Layered Reconfigurable Architecture for Autonomous Cooperative UAV Computing Systems

Grzegorz Chmaj(✉) and Henry Selvaraj

University of Nevada, Las Vegas, Las Vegas, USA
{grzegorz.chmaj,henry.selvaraj}@unlv.edu

Abstract. Cooperative processing in UAV swarms requires efficient architectural and algorithmic solutions to maximize the operational speed and life time span. Solutions need to include communication, data gathering, data processing and general management. In this paper, we address all these needs and we also present a reconfigurable approach to the UAV swarm cooperative processing with maximum extent of decentralization. UAVs with multiple devices (such as cameras, radars), and multiple processing units (CPUs and reconfigurable FPGAs) onboard are considered. We present main elements of the architecture and two reconfiguration algorithms used in the process of management of FPGA chips. Results are evaluated using the dedicated simulation framework and demonstrate the efficiency of the proposed solutions.

Keywords: Reconfigurable · Autonomous · UAV · Cooperative · Computing

1 Introduction

One of the main needs that appear in today's world of interconnected devices is to be able to throw the application onto some processing structure that might be unknown at that time, and let the structure do the processing according to the application specification. This type of process exists both in stationary powered and battery powered systems. In both cases minimizing the operational electrical energy expenditure is important, however in the latter case it is often critical. Application Specific Integrated Circuits are most efficient when it comes to processing power and efficiency, however they are tied to the specific functions or applications that makes them hard to use in universal systems. Reconfigurable FPGA (Field Programmable Gate Array) chips offer the way to program their structure with user-defined design, therefore becoming application-specific chip with very good efficiency. In our research, we focus on the UAV-based cooperative computing systems [1, 2]. They operate on battery power or onboard combustible engines, what makes their energy consumption critical, especially during missions. Decentralization is another issue that must be considered [3]. These are systems working in the field that are distant from the base station, thus they should not rely on one main component, as if it would become inoperable and render the entire swarm down [4]. Next key factor is the flexibility of the UAV swarm. Typically each vehicle is equipped with some extra sensing devices such as cameras, radars etc., they

J. Świątek and J.M. Tomczak (eds.), *Advances in Systems Science*, Advances in Intelligent Systems and Computing 539, DOI 10.1007/978-3-319-48944-5_15

also must have computing capabilities. Use of general purpose CPUs is not efficient for heavy computations required to process the massive amount of data coming from the sensing devices. Therefore, we are using FPGAs to program them with the designs that are currently required, to get efficient processing. They can be reprogrammed for other uses anytime, also *on the fly* reprogramming is supported and is one of key factors of the electrical efficiency of our proposed architecture.

Multiple reconfigurable systems exist. A comprehensive survey of them, including features, challenges, concepts, and applications is presented in [5]. Work presented in [6] shows the development methods and tools used in embedded reconfigurable systems. The reconfigurable computing design patterns are described in [7]. [8] provides a general overall description of reconfigurable systems, together with the characteristics of reconfigurable logic. Spatial computation, configurable data path, distributed control and distributed resources were stated as the fundamental differences between traditional processing architectures and reconfigurable logic approach. Authors of [9] present the methods of workload distribution over processing resources, which are considered to be multi-core CPUs, GPUs (Graphical Processing Units), PPU (Physics Processing Unit) and FPGAs, among others. Presented work focuses on handling applications' requirements in the high-level design process, including the identified concurrencies used to achieve load balancing and efficient task distribution. The behavior of distributed reconfigurable systems is often researched using simulation frameworks. Results in [10] include modeling complex reconfigurable nodes, processor configurations and tasks along with general purpose processors and offers multiple metrics for the evaluation. [11] presents the Open Control Platform for reconfigurable distributed control systems that provides the coordination of distributed interaction among hierarchically organized units, and supports the dynamic reconfiguration.

