j}$: communication cost
- Homogeneous platform:
  - p identical processors
  - fully connected homogeneous network
- Duplex single-port model: Each processor can, in parallel, without contention:
  - execute a task
  - send one data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

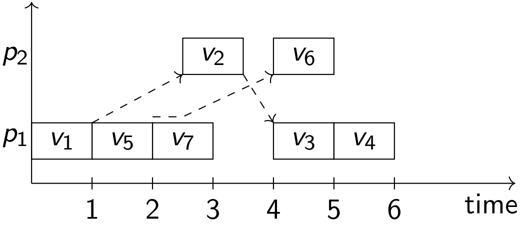
MinMakespan

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 $v_i \in V$, $w_i = 1$





Outline

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!

BL-EST: bottom level / earliest start time

Prioritizing phase

- Prioritizing tasks according to their bottom level:

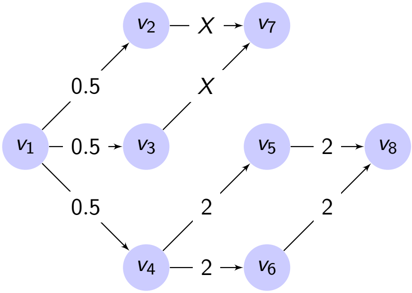
$$\mathtt{bl}(i) = w_i + \begin{cases} 0 & \text{if } \mathtt{Succ}[v_i] = \emptyset; \\ \max\limits_{v_j \in \mathtt{Succ}[v_i]} c_{i,j} + \mathtt{bl}(j) & \text{otherwise.} \end{cases} \tag{1}$$

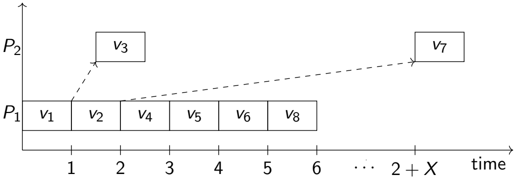
Assigning tasks to processors

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





BL-EST schedule

- 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

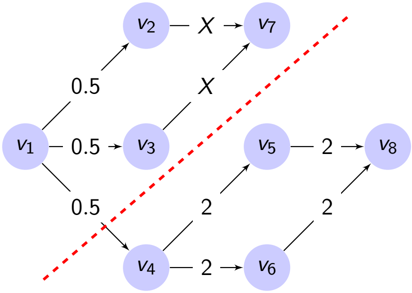
Principle

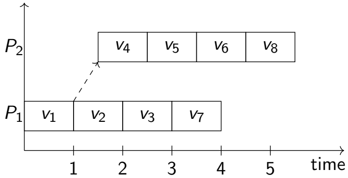
- Partition the DAG into $K > p$ parts to enhance data locality
- Weights of parts are balanced with a 10% ratio (other values give similar results)
- The edge cut is reduced
- The partition is acyclic (dependence graph for parts is acyclic)
- Use the global view of the partition in the list-based scheduling

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

*-Part





BL-EST-Part schedule

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?

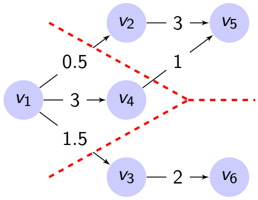
- Maintain list of busy processors (i.e., processors that have been assigned a task from a part but not all of them yet assigned)

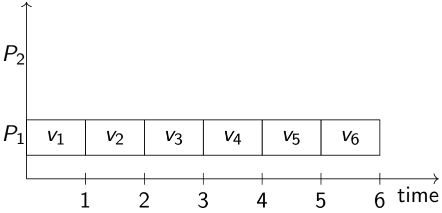
Assigning tasks to processors

Select ready task with highest priority:

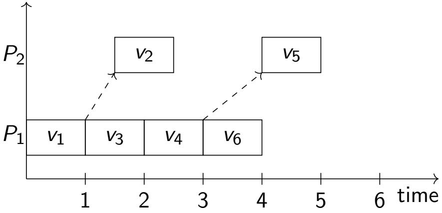
- If a task from the same part has already been assigned to a proc., map it onto the same proc.
- Else, if all processors are busy, behave like list-scheduler
- Else, behave like list-scheduler on non-busy processors only

