



Trade-Off Between Performance, Fault Tolerance and Energy Consumption in Duplication-Based Taskgraph Scheduling

Patrick Eitschberger¹(✉), Simon Holmbacka², and Jörg Keller¹

¹ Faculty of Mathematics and Computer Science, FernUniversität in Hagen, Hagen, Germany

{patrick.eitschberger, jorg.keller}@fernuni-hagen.de

² Faculty of Science and Engineering, Abo Akademi University, Turku, Finland
sholmbac@abo.fi

Abstract. Fault tolerance in parallel systems can be achieved by duplicating task executions onto several processing units, so in case one processing unit (PU) fails, the task can continue executing on another unit. Duplicating task execution affects the performance of the system in fault-free and fault cases, and its energy consumption. Currently, there are no tools for properly handling the three-variable optimization problem: Performance \leftrightarrow Fault Tolerance \leftrightarrow Energy Consumption, and no facilities for integrating it into an actual system. We present a fault-tolerant runtime system (called RUPS) for user defined schedules, in which the user can give their preferences about the trade-off between performance, energy and fault tolerance. We present an approach for determining the best trade-off for modern multicore architectures and we test RUPS on a real system to verify the accuracy of our approach itself.

Keywords: Scheduling · Fault tolerance · Energy efficiency
Trade-off · Power modeling · Optimization · Runtime system

1 Introduction

Fault tolerance is important for parallel systems like manycores and grids, where a permanent failure of a processing unit (PU), resulting from either a hardware or software fault, might occur during the execution of a scheduled parallel program.

The schedules of parallel programs can be created statically, prior to execution with the help of a task graph that represents the tasks and dependencies between them. To maximize performance in static schedules, it is critical to minimize the length of a schedule, the so-called makespan. However, integrating fault tolerance techniques typically results in performance overhead. This leads to increasing makespans. One kind of fault tolerance is the task duplication where for each task a copy – a so-called duplicate – is created on another PU. In case of a failure, the duplicate is used to continue the schedule execution. The performance of the system in the fault case will then benefit from the

duplicates, since the progress of the schedule can seamlessly be continued by the tasks' duplicates. Another issue emerging especially in recent years is the problem of minimizing the energy consumption. Duplicating tasks requires additional resources because the task is actually executing simultaneously on various PUs. In the fault-free case this is regarded as energy wasting. The energy consumption is also affected by scaling down the clock frequency of a PU. By executing at different clock frequencies, the makespan is affected by the altered performance, and the energy consumption is affected by the altered power dissipation. This leads to a three-variable trade-off decision to be made between Performance PE , Energy Consumption E , and Fault tolerance FT .

There are several approaches in the literature for two-dimensional optimizations in the area of performance, energy and fault tolerance for various parallel platforms and with different fault tolerance techniques, e.g. in [3, 10, 12, 13, 15–17, 20]. Although the optimization for all two-dimensional combinations is well researched, the three-dimensional optimization is rarely addressed. There exist a few exceptions that focus on real-time systems where tasks have to be executed in predefined time frames or within a certain deadline. Therefore, PE in corresponding approaches is the major objective. For example Cai et al. [6] present a greedy heuristic to reduce the energy consumption in fault-tolerant distributed embedded systems with time-constraints. Another approach is presented by Alam and Kumar [1]. They assume that only one specific transient fault could occur during the execution of a task. Tosun et al. [19] present a framework that maps a given real-time embedded application under different optimization criteria onto a heterogeneous chip multiprocessor architecture. In all of these approaches, the focus typically lies on transient faults, where checkpointing or backup mechanisms are used to circumvent a fault. In our approach, we focus on permanent faults and present scheduling strategies that combine all three criteria without a real-time constraint. Hence, in this work a broader range is considered, which is not yet addressed in previous work.

We propose a solution for the three-variable optimization problem for cases where the user can inform the scheduler about his preferences. We firstly extend an energy efficient and fault tolerant scheduler by integrating new scheduling strategies that can be set according to the user's preferences. Secondly, to demonstrate the influence of the user preferences we present a runtime system RUPS for scheduling parallel applications with adjustable degrees of fault tolerance on grids, computing clusters or manycore systems. The runtime system utilizes a pre-optimized static schedule with the desired characteristics and trade-off between PE , E and FT . To obtain the energy consumption for a selected schedule, we create a realistic power model based on experiments for an actual real-world processor. Several example models for different platforms are created, and we show that their accuracy is sufficient to predict the requirements for the trade-off between PE , E and FT . Thirdly, with the power model and the given schedule, we can construct the trade-off map to be used during system planning, and we show how the PE , E and FT parameters can affect the planning decisions of parallel fault-tolerant applications. Our results indicate that the power

model is accurate and that the experiments match the predictions. Finally the trade-off map shows in detail the relations between PE , E and FT .