2 System Operation

2.1 Layered Architecture

In order to achieve maximum flexibility in operation, and also in possible redesign of the system – the presented architecture is proposed to be using the layered approach. The following layers are used: (1) *application* (2) *processing* (3) *units* (4) *communication*. The application is defined only at its level, and does have no knowledge about lower layers. The processing layer analyzes the application and manages the processing (control, coordination, results). The processing layer uses units as processing resources: sends the tasks and collects results related to these tasks. Units use the underlying communication layer to exchange information and do any communication. Each layer is connected to others with interfaces. Such layered structure allows replacing any layer without modifying the remaining layers (e.g. using communication layer as TCP/IP, Bluetooth, ZigBee etc., or using the same application on various systems). The proposed layered structure is shown in Fig. 1.

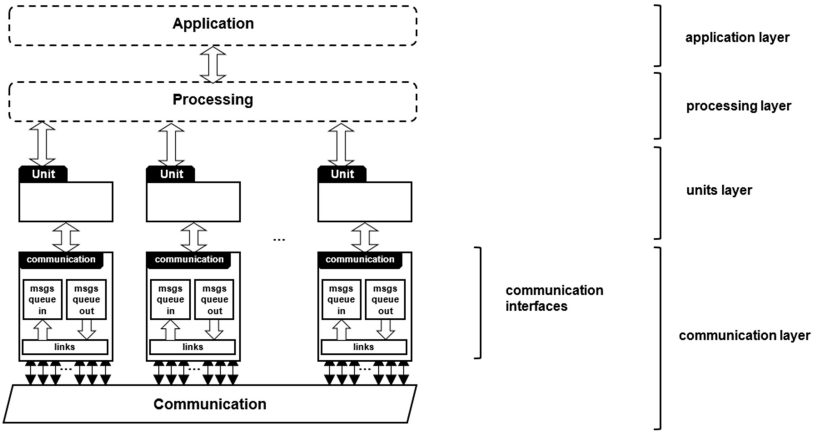


Fig. 1. Layered structure

2.2 General Structure

The cooperative computing system consists of V multiple units v that are autonomous with their decisions, i.e. the role of the centralized algorithm is minimized. Therefore, there is a need to design autonomous local algorithms that will control the local unit operation to the maximum extent. The decision making includes two cases: (1) decisions are taken just based on the local data, without getting the remote data (2) decisions are taken based on local and remote data (remote data must be fetched). The general operation of the system is depicted in Fig. 2. Two main phases are indicated: DP_PHASE_INIT and DP_PHASE_OPERATION.

During the DP_PHASE_INIT stage, the initial setup is done: the entry point is created, roles are assigned to units with the particular special role ROLE_CONTROL. This makes the cooperative system set up and ready for DP_PHASE_OPERATION. During 2nd phase, each unit can register an application that will be further processed on multiple units that participate in the group. The system allows each unit to register its own application, therefore there may be multiple applications running in the system concurrently, and each unit may process many different tasks belonging to different applications. System operates until the defined *end condition* is satisfied. During DP_PHASE_OPERATION roles of the units can be changed and/or reassigned.

