Temperature Management for Heterogeneous Multi-core FPGAs Using Adaptive Evolutionary Multi-Objective Approaches

Renzhi Chen
CERCIA
School of Computer Science
University of Birmingham, UK
Email: rxc332@cs.bham.ac.uk

Peter R. Lewis
School of Engineering
and Applied Science
Aston University, UK
Email: p.lewis@aston.ac.uk

Xin Yao
CERCIA
School of Computer Science
University of Birmingham, UK
Email: X.Yao@cs.bham.ac.uk

***Abstract*—Heterogeneous multi-core FPGAs contain different types of cores, which can improve efficiency when use with an effective online task scheduler. However, it is not easy to find the right cores for tasks when there are multiple objectives or dozens of cores. Inappropriate scheduling may cause hot spots which decrease the reliability of the chip. Given that, our research builds a simulating platform to evaluate all kinds of scheduling algorithms on a variety of architectures. On this platform, we provide an online scheduler which uses multi-objective evolutionary algorithm (EA). Comparing EA and current algorithms such as Predictive Dynamic Thermal Management (PDTM) and Adaptive Temperature Threshold Dynamic Thermal Management (ATDTM), we find some drawbacks. First, current algorithms are overly dependent on manually set constant parameters. Second, those algorithms neglect optimization for heterogeneous architecture. Third, they use single-objective methods, or use linear weighting method to convert a multi-objective optimization into a single-objective optimization. Unlike other algorithms, EA is adaptive and does not require resetting parameters when workloads switch from one to another. EAs also improve performance when used on heterogeneous architecture. A efficient Pareto Front can be obtained with EAs for the purpose of multiple objectives.**

I. Introduction

A chip is considered to be heterogeneous if cores on this chip have different microarchitectures, in terms of function units, size of register, or size of L1 cache. The advantage of heterogeneous multi-core architecture is obvious. A research of heterogeneous architecture from National University of Singapore achieves improvement in both performance and energy consumption compared with homogeneous counterparts [1]. However this result is based on an assumption: we can find the right core for the task. This assumption is invalid when there are dozens of or hundreds of cores - there are too many alternatives which make it impossible to enumerate within limited time.

Meanwhile, the high power density makes FPGAs easily burnt. Temperature control of chips becomes a very important objective besides the pursuit of higher performance. Task migration is an effective method to achieve this objective [2]. Recent works such as [3], [4], [5], [6] have achieved this objective by scheduling. However, most of the schedulers rely on many manually set parameters which actually should fluctuate according to run-time situations. Also these algorithms are either designed for one objective or converted the multi-objectives problem to a single objective one, as they are based on greedy or greedy-like algorithms. However, to tackle temperature control problem, more objectives arise besides minimizing the maximum temperature. For example, smaller gap between different parts on the chip is expected as this helps increase reliability.

It will be more complicated to find out the right core if more objectives are concerned, such as performance, power consumptions, temperature, etc. Scheduling problems on a heterogeneous multi-core FPGA can be formulated as multi-objective optimization problems (MOPs)[7]. Evolutionary Algorithms (EAs) have been widely used to solve MOPs. There are plenty of algorithms using evolutionary approaches to solve MOPs such as NSGA II [8], SPEA II[9] and PESA [10].

Our research intends to build a heterogeneous multi-core chip simulator to evaluate different scheduling algorithms on different kinds of architecture. The simulator should be able to generate all the run-time data, such as temperature and power consumption of every core, and react to outside control, such as migrating tasks among cores and adjusting voltage/frequency of a specified core. We combine simulators - Hotspot[11], McPat[12] and Gem5[13] - to achieve our goal. Then we put forward a evolutionary multi-objective scheduling approach. By comparing our EA with existing algorithms in some particular cases, we show that a scheduling algorithm will gain improvement if it has the following features: firstly it is optimized for heterogeneous systems, secondly its parameters are self-adaptive and lastly it is multi-objective.

II. Related Work

There exists a variety of task scheduling strategies to achieve a trade off between performance and temperature. Most of strategies contain three steps: trigger, predictor, selector. The trigger may be on-chip cycle counter or thermal sensors. When the manually set threshold is exceeded, the future situation of every potential alternatives will be predicted. After that, one solution will be picked out from the potential alternatives according to their objectives and the forecast by the selector.

In 2009 Eisenhardt et al. [3] put forward a row-rotation algorithm which migrates tasks from current row of cores to the cores in the next row periodically. The experiment is based on a series of simulators.

Another classic algorithm Predictive Dynamic Thermal Management (PDTM) evaluated by Yeo et al. [4] tries to control temperature on real homogeneous 4-core and 8-core systems. In this algorithm, when the current temperature of a core triggers a manually set threshold, a future coolest core will be selected as the migration target according to an algorithmic predictor.

There are some improved versions of PDTM, such as temperature-aware scheduler based on applications' thermal behaviour groups[5] and the algorithm from Ayoub et al. [14] in 2011. The latest upgrade is developed by Salami et al. [6] in 2014. It optimizes PDTM with a self-adjusting migration threshold, called ATDTM. In ATDTM, a counter will record the times of migrations. When the counter excessed a manually set threshold, the threshold of temperature will increase or decrease by certain amount.