The remainder of this paper is structured as follows. In Sect. 2 the trade-off problem is discussed. Sections 3, 4 and 5 present the extended scheduler, the runtime system and the power model. In Sect. 6 the results are presented and analyzed. In Sect. 7, we conclude and give an outlook on future work.

2 The Trade-Off Problem

A combination of all three objectives is possible in general, but there does not exist an overall optimal solution. In this context the degree of FT is rated by the overhead in performance (and energy) that results from the fault tolerance techniques in both the fault-free and fault case. Therefore, a compromise between the optimization criteria must be made. While one criterion is improved, either one or both of the others are worsened.

When we focus on PE of a schedule, it is dependent on the mapping of the tasks. The more an application can be parallelized the better is the performance. Additionally, modern processors support several frequencies at which a processor can run. Thus, tasks should be accelerated as much as possible, i.e. use the highest supported frequency of a PU. In contrast, a more parallelized application results in fewer gaps between tasks and thus in fewer possibilities to include duplicates without shifting successor tasks. This results in a high performance overhead in case of a failure, e.g. a low FT . Additionally, running on a high frequency typically leads to a high E . When we focus on FT , duplicates should be executed completely in the fault-free case and available but unused PUs should also be considered for mapping duplicates to minimize performance loss in case of a fault. In this case, duplicates may lead to shifts of original tasks and thus to a low PE in the fault-free case. In terms of E , both executing duplicates completely and using available PUs not necessary for the original tasks result in a high E . Is the focus put on E , low frequencies and short duplicates are preferable. But low frequencies lead to low PE and short duplicates to a high performance overhead (FT) in case of a failure.

In addition, the main focus of a user varies in different situations. For example, in a time critical environment, PE is the most important criterion next to FT . Thus, in this situation PE and also FT is usually favored over minimizing E . Another situation is, that a failure occurs extremely rarely and thus E is becoming more important. Other examples exist in mobile devices where E is the most important criterion next to PE . The main focus is therefore put on E and PE while FT is neglected. However, the alignment of the optimization is very situational and ultimately depends highly on the user preferences.

3 Fault Tolerant and Energy Efficient Scheduling

We start by reviewing the ideas of [10] and briefly introduce our previous work. Then we present two new strategies to improve either FT or E of the schedules.

3.1 Previous Approach

Fechner et al. [10] provides a fault-tolerant duplication-based scheduling approach that guarantees no overhead in a fault-free case. Starting from an already existing schedule (and taskgraph), each original task is copied and its duplicate (D) is placed on another PU than the original task so that in case of a failure the schedule execution can be continued. We assume homogeneous PUs and a fail-stop model, where a failure of a PU might result from a faulty hardware, software or network. We only consider one failure per schedule execution.

If an original task has finished it sends a commit message to its corresponding D so that it can be aborted. Schedules often comprise several gaps between tasks resulting from dependencies. D s can be placed either in those gaps or directly between two succeeding tasks. To avoid an overhead in a fault-free case, in all situations where a D would lead to a shift of all its successor tasks only a placeholder, a so called dummy duplicate (DD) is placed. DD s are only extended to fully D s in case of a failure. To reduce the communication overhead, D s are placed with a short delay, so called slack. Thus, either the results of an original task are sent to its successor tasks or the results of the corresponding D , but not both. Figure 1(a) illustrates an example taskgraph. For a better understanding the communication times and the slack are disregarded. Figure 1(b) and (c) show the resulting schedules of two strategies, the first uses only DD s the second uses D s and DD s.

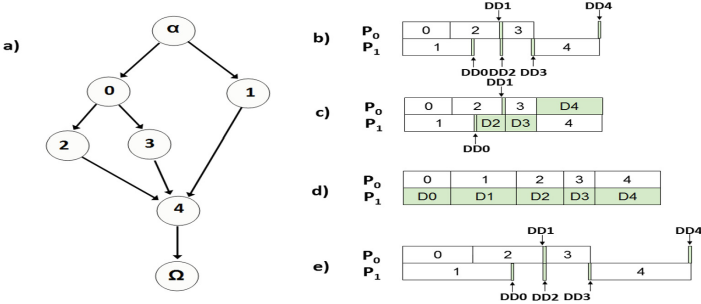


Fig. 1. (a) simplified taskgraph, (b) strategy 1: use only DD s, (c) strategy 2: use D s and DD s, (d) strategy 3: use half of PUs for original tasks, the others for D s, (e) strategy 4: select a lower frequency for original tasks

In our previous work [8], we show the importance of considering communication times for the placement of D s and DD s. We present in [9] an extension to improve E of schedules by calculating a buffer for each task. It indicates how much a task could be slowed down by scaling down the frequency of the corresponding PU without prolonging the makespan. Frequencies are then set to the lowest possibles to fill the buffers. We assume a general power model like explained in [2] and use continues normalized frequencies for our predictions.

