Web supplementary material for the article: Online Scheduling of Task Graphs on Heterogeneous Platforms

Louis-Claude Canon, Loris Marchal, Bertrand Simon, and Frédéric Vivien

1 Omitted proofs of lower bounds on online algorithms competitiveness

Theorem 1. *If $k \geq 2$, no online algorithm has a competitive ratio smaller than τ for the $(Pm, Pk)|prec, online|C_{max}$ problem, even when the bottom level of each task is known. If preemption with migration is authorized, no online algorithm has a competitive ratio smaller than $\frac{\tau}{2}$. If $k = 1$, then we obtain the same bounds divided by a factor 2.*

Proof. The proof relies on the construction used to prove Theorem 1 in the main paper. We first assume that $k \geq 2$. For simplification, we rely on the construction for an integer τ but the modification easily extends to any rational τ.

We add $n\tau$ tasks U^j to the built graph, with $1 \leq j \leq n\tau$, where there is a dependence from U^j to U^{j+1} for each j. Each task has a CPU computing time equal to τ and a GPU computing time equal to 1, as tasks T_i^j. For each task T_i^j, we add a dependence from T_i^j to U^j. See Figure 1 for an illustration of the graph.

The longest path starting from any task T_i^j to an endpoint of the built graph then has a length equal to $n\tau - j + 2$: it is composed for instance of task T_i^j and tasks U^j to $U^{n\tau}$. Note that tasks T_*^j have multiple paths of length $n\tau - j + 2$, see Figure 1.

Therefore, for any j, the $k\tau$ tasks T_i^j have the same bottom level. So when $\mathcal{A}$ terminates task T_i^j, the adversary can choose whether T_*^j is equal to T_i^j or not, while respecting the bottom level furnished to $\mathcal{A}$. Then, the lower bound on the makespan reached by $\mathcal{A}$ (and $\mathcal{A}'$ if preemption with migration is allowed) still holds.

It remains to define the schedule $\mathcal{S}$ with the added tasks, and to show that its makespan is at most $(n+2)\tau$. We add another bucket B^U concerning a different GPU than B (as $k \geq 2$), starting at time 2τ and lasting $n\tau$ units of time, see Figure 2. Task U^j is scheduled in B^U at time $2\tau + j$. Note that for any ℓ, tasks $U^{(\ell-1)\tau}$ to $U^{\ell\tau}$ are executed after bucket B_ℓ, which contains tasks T_i^j for $(\ell-1)\tau < j \leq \ell\tau$. Therefore, every task T_i^j is terminated before task U^j is scheduled, so no precedence constraints are violated.

The lower bounds proved in Theorem 1 of the main paper are then unchanged.

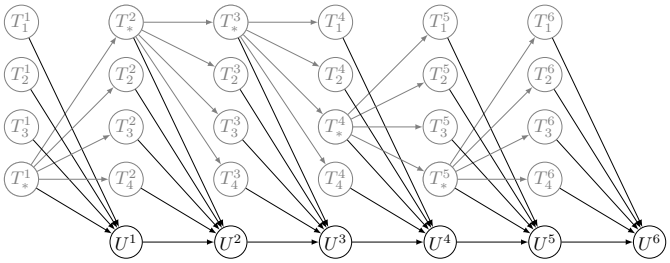

Figure 1. Example of built graph with $\tau = 2$, $k = 2$, $n = 3$. In gray, the tasks and dependences existing in the previous proof.

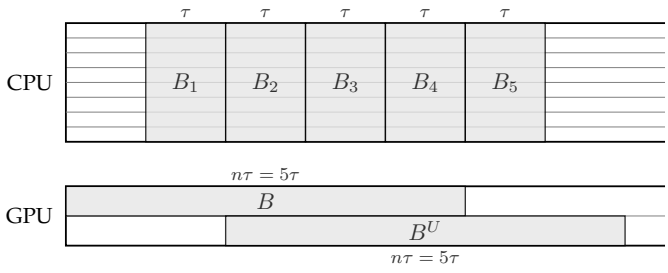

Figure 2. Buckets used by $\mathcal{S}$ with $n = 5$.

