DOI 10.12694/scpe.v19i1.1390
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 19, Number 1, pp. 15–29. http://www.scpe.org 

# SCALE-EA: A SCALABILITY AWARE PERFORMANCE TUNING FRAMEWORK FOR OPENMP APPLICATIONS

SHAJULIN BENEDICT*

**Abstract.** HPC application developers, including OpenMP-based application developers, have stepped forward to endeavor the future design trends of exa-scale machines, such as, increased number of threads/cores, heterogeneous architectures, multiple levels of memories, and so forth; and, they have initiated procedures to address application level challenges, such as, data-driven scalability issues, energy consumption requirements, data availability needs, and so forth. Despite the existence of manual performance tuning solutions, users still deem it to be an intricate process. This paper proposes a scalability aware autotuning framework (SCALE-EA) that automatically identifies an efficient number of threads for OpenMP parallel regions using a Firefly Algorithm (FA) and a newly designed Modeling Assisted Firefly Algorithm (MAFA). MAFA of SCALE-EA was implemented in two approaches: Modeling Assisted Firefly Algorithm with Random Forest Modeling support (MAFA-RFM) and Modeling Assisted Firefly Algorithm with Linear Regression Modeling support (MAFA-LRM). The modeling and prediction algorithms of the proposed MAFA of SCALE-EA were based on the execution time and the hardware performance events of code regions of OpenMP applications. Experiments were conducted on two machines, namely, a Haswell based machine and an AMD Opteron based 48 core machine. The experimental results of the MAFA of SCALE-EA manifested the energy efficiencies of 31.21 to 77.3 percentage and the search time efficiencies of 5.53 to 32.56 percentage for candidate OpenMP applications such as CoMD, Arraybench, Taskbench, and Syncbench.

**Key words:** Auto Tuning, OpenMP Applications, Modeling and Scalability

**AMS subject classifications.** 68M20, 68W10

## 1. Introduction.

High Performance Computing (HPC) application developments are invariably cropping up among various scientific domains, such as, High Energy Physics (HEP), bioinformatics, eyewear computing, visualizations, electronic automation, graph-based machine learning, and so forth. OpenMP based programming model is indeed reaching out to become a prominent programming model among a sector of HPC application developers owing to the adequate doctrine of standards (OpenMP 4.0 and 4.5), ease of use, controlled programming support, smooth applicability to programmers belonging to various scientific disciplines, and due to the notion of having millions of cores in future exascale machines.

However, the realization of efficiently utilizing HPC applications in its present form for future large scale machines requires innovative approaches to mitigate the following possible risky scenarios:

1. the performance of applications becomes more sensitive to data movement, data availability, data provenance, data management policies, and so forth – a future software-cum-hardware computing system must consider the massive storage options of machines, resiliency nature of applications, dynamic computing behavior of applications, and the dynamic nature of the data access patterns of applications (big data).
2. the current implementations of OpenMP applications might not have considered the design aspects of emerging memory models (including data persistence of modern memory architectures), infrastructural improvements, future parallel data structures, and so forth.
3. the scalability of applications might get an impoverished lead as applications are usually not ported and tested for scalable machines.
4. the energy efficiency of applications could exhibit a daunting scenario when executed on machines with varying degrees of parallelism – smaller or larger.
5. the current OpenMP application developers might not have quantified the possible uncertainties that might evolve due to the underlying future parallel software frameworks.

In short, to mitigate these challenges, programmers or developers have to diligently write scalable and energy efficient parallel algorithms by employing the apt scalability features of programming languages and by considering the underlying requirements of machines. Vividly, OpenMP based application programmers have to gain sophisticated knowledge on handling the newer OpenMP constructs in order to attain higher scalability, portability, and energy efficiency – for instances, the synchronization points of OpenMP applications, such as,

*Indian Institute of Information Technology Kottayam, Kerala, India - 695016. (shajulin@iiitkottayam.ac.in) – Formerly the Guest Professor of Technical University Munich, Germany.

OpenMP barrier constructs should be limited or enriched; the granularity of OpenMP parallel regions should be optimized to achieve higher scalability; and, the programmers should efficiently utilize the OpenMP private data clauses in an application.

In general, the non-scalable and the energy/performance inefficient code regions of an application can be identified and manually tuned. However, the manual tuning processes might laid down hefty responsibilities to users – an intricate process. One solution that makes sense and intends to work in large scale HPC scenarios is to establish an autotuning tool or a framework that automatically finds the non-scalable or energy inefficient parts of the code regions of applications and tunes them accordingly. Historically, a notion of framing autonomic computing systems was highly appreciated by researchers for decades [11, 29]. Moreover, researchers had shown their keen interest in counteracting the scalability and the energy consumption issues of applications [2, 4, 15, 31] at various levels of computing systems so that the existing applications could be executed on future machines at ease.

Auto tuning, although a promising solution for tuning HPC applications, easily leads to voluminous combinations of optimization options which might impede the search time of the tuning process. In addition, the autotuning process could inundate the search space with an immense volume of performance data, as autotuning systems are, in general, self-aware, self-configurable [42], context-aware, and self-optimizable systems. In most cases, the autotuning systems are guided in a single-objective or in a multi-objective modes of operations based on pre-defined objectives, such as, minimizing energy consumption of code regions of applications, maximizing the utilization of machines, minimizing the data movement to Dynamic Random Access Memory (DRAM), and so forth. Thus, there is an increasing need for autotuning solutions or frameworks that intelligently minimizes the search time of the tuning processes.

This paper proposes the SCALE-EA framework, a Scalability-Aware performance tuning framework using EnergyAnalyzer (EA), for OpenMP applications – EnergyAnalyzer tool is an online based energy consumption analysis tool for HPC applications. SCALE-EA automatically identifies the efficient number of threads for individual parallel regions of OpenMP applications using FA and MAFA. The proposed FA and MAFA of SCALE-EA improved the search time of the tuning process and enhanced the energy efficiency of applications. The proposed method was evaluated using several OpenMP applications / benchmarks, such as, CoMD from the co-design center of LLNL, USA [14], and OpenMP benchmarks from EPCC [17], on a Haswell processor based HP-Zbook-15G machine and a 48 core HPProliant machine.

In succinct, the paper has the following contributions:

1. an SCALE-EA framework for automatically finding an efficient number of threads for the parallel regions of OpenMP applications was proposed.
2. FA and modeling assisted heuristic algorithms were implemented in SCALE-EA in order to reduce the search time of the tuning process. The modeling assisted heuristic algorithms were implemented in two flavors: a) Modeling Assisted Firefly Algorithm using Linear Regression Modeling (MAFA-LRM) and b) Modeling Assisted Firefly Algorithm using Random Forest Modeling (MAFA-RFM). MAFA algorithms are the modified versions of FA.
3. Experimental evaluations of the proposed SCALE-EA and its associated algorithms were carried out using OpenMP applications, such as, Arraybench, Syncbench, Taskbench, and CoMD applications.

The rest of the paper is organized as follows. Section 2 presents existing autotuning frameworks and their solutions. Section 3 explains the proposed scalability-aware performance tuning framework for OpenMP applications and Section 4 discusses the modified firefly algorithm which adds modeling support towards the search process. Section 5 manifests the proposed mechanism called the SCALE-EA framework. And, finally, Section 6 presents conclusions.