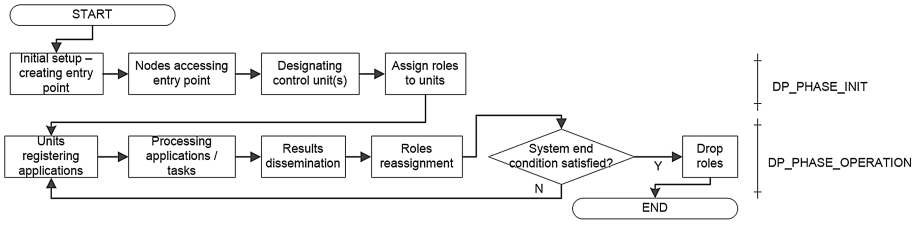


Fig. 2. The general system operation

Application is defined as a set of tasks, functions and data $A = \{T, F, D\}$. Functions operate on data, and are coordinated by operations. Data is not continuous and divided into data parts (if possible). This way such application can be executed in a distributed manner, where multiple functions can be executed on multiple units, and operate on various data parts. An example of such application A is the target location application (*TLA*) for UAV mission team. Set of functions: f_1 = *capture camera image*, f_2 = *capture IR camera image*, f_3 = *read radar data*, f_4 = *read GPS data*, f_5 = *request information from the other unit*, f_6 = *analyze terrain image*, f_7 = *analyze IR terrain image*. Set of tasks: t_1 = *determine position,* t_2 = *analyze terrain fragment,* t_3 = *validate terrain fragment with peers,* t_4 = *determine the next search area for the team.* UAVs are the units and are equipped with various electronic devices (cameras, GPS, radars etc.). As mentioned earlier, the ROLE_CONTROL must be present at least at one unit that serves as the database and control during the mission. A unit that starts the *target location* application (not necessarily the ROLE_CONTROL), registers the TLA at ROLE_CONTROL and therefore gets the ROLE_TASK_OWNER assigned. Remaining units that have ROLE_BASIC by default, also know the ROLE_CONTROL unit and get the information about TLA application from it. Further, they communicate directly with the task owner of the TLA and get the functions to execute according to their local resources (e.g. function f_2 can be executed only at the unit that has the IR camera installed). The proposed DPRS architecture allows dynamic (re) assignment of the functions during the operation. Given unit v_1 can execute task t_2 using functions f_1, f_4 and f_6 at the time T, finish t_2 execution at the time $T + T_1$ and then start the execution of task t_4 at the time $T + T_2$. This minimal set of operational elements provides a great flexibility required for the heterogeneous environment of cooperating units.

Each unit is considered to be a device with the processing capability (e.g. embedded system, PC computer, IoT device etc.). The processing capabilities are nowadays appearing in the form of multiple processors controlled by one operating system (server hardware, but also other architectures), processors with multiple cores, or single-chip single-core processor that is logically divided into processing threads by the operating system. Therefore, the unit v considered in this work is modeled as the device equipped with multiple processing devices – processors p such as CPUs, DSPs etc. These processors are general purpose – i.e. they are not designed for any specific application. The proposed approach allows using the reconfigurable FPGA chips, which can be programmed with some specific functions and thus becoming the application-specialized chips then. Hardware FPGA chips available on the market are programmed with bitstreams that are compiled hardware designs and model the hardware structure. This way, programming an FPGA reflects modeling its internal hardware structure. Market solutions also allow reprogramming on the fly, multiple functions can be programmed at the same time (if the internal chip space is sufficient) and chips handle thousands of reprogramming cycles. All these features are used in the proposed architecture.

Tasks are modeled as XML files that describe the relations between functions and data, also how the results are used. XML is also used to model any data structure in the system and all the relations. Functions f can appear in two forms: (1) programs compiled to the binary executable code (multiple platforms allowed) (2) definition in the

form of script program (Ruby, Python, and similar). If a unit executes the processing of task *t*, it must fetch the functions required by this task – in the form that matches unit's architecture. For the FPGA processing devices, executing a function *f* in such device *p* requires programming the device *p* first, using the bitstream representing the function *f*. Application-specific programmed FPGAs provide more efficient processing, however require time and other resources to get the bitstreams and program them into the chip structure. Each unit is autonomous, but is participating in the team and cooperates with others units. This mechanism, along with the application structure as described above provides a cooperative processing structure with maximum flexibility, able to process multiple applications in the distributed manner with minimum centrality (thus more reliability for failures). The proposed system is especially useful for systems where: (1) several units have different capabilities and need to part and join the team structure constantly (UAV missions); (2) structures with multiple concurrent applications that need to be often switched / turned off / resumed, and benefit from efficient matching the structure to current application / needs (Internet of Things structures).

2.3 Reconfiguration

Each unit runs multiple processes, also each role assigned to the unit is served by one or more process. The very top-level scheme of the node operation is shown in Fig. 3. Two phases already described before (NODE_PHASE_INIT and NODE_PHASE_ROLES) present the operations that unit performs when the entire system remains in these stages. The NODE_PHASE_SCHEDULER is used to execute all the scheduled operations that are not part of schemes /algorithms. Such operations include, among others: leaving the system, new role assignment, role drop, etc. Operations may be scheduled anytime in the advance. The focus of the presented work is the reconfiguration schemes their energy cost.

