Energy-Aware Automatic Tuning on Many-Core Platform via Differential Evolution

Yijun Jiang, Xuan Qi, Chen Liu
Department of Electrical and Computer Engineering
Clarkson University
Potsdam, NY 13699
{*jiangy, qix, cliu*}*@clarkson.edu*

***Abstract*—The need for energy-aware computing has become increasingly demanding. However, high-performance computing is always users' pursuit. The key is how to maintain high performance meanwhile consuming an acceptable amount of energy. One feasible method is the energy-aware automatic tuning. In this paper, we implemented and compared several Differential Evolution (DE)-based auto-tuning algorithms on an innovative many-core platform, in an effort to improve the energy consumption and energy delay product (EDP) of the entire platform. Our results show that Adaptive Differential Evolution algorithm is able to achieve better energy consumption as well as EDP than other DE algorithms tested. It also converges faster. We believe DE algorithm is an effective method that can be applied towards energy-efficient computing on many-core platforms.**

***Keywords*-Differential Evolution; Dynamic Voltage and Frequency Scaling; Energy-Aware Computing; Many-core Processors;**

I. Introduction

With the development of microprocessor technology, Moore's Law has hit the "power wall" due to the continued transistor scaling, physical limits of silicon, overheat problem, etc. As a result, many-core technology becomes a solution which allows us to continue Moore's Law. However, the performance of microprocessors does not ideally increase by packing more cores but the energy consumption does.

The trend of integrating hundreds of millions of transistors and other related components on a single silicon chip about a baby's fingernail leads to hot spots on the chip. These hot spots will impede the hardware performance and reduce the life-cycle of the chip. Additionally, massive energy consumption costs a significant amount of money and natural resources, especially in large data centers. Although the efficient cooling technology may be an alternative solution, the equipment itself is expensive. Therefore, efficient power-aware computing is notably demanding nowadays.

One approach to power-aware computing is through automatic tuning (auto-tuning) based on dynamic voltage and frequency scaling (DVFS). Auto-tuning is the technology where the frequency and voltage level of the system, which we call "gear", is changed adaptively based on the characteristics of programs.

The Single-chip Cloud Computer (SCC) [1], [2] is a 48-core "concept vehicle" created by Intel Labs as a platform for many-core software research. The architecture of SCC is shown in Figure 1. The SCC uses Intel Pentium P54C core and employs 2D mesh communication architecture. On SCC, every eight cores are grouped into one voltage domain where it can be set independently to a specified voltage level. Every two cores are formed into one frequency domain where its frequency level can be individually adjusted. With this unique feature, although it is not the newest many-core processor, SCC is an ideal platform to conduct auto-tuning related research. Furthermore, the computing capability delivered by the 48 cores on the SCC makes it possible to distribute several programs over the platform and execute them simultaneously.

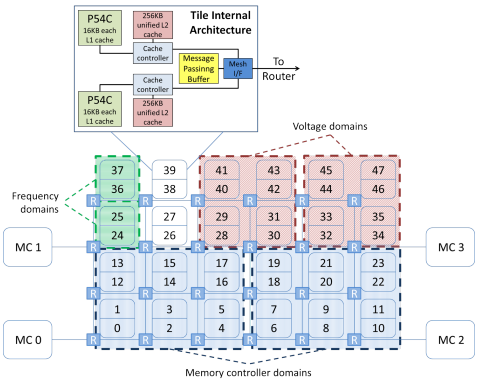

Figure 1. SCC architecture

On the SCC, the voltage level ranges from 0.7v to 1.3v and the frequency level ranges from 100MHz to 800MHz, respectively. There are 69 possible combinations of gears per power domain[1], which gives us a search space on the order of 10^{11} when we look at the 6 power domains. Considering manually brute force all combinations are impossible, the power-aware automatic tuning technique is a practical method to tackle this problem. Specifically, we implement auto-tuning algorithm based on Evolutionary Algorithms (EA). The most representative categories among EA are Genetic Algorithm (GA), Particle Swarm Optimization (PSO) and Differential Evolution (DE). Among these three, DE and DE-variants shows very good performance [3]–[5]. DE

[1]where the 48 cores are divided into six 8-core domains.

algorithm has several advantages:

1) Simplicity: Ease of implementation with a small number of control parameters. DE is much simpler than most evolutionary algorithms. "The main body of DE is less than ten lines of code in any programming language" [6]. The number of control parameters of DE is only two (mutation factor and crossover rate). Without in-depth knowledge, it is possible to determine these values by sensitivity studies.
2) Reliability: Despite its simplicity, DE and DE-variants consistently secure its front ranks in all CEC competitions [6]. It implies that DE-variants outperform most evolutionary algorithms regarding finding the global minima and the converge speed.
3) Applicability: DE is a universal optimizer which is capable of handling non-differentiable, nonlinear, multi-model objective and high-dimensional problems.