3.2 Extensions

We extend the scheduler for supporting also a concrete power model (that we describe in Sect. 5) with discrete frequencies and we include two new simple strategies. In our first strategy, we use a simple list scheduler to create schedules with respect to the dependencies from the corresponding taskgraph. Instead of using as many PUs as possible, only half of the available PUs are used for the placement of original tasks and the remaining PUs are used to include Ds (see Fig. 1(d)). With this strategy we try to focus on *FT*.

In our second strategy, the user can set a frequency level with which the original tasks should run before including Ds and DDs (see Fig. 1(e)). Thus, we leave the mapping of all original tasks as it is and change only the runtime of the tasks by using the selected frequency level. Then, the start times of tasks are corrected according to the dependencies given by the taskgraph.

4 Runtime System

RUPS (Runtime system for User Preferences-defined Schedules) is a scheduling tool for parallel platforms with features allowing the user to input various preferences e.g. *PE*, *E* or *FT* in the schedule. Schedules are then created with the RUPS tool – optimized for the user defined preference in question. RUPS consists of four main parts illustrated in Fig. 2(a). The processor details are extracted in Part 1 and passed to the scheduler (Part 3), which in turn optimizes the schedule based on the processor parameters and user preferences (Part 2). Finally, the schedule is passed to the runtime system (Part 4), and scheduled on the processor. In this section, we describe the details of these four parts.

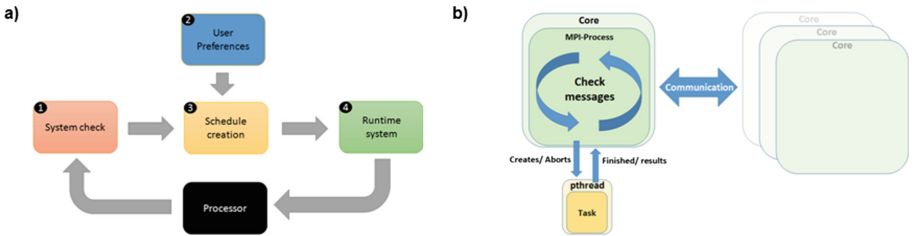


Fig. 2. Overview of (a) RUPS and (b) the runtime system

4.1 System Check Tool

At the first use of RUPS, it has to be initialized once with the system check tool to adjust the power model for the processor used. This tool measures the power consumption of the processor for all supported frequencies and for a different number of cores under full load. We measure the power consumption for 10 seconds (s) with a sampling rate of 10 milliseconds (ms). All cases are repeated

five times to compensate high power values that could occur due to unexpected background processes. Between each case, all cores are set to the lowest frequency in idle mode for 5s to reduce the rise in temperature of the processor and thus the influence on the power consumption. Then the averaged results of the measure points for each case are used as values for the power model.

4.2 Scheduler and User Preferences

The scheduler consists of two main parts. One part for the schedule creation and one part for the simulation of generated schedules to predict the energy consumption in different situations. It supports several strategies for the placement of Ds/DDs and the user can set different options for the behavior during the scheduling process like setting the time for a slack, considering unused cores for the placement of Ds/DDs, simulating failures or only creating the fault-tolerant schedules. The simulator can then be used to simulate for each task in the schedule one failure. It can also handle task slow downs that result from a high load level of a PU. For more details we refer to [8,9].

4.3 Runtime System

The runtime system is based on ULFM-MPI [5], a fault tolerant extension of Open-MPI. For each core a MPI-process is created, that reads the schedule and taskgraph information from files and generates a task queue (sorted by the starting times of tasks). Then, a while loop is executed as long as there is a task in the queue. The loop is used for a polling mechanism that reacts and handles the communication (via messages), starts a task if possible and also aborts a task if necessary. The task execution is separated from the communication process by a (posix) thread. Data transfers between tasks are simulated by only sending the message header, that includes next to others the information about the start time of the sending operation and the transfer time. This simplification has a neglectable effect on the results, as the energy consumption of the communication is not considered in the measurements and models.