The drawbacks of the existing algorithms lie in several aspects:

a) Non-adaptive: All these algorithms rely on some manually set parameters. However, these parameters vary when facing different architectures. It is hard to reach the expected trade off between temperature and performance. Also for different types of workloads or different architectures, most of those parameters have to be reset. It will cause problem if we do not reset them when the workloads hugely change, or the architectures change run-time(like reconfiguration).

b) Single-objective: They can only can have few objectives. Since most of the algorithms rely on a greedy or greedy-like algorithms in the selection part, it is hard to integrate them into a final parameter for greedy algorithm. Current works use the linear weighting method to convert the multi-objective optimization into a single-objective one. But if there is more objectives, the method will have problem in the process of conversion. However, in a real situation, we may have a variety of objectives required for a chip such as performance, power consumption, and temperature. Optimising these objectives surely will be beneficial.

c) Homogeneous: They are not suitable for heterogeneous systems. First, heterogeneous system enlarges the number of alternatives, which brings problems. It is impossible to enumerate when facing hundreds of different types of cores because there will be too many alternatives but limited time. Second, a wrong migration in heterogeneous system usually cost larger losses than that on a homogeneous system, while current algorithms cannot avoid repeating the same mistake.

III. Research Framework

Our research has two purposes. First we intend to build a heterogeneous multi-core chip simulator to evaluate different scheduling algorithms on different kinds of architecture. Second, we want to show that a algorithm, like our multi-objective evolutionary scheduling algorithm should be used in the scheduling problem for a heterogeneous multi-core platforms.

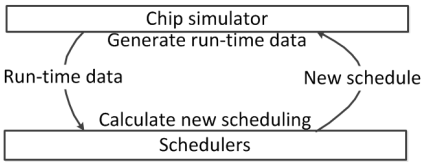


Fig. 1: Simulating Process

The research flow can be like this: in every simulative cycle, the chip simulator generates run-time data for the scheduler; then the scheduler reckons the best scheduling for the future cycles and returns this result to the simulator. After that, the simulator carries out migration according to the scheduling and simulating process for the next period, and generates new run-time data again. This process begins to loop till all tasks are finished. The process is described in Figure 1.

According to the process, the chip simulator in our research must achieve:

1) Simulatively executing some tasks on a heterogeneous multi-core chip.
2) Generating run-time data (power, performance and temperature) of every core on the chip.
3) Responding to an outside control, including simulating migrations and changing frequency and voltage of the core.

A. Chip Simulator

The chip simulator is constituted by three existing simulators: Gem 5 [13], MCPAT [12] and Hotspot [11].

1) Gem 5: Gem 5 is an architecture simulator, which can simulate tasks run on a chip. By editing the configure files, this simulator simulates every kind of architecture. Another input of Gem 5 is the binary code. The output of this simulator is the run-time statistical data, such as the number of register access, cache miss rate.

2) MCPAT: MCPAT is an integrated power, area, and timing modelling framework [12]. It can calculate the run-time power trace of every unit on the chip based on run-time statistical trace generated by Gem 5 and the hardware configuration.

3) Hotspot: Hotspot by HP was utilized to convert the power measurements to temperature. The input files for Hotspot include a floorplan describing hardware configure, and a file of power trace.

We use Gem 5 to run different benchmarks which generates run-time performance details, such as execution time and times of register read/write operation. MCPAT is used to generate power trace according to the data from Gem 5. Also the area of every unit on the chip will be calculated by MCPAT according to the hardware configuration. Hotspot can figure out the temperature for every part of the chip based on the power trace and the area of every unit on the chip generated by MCPAT.

However, when Gem 5 begins to simulate running tasks, it cannot response to an outside control, like migrations, or output the intermediate state when it is still running. That

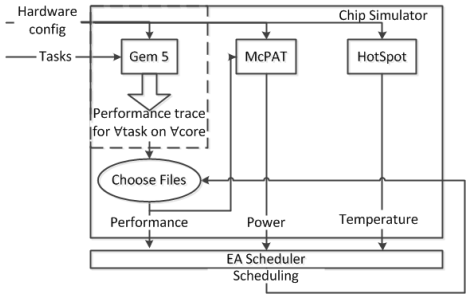


Fig. 2: Chip Simulator

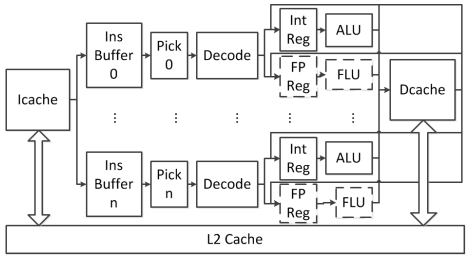
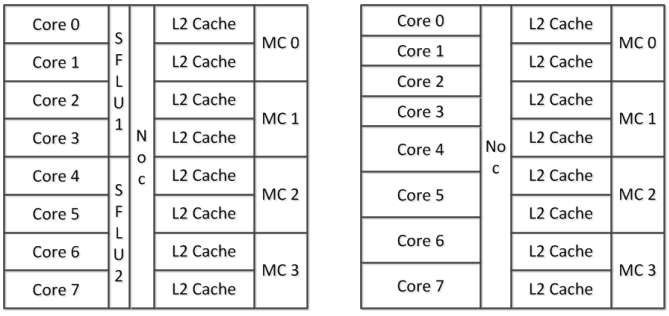