That is why we choose DE to handle the auto-tuning problem on the SCC. The contribution of this paper is that we performed a quantitative analysis by cross-comparison among four different DE/DE-variant algorithms (FSF, RSF, TVSF, and JADE). The objective is to optimize the energy and energy-delay product (EDP) consumption of the many-core chip when executing multiple programs at the same time.

The rest of the paper is organized as follows: we review related work in Section II. Our methodology of this work is presented in Section III. Sections IV and V are the experiment setup and the experimental results. Finally, we conclude our work in Section VI.

II. Related Work

Dynamic voltage and frequency scaling (DVFS) has been studied and proven to be an effective way to reduce the power consumption of processors. The DVFS method can be applied on both uniprocessor platforms and cluster systems. One previous study is called β algorithm [7], which dynamically changes the frequency by profiling the feedback on system's performance in a fixed time interval. The scaling of CPU running frequency in next interval is determined by the calculation of β value, which takes computation and memory intensiveness, current frequency into account [7].

One category of DVFS methods is optimization-based algorithms. The method proposed by Hotta et al. [8] described a profiling-based optimization algorithm that profiles execution time and power consumption of the cluster. They divided the program's execution into several stages and selected the best gear. They achieved almost 40% reduction of EDP.

There are some work related to auto-tuning developed on the SCC. One work by Pankratius et al. [9] investigated using auto-tuning algorithms to optimize the performance of MPIBZIP on the SCC by modifying two parameters that directly affect the performance of the program. Three different algorithms: Nelder-Mead Simplex (NMS), Differential Evolution (DE) and the hybrid of NMS and DE were examined. Their work proved that optimization-based auto-tuning method for performance improvement is feasible. However, their scheme requires to have prior knowledge of the parameters that the program will use to facilitate the auto-tuning.

Another work proposed by Berry et al. [10] was also to investigate both the Nelder-Mead Simplex and Differential Evolution's performance on the SCC. In their work, the workload on the SCC was divided into four phases where each sub-program is a phase. The purpose was to simulate the behavior of a multi-phase program. Their result showed that auto-tuning algorithms have the ability to converge to a final solution candidate, e.g., one gear for each phase, without traversing the entire search space. Another benefit of their work is that there is no need to choose parameters of the workload, as opposed to [9]. But in their work only one sub-program can be running at a given time on the SCC, which can use up all 48 cores.

In the work by Roscoe et al. [11], they presented a multi-program workload where three programs are executed at the same time on SCC. They used the classical DE algorithm, which is called Fixed Scale Factor (FSF), to find the final solution and achieved more than 50% EDP consumption reduction of the entire SCC platform. However, their work would be more convincing if different algorithms or baseline comparisons had been provided. Another limitation of their work is the maximum number of cores each program can use is limited to 16.

III. Methodology

A. DE Algorithms

In this section, we introduce the classical DE algorithm [12] and the three DE variant algorithms.

1) Fixed Scale Factor (FSF): The FSF is the classical DE algorithm. The flowchart of FSF is shown in Figure 2. To begin with, a fixed number of vectors are randomly initialized within an n-dimensional search space. These vectors form the population of the potential solutions. Then, this population evolves over time to explore the search space and to locate the minima of the objective function. At each iteration, new vectors are generated by the linear combinations of three vectors randomly chosen from the current population. This operation is called mutation. The outcoming vectors are then mixed with a predetermined target vector. This operation is called recombination and produces the trial vector. Finally, the trial vector is accepted for the next generation if and only if it yields a reduction in the value of the objective function. This last operator is referred to as selection [13].

2) Random Scale Factor (RSF): In RSF, the differential factor F is generated as in Equation 1. F is randomized

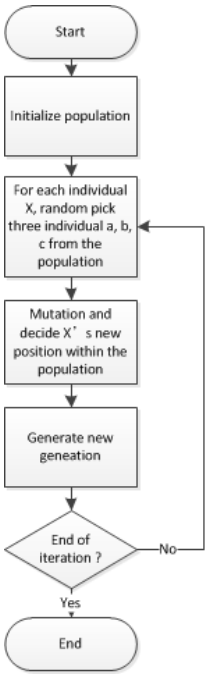

Figure 2. Flowchart of the classical DE algorithm

within the range of $[0.5, 1.0]$. Hence, the mean value of F is 0.75. Besides, the crossover rate CR is 0.5 [4].