Figure 2(b) illustrates a short overview of the runtime system. We simulate a failure by exiting a MPI-process just before the corresponding task is started. The other processes are then informed about the failure by an error handler. We integrate a testing mode where one additional MPI-process is started to measure the energy consumption with the help of Intel RAPL. The measurement process measures the energy with a sample rate of 10 ms.

5 Power Model

To predict the energy consumption for a schedule, an appropriate power model for the processor is necessary. Basically, a model is a simplified representation of the reality. The complexity of a model increases significantly with its accuracy. As the power consumption of a processor depends on several factors, like the

temperature, instruction mix, usage rate and technology of the processor, there exist numerous approaches in the literature to model the power consumption of a processor with varying complexities and accuracies, like in [4, 7, 11] or [18].

In general the power consumption can be subdivided into a static part, that is frequency-independent and a dynamic part, that depends both on the frequency and on the supply voltage.

$$P_{processor} = P_{static} + P_{dynamic} \quad (1)$$

The static power consumption consists of the idle power P_{idle} and a device specific constant s , that is only needed when the processor is under load.

$$P_{processor} = \begin{cases} P_{idle} + s + P_{dynamic} & \text{if under load} \\ P_{idle} & \text{else} \end{cases} \quad (2)$$

The dynamic power consumption is typically modeled as a cubic frequency function [2], as the frequency and voltage are loosely linearly correlated¹. Additionally the supply voltage and thus the dynamic power consumption depends on the load level of a core. As we only consider fully loaded cores or cores that are in idle mode (at the lowest frequency) the influence of a load level can be given by a parameter $w \in \{0, 1\}$. If we assume a homogeneous multi-core processor with n cores, a simple power model for the dynamic part can be given by the following equation, where a , b and β are device specific constants, i is the core index and $f_{curr,i}$ is the current frequency of core i :

$$P_{dynamic} = \sum_{i=0}^{n-1} w_i \cdot \beta (f_{curr,i}^3 + a \cdot f_{curr,i}^2 + b \cdot f_{curr,i}) \quad (3)$$

Only if a core runs at a higher frequency under full load, the dynamic part of the power consumption for the processor is considered.

5.1 Model Validation

To prove the accuracy of the power model, we used three different computer systems with Intel processors as test platforms:

1. Intel i7 3630qm Ivy-Bridge based laptop
2. Intel i5 4570 Haswell based desktop machine
3. Intel i5 E1620 server machine

To construct the power model, we extracted the power values by physical experiments using the Intel RAPL tool. As described in Sect. 4.1, we measured the power consumption for each frequency combination for 10s with a sampling rate of 10 ms and repeated all measurements five times. We test the power

¹ For a given voltage there is a maximum frequency and for a desired frequency there is a minimum voltage required.

model for six different workload scenarios: ALU-, FPU-, SSE-, BP- and RAM-intensive workloads and for a combination of these tests as mixed workload. The measured power values were used to construct the power model for each platform and scenario. The architecture specific tuning parameters (s , β , a and b) in Eqs. 2 and 3 were then determined using a least squares analysis.

Table 1 shows exemplary the individual parameters for each platform for a mixed workload after fitting the physical measurements to Eqs. 2 and 3 and optimizing the tuning parameters. The results of the least squares analysis for the other tests only differ slightly from the mixed workload scenario. The different parameters for the power model can be determined and saved in advance and used for several classes of applications with a specific workload type dominating. Then the power consumption can be measured during the execution of the first application and compared to the different power models to find the best suitable for the whole class.

Table 1. Values of the architecture specific tuning parameters for a mixed workload

	P_{idle}	s	β	a	b
i7 3630	3.781 W	1.29 W	0.340 W/Hz ³	-3.42 Hz	5.88 Hz ²
i5 4570	5.976 W	0.42 W	0.091 W/Hz ³	1.02 Hz	12.08 Hz ²
i5 E1620	8.728 W	3.83 W	0.344 W/Hz ³	-2.87 Hz	6.13 Hz ²

Table 2 shows the difference between the data and model as the maximum and average deviation. The maximum deviation was lowest using the desktop CPU (i5 4570). The reason for having a less exact fit using the server (i5 E1620) and laptop CPU (i7 3630) is because of the significantly higher power output using the turbo boost on these CPUs, which is more difficult to fit to the curve than the more smooth power curve of the i5 4570 CPU. However, with a low average error value we consider this model feasible for our experiments. In Fig. 3 we present exemplary the resulting power curve for the server test platform for the real data and for the model.