**2. Related Works.** HPC applications, specifically data-driven applications, are increasing in the scientific market from various scientific disciplines, such as, massive graphs, HEP, bioinformatics, healthcare, distributed manufacturing, electronic automation, visualization applications (social networks), multi-physics simulations, and so forth. In the meantime, technological advances in hardware architectures are nearing exascale speed through co-design architectural designs, abundant General Purpose Graphical Processing Units (GPGPUs), hierarchical clustering of heterogeneous machines, and so forth. Despite the growth seen in the application sector and in the hardware architectural design sector of HPC, the performances of applications, including the

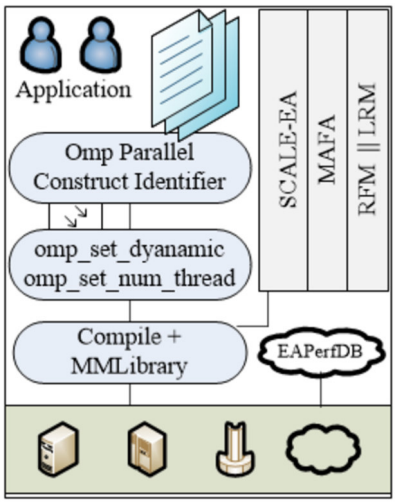


FIG. 3.1. *SCALE-EA: Scalability Aware Performance AutoTuning Framework for OpenMP Applications*

scalability and the energy efficiency of applications, should be concerned about by the researchers.

The HPC community, therefore, oriented their mindset to mitigate the effects of known performance issues of large scale systems such as the dynamic nature of big data in applications (data sizes), heterogeneous hardware architectures [7], energy consumption issues, scalability issues [22, 39], uncertainty of resources (including data resources), and so forth [18].

In fact, the energy consumption issues of HPC applications should be addressed due to the scarcity of power sources (especially in the developing countries, such as, India), owing to the emission of carbon footprints [10] which lead to environmental hazards, and, due to the emerging power wall problem of dark silicons [31].

There exists a few standalone energy reduction mechanisms for HPC applications [28, 36, 21]. The most of the existing approaches are either manual [34] or application-centric.

Researchers have proposed autotuning frameworks/solutions [16, 23, 24, 25, 13] in order to achieve performance/energy improvements. Studies have also led a subset of researchers to frame autotuning solutions at compile time [2, 9, 12, 27, 32] and a few others at run time of applications [37, 3, 40, 8]. In addition, a few autotuning solutions with more emphasis to IOs were designed in [5]. To improve the performance of heterogeneous systems and applications, the authors of [6] have designed an autotuning solution. Additionally, tool developers are constantly finding mechanisms to assist application developers in terms of automatically improving the performance efficiency of HPC applications.

To avoid the burdens caused due to the search time and the emergence of abundant performance data, a few researchers have utilized modeling mechanisms [26, 1] to forecast the performance issues of applications and to tune applications. In addition, modeling assisted automated systems have been successful in cloud and distributed environments [30]. Recently, researchers have adopted autotuning using Domain Specific Languages for autotuning applications in ANTAREX project [13].

This paper proposed a combination of heuristics and modeling approach for the tuning process of OpenMP based HPC applications. To do so, Firefly Algorithm (FA), a heuristic method, is modified with modeling algorithms, such as, RFM and LRM, in this paper. A detailed description about the proposed SCALE-EA framework based on FA and MAFA algorithms is discussed in Section 3.

**3. Scalability Aware AutoTuning using EnergyAnalyzer (SCALE-EA).** SCALE-EA framework does scalability aware performance autotuning of OpenMP applications. The framework is built based on EnergyAnalyzer tool, an online based energy consumption analysis tool. This section explains the SCALE-EA framework and the entities involved in pursuing the performance aware autotuning mechanism for OpenMP applications.

**3.1. SCALE-EA Framework Functionalities.** The functionalities of the proposed SCALE-EA framework, the extensions made towards EnergyAnalyzer tool, are described as follows:

*SSTranslator and its Extension.* SSTranslator, an entity of EnergyAnalyzer tool, is extended to support the proposed SCALE-EA framework. It is a source-to-source translator which includes program statements, such as,

```
#pragma start_user_region
---
#pragma end_user_region
```

These statements notify the Monitoring Manager entity of EnergyAnalyzer about the user-specified code regions of applications. The performance / energy consumption values of these user-specified regions are measured using the Monitoring Manager entity of EnergyAnalyzer at the runtime of applications.

The extended version of SSTranslator additionally identifies each parallel regions of OpenMP based applications. These parallel regions are marked with two additional statements, such as,

```
omp_set_dynamic(0);
omp_set_num_threads(SCALEEA_NUM_THREADS_filename_n);
```

where $n$ resembles the unique number for each *omp parallel* regions in an application and $filename$ is the filename representation of applications.

The line `omp_set_dynamic(0)` is automatically added before the *omp parallel* regions of OpenMP-based applications using the extended version of SSTranslator. This statement instructs compilers to disable dynamic allocation of threads. The variable `SCALEEA_NUM_THREADS_filename_n` is automatically defined at the initial stage of the file using $\#ifdef$ conditions of C/C++ definitions. This is mandatory for the proposed SCALE-EA framework as the framework would later assign the number of threads for each *omp parallel* regions. The heuristics of SCALE-EA are responsible for assigning the values to these variables.

In addition, the line numbers for *omp parallel* regions and the file names of applications where the regions belong to are noticed in a separate *.sst* file (static source-to-source file) of the SCALE-EA framework.

*Performance/Energy Measurements.* After inserting the required statements for SCALE-EA, (for instances, the statements which would disable the dynamic allocation of threads to OpenMP applications and enable performance measurements), the application is compiled with the MMLibrary of EnergyAnalyzer. The MMLibrary of EnergyAnalyzer measures the performance values of the user-specified code regions of applications and stores the performance values in EAPerfDB, a no-sql (Mongodb) based performance database of EnergyAnalyzer tool.

*Heuristic Support.* Selecting efficient numbers of threads for each *omp parallel* regions of OpenMP application is a time consuming task. Thus, SCALE-EA framework depends on heuristics which suggest efficient numbers of threads for each *omp parallel* regions.

In this paper, Firefly Algorithm (FA) is applied in the SCALE-EA framework in order to find the efficient number of threads for OpenMP applications. In addition, the time spent for finding the efficient number of threads (the search time) is further reduced using the modified versions of FA – MAFA-RFM and MAFA-LRM. Detailed discussions on the Firefly Algorithm (FA) and MAFA can be found in Section 4.

**4. FA and MAFA of SCALE-EA.** In this paper, FA and a modified version of FA, namely, Modeling Assisted Firefly Algorithm (MAFA) are proposed for identifying an efficient number of threads for OpenMP applications. MAFA is implemented in two approaches, namely, MAFA-RFM and MAFA-LRM. This section describes the FA and MAFA in detail.

**4.1. Firefly Algorithm (FA).** Firefly Algorithm (FA) is a meta-heuristic algorithm which is actively utilized by a few researchers [41, 19, 35] in order to solve their real-world combinatorial optimizations problems. FA was developed in 2007 by Xin-She Yang at Cambridge University [41]. In fact, the idea of meta-heuristics intrigued the real world problem solvers when compared to the classical optimization techniques for decades owing to the algorithms' problem independent nature – the classical optimization techniques find solutions based on analytical models (continuous or differentiable functions).