If $k = 1$, we define the bucket B^U on the unique GPU, starting at time $n\tau$ and terminating at time $2n\tau$. The makespan obtained by $\mathcal{S}$ is then twice longer, so the lower bounds obtained are twice smaller. □

Theorem 2. *No online algorithm has a competitive ratio smaller than $\frac{1}{2}\left\lfloor\sqrt{2\tau^*}\right\rfloor$ for the $(Pm, Pk)|prec, online|C_{max}$ problem, even when both the bottom level and the total weight of the descendants of each task is known. If preemption with migration is authorized, no online algorithm has a competitive ratio smaller than $\frac{1}{4}\left\lfloor\sqrt{2\tau^*}\right\rfloor$.*

- *L. Marchal and F. Vivien are with CNRS, INRIA and University of Lyon, LIP, ENS Lyon, 46 allée d'Italie, Lyon, France. E-mail: {loris.marchal,frederic.vivien}@ens-lyon.fr*
- *L.-C. Canon is with FEMTO-ST Institute – Université de Bourgogne Franche-Comté, 16 route de Gray, 25 030 Besançon, France. E-mail: louis-claude.canon@univ-fcomte.fr*
- *B. Simon is with University of Bremen, Bibliothekstr., 28359 Bremen. E-mail: bsimon@uni-bremen.de*

In these bounds, τ^* is the largest triangular integer not larger than τ. Recall that τ is a triangular integer if we have $\tau = 1 + 2 + \ldots + \left\lfloor \sqrt{2\tau} \right\rfloor$.

Proof. The proof relies on the construction used to prove Theorem 1, but using less phases and adding several tasks. We first assume that τ is a triangular integer larger than 1, which means that there exists an integer $q > 1$ such that $\sum_{i=1}^{q} i = \tau$. The exact value of q is $\frac{1}{2}\sqrt{1+8\tau} - \frac{1}{2} = \left\lfloor \sqrt{2\tau} \right\rfloor$. The graph built in this proof contains $q+1$ phases of $k\tau$ tasks each.

We add m tasks V_i^j to the built graph, with $1 \leq j \leq q$ and for each j, with $1 \leq i \leq (q+1-j)k\tau$. By definition of q, this indeed sums to $k\tau^2 = m$ additional tasks. All these tasks have a CPU computing time equal to τ and a GPU computing time equal to 1, as tasks T_i^j.

We now build the graph so that for each j, the $k\tau$ tasks T_i^j have the same number of descendants. For each $j \leq q$, we add dependences from every task T_i^j except T_*^j to every task $V_i^{j'}$ such that $j' \geq j$. See Figure 3 for an illustration of the graph, where all tasks T_*^j have been set to $T_{k\tau}^j$ for simplification, and where tasks sharing the same successors and predecessors have been agglomerated. In this example, we have $q = 3$, $\tau = 6$ and $m = 36$. For instance, T_1^3 has 6 new successors (tasks $V_{1\ldots6}^3$), T_1^2 has 18 new successors and T_1^1 has $m = 36$ new successors.

Let j be fixed. In this graph, the descendants of task T_*^j are tasks $U^{j'}$ for $j' \geq j$, tasks $T_i^{j'}$ for $j' \geq j$ and tasks $V_i^{j'}$ for $j' > j$. The descendants of any task T_i^j except T_*^j are tasks $U^{j'}$ for $j' \geq j$ and tasks $V_i^{j'}$ for $j' \geq j$. The difference is that task T_*^j has the $(q+1-j)k\tau$ tasks $T_i^{j'}$ as successors but not the $(q+1-j)k\tau$ tasks V_i^j. Therefore, at j fixed, the number of successors is the same for each task T_i^j.

