Design Methodology for Energy Efficient Unmanned Aerial Vehicles

JINGYU HE, University of Southern California
YAO XIAO, University of Southern California
CORINA BOGDAN, Northeastern University
SHAHIN NAZARIAN, University of Southern California
BOGDAN PAUL, University of Southern California

In this paper, we present a load-balancing approach to analyze and partition the UAV perception and navigation intelligence (PNI) code for parallel execution, as well as assigning each parallel computational task to a processing element in a Network-on-chip (NoC) architecture such that the total communication energy is minimized and congestion is reduced. First, we construct a data dependency graph (DDG) by converting the PNI high level program into Low Level Virtual Machine (LLVM) Intermediate Representation (IR). Second, we propose a scheduling algorithm to partition the PNI application into clusters such that (1) inter-cluster communication is minimized, (2) NoC energy is reduced and (3) the workloads of different cores are balanced for maximum parallel execution. Finally, an energy-aware mapping scheme is adopted to assign clusters onto tile-based NoCs. We validate this approach with a drone self-navigation application and the experimental results show that our optimal 32-core design achieves an average 82% energy savings and 4.7x performance speedup against the state-of-art flight controller.

CCS Concepts: •Networks → Network on chip; •Theory of computation → Graph algorithms analysis; •Computer systems organization → Embedded hardware; Multicore architectures;

Additional Key Words and Phrases: Unmanned Aerial Vehicles

ACM Reference format:
Jingyu He, Yao Xiao, Corina Bogdan, Shahin Nazarian, and Bogdan Paul. 2019. Design Methodology for Energy Efficient Unmanned Aerial Vehicles. 1, 1, Article 1 (January 2019), 13 pages.
DOI: 10.1145/1122445.1122456

1 INTRODUCTION

Unmanned aerial vehicles (UAVs) are emerging as critical tools for mapping large areas, patrolling, searching, and rescuing applications. These tasks are usually dangerous, repetitive and have to be carried out in extreme conditions, making them ideal for autonomous drones. From the traditional quadrotors [21] to a fully-actuated hexarotors [22], and to the futuristic volocopoters [23], many researchers demonstrated various designs of the UAVs for different application scenarios, as shown in Fig. 1. The increasingly complicated self-navigation and collision-avoiding applications of the evolving UAVs ask for a high-performance and low-power flight controller.

We cannot stress the importance of the performance of flight control applications enough. The lack of data-processing speed of the flight control computer chips has led to plane crashes reportedly. At the same time, low-power design is critical for UAVs as well. One reason is that high power dissipation brings tremendous...
cooling challenges to maintain the hardware at a suitable temperature. Another is that batteries are the only energy source for drones, limiting the running time of drones.

In order to push the performance and energy boundary of systems-on-chips, Dally and Towles [1] proposed the tile-based Network-on-chips (NoC) as the ideal architecture for scalable and low-power on-chip communication. Such chips use tiles as building blocks such as CPUs, GPUs, ASIC and memory. A standard interface is embedded into each tile to route flits for communication. There have been many previous studies on energy-aware NoC designs. In contrast to prior NoC work, the goal of this paper is to investigate the parallelization of the UAV perception and navigation intelligence while taking the computation and communication power consumption into consideration. As shown in Fig. 2, we first compile the navigation program into LLVM IR and construct the DDG, where each node denotes only a useful instruction with its power consumption and each edge represents the data dependency with the weight being data size times latency. Second, based on DDG graph, we propose a scheduling algorithm to partition the PNI application into clusters such that (1) inter-cluster communication is minimized, (2) NoC energy is reduced and (3) the workloads of different cores are balanced for maximum parallel execution. Finally, we incorporate topological sort into the our energy-aware mapping scheme to further reduce static power consumption resulted by congestion.

Towards this end, the main contributions of this paper are as follow:

- To the best of our knowledge, our work is the first to incorporate the static energy consumption analysis of application into a compiler-based task partition.
- Besides volume, we propose a mapping strategy to also consider the timing of inter-core communications, reducing the congestion time and static energy consumption of hardware resources.