FA, an iterative based heuristic technique or a population based meta-heuristics, was introduced in several flavors, such as, adaptive, multi-objective, hybrid, DiscreteFA, and so forth. A wide survey of FA and its

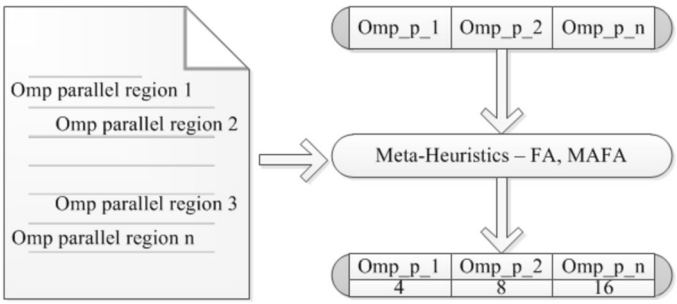


FIG. 4.1. *Realization of FA in SCALE-EA – Thread String Formation*

derivatives was elaborated in [20].

FA was named after fireflies which exist in tropical geographical locations. In general, FA depends on the flashing and mating communication behavior of fireflies. In succinct, the male fireflies lit a specific flashing behavior and the females subsequently respond to the flash with unique characteristics. The bonding between the male and the female fireflies heavily rely upon the distance between them. FA assumes the following:

1. Fireflies of FA are unisex. This ensures that each firefly would be attracted to the other available fireflies irrespective of what sex they belong to.
2. The attractiveness of fireflies is dependent on the brightness of the fireflies, which is the the objective function of FA.

**4.2. Realization of FA in SCALE-EA – Problem Definition.** SCALE-EA based on FA identifies efficient number of threads for each *omp parallel* regions of OpenMP applications. To do so, the following steps are carried out in the SCALE-EA framework:

*SCALE-EA Preparatory Phase.* As mentioned earlier, SSTranslator of EnergyAnalyzer includes statements, such as, `omp_set_num_threads(SCALEEA_NUM_ THREADS_filename_n)` for each OpenMP parallel regions of applications. Later, in the SCALE-EA preparatory phase, SCALE-EA represents those statements in a string form so that the heuristics could literally assign the number of threads for applications at the initialization and the iterative phases of SCALE-EA. The string form representation of the parallel regions of OpenMP applications is given as shown below:

```
ompP_1 ompP_2 ompP_3 ... ompP_n
```

where $n$ represents the number of parallel regions in an application. For instance, Figure 4.1 shows the assignment of threads 4, 8, and 16 for three parallel regions `ompP_1 ompP_2 and ompP_n`.

The string form based on the parallel regions of applications are uniquely represented irrespective of the files of applications. For instance, the eleven *omp parallel* regions from 3 files, namely, eam.c, initAtoms.c, and ljForce.c, of the CoMD application are represented as `ompP_1 ompP_2 ompP_3 ompP_4 ompP_5 ompP_6 ompP_7 ompP_8 ompP_9 ompP_10 ompP_11` based on parallel regions `SCALEEA_NUM_THREADS_eam_1` (for `ompP_1`) to `SCALEEA_NUM_THREADS_ljForce_2` (for `ompP_11`).

*FA Initialization Phase.* During the initialization phase, the initial values of FA parameters are set. The FA paramters include the number of initial population of sequences (fireflies), the objective functions, the number of generations applied in FA, and epsilon, alpha, beta, and gamma values. In the experiments, the impacts of six combinations of FA parameter variations are studied as shown in Table 5.3.

In addition, during this initialization phase of SCALE-EA, the initial population of sequences ($F_n$) is generated. The pictorial representation of the sequences in the initialization phase of the SCALE-EA mechanism for an application is shown in Figure 4.2.

Based on the initialization phase values, SCALE-EA assigns the number of threads for each *omp parallel* regions and execute them on the underlying HPC machine. Meantime, the performance values for the user-specified code regions of applications are recorded in the EAPerfDB of SCALE-EA. Subsequently, the objective function values of these sequences are noticed for the next phase of FA.

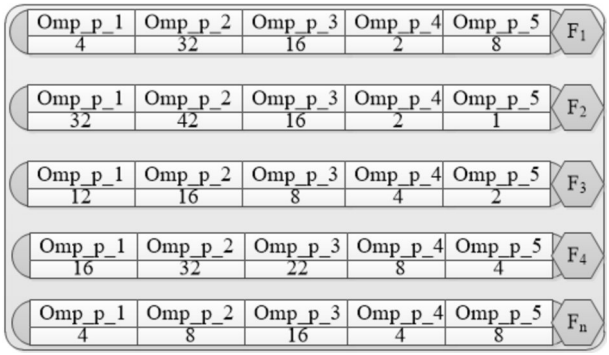


FIG. 4.2. *Initialization Phase of FA at SCALE-EA*

*FA Iteration Phase.* During this phase, FA iteratively evaluates the light intensities of the two series of sequences ($F_{i=1\ to\ n} and F_{j=1\ to\ n}$) based on the objective functions. In fact, the objective functions of sequences decrease with respect to the distance between them. The objective functions applied in SCALE-EA is a Combined Simultaneous Objective (CSO) function with weight functions as shown in equation 4.1. The weight functions are mandatorily devised for CSO – 50 percentage weights were given to the scalability parameter and the other 50 percentage weights were splitted into 20, 20, and 10 for execution time, energy and performance parameters of equation 4.1:

$$CSO = 0.5 * (\frac{ET}{ET_{p=1}}) + 0.2 * ET + 0.2 * EY + 0.1 * gistPerf \tag{4.1}$$

where: $ET$ is the execution time of $F_n$ in seconds; $EY$ is the energy consumption of $F_n$ in Joules; $gistPerf$ is represented using formula 4.2, where the performance values are converted to a two-decimal rounded off values:

$$gistPerf = \sum_{i=1...pf} Perf_i \tag{4.2}$$

where $pf$ is the number of performance counters, and $Perf_i$ is the performance value for each performance counters such as Level 1 cache misses, Level 3 cache misses, unconditional branches, and so forth. For example, if the total number of instructions, level1 data cache misses, unconditional branches, and level3 total cache misses for an application are measured as 11960995473, 5486, 869583951, and 3801, the $gistPerf$ value would be rounded off as 1.28 (actual value = $1.28 * 10^{10}$).

If the light intensity $IF_j$ of $F_j$ is better than $IF_i$, then FA moves $F_i$ to $F_j$. In SCALE-EA, the movement of firefly $F_i$ to an attractive firefly $F_j$ is determined using the formula given in equation 4.3:

$$Movement\ F_i^{t+1} = F_i^t + \beta_0 e^{-\gamma r_{ij}^2}(F_j^t - F_i^t) + \alpha_t \epsilon_i^t \tag{4.3}$$

where the first component represents the previous $F_i$ sequence (thread sequence); the second component represents the essence of attraction parameter. This component relates to the distance between two fireflies – $\beta_0$ represents attractiveness at distance 0 and $r$ relates to the distance between them at a constant value $\gamma$; and the third component represents the randomization parameter – $\alpha_t$ is a scalable value specified at time $t$ and $\epsilon$ is a random number for each iteration of FA.

The second component of equation 4.3 moves the sequences in a discrete manner instead of pursuing a bare continuous calculation.

*FA Ranking Phase.* In this phase, the sequences are ordered and the local best sequence $LB\_F\_g$ for that particular generation of sequences are noticed. FA continues the ranking process for each generations until the final iteration are complete. And, finally, FA searches for the global best sequence $GB\_F\_g$ for the OpenMP application from the list of available $LB\_F\_g$ and their corresponding light intensity values.

Thus, identifying the efficient number of threads for the parallel regions of OpenMP applications is based on the global best sequence which is derived from the pre-defined objectives (minimizing execution time, minimizing energy consumption and increasing speedup) calculated using the CSO formula (see equation 4.1).