So when $\mathcal{A}$ terminates a task $\frac{j}{i}$, the adversary can choose whether T_*^j is equal to T_i^j or not, while respecting the bottom level and number of descendants furnished to $\mathcal{A}$. Then, the lower bound on the makespan reached by $\mathcal{A}$ (and $\mathcal{A}'$ if preemption with migration is allowed) on each phase still holds. Therefore, $\mathcal{A}$ leads to a makespan of at least $(q+1)\tau$ and $\mathcal{A}'$ to a makespan of at least $\frac{1}{2}(q+1)\tau$.

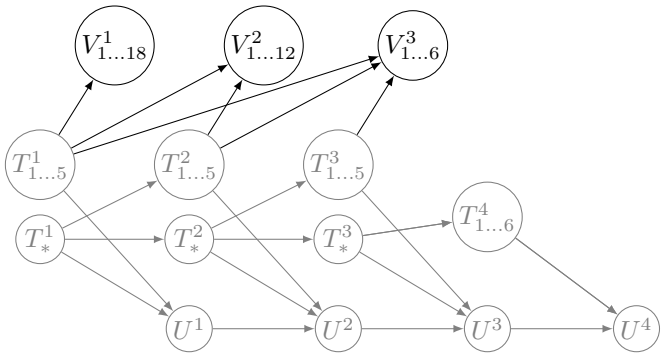

Figure 3. Example of built graph with $\tau = 6$, $q = 3$, $k = 1$, $m = 36$. In gray, the tasks and dependences existing in the previous proof.

It remains to define the schedule $\mathcal{S}$ with the added tasks, and to show that its makespan is at most $2\tau + q + 1$. The schedule $\mathcal{S}$ is similar to the one of the previous proof, except that tasks V_i^j are executed on CPU after tasks T_i^j. As there are m such tasks, this takes a time τ. To summarize, tasks T_*^j are executed on GPU in time q, then the remaining tasks T_i^j are executed on CPU in time τ, then, in parallel, tasks U^j are executed on GPU in time $q+1$ and tasks V_i^j are executed on CPU in time τ, see Figure 4. The makespan obtained is then equal to $\tau + q + \max\{\tau, q+1\} = 2\tau + q$. The last equality is valid as for $\tau \geq 3$, we have $q \leq \tau - 1$.

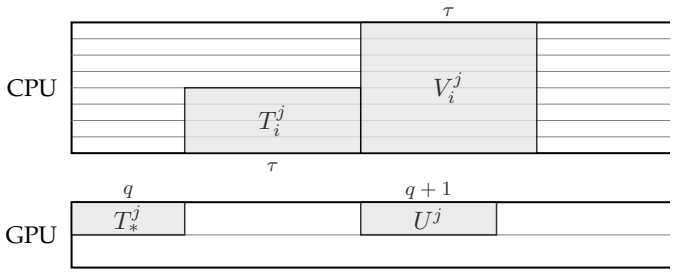

Figure 4. Shape of the schedule $\mathcal{S}$.

Recalling that $\tau = q(q+1)/2$, the competitive ratio of $\mathcal{A}$ is then at least:

$$\frac{(q+1)\tau}{2\tau+q} = \frac{1}{2}\frac{q(q+1)^2}{q(q+1)+q} > \frac{q}{2}\frac{(q+1)^2}{(q+1)^2} = \frac{q}{2} > \frac{1}{2}\left\lfloor \sqrt{2\tau} \right\rfloor.$$

Similarly, the competitive ratio of $\mathcal{A}'$ is at least $\frac{1}{4}\left\lfloor \sqrt{2\tau} \right\rfloor$.

If τ is not a triangular integer, or even not an integer, the same proof applies where q is the maximal integer such that $\sum_{i=1}^{q} i \leq \tau$. The exact value of the computing ratio obtained is then:

$$\frac{1}{2}\left\lfloor \sqrt{2\tau + \frac{1}{4}} - \frac{1}{2} \right\rfloor = \Theta\left(\sqrt{\tau}\right).$$

□

Remark 1. *When $4m \leq (k-2)^3$, no online algorithm has a competitive ratio smaller than $\lfloor\tau\rfloor + 1$, even when the bottom level of each task is known.*

Proof. We assume throughout this proof that $k \geq 2(\lfloor\tau\rfloor + 1)$, which holds if $4m \leq (k-2)^3$. This proof is adapted from the proof of Theorem 1 when τ is an integer, and where each occurrence of τ is replaced by $\lfloor\tau\rfloor + 1$.