Figure 4 presents the operational scheme for ROLE_RECONFIGURABLE, that is assigned to every unit that is equipped with FPGA chip(s). The scheme assigned to ROLE_RECONFIGURABLE works as follows for processor *p*. The reconfiguration algorithm AL_RECONFIGURE_FPGA is executed regularly for *p*, and the period of the execution is determined by RECONFIGURATION_PERIOD value. RECONFIGURATION_PERIOD value can be either determined once, before the system

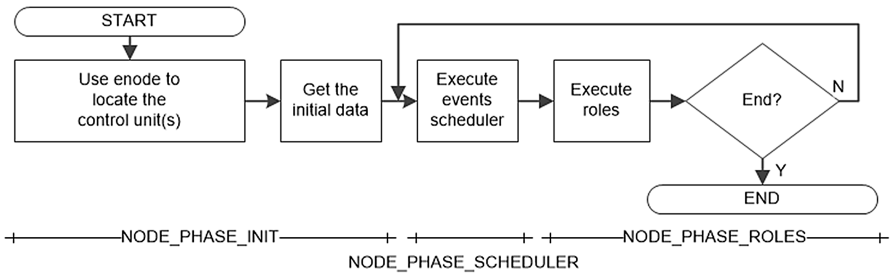


Fig. 3. Top-level operation of the unit

operation starts, or changed dynamically during system operation. In the work presented in this paper, the former way is used. The variable *reconf_counter* is set to RECONFIGURATION_PERIOD and then decremented each time the ROLE_RECONFIGURE_FPGA scheme is executed. Once the value reaches zero, the algorithm performs the *reconfiguration attempt*.

If *reconf_counter* = 0, but the related FPGA processor still executes a function, then the procedure waits till the function execution ends. Once the reconfiguration attempt is started, the AL_RECONFIGURE_FPGA algorithm is executed to determine if the reconfiguration should be done for the processor. This determination is done based on local knowledge and additional metrics that can be requested from other nodes (especially from ROLE_CONTROL and ROLE_TASK_OWNER. Based on all the inputs, the decision is taken whether the reconfiguration process should be started immediately or the next attempt should be taken next time the *reconf_counter* reaches zero.

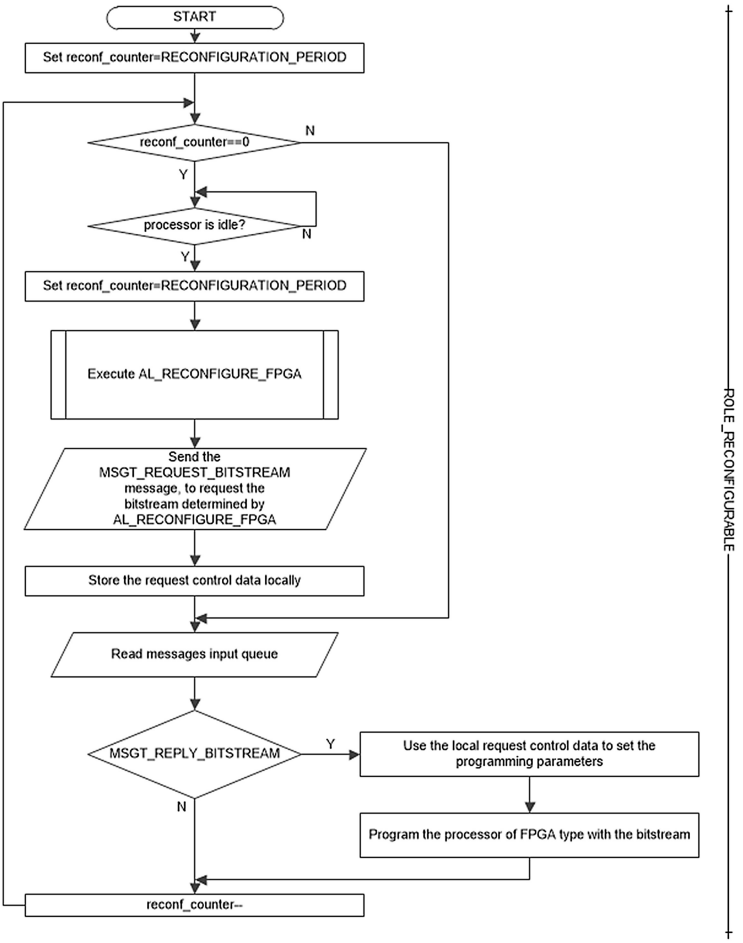
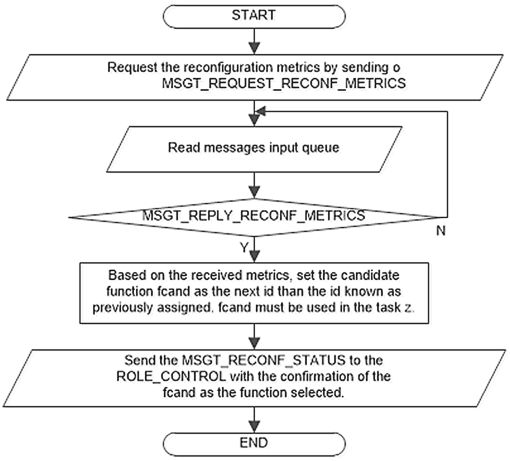