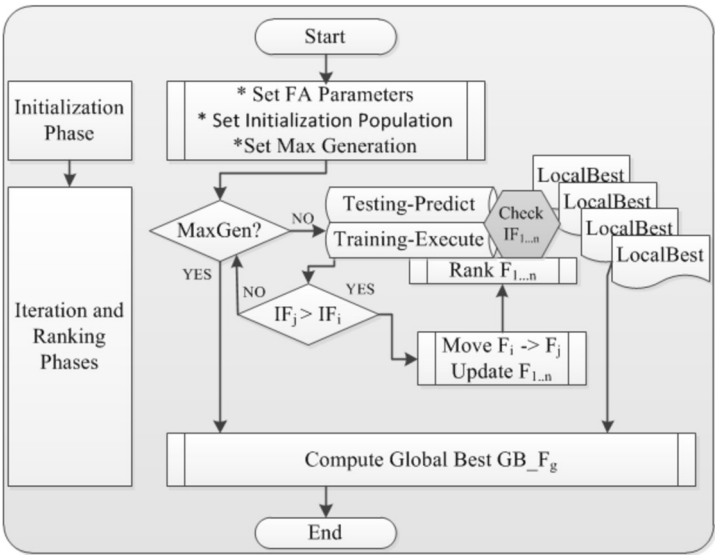


FIG. 4.3. *Modeling Assisted Firefly Algorithm (MAFA)*

**4.3. Modeling Assisted Firefly Algorithm (MAFA) – a modified FA.** Although classical FA would reduce the time for searching the efficient number of threads in OpenMP applications, the search time of the SCALE-EA tuning process can further be improved using prediction models. Reducing the search time of tuning process is crucial while pursuing autotuning in large scale machines. For instance, an application with 9 OpenMP parallel regions might have 134217728 number of combinations of thread sequences when experimented with 8 threads. This could deliberately issue hefty number of performance data for each combinations of thread sequences. SCALE-EA, thus, proposed a Modeling Assisted Firefly Algorithm (MAFA) in order to further reduce the search time of the SCALE-EA tuning process.

In MAFA, a few iterative portions of FA are executed and the other remaining iterative portions of FA are predicted using prediction algorithms, such as, Random Forest Modeling (RFM) and Linear Regression Models (LRM). In the previous works, prediction algorithms were applied for predicting the efficient problem sizes [38] of applications at compile time. In this paper, these prediction algorithms are applied for the first time for assisting the search process of FA based meta-heuristics – i.e., a modified implementation of firefly algorithm. Based on the utility of RFM or LRM, MAFA is represented as MAFA-RFM and MAFA-LRM.

The working principle of MAFA is pictorially represented in flow chart (see Figure 4.3). In general, MAFA assumes the four phases as similar to FA – Preparatory, Initialization, Iteration, and Ranking phases. During the initialization phase of MAFA, the initial parameters of FA are set; initialization population is defined; and, the maximum number of generations for the tests is initialized. Later, MAFA iteratively evaluates sequences and their corresponding intensities (objective functions) for the pre-assigned number of generations as discussed in Section 3.

The modification adopted to FA resides in the iterative phase of MAFA. During the iteration phase of MAFA, all of the combinations of thread sequences that are suggested by FA are not executed. Thus, instead of executing and evaluating all of the combinations of sequences (the thread sequences of OpenMP applications), some of the combinations of thread sequences are executed and the others are predicted using prediction algorithms. To do so, initially, the sequences are enrolled in the Training-Execute list (see Figure 4.3). This list ensures that SCALE-EA executes the OpenMP application during the evaluation step of FA. After a few iterations, there are sufficient number of entries in the Training-Execute list (for instance, 100 entries in the Training-Execute list of SCALE-EA), the next 30 percentage of the population in the FA generation is enrolled in the Testing-Predict list of SCALE-EA.

If the sequence is listed in the Testing-Predict list of SCALE-EA, the OpenMP application with the corresponding thread sequence would not be compiled or executed in the underlying machine. Rather, the modeling-

cum-prediction component of MAFA is invoked by SCALE-EA. During this step, the RFM/LRM prediction algorithms model (and predict) the outcomes of sequences based on the performance counter values (based on the experimental results) available in the EAPerfDB of SCALE-EA.

A detailed insights on the RFM and LRM algorithms are explained in the following paragraphs:

*Random Forest Modeling.* RFM applies tree based technique for modeling the independent variables and for predicting the dependent variables. It gains knowledge using ensemble learning methods; it predicts the unknown variables after a creation of the high variance of de-correlated trees was done. Thus, RFM has modeling and prediction phases for predicting the dependent variables in a regression form or in a classification form.

RFM undergoes two processes in the modeling phase, namely, the bagging and ensembling processes. During the bagging process, random forest trees (RFTrees) are grown with high variances and they are grouped in separate bags. The ensembling process of RFM ensures that there is enough information from the created trees to generate models. At the end of this phase, a model is created using RFM.

RFM, at the prediction phase, tries to reduce the noises of the generated trees of the modeling phase of RFM. Testing data is utilized at this phase of RFM for the prediction process.

*Linear Regression Modeling.* LRM predicts the dependent variables based on the linear predictor functions. LRM suits well if the dependent function is mostly linear in nature as it attempts to find the best fitting line. In both the prediction cases (RFM and LRM), a squared error based calculation is utilized in order to find the error.

**5. Experimental Results.** This section demonstrates the proposed mechanism, SCALE-EA, the scalability aware performance tuning of OpenMP applications framework using EnergyAnalyzer (EA) tool by: i) conducting a study on the impact of variants of FA parameters, ii) understanding the energy efficiencies of FA and MAFA in SCALE-EA, and iii) analyzing the search time efficiencies of MAFA-LRM and MAFA-RFM based tuning processes.

**5.1. Experimental Setup.** Experiments were conducted on two machines, namely, a HP-Zbook-15G machine – a Haswell processor based workstation – and a 48 core HPProliant machine. In all experiments, the EnergyAnalyzer tool was utilized to measure the energy / performance values for the code regions of applications.

The following OpenMP applications / benchmarks are considered to demonstrate the proposed SCALE-EA mechanism:

1. *Arraybench*: This benchmark suite of EPCC [17] is written in OpenMP-C and it consists of ten benchmarks which reflect the performance of shared memory architectures in terms of the impact due to array operations of OpenMP applications. The array operations of OpenMP benchmarks are considered based on the private clauses of OpenMP, such as, copyin, threadprivate, and so forth, within loop blocks. The thread private data that are operated within the functional blocks of Arraybench are in the range of 3, 9, 27, 81, 243, 729, 2187, 6561, 19683, and 59049. The Arraybench application of the benchmark suite consists of four OpenMP parallel regions.
2. *Syncbench*: This benchmark consists of nine OpenMP parallel regions which are responsible for studying the impact of utilizing synchronization points in OpenMP. In general, OpenMP applications can enter into synchronization points at nine OpenMP constructs, such as, *parallel* regions, *parallel for* regions, *barriers*, *single* regions, *critical* regions, *ordered* sections, *atomic* regions, reduction states, and OpenMP locks. The impacts of these synchronization points of OpenMP constructs could be analyzed using Syncbench application.
3. *Taskbench*: Taskbench studies the effects of task creation and scheduling aspects of OpenMP 3.0. For instances, parallel task generation, master task generation, master task generation with busy slaves, conditional task generation, and so forth, are analyzed in Taskbench application. This benchmark consists of ten OpenMP parallel regions which could be tuned when executed on machines.
4. *CoMD application*: CoMD application [14] is an algorithmic representation of molecular dynamics simulations. It was developed at the co-design center of LLNL, USA. For the tests, OpenMP version of CoMD v.1.1 was utilized. This version of CoMD has eleven OpenMP parallel regions that are spread across the files, such as, eam.c (4 regions), initAtoms.c (5 regions), and ljForce.c (2 regions).