Specifically, the graph built in this proof is composed of $n(\lfloor\tau\rfloor + 1)$ phases of $k\lfloor\tau\rfloor + 1$ tasks, along with a chain of $n(\lfloor\tau\rfloor + 1)$ tasks, for an arbitrarily large n. The CPU computing time of each task is equal to $\lfloor\tau\rfloor + 1$ and the GPU computing time is equal to 1. The dependences are built as in the proof of Theorem 1.

Any online algorithm needs a time $\lfloor\tau\rfloor + 1$ to schedule each phase. Indeed, in each phase, either $\lfloor\tau\rfloor + 1$ tasks are scheduled on a single GPU, or one task is scheduled on CPU. Therefore, the makespan obtained by any online algorithm on the total $n(\lfloor\tau\rfloor + 1)$ phases is at least $n(\lfloor\tau\rfloor + 1)^2$.

We now build an offline schedule similar to the one of Theorem 1, except that $k-1$ tasks of each phase are scheduled simultaneously on GPUs. Therefore, with the tasks of the additional chain, the GPUs are all busy (except at the beginning and the end of the schedule). As each phase contains $k\lfloor\tau\rfloor + 1$ tasks, it remains to schedule $k(\lfloor\tau\rfloor - 1) + 2$

tasks per phase on the CPUs. The objective is to schedule $\lfloor \tau \rfloor + 1$ phases simultaneously in each bucket, where the length of a bucket is equal to $\lfloor \tau \rfloor + 1$ units of time. The number of tasks per bucket is then:

$$\begin{aligned}(k(\lfloor \tau \rfloor - 1) + 2)\,(\lfloor \tau \rfloor + 1) &= k\left(\lfloor \tau \rfloor^2 - 1\right) + 2(\lfloor \tau \rfloor + 1) \\ &\leq m - k + 2(\lfloor \tau \rfloor + 1) \\ &\leq m.\end{aligned}$$

The last inequality comes from the assumption on k. Each task can be scheduled on one CPU so the relevant tasks of $\lfloor \tau \rfloor + 1$ phases fit into each bucket. Therefore, by similar arguments to the proof of Theorem 1, we conclude that the optimal schedule has a makespan at most $(n+2)(\lfloor \tau \rfloor + 1)$.

When n increases, we obtain the result. □

2 Omitted proofs on competitive algorithms

Lemma 1. *Consider a graph where tasks are allocated arbitrarily to CPU or GPU, and a schedule $\mathcal{S}$ computed by a list algorithm, of makespan C_{max}. Let W_c (resp. W_g) be the total load on the CPUs (resp. GPUs), and CP be the critical path length. Then, we have:*

$$C_{max} \leq \frac{W_c}{m} + \frac{W_g}{k} + \left(1 - \frac{1}{m}\right) CP.$$

Proof. We consider the path p defined as being the longest path (in terms of execution time in $\mathcal{S}$) that contains a task that terminates at time C_{max} in $\mathcal{S}$. By definition, the length of p is at most CP. In order to simplify the reasoning, we assume that there is a task T_0 of null processing time that is the predecessor of every task in the graph, and that this task belongs to p.

Consider a moment t when no task of p is being executed in $\mathcal{S}$. Let T_ℓ be the last task of p to be executed before t and T_n be the next task of p to be executed after t in $\mathcal{S}$. Note that both tasks always exist because T_0 is executed at the start of the graph and a task of p is executed at the end of the schedule $\mathcal{S}$. Suppose first that T_n is executed on CPU. As T_n is not scheduled immediately after T_ℓ, and the schedule has been obtained by a list algorithm, no CPU is idling between the termination of T_ℓ and the start of T_n. Symmetrically, if T_n is executed on GPU, no GPU is idling in this period.