Fig. 3: The Pipeline of cores



(a) Floorplan of Arch. I (b) Floorplan of Arch. II

Fig. 4: Floorplan of the Arch. I&II

Program	Total Instructions (Billions)	INT	FLOPS	Reads&Writes
Blackscholes	2.7(small)	25%	42%	33%
Bodytrack	14.0(large)	27%	30%	33%
Canneal	7.2(small)	53%	6%	41%
Freqmine	33.5(large)	52%	0%	48%
Streamcluster	22.1(large)	4%	53%	43%

TABLE II: Character of Workload Set

means we can only get the statistical data until all tasks are finished. Hence we cannot combine these three simulators directly. First, in order to collect the trace data instead of the final state, we can save checkpoints in Gem 5 at every simulative cycle, then stop Gem 5 to generate a "final" state and rerun Gem 5 from that checkpoint again. These "final" states constitute the trace file for MCPAT. Second, if we ignore the impact of L3 cache and cache coherence, we can assume that the performance of one core will not be influenced by other cores. An optional method simulating migrations is first to use Gem 5 to generate all traces for every task running on every type of cores first. Then the chip simulator begins to simulate a running chip. If a migration occurs, the MCPAT will select the right trace file to read from files which is already generated by Gem 5. The process is indicated in Fig. 2.

B. Chip Architecture

Then we need to specify which chip architectures are going to simulate. In our experiment, we have two architectures-Arch. I and II.. The first one is the architecture of OpenSPARC T1[15], while Arch. II is an 8-core T1-like architecture with 2 types of core. All cores on the chips are in-order Harvard architectures. The difference between cores only lie in the number of read/write ports to cache, the size of data/instruction register and the number of function units such as ALU and FPU. Details for the two chips are demonstrated in the Table I. There may be multiple pipelines on one chip. The amount of pipelines equals to the number of ALU ,while FPU always share pipelines with ALU. Fig. 3 shows the micro-architecture of the pipelines on a core. The floorplan of Arch. I and II are descripted in Fig. 3.

C. Workloads

We use the workloads in a benchmark set called PARSEC[16]. It includes 12 workloads such as Blackscholes, Bodytrack and Ferret. In our experiment we only use Blackscholes, Bodytrack, Freqmine, Streamcluster and Canneal. The characteristics of these workloads are described in Table II.

The Blackscholes calculates the prices for a portfolio of European options analytically with the Black-Scholes partial differential equation (PDE) [17]. It contains ALU, FPU and Load and Store operations which have similar proportions. The bodytrack tracks a 3D pose of a marker-less human body with multiple cameras through an image sequence [18]. It also comprises similar proportions of different types of operations, but needs longer execution time. Canneal uses cache-aware simulated annealing to minimize the routing cost of a chip design [19]. It is constituted by nearly half ALU operations and half memory operations. Freqmine employs an array-based version of the FP-growth method for Frequent Itemset Mining [20]. This workload is similar to that of Canneal but longer execution time is required. Stream-cluster finds a predetermined number of medians so that each point is assigned to the nearest centre [21]. This workload has little ALU operations but half FPU and half memory operations.

D. Assumptions

As our research mainly focuses on schedule algorithm, we do not concern how to gather run-time data and do prediction. In this case we can assume:

1) All the algorithms can gather run-time data and predict future situation perfectly.
2) The migration from one core to another lasts for a constant cycle's time.
3) No delay for scheduler. We do not consider the time for the scheduling algorithm itself.

IV. Evolutionary Scheduler

We provide a adaptive evolutionary multi-objective scheduler optimizing for heterogeneous architecture in our

Architecture	Cores	Icache		Dcache		num of	num of	ALU		FPU		L2 Cache		
		size	port	size	port	Int Reg	FP Reg	num	delay	num	delay	size	way	penalty
Arch I	I-A	16KB	1R/1W	8KB	1R/1W	32	16	1	1	1/4	16	256KB	12	22
Arch II	II-A	16KB	1R/1W	8KB	1R/1W	32	16	1		1				
	II-B	32KB	2R/2W	16KB	2R/2W	64	32	2		1				

TABLE I: Detail of Micro Architecture of Arch. I&II

research. Our problem definition makes use of the following notation:
•Number of tasks: n
•Number of cores: m
•Number of objectives: o
•Size of Population: s
•A set of tasks: $E = \{\epsilon_1, \epsilon_2, ..., \epsilon_n\}$
•A set of cores: $C = \{c_1, c_2, ..., c_m\}$
•A set of notional waiting core: $C_n = \{c_{m+1}, ..., c_{m+n}\}$, to which a task may be assigned to indicate that it is in a waiting state (i.e. the task is not currently being executed).
•A set of time intervals: $T = 0, 1, ..., t_{max}, t_{max} = |T| + 1$
•A set of coming time of tasks: $T_c = \{t_{c1}, t_{c2}, ..., t_{cn}\}$
•A set of leaving time of tasks: $T_l = \{t_{l1}, t_{l2}, ..., t_{ln}\}$
•A vector : $\overrightarrow{\sigma} = (u_1, u_2, ..., u_{m+n}), u_i \in (1, n) \cup \{-1\}$, which stands for a schedule of tasks to cores. $u_i \in (1, n)$ means task u_i is running on core i(including notional cores), otherwise it means nothing is on core i. This vector allows 2 operations: add a new task or delete a finished task.
•A set of possible choice of schedule(the population for EA): $\Sigma = \{\overrightarrow{\sigma_1}, \overrightarrow{\sigma_2}, ..., \overrightarrow{\sigma_s}\}$
•A set of evaluating functions: $O = \{F_1(\overrightarrow{\sigma_k}), ..., F_o(\overrightarrow{\sigma_k})\}$
•A set of normalized fitness value: $O_{normalized} = \{f_1, ..., f_o\}$
•The temperature at time t_i of core j: $\theta_{t_i,j}$
•The performance at time t_i of core j: $p_{t_i,j}$
•The predicted temperature at time t_i of core j for a schedule $\overrightarrow{\sigma_k}$: $\overline{\theta}_{t_i,j}(\overrightarrow{\sigma_k})$
•The predicted performance at time t_i of core j for a schedule $\overrightarrow{\sigma}_k$: $\overline{p}_{t_i,j}(\overrightarrow{\sigma}_k)$