Table 2. Difference between the data and model as error values squared from Fig. 3

	i7 3630	i5 4570	i5 E1620
Avg. deviation	1.09%	0.84%	1.13%
Max deviation	15.56%	7.28%	17.07%

5.2 Real-World Evaluation

For our real-world evaluation we used the server system as a common platform for clusters and grids. We tested 922 schedules in total that are related to 40 taskgraphs with random properties and between 19 and 24 tasks (see Sect. 6).

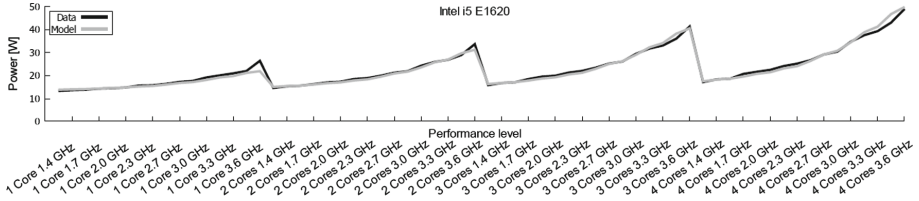


Fig. 3. Power consumption and power model for the server platform

For each taskgraph we first let the already existing schedule run without any changes and thus without any failures. Then, the fault-tolerant schedules that result from the first strategy – using only DDs – (see Sect. 3) were calculated and executed by the runtime system. And we let run all fault-tolerant schedules with a simulated failure at each task by exiting the corresponding MPI-Process directly before the task execution started.

We validate the accuracy of the prediction by comparing the predicted energy values that result from the scheduler with the real measurements of the runtime system. In Fig. 4 we present the predicted and real energy consumption for all schedules. With a maximum deviation of 7.14% and 1.64% on average, our prediction fits the reality quite well.

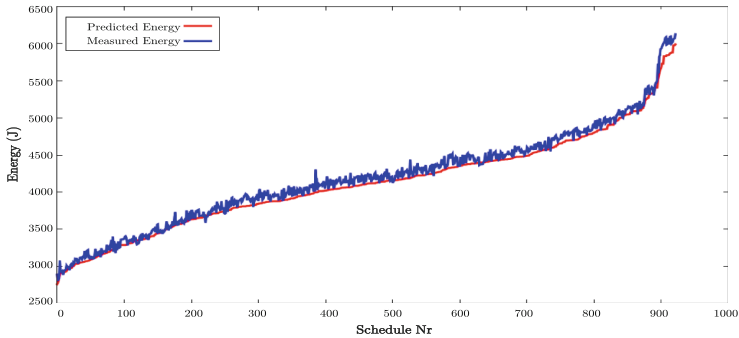


Fig. 4. Predicted and measured energy consumption (for a mixed workload)

6 Experimental Results

For our experiments we used a benchmark suite of synthetic taskgraphs [14] with 36000 performance optimal schedules, that can be subdivided by the number of PUs (2, 4, 8, 16 and 32), the number of tasks (7–12, 13–18 and 19–24), the edge density and length and the node and edge weights. The schedules were generated with a PDS-algorithm (Pruned Depth-first Search). To find optimal solutions in an acceptable time, the search space is reduced by pruning selected paths in the search tree. As the scheduling problem is NP-hard, there have been

some taskgraphs where no optimal schedule could be found even after weeks of computation. Those taskgraphs are excluded from this study. As seen in Sect. 5.2, our system model closely reflects the real system in terms of energy consumption. We used this fact to simulate nearly 34500 of the given schedules using the RUPS system. We evaluate the trade-off between PE , FT and E with four scenarios in which we use the four strategies from Sect. 3. These scenarios reflect system setups with one of the three parameters as inherently dominating. This choice will give a wide range of experiments with the extreme corner cases covered, and everything between them. The following scenarios were used for our simulation, where we do not consider the turbo frequency to avoid throttling effects:

- (A) Strategy 1: Use only DDs and start with the highest supported frequency (3.5 GHz). In this scenario we focus on PE .
- (B) Strategy 2: Use Ds and DDs and start with the highest supported frequency (3.5 GHz). This scenario mainly targets on PE , but also on FT .
- (C) Strategy 3: Create the schedules with a simple List Scheduler that uses half of the PUs for original tasks, the other for the Ds and start with the highest supported frequency (3.5 GHz). Here the focus is on FT .
- (D) Strategy 4: Select a lower frequency for original tasks and start with frequency level 7 (2.3 GHz). With this scenario we try to focus on E .