Therefore, when no task of p is being executed, either all the CPU are busy or all the GPU are busy. The CPU (resp. GPU) can be all busy for at most a time of W_c/m (resp. W_g/k). And the tasks of p are executed during a time at most CP.

Hence, we have:

$$C_{max} \leq \frac{W_c}{m} + \frac{W_g}{k} + CP.$$

We can further refine this inequality. Let P_c (resp. P_g) be the processing time of the tasks of p on CPU (resp. GPU). Then, these processing times can be removed from the total loads in the inequality:

$$\begin{aligned}C_{max} &\leq \frac{W_c - P_c}{m} + \frac{W_g - P_g}{k} + P_c + P_g \\ &\leq \frac{W_c}{m} + \frac{W_g}{k} + \frac{m-1}{m} P_c + \frac{k-1}{k} P_g \\ &\leq \frac{W_c}{m} + \frac{W_g}{k} + \frac{m-1}{m} (P_c + P_g) \\ &\leq \frac{W_c}{m} + \frac{W_g}{k} + \frac{m-1}{m} CP.\end{aligned}$$

□

3 Simulations

In this section, we provide additional and more detailed simulation results.

In the STG data set, 45 graphs are generated by each random DAG generator (`layrpred`, `layrprob`, `samepred` and `sameprob`). Both `layrpred` and `layrprob` generators lead to graphs with nodes structured by layers, whereas `samepred` and `sameprob` lead to more intricate graphs. In contrast to their counterpart with the suffix `-prob`, generators with the suffix `-pred` specify the average number of predecessors for each task. Figure 5 shows that `layrpred` graphs from the STG data set yield the largest difference between QA/ER-LS and the best algorithms (HEFT and EFT). Additionally, RATIO is often better than QUICKEST. This suggests that using additional CPUs increases the efficiency and that `layrpred` graphs have some parallelism. `sameprob` graphs leads to opposite conclusions because QUICKEST performs well. Contrarily to `layrpred` graphs in which each layer becomes ready step by step, allowing the CPUs to execute some of the tasks without slowing down the GPUs, `sameprob` graphs have more intricate dependences that provide limited parallelism.

Figure 6 shows that MIXEFT behaves like QA when its parameter λ is smaller than 1, and rapidly changes to mimic EFT when the parameter increases and exceeds 1. This transition occurs for a lower λ for Cholesky instances than for STG and ad hoc ones.

Figure 7 shows the performance for various platform sizes for the Cholesky dataset. EFT is always the best online heuristic and its ratio to HEFT is never more than 1.2 (i.e., 20% worse than HEFT). This also applies to MIXEFT. Depending on the number of CPUs and GPUs, the other algorithms (QA, ER-LS, RATIO, and QUICKEST) follow one of the following three strategies: 1) all tasks on CPUs – this is the case for RATIO when $m/k = 20$; 2) POTRF tasks (the least accelerated tasks) on CPUs and other tasks on GPUs – this is the case for QA and ER-LS when $m/k \geq 5$, and RATIO when $3 \leq m/k \leq 10$; 3) all tasks on GPUs – all the other cases. This first strategy is the worst one except when there are many tasks and CPUs, and a single GPU. In this case, it outperforms the third strategy because the instances present a large parallelism for which CPUs can be exploited. The second strategy is often inefficient for small instances because POTRF tasks are on the critical path and benefit from being accelerated on the GPUs. Finally, the last strategy significantly deviates from EFT only for low k and large number of tasks, which suggests that it is advantageous to exploit CPUs for large graphs when there are few GPUs.