A. EA process

At time t, the EA can be described as pseudo-code:

EA-SCHEDULOR()
1 **for** $\forall t_{cj} \in T_c$
2 **do if** $t = t_{cj}$
3 **then for** $\forall \sigma_i \in \Sigma_t$
4 **do** $\sigma_i.add(t_{cj})$
5 **for** $\forall t_{lj} \in T_l$
6 **do if** $t = t_{lj}$
7 **then for** $\forall \sigma_i \in \Sigma_t$
8 **do** $\sigma_i.delete(t_{lj})$
9 $Evaluate(\Sigma_t)$
10 **for** $i < Iterations$
11 **do** $\Sigma_t^{'} \leftarrow GetChildren(\Sigma_t)$
12 $Evaluate(\Sigma_t^{'})$
13 $\Sigma_t \leftarrow Select(\Sigma_t^{'} \cup \Sigma_t)$
14 $\Sigma_{t+1} \leftarrow \Sigma_t$
15 $\sigma_{t+1} \leftarrow FinalSelect(\Sigma_{t+1})$
16 **return**σ_{t+1}

1) Evaluate(): We may set multiple objectives according to actual demand. The objectives are set by adding evaluate functions. For example, in order to control the temperature on the chip while maximizing the performance, we set 4 objectives-maximizing the overall performance of all cores guarantees the performance for the whole chip, maximizing the minimum performance of tasks can avoid any tasks not executed for too much cycles, minimizing the average temperature of chip and minimizing the temperature gap between any cores for the next period can fulfil our temperature goals. The evaluating functions for the objectives mentioned above are(at t_i):

- Maximum the overall performance of chip for next period:

$$F_1(\overrightarrow{\sigma_k}) = \max_{\overrightarrow{\sigma_k} \in \Sigma} (\sum_{j=1}^{m} \overline{p}_{t_{i+1},j}(\overrightarrow{\sigma_k}))$$

- Maximize the minimum performance of any tasks:

$$F_2(\overrightarrow{\sigma_k}) = \max_{\overrightarrow{\sigma_k} \in \Sigma} (\min_{1 \leqslant j \leqslant m} \overline{p}_{t_{i+1},j}(\overrightarrow{\sigma_k}))$$

- Minimum the temperature gap between every core for next period:

$$F_3(\overrightarrow{\sigma_k}) = \min_{\overrightarrow{\sigma_k} \in \Sigma} (\max_{1 \leqslant j_1, j_2 \leqslant m} (\overline{\theta}_{t_{i+1},j_1}(\overrightarrow{\sigma_k}) - \overline{\theta}_{t_{i+1},j_2}(\overrightarrow{\sigma_k})))$$

- Minimum the average temperature of chip for next period:

$$F_4(\overrightarrow{\sigma_k}) = \min_{\overrightarrow{\sigma_k} \in \Sigma} ((\sum_{j=1}^{m} \overline{\theta}_{t_{i+1},j}(\overrightarrow{\sigma_k}))/m)$$

In order to make comparison between different objectives, we need to normalize all fitness values. One example is using the linear method as:

$$f_n(\overrightarrow{\sigma_k}) = \frac{\max_{\sigma_i \in \Sigma} F_n(\sigma_i) - F_n(\overrightarrow{\sigma_k})}{\max_{\sigma_i \in \Sigma} F_n(\sigma_i) - \min_{\sigma_j \in \Sigma} F_n(\sigma_j)}, n \leqslant o$$

2) Get Children(): A function generates children from parents. In the process of $GetChildren()$ we randomly select two vectors from parents and do crossover and variation. The higher sum of fitness values is, the larger possibility that the individual is chosen as parent is. Both two vectors will be split into 2 parts. We combine the first part from one parent and the second part from the other parent as crossover, and randomly exchange some components in the new vector as variation, then check whether this new child meets our requirement. One child is acceptable if it satisfies following criteria:

1) No number in child vector appears twice(except -1).
2) Every number appeared in parents should be appeared in child.
3) It is not the same as parents.

4) It is not the same as its ancestors. We record the scheduling for the last 5 simulating cycles as "ancestor". This can avoid migrating tasks into and out of a core repeatedly.
5) Child is not on the blacklist. We add every "bad" scheduling to the blacklist. A scheduling will be in the blacklist if a new scheduling for the same cores is still carried out in the following cycle after that. All record in the blacklist will be removed after 5 simulating cycles. This will avoid wrong migrations based on to the history knowledge.