- $p = 2$ and $K = 3$





BL-EST-PART schedule



BL-EST-BUSY schedule

*-Macro

Concept

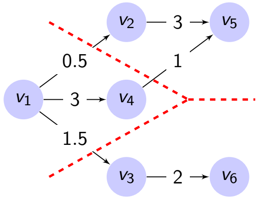
- Map a whole part before moving to the next one
- Priority of a part is the maximum bottom level of its tasks
- Maintain list of ready **parts**

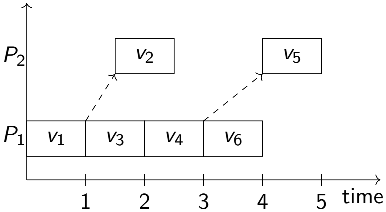
Assigning tasks to processors

- 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

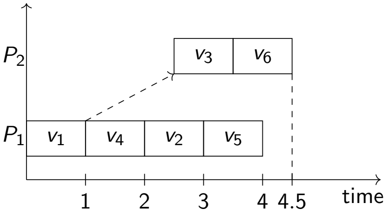
BL-EST-BUSY VS BL-MACRO

- $p = 2$ and $K = 3$





BL-EST-BUSY schedule



BL-MACRO schedule

Outline

Graph instances

Instances from the SuiteSparse Matrix Collection (denoted UFL):

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

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

Graph	$\lvert V \rvert$	$\lvert E \rvert$	Degree max.	Degree avg.	#source	#target
cholesky	1,030,204	1,206,952	5,051	2.34	333,302	505,003
fibonacci	1,258,198	1,865,158	206	3.96	2	296,742
quicksort	1,970,281	2,758,390	5	2.80	197,030	3
RSBench	766,520	1,502,976	3,074	3.96	4	5
Smith-water.	58,406	83,842	7	2.88	164	6,885
UTS	781,831	2,061,099	9,727	5.28	2	25
XSBench	898,843	1,760,829	6,801	3.92	5	5

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

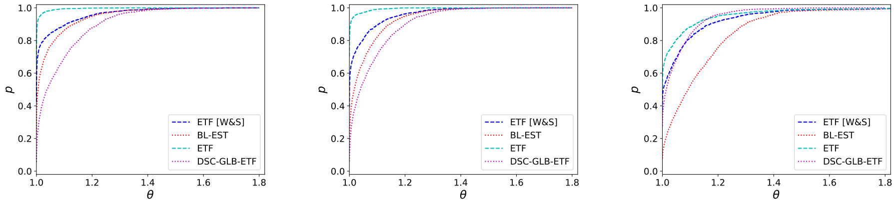
For a graph $G = (V, E)$, the CCR is formally defined as $CCR = \frac{\sum_{(v_i,v_j)\in E} c_{i,j}}{\sum_{v_i \in V} w_i}$

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

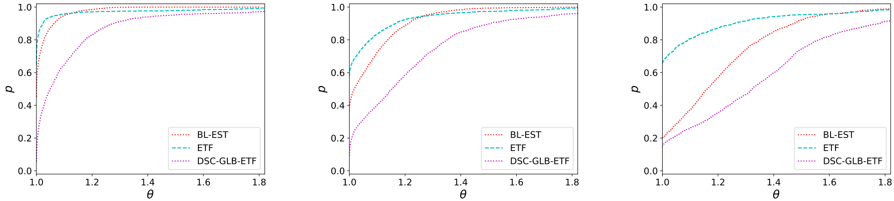
Communication-delay model vs. realistic model

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

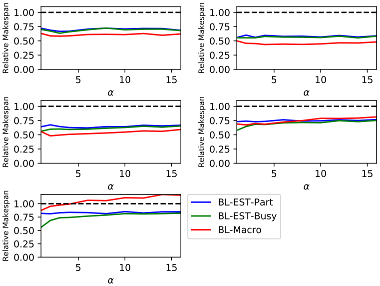
Duplex single-port: baselines on *small* data set, CCR$=0.1, 1, 10$