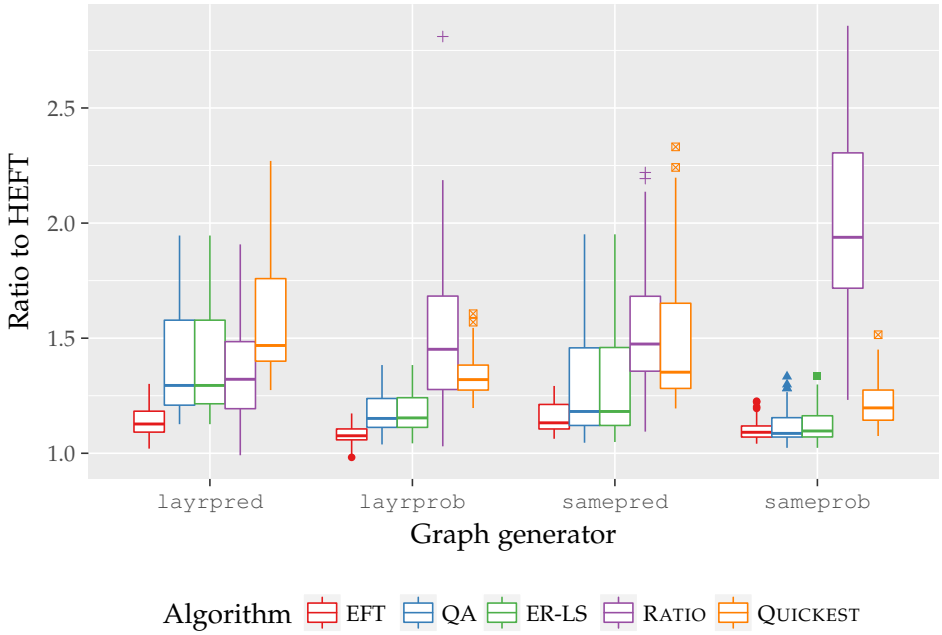

Figure 5. Ratios of the makespan over HEFT for EFT, QA, ER-LS, RATIO, and QUICKEST with $m = 20$ CPUs and $k = 2$ GPUs on random instances with $n = 300$ tasks from the STG data set. MIXEFT is not shown because it performs exactly as EFT.

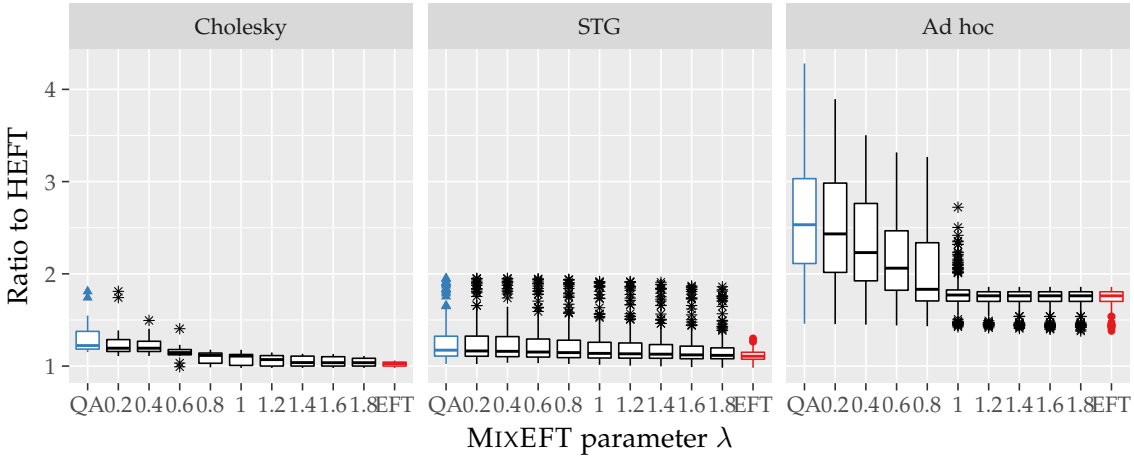

Figure 6. Ratios of the makespan over HEFT for QA, MIXEFT, and EFT with $m = 20$ CPUs and $k = 2$ GPUs on 14 Cholesky, 180 STG, and 300 ad hoc instances. ER-LS, RATIO, and QUICKEST are discarded.