The rest of the paper is organized as follows: Section II discusses the related work. Section III introduces the basics of UAV control. Section IV illustrates the energy model for NoCs, the load-balancing and energy-aware community detection algorithm, and the low-power mapping. Section V validates the framework and shows experimental results compared to the baseline model.

2 RELATED WORK

As UAV’s core avionic part, flight controller is a system that integrates hardware and software to make drones fully capable of flying without human beings’ guidance. Basic linear controllers such as PID or LQR have been widely used [25, 26] due to their simplicity in design and implementation. However, they are mostly limited to simple non-agile use cases. To model the non-linearity of the motion control more effectively, nonlinear controllers such as backstepping and feedback linearization were proposed [27]. In addition, learning-based controllers were proposed for acrobatic moves [28]. They utilize the approximation ability of the neural networks for learning the dynamics of the multirotors.

To address the increasing computation and energy demand of flight controllers, there has been a significant amount of previous research on energy-aware and load-balancing scheduling and mapping on multicores. From a mathematical and control perspective, Bogdan et al. in [2, 3] provide a complex approach to
Fig. 2. Overview of the UAV intelligent processing architecture workflow. (A): An UAV and its control basics. (B): The perception and navigation intelligence application (as a high level program) is compiled into LLVM IR trace through compiler analysis. This allows to remove the unnecessary computation and communication overhead of high level programs. (C): We transform the trace into the DDG and detect communities. (D): Each processing community is mapped onto an NoC processing element in such a way that its communication energy is minimized and congestion is reduced. The unused cores are clock-gated to save energy, indicated by the blue tiles.

dynamically characterize the workload of multicore systems for performance and power optimization. Xiao et al. propose a complex network-inspired application partitioning tool to improve multicore parallelization [5]. Tan et al. develop a low-power customizable manycore architecture for wearables using a lightweight message-passing scheme [16]. Navion [12] design an energy-efficient accelerator to fully integrate visual-inertial odometry system-on-chip while eliminating expansive off-chip processing and storage for autonomous navigation of drones. In terms of mapping and routing, an efficient branch-and-bound algorithm proposed by Hu et al. [4] automatically maps the IPs onto a generic NoC so that the communication cost is minimized while the timing constraint is met. Lyons et al. create the accelerator store by sharing memories between accelerators to combine the scalability of homogeneous multi-core and SoCfs high performance and power-efficient hardware accelerators [29]. Esmaeilzadeh et al.’s work [30] provides a novel and extendible model that integrates device scaling trends, core design tradeoffs, and multicore configurations. In contrast to prior work, we present an energy-aware load-balancing community detection algorithm together with a mapping strategy and test it using a UAV self-navigation application.

3 BRIEF OVERVIEW OF THE BASICS OF THE UAV NAVIGATION CONTROLLER

Fig. 2(A) shows a UAV with six degrees of freedom. Three degrees of freedom describe the translational motions \((x, y, z)\) and the other three are the rotational motions \((r, p, q)\). Each of the four propellers is equipped with a rotor providing the angular velocity. These four angular velocities correspond to the inputs of the quadrotor, \(\omega_i = [\omega_1, \omega_2, \omega_3, \omega_4]\). Twelve outputs are generated from the quadrotor, \(X = [x, y, z, r, p, q, \dot{x}, \dot{y}, \dot{z}, \dot{r}, \dot{p}, \dot{q}]\), corresponding to the translational and rotational positions, and their corresponding velocities [15].