To visualize the trade-off between PE , FT and E the results of the four strategies are relatively related to the following estimated upper and lower boundaries for each criterion (see Table 3) where m is the makespan in cycles, m_{seq} is the makespan, when all tasks are running in sequence and m_{ft} is the makespan in case of a failure. $p_{max} \in PU$ is the maximum number of PUs used and $f_{highest/lowest}$ is the highest or lowest frequency respectively.

Table 3. Upper and lower boundaries for PE , FT and E

	Best case	Worst case
PE	$\frac{m_{seq}}{p_{max} \cdot f_{highest}}$	$\frac{m_{seq}}{1 \cdot f_{lowest}}$
FT	$\frac{m_{ft} - m}{m} \cdot 100 = 0\%$	$\frac{2 \cdot m - m}{m} \cdot 100 = 100\%$
E	$\frac{m_{seq}}{p_{max} \cdot 2.3 \text{ GHz}}$	$\frac{m_{seq}}{1 \cdot f_{lowest}}$

Focusing on performance PE , the best solution is to parallelize an application as much as possible. Furthermore, the highest available frequency $f_{highest}$ should be selected, if the system in use supports different frequencies. A lower bound for the performance can be achieved by running all tasks in sequence on one PU with the lowest possible frequency f_{lowest} .

While a schedule is either fault-tolerant or not, the fault tolerance FT is rated by the performance overhead in case of a failure. Therefore, when focusing on the fault tolerance the best solution is to copy the whole schedule and execute it simultaneously (completely independent) to the original one on other PUs.

Then, both the performance, i.e. the makespan m_{ft} in case of a failure and in a fault-free case m are equal. Accordingly, the performance overhead results to zero percent. However, the worst solution is when the schedule is not fault-tolerant and a failure occurs directly before the end of the schedule execution. Then, the whole schedule has to be repeated on $p - 1$ PUs and the makespan $m_{ft} = 2 \cdot m$ in case of a failure is at least doubled in comparison to the fault-free case m . Thus, the performance overhead in case of a failure results in 100%.

While the estimation of upper and lower bounds for PE and FT are independent of a certain system, E depends highly on the system in use. Therefore, we calculated the best and worst energy consumption of the i5 E1620 processor with the measured power values from the system check tool for a perfectly divisible workload. In this case, the most efficient frequency is at 2.3 GHz. The boundaries for E in Table 3 have to be multiplied with the corresponding power values from the system to get the energy consumption in Joule.

In Fig. 5 the results of all scenarios are presented. For a better illustration we only show the results for systems with 4 PUs (in total 6500 schedules with different properties). But the results for the other number of PUs (2, 8, 16 and 32 PUs) are similar with respect to the overall trends. They differ only slightly by small shifts. The left column of the figure presents for all scenarios (A, B, C and D) the trade-off between E and PE , the middle column between E and FT and the right column between PE and FT .

Starting with scenario A, we can see that a better performance also leads to a better energy consumption. With a performance of nearly 100% the energy consumption goes down to around 5% (related to the best and worst cases from the boundaries). This behavior seems to be related to the high idle power of the system compared to the dynamic power. The higher the idle power is, the better it is to run on a high frequency, e.g. at the highest like here. If we now focus on the trade-off between E and FT we can see, that the lower the energy consumption in the fault free case is, and thus the higher the performance of the schedule, the higher is also the performance overhead in case of a failure. This behavior results from the decreasing number and size of gaps within a schedule, when improving the performance. Because then each DD leads directly to a shift of its successor tasks. The trade-off between PE and FT shows directly the same behavior. The higher the performance the higher is also the performance overhead. In scenario B we used Ds and DDs for the fault tolerance. We see that the left part ($E \leftrightarrow PE$) of the figure is more spread. This indicates, that especially for a lower performance more gaps can be filled with Ds. This leads to an increased energy. The middle part of the figure ($E \leftrightarrow FT$) shows the resulting improvement of the performance overhead in case of a failure. And also on the right part ($PE \leftrightarrow FT$) we see the slightly shift of all results to the left. In scenario C we try to use a simple strategy to get a good FT result. Looking on the left side, we see that the performance is much lower and the energy consumption is much higher than for scenario A and B. As the performance does not change in case of a failure, the middle and right part of the figure are empty. Scenario D shows the results for schedules that run with a lower frequency (frequency level 7, 2.3 GHz). Here we