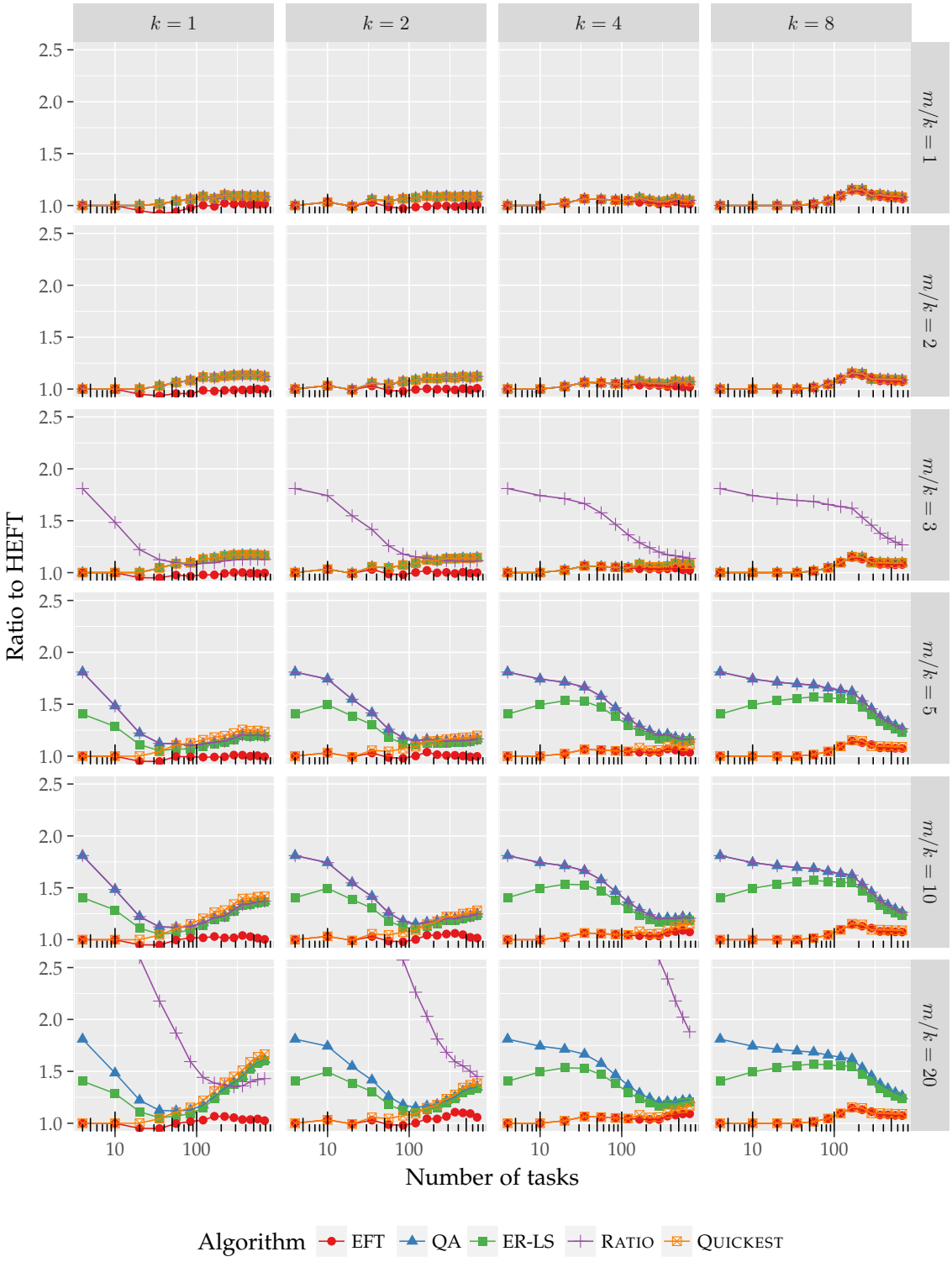

Figure 7. Ratios of the makespan over HEFT for EFT, QA, ER-LS, RATIO, and QUICKEST on Cholesky instances. MIXEFT is not shown because it performs exactly as EFT. In the bottom-right plot, RATIO does not appear because its ratio is too large.