For real-time applications, the error between the actual UAV position, estimated by a navigation system, and the desired position is fed into a PD-controller to determine the required control inputs. The required rotor speeds are then calculated from the respective torques using:

\[
\begin{bmatrix}
-b & -b & -b & -b \\
0 & -db & 0 & db \\
-db & 0 & db & 0 \\
k & -k & k & -k
\end{bmatrix}
\begin{bmatrix}
\omega_1^2 \\
\omega_2^2 \\
\omega_3^2 \\
\omega_4^2
\end{bmatrix}
\]

where \(T\) is the thrust vector for each propeller, \(\Gamma\) is the torque vector applied to the airframe, \(b\) represents the lift constant, \(d\) is the distance from the rotor to the center of the mass and \(k\) is secondary lift constant. The control
structure employed to fly the quadrotor can be found in [14, 15], and is based on Proportional Derivative action to get the quadrotor’s attitude (roll, pitch, yaw) and altitude.

4 PARALLELIZATION DISCOVERY AND ENERGY OPTIMIZATION APPROACH

4.1 Energy Model

Both IP cores and interconnection consume energy. While most of the mapping algorithms based on the one in [4] only compute dynamic energy, our model considers both static and dynamic power dissipation. N. Grech et al. [8] propose an application static energy analysis technique to determine the instruction energy model directly at the LLVM IR level. Through analysis and measurement of a large set of target ISA instructions, it was found that LLVM IR instructions can be divided roughly into four groups: memory, M, program flow, B, division, D, and all other instructions, G. This yields an energy model $E_N$ of a program executed sequentially in a computing node:

$$E_N = \sum_{i \in \{M, B, D, G\}} E_i N_i$$

(2)

where $E_i$ is the energy cost of a single instruction in group $i$, $N_i$ is the number of the instructions executed in that group, and $n$ denotes the number of instructions.

Using the bit energy concept proposed by Ye et al. in [7], the total dynamic energy consumption can be computed by:

$$E_{DgNoC} = \sum_{i=1}^{a} \sum_{j=1}^{b} w_{ij} (\eta_{ij} E_{Sbit} + (\eta_{ij} - 1) \times E_{Lbit})$$

(3)

where $E_{Sbit}$ and $E_{Lbit}$ represent the energy consumed by switch and link; $\eta_{ij}$ is the number of routers the packet from tile $t_i$ to tile $t_j$ passes through along the way; $w_{ij}$ is the size of the packet; $a$ and $b$ denote the number of tiles on $x$ and $y$ respectively.

The static power is defined to characterize the energy consumed when packets are congested in the buffers. For simplicity, static power is defined as:

$$E_{StNoC} = \sum_{i=1}^{n} P_{S} \times w_i \times t_i$$

(4)

where $n$ is the number of times that congestion occurs; $P_{S}$ is the energy consumption of one bit of data stored in the buffer for one unit of time; $w_i$ is the data size of the $i$th congestion; and $t_i$ is time of the $i$th congestion.

Equation (5) gives the total energy consumption for the interconnect.

$$E_{NoC} = E_{StNoC} + E_{DgNoC}$$

(5)

Finally, given the total number of tiles $n$, the energy consumption of the entire chip is computed as:

$$E = \sum_{i=1}^{n} E_{Ni} + E_{NoC}$$

(6)

4.2 Compiler Analysis and Model of Computation Extraction

In order to generate the data dependency graph (DDG), we adopt the LLVM IR [6]. The rationale behind this is that LLVM is a language-independent system that exposes the commonly-used primitives to implement high-level language features, which makes it very easy to generate back-end for any target platform.

With the help of Clang, C/C++ applications are compiled into a dynamic IR execution trace. We developed a parser to construct a data dependency graph from the IR trace. The parser analyzes memory operations to obtain latency and data sizes. Because the execution times and energy vary on data sizes and where the data...
resides, taking those values into account could potentially reduce inter-core communications by grouping the source and destination instructions of a register into one cluster. Three hash tables are created and updated when parsing: the source table, the destination table and the dependency table. The source/destination tables are used to keep track of source/destination registers with keys being source or destination registers and values being the corresponding line number. The dependency table is to store dependencies between nodes with keys being the line number for current instruction, and values being clock cycles, data sizes and line numbers of previous instructions dependent on the same virtual register.