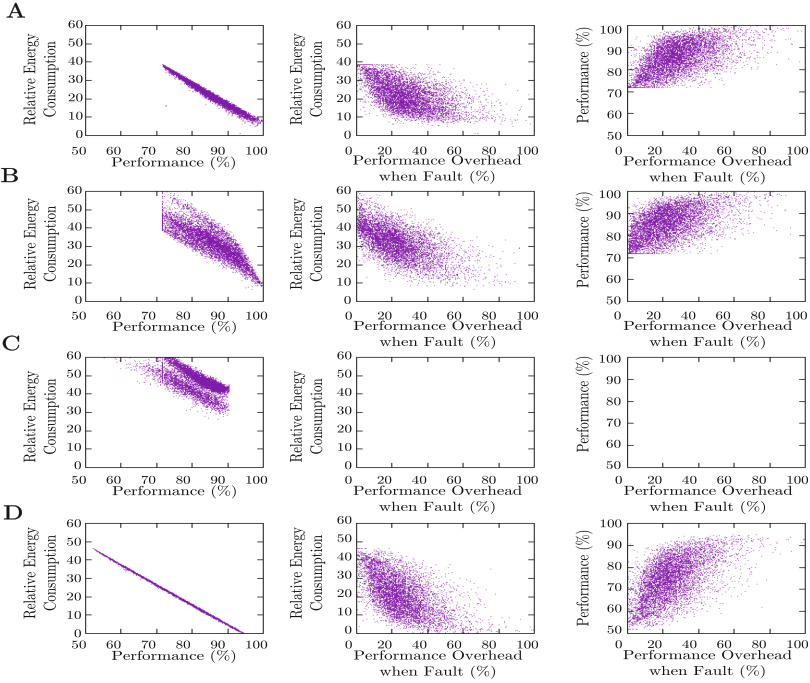


Fig. 5. Results when scheduling according to scenarios A, B, C, D showing: relative energy consumption (lower is better), performance (higher is better), performance overhead when fault (lower is better)

can see that running on a lower frequency results in a better energy consumption, but only if the performance increases. Then we can reach nearly the best energy consumption. The other both trade-offs are the same like for scenario A. They are just a little bit stretched.

We could show, that there does not exist any overall solution for that three-variable problem without giving up at least one of the three parameters. Thus, the decision on which parameter the main focus lies must be made by the user.

Exemplary user preferences and favored strategies are summarized in Table 4. As seen in Table 4, various user preferences are represented by the proposed

Table 4. User preferences and favored strategies.

User preferences		Favored strategies
Fault-free case	Fault case	
PE (E)	–	S1
PE (FT)	PE	S2
FT	PE	S3
E	E	S4

strategies. Next to major objectives, also minor criteria can be considered, resulting in a variety of possible solutions with reasonable results. The worsening of criteria that are not focused is moderate. Thus, the investment for improving favored objectives is low. In addition, the strategies can be hidden from users that do not have any background knowledge about scheduling, so that they only have to give their preferences by selecting a combination of objectives. Then, the corresponding strategies can be chosen automatically by the scheduler.

Please note that the user preference might not only depend on the user, but also on the taskgraph, schedule and deadline at hand. If e.g. the deadline is close to the makespan of the corresponding schedule (i.e. all cores must execute tasks at one of the highest frequencies), then energy savings in the fault-free case are hardly possible, and the user will be better off to focus on other preferences. If the deadline is farther away from the makespan, then energy efficiency can be considered. If the deadline is hard, then the preference will be on keeping the deadline even in each possible fault-case, and energy will only be a secondary preference.

7 Conclusions

We presented a method to quantitatively handle the trade-off between PE , E and FT when scheduling taskgraphs onto parallel machines with DVFS. We also presented a scheduling and execution tool called RUPS that implements these schedules on real machines. Fault tolerance is achieved by adding task duplicates in parallel with the original tasks; affecting both the energy consumption and the time-to-recovery in case a fault occurs in the system. This tool is intended to bridge the gap between Performance, Energy efficiency and Fault tolerance (PE , E , FT), which are the parameters the scheduling decisions are based on. We demonstrate the trade-off between PE , E and FT with four corner case studies, which can heavily impact the decisions needed during system planning. The experiments on real machines also provide evidence on the accuracy of the underlying performance and energy model used in the scheduler. As future work, we plan to extend the scheduler for tolerating more than one failure per schedule and for integrating reconnected PUs after a failure. We also plan to investigate in more strategies that focus on the corner cases E and FT and to integrate real transfer datas next to the message headers into the runtime system.