In experiments, SCALE-EA identified an efficient number of threads for the OpenMP parallel regions of the above mentioned applications based on the energy / performance values of user-specified regions of applica-

TABLE 5.1
*OpenMP Applications and Parallel Regions*

| Application | No.of Parallel regions | Parallel region files | User region (lineNo) | User region file | No.of Perf. Measurements |
|---|---|---|---|---|---|
| ArrayBench-3 to ArrayBench-59049 | 4 | arraybench.c | lineNo: 46-88 | arraybench.c | 6 |
| SynchBench | 9 | syncbench.c | lineNo: 45-101 | syncbench.c | 6 |
| TaskBench | 10 | taskbench.c | lineNo: 44-97 | taskbench.c | 6 |
| CoMD | 4 | eam.c | lineNo: 119-136 | CoMD.c | 6 |
| | 5 | initAtoms.c | | | |
| | 2 | ljForce.c | | | |

TABLE 5.2
*Energy and Performance Values of OpenMP Applications*

| Application | Energy (Joules) | ExecutionTime (secs) | TOT_INS | L1_DCM | BR_UCN | L3_TCM | EA Time (secs) |
|---|---|---|---|---|---|---|---|
| ArrayBench-3 | 16.04 | 0.395 | 1.05E+09 | 1.01E+07 | 2.80E+07 | 206 | 0.547 |
| ArrayBench-9 | 13.43 | 0.37 | 9.70E+08 | 7.19E+06 | 2.70E+07 | 181 | 0.541 |
| ArrayBench-27 | 15.27 | 0.363 | 8.60E+08 | 9.50E+06 | 2.70E+07 | 229 | 0.535 |
| ArrayBench-81 | 11.58 | 0.279 | 7.20E+08 | 7.60E+06 | 2.50E+07 | 201 | 0.448 |
| ArrayBench-243 | 14.29 | 0.349 | 9.20E+08 | 1.00E+07 | 2.40E+07 | 202 | 0.51 |
| ArrayBench-729 | 12.15 | 0.284 | 7.40E+08 | 8.30E+06 | 2.60E+07 | 254 | 0.43 |
| ArrayBench-2187 | 17.54 | 0.399 | 1.21E+09 | 4.60E+07 | 2.10E+07 | 279 | 0.56 |
| ArrayBench-6561 | 16.71 | 0.35 | 1.04E+09 | 7.30E+07 | 1.59E+07 | 246 | 0.68 |
| ArrayBench-19683 | 17.15 | 0.35 | 1.03E+09 | 6.79E+07 | 1.19E+07 | 373 | 0.537 |
| ArrayBench-59049 | 13.75 | 0.277 | 7.40E+08 | 5.08E+07 | 9.70E+06 | 533 | 0.44 |
| SynchBench | 32.75 | 0.867 | 2.01E+09 | 1.30E+07 | 3.30E+07 | 262 | 1.02 |
| TaskBench | 25.42 | 0.69 | 2.14E+09 | 9.60E+06 | 3.50E+07 | 247 | 0.846 |
| CoMD | 285.99 | 7.845 | 5.41E+10 | 6.4E+07 | 9.8E+07 | 1.39E+07 | 8.23 |

tions. The number of parallel regions, the user-specified regions, and the number of performance measurements undergone for OpenMP applications are listed in Table 5.1. The six performance measurements comprise of four hardware performance events (mentioned in the following section), the execution time in seconds, and the energy consumption of the code regions of applications in Joules.

**5.2. Performance and Energy Consumption Values.** In order to study the efficiency of SCALE-EA, at first, applications were experimented without SCALE-EA and the performance / energy consumption values of user-regions were recorded using EnergyAnalyzer tool. Table 5.2 shows the values obtained using the EnergyAnalyzer tool when the applications were experimented using eight threads for all parallel regions of OpenMP applications. The EnergyAnalyzer tool measured energy consumption values in Joules and four hardware performance events, such as, total number of instructions (TOT_INS), data cache misses in level 1 (L1_DCM), unconditional branches (BR_UCN), and total cache misses in level 3 (L3_TCM).

From Table 5.2, it could be observed that CoMD application had the highest energy consumption value when experimented with 8 threads (285.99 Joules). In this application, the total cache misses were in the order of $10^7$. The other applications had energy consumption values in the range of 11 to 32 Joules.

**5.3. Impact of FA Parameters.** In previous experiments, OpenMP applications were executed without FA or MAFA. In this subsection, experiments were conducted using the FA algorithm of SCALE-EA for studying the impacts of the variants of FA parameters.

In general, FA consists of seven parameters which control the performance of FA in terms of the convergence speed of the algorithm. These seven parameters are named as number of generations, population size of

TABLE 5.3
*FA Parameters and their Variants*

| FA-Variant | No.of Generations | Population Size | Initial Randomness | Epsilon | Alpha | Beta | Gamma |
|---|---|---|---|---|---|---|---|
| FA-Variant-1 | 10 | 10 | 0.1 | 0.009 | 0.5 | 0.5 | 0.0001 |
| FA-Variant-2 | 10 | 10 | 0.0001 | 0.009 | 1 | 1 | 0.0001 |
| FA-Variant-3 | 10 | 10 | 0.0001 | 0.009 | 0.5 | 0.5 | 0.0001 |
| FA-Variant-4 | 10 | 10 | 0.001 | 0.09 | 1 | 1 | 0.009 |
| FA-Variant-5 | 10 | 10 | 0.5 | 0.009 | 0.5 | 0.5 | 0.005 |
| FA-Variant-6 | 50 | 10 | 0.5 | 0.009 | 0.5 | 0.5 | 0.005 |

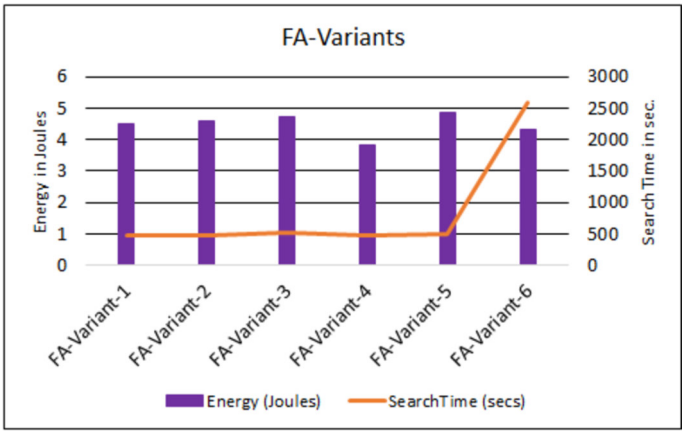


FIG. 5.1. *FA Parameter variants and their Impacts on Energy / Search Time Efficiency on the Haswell Machine.*

sequences, initial randomness, epsilon value, alpha value, beta value, and gamma value. The equations involved to describe two fireflies (sequences) mate each other are discussed in the algorithmic section of this paper (see 4.1).

It is crucial to understand the impact of these FA parameters during the process of identifying the efficient number of threads for parallel regions of OpenMP applications. Thus, experiments were conducted using Arraybench_19683 benchmark with seven varying combinations of FA parameters as shown in Table 5.3 on two machines in order to observe the impact of these parameters on the quality of observations.