For example, in Table I, a LLVM IR snippet is extracted from an application compiled by Clang front-end. As the parser reads the first line, a source table and a destination table are created. The source table is updated with the key being %5 and the value being 1 and its destination register is hashed into the destination table with the key being %1 and value the being 1. When line 2 is read, the source register %1 happens to be the destination register in line 1. A dependency table is created and updated with the key being 2 (line number of current instruction) and value being 1 (line number of the dependent instruction). Following the same procedure, the three hash tables will look like what is shown in Table 1.

### 4.3 Discovering the Processing Community Structure

To formulate this problem, we introduce the following concepts:

**Definition 4.1.** A data dependency graph (DDG) is a weighted directed graph \( G = (a_i, b_{ij}, e_i, w_{ij} | i, j \in N) \) where each vertex \( a_i \) represents one LLVM IR instruction; each edge \( b_{ij} \) with weights \( w_{ij} \) characterizes either the dependency from \( a_i \) to \( a_j \) or the control flow such as jumps or branches from one block to another; and \( e_i \) stands for the estimated energy of the vertex given in Section IV.A.

**Definition 4.2.** A weight \( w_{ij} \) between \( a_i \) and \( a_j \) is calculated by latency times data size. Latency characterizes the delay from \( a_i \) to \( a_j \) based on the timing information. Data size represents the number of bytes transferred.

**Definition 4.3.** A quality function determines how efficient the LLVM IR instructions are grouped together in terms of energy consumption, parallelism, load balancing, hardware utilization and inter-cluster data movements.
The discovery of the processing community structure problem can now be formulated as follows: Given a DDG, find non-overlapping processing communities which maximize the quality function:

$$Q = \sum_{c=1}^{n_c} \left( \frac{(W_c - S_c)}{W} - \frac{(W_c - \bar{W})^2}{W} \right) - \frac{\sum_{c=1}^{n_c} E_{N_c} + E_L}{E} \quad (7)$$

and satisfy:

$$N \geq n_c \quad (8)$$

where $n_c$ denotes the number of clusters; $W_c$ stands for the sum of edge weights within cluster $c$; $W$ is the sum of all edge weights; $S_c$ represents the sum of edges weights connected to cluster $c$; $N$ is the core count.

The first term in equation (7) confines the data flow within the cluster as much as possible. It indicates the difference between the sum of the weights in a cluster and the sum of the weights of the edge connected to the cluster. The greater this term is, the fewer inter-cluster data movements, and the more energy is saved.

The second term in equation (7) measures the standard deviation squared between sum of weights in cluster $c$ and average sum of weights in all clusters. Minimizing this term ensures load balancing and fully takes advantage of parallel execution.

The third term in equation (7) characterizes the energy model of the application, where $E_{N_c}$ calculates the energy consumed at each node using Equation (2) and $E_L$ computes the energy consumption for communication transactions. To maximize quality $Q$, this term needs to be minimized in order to save energy.

While optimal communities discovery is an NP-hard problem, we use the Ollivier-Ricci Curvature (ORC) based community detection algorithm [19] to decide which node should be grouped in which cluster. The complexity of this algorithm is $O(EV^2)$, where $E$ is the number of edges in DDG, and $V$ is the average degree.

**Algorithm 1:** ORC-based method for Communities Discovery Algorithm

1. **Input:** Data Dependency Graph (DDG) with a list of nodes and edges and core count $n$
2. **Output:** Communities
3. **count** = 2
4. while count $\leq n$
5. \hspace{1em} DDG' $\leftarrow$ DDG Calculate Ollivier-Ricci curvature for all edges;
6. \hspace{1em} while there is negative edge curve do
7. \hspace{2em} Remove the most negatively curved edge;
8. \hspace{2em} Re-calculate the Ollivier-Ricci curvature for the affected existing edges;
9. \hspace{1em} PreferentialAttachment(DDG', number_of_communities, minimum_community_size);
10. \hspace{1em} Pick the solution with the minimum inter-cluster weights;

### 4.4 Compact Intelligence Mapping into Constrained Hardware

The tile to which each cluster is mapped significantly affects the power consumption of the application since it determines the dynamic and static communication cost. Consequently, an approach, which is similar to the one in [4], is proposed, but it takes cluster ordering into consideration as well so that it reduces static energy consumption caused by congestion and contention of hardware resources.