$$F = 0.5 \times [1.0 + Rand(0.0, 1.0)] \tag{1}$$

3) Time Vary Scale Factor (TVSF): In TVSF, the crossover rate CR is also 0.5 [4]. The IT_{max} is the maximum iteration number of DE. IT represents the current iteration number, where F_{max} and F_{min} are the preset maximum and minimum differential weights. Here we set $F_{max} = 1.2$ and $F_{min} = 0.4$. In most evolutionary algorithms, it is encouraged to apply a big value at early stages to seek a larger searching space and apply a small value at later stages for a faster convergence rate [4].

$$F = (F_{max} - F_{min}) \times (IT_{max} - it)/IT_{max} \tag{2}$$

The work of Roscoe et al. [11] used the classical DE with the fixed differential factor $F = 1.0$ and the crossover rate $CR = 0.5$. But in the work of Swagatam et al. [4], by using RSF and TVSF, they achieved superior performance over FSF. We want to verify how much improvement could be attained with these two algorithms in the domain of power-aware computing.

4) Adaptive Differential Evolution (JADE): JADE algorithm [14] is a greedy DE algorithm.

Many greedy DE algorithms have better convergence speed than the classic DE, but they are usually not reliable and may have problems such as premature convergence. Generally, most greedy DEs only mutate the best individual of current generation. However, JADE overcomes this problem by introducing a new mutation strategy which mutates the top $p\%$ of current generation:

$$v_{i,g} = x_{i,g} + F_i \cdot (x^p_{best,g} - x_{i,g}) + F_i \cdot (x_{r1,g} - x_{r2,g}) \tag{3}$$

where $x_{i,g}$ is a parent vector, $x^p_{best,g}$ is the top $p\%$ of current generation. F_i is the differential weight of current generation. $r2$ and $r1$ are randomly selected from parent generation.

Besides, JADE changes the CR and F value generation by generation as follows:

$$CR_i = randn_i(\mu_{CR}, 0.1) \tag{4}$$

$$\mu_{CR} = (1 - c) \cdot \mu_{CR} + c \cdot mean_A(S_{CR}) \tag{5}$$

where μ_{CR} is the mean of the normal distribution with 0.1 variance at first generation. c is one constant set to 0.1 by default. S_{CR} is the set of "successful" CR of last generation population. The mean here is the arithmetic mean.

$$F_i = randc_i(\mu_i, 0.1) \tag{6}$$

$$\mu_F = (1 - c) \cdot \mu_F + c \cdot mean_L(S_F) \tag{7}$$

where μ_F is the mean of the Cauchy distribution with 0.1 variance. c is the same constant as in Equation 5. CR_i is the set of "successful" CR of all the population. The $mean_L$ is Lehmer mean.

The idea behind these equations is searching for a better solution depends on how "successful" the previous generation are. For example, if all the nth generation has the better performance than the $(n-1)th$ generation. Then, all the F and CR values of the nth generation are stored into S_F and S_CR, respectively. Then, the control parameters of $(n+1)th$ generation are calculated by using F and CR values from the nth generation.

B. Problem statement

With three programs running concurrently on the SCC, we want to find the optimal setting in a vector format $\vec{x}$ so that the SCC consumes the lowest energy-delay product(EDP) or energy.

The EDP and energy minimization problems are:
minimize

$$f(\vec{x}) = EDP = Energy \times Delay \tag{8}$$

$$f(\vec{x}) = Energy = \int_0^T Pdt \tag{9}$$

where P is the measured power of the entire SCC chip, and T is the total execution time.

Table I
SELECTED GEARS ON THE SCC

Gear	Frequency	Voltage
1	800 MHz	1.1 V
2	533 MHz	1.0 V
3	400 MHz	0.9 V
4	320 MHz	0.8 V
Idle	100 MHz	0.7 V

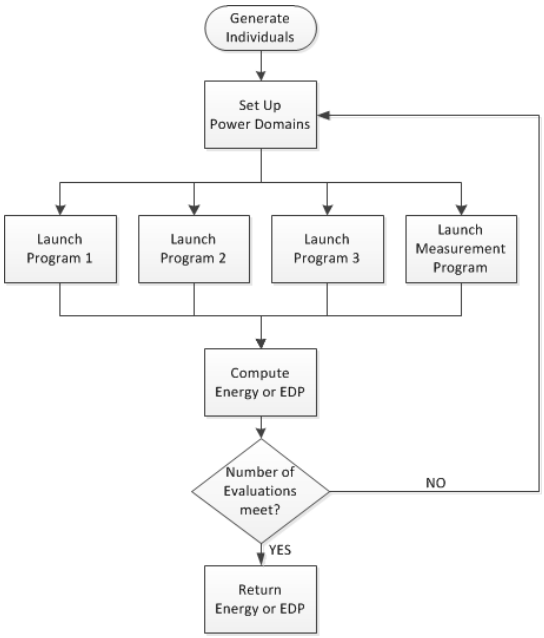

Figure 3. Execution Flowchart on the SCC

$\vec{x}$ is a vector consisting of the gears and the number of cores for each program. For example, C_1 and G_1 represent the number of cores and the gear number of the first program. Each $\vec{x}$ is generated by DEs and evaluated on the SCC.