If a child is not acceptable, we discard it and do selection, crossover and variation again. This process will loop until we find enough children.

3) Select(): A function selecting next generations from both children and parents. We mainly use NSGA-II to select the non-dominated set according to Pareto sorting.

Since a trade off among different objectives is expected, we need to limit those extreme solution in the non-dominated set. By calculating the variance of fitness values, we can delete those of which variance exceeded the threshold before the Pareto sorting. When $Var(f_1, ..., f_i), i \leqslant o$ exceeds the threshold, this alternative will not be considered.

Then we calculate non-dominated sorting with a toleration ϵ. We redefined non-dominated relation $\prec$ to:

$$a \prec b \iff \exists i, f_i(a) < (f_i(b) - \epsilon) \cup \forall j, f_j(a) \leqslant (f_j(b) - \epsilon)$$

This makes us get a larger non-dominated set with toleration which brings more choices for us.

The process of Selecting can be described as:

SELECT($Union$)
for $\sigma \in Union$
  do if $Var(sigma) > VarThreshold$
    do $Union \leftarrow (Union - \sigma)$
$Fronts \leftarrow NondominatedSort(Union)$
$\Sigma \leftarrow \emptyset$
$Front_L \leftarrow \emptyset$
for $Front_i \in Fronts$
  do $CrowdingDistanceAssignment(Front_i)$
    if $Size(\Sigma) + Size(Front_i) > s$
      do $Front_L \leftarrow i$
      else $\Sigma \leftarrow Merge(\Sigma, Front_i)$
if $(Size(\Sigma) < s)$
  do $Front_L = SortByRankAndDistance(Front_L)$
    for P_1 **to** $P_{s-Size_{Front_L}}$
      do $\Sigma \leftarrow P_i$

$CrowdingDistanceAssignment$ calculates the average distance between individuals of each front on the front itself. $SortByRankAndDistance$ functions sort individuals first by rank and then distance within the front.

4) Final Select(): A function selecting the 'best' scheduling from the next generation. We use greedy for the rank-sum. For every alternative we calculate the rank r_i of every fitness values f_i compared to other alternative, then we use greedy algorithms to minimize R:

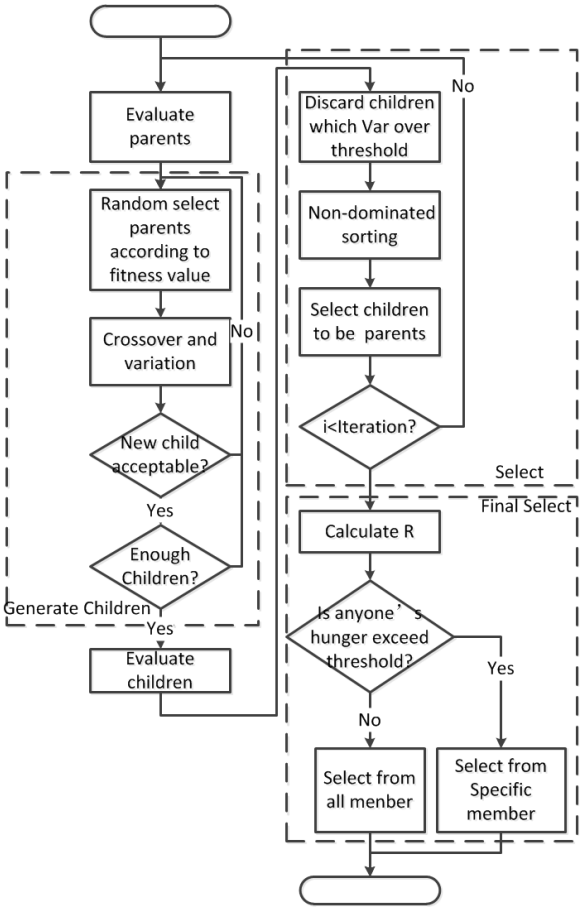


Fig. 5: Flow Chart of EA Process

$$R = \alpha_1(r_1) + ... + \alpha_o(r_o), \sum_{i=1}^{o} \alpha_i = 1$$

In which the α_i are the parameters to control the balance between different objectives.

Besides, in order to avoid that some tasks are not executed in too many cycles, we add a parameter $hunger$ for every tasks. If a task is on a notional core (which means this task is not executed), we increase $hunger$ of this task. If a task is on a real core which means this task is running, we decrease $hunger$, and $hunger \geqslant 0$. Once the $hunger$ of a task exceeds the threshold, only those alternatives which run this task on real cores will be considered.

The flow chart for the EA process is described in Fig. 5.

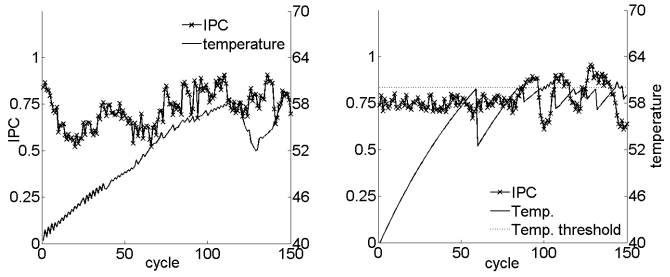
V. EXPERIMENT RESULTS AND ANALYSIS

By comparing our EA and current algorithms such as PDTM and ATDTM, from the experiment we can learn that the EA will benefit because of adaptive parameters, optimization for heterogeneous and multiple objectives. The frequency of the chip is 1000MHz, and the simulating period is (100M machine cycle)100ms, and all the "cycle" mentioned in the experiment is the simulating cycle. We use centigrade as

temperature metric, and instruction per cycle(IPC) as the run-time performance metric.