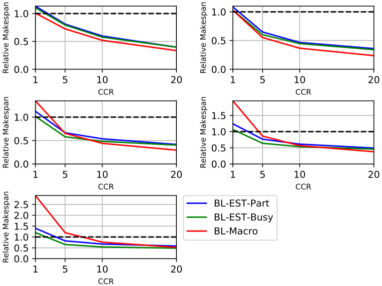


DSC-GLB-ETF not well suited to realistic communication model

Impact of number of parts, CCR, edge cut (big dataset)

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

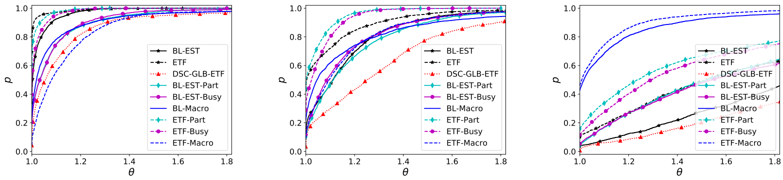




Smaller edge cut in DAG partitioning $\rightarrow$ better makespan 82% of the time (CCR=10)

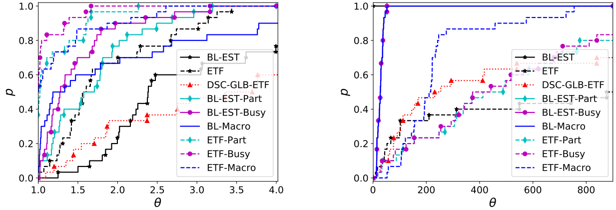
Comparing all algorithms: small and medium datasets

- *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

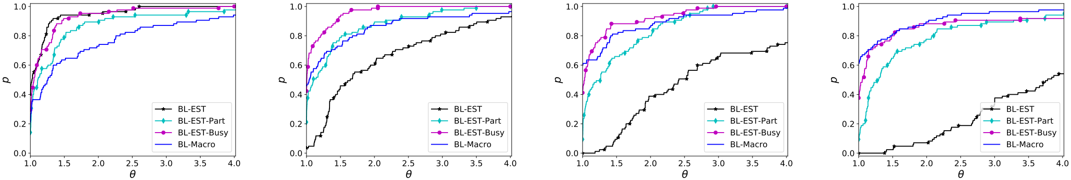


→ ETF and ETF-based algorithms perform better but at the cost of much higher time complexity; overhead of partitioner negligible for BL-EST variants; `XSBench` graph: 9.5 seconds to partition, plus 0.5 second for BL-EST variants, while ETF takes 4759 seconds on two processors

Comparing algorithms: big dataset

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

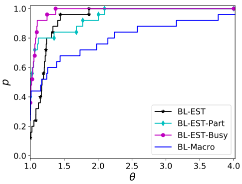
- CCR=1, BL-EST performs best, BL-EST-BUSY is very close
- Increasing CCR: need to handle communications correctly
- CCR=5: 90% of all cases, BL-EST-BUSY's makespan within 1.5× of best result; only 40% of cases for BL-EST
- BL-EST-MACRO works only for high values of CCR

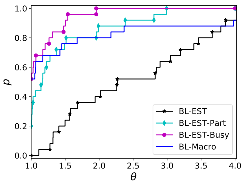


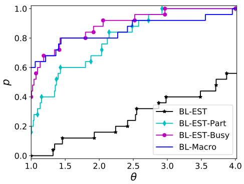
Comparing algorithms: big dataset with many source nodes

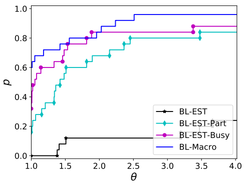
CCR=$\{1, 5, 10, 20\}$, BL-EST variants only, with many source nodes

- 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

Outline

Conclusion

Contributions

- Usage of partitioning to enhance data locality in list-based scheduling heuristics
- Acyclic partitions allow us to design specific list-based scheduling techniques, by identifying 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