It could be observed from Figure 5.1 that FA-Variant-4 achieved better energy consumption values at the reasonable amount of search time (3.841 Joules and 467.03 sec.) when compared to the others in the Haswell machine. FA-Variant-6, which was experimented with 50 number of generations, recorded the highest search time. Similar was the case when execution time efficiency was considered for the 48-core machine (see Figure 5.2). FA-Variant-4 achieved only 2275.8 seconds of search time (represented as line in Figure 5.2) when compared to FA-Variant-5 and FA-Variant-6 (2490.5 and 11969.5 seconds). The efficient number of threads identified by SCALE-EA when FA-Variant-4 was utilized in SCALE-EA was listed as 7, 10, 12, and 11 for the parallel regions of Arraybench-19683 benchmark. It should be noticed that the energy measurements could not be performed for the 48-core machine owing to the lack of RAPL counters in it.

In subsections 5.4 and 5.5, FA and MAFA based SCALE-EA were, therefore, experimented based on the FA-Variant-4 based parameter setting.

**5.4. Energy Efficiency of OpenMP Applications – FA and MAFA.** Fixing FA-Variant-4, SCALE-EA was executed for OpenMP applications using FA on the Haswell machine. SCALE-EA identified the efficient number of threads for parallel regions of applications as shown in Table 5.4. In addition, the energy consumption values for the user-specified regions of OpenMP applications as identified by FA and MAFA are given in Figure 5.3. As seen in Figure 5.3, all the applications showed better energy improvements when FA

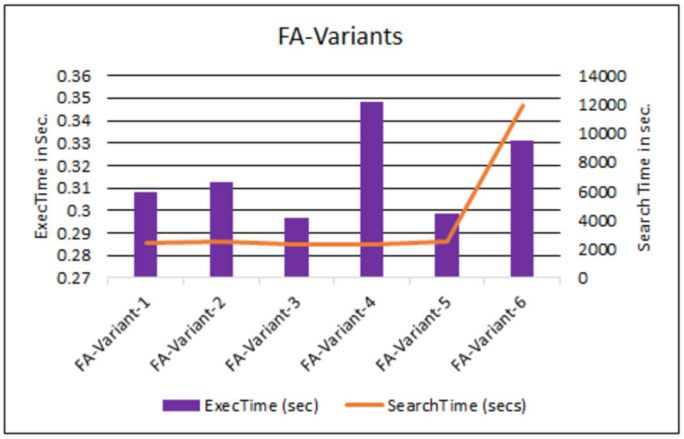


FIG. 5.2. *FA Parameter variants and their Impacts on Exec.Time / Search Time Efficiency on 48core Machine.*

TABLE 5.4
*Optimal Number of Thread Sequence Identified using FA of SCALE-EA*

| Application | Optimal Thread Sequence |
|---|---|
| ArrayBench-3 | 2 3 2 3 |
| ArrayBench-9 | 4 4 2 3 |
| ArrayBench-27 | 4 2 3 5 |
| ArrayBench-81 | 4 2 2 6 |
| ArrayBench-243 | 2 1 4 5 |
| ArrayBench-729 | 3 2 2 3 |
| ArrayBench-2187 | 5 2 2 5 |
| ArrayBench-6561 | 3 2 2 3 |
| ArrayBench-19683 | 3 2 2 3 |
| ArrayBench-59049 | 3 4 2 1 |
| SynchBench | 6 3 4 2 4 5 2 5 5 2 |
| TaskBench | 7 5 2 4 6 2 5 3 4 4 |
| CoMD | 1 3 1 7 2 4 1 5 2 3 2 |

and MAFA were applied. For instance, CoMD observed better energy consumption value when the eleven OpenMP parallel regions of three files were automatically executed with 1 3 1 7 2 4 1 5 2 3 2 thread numbers – i.e., the parallel regions of eam.c were executed with 1, 3, 1, and 7 threads; the parallel regions of initAtoms.c were executed with 2, 4, 1, 5, and 2 threads; and, the parallel regions of ljForce.c were executed with 3 and 2 threads. Similarly, the other applications, such as, Arraybench-2187 and Syncbench showed improved energy consumption values when experimented with SCALE-EA.

Figure 5.4 reveals the energy efficiencies of applications due to FA, MAFA-LRM and MAFA-RFM. The energy efficiencies of applications due to FA and MAFA were compared with the normal executions of applications constituting eight threads throughout the OpenMP parallel regions of applications. For instance, the energy efficiencies of ArrayBench-3 while experimenting with FA, MAFA-LRM and MAFA-RFM were identified as 69.01, 77.30, and 74.05 – i.e., Energy=16.04J for normal execution (all parallel regions with 8 threads), Energy=4.97J for FA, Energy=3.64J for MAFA-LRM, and Energy=4.161J for MAFA-RFM. Similarly, the experiments were carried out for all test applications (see Figure 5.4).

As seen in Figure 5.4, the following points could be inferred:

1. FA and MAFA, in general, performed well while identifying the efficient number of threads for OpenMP applications. In all experiments, the energy efficiencies achieved for the applications were from 31.21 percentage (MAFA-LRM for CoMD application) to 78.51 percentage (MAFA-LRM for Arraybench-

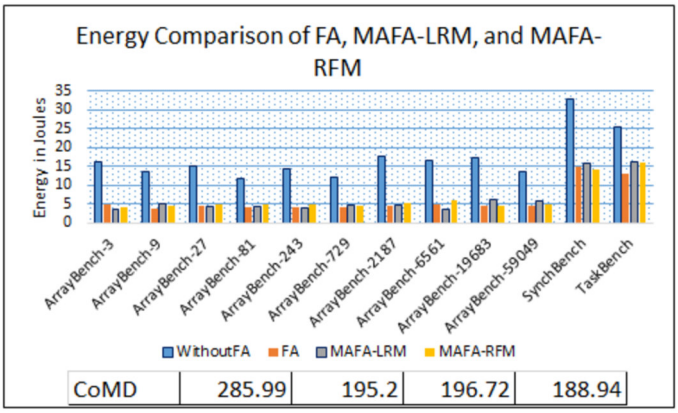


FIG. 5.3. *Comparison of Energy Consumption Values – FA, MAFA-LRM, and MAFA-RFM*

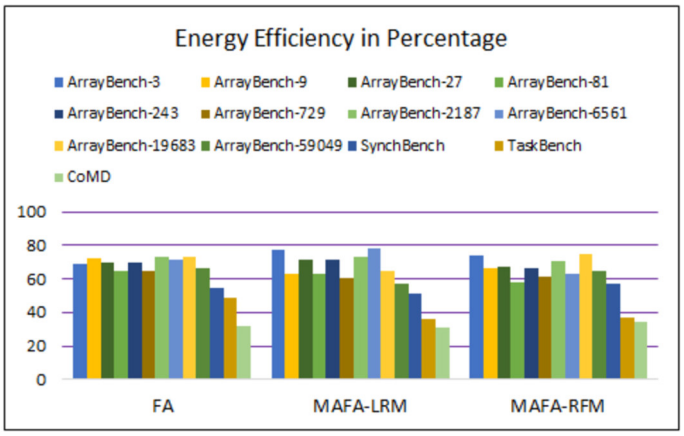


FIG. 5.4. *Energy Efficiency of MAFA-LRM and MAFA-RFM based on FA*

6561).

2. MAFA-LRM could achieve better energy efficiency in applications, such as, Arraybench-3,27,243, and Arraybench-6561. In some cases, FA produced better energy efficiency than the other two MAFA based algorithms – see the results of Arraybench-9,81,729, 2187,59049, and Taskbench application in Figure 5.4.