$$\vec{x} = \begin{bmatrix} G_1 & G_2 & G_3 & C_1 & C_2 & C_3 \end{bmatrix}^T \tag{10}$$

Other constraints are:

1) The test workload on the SCC has three programs, and the total number of cores used by all three programs at any given time cannot exceed 48.
2) All the test programs are at the same priority level.
3) Due to the hardware constraints, each individual program must be run under one same gear, even if it occupies more than one power domain.

C. Implementation on the SCC

For each program in the workload, we select one gear out of four active gears and one idle gear as shown in Table I. The reasons why we selected 4+1 representative gears in our study are:

1) The tuning time will become infeasible and probably never converge due to the huge search space mentioned in Section I.
2) To maintain the SCC's stability, some frequencies are forbidden to reach under a specific voltage. For example, it is prohibited to run one program at a very high frequency with a low voltage.

The SCC platform consists of two computers: the MCPC and the BMC. The MCPC is a host computer which sends the commands via Secure Shell (SSH) to control the BMC. The BMC is a computer which houses the SCC board. The DE algorithms are running on the MCPC.

Figure 3 shows the flowchart of our algorithm on the SCC. To begin with, all the individuals (vectors) of the first generation are randomly generated. Then, they are parsed and sent to BMC so that the number of cores and the gear are distributed to each program (set up power domains) before execution. After all the three programs are running simultaneously on the SCC, the next step is the power measurement. The power consumption of the SCC chip is continuously measured by the auto-tuning program which invokes APIs in the RCCE library. The RCCE is a library supporting message passing interface (MPI) programming and the power measurement of the SCC. Finally, We calculate the EDP or energy of this generation based on Equations 8, 9 after measuring the power and execution time. This whole process is one iteration of the DE algorithm. If the maximum iterations are meet, the final result will be outputted. Otherwise, the DE algorithm mutates the parent generation to generate the new offspring generation and start the evaluation again.

IV. EXPERIMENT SETUP

In our experiments, we run the workload on the SCC with Intel SCCkit v1.4.0. The MCPC runs Ubuntu 10.04 64-bit with 8GB RAM. The executable files were generated by a customized GCC v4.3 compiler by Intel.

The experiments consist of two optimization categories: Energy (in Joules) and EDP (in Joules·Seconds). For energy, we only focus on optimizing the energy consumption of the SCC chip. For EDP, however, we try to find a compromise between performance (execution time) and energy consumption based on the definition of EDP in Equation 8. In both cases, the smaller the better.

In each category, the performance of four DE algorithms (JADE, FSF, RSF, and TVSF) is compared by setting the maximum number of cores of the workload to be either 16 or 32, and the population (all the individuals of one generation/iteration) to be either 15 or 30.

Concerning the control parameters of DE algorithms, the parameters of JADE are fixed as $p = 0.2$ and $c = 0.1$, which are experimental defined. To maintain comparative fairness, the parameters of other three algorithms are set to the same values as described in Section III.

The maximum iteration/generation of DEs in our experiments is set to 10. From a few experiments, we find that, at the early iterations, both the best and worst EDP or energy value of the current generation are declining. But usually after eighth iterations, the best EDP or energy value of the current generation is almost the same as previous generations, only worst case continues declining. The total population is $P \times G$, where P is the population of each generation and G is the maximum generation number.

A. Multi-Programs Workload

In this work, four algorithms are tested under one multi-program workload: cpi (an MPI program), seqpi2 (an sequential program), and nparticles (an serial/parallel hybrid program).

1) cpi: cpi is a parallel program to compute the value of π. The formula used is given in Equation 11. The integration of Equation 11 ranging from 0 to 1 equals the value of π. The cpi program divides the integral range and dispatches different pieces of calculation to all SCC cores via the MPI interface.

$$\pi = \int_0^1 \frac{4}{1+x^2} dx \tag{11}$$

2) seqpi2: seqpi2 is a sequential version of cpi which also computes the value of π. Since this program only uses one single thread, using more cores does not contribute to faster execution time.
3) nparticles: nparticles is a mixed parallel and sequential program. The n-particle simulation is one of the classical N-body problem simulating the evolution of a system of N bodies, where the force exerted on each body arises due to its interaction with all the other bodies in the system. The simulation proceeds in time steps, each time computing the net force on every body and thereby updating its position and other attributes.