**Definition 4.4.** A task graph (TG) is a weighted directed acyclic graph $TG = G(c_i, a_{ij}, v(a_{ij}), b(a_{ij}) | i, j \in N)$ where each vertex $c_i$ represents a cluster of LLVM IR instructions that are grouped together by our community detection algorithm, and each edge $a_{ij}$ represents communication from node $c_i$ to node $c_j$. 
Design Methodology for Energy Efficient Unmanned Aerial Vehicles

- $v(a_{ij})$: data size from $c_i$ to $c_j$.
- $b(a_{ij})$: bandwidth requirement from $c_i$ to $c_j$.

**Definition 4.5.** An architecture graph (AG) is a directed graph $AG = (t_i, p_{ij}, e(p_{ij}), |i, j \in N|)$ where each vertex $t_i$ represents a tile, and each edge $p_{ij}$ represents a routing path from $t_i$ to $t_j$.
- $e(p_{ij})$: energy consumption from $t_i$ to $t_j$.
- $L(p_{ij})$: set of links that makes up $p_{ij}$.

In order to exploit parallelism and pipelining, we apply topological sort to the task graph before mapping. The depth of cluster $c_j$ is defined as the maximum number of edges from the root to $c_j$. In Fig. 3, cluster $D$ cannot execute before cluster $B$ and $C$ because it needs data from both of them. However, cluster $B$ and $C$ can execute in parallel because they are at the same depth.

![Topological Sort](image)

**Fig. 3.** Application of a topological sort to task graph.

**Algorithm 2: Compact Intelligence Mapping Algorithm**

**Input:** TG and AG

**Output:** Mapping from TG to AG

1. $count = 0$
2. while TG is not empty do
3.   if $count == 0$ then
4.     Get the cluster with depth of zero and map to (0,0)
5.   else
6.     Create a set $S_{count}$ of all clusters with depth of $count$;
7.     Map $S_{count}$ to the available tile in AG so that:
8.     $\min \{ E = \sum_{v(a_{ij})} v(a_{ij}) e(p_{map(c_i), map(c_j)}) \}$
9.     $count += 1$
10. if Any idle tile $t$ left in AG then
11.   Power gate $t$

**4.4.1 Energy and Congestion Analysis.** The communication-weighted mapping (CWM) proposed in [4] (we refer to it as H) fails to consider the order of the clusters, leading to significant potential congestion and static energy consumption in NoCs. This section shows how our communication dependency mapping (CDM) mitigates this problem.

For illustration purposes, we assume $E_{S_{bit}} = E_{L_{bit}} = 1 \times 10^{-12}/bit$. Applying the CWM to the TG in Fig. 3 may yield the following two different mappings in Table 2. For instance, using Equation (3) in CWM, $E_{D_{AC}} = 6 \times (3 \times E_{S_{bit}} + (3 - 1) \times E_{L_{bit}}) = 30 \times 10^{-12}/bit$. Both mappings’ dynamic energy costs are $109 \times 10^{-12}/J$. 

, Vol. 1, No. 1, Article 1. Publication date: January 2019.
In terms of static energy, we assume $P_{St} = 1 \times 10^{-12} J$, and the execution time is 10ns for all clusters. Also assume one packet flit is 1bit and the time for a flit to pass through a switch ($t_s$) is 2ns and a link ($t_l$) is 1ns. Fig. 4 shows the timing diagram of all computations and all packet deliveries of both mappings. For instance, in CWM, the first flit of the packet from cluster A to B takes $2 \times t_s + t_l = 5$ns to arrive (routing delay), while the rest of the packet needs another 9ns (packet delay).