A. Adaptive vs. Constant

Most current works rely on some manually set parameters to achieve balance between different objectives. The parameters will be constant once it is set. However, those parameters are not supposed to be constant. In one case, if the workloads changes, usually those parameters should fluctuate. A case study of temperature threshold in [6] shows that temperature threshold should be run-time adjustive. We compare situations of PDTM, ATDTM and EA with light workloads and situations with heavy workloads on the heterogeneous multi-core chip. PDTM is a non-adaptive algorithm, while ATDTM and EA are run-time adaptive. We use all the five tasks as the heavy workloads, and five empty-loop tasks as light workloads. Fig. 6 shows the temperature and IPC of PDTM and EA when running light workloads. The temperature threshold is set at 60°, which is the best for the light workloads. Both algorithms works well during the whole process. The small difference is that IPC of EA is a little bit lower than PDTM, while temperature do not raise so fast as that of PDTM.



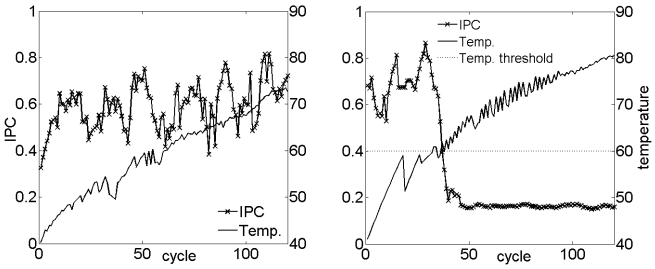
(a) EA (b) PDTM

Fig. 6: Maximum Temperature and IPC for light workloads

However, the result above is relying on we have a right temperature threshold. But for different workloads, we may have different thresholds. Fig. 7 shows the temperature and IPC of three different algorithms when running heavy workloads when keeping all the parameters the same.

As we use the same threshold 60° for PDTM, which is too low for the heavy workloads, the algorithm fails since most of the cores exceed the threshold. In Fig. 7b we can see that, since the maximum temperature exceeds the threshold around cycle 40, the IPC drops from 0.7 to around 0.2 as too frequent and useless migrations have been carried out. In EA's case, the migrations will be carried out once EA thinks it has found out a better alternative, thus there is no strong relation established between temperature and IPC. The statistical data for this experiment is shown in TABLE III. From the table, we can easily find that the IPC for PDTM with heavy workloads is irregular.

There is more than workloads will cause this problem. Another case, if the chip is run-time reconfigurable, for PDTM the parameters have to be reset. Here comes to a conclusion that adaptive parameters, instead of manually set constants, can keep the algorithms working when running fluctuate workloads.



(a) EA (b) PDTM

Fig. 7: Maximum Temperature and IPC for heavy workloads

Algorithms	light workloads		heavy workloads	
	Aver. Temp	IPC	Aver. Temp	IPC
PDTM	50.7°	0.75	67.3°	0.29
EA	48.9°	0.72	61.9°	0.61

TABLE III: Statistical Data for Different workloads

B. Heterogeneous vs. Homogeneous

For a heterogeneous system, migrating tasks to a wrong core may cause huge lose which should have been avoided. However currently there is not any temperature management algorithms for heterogeneous systems. Existing researches are mainly based on homogeneous systems. These algorithms may have problems when used on a heterogeneous system. In EA, we avoid wrong migrations by setting those alternatives as an unacceptable child.

We compare EA and ATDTM on the homogeneous architecture Arch. I and the heterogeneous architecture Arch. II. The workload set is the same, which contains all 5 tasks. By adjusting parameter in the process $FinalSelect()$ of EA, we make EA have the same average maximum temperature around 70° as ATDTM does.

The TABLE IV shows the total executed instructions per running core. Because EA need some cycles to converge, the performance for EA is not good at the beginning of execution. On the homogeneous chip, EA and ATDTM is very close, with only 0.2% difference on cycle 100(10 seconds). But for a heterogeneous chip, EA have about 7% performance lift when some simple techniques are used. We can draw a conclusion that it is worthy to optimize the scheduling algorithms aiming to heterogeneous architecture.

Arch.	Algorithm	executed inst.(billion)				
		cycle 20	cycle 40	cycle 60	cycle 80	cycle 100
Homo-	EA	11.49	23.28	36.98	49.73	62.77
	ATDTM	11.52	23.35	37.23	50.16	62.67
Speed-up		97.9%	99.7%	99.07%	99.2%	100.2%
Heter-	EA	11.65	24.08	39.98	53.51	67.42
	ATDTM	11.55	23.47	37.35	50.29	62.84
Speed-up		100.8%	102.6%	107.0%	106.4%	107.3%

TABLE IV: Statistical Data for Different Architectures

C. Multi-objective vs. Single-objective

Temperature management is a multi-objective problem, at least it contained objectives of temperature and performance. However, for a multi-core system it usually has more objectives, such as average temperature, maximum temperature and temperature gap between every core. All objective are important to ensure the reliability of the chip. But current algorithms usually is not multi-objective. They usually use linear weighting method to convert the problem of multi-objective optimization into a single-objective one. There are some drawbacks for this method: first, it requires manually set weights for every objective, which is difficult because it requires consistent normalization despite run-time features such as changing workloads and preferences. Second, it works worse when there are more objectives. We convert ATDTM into a multi-objective version by using linear weighting method that is the same as $FinalSelect()$ in EA. Then we compare ATDTM with EA under multiple objectives. The objectives in the experiments are:

- **Obj. 1**: Maximize IPC.
- **Obj. 2**: Minimize maximum temperature of the chip.
- **Obj. 3**: Minimize temperature gap between different cores in the chip.