V. Experimental Results

In this section, the experiment results of EDP optimization and energy optimization are presented. In both cases, the results of maximum 16 and 32 cores for each program are compared.

A. Result Comparison for EDP

1) EDP with 16 max cores: Figure 4 shows the result of EDP with 15 individuals and maximum 16 cores. The population of the first generation, regarding DE and DE variants, is uniformly picked among the entire search space. This is why the first generation normally has similar EDP readings across the four algorithms. From the second to the last generation, the behavior of four algorithms differs mainly due to different mutation and crossover strategies. Although FSF, TVSF, and RSF find its optimal result at the 8th generation, JADE shows superior performance in terms of EDP reduction.

Figure 5 shows the result of EDP consumption with 30 individuals and maximum 16 cores. Since population doubles at every generation, DEs explore larger search space than 15 individuals. Therefore, the result of 30 individuals outperforms the counterpart of 15 individuals at almost every generation. We can also conclude that JADE converges faster and achieve better results than other three algorithms. The main reason for that is JADE performs internal rankings in each generation, and the offspring have high possibilities to be generated based on top $p\%$ ranking parents according to its mutation strategy.

2) EDP with 32 max cores: Figure 6 shows the result of EDP consumption with 15 individuals and maximum 32 cores. In this case only one program will be distributed to 32 cores since the SCC hosts only 48 cores. We can see all the algorithms find closer EDP values from the 6th generation onwards. The reason is the range of EDP values shrinks remarkably as long as more than 20 cores are distributed to cpi, which also explains why the result of 30 individuals with 32-core scenario has even closer EDP values as shown in Figure 7. However, the optimal result of the 32-max-core scenario achieves about 20% improvement over the counterpart of the 16-max-core scenario. This verifies that more adaptive cores distribution is necessary to achieve higher energy efficiency and better execution time.

3) Population Selection: In the 16-max-core scenario, the best result of 30 individuals shows superior performance which consumes 25% less EDP than 15 individuals. Moreover, in the 32-max-core scenario, the best result of 30 individuals outperforms the counterpart of 15 individuals by 19.6%. Therefore, all the data in the following experiments are generated with 30 individuals unless otherwise mentioned. The overhead of doubling the population nearly extended the experimental period by 50%. However, compared with manual tuning, the overhead of 30 individuals is trivial since the total population explored by DEs is negligible relative to the entire search space.

B. Result Comparison for Energy

1) Energy with 16 max cores: Figure 8 shows the result of energy consumption with maximum 16 cores. Initially, the four algorithms all started at around 420J. As we move forward from th second to the last generation, we can observe that the huge gap exists between JADE and other three algorithms. JADE not only shows better performance in energy consumption but also converges 3 generation ahead of others. For other three algorithms, they are "trapped" at local minima around 300J.

2) Energy with 32 max cores: Figure 9 shows the comparison of energy consumption with maximum 32 cores. Comparing with the 16-max-core case, the result range of the 32-max-core case in energy and EDP is wider between