In CWM, when cluster B finishes execution and is about to route the packet to D, D’s input buffer is busy because of A → D and A → C packet transmissions. Thus, B must wait until A → C is done. While the two mappings yield the same execution time of 67ns, the packets from B to D in CWM experiences a 10ns longer congestion delay, hence consuming more static energy. Applying Equations (4) and (5), CWM consumes 17% more energy in interconnect.

5 EXPERIMENTAL RESULTS

We use gem5 [9] together with McPAT [11] for architectural and power simulation. Our hardware model is ARM processor connected in a 2D mesh topology NoC from 4 to 128 cores [10] with MESI cache protocol as shown in Table 3. Detailed parameters are listed in TABLE III. We compare our community discovery algorithm with a range of existing scheduling algorithms, including Greedy Scheduling [18], Local Search [17] and Hierarchical Hungarian[?].
Table 3. Parameters of simulation processors

| Cores                | Up to 128 cores in-order ARM cores at 500MHz |
|----------------------|----------------------------------------------|
| L1 Private Cache     | 32KB, 4-way, 32-byte block                   |
| L2 Shared Cache      | 128KB, 8-way                                 |
| Topology             | 2D Mesh with XY routing                      |

First, we examine our ORC-based community discovery algorithm’s (Algorithm 1) computational complexity as the number of core grows. The processing community discovery is done offline (only once), so the run time will not affect the controller speed during UAV navigation. Fig. 5 shows the run time overhead of the referenced scheduling against our algorithm over a range of 4 to 128 core organization. For a small number of cores, the run time overhead is acceptable, but increase rapidly for the Local Search and Hierarchical Hungarian, both of which have $O(n^2)$ complexity. Meanwhile, the Greedy Scheduling with $O(n (\log n)^2)$ is more scalable. Their overheads are reasonably small even for 128 cores.

![Fig. 5. Scheduling runtime overheads over our community discovery algorithm](image1)

Fig. 6 evaluates the performance loss of the various scheduling algorithms over our ORC-based communities discovery algorithm. The Greedy Algorithm exhibits the largest performance loss of roughly 9% for large core

![Fig. 6. Scheduling performance loss over our community discovery algorithm](image2)
count. This is because it fails to consider the complexity of scheduling while only focusing on ranking applications by Instruction Per Clock (IPC). A particular application may suffer significantly if one core has much lower IPC than others. The Local Search delivers decent performance as the core count increases while the Hierarchical Hungarian offers the best combination of run time and performance. However, they both struggle to detect small clusters compared to the graph as a whole because they have a resolution limit. Since ORC-based algorithm takes advantage of the geometric concepts of graph curvature, it identifies better instruction parallelism within the densely-connected community structure and minimizes the communication between clusters. The Data Dependency Graph of UAV application is shown in Fig. 8.

![Data Dependency Graph of UAV navigation application.](image)

Next, we compare the energy consumption of our CDM and CWM mapping (Fig. 7). Both mappings use the same communities generated by our Fast Spectral Clustering Algorithm. The power values are collected by feeding the outputs from gem5 to McPAT. Having fully taken advantage of parallel execution, load-balancing and optimal inter-community communication, our design has achieved an average of 17% less Power Delay Product over CWM as the maximum PDP saving of 22% is achieved for 64-core configuration. As the core count grows, the PDP decreases significantly from 4-core to the lowest at 32-core for 76%. Starting from 64 cores, the number of flits needed to be routed between cores soars as PDP increases significantly. The lowest PDP is achieved by 32-core configuration at $13.7 \mu S \times mW$. 