By adjusting the weights in ATDTM, we keep the average IPC of EA and ATDTM the same at 0.61, and set the same weight for both Obj. 2 and Obj.3. The result for Obj. 2 and Obj. 3 is described below.

1) Maximum Temperature: Fig. 8 shows the maximum temperature of the chip under ATDTM and EA. Although EA is not obviously more advanced than ATDTM in the beginning several cycles, the maximum temperature of EA is always lower than that of ATDTM after approximately 40 cycles. That is because EA needs some cycles to converge, and during that process, the performance of EA usually is low. On the average, the Maximum temperature of EA is sightly lower than that in ATDTM, which is $2°$(EA is $58.54°$ while ATDTM is $60.03°$).

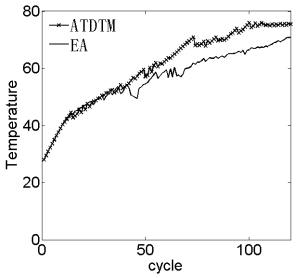
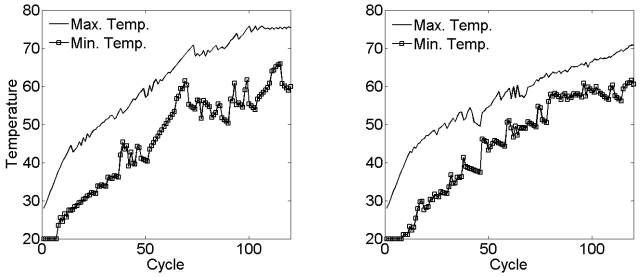


Fig. 8: Maximum Temperature for Different Algorithms

2) Temperature Gap: Fig. 8 shows the maximum and minimum temperature of the chip under ATDTM and EA. Both two algorithm begin migrating at around 5 cycle, but EA will decrease the temperature gap sharply after around 40 cycles. Although the temperature gap of ATDTM is also under control, but on average, the temperature gap of EA is around $9.52°$ while that for ATDTM is $12.74°$. That is because although ATDTM take temperature gap as one of the objectives, the importance of that temperature gap can be lost during the process of converting the multi-objective optimization to single objective one. As a comparison, although EA also have the same process of conversion in the process of $FinalSelect()$, but those unsatisfactory alternative already have been discarded during the evolutionary process. Another reason is that although the threshold of ATDTM is adaptive, but it requite some cycles for ATDTM to find the suitable threshold. When the features of the workloads fluctuate, the performance of ATDTM will soon decrease.



(a) ATDTM (b) EA

Fig. 9: Max. and Min. Temperature for Different Algorithms

3) Pareto Front: By adjusting the weight we can get the Pareto front of Obj. 2 and Obj. 3. The TABLE V shows the results of 5 set of experiments.

	Weight(Obj.2: Obj.3)	1:0	2:1	1:1	1:2	0:1
EA	Aver. Max. Temp.	57.77	58.21	58.54	59.53	63.34
	Aver. Temp. Gap	17.07	12.03	9.52	7.3	4.5
ATDTM	Aver. Max. Temp.	58.25	59.38	60.03	60.11	63.48
	Aver. Temp. Gap	18.11	14.32	12.74	10.54	4.9

TABLE V: Statistical Data for Different Weights

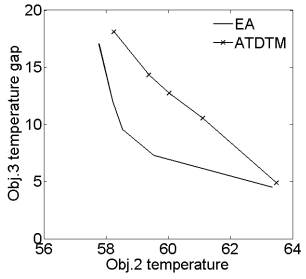


Fig. 10: The Pareto Front for Obj. 2 & Obj. 3

The Fig. 10 is the Pareto front, we can find that at weights 1:0 and 0:1(which reduce the problem into 2 objectives), The results of ATDTM and EA are very close to each other, while ATDTM behave poorest near weights 1:1. That is because actually ATDTM does not work well under multi-objective situation. Although we use linear weighting method to convert the problem into a single-objective one, performance of ATDTM still decreases when objectives are added. Here we come to the conclusion that current algorithms cannot work well when multiple objectives are required.

VI. Conclusion and Future Works

In our research we build a platform to simulate run-time scheduling problems. Our chip simulator can generate run-time data includes temperature, performance and power consumptions. By providing an online multi-objective evolutionary scheduler, it shows that current algorithms such as PDTM and ATDTM have weaknesses because they require manually set constant parameters, do not account for heterogeneous architecture and are single-objective methods. Our scheduler does not have any parameter related to workloads, so without changing any parameters, EAs can still work well while the performance of PDTM drops seriously (IPC drops from 0.7 to 0.2) because of the unsuitable parameters. Using techniques such as child blacklisting in the EA, in the experiment we find that the EA can work better(7% improvement on IPC) than ATDTM on a heterogeneous architecture. Also our EA uses a multi-objective method, rather than converting the multi-objective problem into a single-objective one. The experiment shows that the EA can gain a better Pareto front than ATDTM does.