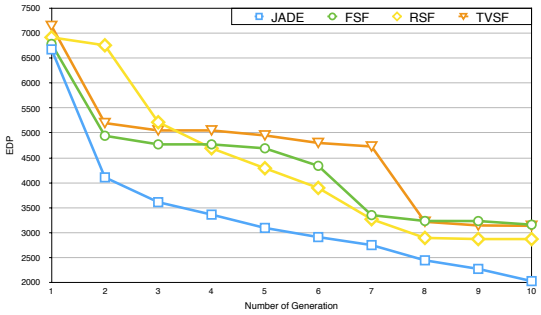

Figure 4. EDP consumption with 15 individuals and 16 max cores

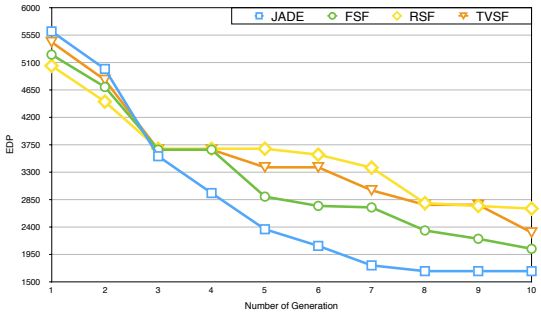

Figure 5. EDP consumption with 30 individuals and 16 max cores

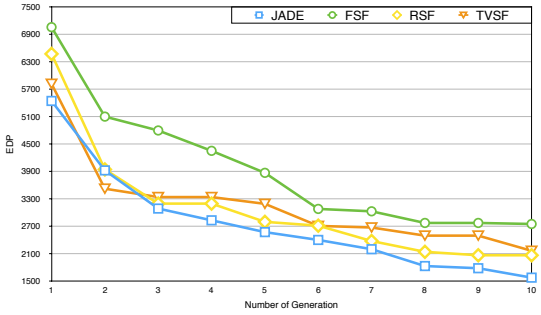

Figure 6. EDP consumption with 15 individuals and 32 max cores

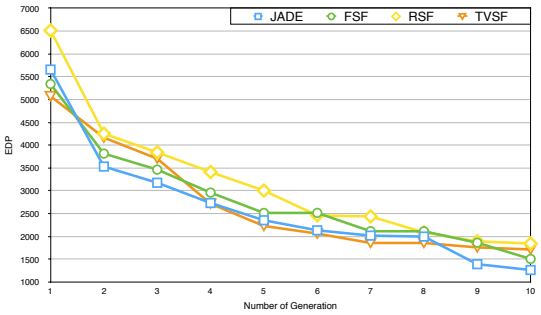

Figure 7. EDP consumption with 30 individuals and 32 max cores

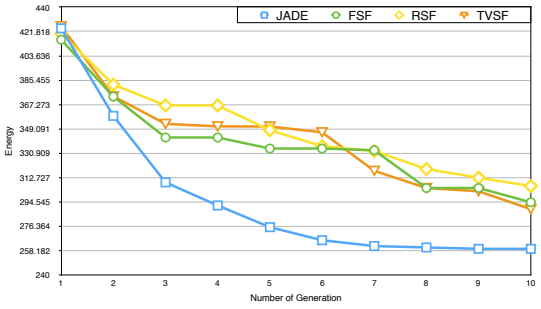

Figure 8. Energy consumption with 30 individuals and 16 max cores

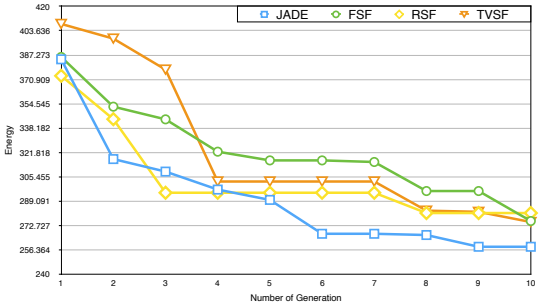

Figure 9. Energy consumption with 30 individuals and 32 max cores

the low core count and high core count. The DE algorithms would know better which direction to optimize. As a result, the gap between JADE and FSF, RSF, TVSF is narrower than that of 16 max cores. FSF, RSF, and TVSF can achieve even lower energy consumption by 6.4%, 8.1%, and 4.8%, respectively. The energy consumption of the optimal candidate of JADE is almost the same as that of 16 max cores. Ideally, the energy consumption of the best candidate of JADE with 32 max cores should be even lower, the reason of which will be explored next.

C. The Best Candidate Settings

In this section, we choose the JADE's best candidate of each generation to analyze the optimization direction and compare their optimal settings (the number of cores and the gear) in terms of different criteria: EDP and energy. The setting mappings to programs in following figures are listed as follows:

- nparticles — core1 and gear1;
- cpi — core2 and gear2;
- seqpi2 — core3 and gear3.

1) EDP vs energy in 16 max cores: In Figures 10 and 11, although the search directions of them are different from beginning to the 6th generation, the DE algorithms ended up with the same settings for cpi and seqpi2. cpi used up all the computing resources (16 cores and Gear 1); seqpi2 ended with only one core and Gear 1. Adding more cores to the sequential program seqpi2 does not contribute to better performance in terms of either execution time or power consumption. The only difference between EDP and energy settings is for nparticles. Only using two cores for nparticles is more energy efficient than EDP's final setting with shorter execution time using 6 cores. Based on the benchmark programs we studied in this work and the SCC

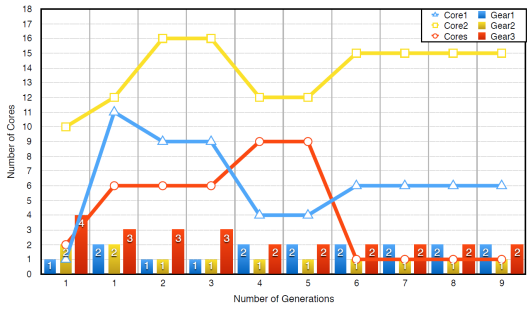

Figure 10. The best candidate settings of EDP in each generation with 16 max cores