Vol. 1, No. 1, Article 1. Publication date: January 2019.
Finally, we illustrate the potential of our design by comparing it with the state-of-art flight controllers used in Intel Aero Ready to Fly Drone [20]. As shown in Fig. 9, the Intel Aero Flight Controller is built upon a STM32 MCU, which is a single core ARM Cortex-M4 CPU. We conducted FS simulation in gem5 by running the same navigation algorithm on our 16-core and 32-core configurations and on a single core ARM CPU. As shown in Fig. 10, the navigation algorithm running with our scheduling and mapping scheme has a 3.6x speedup on 16 cores and 4.7x on the 32-core configuration. Meanwhile, the PDP is also improved by 82% on our 32-core platform.

Fig. 9. Hardware Block Diagram - Aero Flight Controller

![Hardware Block Diagram - Aero Flight Controller](image)

Fig. 10. Comparison against Intel Aero Flight Controller

![Comparison against Intel Aero Flight Controller](image)

6 CONCLUSION

In this paper, we first develop an LLVM IR parser to construct the DDG for UAV autonomous navigation application. Next, we analyze the DDG structure and discover its best parallelization degree by applying our load-balancing and energy-aware processing community discovery algorithm so that data movement is confined within clusters and static energy consumption is minimized. Finally, a congestion-aware mapping scheme based on topological sort is proposed to map clusters onto the NoCs for parallel execution. Simulations show that our optimal 32-core design achieves an average 82% energy savings and 4.7x performance speedup against the state-of-art Intel Aero Flight Controller.
REFERENCES

[1] Dally, William J and Towles, Brian. Route packets, not wires: on-chip interconnection networks. In Proceedings of the 38th DAC, pages 684–689. ACM, 2001.

[2] Bogdan, Paul. Power-aware scheduling for AND/OR graphs in real-time systems. In Proceedings of the 9th NOCS, pages 21. ACM, 2015.

[3] Bogdan, Paul and Xue, Yuankun. Mathematical models and control algorithms for dynamic optimization of multicore platforms: A complex dynamics approach. In Proceedings of the ICCAD, pages 170–175. IEEE, 2015.

[4] Hu, Jingcao and Marculescu, Radu. Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures. In Proceedings of the DATE, pages 688–693. IEEE, 2003.

[5] Xiao, Tao and Xue, Yuankun and Nazarian, Shahin and Bogdan, Paul. A load balancing inspired optimization framework for exascale multicore systems: A complex networks approach. In Proceedings of the 36th ICCAD, pages 217–224. IEEE, 2017.

[6] Lattner, Chris and Adve, Vikram. LLVM: A compilation framework for lifelong program analysis & transformation. In Proceedings of the CGO’04, pages 75. IEEE, 2004.

[7] Ye, Terry Tao and Benini, Luca and De Micheli, Giovanni. Analysis of power consumption on switch fabrics in network routers. In Proceedings 2002 DAC, pages 524–529. IEEE, 2002.

[8] Grech, Neville and Georgiou, Kyriakos and Pallister, James and Kerrison, Steve and Morse, Jeremy and Eder, Kerstin. Static analysis of energy consumption for LLVM IR programs. In Proceedings of the 18th SCOPES, pages 12–21. ACM, 2015.

[9] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al. The gem5 simulator. ACM SIGARCH Computer Architecture News, 39(2):1–7, 2011.

[10] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha. Garnet: A detailed on-chip network model inside a full-system simulator. In 2009 ISPASS, pages 33–42. IEEE, 2009.

[11] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In Proceedings of the 42nd MICRO, pages 469–480. ACM, 2009.

[12] A. Suleiman, Z. Zhang, L. Carlone, S. Karaman, and V. Sze. Navion: A 2-nw fully integrated real-time visual-inertial odometry accelerometer for autonomous navigation of nano drones. IEEE Journal of Solid-State Circuits, 2019.

[13] C. Tan, A. Kulkarni, V. Venkataramani, M. Karunaratne, T. Mitra, and L.-S. Peh. Locus: Low-power customizable many-core architecture for wearables. TEECS, 17(1):1–6, 2018.