In our future works we mainly want to provide a scheduler suitable for any workloads, architectures, many-core and reconfigurable system. That can be divided into three sections:

•Convert multi-objective EA into a dynamic one. Scheduling problems on heterogeneous multi-core FPGAs can also be formulated as dynamic multi-objective optimization problems (DMOPs). A few studies have addressed EA for DMOPs, such as dynamic orthogonal multi-objective EA[22] and competitive-cooperative coevolutionary algorithms for DMOPs[23]. We can use some existing EAs to solve DMOPs.

•Optimize our scheduler for the reconfigurable system. Reconfigurable FPGA allows chip to change architecture run-time, which will bring even lager challenge to current scheduling algorithms. However, EA can achieve this easily by formulating reconfigure problem as a dynamic problem.

•Do experiment on many-core system. By setting suitable size of population and times of iteration we can reduce the calculated amount compared with current greedy or greedy-like algorithms.

VII. Acknowledgement

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement no 257906.

References

[1] M. Pricopi and T. Mitra, "Bahurupi: A polymorphic heterogeneous multi-core architecture," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 8, no. 4, p. 22, 2012.

[2] J. Kong, S. W. Chung, and K. Skadron, "Recent thermal management techniques for microprocessors," *ACM Computing Surveys (CSUR)*, vol. 44, no. 3, p. 13, 2012.

[3] S. Eisenhardt, T. Schweizer, A. Bernauer, T. Kuhn, and W. Rosenstiel, "Prevention of hot spot development on coarse-grained dynamically reconfigurable architectures," in *Reconfigurable Computing and FPGAs, 2009. ReConFig'09. International Conference on*. IEEE, 2009, pp. 12–17.

[4] I. Yeo, C. C. Liu, and E. J. Kim, "Predictive dynamic thermal management for multicore systems," in *Proceedings of the 45th annual Design Automation Conference*. ACM, 2008, pp. 734–739.

[5] I. Yeo and E. J. Kim, "Temperature-aware scheduler based on thermal behavior grouping in multicore systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009, pp. 946–951.

[6] B. Salami, M. Baharani, and H. Noori, "Proactive task migration with a self-adjusting migration threshold for dynamic thermal management of multi-core processors," *The Journal of Supercomputing*, pp. 1–20, 2014.

[7] P. R. Lewis, W. C. Chibamu, and X. Yao, "Towards a dynamic evolutionary approach to fpga temperature management," in *2 nd Workshop on Self-Awareness in Reconfigurable Computing Systems (SRCS13)*, 2013, p. 22.

[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182–197, 2002.

[9] E. Zitzler, M. Laumanns, L. Thiele, E. Zitzler, E. Zitzler, L. Thiele, and L. Thiele, "Spea2: Improving the strength pareto evolutionary algorithm," 2001.

[10] D. W. Corne, N. R. Jerram, J. D. Knowles, and M. J. Oates, "Pesa-ii: Region-based selection in evolutionary multiobjective optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2001*. Citeseer, 2001.

[11] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "Hotspot: A compact thermal modeling methodology for early-stage vlsi design," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, no. 5, pp. 501–513, 2006.

[12] 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 *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*. IEEE, 2009, pp. 469–480.

[13] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[14] R. Ayoub, K. Indukuri, and T. S. Rosing, "Temperature aware dynamic workload scheduling in multisocket cpu servers," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, no. 9, pp. 1359–1372, 2011.

[15] I. Parulkar, A. Wood, J. C. Hoe, B. Falsafi, S. V. Adve, J. Torrellas, and S. Mitra, "Opensparc: An open platform for hardware reliability experimentation," in *Fourth Workshop on Silicon Errors in Logic-System Effects (SELSE)*, 2008.

[16] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 2008, pp. 72–81.

[17] F. Black and M. Scholes, "The pricing of options and corporate liabilities," *The journal of political economy*, pp. 637–654, 1973.

[18] A. O. Balan, L. Sigal, and M. J. Black, "A quantitative evaluation of video-based 3d person tracking," in *Visual Surveillance and Performance Evaluation of Tracking and Surveillance, 2005. 2nd Joint IEEE International Workshop on*. IEEE, 2005, pp. 349–356.

[19] P. Banerjee, *Parallel algorithms for VLSI computer-aided design*. Prentice-Hall, Inc., 1994.

[20] G. Grahne and J. Zhu, "Efficiently using prefix-trees in mining frequent itemsets." in *FIMI*, vol. 3, 2003, pp. 123–132.

[21] L. OCallaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani, "High-performance clustering of streams and large data sets," in *Proceedings of the 18th International Conference on Data Engineering*, 2002.

[22] S.-y. Zeng, G. Chen, L. Zheng, H. Shi, H. de Garis, L. Ding, and L. Kang, "A dynamic multi-objective evolutionary algorithm based on an orthogonal design," in *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*. IEEE, 2006, pp. 573–580.

[23] C. K. Goh and K. C. Tan, "A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization," *Evolutionary Computation*, vol. 13, no. 1, pp. 103–127, 2009.