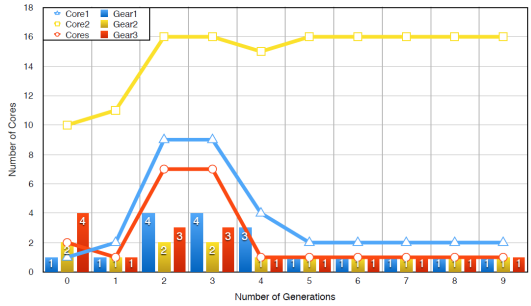

Figure 11. The best candidate settings of energy in each generation with 16 max cores

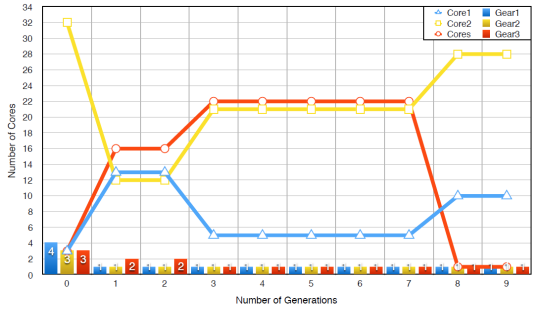

Figure 12. The best candidate settings of EDP in each generation with 32 max cores

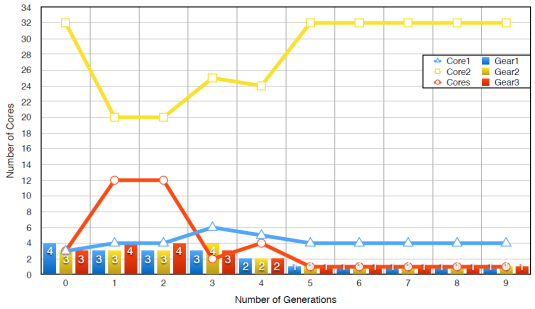

Figure 13. The best candidate settings of energy in each generation with 32 max cores

hardware platform, the trade-off between performance and energy or EDP consumption is the shorter execution time weights more than lower gear settings with longer time.

2) EDP vs energy in 32 max cores: In Figures 12 and 13, our results are very similar to the results of the 16 max cores, except cpi in energy scenario does not use up all the 32 cores as in EDP scenario. In the ideal case, cpi should use up all the 32 cores so as to minimize the delay. Therefore, it implies that the final candidate of JADE is suboptimal in Figure 12, which we will discuss next.

D. The Best Candidate Settings of JADE vs. Baseline

In this section, we create a baseline approach. This approach is no longer auto-tuning based, but rely on a priori knowledge of the programs. We want to verify how close our best candidate from JADE is to the minima of this baseline approach. From previous sections, the results of DEs show that the short execution time is more important. Taking advantage of this, our baseline approach is set up as follows: 1) Running all the three programs at the highest gear; 2) The cpi runs with the maximum number of cores; 3) The seqpi2 runs with only one core; 4) The nparticles runs with the cores from 1 to maximum. The results are shown in Tables II, III, IV, and V. Although JADE only find the same best candidate as baseline approach in Table V, overall we feel it is very impressive that JADE can achieve results very close to the baseline case, considering JADE has no a priori knowledge of the workload while need to transverse a huge search space.

VI. Conclusions

In this work, we achieved energy-aware automatic tuning of the voltage of frequency level of a many-core platform by employing Differential Evolution (DE) algorithms. We conducted quantitative analysis by cross-comparisons among the four different DE algorithms, i.e., FSF, RSF, TVSF, and JADE. Compare with other algorithms, JADE is able to achieve better energy consumption and energy-delay product with a faster converge rate. When repeated tasks occur regularly on many-core platforms, by employing our approach we can attain both energy-aware and high-performance computing afterwords. Another advantage of DE algorithm is it can help us improve the energy consumption and energy delay product by only exploring a small portion of the entire search space, which would be infeasible to explore with manual approach otherwise.

Theoretically, our approach can handle a great number of programs without knowing any properties of them. Concerning the future work, we want to study how our approach scales with more programs. Besides, it is interesting to test memory-intensive and I/O-intensive programs. We desire to compare our results with other optimization algorithms. We want to continue study how to find out the global minimum of the entire search space, how far away we are currently and how to achieve it.