[14] S. Armah, S. Yi, W. Choi, and D. Shin. Feedback control of quad-rotors with a matlab-based simulator. American Journal of Applied Sciences, 2016.

[15] P. Corke. Flying Robots Book: Robotics, Vision and Control. Springer, 2017.

[16] Goldberger, Jacob and Tassa, Tamir. A hierarchical clustering algorithm based on the Hungarian method. Pattern Recognition Letters, 29(11):1632-1638, 2018.

[17] Winter, Jonathan A and Albonesi, David H. Scheduling algorithms for unpredictably heterogeneous cmp architectures. In Proceedings of the 2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN), pages 42–51. IEEE, 2008.

[18] Teodorescu, Radu and Torrellas, Josep. Variation-aware application scheduling and power management for chip multiprocessors. ACM SIGARCH computer architecture news, 36(3):363-374, IEEE Computer Society, 2008.

[19] Sia, Jayson and Jonckheere, Edmond and Bogdan, Paul. Ollivier-ricci curvature-based method to community detection in complex networks. Scientific reports, 9(1):8900, Nature Publishing Group, 2019.

[20] Intel. 2018 Intel Aero Compute Board Hardware Features and Usage. Retrieved Oct 27, 2019 from https://www.intel.com/content/dam/support/us/en/documents/drones/development-drones/intel-aero-compute-board-guide.pdf

[21] Hoffmann, Gabriel and Huang, Haoxiao and Waslander, Steven and Tomlin, Claire. Quadrotor helicopter flight dynamics and control: Theory and experiment. In AIAA guidance, navigation and control conference and exhibit, 2017.

[22] Rajappa, Sujiit and Ryll, Markus and Büllthoff, Heinrich H and Franchi, Antonio. Modeling, control and design optimization for a fully-actuated hexrotor aerial vehicle with tilted propellers. In Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA), pages 4006–4013. IEEE, 2015.

[23] Ullman, David G and Homer, Vincent and Horgan, Patrick and Ouellette, Richard. 2017. Comparing electric sky taxi visions.

[24] Das, Abhijit and Subbarao, Kamesh and Lewis, Frank. Dynamic inversion with zero-dynamics stabilisation for quadrotor control. IET control theory & applications, 3(3):303–314, IET, 2009.

[25] Bouabdallah, Samir and Notth, Andre and Siegwart, Roland. PID vs LQ control techniques applied to an indoor micro quadrotor. 22004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)/IEEE Cat. No. 04CH37566, 3:2451–2456, IEEE, 2004.

[26] Cowling, Ian D and Yakimenko, Oleg A and Whidborne, James F and Cooke, Alastair K. Direct method based control system for an autonomous quadrotor. Journal of Intelligent & Robotic Systems, 60(2):285–316, Springer, 2010.

[27] Lee, Daewon and Kim, H Jin and Sastry, Shankar. Feedback linearization vs. adaptive sliding mode control for a quadrotor helicopter. International Journal of control, Automation and systems, 7(3):419–428, Springer, 2009.
[28] Lupashin, Sergei and Schöllig, Angela and Sherback, Michael and D’Andrea, Raffaello. A simple learning strategy for high-speed quadrocopter multi-flips. In Proceedings of the 2010 IEEE international conference on robotics and automation, pages 1642–1648. IEEE, 2010.

[29] Lyons, Michael J and Hempstead, Mark and Wei, Gu-Yeon and Brooks, David. The accelerator store: A shared memory framework for accelerator-based systems. ACM Transactions on Architecture and Code Optimization (TACO), 8(4):48, ACM, 2012.

[30] Esmaeilzadeh, Hadi and Blem, Emily and Amant, Renee St and Sankaralingam, Karthikeyan and Burger, Doug. Dark silicon and the end of multicore scaling. In Proceedings of the 2011 38th Annual international symposium on computer architecture (ISCA), pages 365–376. IEEE, 2011.