Fig. 4. Operation of ROLE_RECONFIGURABLE



a) operation

Algorithm. AL_RECONFIGURE_FPGA_1

1. Request the reconfiguration metrics from ROLE_CONTROL
1.1. Attach the current information about the unit and processors
2. The unit v requesting reconfiguration metrics for application z from ROLE_CONTROL unit sets the f_{cand} as:

$$f_{cand} \rightarrow f_{next,z}$$

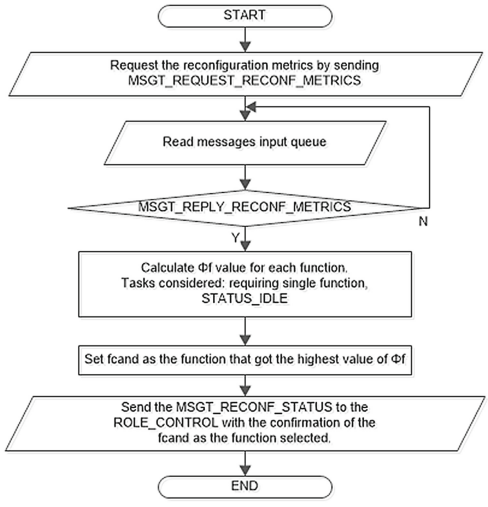
where $f_{next,z}$ determines the next function compared to previously assigned f_{cand}, that is

$$f_{next,z}: \sum_t^T x_{t,z} n_{t,f_{next}} > 0$$

3. Send the information MSGT_RECONF_STATUS to the ROLE_CONTROL about programmed function.

b) pseudocode

Fig. 5. AL_RECONFIGURE_FPGA_1 algorithm



a) operation

Algorithm. AL_RECONFIGURE_FPGA_2

1. Request the reconfiguration metrics from ROLE_CONTROL
1.1. Attach the current information about the unit and processors
2. The unit v requesting reconfiguration metrics for application z (active and considered application) from ROLE_CONTROL unit, calculate the Φ_f value for each function f:

$$\Phi_f = \sum_b^B x_{b,z} n_{b,f}$$

where t status is *STATUS_IDLE*

where $\sum_f^F n_{b,f} = 1$

3. Select $f_{cand} \rightarrow \forall \Phi_f: f_{cand} \geq \Phi_f$
4. Send the information MSGT_RECONF_STATUS to ROLE_CONTROL for function to be programmed.

b) pseudocode

Fig. 6. AL_RECONFIGURE_FPGA_2 algorithm

Two reconfiguration algorithms are used and compared. The following notation is used to describe their operation: $x_{t,z} = 1$ if task t belongs to application z, 0 otherwise; $n_{t,f} = 1$ if task t requires function f, 0 otherwise; AL_RECONFIGURE_FPGA_1 works

according to the diagram Fig. 5(a) and pseudocode Fig. 5(b). The main idea of this algorithm is to provide the even distribution of the functions programmed in the FPGAs over the system. AL_RECONFIGURE_FPGA_2 works according to the diagram Fig. 6 (a) and pseudocode Fig. 6(b). This algorithm takes the popularity of the requirement into consideration (how many tasks require a function *f*), considering tasks that require single function. Descriptions are using multiple message types: MSGT_REQUEST_RECONF_METRICS – message to request all the data, known to the ROLE_TASK_OWNER and/or ROLE_CONTROL that can be useful for determining reconfiguration; MSGT_REPLY_RECONF_METRICS – the reply for MSGT_REQUEST_RECONF_METRICS; MSGT_RECONF_STATUS – an update message to the ROLE_CONTROL unit.