Table II
THE BEST CANDIDATE OF JADE VS. MANUAL TUNING FOR EDP WITH 16 MAX CORES

	JADE		Manual	
Program	Gear	Cores	Gear	Cores
cpi	1	16	1	16
seqpi2	1	1	1	1
nparticles	1	6	1	3
Best EDP (J · s)	1676.01		1424.15	

Table III
THE BEST CANDIDATE OF JADE VS. MANUAL TUNING FOR EDP WITH 32 MAX CORES

	JADE		Manual	
Program	Gear	Cores	Gear	Cores
cpi	1	28	1	32
seqpi2	1	1	1	1
nparticles	1	6	1	4
Best EDP (J · s)	1364.79		1325.89	

Table IV
THE BEST CANDIDATE OF JADE VS. MANUAL TUNING FOR ENERGY WITH 16 MAX CORES

	JADE		Manual	
Program	Gear	Cores	Gear	Cores
cpi	1	16	1	16
seqpi2	1	1	1	1
nparticles	1	2	1	4
Best Energy (J)	258.18		242.32	

Table V
THE BEST CANDIDATE OF JADE VS. MANUAL TUNING FOR ENERGY WITH 32 MAX CORES

	JADE		Manual	
Program	Gear	Cores	Gear	Cores
cpi	1	32	1	32
seqpi2	1	1	1	1
nparticles	1	4	1	4
Best Energy (J)	256.36		256.36	

ACKNOWLEDGMENT

The authors would like to thank Intel Labs for providing us with the SCC platform to conduct this research. The authors would also like to thank the anonymous reviewers for their feedbacks that greatly improved the quality of this paper. This material is based upon work supported by the National Science Foundation under Grant No. ECCS-1301953. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of either Intel or the National Science Foundation.

REFERENCES

[1] I. Labs, "Scc platform overview," *Revision 0.75,Intel Many-Core Application Research Community*, 2010.

[2] P. Salihundam, S. Jain, T. Jacob, S. Kumar, V. Erraguntla, Y. Hoskote, S. Vangal, G. Ruhl, P. Kundu, and N. Borkar, "A 2 tb/s 6× 4 mesh network with dvfs and 2.3 tb/s /w router in 45nm cmos," in *VLSI Circuits (VLSIC), 2010 IEEE Symposium on*. IEEE, 2010, pp. 79–80.

[3] M. A. Panduro, C. A. Brizuela, L. I. Balderas, and D. A. Acosta, "A comparison of genetic algorithms, particle swarm optimization and the differential evolution method for the design of scannable circular antenna arrays," *Progress In Electromagnetics Research B*, vol. 13, pp. 171–186, 2009.

[4] S. Das, A. Konar, and U. K. Chakraborty, "Two improved differential evolution schemes for faster global search," in *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. ACM, 2005, pp. 991–998.

[5] J. Vesterstrøm and R. Thomsen, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems," in *Evolutionary Computation, 2004. CEC2004. Congress on*, vol. 2. IEEE, 2004, pp. 1980–1987.

[6] S. Das and P. N. Suganthan, "Differential evolution: a survey of the state-of-the-art," *Evolutionary Computation, IEEE Transactions on*, vol. 15, no. 1, pp. 4–31, 2011.

[7] C.-h. Hsu and W.-c. Feng, "A power-aware run-time system for high-performance computing," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 2005, p. 1.

[8] Y. Hotta, M. Sato, H. Kimura, S. Matsuoka, T. Boku, and D. Takahashi, "Profile-based optimization of power performance by using dynamic voltage scaling on a pc cluster," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 2006, pp. 8–pp.

[9] V. Pankratius and S. Blaese, "Application-level automatic performance tuning on the single-chip cloud computer," in *3rd Many-Core Aplications Research Community (MARC) Symposium*. Citeseer, 2011, pp. 1–6.

[10] K. Berry, F. Navarro, and C. Liu, "Application-level voltage and frequency tuning of multi-phase program on the scc," in *Proceedings of the 3rd International Workshop on Adaptive Self-Tuning Computing Systems*. ACM, 2013, p. 1.

[11] B. Roscoe, M. Herlev, and C. Liu, "Auto-tuning multi-programmed workload on the scc," in *Green Computing Conference (IGCC), 2013 International*. IEEE, 2013, pp. 1–5.

[12] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[13] V. Plagianakos and E. W. Weisstein, "Differential evolution," 2015. [Online]. Available: http://mathworld.wolfram.com/DifferentialEvolution.html

[14] J. Zhang and A. C. Sanderson, "Jade: adaptive differential evolution with optional external archive," *Evolutionary Computation, IEEE Transactions on*, vol. 13, no. 5, pp. 945–958, 2009.