**5.5. Search Time Efficiency of MAFA.** Although FA achieved better energy efficiencies in six OpenMP applications (see the previous subsection), it resulted from challenging the search time of the tuning process. Figure 5.5 shows the search time efficiency of MAFA algorithms (MAFA-RFM and MAFA-LRM) with respect to FA. The points represented in the graph clearly manifests the inability of FA when compared to MAFA based algorithms – i.e., MAFA-RFM and MAFA-LRM outperformed traditional FA in terms of search time efficiency: MAFA-RFM reached up to 32.56 percentage of search time improvement for Arraybench-2187 when compared to FA. It is also important to notice the improved energy consumption values achieved for Arraybench-2187 application based on these algorithms (without-FA–16.71 (J); FA–4.72 (J); MAFA-LRM–3.59 (J); and MAFA-RFM–6.192 (J)). Vividly, MAFA based algorithms outperformed FA and without-FA in terms of energy efficiency and search time efficiency when compared to the other approaches.

In addition, the efficiency chart of MAFA-RFM and MAFA-LRM with respect to FA is represented as a 100 percentage stacked column chart (see Figure 5.5). As seen, the search time efficiency values of MAFA-RFM and MAFA-LRM for ArrayBench_3 had higher variations, while the values for CoMD had lower variations.

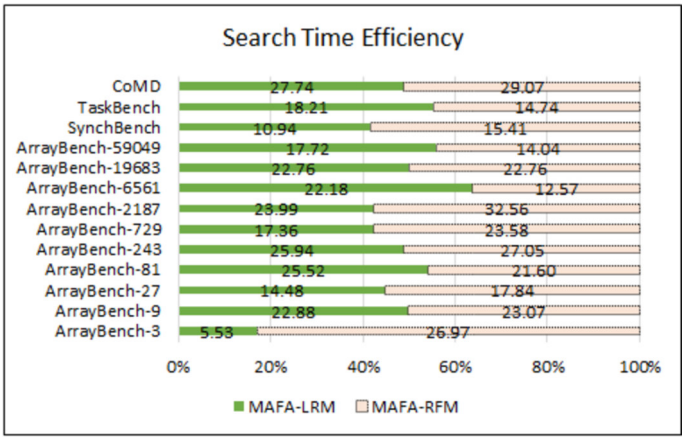


FIG. 5.5. *Search Time Efficiency of MAFA-LRM and MAFA-RFM based on FA*

**6. Conclusion and Outlooks.** Scalability and energy consumption issues must be considered for future HPC applications. These issues are dependent on the underlying HPC machines and the software frameworks. Autotuning is one of the solution that supports application developers in these contexts. However, autotuning solutions suffer from hefty search time for obtaining better solutions. This paper proposed a scalability-aware performance tuning framework (SCALE-EA) using the Firefly Algorithm and using Modeling Assisted Firefly Algorithm (MAFA) for OpenMP applications. SCALE-EA identified an efficient number of threads for each OpenMP parallel regions of applications at reduced search time. The results when experimented on the machines manifested that SCALE-EA achieved energy efficiencies of of 31.21 to 77.3 percentage and search time efficiencies of 5.53 to 32.56 percentage for candidate applications, such as, Arraybench, Syncbench, Taskbench, and CoMD applications.

**Acknowledgements.** The author thanks Rejitha R.S who has jointly worked for the paper when she was affiliated to St. Xaviers Catholic College of Engineering, India.

REFERENCES

[1] ABDUL WAHID MEMON AND GRIGORI FURSIN, *Crowdtuning: systematizing auto-tuning using predictive modeling and crowd-sourcing*, in PARCO mini-symposium on Application Autotuning for HPC (Architectures), 2013.

[2] ANANTA TIWARI, CHUN CHEN, JACQUELINE CHAME, MARY W. HALL, JEFFREY K. HOLLINGSWORTH, *A scalable auto-tuning framework for compiler optimization*, IPDPS, Italy, pp. 1-12, 2009.

[3] ANNA SIKORA, EDUARDO CÉSAR, ISAAS COMPRÉS, AND MICHAEL GERNDT, *Autotuning of MPI Applications Using PTF*, In Proceedings of the ACM Workshop on Software Engineering Methods for Parallel and High Performance Applications (SEM4HPC '16), pp. 31-38, 2016.

[4] ANANTA TIWARI, MICHAEL A. LAURENZANO, LAURA CARRINGTON, AND ALLAN SNAVELY, *Auto-tuning for energy usage in scientific applications*, in Euro-Par'11, pp. 178–187, 2012.

[5] BABAK BEHZAD, HUONG VU THANH LUU, JOSEPH HUCHETTE, SURENDRA BYNA, PRABHAT, RUTH AYDT, QUINCEY KOZIOL, MARC SNIR, *Taming parallel I/O complexity with auto-tuning*, in SC13, Vol. 28, doi:10.1145/2503210.2503278, 2013.

[6] SIEGFRIED BENKNER, SABRI PLLANA, JESPER LARSSON TRÄF, PHILIPPAS TSIGAS, ANDREW RICHARDS, RAYMOND NAMYST D., BEVERLY BACHMAYER E., CHRISTOPH KESSLER F., DAVID MOLONEY G., PETER SANDERS H., *The PEPPHER Approach to Programmability and Performance Portability for Heterogeneous many-core Architectures*, in ParCo, Belgium, pp.1-8, Aug. 2011.

[7] BRIAN J. N. WYLIE AND WOLFGANG FRINGS, *Scalasca support for MPI+OpenMP parallel applications on large-scale HPC systems based on Intel Xeon Phi*, In Proc. of the Conf. on Extreme Science and Engineering Discovery Environment: Gateway to Discovery (XSEDE '13), New York, NY, USA, Vol. 37, pp. 1-8, 2013.

[8] BUSE YILMAZ, BARI AKTEMUR, MARÍA J. GARZARÁN, SAM KAMIN, AND FURKAN KIRAÇ, *Autotuning Runtime Specialization for Sparse Matrix-Vector Multiplication*, ACM Trans. Archit. Code Optim., Vol. 13, No.1, pp. 1-26, 2016.

[9] CAVAZOS, JOHN AND FURSIN, GRIGORI AND AGAKOV, FELIX AND BONILLA, EDWIN AND O'BOYLE, MICHAEL F. P. AND TEMAM, OLIVIER, *Rapidly Selecting Good Compiler Optimizations Using Performance Counters*, doi:10.1109/CGO.2007.32, pp. 185-197, 2007.

[10] CarbonFootprint, in `CarbonFootprint_and_Energy_Efficiency_Rev_1.0_2.pdf`, accessed in 2016.