3 Experimentation Results

The most important efficiency metric for the cooperative processing system is the electrical energy efficiency, as the UAV systems often are built using devices that rely on the battery power. Thus lowering the electrical energy consumption increases the operational timespan, also for the specific UAV systems reduces the frequency of a vehicle to be forced to come back to the ground base for recharging. The following elements of the electrical energy expenditure were included: E_GETTING_TASKS – process of getting the task definition to the unit, including control data and incoming transmission costs; E_GETTING_FF – getting the definitions of the functions from ROLE_TASK_OWNER; E_GETTING_EXT_DATA – acquiring any values that are needed for task processing, such as remote data from other units; E_PROCESSING_TASK – computational process (executing task); E_BASIC_COST – all costs including configuration, idle operations, and others not included in the remaining elements; E_OUTBOUND_TRAFFIC – cost of outgoing data transmission; E_FPGA_PROGRAMMING_BITSTREAM – cost of programming the bitstream into FPGA; E_FPGA_DOWNLOAD_BITSTREAM – cost of downloading the bitstream files. E_FPGA_DECIDE_RECONF – cost of ROLE_RECONFIGURATION except AL_RECONFIGURE_FPGA and programming bitstream;

The total cost is then:

$$E = \sum_{v}^{v} \sum_{s}^{s} \sum_{p}^{p} (v^{s}_{\mathrm{E\_GETTING\_TASKS}} + v^{s}_{\mathrm{E\_GETTING\_FF}} + v^{s}_{\mathrm{E\_GETTING\_EXT\_DATA}} + p^{s}_{\mathrm{E\_PROCESSING\_TASK}} + v^{s}_{\mathrm{E\_OUTBOUND\_TRAFFIC}} + v^{s}_{\mathrm{E\_FPGA\_DOWNLOAD\_BITSTREAM}} + p^{s}_{\mathrm{E\_FPGA\_DECIDE\_RECONF}} + p^{s}_{\mathrm{E\_FPGA\_PROGRAMMING\_BITSTREAM}}) + Tp_{\mathrm{E\_BASIC\_COST}}$$

$$C^{AL1}_{AL2} = \frac{AL2 - AL1}{AL2} \cdot 100\%$$

obj^{s}_{elem} indicates the energy that object *obj* spends on operations executing element *elem* during time slot *s*. C^{AL1}_{AL2} compares two *E* values.