References

1. Alam, B., Kumar, A.: Fault tolerance issues in real time systems with energy minimization. *Int. J. Inf. Comput. Technol.* **3**(10), 1001–1008 (2013)
2. Albers, S.: Energy-efficient algorithms. *Commun. ACM* **53**(5), 86–96 (2010)
3. Aupy, G., Benoit, A., Renaud-Goud, P., Robert, Y.: Energy-aware algorithms for task graph scheduling, replica placement and checkpoint strategies. In: Khan, S., Zomaya, A. (eds.) *Handbook on Data Centers*, pp. 37–80. Springer, New York (2015). https://doi.org/10.1007/978-1-4939-2092-1_2

4. Basmadjian, R., de Meer, H.: Evaluating and modeling power consumption of multi-core processors. In: Proceedings of the 3rd International Conference on Future Systems: Where Energy, Computing and Communication Meet (e-Energy 2012), pp. 1–10 (2012)
5. Bland, W.: User level failure mitigation in MPI. In: Caragiannis, I., Alexander, M., Badia, R.M., Cannataro, M., Costan, A., Danelutto, M., Desprez, F., Krammer, B., Sahuquillo, J., Scott, S.L., Weidendorfer, J. (eds.) Euro-Par 2012. LNCS, vol. 7640, pp. 499–504. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36949-0_57
6. Cai, Y., Reddy, S.M., Al-Hashimi, B.M.: Reducing the energy consumption in fault-tolerant distributed embedded systems with time-constraint. In: 8th International Symposium on Quality Electronic Design (ISQED 2007), pp. 368–373 (2007)
7. Cichowski, P., Keller, J., Kessler, C.: Modelling power consumption of the Intel SCC. In: Proceedings of the 6th Many-Core Applications Research Community Symposium (MARC 2012), pp. 46–51 (2012)
8. Eitschberger, P., Keller, J.: Efficient and fault-tolerant static scheduling for grids. In: Proceedings of the 14th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC 2013), pp. 1439–1448 (2013)
9. Eitschberger, P., Keller, J.: Energy-efficient and fault-tolerant taskgraph scheduling for manycores and grids. In: an Mey, D., et al. (eds.) Euro-Par 2013. LNCS, vol. 8374, pp. 769–778. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54420-0_75
10. Fechner, B., Hönig, U., Keller, J., Schiffmann, W.: Fault-tolerant static scheduling for grids. In: Proceedings of the 13th IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems (DPDNS 2008), pp. 1–6 (2008)
11. Goel, B., McKee, S.A.: A methodology for modeling dynamic and static power consumption for multicore processors. In: Proceedings of the 30th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2016), pp. 273–282 (2016)
12. Hashimoto, K., Tsuchiya, T., Kikuno, T.: Effective scheduling of duplicated tasks for fault tolerance in multiprocessor systems. *IEICE Trans. Inf. Syst.* **85**, 525–534 (2002)
13. Hongxia, W., Xin, Q.: Dynamic replication of fault-tolerant scheduling algorithm. *Open Cybern. Syst. J.* **9**, 2670–2676 (2015)
14. Hönig, U., Schiffmann, W.: A comprehensive test bench for the evaluation of scheduling heuristics. In: Proceedings 16th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2004), pp. 437–442 (2004)
15. Kianzad, V., Bhattacharyya, S., Ou, G.: CASPER: an integrated energy-driven approach for task graph scheduling on distributed embedded systems. In: Proceedings of the 16th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP 2005) (2005)
16. Pruhs, K., van Stee, R., Uthaisombut, P.: Speed scaling of tasks with precedence constraints. *Theory Comput. Syst.* **43**(1), 67–80 (2008)
17. Singh, J., Auluck, N.: DVFS and duplication based scheduling for optimizing power and performance in heterogeneous multiprocessors. In: Proceedings of the High Performance Computing Symposium (HPC 2014), pp. 22:1–22:8 (2014)
18. Takouna, I., Dawoud, W., Meinel, C.: Accurate multicore processor power models for power-aware resource management. In: Proceedings of the 9th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC 2011), pp. 419–426 (2011)

19. Tosun, S., Mansouri, N., Kandemir, M., Ozturk, O.: An ILP formulation for task scheduling on heterogeneous chip multiprocessors. In: Levi, A., Savaş, E., Yenigün, H., Balcısoy, S., Saygın, Y. (eds.) ISCS 2006. LNCS, vol. 4263, pp. 267–276. Springer, Heidelberg (2006). https://doi.org/10.1007/11902140_30
20. Zhao, L., Ren, Y., Xiang, Y., Sakurai, K.: Fault-tolerant scheduling with dynamic number of replicas in heterogeneous systems. In: 12th IEEE International Conference on High Performance Computing and Communications (HPCC), pp. 434–441 (2010)