[11] Cesar E., A. Moreno, J. Sorribes, E. Luque, *Modeling Master/Worker applications for automatic performance tuning*, in Parallel Computing, Vol. 32, No. 78, pp. 568-589, 2006.
[12] Cheng Y., and Zeng Y., *Automatic Energy Status Controlling with Dynamic Voltage Scaling in Power-Aware High Performance Computing Cluster*, in proc. of 12th Int. Conf. on PDCAT, pp. 412 - 416, 2011.
[13] Cristina Silvano, Giovanni Agosta, Stefano Cherubin, Davide Gadioli, Gianluca Palermo, Andrea Bartolini, Luca Benini, Jan Martinovic, Martin Palkovic, Katerina Slaninova, Joao Bispo, Joao M. P. Cardoso, Rui Abreu, Pedro Pinto, Carlo Cavazzoni, Nico Sanna, Andrea R. Beccari, Radim Cmar, Erven Rohou, *The ANTAREX approach to autotuning and adaptivity for energy efficient HPC systems*, in ACM International Conference on Computing Frontiers, 2016.
[14] CoMD Application, from LLNL, USA, in `http://www.exmatex.org/comd.html`, accessed in Sep.2016.
[15] Daniel A. Ellsworth, Allen D. Malony, Barry Rountree, Martin Schulz, *POW: System-wide Dynamic Reallocation of Limited Power in HPC*, in Proc. of HPDC 2015, pp. 145-148, 2015.
[16] Enes Bajrovic, Siegfried Benkner, Jir Dokulil, Martin Sandrieser, *Autotuning of Pattern Runtimes for Accelerated Parallel Systems*, in Proc. of PARCO 2013, pp. 636-645, 2013.
[17] EPCC OpenMP Benchmark suite, in `https://www.epcc.ed.ac.uk/research/computing/performance-characterisation-and-benchmarking/epcc-openmp-micro-benchmark-suite`, accessed in Sep 2016.
[18] Erika Abraham, Costas Bekas, Ivona Brandic, Samir Genaim, Einar Broch Johnsen, Ivan Kondov, Sabri Pllana, and Achim Streit, *Preparing HPC Applications for Exascale: Challenges and Recommendations*, in ADPNA at NBiS, pp. 1-6, 2015.
[19] Fister, I., Yang, X.S., Brest, J. and Fister Jr, I., *Memetic self-adaptive firefly algorithm*, in Swarm intelligence and bio-inspired computation: theory and applications, pp.73-102, 2013.
[20] Fister I., Yang X. S., Brest J., *A comprehensive review of firefly algorithms*, in Swarm and Evolutionary Computation, Vol. 13, pp. 34-46, 2013.
[21] Giorgio Luigi Valentini, Walter Lassonde, Samee Ullah Khan, Nasro Min-Allah, Sajjad A. Madani, Juan Li, Limin Zhang, Lizhe Wang, Nasir Ghani, Joanna Kolodziej, Hongxiang Li, Albert Y. Zomaya, Cheng-Zhong Xu, Pavan Balaji, Abhinav Vishnu, Fredric Pinel, Johnatan E. Pecero, Dzmitry Kliazovich, Pascal Bouvry, *An overview of energy efficiency techniques in cluster computing systems*, in Cluster Computing, Vol. No. 1, pp 315, 2013.
[22] Greg Faanes, Abdulla Bataineh, Duncan Roweth, Tom Court, Edwin Froese, Bob Alverson, Tim Johnson, Joe Kopnick, Mike Higgins, and James Reinhard, *Cray cascade: a scalable HPC system based on a Dragonfly network*, In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '12), Los Alamitos, CA, USA, 9 pages, 2012.
[23] Guojing Cong, I-Hsin Chung, Hui-Fang Wen, David J. Klepacki, Hiroki Murata, Yasushi Negishi, Takao Moriyama: *A Systematic Approach toward Automated Performance Analysis and Tuning*, in IEEE Trans. Parallel Distrib. Syst., Vol. 23, No. 3, pp. 426-435, 2012.
[24] Haihang You., Q. Liu, Z. Li, and S. Moore, *The Design of an Auto-tuning I/O Framework on Cray XT5 System*, in Proc. of Cray Users Group Conference (CUG'11) Alaska, May 2011.
[25] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe, *OpenTuner: an extensible framework for program autotuning*, In Proceedings of the 23rd international conference on Parallel architectures and compilation (PACT '14), USA, pp. 303-316, 2014.
[26] Jee W. Choi, Amik Singh, and Richard W. Vuduc, *Model-driven autotuning of sparse matrix-vector multiply on GPUs*, In Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '10), USA, pp. 115-126, 2010.
[27] Jordan, Herbert and Thoman, Peter and Durillo, Juan J. and Pellegrini, Simone and Gschwandtner, Philipp and Fahringer, Thomas and Moritsch, Hans, *A Multi-objective Auto-tuning Framework for Parallel Codes*, Proc. of the Int. Conf. on HPC, Networking, Storage and Analysis, SC12, pp. 10:1-10:12, 2012.
[28] Knobloch M., Mohr B., and Minartz T., *Determine energy-saving potential in wait-states of large-scale parallel programs*, in Computer Sci. - Res. and Dev. pp. 1-9, 2011.
[29] Manish Parashar, Salim Hariri, *Autonomic Computing: An Overview*, in Unconventional Programming Paradigms, Vol. 3566, LNCS, Springer, pp 257-269, 2005.
[30] Matthieu Doriera, Orcun Yildizc, Shadi Ibrahimc, Anne-Cécile Orgerieb, Gabriel Antoniuc, *On the energy footprint of I/O management in Exascale HPC systems*, in FGCS, Vol. 62, pp. 17-28, 2016.
[31] Dark Silicon, in `https://www.dagstuhl.de/en/program/calendar/semhp/?semnr=16052`, 2016.
[32] Pan Z. and Eigenmann R., *Fast and effective orchestration of compiler optimizations for automatic performance tuning*. In Proc. of the Int. Symp. on Code Generation and Optimization, pages 319?332, 2006.
[33] Prasanna Balaprakash, Stefan M. Wild, Boyanna Norris, *SPAPT: Search Problems in Automatic Performance Tuning*, in Procedia Computer Science, Vol 9, pp. 1959 - 1968, 2012.
[34] Robert Schöne, Daniel Molka, and Michael Wern, *Wake-up latencies for processor idle states on current x86 processors*, in Comput. Sci. Res. Dev. Vol. 30, pp. 219227, 2015.
[35] Senthilnath J., Omkar S. N., Mani V., *Clustering using firefly algorithm: performance study*, in Swarm and Evolutionary Computation, Vol. 1, No. 3, pp. 164-171, 2011.
[36] Shajulin Benedict, *Application of Energy Reduction Techniques using Niched Pareto GA of EnergyAnalzyer for HPC Applications'*, in 7th IEEE IC3 2014, http://dx.doi.org/10.1109/IC3.2014.6897234, 2014.
[37] Shajulin Benedict, *Threshold Acceptance Algorithm based Energy Tuning of Scientific Applications using EnergyAnalyzer*, ISEC2014, ACM publishers, 2014.
[38] Shajulin Benedict, Rejitha R.S., Phillip G., Radu Prodan, Thomas Fahringer, *Energy Prediction of OpenMP Ap-*

*plications using Random Forest Modeling Approach*, in iWAPT2015 @ IPDPS 2015 DOI 10.1109/IPDPSW.2015.12, pp. 1251-1260, 2015.

[39] Sukhyun Song and Jeffrey K. Hollingsworth, *Computationcommunication overlap and parameter auto-tuning for scalable parallel 3-D FFT*, in Journal of Computational Science, Vol. 14, pp. 3850, 2016.

[40] Weifeng Liu, Michael Gerndt, Bin Gong, *Model-based MPI-IO tuning with Periscope tuning framework*, Concurrency and Comp.: Prac. and Exp., Vol. 28, No.1, pp: 3-20, 2016.

[41] Xin-She Yang, *Firefly algorithm, stochastic test functions and design optimisation*, in Int. Journal of Bioinspired Computation, Vol. 2, No. 2, pp. 78-84, 2010.

[42] Yury Oleynik, Michael Gerndt, Joseph Schuchart, Per Gunnar Kjeldsberg, and Wolfgang E. Nagel, *Run-Time Exploitation of Application Dynamism for Energy-Efficient Exascale Computing (READEX)*, CSE 2015, pp:347-350, 2015.

*Edited by:* Dana Petcu
*Received:* Oct 17, 2017
*Accepted:* Dec 23, 2017