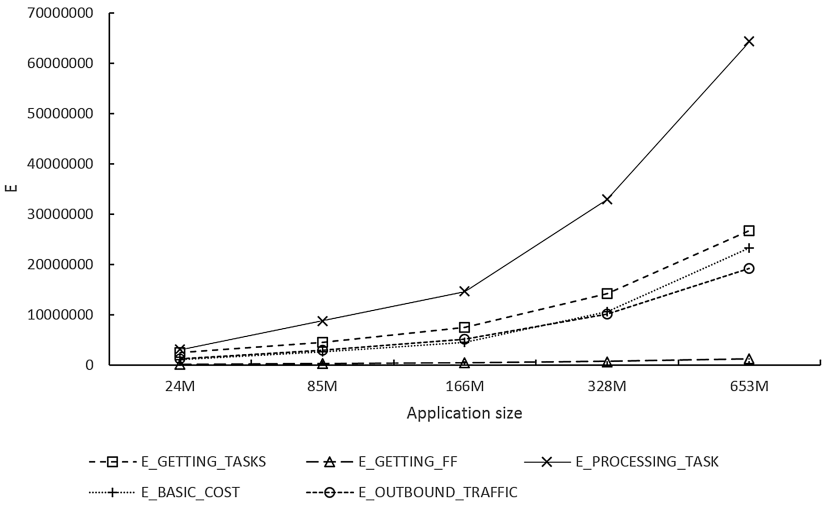


Fig. 7. Energy elements for different data volumes

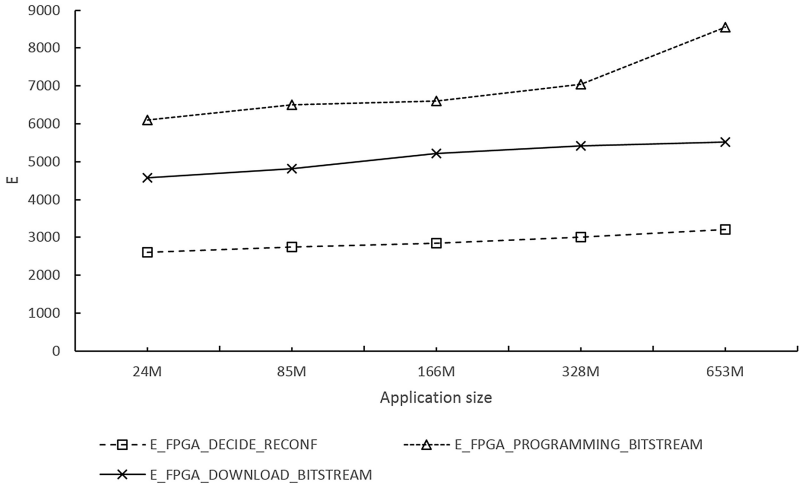


Fig. 8. Energy elements related to reconfiguration

Experiments were done for the *target search* application *A* described in Sect. 2, modified to incorporate five different total tasks sizes (24 M, 85 M, 166 M, 328 M, 653 M respectively). Different task sizes reflect different search area and different data load coming from data sensing equipment. Figures 7 and 8 show energy elements for five cases. Results show, that for the same application the size of the processing data has the most impact to E_PROCESSING_TASK, moderate impact to E_GETTING_TASKS, E_BASIC_COST, E_OUTBOUND_TRAFFIC and minimal to E_GETTING_FF. Impact on functions is small as the same application logic contains

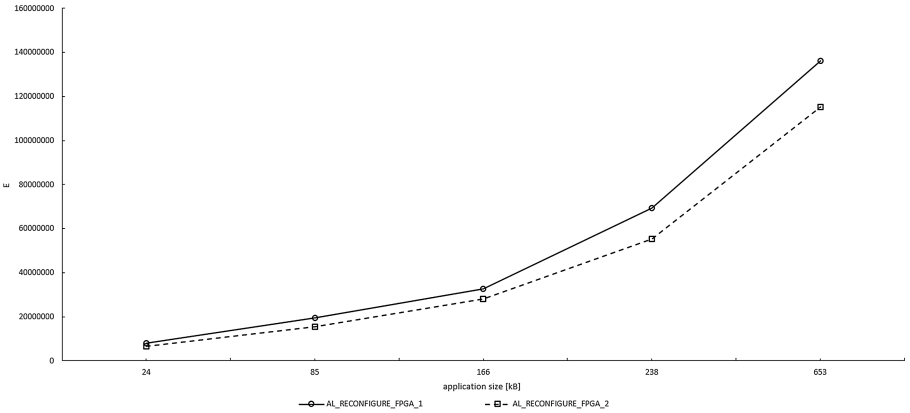


Fig. 9. Energy consumption for two AL_RECONFIGURE_FPGA algorithms

the same set of function, just differs with how many these functions will be sent to units. Regarding the reconfiguration-related energy elements, the E_FPGA_DECIDE_RECONF depends on the timespan of the system operation, so its cost is linear. The largest impact of the size can be observed to the process of programming the bitstream into FPGA (also, being the most costly element of reconfiguration). The even distribution of the functions selected to be programmed onto FPGA, used in AL_RECONFIGURE_FPGA_1 exposed high operational cost, and the differences compared to the AL_RECONFIGURE_FPGA_2 differ depending on the size of processed tasks (Fig. 9, up to 22 % calculated as *C*). AL_RECONFIGURE_FPGA_2 uses the information about tasks' requirements and returns f_{cand} being most universal for the current task resources and suitable for most tasks.

The advantage of AL_RECONFIGURE_FPGA_2 comes from the periodic reconfiguration not limited by remote resources matching. Thus, AL_RECONFIGURE_FPGA_2 is much more flexible and adapts better to the current processing needs of the system. AL_RECONFIGURE_FPGA_1 required the longest time of processing, and the difference compared to the remaining two algorithms increased with the increase of the task(s) size. AL_RECONFIGURE_FPGA_2 is 31 %–38 % faster than AL_RECONFIGURE_FPGA_1.

4 Conclusions

Development of cooperative processing systems that operate on battery power creates demand for efficient algorithms for management. The centralized management is not that much desired for its single point of failure design. At the same time the flexibility of cooperative systems is demanded. In this work, we propose an architecture that addresses all of the above challenges, among others. The use of FPGA chips and the proposed efficient management algorithms create an effective and flexible solution. Research showed, that the proposed algorithms can save both electrical energy and time, while using the same resources for the same application. We also describe the

proposal of the general architecture with multiple layers, multiple roles and various types of resources, along with the universal format of defining the cooperative application suitable for reconfigurable environment. Therefore, it creates a basis for further research, especially in the areas of advanced scheduling algorithms for reconfigurable systems, efficient applications management and improving the life of battery power for decentralized cooperative systems with autonomous nodes.

References

1. Chmaj, G., Selvaraj, H.: Distributed processing applications for UAV/drones: A Survey. In: Selvaraj, H., Zydek, D., Chmaj, G. (eds.). AISC, vol. 366, pp. 449–454Springer, Heidelberg (2015). doi:10.1007/978-3-319-08422-0_66
2. Chmaj, G., Selvaraj, H.: UAV cooperative data processing using distributed computing platform. In: Selvaraj, H., Zydek, D., Chmaj, G. (eds.). AISC, vol. 366, pp. 455–461. Springer, Heidelberg (2015). doi:10.1007/978-3-319-08422-0_67
3. Chmaj, G., Latifi, S.: Decentralization of a multi data source distributed processing system using a distributed hash table. Int. J. Commun. Netw. Syst. Sci. **6**(10), 451–458 (2013)
4. Department of Defense, Unmanned Systems Integrated Roadmap FY2013-2038 (2013)
5. Jóźwiak, L., Nedjah, N.: Modern architectures for embedded reconfigurable systems a survey. J. Circuits Syst. Comput. **18**(2), 209–254 (2009)
6. Jóźwiak, L., Nedjah, N., Figueroa, M.: Modern development methods and tools for embedded reconfigurable systems: a survey. Integr. VLSI J. **43**(1), 1–33 (2010)
7. DeHon, A., et al.: Design patterns for reconfigurable computing. In: 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2004, pp. 13–23 (2004). doi:10.1109/FCCM.2004.29
8. Bondalapati, K., Prasanna, V.: Reconfigurable computing systems. Proc. IEEE **90**(7), 1201–1217 (2002). doi:10.1109/JPROC.2002.801446
9. Freitas, E., Binotto, A., Pereira, C., Stork, A., Larsson, T.: Dynamic reconfiguration of tasks applied to an UAV system using aspect orientation. In: International Symposium on Parallel and Distributed Processing with Applications, ISPA 2008, Sydney, NSW, pp. 292–300 (2008). doi:10.1109/ISPA.2008.69
10. Nadeem, M., Ostadzadeh, S., Nadeem, M., Wong, S., Bertels, K.: A simulation framework for reconfigurable processors in large-scale distributed systems. In: 2011 40th International Conference on Parallel Processing Workshops (ICPPW), Taipei City, pp. 352–360 (2011)
11. Wills, L., Sander, S., Kannan, S., Kahn, A., Prasad, J., Schrage, D.: An open control platform for reconfigurable, distributed, hierarchical control systems. In: 2000 Proceedings of the 19th Digital Avionics Systems Conference, DASC, Philadelphia, PA, pp. 4D2/1–4D2/8 (2000)